1975

An interface processor for a high speed recirculating data network

Chong Chun Lee
Iowa State University

An interface processor for a high speed recirculating data network

by

Chong Chun Lee

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

Major: Electrical Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Iowa State University
Ames, Iowa

1975

TABLE OF CONTENTS

A LIST OF SYMBOLS

SBAT : Source Buffer Address Table

DBAT : Destination Buffer Address Table

SBST : Source Buffer Status Table

DBST : Destination Buffer Status Table

TAT : Table Address Table

TSAT : Time Slot Acknowledge Table

PNT : Process Name Table

ADT : Allowed Destination Table

TUT : Table Updating Table

IIT : Interrupt Information Table

OS : Output Sequencer

TM : Table Matcher

Micro : Micro-processor

PROM : Programmable Read Only Memory

RAM : Random Access Memory

ABSTRACT

The objective of this work is to make an original contribution to the knowledge about general purpose interface processors for loop connected data communication systems. Specifically, three of the research goals are (1) to discover critical parameters in interfacing such a system, (2) to determine a near optimum organization for an interface processing unit, (3) to demonstrate on a general and detailed basis that such a system is feasible.

The results of the research goals are: (1) Critical parameters in the system are found to be the processing rates for the various table matches; this problem was solved by a processor specially developed for high speed sequential table searching operations. (2) The interface is organized as a multi-processor system which has three small processors working concurrently for different interfacing problems. (3) Feasibility of such a system is demonstrated in two ways: a) general discussion of the system at the block diagram and flowchart levels; b) detailed discussion of the system.

INTRODUCTION

A computer network is an interconnected set of dependent or independent computer systems which communicate with each other in order to share certain resources such as programs or data. There are two major capabilities desired in a data network. One is load sharing which is the ability to take a given work load and to distribute it among the computers of a network in order to make efficient use of the resources of the network. The other is program sharing which is the ability to allow data to be sent to a system at which a desired program is resident.

The techniques involved in achieving these two major requirements on a computer network are greatly dependent on the data communication link topology. The simplest kind of data communication link is called a point-to-point link as shown in Figure 1. Figure 2 and Figure 3 show two other communications link topologies; a nonswitched multipoint system and a switched system. The multipoint link system connects more than two stations together. The switched link system allows communication between pairs of stations which are temporarily connected in a point-to-point fashion by the switching facility. These three basic link configurations are currently used to construct complex communications networks, in which interconnections are achieved by allowing the stations to have both transmission and receiving capabil-

Figure 1. Point-to-point system.

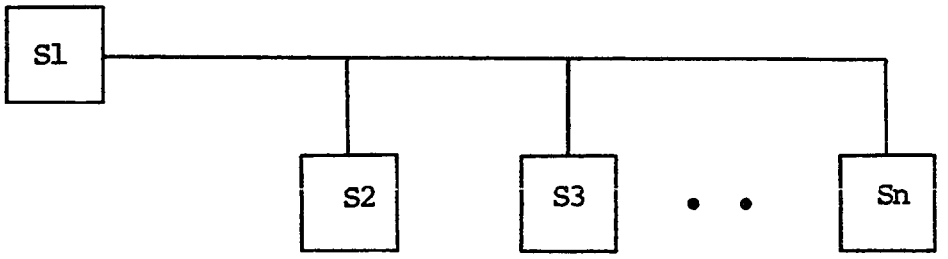


Figure 2. Nonswitched multipoint system.

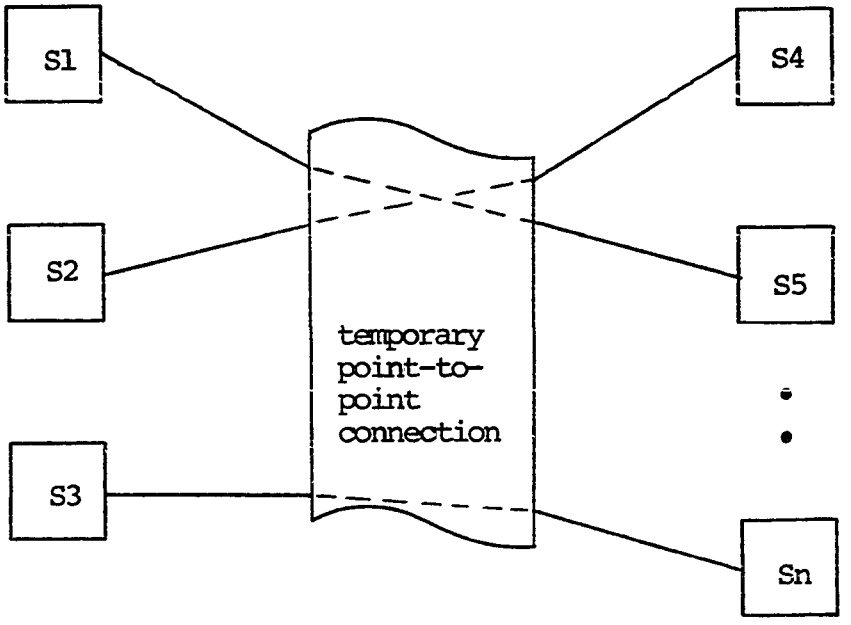


Figure 3. Switched system.

ities.

To increase the potential capability of a network, temporary point-to-multipoint connections are required occasionally. This is not generally possible in a switched system operating on a point-to-point fashion. When the number of stations connected to the system increases linearly, the complexity of control functions on the central switching device grows exponentially to provide for all the possible connections of the stations. Due to these reasons, the central switching device of the switched system is not economically feasible when the control functions become complex.

A possible approach to provide the same capabilities as the switched system, using a central device, is a loop topology technique. The control function of the complex central switching device is to be broken into two parts; one for the communication line control and the other for interfacing problems. The line control functions of the loop are assigned to the repeater and a master repeater and the switching control functions are distributed over the interfaces, as shown in Figure 4. The control functions of the complex central switching device are moved to the interface and the line controller. This allows a complex switched communication system to be physically implemented using a closed loop connecting each station together through a locally owned interface.

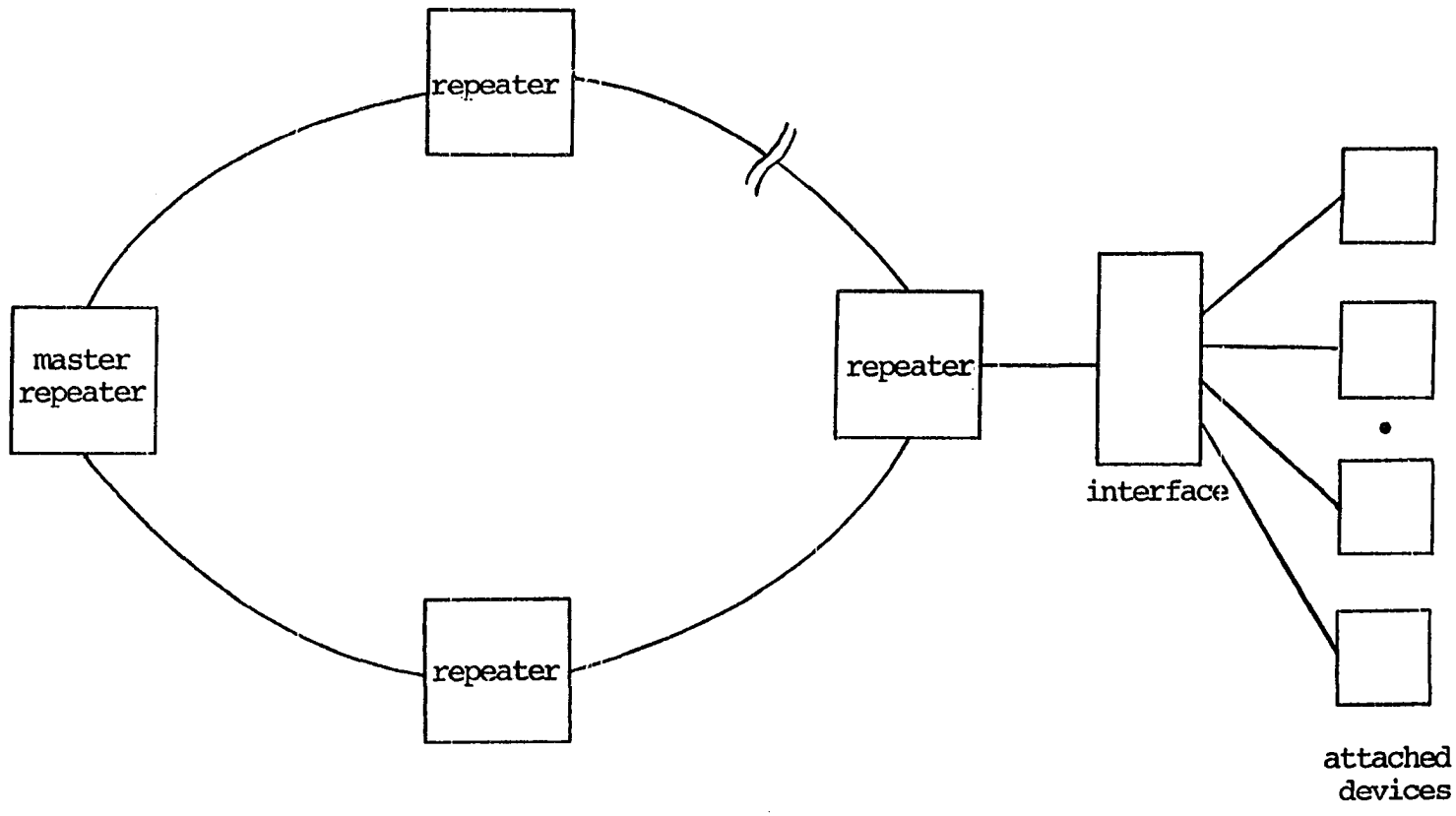


Figure 4. A loop connected system.

In a university environment, an experimental communication loop for computer access should have the following characteristics.

1. A given station must be able to access any other station on the loop.
2. The data loop should be constructed to have enough bandwidth to meet the user's needs.
3. The initial cost of the data loop should be kept to a minimum.
4. The incremental cost of adding stations to the existing loop should be modest.
5. The data loop must be reliable.

In January 1973, the I.S.U. Computation Center and the I.S.U. College of Engineering undertook a development project to provide a local high speed recirculating data network which is formed by interconnecting various minicomputers, teletype compatible devices and data collection devices throughout the engineering college buildings as one loop. The purpose of this loop is to share computing power and secondary storage facilities among the various research and laboratory facilities that may exist in the engineering departments. Currently, two repeaters, one master repeater and three interface processors are being constructed to form a loop with three stations. The loop will be ready for testing some time before summer, 1975. The plan may be ex-

panded to the whole Iowa State University campus connecting a number of loops tied together.

In conjunction with the project, this dissertation describes the following three research goals pursued for making an original contribution to the knowledge about general purpose interface processors for loop connected systems.

1. discover critical parameters in interfacing such a system.
2. determine a near optimum organization for an interface processing unit.
3. demonstrate on a general and detailed basis that such a system is feasible.

The results of the research goals are discussed in the following ways.

1. The discovered critical parameters are examined and their impact on the system are evaluated.
2. The determined organization is justified and evaluated for a near optimum organization which satisfies three of the initial design goals: a) high speed; b) low cost; c) system flexibility.
3. Feasibility of such a system is discussed in two ways: a) general discussion of such a system in block diagram and flowchart level; b) detailed discussion of the system.

REVIEW OF LITERATURE

A data network which consists of a number of closed loops around which data circulate was discussed by Pierce (1) and Kropfl (2). The network used asynchronous multiplexing, buffered switching and a distributed control system. In this loop switching system, users are connected to the network by stations which are interconnected by a local loop transmission line. An experimental network has been implemented by interconnecting two laboratory computers through an addressed block data transmission system consisting of a single loop. The actual hardware and operations of the network is discussed in a paper by Coker (3).

There are several communication networks being developed and constructed following the loop technique. One of these is the Distributed Computing System (DCS) developed at the University of California, Irvine by Farber et al. (4), (5), (6), (7). The main objective of the DCS is to provide reliable, fail-soft service at relatively low cost to a large class of users with modest requirements. The distributed organization of this system incorporates redundancy and isolation in both the hardware and software. Reliability is achieved by minimizing the probability of a total system failure, using isolation and keeping local failures from spreading and causing a global failure, and using redundancy to negate the effects of local failures.

The DCS hardware system is a collection of computing system components connected to a digital communication ring by ring interfaces. The communication ring serves as a unidirectional information path and the ring interfaces assist in information routing. The DCS software system is process oriented, that is, all activities are carried out by processes rather than physical hardware address.

A packet switching data communications network is being developed at the National Security Agency for resource sharing and the future development of distributed processing and filing systems by Hassing et al. (8). The design goal of this system is to interconnect a large, heterogeneous group of computers, batch terminals and conversational terminals to form a general purpose network of computing resources. This system consists of three nodes situated around each of two concentric rings running in opposite directions. This arrangement provides for redundancy in the network to protect the ring connectivity. The ring interface at each node is a hard-wired device called a Network Interface Port (NIP). This NIP in turn interfaces to a minicomputer which is responsible for network protocol and activities on the one hand, and for interfacing some device or devices to the network on the other.

Hayes (9) studied the performance of an experimental computer communication network which is currently being de-

signed and built at Bell Labs. The network consists of synchronous digital transmission lines connected in loops to a central switch. User traffic enters the system through multiplexers connected to the synchronous lines. The central switch has the two-fold function of routing and controlling traffic. The system consists of several loops connected to a central switch. The system is accessed through a Terminal Interface Unit (TIU) which is connected between the user terminals and the loop. In addition to forming an interface between the user and the loop, the TIU also does signaling which plays a role in switching calls and controlling the traffic flow.

All of the loop systems mentioned rely on the Bell System T1 carrier technology. The T1 carrier is a Pulse Code Modulated (PCM) repeaterized voice channel system designed for high grade long distance communications. Because transmission in this system is digital it lends itself well to high speed data communication applications.

There are two common techniques which are used for achieving multiplexing on a loop connected system. These are Demand Multiplexing (DM) and Synchronous Time Division Multiplexing (STDM). In STDM each terminal is assigned a time slot which recurs periodically. The terminal may multiplex data into its dedicated slot and/or receive data only from this time slot. In contrast, for DM, data is

multiplexed on the line asynchronously into unoccupied time slots. If a terminal has a message to transmit to the net, it inserts the message into the first empty slot. The delay performances of these two techniques are compared by Hayes (9). His simulation results on the Bell Labs loop connected system show that DM is superior to STDM in terms of delay performances.

In any of these system using a loop topology, the number of interconnections required to allow each terminal to communicate with any other terminals is minimized. In addition to this obvious advantage, another benefit is present since the number of nodes on a loop may be increased with no significant increase in line requirement. Routing is also considerably simplified because any node can reach any other node directly, as long as there is sufficient redundancy to keep the loop intact.

RESEARCH PROJECT OVERVIEW

The high speed data network is a recirculating data communications system formed by connecting line repeaters along the line. One of these repeaters is the master repeater which is a line controller as well as a general repeater. The functions and the design of these repeaters and the master repeater has been described by Koenck (10). However, some of the features involved in the repeaters and the master repeater must be introduced briefly for discussing the interfacing problems.

The network is implemented using two shielded, twisted pairs for the transmission line forming a closed loop. Each twisted pair is used as a transmission path so that the one cable can have transmissions going in either direction depending on the physical environment. The signal frequency on this transmission line has been set at 1.544 MHz which makes the net compatible with the Bell T1 carrier system. This allows the loop to be extended for long distance communication by inserting T1 repeaters at regular intervals on the transmission line.

The information passing along the net can be thought of as a string of binary values. This string of bits is logically thought of as being divided into 'slots' of 160 bits each. The information that is represented by the bits in a slot will be called a 'message'. A slot is further broken

into 16 pieces. Each of these pieces begins with two character synchronization bits '10' followed by an eight bit data byte. The two synchronization bits at the beginning of each byte are used by the repeater to insure correct assembly of the signal from the line. They are inserted by the repeater as the data is put on the line and are stripped off by the repeater before it passes the data to the interface. Therefore, a byte, within the interface, consists of 8 bits and the basic message, a time slot, consists of 16 bytes of 8 bits each as shown in Figure 5.

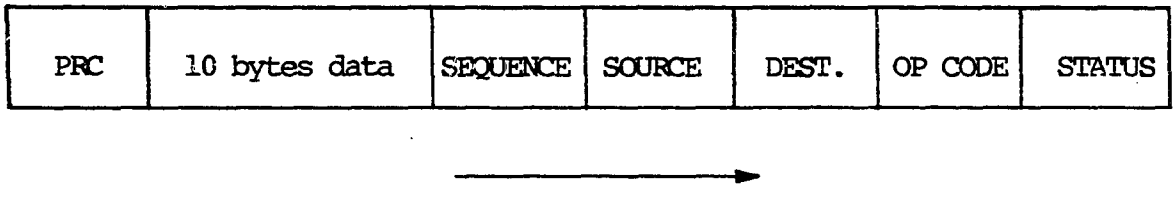


Figure 5. Organization of a time slot.

The first incoming byte of a time slot is the STATUS byte containing flags which indicate the status of the time slot and the following OP CODE byte specifies the type of message represented by the slot. Depending upon the OP CODE, the DEST byte can indicate a physical device address or a process name, defined on the net, to which the message is directed. The SOURCE is the sender identification indicating where the message originated. The SEQUENCE byte is used to

provide sequencing information to the receiver. The following ten bytes are the actual data and the last byte, the Polynomial Redundancy Check byte, is used for detecting transmission errors between the previous repeater and the current repeater station.

The loop has 32 of these 160 bit slots in continuous circulation, carrying information from one node to another. Also, there is a master synchronization slot, two bytes in length, consisting of a binary 1 followed by 19 zeros. This master sync pulse is generated in the master repeater and is sent along the net to synchronize the repeaters signalling the beginning of the 32 time slots. This amounts to a total capacity of 514 bytes or 5140 bits, in continuous circulation along the loop. Since the data rate of the loop is fixed at 1.544 Mhz, the total loop delay amounts to 3.3 ms. That is, a message originating from a node will be returning to the original sender in 3.3 ms.

Up to 16 repeaters will be provided on the net and these repeaters obtain their operational power directly from the line that provides the data path. In this way, the operation of the net is independent of the local power condition of any repeater station.

The repeater also has the following functions as well as those already described:

1. Regenerate the incoming data after it has been

distorted by noise and attenuation on the transmission line.

2. Keep count of bytes passed so that time division multiplexing can be implemented.

3. Supply enough delay that a whole time slot can be held in the repeater.

4. Control transmission errors.

5. Recognize when the net is in test condition and indicate when errors have occurred during the test.

6. Provide data paths so that a message can be either received or transmitted at each interface.

7. Transform the coded signal from the line to a binary code.

8. Generate a clock signal from the incoming coded data stream.

9. Transmit the binary data back to the line in the phase code used on the line.

In addition to the functions of a repeater, the master repeater also has following functions:

1. Generate master sync pulses.

2. Buffer the incoming messages to allow for variable amounts of delay on the line.

3. Log and display occurrences of errors on the line.

4. Generate and check start-up test patterns.

5. Indicate line failures.

Connected to each of these repeaters and to the master repeater is a general purpose interface. The general purpose interface has been designed with three goals in mind: 1) high speed 2) low cost 3) system flexibility. The interface is a hardware unit through which a device attached to the interface can communicate with any other devices or processes which are currently active on the net. Directly connected to each interface is a combination of devices such as teletypes(TTY), line printers, card readers, CRT displays and/or a minicomputer. Since these devices and the interface require an external source of power, they may fail to operate locally. However, this local power failure will not cause the complete loop to fail as the repeaters are powered independently from the line itself. Pictorially, the loop may be viewed as in Figure 6.

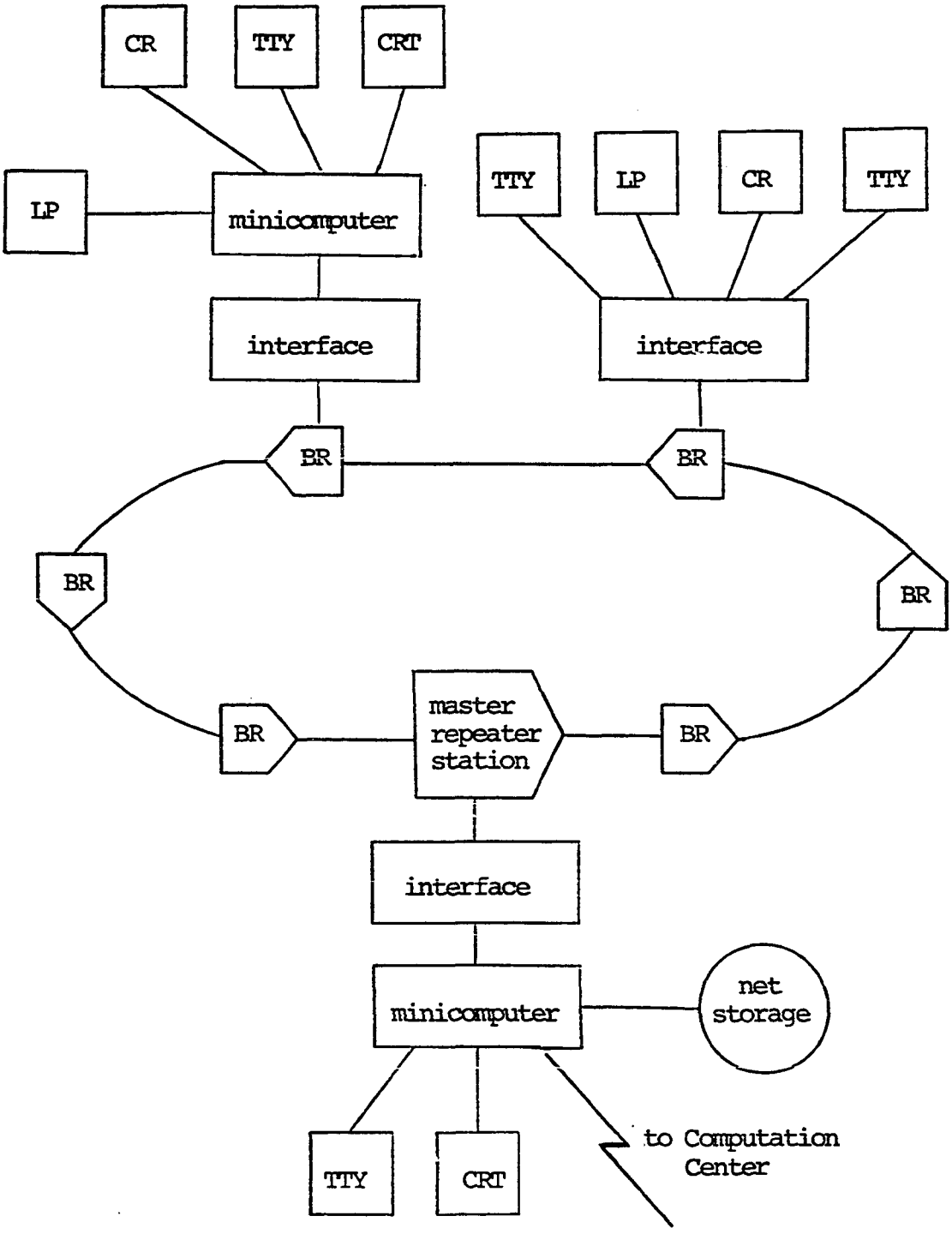


Figure 6. The ISU high speed data net.

THE INTERFACE

Problem Considerations

Each repeater has a 9 bit binary counter for counting up to 512 characters and a 4 bit binary counter for counting ten binary bits. The two byte master synch pulses generated by the master repeater reset these counters to establish the repeater synchronization. The low order four bits of the character counter indicate the current message byte and the high order 5 bits indicate one of the 32 different time slots. Synchronization of the interface operations to the repeater is to be done by looking at these two counters.

Since one time slot is 16 bytes, every 16 bytes of delay a new time slot comes into a repeater station. Whenever a new time slot comes into the repeater station, the low order 4 bits of the character counter will be updated to 0000 and the upper 5 bits will be incremented by one, indicating the incoming time slot number. An interface can take only one byte of the message at a time from the associated repeater station and can transmit only one byte at a time to the associated repeater station. In order to allow the interface time to complete processing, based on the result of the polynomial redundancy check in the last byte of a time slot, the repeater will buffer 17 bytes in a shift register thus providing a 1 byte delay between receiving the last byte of a

slot and sending of the first byte to the repeater. This is shown in Figure 7. This provides a total time of 17 bytes or 110 microseconds from receiving the STATUS byte of a slot until the interface sends a new STATUS byte to the repeater for that slot.

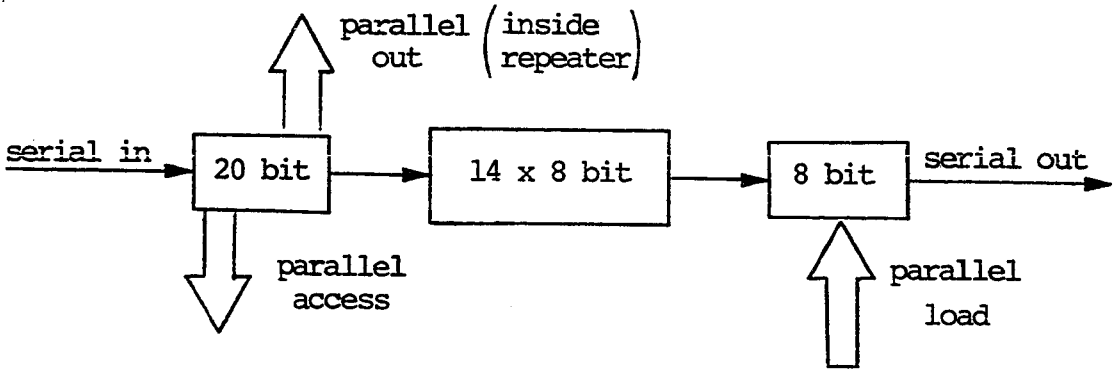


Figure 7. Repeater shift registers.

This implies that the interface needs to handle all interfacing problems such as checking STATUS and OP CODE bytes, checking different tables, receiving a message, updating tables, updating the STATUS byte,seconds. Taking etc. within 110 micro into account the complexity of the interfacing problems, no currently available general purpose processor can handle these problems within 110 microseconds for a reasonable price. An attempt has been made to design a fast general purpose processor using standard TTL components.

As a result, a processor with a 500 ns instruction execution time has been designed. However, even this speed was not fast enough to handle these fairly sophisticated interfacing problems.

Therefore, three main design goals have been setup for developing the interface. These are:

1) high speed: The interface must be fast enough to handle the complex interfacing problems within 110 micro seconds.

2) low cost: Since one of the main objectives of the loop is to achieve a low incremental cost of adding a station to the loop, the price of the interface must be cheap enough to meet this requirement.

3) flexibility: since the data loop is still in an experimental stage, those algorithms related to the defined interfacing functions must not be fixed. This implies that the interface must be flexible for future modification.

Status of a Time Slot

In general, the stations on a data communication link are quite some distance apart and the link is the sole means of communication among them. This requires that the data communication link must carry, in addition to the message data, the control information needed to coordinate station activities, e.g., the acknowledgement sentence advising an originating station whether a message sentence was received

correctly. This would seem to be an obvious, even trivial, observation, but in fact this situation leads to the need for quite elaborate protocols to control communication and to distinguish data from control information.

This sharing of a facility for use in both control and data flow activities is a very unnatural practice in the world of computing. In computer design, data buses are almost isolated from the control lines, both for reasons of speed and for simplifying control logic. In data communication systems, on the other hand, it is necessary to send both data and control information in the same channel so as to minimize channel costs and to permit the use of existing, widely available, channels without any special modifications. Most commonly available communication facilities are serial in nature. Even on parallel communication links the additional bit paths are almost invariably used for additional data bits rather than for control purposes. Thus, nearly all data communication techniques are based on separation of control from data signals in the time domain rather than in the frequency or space domain. Because of this temporal separation of control and data information, and also because many stations may share a communication link, a station can not simply emit messages onto the link at arbitrary times. It must monitor the link traffic and send messages only at appropriate times. Further, messages must be appropriately

structured so that the receiver can distinguish control information from data. The appropriate times and message structures are defined by a protocol that is established to allow orderly data flow among the stations that use a data communication link.

This set of rules, which governs the activity on a communication link, is summarized in terms of the following classes of items:

1. data communication control procedure
2. data link control procedure
3. line control
4. line discipline
5. message discipline

The state of a time slot is specified by the STATUS byte, the first byte of a time slot, according to the code shown in Figure 8.

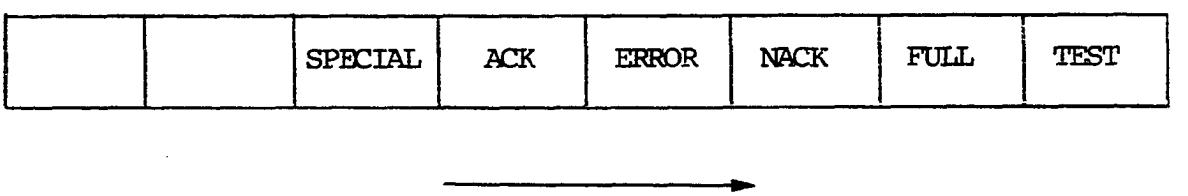


Figure 8. STATUS information.

Each bit contains the following information:

TEST: When this bit is set, it indicates that the time slot

is in test condition by the master repeater station and the time slot does not have valid data for the interface.

FULL: This bit specifies that the time slot is carrying a message.

NACK: This bit specifies the result of the transmission on that time slot. When this bit is set, it indicates that the message on the time slot was not received by the required destination(s).

ERROR: This bit is set when any transmission error has been detected either by polynomial redundancy check or by character sync pulses check or both.

ACK(EXIST): This bit indicates the result of a transmission by setting it when the message on the time slot has been received by the requested destination(s). When both NACK and ACK bits are set, this indicates that some of the requested destinations received the message correctly and rest of them could not. This happens for the case in which multiple destinations are requested such as a broadcasting message.

SPECIAL: When a transmission error has been detected on a time slot which carried a regular message to a process, this bit will be used for checking if the error has been detected after the destination or before the destination. If the error has been detected before the destination, the sender retransmits the message. Otherwise, retransmission of the message is not needed, because the required destination al-

ready received the message. This provision has been made to prohibit multiple communication contracts for a single communication contract requirement.

Different combinations of these bits and the related information are shown in Table 1.

Table 1. STATUS information

STATUS	Information
xxxxxxx1	The time slot is in test condition.
xxxx0x10	The time slot has a valid message.
xxxx0110	The message on this time slot has not been acknowledged by at least one required destination.
xxx10110	Some of the required destinations received the message on this time slot and some of them could not receive the message.
xx010010	The message on this time slot is acknowledged by the requested destination(s).
xxxx1x10	Transmission error has been detected on this time slot.
xx100010	The time slot is being used to check whether the previous transmission error on this time slot has been detected before or after the requested destination.

Functional Descriptions

The interface developed as shown in Figure 9 is a multiprocessor system which has three small processors working concurrently for different operations. These three processors are an Output Sequencer, a Table Matcher and a Micro-processor.

The Micro-processor is a commercially available low cost single chip LSI general purpose computer. The Micro-processor is responsible for data transfer to and from the devices that are connected to an interface. It also must provide a partial scan of the text that is entered from a device to recognize certain commands or requests for service. At the very least, these need to include some method of resource request and resource release capability. The Micro-processor also must handle the breakup of a message into 10 byte sections that can be put into a slot as well as the reassembly of these sections at the destination. When a slot is filled, the source, destination and op code information must be provided by the Micro-processor before the message is marked ready for the interface to send.

The Output Sequencer transmits a message from a selected source buffer under the control of the Table Matcher, one byte at a time, when the data transmission is required. The rest of the interfacing problems are to be handled by the Table Matcher. The Table Matcher is particularly effective

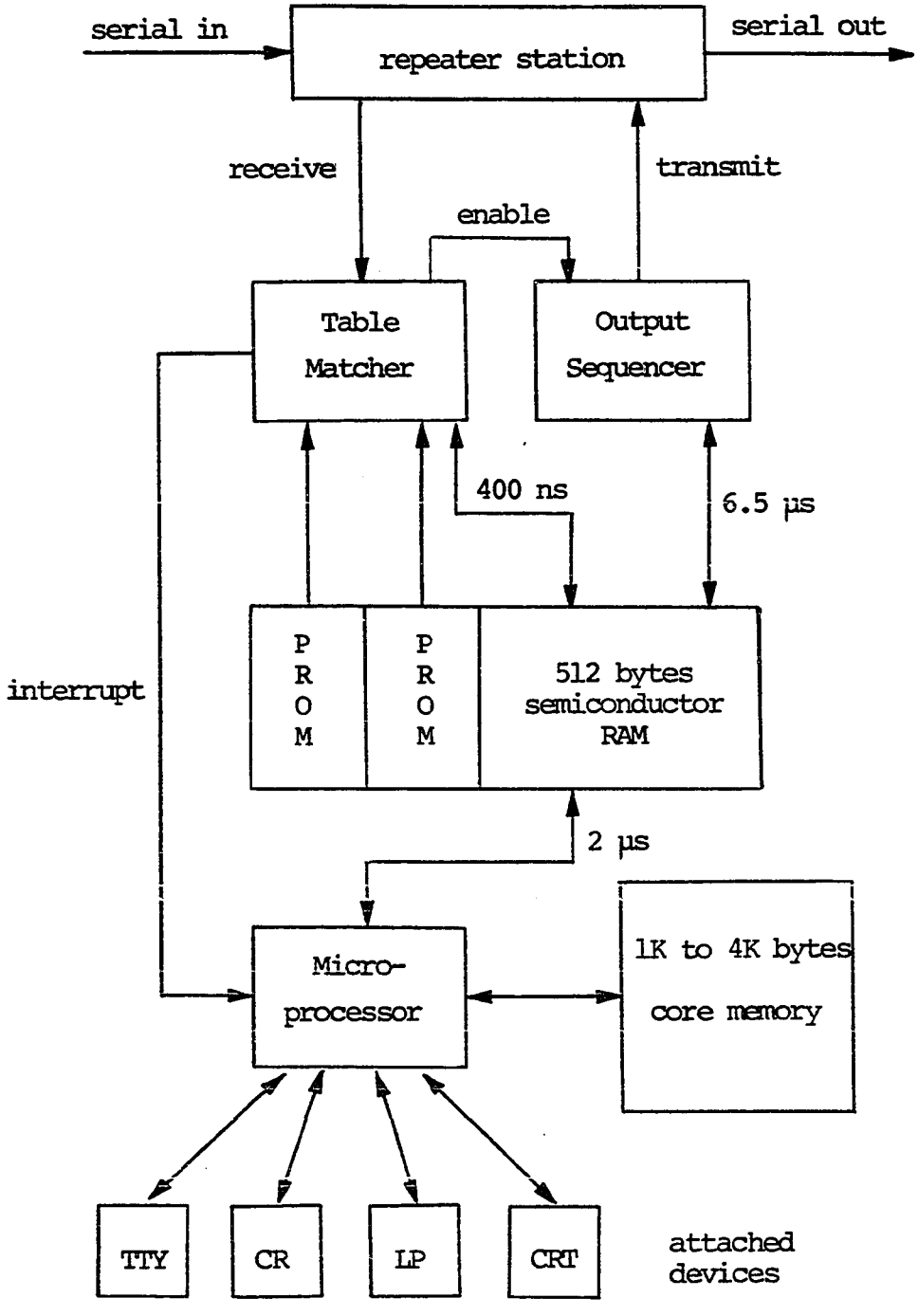


Figure 9. Functional block diagram of the interface.

in searching table entries in memory sequentially. A success or failure indication as well as the match location in the table will be returned when the Table Matcher finishes a table lookup. During periods of time in the processing of a slot when the Table Matcher is not needed by the interface, it will be available to the Micro-processor.

Three different types of memories, a core memory, a semiconductor random access memory and a programmable read only memory are used in this interface. A 512 byte random access memory with 60 ns memory cycle time is the actual communication region among the Micro-processor, Table Matcher and Output Sequencer. Memory service requests by these three different processors are served on a first come, first serve basis. The Micro-processor for most applications will be the Intel Micro Computer (Intel 8080) which has a 2 microseconds machine instruction cycle time and an 8 bit data bus and a separate 16 bit address bus. Therefore one can expect that this Micro-processor will access the memory every 2 microseconds when the Micro-processor is executing from this memory.

The Output Sequencer will access this memory every 6.5 microseconds when data transmission to the net is required, because the time difference of the two adjacent data bytes is 6.5 microseconds. The Table Matcher is the processor which places the greatest demand on this memory. In its busiest

state, such as the table searching state, the Table Matcher will access the memory every 400 ns. Since these three processors are working concurrently on the same memory and memory service requests are to be served on a first come, first serve basis, the worst case memory service waiting time of a request is twice the memory cycle time or approximately 400 ns.

A 1K to 4K core memory is used for the program memory for the Micro-processor as well as the buffer area for the attached peripheral devices. The present memories, FABRI-TEK MM 4096 x 24 - 1.75, have a memory cycle time of 1.75 microseconds. The Micro-processor can access either the semiconductor random access memory or this core memory depending upon the requested memory address. In other words, for the Micro-processor, these two different memories may be regarded as a single memory which has consecutive addresses with the semiconductor memory to have addresses from 0 to 511 and the core memory to have addresses 512 to 1023(4507). The core memory can be accessed only by the Micro-processor.

There are two separate programmable read only memories to govern the Table Matcher operation and all other interfacing operations. One has 512 bytes (12 bits per byte) which will contain 20 different encoded procedures to direct the interface operations depending upon the processing requirements to be done. The other has 256 bytes (12 bits

per byte) which contains 40 basic functions that will be used as branch and return subroutines by the above procedures. The procedure routines in the bigger ROM are basically composed of a sequence of subroutine calls to the smaller ROM. These two separated programmable read only memories will be discussed in detail later.

Types of Messages

There are 8 classes of messages that can be sent and these classes can be divided into three groups which include regular messages, system messages and special messages depending upon the destination(s) accessing modes. The low order three bits of the OP CODE byte which is the second byte in each time slot are used to indicate which type of message is being sent.

Regular Messages

These are messages for one to one data communication between a sender and a receiver, which is mostly used for general data ccmmunication on the net. Two different regular message types are provided as follows depending upon the destination accessing modes.

a) regular message to a process name: Only one device which has the requested process in it can receive this message even though more than one matching process is active on the net. The first station in which the requested process is available

Table 2. OP CODE and message types

OP CODE	Message Type
xxxxx111	a broadcast message to all active stations
xxxxx101	a system message to a 'process'
xxxxx001	a system message to a device address
xxxxx100	a regular message to a 'process'
xxxxx000	a regular message to a device address
xxxxx010	a special message to synchronize system destination buffer for a broadcast message
xxxxx110	a special message to synchronize system destination buffer for a system message to a 'process'
xxxxx011	a special message to free system destination buffers after a system message transmission is not satisfied

receives the message and makes a contract with the sender of the message for further communications. In other words, the interface, where the available matching process exists, updates the allowed source byte associated with the process name to the sender's identification and transmits the device address, where the process exists, in the 6th byte of the same time slot to the original sender. Therefore, subsequent communications between the sender and the receiver can be made by the device address instead of the process name.
b) regular message to a device address: Only the device indicated on the destination byte (the third byte of each time slot) can receive this message when the source is allowed and the assigned destination buffer to that device is empty. If the message was correctly received by the interface to which the requested device is attached, the receiving station interface will set the ACK bit on the STATUS byte of the time slot to inform the sender that the transmission is satisfied.

System Messages

In the group of source buffers and destination buffers in each interface, one 15 byte system source buffer and one 15 byte system destination buffer are provided to handle system messages. Every system message is to be originated from a system source buffer and sent to a system destination buffer. Here a system message is defined as a message which must be delivered to the required destination through a system destination buffer independent of the state of the required destination. Three different classes of system

messages are provided depending upon the destination requirements as follows:

a) system message to a device address: only the station where the required device exists can receive this message if the system destination buffer of the interface is available to receive the message.

b) system message to a process name: Every station which has the required process must copy this message in the system destination buffer to satisfy the transmission of this message. Upon receiving this message, the status of the filled buffer must be updated from 'empty' to 'full and not ready' until synchronization of these buffers is requested.

If there is a failure in receiving this message by any station which has the matching process, the original sender can distinguish this fact from the returning STATUS and OP CODE information of the time slot and will transmit the system message for a fixed number of times. The number of retransmissions of this message can be checked on a transmission counter which is a 4 bit binary counter provided inside the Output Sequencer of the interface. When every interface which has the matching process has received this message, the original sender transmits a special message to synchronize all the system destination buffers which received the message.

c) broadcasting message to all active stations: Every station currently active on the net must receive this broadcasting message to satisfy the requested broadcasting. Upon receiving this message, the status of the filled buffer must be updated from 'empty' to 'full and not ready' until synchronization of these buffers is requested. If there is any failure in receiving this message by any station, the sender will also retransmit this broadcasting message for a fixed number of times which is specified on the transmission counter. Since there is only one system source buffer, one transmission counter can be used to count the number of retransmissions for both broadcasting message and system message to a process. When every station has received the broadcasting message correctly, the original sender will transmit a special message to synchronize all the system destination buffers.

Special Messages

There are three different classes of special messages which are used to synchronize the system destination buffers depending upon the system message transmission results:

a) special message after a broadcasting is satisfied: When every station received a broadcasting message correctly within a fixed number of retransmissions, the original sender of the broadcasting message transmits this special message to enable all system destination buffers. Every interface

updates the system destination buffer status from 'full and not ready' to 'full and ready' upon receiving this special message.

b) special message after transmission of a system message to a process name is satisfied: When every station where the requested process is resident received the system message to the process correctly, the original sender will transmit this special OP CODE to synchronize those system buffers which are filled with the transmitted system message. Every interface which received this system message updates the system destination buffer status from 'full and not ready' to 'full and ready' upon receiving this special message.

c) special message for the case in which class b) or class c) system message transmission is not satisfied: Transmission of a broadcasting message can be satisfied only if all the active stations received the message and transmission of a system message to a process name can be satisfied only if every station where the requested process exists received the message. If either of the message requirements has not been satisfied after a fixed number of trials, the original sender transmits this special message to synchronize the related buffers. In other words, even if either of these transmissions has not been satisfied, there may be some system destination buffers which are already filled with the message and the related buffer status has been updated from

'empty' to 'full and not ready'. These updated buffer status must be released, because re-updating these buffer status from 'full and not ready' to 'empty' is not possible otherwise.

To prevent a type of a permanent blocking of these system destination buffers, these special messages are regenerated and transmitted automatically by the original sender if any transmission error has been detected on a special message. This will be discussed in detail later.

Tables

The 512 byte semiconductor random access memory is primarily to be used for 10 different tables, buffers and interrupt information storage.

Source Buffers and Destination Buffers

Two different sets of buffers are to be provided in the random access memory. One set is the source buffers which will contain data to be transmitted from the device attached to the interface to some other device(s) or process names. The other is a set of destination buffers which will receive the incoming message for the devices or process names which are currently active in that station. In these two sets of buffers, a system source buffer and a system destination buffer are reserved respectively for handling the system type messages and broadcasting messages. A buffer of any type is

15 bytes long and the number of buffers is changeable without changing any hardware.

Source Buffer Address Table (SBAT)

This table has the beginning addresses of the source buffers. Two bytes are used to specify the beginning address of each buffer. This SBAT is organized as shown in Figure 10. Since buffer addresses can be changed, the buffers can be located in any places inside the random access memory.

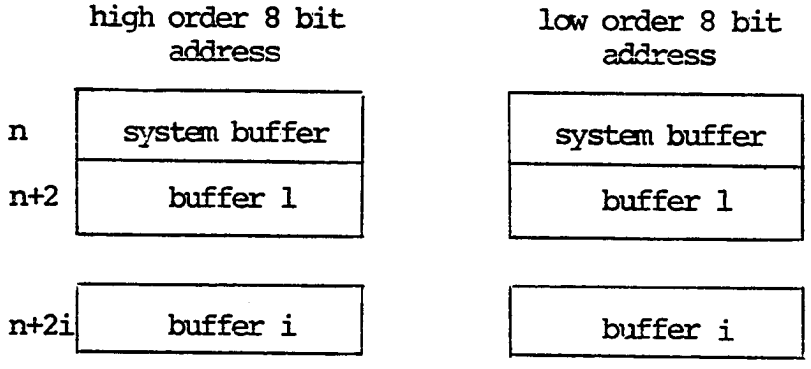


Figure 10. Organization of SBAT.

Destination Buffer Address Table (DBAT)

This table has the beginning addresses of the destination buffers. As in SBAT, two bytes are used to specify the beginning address of each buffer. This DBAT is organized as shown in Figure 11. Since buffer addresses can be changed, the buffers can be located in any place inside the random access memory.

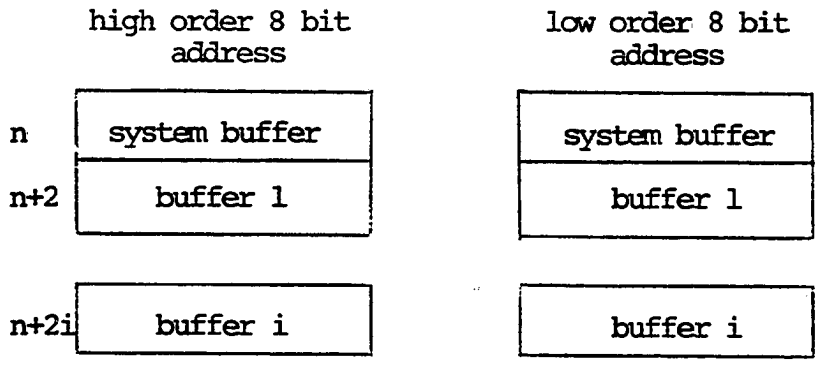


Figure 11. Organization of DBAT.

Buffer status associated with each buffer is stored in the Source Buffer Status Table or the Destination Buffer Status Table in the random access memory depending upon the type of a buffer. This buffer status must be checked by the interface before moving data into a destination buffer or transmitting data from a source buffer and also must be updated properly after moving data into a destination buffer or transmitting data from a source buffer. Transitions of buffer status will be considered separately depending upon the type of a buffer.

Source Buffer Status Table (SBST)

This table has information about the status of source buffers. The low order two bits of each byte in this table are used to indicate the three different states of a source buffer which are 'empty', 'full' and 'transmitted'. These are specified as 00, 01 and 11 respectively. When the

interface is processing a transmission routine, the interface needs to find a source buffer ready to be transmitted. As soon as the interface transmits the message which is in the selected source buffer, the source buffer status must be updated from 'ready' to 'transmitted'.

When the message originally transmitted from the source buffer returns to the originating station without transmission errors, the interface loads the first 6 bytes of the returning message into the original source buffer and interrupts the Micro-processor with a proper interrupt code. Upon receiving the interrupt, the Micro-processor looks at the STATUS and OP CODE byte of the source buffer in which the first 6 bytes of the returning message have been loaded and determines the result of the transmission. If the transmission was satisfied, the Micro-processor will update the buffer status from 'transmitted' to 'empty'. If the transmission was not satisfied, a decision on the further transmission of the message must be made by the Micro-processor. This decision is to be made considering the restored STATUS and OP CODE byte combinations which will be discussed later. The organization of this SBST is shown in Figure 12 and the transition diagram of a source buffer status is shown in Figure 13.

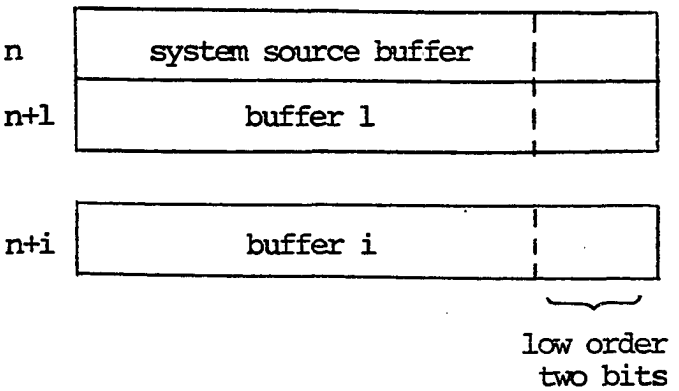


Figure 12. Organization of SBST.

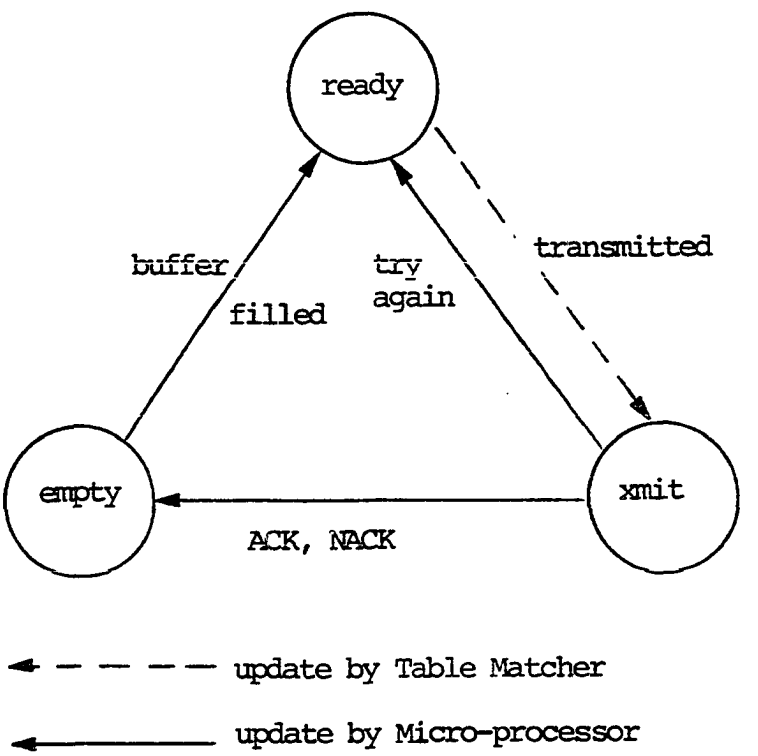


Figure 13. Transition diagram of source buffer status.

Destination Buffer Status Table (DBST)

This table has information about the status of destination buffers. The low order two bits of each byte in this table are used to indicate the status of a destination buffer such as 'empty', 'full and not ready' to process and 'full and ready' to process. These are specified as '00', '01', and '11' respectively. When the interface is processing one of receiving routines, the interface needs to check the status of the buffer assigned to the requested device or the requested process, to determine whether the buffer is available or not, before loading data into the buffer. The interface can load the data into the destination buffer only if the buffer status is empty. After loading the data, the interface needs to update the buffer status from 'empty' to 'full and ready' or to 'full and not ready' depending upon the type of a message loaded. Direct updating from 'empty' to 'full and ready' is allowed for a regular message and a system type message to a physical device address.

For a system message to multiple destinations, such as a broadcasting type system message or a system message to a process, this direct updating is not allowed because more than one system destination buffer needs to be filled with the message to satisfy the transmission of the message. In other words, it is not allowable to enable a destination buffer by updating the status from 'empty' to 'full and ready'

directly without considering the status of other destination buffers which must be filled with the message. Therefore, for this type of message, when the buffer is 'empty', the interface will load the data into the buffer and update the buffer status from 'empty' to 'full and not ready' temporarily and wait for the result of the transmission of the message. If the message has been loaded into all the required destination buffers correctly and the buffer status has been updated from 'empty' to 'full and not ready', the interface from which the message was originally transmitted will transmit a special message to synchronize the buffers where the message has been loaded. Upon receiving this special message, the related buffer status is to be updated from 'full and not ready' to process to 'full and ready' to process.

However, when the transmission of the message is not satisfied, for a fixed number of retransmissions, the interface in the originating station of the message will transmit a special message to clear the buffer status, which is updated from 'empty' to 'full and not ready', from 'full and not ready' to 'empty'.

Transition of a destination buffer status from 'full and ready' to 'empty' is to be done by the Micro-processor when the Micro-processor has finished the required processing of a buffer whose status was 'full and ready' to process. This

synchronization of system destination buffers will be discussed again as well as the number of retrials for the satisfactory transmission of the message. The organization of this DBST is shown in Figure 14 and the transition diagram of a source buffer status is shown in Figure 15.

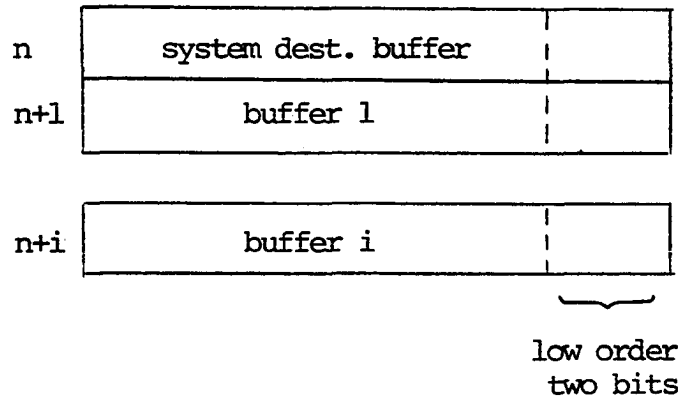
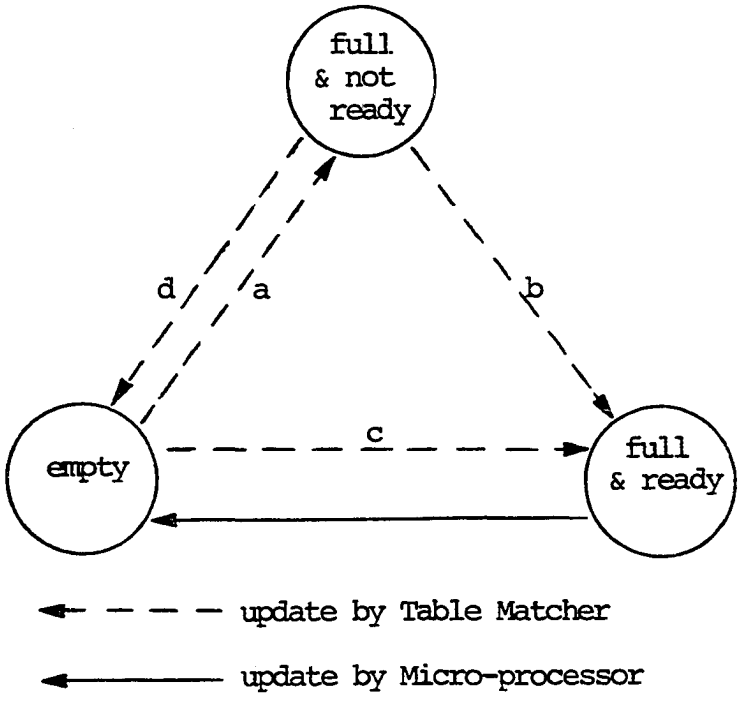


Figure 14. Organization of DBST.

Table Address Table (TAT)

This table has the beginning address of each table, number of entries and the amount of increment for next entry in the table. Two consecutive reserved words(one byte per word) are used for the beginning address of each table. The first reserved word contains the high order 8 bits address and the following byte contains the low order 8 bits address. Therefore the beginning address of each table is specified in a 16 bit address space where only 9 bits are used to specify the physical address of each table in the 512 byte RAM.



a; received a system message to a process or a broadcasting message.

b; received a special message after a system message transmission is satisfied.

c; received a regular message or a system message to a process.

d; received a special message after a system message transmission has not been satisfied.

Figure 15. Transition diagram of destination buffer status.

One reserved word broken into two parts is used for the number of entries and the amount of increment for the next entry in each table. The number of entries in the table is stored in the low order 5 bits in the third byte and the amount of increment is stored in the high order three bits in the same byte. The maximum allowable size of a table is, therefore, 256 bytes which has 32 entries with 8 bytes of an increment for the next entry. The Table Matcher can access this TAT for preparing a table lookup by direct addressing mode instructions provided, such as LOAD or STORE. Having this TAT, all the related information can be changed without changing any hardware. The reserved memory locations and the associated usages are shown in Appendix D.

Time Slot Acknowledge Table (TSAT)

This table has information about the time slots in which messages have been transmitted to the net by the interface as well as the source buffers from which these messages were obtained. This TSAT is organized as shown in Figure 16.

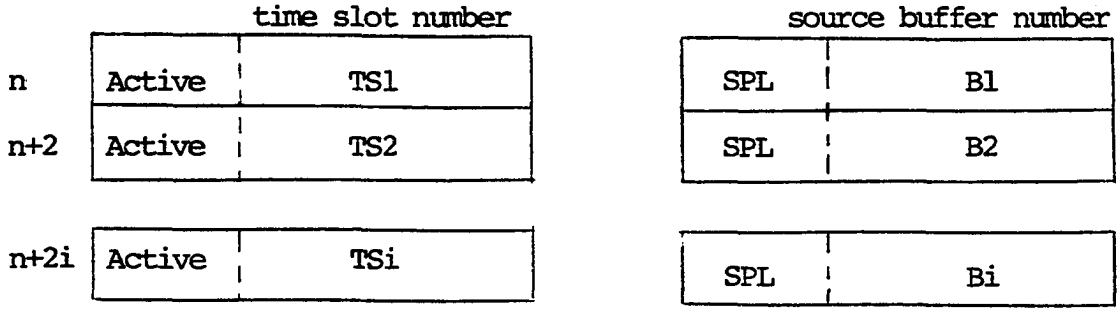


Figure 16. Organization of TSAT.

As explained earlier, there are 32 different time slots specified from binary 00000 to binary 11111. The current time slot number active on the associated repeater station can be obtained from the high order 5 bits of 9 bit character counter provided in the repeater station. The high order 3 bits (Active) of a time slot number are used for indicating the current status of the byte. In other words, when these Active bits are on, the low order 5 bits of the time slot number byte contains a valid acknowledged time slot number. This provision has been made to distinguish the time slot number 00000 from the empty state of the time slot location. The following byte of each time slot number byte contains the source buffer number which has been transmitted to the net on the time slot specified in the time slot number byte. The low order 5 bits of each source buffer number byte are used to specify the source buffer transmitted on the time slot.

Up to 32 different source buffers can be specified in these low order 5 bits.

The high order 3 bits (SPL) in the source buffer number byte are used to indicate one of the three special messages. Since the special messages which are used for synchronizing the system destination buffers for some system messages are partially generated inside the Table Matcher and are transmitted with the necessary source buffer message instead of originated totally from a source buffer, some kind of protection from transmission errors on these special messages should be provided. The high order 3 bits of the source buffer number byte are used for this purpose. When the interface needs to transmit a special message, the interface will load 100, 010 or 001 into these 3 bit positions depending upon the special message transmitted.

The multiplexing technique used in this system is a restricted demand multiplexing. No time slots are dedicated to a specific station. However, the number of time slots which can be used by a station are limitted due to the size of this TSAT. In other words, any station can use any time slot as long as the station has a source buffer ready to be transmitted and an empty location in TSAT is found. This provision has been made to provide the flexibility in determining the number of time slots which can be used by a station depending upon the communication demands in each

station. The communication demands of a station are the function of the answers to the following questions: "How often does the station access the net and how much of data are to to be transmitted from the station in unit time?"

When the interface has a source buffer ready to be transmitted to the net, the current time slot is not being used by any other station, an empty location in TSAT was found and no transmission error has been detected on the time slot between the previous station and the current station, then the following operations will be performed by the interface: store the current time slot number and the ready source buffer number in the empty location in TSAT; sets the Active bits; resets the SPL bits and transmits the message in the ready source buffer to the net. When one of the three special messages is transmitted for synchronizing system destination buffers, the three SPL bits are updated to 100, 010 or 001 depending upon the special message transmitted.

Selecting a source buffer which is ready to be transmitted is done as follows. A source buffer pointer which is a four bit binary ring counter is provided to control the SBST checking. The current contents of this source buffer pointer indicates the next source buffer to be checked. When the Table Matcher needs to check SBST to select a ready source buffer to be transmitted, the Table Matcher initiates SBST checking from the buffer number indi-

cated by the source buffer pointer. If the status of the buffer indicated by this source buffer pointer is not ready to be transmitted, the Table Matcher will increment the contents of the source buffer pointer by one and continue checking the SBST. This process will be repeated until the Table Matcher finds a source buffer which is ready to be transmitted or the whole SBST has been checked through. In either case, the contents of the source buffer pointer is to be incremented by one, indicating the source buffer to be checked first in the next SBST checking. This source buffer pointer is provided to assign an equal probability to be checked, associated with each source buffer. This source buffer pointer counts up to the number of source buffers provided in each interface. For example, the source buffer pointer in a station where 5 different source buffers are provided counts repeatedly from binary 0000 to binary 0100.

When a time slot that was filled by the interface returns with a satisfied transmission status, then the time slot number will be deleted from the TSAT and an interrupt with a proper interrupt code will be given to the Microprocessor to inform it of the satisfied transmission of the related source buffer. When transmission errors have been detected on an incoming time slot, which was originally transmitted by the interface, the interface will retransmit the associated source buffer without updating this TSAT if

the SPL bits in the source buffer number are 000. If these SPL bits in the associated source buffer number are 100, 010 or 001, the related special message will be generated and retransmitted with the necessary messages in the source buffer without updating TSAT.

Process Name Table (PNT)

This table has the names of processes that are available in the devices that are attached to the interface. Each process name has three associated addresses as shown in Figure 17: a destination address which indicates the location of the device in which the process exists; a source address which indicates the source identification allowed to communicate with this process; a buffer address which indicates the buffer number assigned to the process.

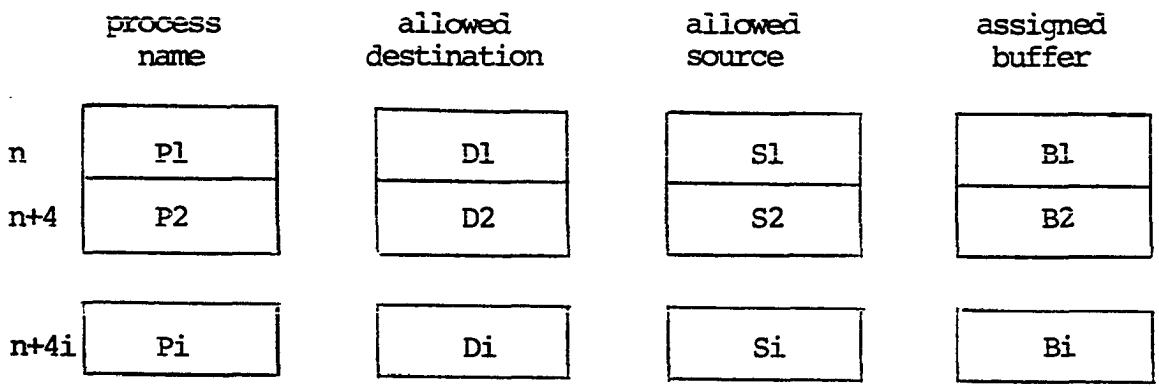


Figure 17. Organization of PNT.

Since multiple process names, i.e., more than one identical process name in a station, are allowed, all the process names in PNT must be checked for a required process name. The allowed source byte is to be used to make communication contract between a sender and a required process by storing the sender's identification in it. If the content of an allowed source byte is zero, any source can access the associated process name. Otherwise, only the source identified by the allowed source byte can access the process name.

When a required process name was found in the PNT checking, the interface needs to check the allowed source byte to determine whether the source accessing the process name is allowed to that process name or not. If the source is allowed to that process, the status of the assigned buffer must be checked to determine whether the assigned destination buffer is available to receive the message. If the source is allowed as well as the assigned buffer is empty and no transmission error has been detected on the time slot being processed, the interface will copy the message into the assigned destination buffer, update the buffer status properly, update the allowed source byte if necessary by loading the original sender's name in it and send out the allowed destination byte of the accessed process name in the 6th byte on the same time slot.

A communication contract has been made between the original sender and the accessed process name in this way which allows subsequent communications between these two nodes to be made in terms of the device address instead of the process name.

Allowed_Destination_Table_(ADT)

This table has the device addresses, defined on the net, which are attached to the interface. Each allowed destination has an associated allowed source and an assigned buffer. The organization of this ADT is shown in Figure 18.

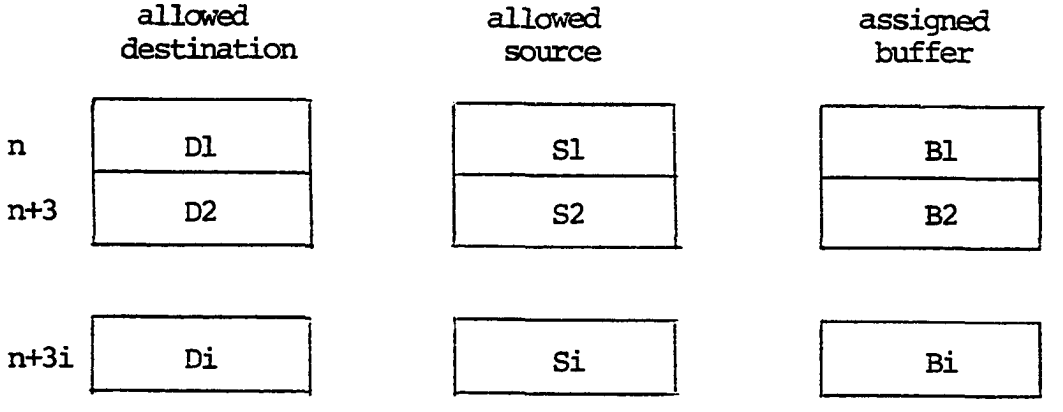


Figure 18. Organization of ADT.

The usage and operations on this ADT are the same as those on PNT except that ADT is used for a message to a device address while PNT is used for a message to a process name. Every device address specified in each allowed desti-

nation byte is unique. In other words, no two allowed destinations are same while multiple processes are allowed in PNT. When the requested destination was found in ADT lookup, the allowed source and the assigned buffer status needs to be checked and updated properly same as in the PNT case. Practically, the allowed source and assigned buffer of the Process Name Table and ADT can be overlapped.

Table Updating Table (TUT)

Before the interface begins to operate on the incoming time slot, the interface needs to check the error bit in the STATUS byte to determine if a transmission error has been detected or not on the time slot. If a transmission error has already been logged or if no errors have been detected, the interface will select a procedure depending upon the processing requirement, to continue processing of the slot. However, during the operation of the selected procedure, if the interface needs to update a table entry, the interface must not update the table entry right away because at that point of time the information on which the interface has worked might not be a valid message. In other words, until the interface gets the results of the polynomial redundancy check, even though the current STATUS byte says there has not been any transmission error, there might be a transmission error between the previous repeater and the current repeater station.

Therefore, when the interface needs to update a table entry, instead of updating that table entry right away, the interface will store this table updating information into the TUT and wait for the result of the transmission error check. This information includes updated data, memory addresses where the updated data will be stored, etc. If no transmission error is detected, the interface will update the table entry (or entries) according to the information saved in Table Updating Table. If any transmission error is detected, the interface will neglect all the information, in the TUT, obtained from the message and further consideration of this message is not needed. The interface can access this TUT by direct addressing mode instructions such as LOAD or STORE. The reserved memory locations for TUT and the associated usages are shown in Appendix D.

Interrupt Information Table (IIT)

Communication between the Micro-processor and the Table Matcher is to be done by an interrupt scheme and this table has interrupt information such as interrupt codes and additional information if necessary. Whenever the interface updates a buffer status or fills a buffer or empties a buffer, the interface will interrupt the Micro-processor and let the Micro-processor know which buffer is filled, or emptied or which buffer status is updated respectively. All these interrupt codes and necessary information will be stored in

this table before the interrupt.

When the Table Matcher is not busy and the Micro-processor wants to check some of the tables described, then the Table Matcher can be used by the Micro-processor for a fast table checking operation. Before the Micro-processor can enable this idle Table Matcher for its job, the Micro-processor must give the table checking information (such as beginning address of the table, number of entries of that table and the number of bytes in an entry increment) to the Table Matcher by storing it in the IIT. When the Table Matcher finishes the job for the Micro-processor, the Table Matcher will give the result of its checking to the Micro-processor with a proper interrupt code to represent this result. When the Table Matcher finds a match during the process of Micro-processor's job, additional information of the memory locations where the match was found will be given to the Micro-processor through this Interrupt Information Table.

The interface can detect a power failure of the local power supply before the power becomes unusable for normal interface operations. Detecting the power failure, the interface will interrupt the Micro-processor with a proper interrupt code to denote the power failure.

The interface is to be started by pushing a switch provided in the front panel by a console operator. When the

interface is started, the Table Matcher will interrupt the Micro-processor with a proper interrupt code to activate the Micro-processor. These start-up and shut-down procedures will be discussed later in detail as well as the related interrupt codes and the instructions. The reserved memory locations for IIT and the associated usages are shown in Appendix D.

Processing Requirements

Depending upon the type of message passing through the repeater station and depending upon the state of the interface, operation requirements for the interfacing problems are all different. Therefore, the interface needs to select a procedure which meets the processing requirement and needs to follow that procedure afterwards. Most of the procedures can be selected by checking the incoming STATUS and OP CODE bytes while some procedures depend on the state of the interface such as the procedure to allow the Micro-processor use of the idle Table Matcher, the shut-down procedure for the case of the power failure and the start-up procedure to initiate the normal operations of the interface. Flowcharts for different procedures are shown in Appendix A as well as a sample explanation of the interface actions for the transmission procedure.

Different procedures selected according to the processing requirements are described briefly.

Shut-down procedure: To protect the information currently stored in the semiconductor random access memory from local power failure, an automatic shut-down procedure is provided. When power goes down, a built-in power failure detection device will detect this power failure and hold the power for a required period of time. During this time, the Table Matcher will free the time slots entered in the TSAT of that station and the Micro-processor will be interrupted to dump the current information in the semiconductor RAM to core memory. This way, information stored in the semiconductor memory can be protected even in the case of power failure.

Start-up procedure: Initiating the interface functions will be done automatically by the start-up procedures. When the normal operation of the interface is requested on the front panel START switch, operated by the console operator, the Table Matcher will interrupt the Micro-processor to enable the initiation procedure of the interface. Upon receiving this interrupt, the Micro-processor will transfer the necessary information, to be stored in the high speed semiconductor memory, such as table address, table information, previously saved information from power failure, process names, etc. When the Micro-processor finishes this procedure, it will inform the Table Matcher of this fact.

Then the interface will complete initialization and enter into normal operation mode.

Allow Micro-processor to use Table Matcher: As discussed before in the Interrupt Information Table section, when the Table Matcher is idle and the Micro-processor has a table checking job, then the Micro-processor can enable the Table Matcher to do the Micro's job for high speed checking operation. When the Table Matcher finishes the job for the Micro-processor, the Table Matcher will give the result of its checking to the Micro-processor with a proper interrupt code to represent this result.

Transmission Error procedure: Transmission errors on the net can be detected by either polynomial redundancy check or by character synchronization pulses. If any transmission error has been detected on a time slot, the station which detected the transmission error loads its station number on the destination byte of the time slot, resets the log bit and sets the error bit on the STATUS byte of the time slot. The master repeater station of the net, then, logs the error and sets the log bit of the time slot. When the time slot with transmission error returns to the originating interface after logging the error, the interface retransmits the message except for the case of a regular message to a process name.

For the case in which a regular message to a process has been in error, the originating interface checks if the

transmission error has been detected after the requested process or before the requested process. This can be done by retransmitting the message with the SPECIAL bit of the STATUS byte on and checking the returning SPECIAL bit. If the transmission error has been detected after the requested process received the message correctly, the SPECIAL bit in STATUS byte of the returning message will be reset and ACK bit is set by the station where the requested process is. This indicates the satisfactory transmission of the message even though errors have been detected. If the error has been detected before the requested process received the message, the SPECIAL bit in STATUS of the returning message remains set which indicates retransmission requirement of the original message. This provision has been made to prohibit multiple communication contracts for a single communication contract requirement.

Transmission procedure: When the time slot currently active on the attached repeater station is empty, then the interface initiates the transmission procedure which requires selecting a ready source buffer to be transmitted, checking for an empty location in TSAT. If all these checks are positive and no transmission error has been detected between the previous repeater station and the current repeater station, the interface will transmit the message in the selected source buffer and update the necessary table entries, such as SBST,

TSAT, etc.

Message Receiving procedure: There are 10 different message receiving procedures, each of which is selected by the STATUS and OP CODE combination of an incoming time slot. Rather than giving an extensive treatment of these procedures by showing the actions of the interface, the basic communication philosophy is described. Details of these procedures are shown in Appendix A in terms of flowcharts.

The interface needs to look at the STATUS and OP CODE byte of each incoming time slot to select a necessary procedure to process the slot. The destination byte which is the third byte of a slot has a device address or a process identifier which is an argument in checking the Allowed Destination Table of device address or the Process Name Table of process identifiers. If a match is not found in this table lookup, further consideration of this slot is not needed. Otherwise, the availability of the required destination and the buffer assigned to it for input needs to be checked. If these checks fail, the processing of this slot is terminated and NACK bit of the STATUS byte is set. This updated STATUS byte will be sent out to the net as soon as the repeater indicates that no error has been detected on the time slot. If any transmission error has been detected and indicated to the interface by the repeater, the interface must not send out the updated STATUS byte because the message in the time

slot processed by the interface might be an invalid message containing a transmission error between the previous repeater and the current repeater station.

However, if all checks are positive and the repeater indicates that no transmission error has been detected on the time slot, the message is copied into the proper destination buffer and the information saved in the Table Updating Table is used to update the necessary table entries, such as buffer status, allowed source, etc. Then the original message, with status indicating the success or failure of the transmission, is passed along the loop to the original sender. The interface in the originating station can get all the different information about the returning message by looking at the STATUS and OP CODE byte combinations. These combinations are shown in Table 3 and the related informations are explained in Table 4.

Local communication capability is provided on the transmission of a message. In other words, a message originated from a source buffer can be delivered to a device or a process existing in the same station, when the message is returning to the originating station. This local communication capability reduces a considerable amount of the Microprocessor's job, because most of the interface functions must be duplicated in software for handling the local communication problem by the Micro-processor otherwise. Since the

whole net is used for a local communication, this scheme looks undesirable at first glance. However, once a communication contract has been made between the sender and the receiver locally, further communications are to be done not through the net but by the Micro-processor's control locally. In other words, after the communication contract has been made between the sender and the receiver, both of which exist in a same physical station, the Micro-processor in the same station may be used for controlling data transfer between the sender and the receiver without accessing the net.

Table 3. STATUS and OP CODE combination

OP CODE (3 bit) / STATUS	000	001	010	011	100	101	110	111
xx000000	The time slot is empty.							
xx000010	A2	B2	C2	D2	E2	F2	G2	H2
xx000110	A3				E3			
xx010110	A4	B4			E4	F4		H4
xx010010	ACK	ACK			ACK	ACK		ACK
xx100010					E6			
xxxx1xx0	The time slot is in error.							

Table 4. Information of STATUS and OP CODE combination

STATUS/OP CODE	Information
A2	The requested device address does not exist on the net.
A3	The requested address exists on the net, but the source is not allowed for that device address.
A4	The requested device address exists on the net and the source is allowed for that destination, but the assigned destination buffer is busy.
B2	The requested device address does not exist on the net.
B4	The system destination buffer which is in the station where the requested device address has been found is busy.
C2	Broadcasting type system message was satisfied because every station currently active on the net received the message correctly.
D2	Every station in which the requested process was received the message correctly.
E2	The requested process does not exist on the net.
E3	The requested exists on the net, but the source is not allowed for that process.
E4	The requested process exists on the net and the source is allowed for that process but the assigned buffer is busy.

Table 4. (continued)

STATUS/OP CODE	Information
E6	Retransmission of the message is required because the previous error of the message was occurred before the station where the requested process exists.
F2	The requested process does not exist on the net.
F4	At least one system destination buffer which is in the station where the requested process exists is busy.
G2	System message transmission can not be satisfied for the given number of trials.
H2	All stations are in power failure except the sender.
H4	At least one of the system destination buffers is busy.
slash	No valid information.
ERROR	Retransmission of the message is required if the error has been logged already.

DESIGN CONSIDERATIONS

As mentioned earlier, the interface processor is organized as a multi-processor system which has three small processors working concurrently on different operations of the interfacing requirements. Three different kinds of memories such as core, semiconductor RAM and PROM are provided in the interface. The RAM is the actual communication region among the three processors and the core is the program memory of the Micro-processor. The rest of the interface system, the processors and the PROMs, are considered in this chapter. Among the TTL logic families, the standard power components are selected for designing the interface to satisfy the compromise between the speed and cost requirements. Some of the schottky TTL family are used for timing control sections due to speed and accuracy reasons.

Programmable Read Only Memories

Two different sizes of Programmable Read Only Memories are provided as control memories. Twenty different procedure routines are stored in 512 bytes (12 bit per byte) PROM and forty different basic subroutines are stored in the 256 bytes (12 bit per byte) PROM. All interfacing problems are partitioned and encoded to be stored in these two PROMs following the instructions in Appendix B. These control codes are

shown in Appendix F.

Due to the desire to keep the instruction word short and to reduce the number of memory accesses for speed up the operations, a decision was made to use 12 bit instructions. All the instructions are 12 bits long and every instruction is a one byte (12 bit per byte) instruction, even for jump for instruction execution. Each PROM has its own address register and data register. Corresponding outputs of the data registers are OR-tied to form a 12 bit instruction bus. A PROM control logic is provided to control two PROMs, such as indicating an active ROM, etc. A block diagram for these ROMs and the control is shown in Figure 19.

There are two unconditional branch instructions which are CALL and JUMP. The CALL instruction causes a branch out to the subroutine in the 256 byte PROM starting at the specified address in the CALL instruction. When the subroutine return is required after the called subroutine processing is over, the PROM controller will activate the 512 byte PROM and disable the 256 byte PROM. The JUMP instruction causes a branch out to the specified location inside the 512 byte PROM. These actions are shown in Figure 20.

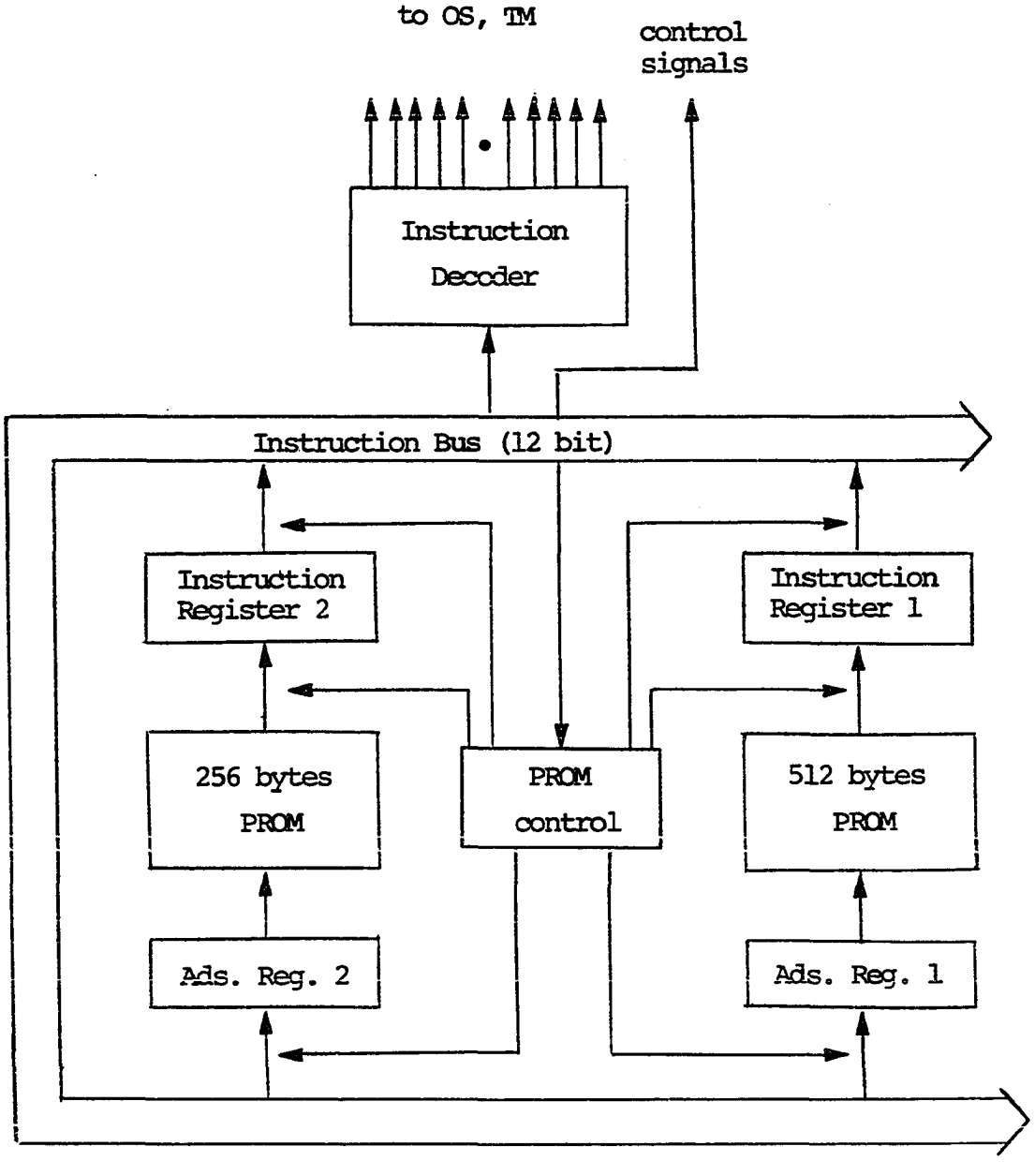


Figure 19. Two separate PROMs.

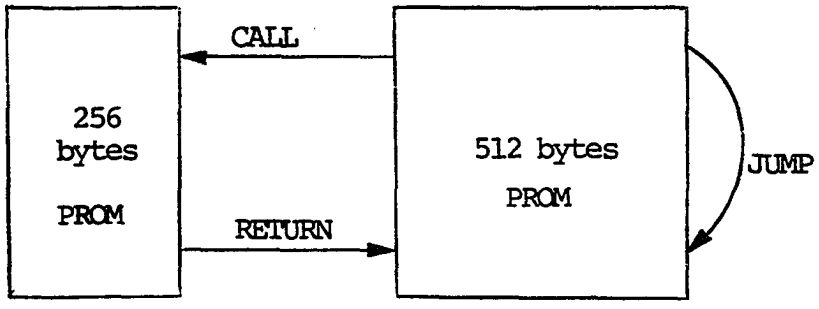


Figure 20. Directions of CALL, JUMP and RETURN.

Capabilities for multiple nesting subroutines are not provided because basic subroutines are defined such that such nesting is unnecessary. In defining the subroutines as described above, more memory space for storing basic routines is needed than if subroutine nesting capability was provided, because some of the subroutines are encoded as another basic routine. At first glance, this approach is not economical. However, considering the sharp reduction in cost of LSI components due to the rapid advances in semiconductor technology, this approach may be preferable to the conventional single memory control system. Its advantages are:
a) Fast speed: Every procedure is composed of 20 to 40 instructions. For any procedure, the number of subroutine call instructions is quite high and these CALL instructions request processing from the basic subroutine ROM. Thus, when

a subroutine CALL is finished, the next instruction of the procedure ROM is already available in the instruction register of the procedure ROM. This eliminates the need to save and restore processor status on a subroutine call because each ROM has its own instruction register.

b) Short address field: Since subroutine addresses are fixed (recall that the basic subroutine ROM is 256 bytes), the subroutine addresses can be specifed in 8 bits even if the size of the procedure ROM is expanded.

c) Flexibility: Most of changes, if required to modify the interfacing problems are to be expected inside the procedure routines instead of the basic subroutines.

Instructions stored in two ROMs can be subdivided depending upon the type of operation:

a) Multiple Operation Instructions: This type of instruction enables multiple operations, or functions, to be specified by a single procedure instruction. For example, ETM A (Enable Table Matcher type A) is an instruction which enables the Table Matcher to check the table requested, with the provided data and to return the location of the match. Therefore, reading data from the memory, checking for a match and incrementing the memory address for the next location to be checked is to be repeated until finding a match or the end of table is reached. This function is done in hardware initiated by a single procedure statement, thus saving time

by eliminating the overhead of executing the instructions necessary to provide the same result.

Another instruction of this type is RCV n (Receive n bytes of data from the repeater to memory) which will enable the following sequence of operations which is repeated n times: receiving one byte of data from the 16 byte temporary buffer, store it in the memory, and increment the memory address. The value of n can be 1 to 15. This way, table checking operations and data receiving operations, which are very time consuming operations, can be speeded up.

b) Double Operation Instructions: ADR (Add and Read) is provided because most of the addition operations accompany the memory read operation. Therefore, upon finishing the add operation, if the read bit is set, a read operation from memory will be performed. For most of the instructions, the last bit is a RETURN bit. That is, if the last bit is set, upon finishing the instruction execution, the subroutine return operation will be performed. This saves memory space and reduces the number of memory accesses, both of which speed up the processing of a slot.

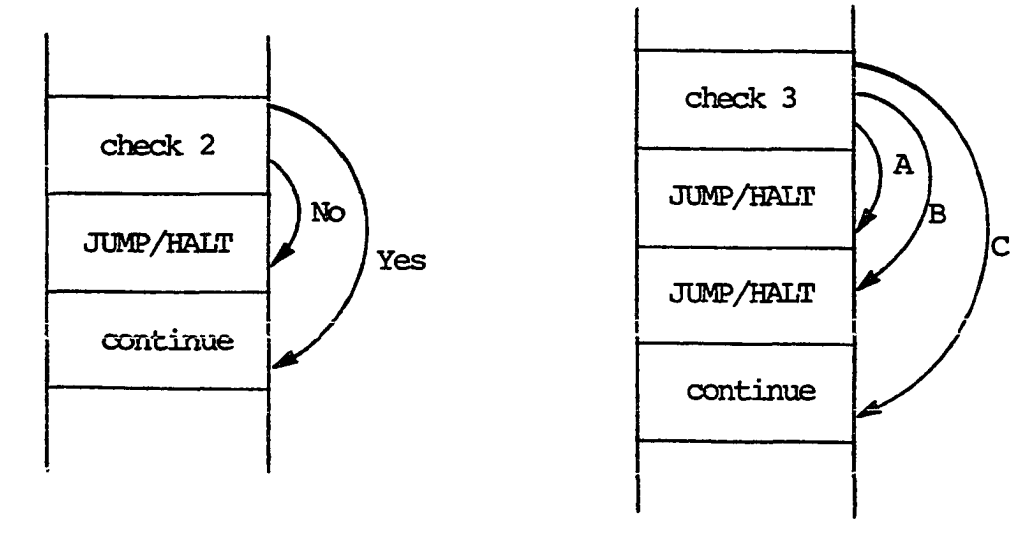
c) Single Operation Instructions: Some of the instructions, such as JUMP, MOVE, etc., require just a single cycle of operation.

A checking operation and subsequent conditional jump operation depending upon the result of the checking operation

are combined to be performed in one instruction instead of two instructions such as a conditional jump instruction followed by a checking instruction. Considerable amount of hardware to be involved in the condition flip-flops and logic for saving the states of the previous operation result can be saved in this way. There are two groups of checking instruction depending upon the number of branch-out corresponding to the checking result. Most of these checking instructions, such as ETM A, ETM B, CK1, CKAB, CKOP and CKXMITC, require a two-way branch-out depending upon the match or non-match of the checking result. However, one instruction, CK2, requires a three-way branch-out depending upon the information checked. Clearly, the instruction which requires a three-way branch-out can be converted to a sequence of two instructions, each of which requires a two-way branch-out depending upon the result of each checking. However, the instruction which requires a three-way branch-out was preferred, because this instruction was used much more than the other type checking instructions. In this way, the number of memory accesses can be reduced which also speeds up the processing of the interface operations as well as conserving memory space.

For those two-way branch-out instructions, the next instruction to be executed is to be fetched from the memory address obtained by incrementing the current address by one or

two depending upon the non-match or match of the checking respectively. For the three-way branch-out instruction, the next instruction to be executed is to be fetched from the memory address obtained by incrementing the current address by one, two or three depending upon the information checked. Examples of these actions are shown in Figure 21. For both cases, the current memory address is automatically incremented to get the memory address of the next instruction to be executed depending upon the type of checking instruction and the result of the checking.



two way branch out

three way branch out

Figure 21. Branch-out locations.

The complete descriptions of the instructions are provided in Appendix B as well as each instruction execution

time. Physically, these PROMs are implemented on a 14"x11" printed circuit board which has following hardware components:

1) 512 byte procedure PROM
2) 256 byte subroutine PROM
3) PROM controller
4) address register and instruction register of each PROM
5) instruction decoder
6) clock generator and timing phase decoding logics for instruction execution
7) parity checkers for the instruction decoder.

There are about 130 TTL Integrated Circuits (IC) provided on this board.

Table Matcher

The Table Matcher is specially designed to fit the high speed sequential table searching operation. Also, the Table Matcher is responsible for handling overall interfacing problems controlled by the encoded procedures and subroutines stored in the two separate PROMs. The Table Matcher contains the following hardware components.

a) one 8 bit STATUS byte register and status updating logic
b) STATUS and OP CODE byte decoding logic to determine

a procedure to use in processing a slot.

c) two 8 bit address registers

d) one 8 bit temporary register which contains the table checking information such as number of table entries and the amount of an entry increment.

e) two 8 bit data registers

f) character counter and bit counter decoding logic to synchronize interface operations to the repeater stations.

g) two 16 byte buffers for storing the message passing through the repeater.

h) a 12 bit binary adder and the adder control

i) a 5 bit binary counter to count the number of checks being done or number of data bytes received.

j) a 4 bit source buffer pointer which is a ring counter specifying which source buffer needs to be checked for next transmission request

k) a 4 bit binary transmission counter which counts the number of transmissions on a same system message.

l) an automatic table checking controller

m) an automatic data receiving controller

n) checking logic to compare the contents of two data registers

o) a timing controller for buffering a message from the repeater and clock generator

The block diagram of the Table Matcher is shown in Figure 22. Rather than describing all the hardware actions involved, some of the logic and operations involved in the automatic table checking will be discussed.

For an automatic table checking operation, information required for checking the table must be preloaded into the proper registers. This information includes the beginning address of the table, number of table entries, amount of entry increment and the data to be compared.

The beginning address of the table is loaded into the two address registers, RB and RA, and the number of table entries and the amount of entry increment are loaded into the 8 bit data register, RC. Here the address register RB contains the high order 4 bit address and the address register RA contains the low order 8 bit address. The high order 3 bits of the register, RC, contain the number of bytes in entry increment and the low order 5 bits contain the number of entries to be checked in the table. The data to be compared for a match is loaded in the data register, DB. One check counter is provided which is a five bit binary counter counting number of checks being done. This counter must be cleared before an automatic checking operation is initiated.

After all this information is preloaded, when an automatic table checking operation is initiated by decoding proper instruction, the Table Matcher will read one byte data

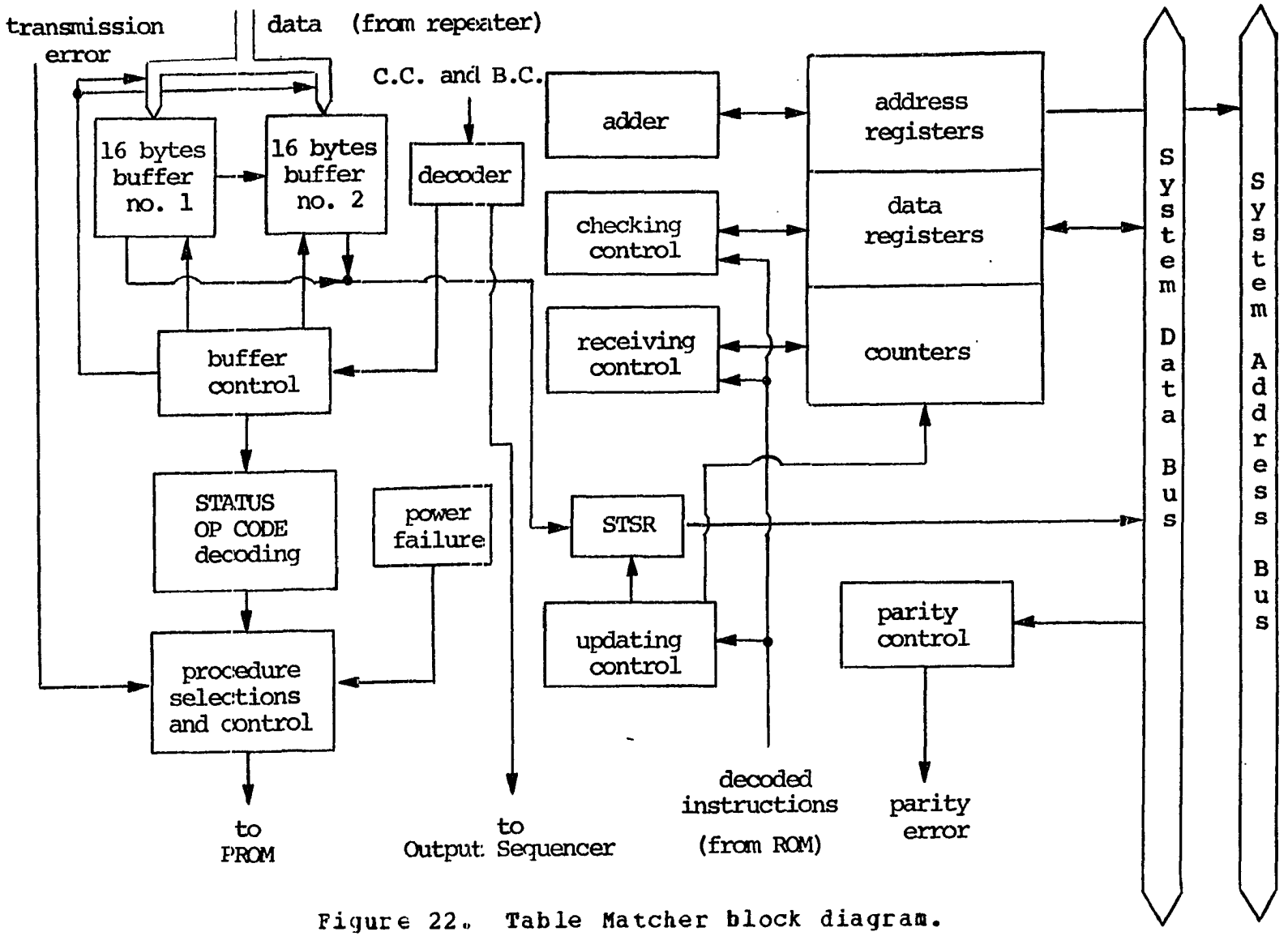


Figure 22. Table Matcher block diagram.

from the memory address specified by the contents of RB and RA and load it into data register DA. Then the contents of DB and DA are compared. If they are not matched, the next entry address of the table is calculated through the 12 bit binary full adder. The amount of an entry increment which is in the high order 3 bits of RC is added to the current address, the contents of RB and RA, through the adder and the sum is loaded back into the RB and RA. Whenever a new address is calculated, the contents of the 5 bit check counter is incremented by one indicating number of entry checks being done. This process, read-check-increment, is continued repeatedly until a match is found or until the contents of the check counter is matched with the low order 5 bits of RC, indicating that all the table entries are checked.

When a match is found during this process, the contents of RB and RA is saved in the Table Updating Table with the properly updated data. The hardware logic involved in this process is shown in Figure 23. Physically, this Table Matcher is implemented with three 14"x11" printed circuit boards each of which contains about 100 IC packages.

Output Sequencer

The Output Sequencer contains the following hardware components:

1) one 8 bit data register

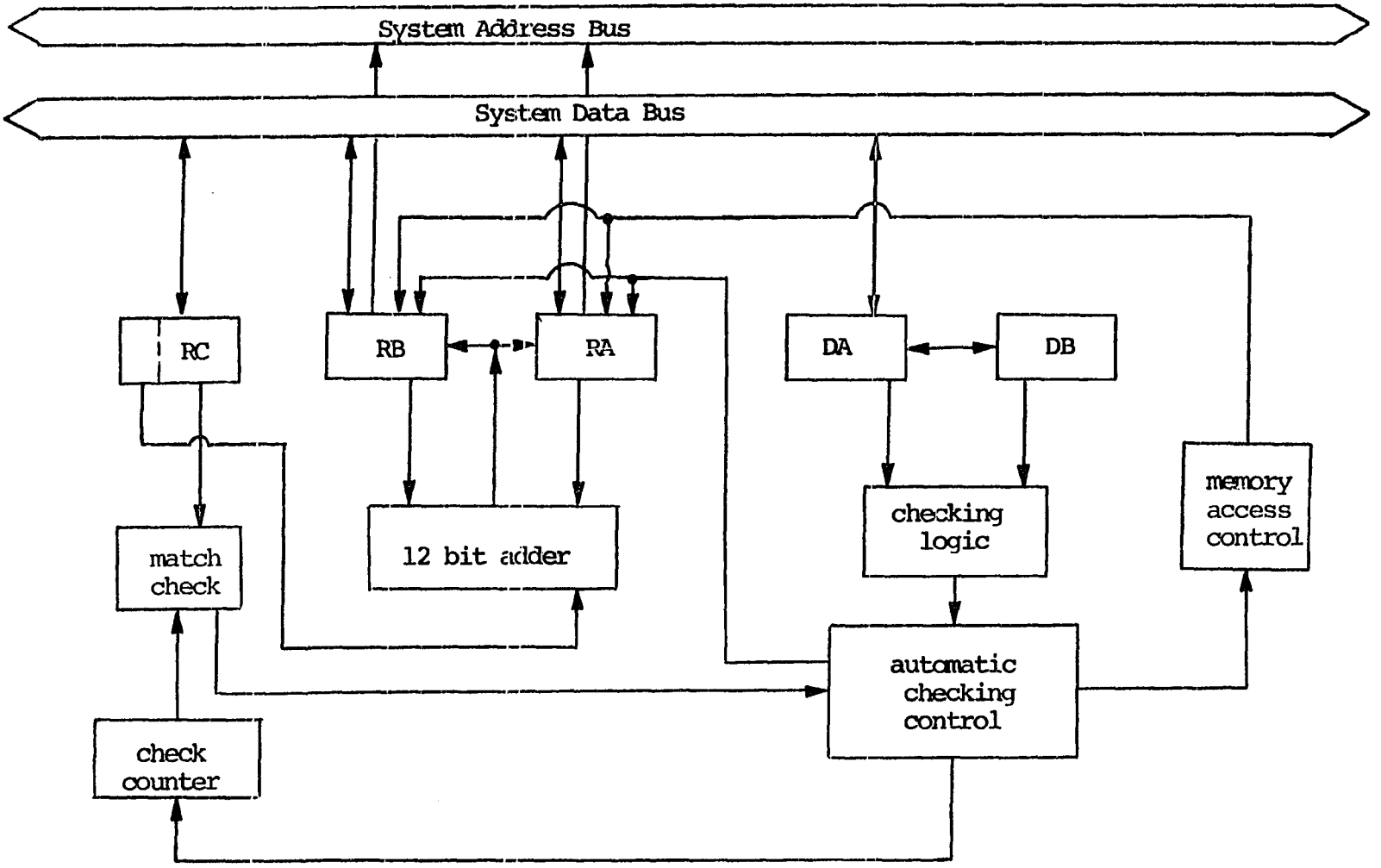


Figure 23. Logic for automatic table checking.

2) two 8 bit address registers

3) sequencing control including memory request control

4) special OP CODE generator

5) one 8 bit Output Sequencer control register

This Output Sequencer is enabled by the Table Matcher when a message transmission is needed by storing a proper information in the Output Sequencer control register. This register is loaded by the OSC instruction and depending upon the loaded information in this register, the Output Sequencer transmits proper message. The block diagram of the Output Sequencer is shown in Figure 24. This output Sequencer is physically implemented on a 14" x 11" printed circuits board on which following hardware componets are also provided.

1) memory request first come, first serve network

2) parity bit generator

3) parity checker

4) memory control clock generator and timing phase decoding logic

The block diagram of the random access memory and memory controller is shown in Figure 25.

Micro-processor

The Intel 8080 which is a 40 pin single chip LSI general purpose computer with 2 microseconds instruction cycle time is used for the Micro-processor. The Micro-processor is used

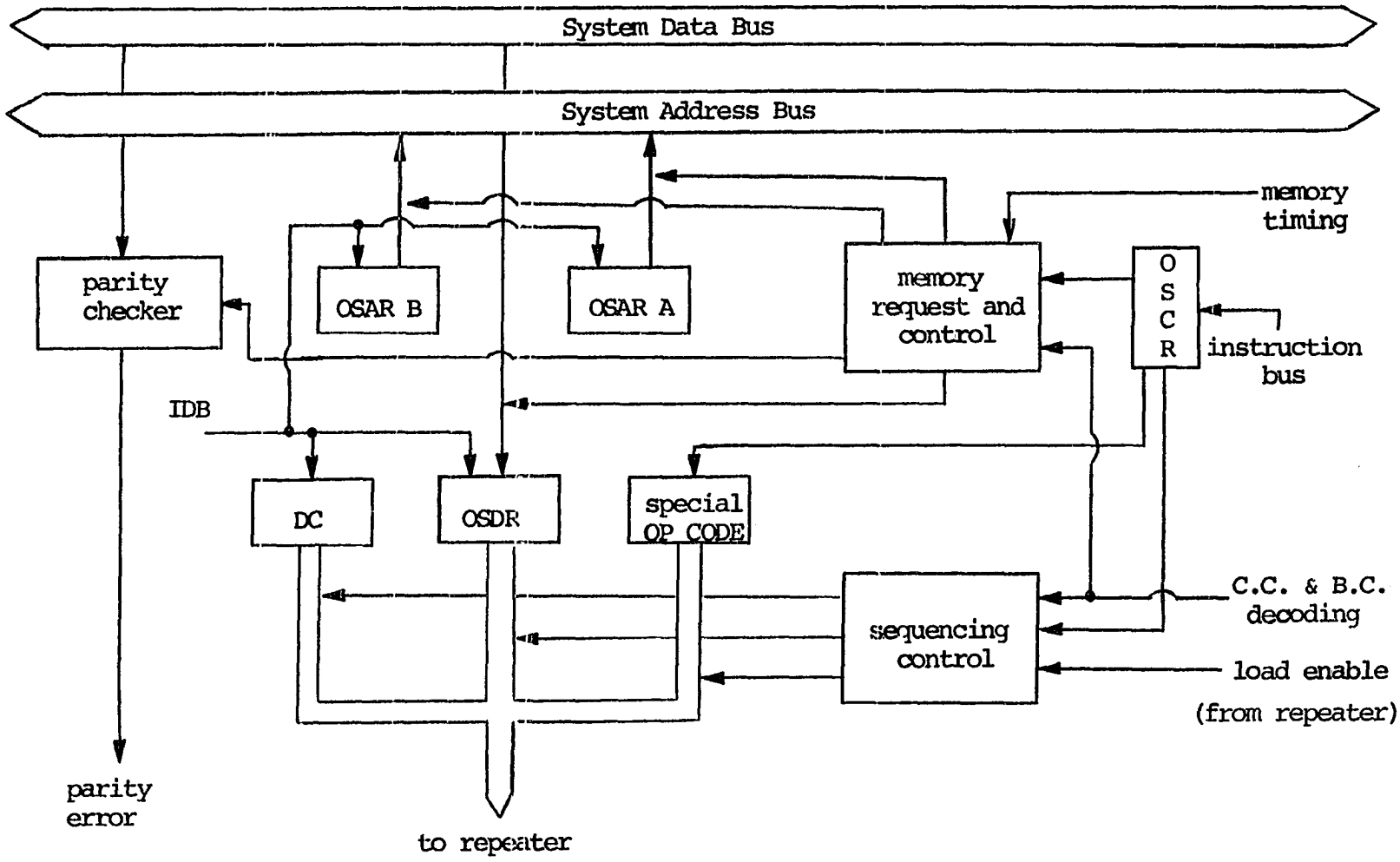


Figure 24. Output Sequencer block diagram.

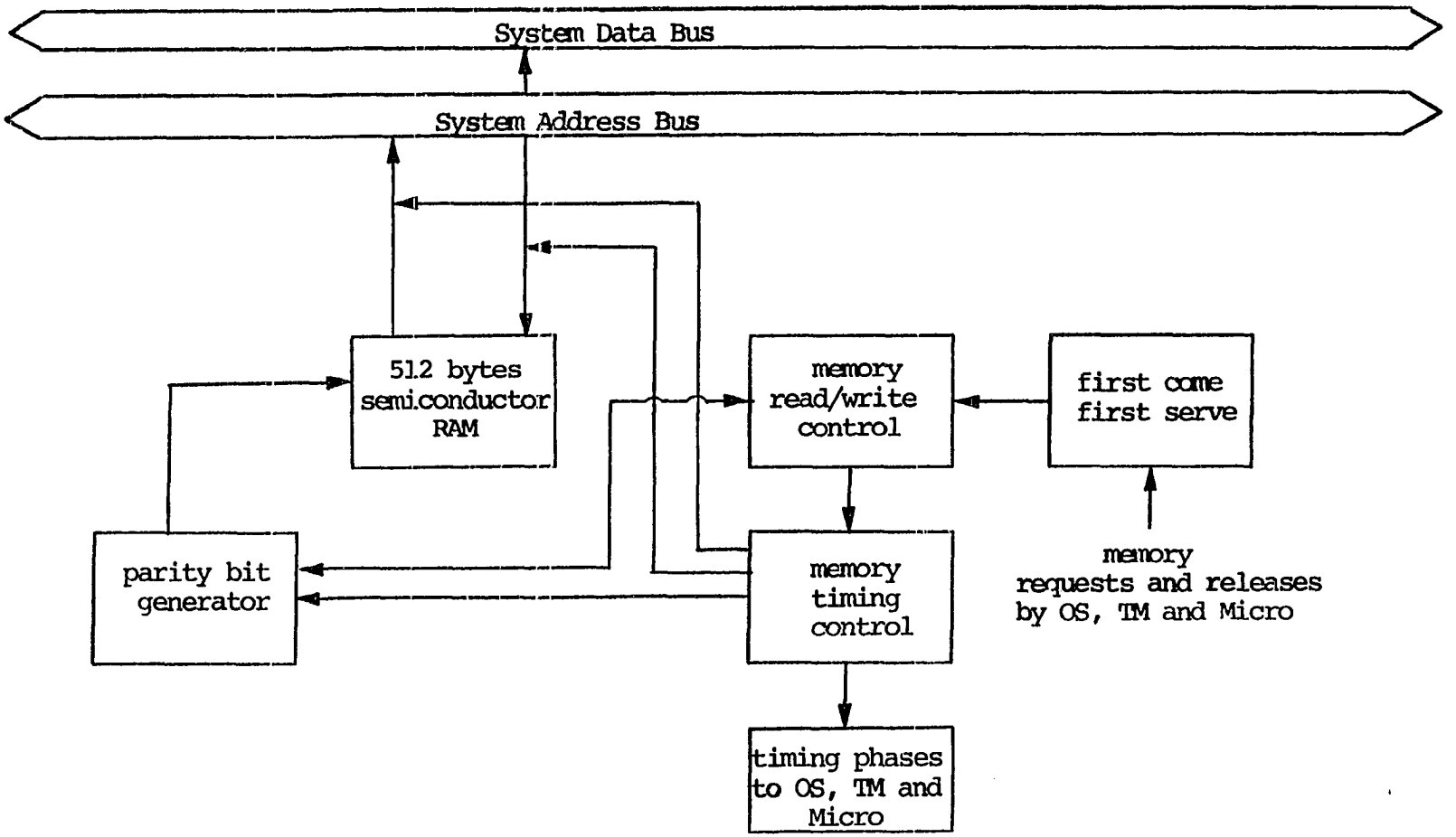


Figure 25. RAM and controller.

for controlling interrupts provided by the Table Matcher, data transfers to and from the devices connected to the interface and message editing and some other control functions. The message editing includes breaking up of a message into 10 bytes sections that can be put into a slot as well as the reassembly of these sections at the destination.

The Table Matcher can interrupt the associated Micro-processor with a proper interrupt code which is to be stored in the Interrupt Information Table of the 512 byte random access memory. Two reserved locations in the Interrupt Information Table are used for storing the interrupt codes. There are three different interrupt instructions each of which has its own usage and priority. The interrupt controller connected to the Micro-processor is designed to provide the multiple interrupt capability to the Micro-processor according to the assigned priorities. Three different interrupt instructions which are used by the Table Matcher to interrupt the Micro-processor are shown in Table 5 as well as the interrupt codes, reserved locations for interrupt codes, priorities and the related usage.

When the Micro-processor is interrupted by instruction, INT.A, the Micro-processor will start either the start-up procedure or the shut-down procedure depending upon the code stored in the code location 001010110. The instruction, INT.B, is for interrupting the Micro-processor for control-

Table 5. Interrupt instructions

Instruction	Priority	Location	Code	Usage
INT A	3	001010110	00000110	shut-down
INT A	3	001010110	00000111	start-up
INT B	2	001001110	xxxxSBNO	source buffer
INT B	2	001001110	DBNOxxxx	dest. buffer
INT C	1	001010110	00000001	TMMC ok
INT C	1	001010110	00000011	TMMC fail

ling normal message handling problems, primarily for controlling buffers. Upon receiving this interrupt, the Microprocessor needs to make various decisions depending upon the information stored in the buffer indicated by the interrupt code in the memory location 001001110.

The interrupt code which is used for this interrupt given by the instruction, INT.B, indicates the buffer number(s) on which the Micro-processor must serve the interrupt. The high order four bits in the code indicates one of the destination buffers and the low order four bits indicate one of the source buffers, each of which can be a binary value between 0000 to 1110. This indicates that up to 15 source buffers and 15 destination buffers can be allowed in an interface and the binary 1111 in either location indicates an

invalid buffer number.

For example, when the Micro-processor is interrupted by the Table Matcher with interrupt instruction, INT.B, and interrupt code 11110010 in memory address 001001110, the Micro-processor must examine the contents of the third source buffer to determine proper operation to serve this interrupt. Since the high order four bits in the interrupt code, 1111, indicates an invalid destination buffer, the Micro-processor need not to consider about the destination buffers at all.

The instruction, INT.C, will be used to inform the results of a table checking when the Table Matcher has been used on Micro-processor's table checking operations.

This requires that the Micro-processor must serve an interrupt within a time slot delay to be ready for the interrupt which may be given to it while the interface is processing the next time slot. Considering the speed of the Micro-processor and the speed requirement of the interfacing problem, the Micro-processor will save the given interrupt code in the core memory instead of completing the whole operations required by the interrupt. Since the buffer status are properly updated to block the buffers from additional messages by the Table Matcher before the interrupt, the operations which must be done to serve the interrupt can be done any time when the Micro-processor is available.

A sample program which saves the interrupt code has been written to see the amount of time required for this routine following the Micro-processor's instruction set as shown in Table 6. This routine is supposedly stored in the semiconductor memory instead of in core memory to reduce the time required for instruction fetches. Total amount of time required for handling this routine was obtained as follows: There are total 137 states each of which is 500 ns, which amounts to 69.5 microseconds. There are 9 writing operations into the core memory each of which is 1.5 microseconds which totally amounts to 13.5 microseconds. Also, there are 9 read operations from the core memory each of which is 0.5 microseconds, which totally amounts to 4.5 microseconds. Summing up these values, total 86.5 microseconds is obtained including memory cycle times which is faster than a time slot delay 110 microseconds. However, practically, the Table Matcher can give the next interrupt at STEP 11 of the Table 6. Therefore, the minimum interval of 68.5 microseconds between two consecutive interrupts can be obtained.

Table 6. An interrupt code saving routine

STEP	Instruction	byte	state	fetch	memory	RAM	CORE
1	RST	1	11	-	RAM	-	2W
2	PUSHA	1	11	1	RAM	-	2W
3	PUSHH	1	11	1	RAM	-	2W
4	LHLD INTCODE	3	17	3	RAM	-	2R
5	LAM	1	7	1	RAM	1W	-
6	LHLD INTLIST	3	17	3	RAM	-	2R
7	LMA	1	7	1	RAM	-	1W
8	INXH	1	5	1	RAM	-	-
9	SHLD INTLIST	3	17	3	RAM	-	2W
10	EI	1	4	1	RAM	-	-
*11	POPH	1	10	1	RAM	-	2R
12	POPA	1	10	1	RAM	-	2R
13	RET	1	10	1	RAM	-	2R

As mentioned earlier, the Micro-processor is also used for controlling data transfers to and from the devices which are connected to the interface. Four different kinds of peripheral devices can be directly attached to the interface, through the Micro-processor, without the aid of any other processors. These are card readers, teletypes, line printers and CRT displays. The Micro-processor serves these peripheral devices on a character by character interrupt basis. Considering the speed requirements and operational characteristics of these peripheral devices, overall priorities are assigned to the interrupts given by these devices as shown in Table 7.

Table 7. Interrupt priorities

Priority	Type of INTERRUPT
6	shut-down, start-up procedure
5	Table Matcher normal INT
4	card reader
3	CRT display
2	teletype (TTY)
1	line printer

A sample program which moves a character between the core memory and a peripheral device, similar to the interrupt code saving routine, has been written as shown in Table 8. This routine takes 91 microseconds for execution.

Table 8. A character transfer routine

STEP	Instruction	byte	state	fetch	memory	CORE
1	RST	1	11	-	-	2W
2	EI	1	4	1	RAM	-
3	PUSHA	1	11	1	RAM	2W
4	PUSHH	1	11	1	RAM	2W
5	JMP CON	3	10	3	RAM	-
6	CON LHLD BFAD	3	17	3	CORE	3R
7	IN DEVICE	2	10	2	CORE	-
8	LMA	1	7	1	CORE	1W
9	INXH	1	5	1	CORE	-
10	SHLD BFAD	3	17	3	CORE	2W
11	POPH	1	10	1	CORE	2R
12	POPA	1	10	1	CORE	2R
13	RET	1	10	1	CORE	2R

Assuming typical speeds of the peripheral devices, the percentage usage of the Micro-processor for controlling data transfer problems can be obtained as follows: Assuming that there are 16 stations and 32 time slots on a loop and also assume that every time slot has equal probability to be used by a station, then on the average four interrupts (two for source buffers and the other two for destination buffers) will be sent to the Micro-processor by the Table Matcher for a loop delay which is 3.3 ms. If eight different peripheral devices such as one card reader, one CRT display, one line printer and four teletypes are provided and all of these devices are busy for handling data transfers, then the percentage usage of the Micro-processor for transferring data to and from these devices is obtained as follows:

Table Matcher	87 μs x 4/3.2 ms	=0.109
Card Reader	91 μs/2.5 ms	=0.036
Teletype	(200 μs/100 ms) x 5	=0.01
Line Printer	150 μs/2 ms	=0.075
CRT display	150 μs/1.67 ms	=0.09
		Total=0.32

Therefore, the Micro-processor can still spend 68% of the time for its own jobs which are not time critical such as updating buffer status, transferring messages between the random access semiconductor memory and the core memory,

updating tables and editing messages, etc.

Consequently, the Micro-processor is expected to be capable of handling any combination of these peripheral devices without the aids of any other processors. Clearly, for a station where a minicomputer is connected to the interface, these Micro-processor's control functions will be even more simplified.

The block diagram of the Micro-processor and the control logics associated with it are shown in Figure 26. Physically, 80 TTL IC's are used for implementing this Micro-processor on a 14" x 11" printed circuits board.

Three different buses are provided to connect the Output Sequencer, Table Matcher, Micro-processor and the different memories. The System Data Bus is a 9 bit data bus including one parity bit which the Output Sequencer, Table Matcher, and Micro-processor can use, one at a time to access the 512 bytes semiconductor memory. The System Address Bus is a 12 bit address bus to send memory addresses to the memory by the processor which gets the memory service granted. The high order two bits of this address bus will be used as a memory cycle control code to specify the memory read or write request. The Internal Data Bus is an 8 bit data bus to connect all the registers inside Table Matcher and Output Sequencer. Through this route, all data transfer operations will be performed inside the Table Matcher and between the

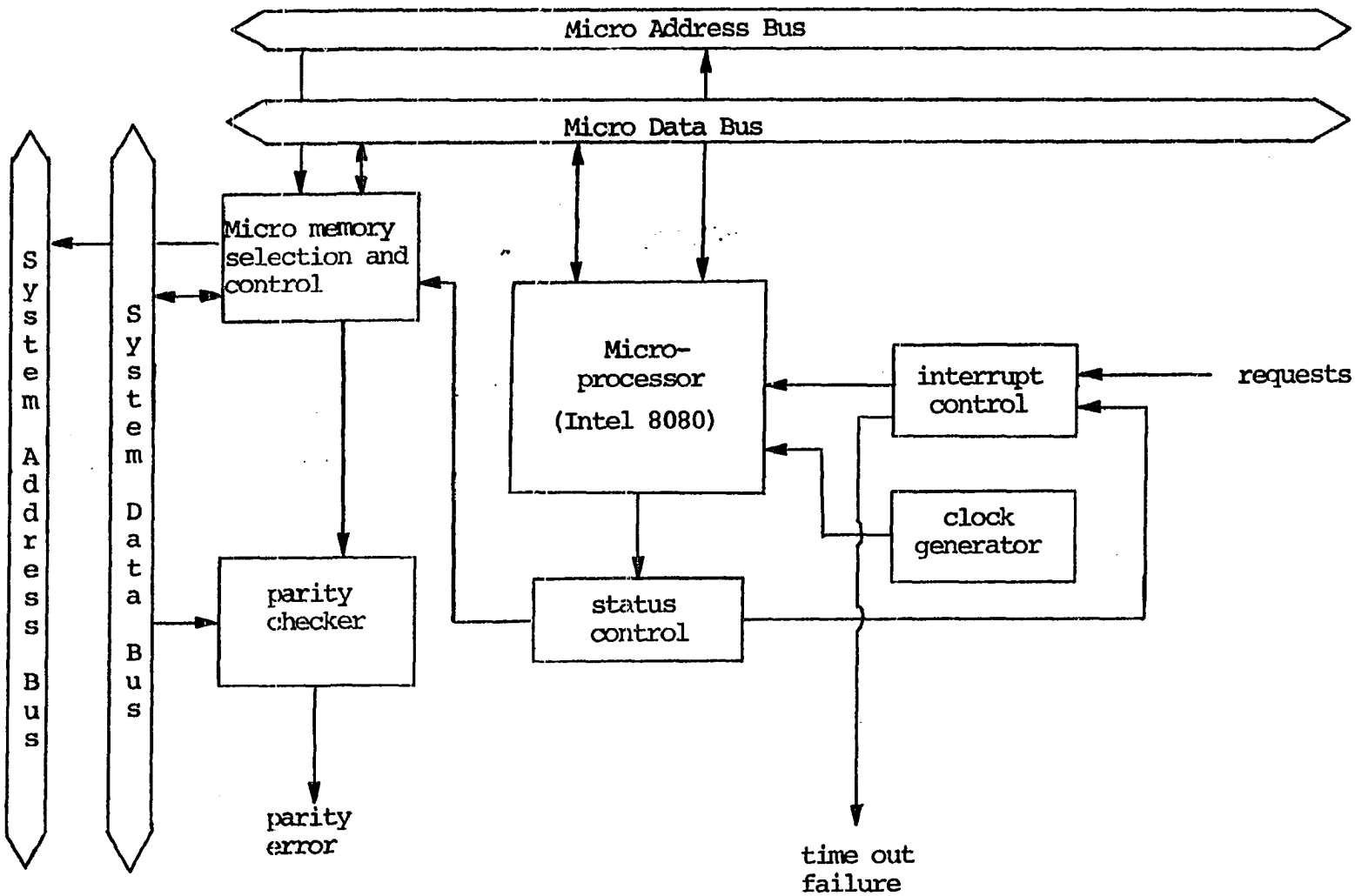


Figure 26. Micro-processor and controller.

Table Matcher and Output Sequencer.

The Micro-processor has its own 16 bit address bus and 8 bit data bus to connect the Micro-processor to the core memory and also to the 512 byte semiconductor memory when the Micro-processor wants to access the commonly owned memory. In other words, the system address bus of the interface is connected to the address bus of the Micro-processor and the System Data Bus of the interface is connected to the data bus of the Micro-processor, to allow the Micro-processor access the semi-conductor memory. The bus connections among these processors and the memories are shown in Figure 27.

System Maintenance

There are some features involved in the interface which will be used for system debug and maintenance. During the normal operations of the interface, five major probable system malfunctions can be detected and displayed on the front pannel of the station. These are:

a) Parity check of the semiconductor memory: Whenever a data is stored in the semiconductor memory, an even parity bit is generated through a 8 bit parity generator and stored in the memory with the data. Therefore whenever a data is read, the retrieved 9 bit data would be checked on a 9 bit parity checker.

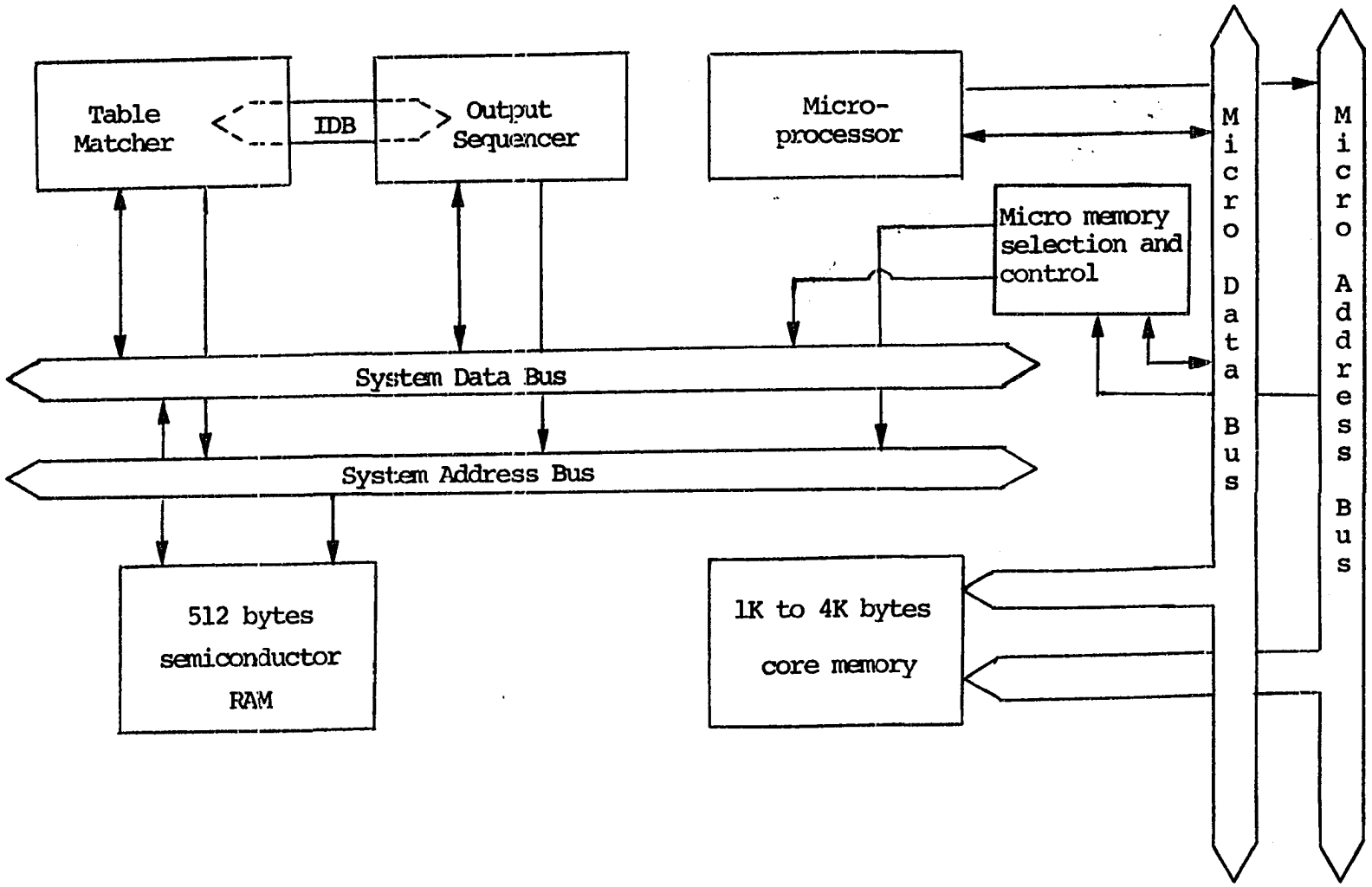


Figure 27. Overall system connections.

B) Parity check of core memory: The same thing is done for the core memory as for the semiconductor memory.

C) Parity check on the system data bus: The system data bus is the main data path through which communication between a processor and the semiconductor memory can be made. The Table Matcher, Output Sequencer and the Micro-processor are physically separated and implemented on different printed circuit boards and each processor has its own 9 bit even parity checker which will be used for checking the 9 bit (including one parity bit) system data bus.

D) Mutually exclusive operation such as instruction decoding: Only one instruction must be decoded from the 12 bit instruction bus. This can be checked, partially, by applying parity checkers.

E) Time out failure: When the Table Matcher interrupts the Micro-processor, the Micro-processor must acknowledge the interrupt within a certain amount of time by sending out the interrupt acknowledge signal, because the Table Matcher's interrupt has the highest priority.

A maintenance unit which is composed of a display board, a simulated repeater and four connector boards have been constructed. The display board has 300 Light Emitting Diodes (LED) on it to display the contents of registers, buses and some other major control signals to test the interface on a static basis. The simulated repeater is a logically

simplified repeater which can be operated on a manual mode for testing the interface. The connector boards are used for connecting the various signals to the display board.

The interface has two different operation modes. One is the normal mode and the other is the test mode. These are controlled by a switch on the front panel of the station. When the interface is in the normal mode, it operates in conjunction with the attached line repeater. Otherwise, the interface is logically disconnected from the line repeater and is operated in conjunction with the simulated repeater. Therefore, the interface can be tested without disturbing the normal operations of the whole net. Also, an extension board has been constructed to extend the bus connector signals out of the interface system box, which will permit testing and probing a specific board.

CONCLUSIONS

The critical parameters for interfacing a loop connected system have been discussed and the near optimum organization for the interface processor unit has been determined. The feasibility of such a system has been demonstrated on a general and detailed basis.

The critical parameters in the system were found to be the processing rates for the various table matches; this problem was solved by the Table Matcher which is a specially developed processor for high speed sequential table searching operations. Because of different speed requirements relative to interfacing, the system was organized into three main sections which are called the Table Matcher, the Output Sequencer and the Micro-processor respectively, forming a high speed, low cost and flexible multi-processor system.

Partitioning the control memory into two separate parts depending upon the pre-defined procedures and subroutines of the interfacing problems appears to be a very efficient technique not only for the discussed system but also for any other systems for handling the problems to be partitioned into procedures and subroutines. Clearly, the number of partitions may be expanded depending upon the practical environment of a system requirements.

Special emphasis has been made on automatic checking operations, partitioning control memory into two separate parts and the application of a Micro-processor as well as table manipulations and system maintenance. To provide flexibility in allocating resources and in sharing the network facilities, the interface was designed as general as possible. Control information related with sharing resources and network itself can be easily modified dynamically depending upon the dynamic increase or decrease of the communication demand of a station. The restricted demand multiplexing technique adopted in this system allows the network itself to be dynamically distributed over its resources. The local communication capability allows the complete distribution of resources and facilities over the network can be physically implemented. The features involved in the interface insure that the developed general purpose interface is suitable as a low cost, high speed and flexible communication processor for handling the fairly sophisticated interfacing problems on a loop computer network.

In the future, if a Micro-processor with an order of magnitude increase in speed over the currently available micro-processors becomes commercially available, then the interface may be implemented with a set of the interconnected Micro-processors instead of using the specially designed processors. The number of required micro-processors in the

set is greatly dependent on the boundary between generality and specificity of the instruction set of the future micro-processor. Economic feasibility can be determined considering the required number of Micro-processors and the improved reliability of the emulated system.

BIBLIOGRAPHY

1. Pierce, J. R. "Network for Block Switching of Data." Bell System Technical Journal 51, No. 6 (July-August, 1972): 1133-1145.

2. Kropfl, W. J. "An Experimental Data Block Switching System." Bell System Technical Journal 51, No. 6 (July-August, 1972): 1147-1165.

3. Coker, C. H. "An Experimental Interconnection of Computers Through a Loop Transmission System." Bell System Technical Journal 51, No. 6 (July-August, 1972): 1167-1175.

4. Farber, D. J.; Feldman, J.; Heinrich, F. R.; Hopwood, M. D.; Larson, K. C.; Loomis, D. C.; and Rowe, L. A. "The Distributed Computing System." 7th Annual IEEE Computer Society International Conference, Feb. 27,28, March 1, 1973.

5. Farber, D. J. and Larson, K. C. "The Structure of a Distributed Computing System-Software." Symposium on Computer - Communications Networks and Teletraffic, Polytechnic Institute of Brooklyn, April 4-6, 1972.

6. Lawrence, A. R.; Hopwood, M. D.; and Farber, D. J. "Software Method for Achieving Fail-Soft Behavior in the Distributed Computing System." 1973 IEEE Symposium on Computer Software Reliability, April 30, May 1-2, 1973.

7. Farber, D. J. and Larson, K. C. "The System Architecture of the Distributed Computer System - The Communication System." The Polytechnic Institute of Brooklyn Symposium on Computer Networks, April, 1972.

8. Hassing, T. E.; Hampton, R. M.; Bailey, G. W.; and Gardella, R. S. "A Loop Network for General Purpose Data Communications in a Heterogeneous World." 3rd Data Communications Symposium, November 13-15, 1973.

9. Hayes, J. F. "Modeling An Experimental Computer Communication Network." 3rd Data Communications Symposium, November 13-15, 1973.

10. Koenck, S. E. "The Design and Evaluation of A High Speed Recirculating Data Network." Unpublished Master's Thesis, Iowa State University, May, 1974.

ACKNOWLEDGEMENT

The author wishes to thank to Dr. A. V. Pohm for his valuable suggestions and immense encouragement during the pursuit of this research. The author also expresses his appreciation to Dr. T. A. Smay and Dr. C. G. Maple for their patience and continuous guidance. Special thanks are due Chris Reschly for many rewarding discussions. Last of all, the author would like to thank the many people including his parents and wife who have assisted him in the furthering of his education.

This work was supported jointly by the I.S.U. Computation Center and the I.S.U. College of Engineering.

APPENDIX A. FLOWCHARTS OF INTERFACE OPERATIONS

Fifteen different flowcharts provided herein define interface functions depending upon the processing requirements. In the flowcharts, (Z) indicates the exit for checking other processing requirements.

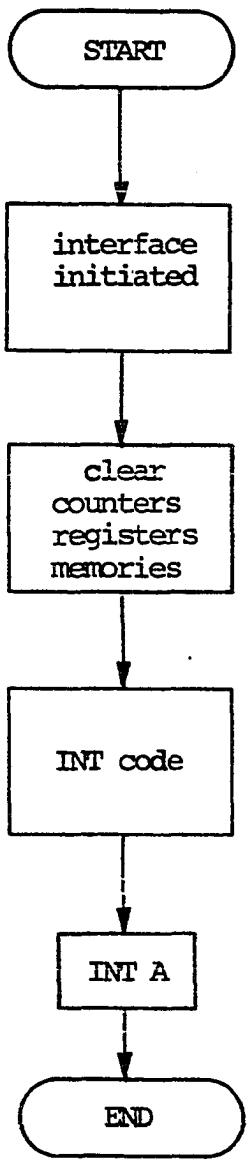


Figure A1. Start-up procedure.

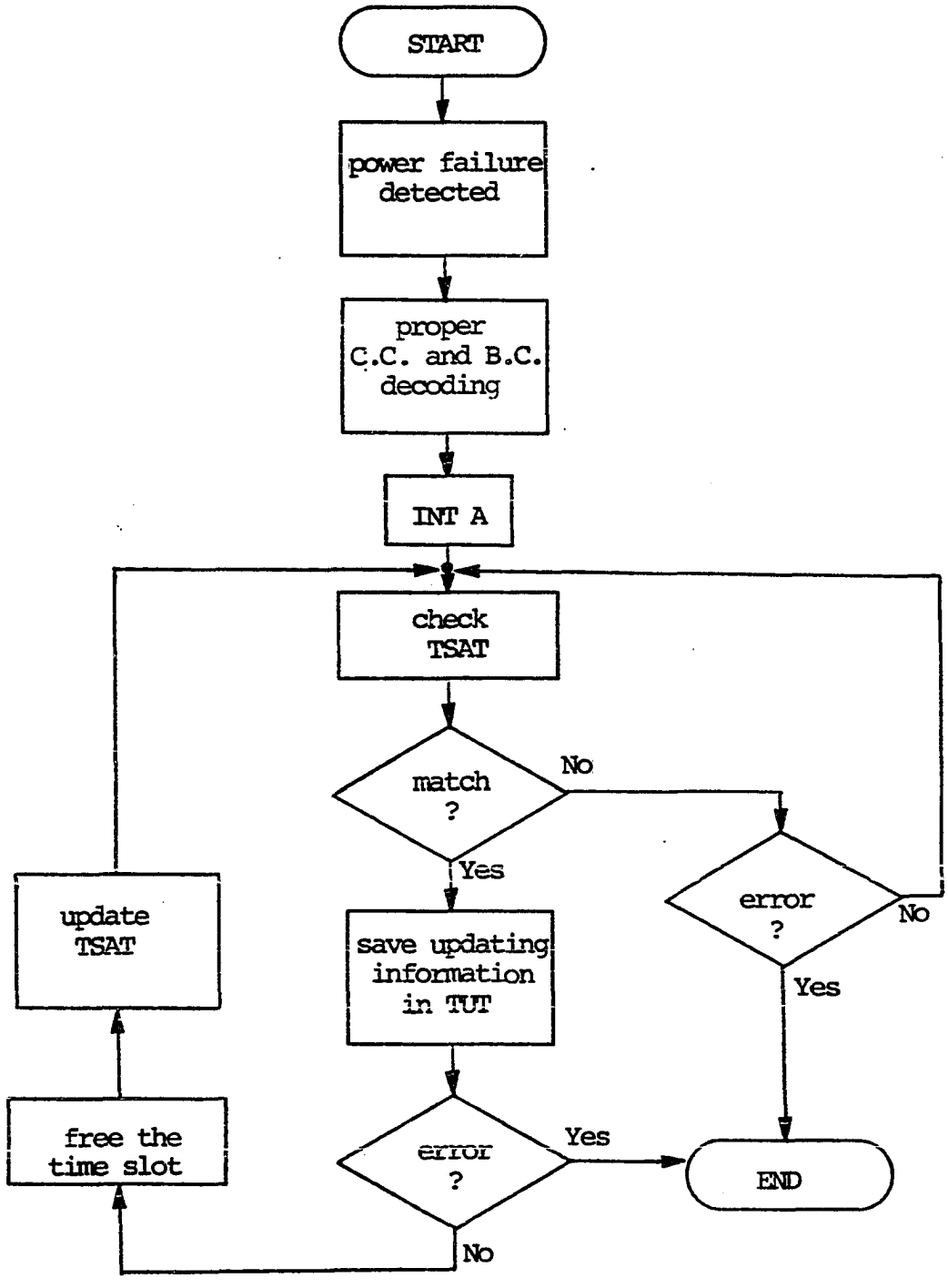


Figure A2. Shut-down procedure.

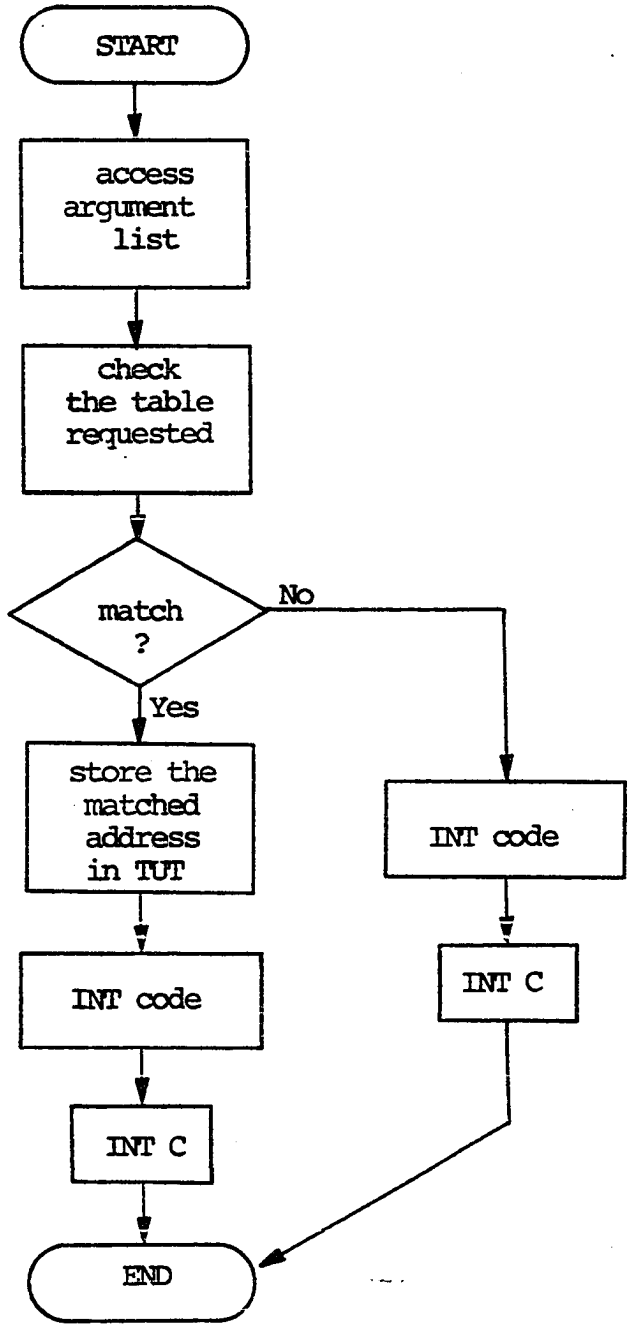


Figure A3. Allow Table Matcher to Micro-processor.

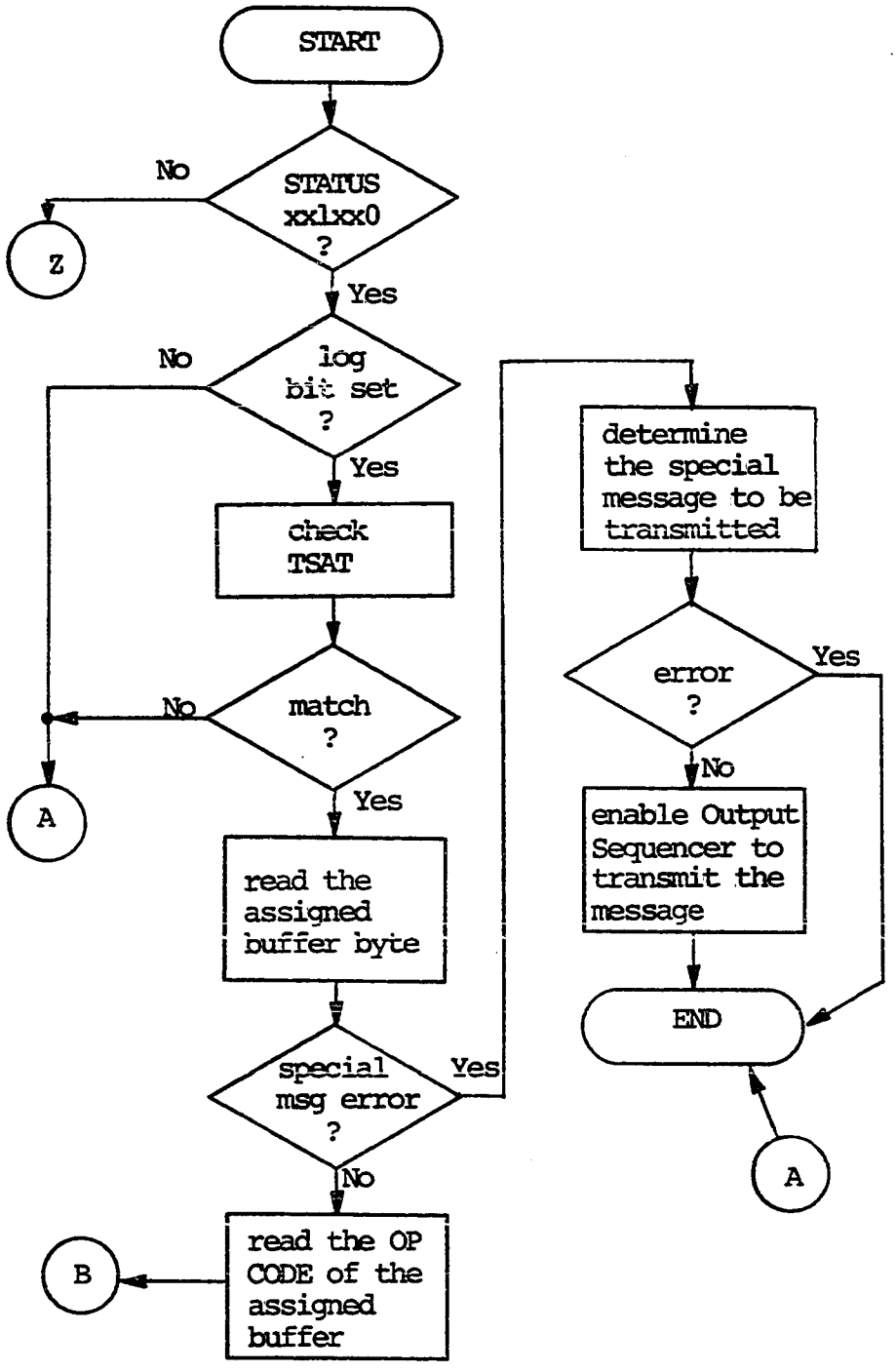


Figure A4. Transmission error procedure.

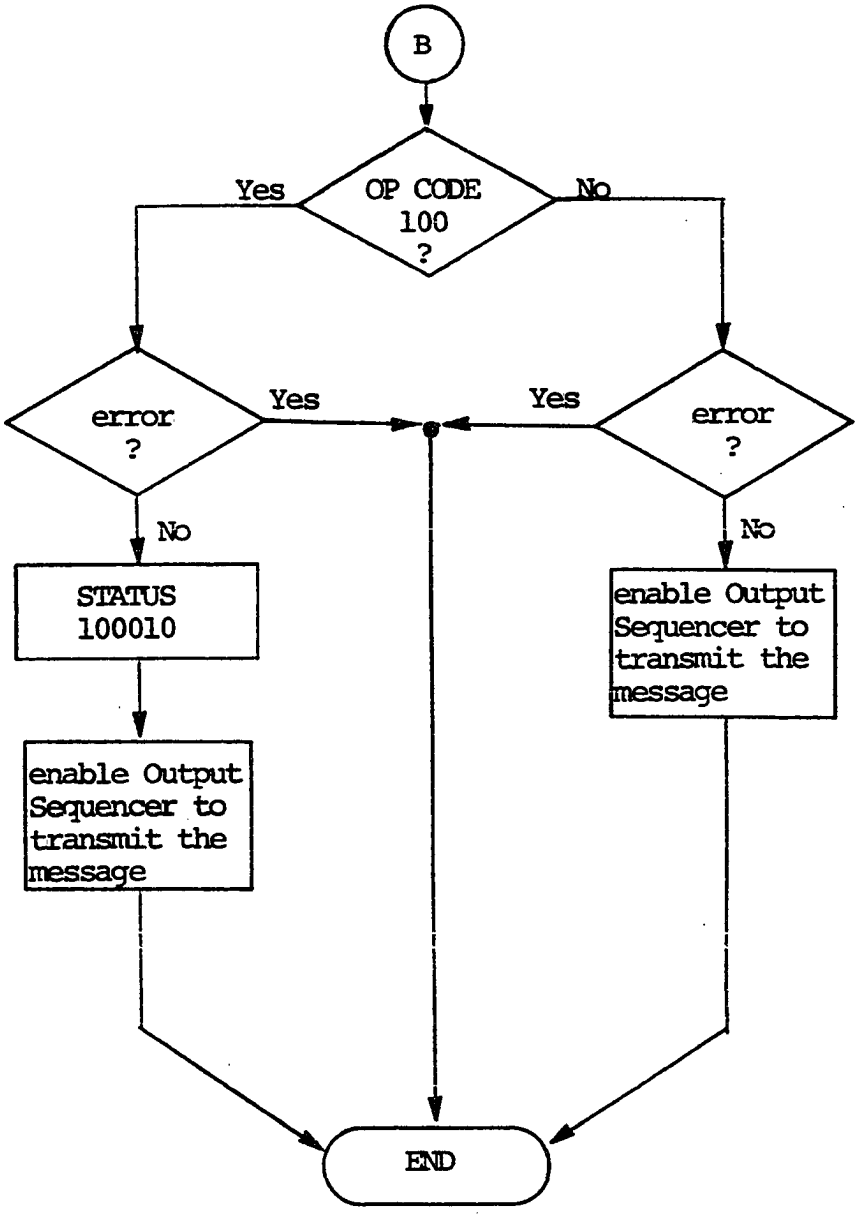


Figure A4. (continued)

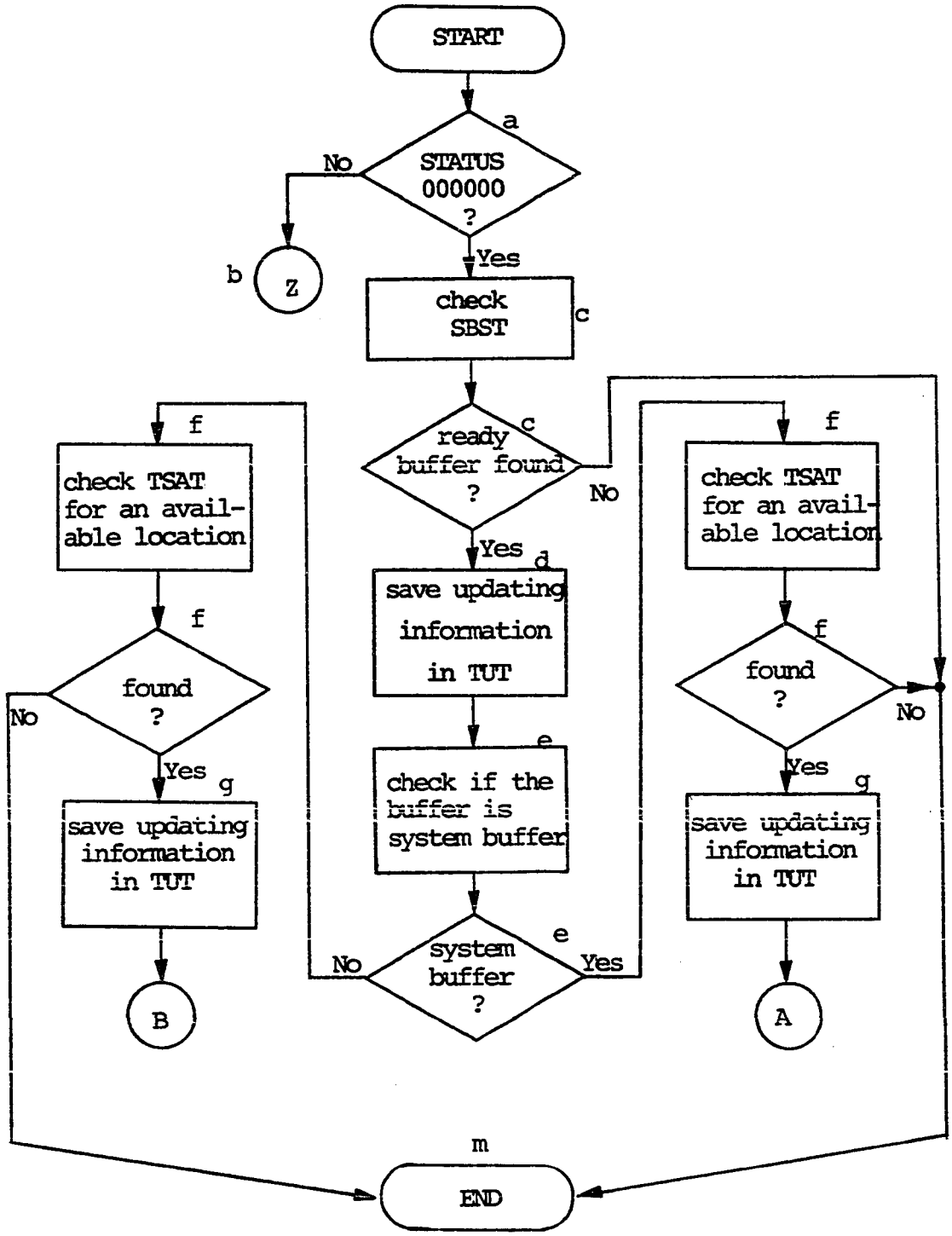


Figure A5. Data transmission procedure.

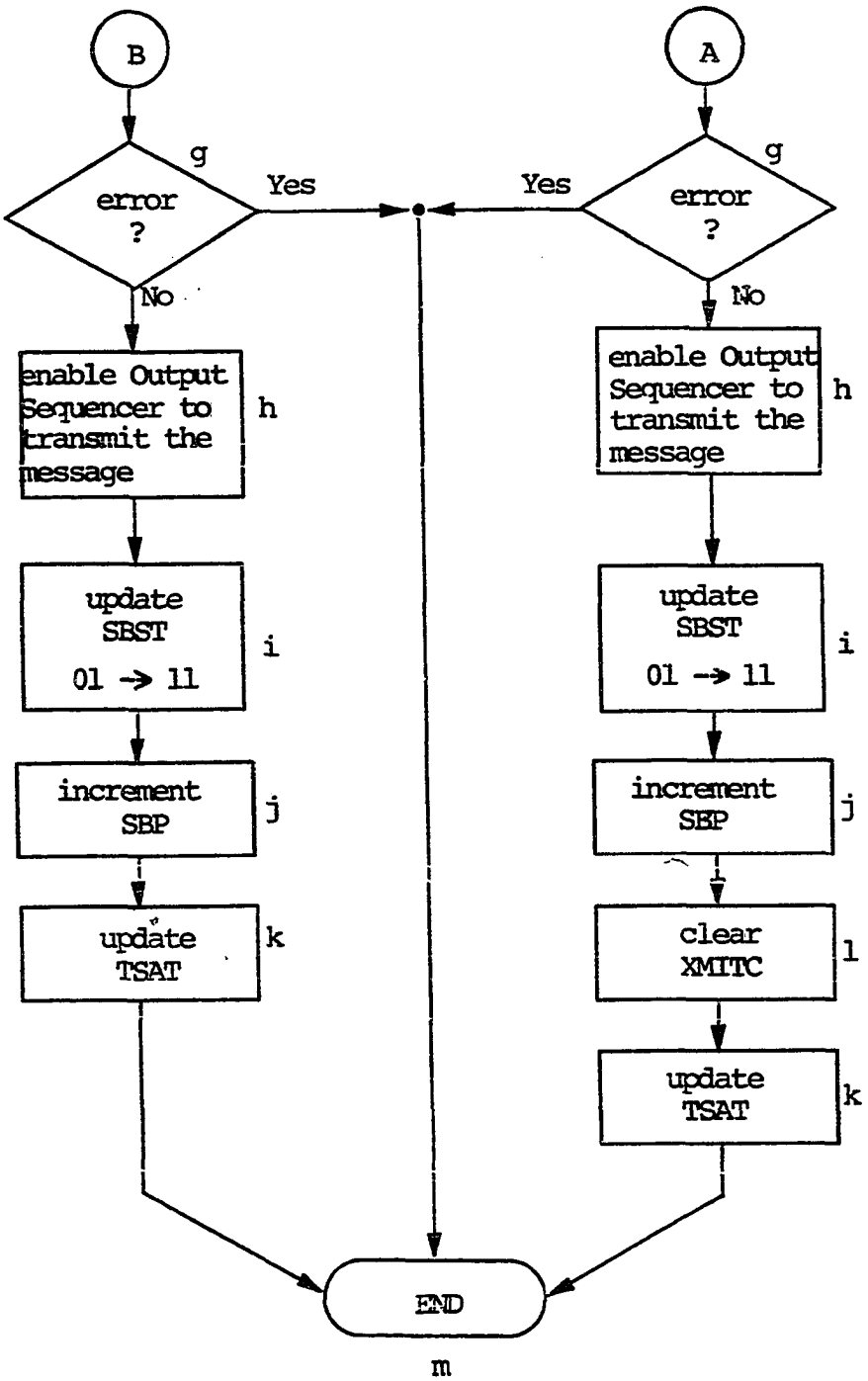


Figure A5. (continued)

Table A1. Data transmission procedure

STEP	Operations
a	Check the STATUS byte to determine if the incoming time slot is empty or not.
b	Determine the necessary procedure according to the STATUS and OP CODE bytes if the STATUS byte is not empty.
c	Check the Source Buffer Status Table for a ready source buffer to be transmitted.
d	Store the buffer status updating information in Table Updating Table.
e	Check if the selected buffer is a system buffer.
f	Check the Time Slot Acknowledge Table for an empty time slot entry.
g	Store the Time Slot Acknowledge Table updating information in Table Updating Table and wait for transmission error checking.
h	Set up the address of the ready source buffer to be transmitted and transfer this buffer address to the address registers of the Output Sequencer and enable the Output Sequencer transmit the data in the buffer. The Output Sequencer will transmit the message in the buffer one byte at a time every 6.5 micro-seconds according to the proper decoding of the character counter and the bit counter.

Table A1. (continued)

STEP	Operations
i	Update the source buffer status from 'ready' to 'transmitted' in the Source Buffer Status Table.
j	Increment the Source Buffer Pointer by one to indicate the next source buffer to be checked.
k	Store the time slot number used by that station and the source buffer transmitted on that time slot into the Time Slot Acknowledge Table.
l	Clear the contents of the transmission counter.
m	The transmission procedure ended. The interface is ready to serve the next time slot.

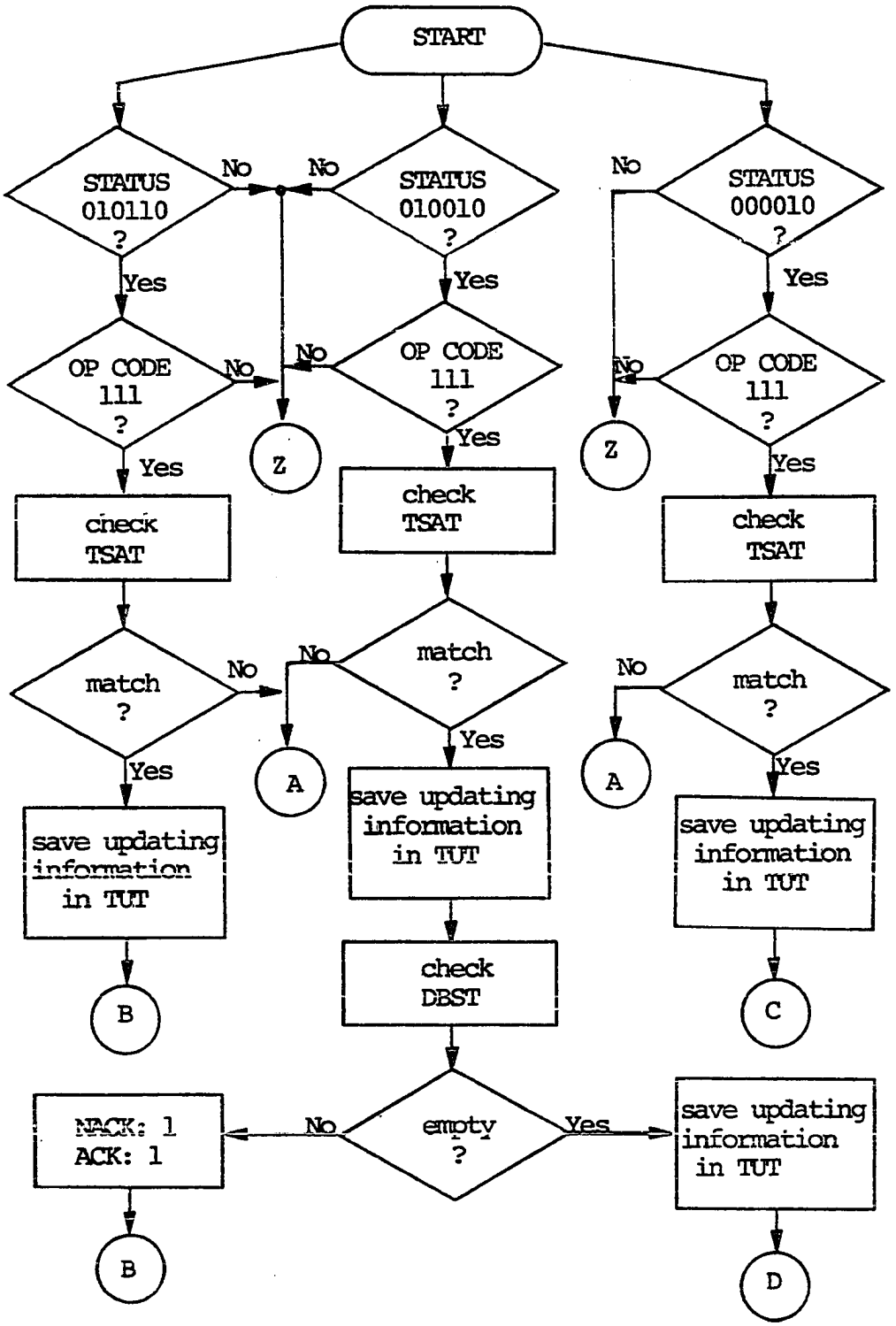


Figure A6. Broadcasting message.

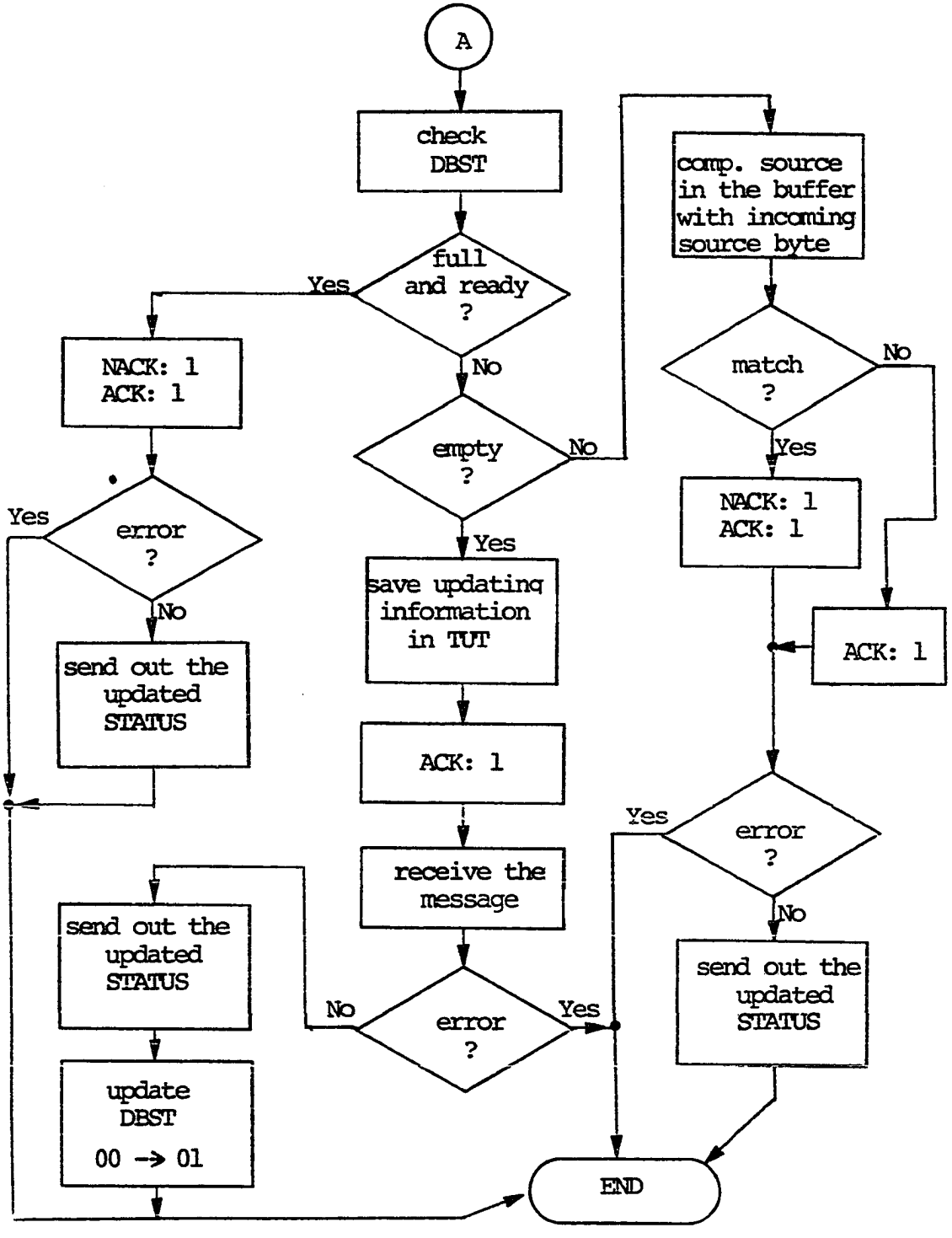


Figure A6. (continued)

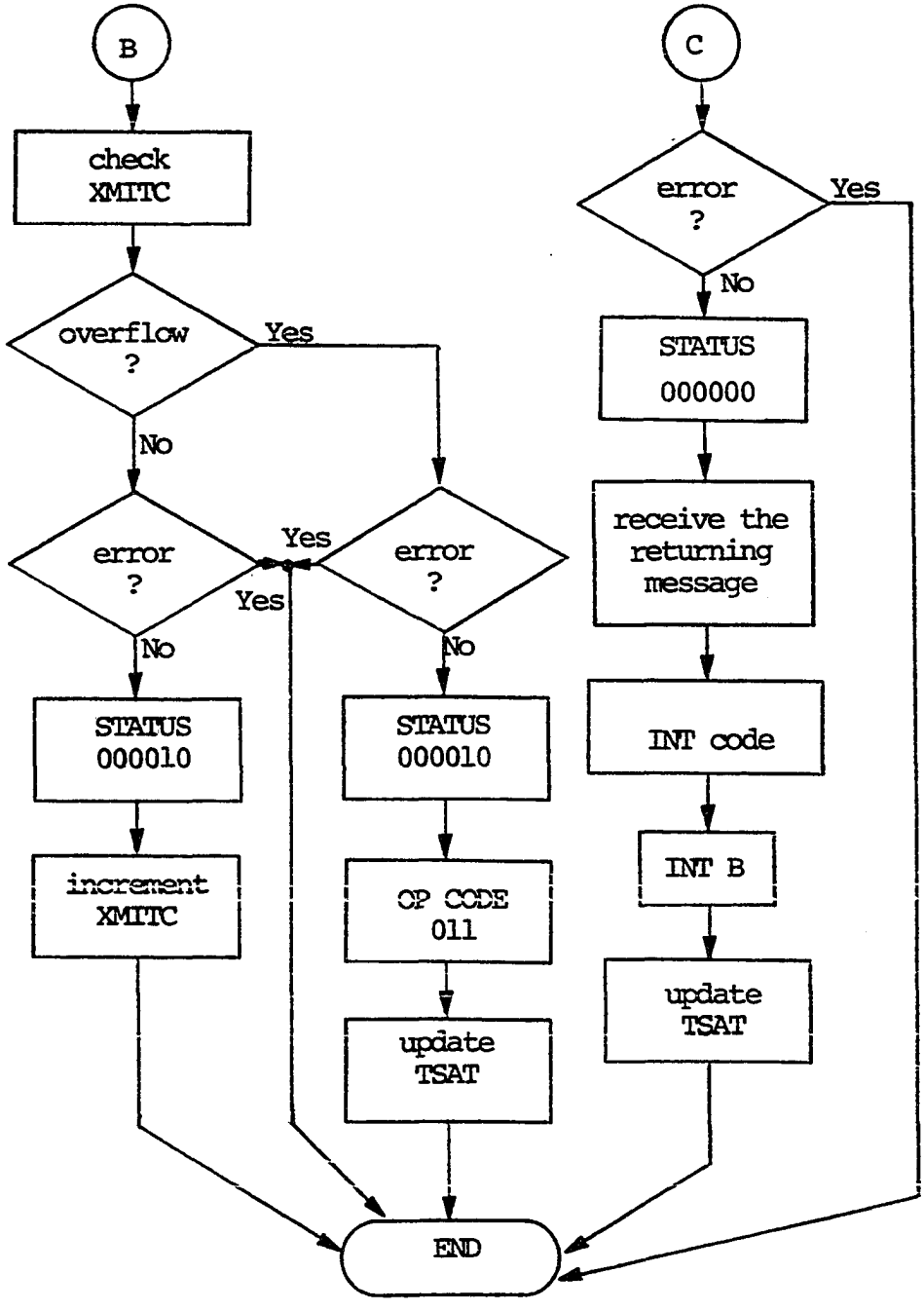


Figure A6. (continued)

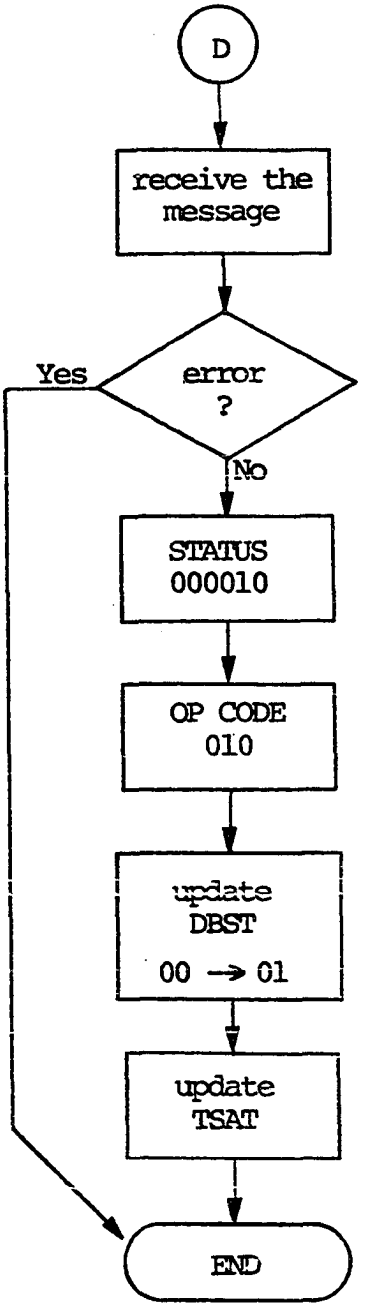


Figure A6. (continued)

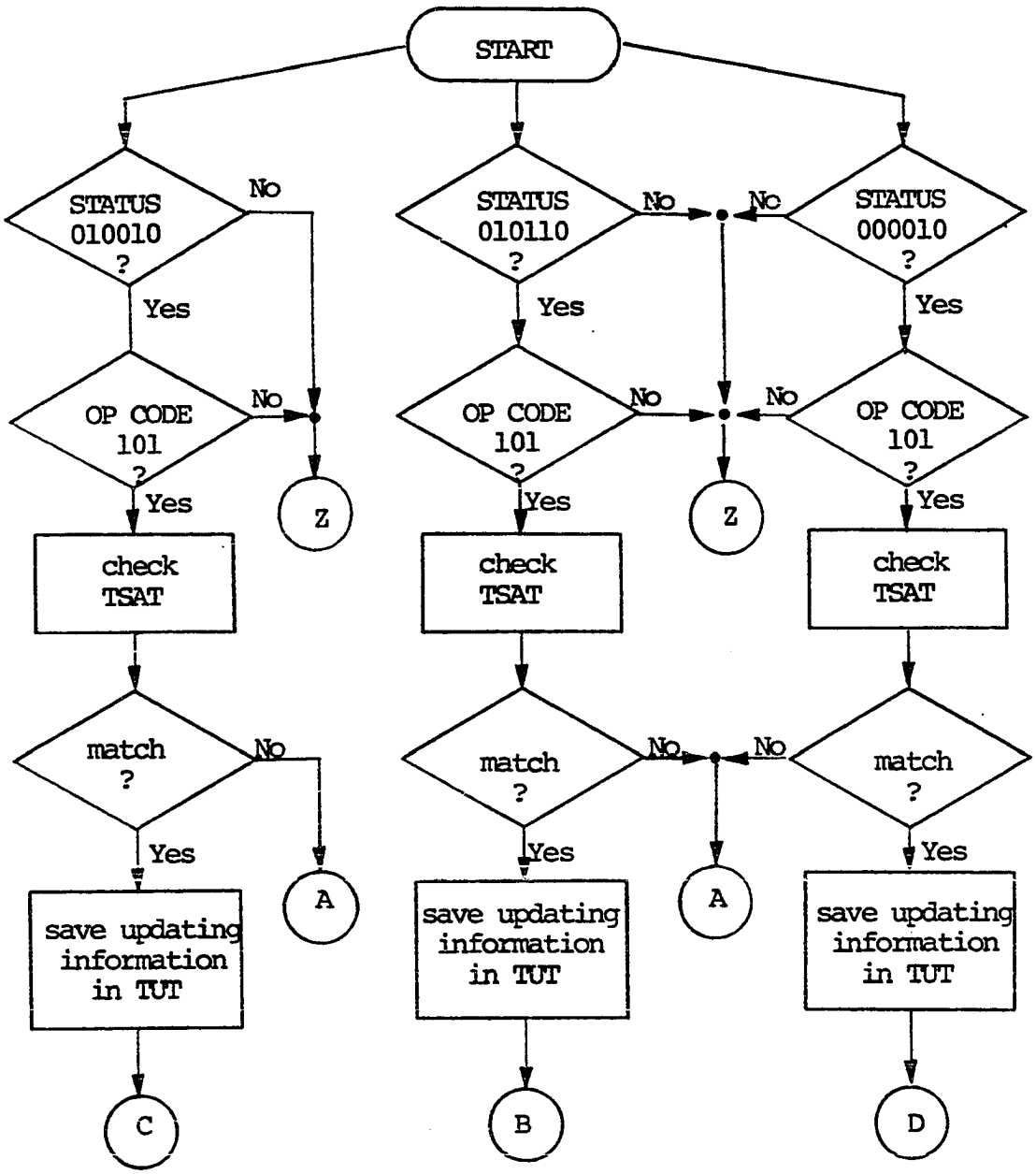


Figure A7. System message to a process.

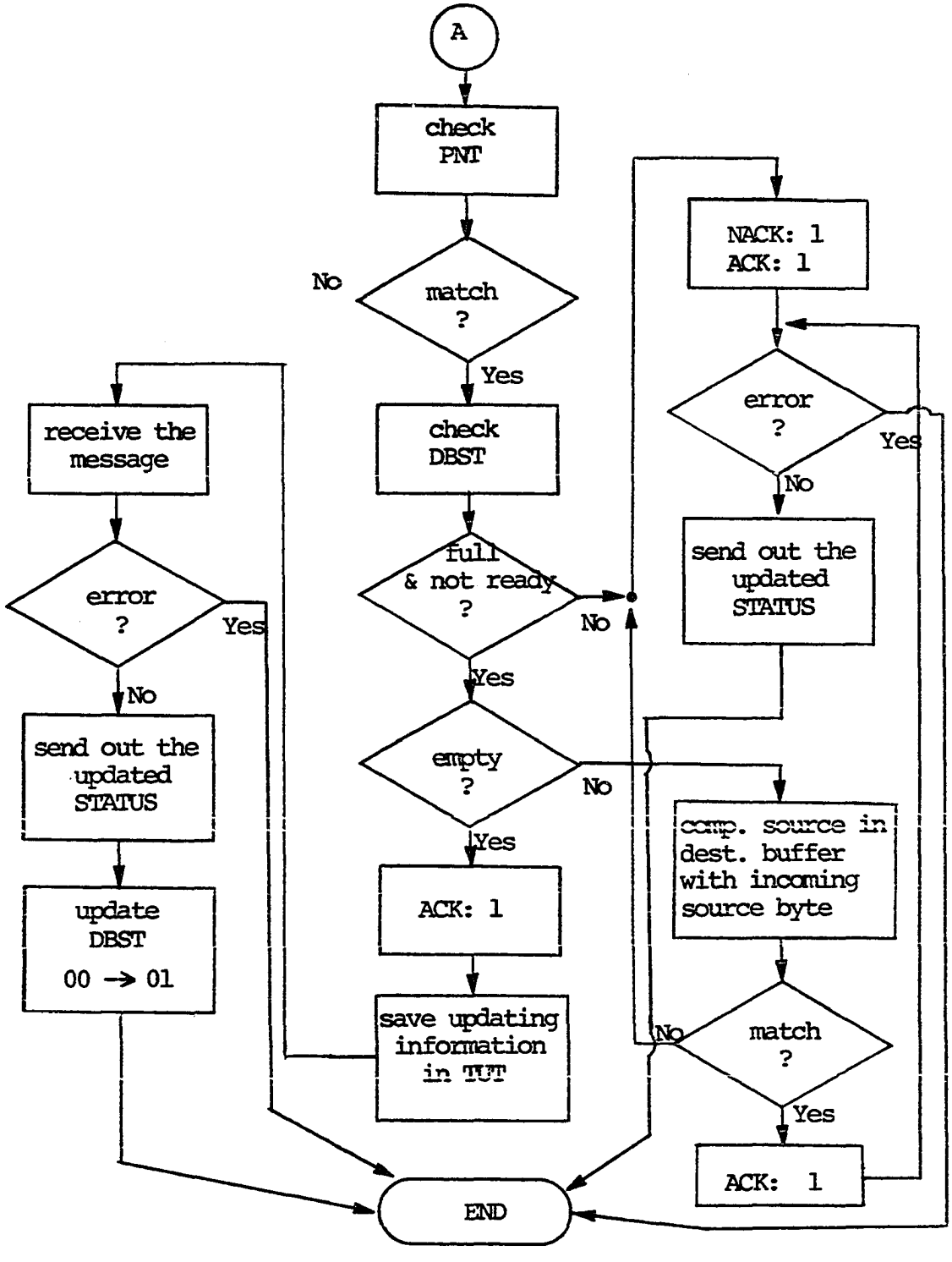


Figure A7. (continued)

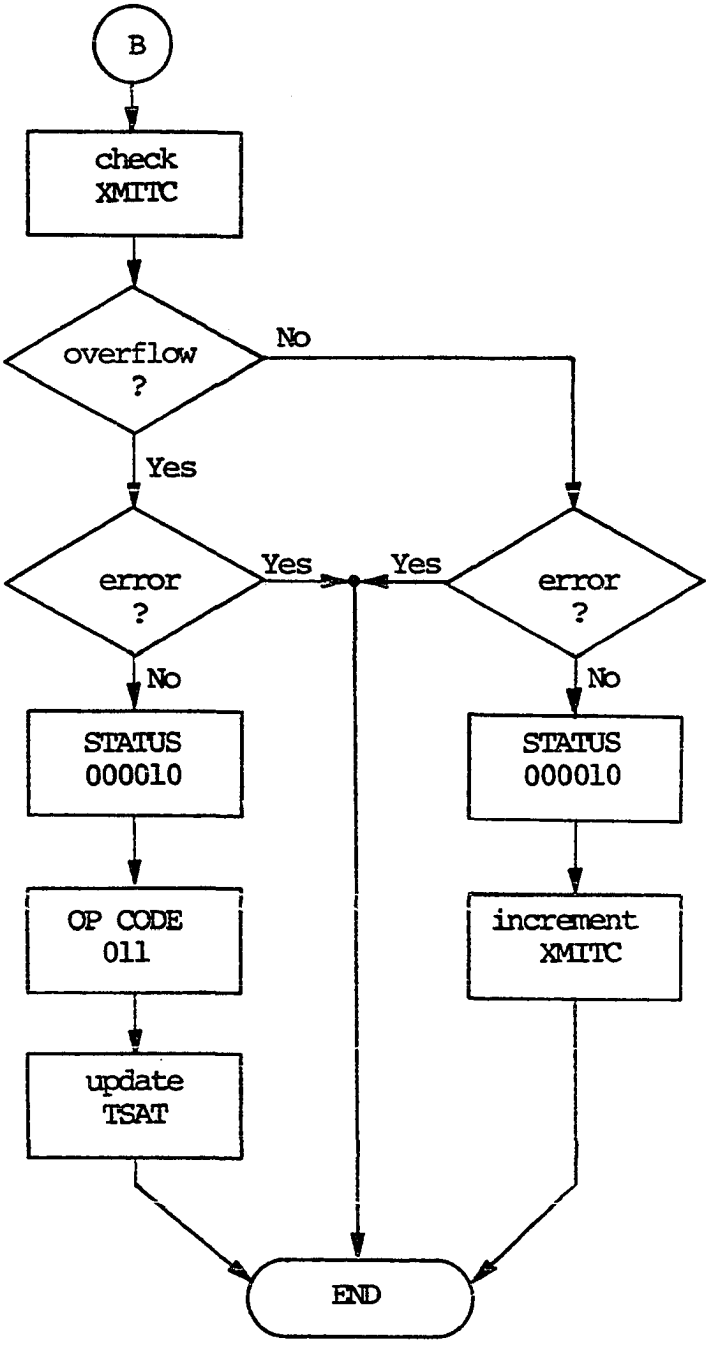


Figure A7. (continued)

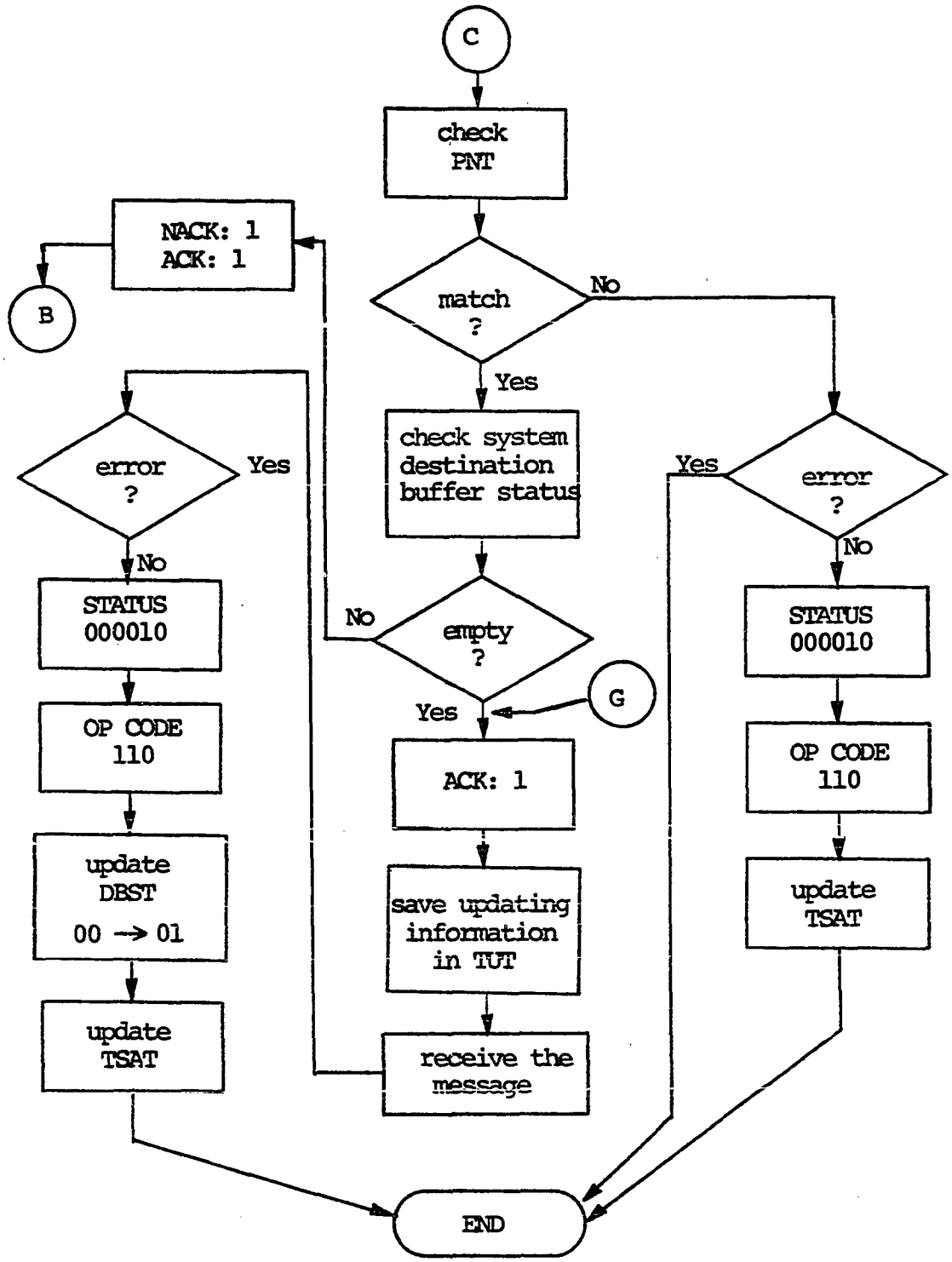


Figure A7. (continued)

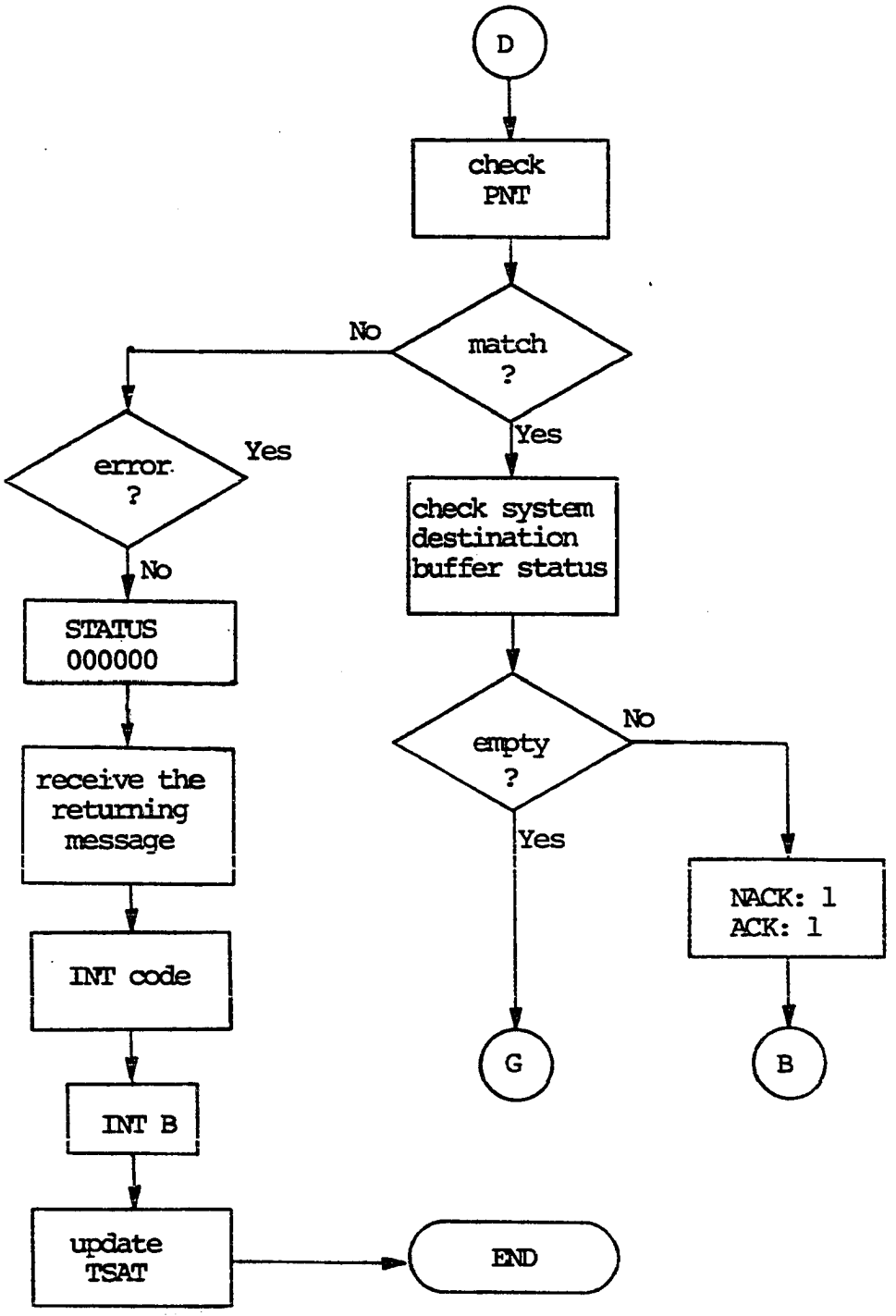


Figure A7. (continued)

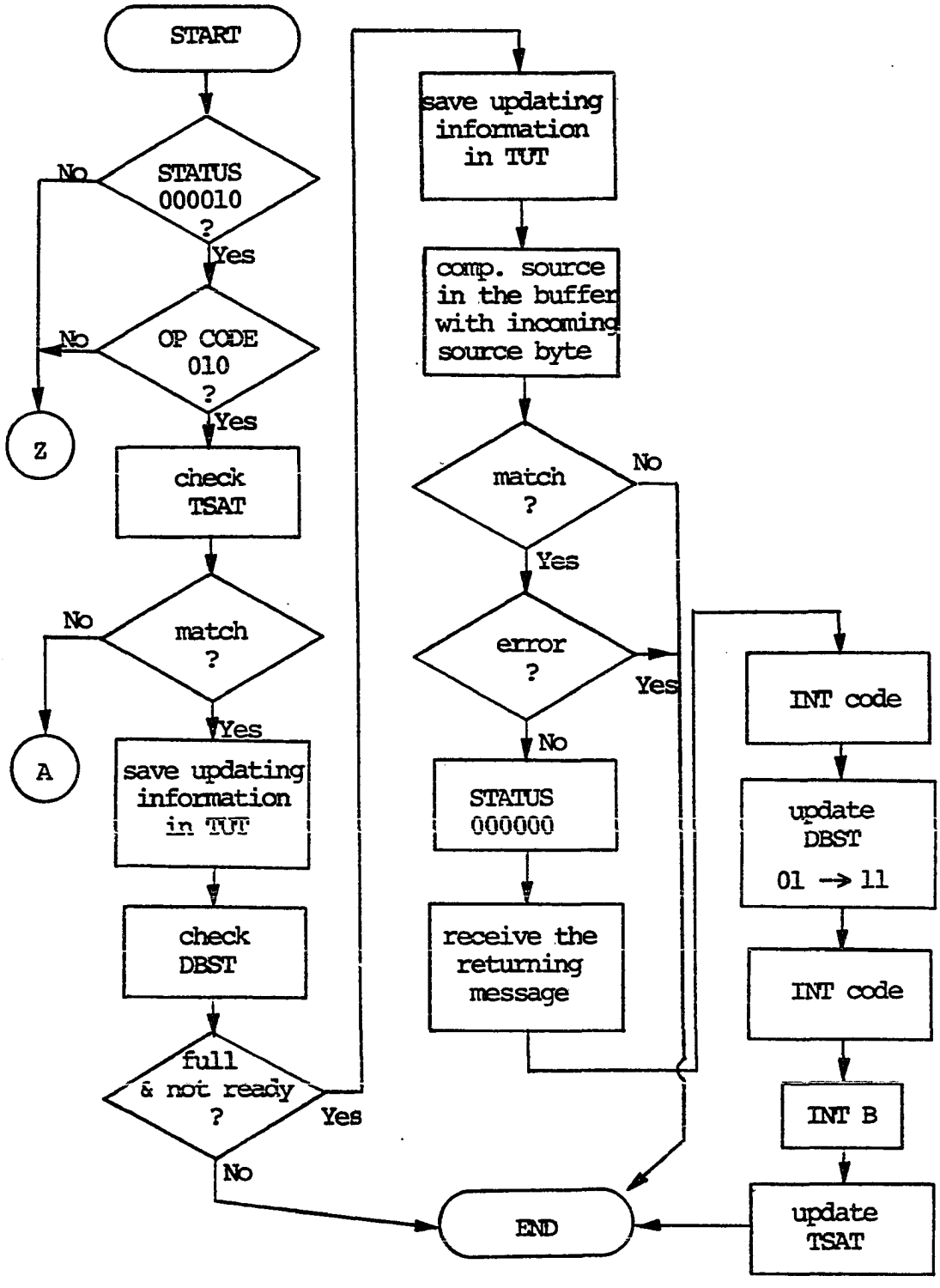


Figure A8. Special message S1.

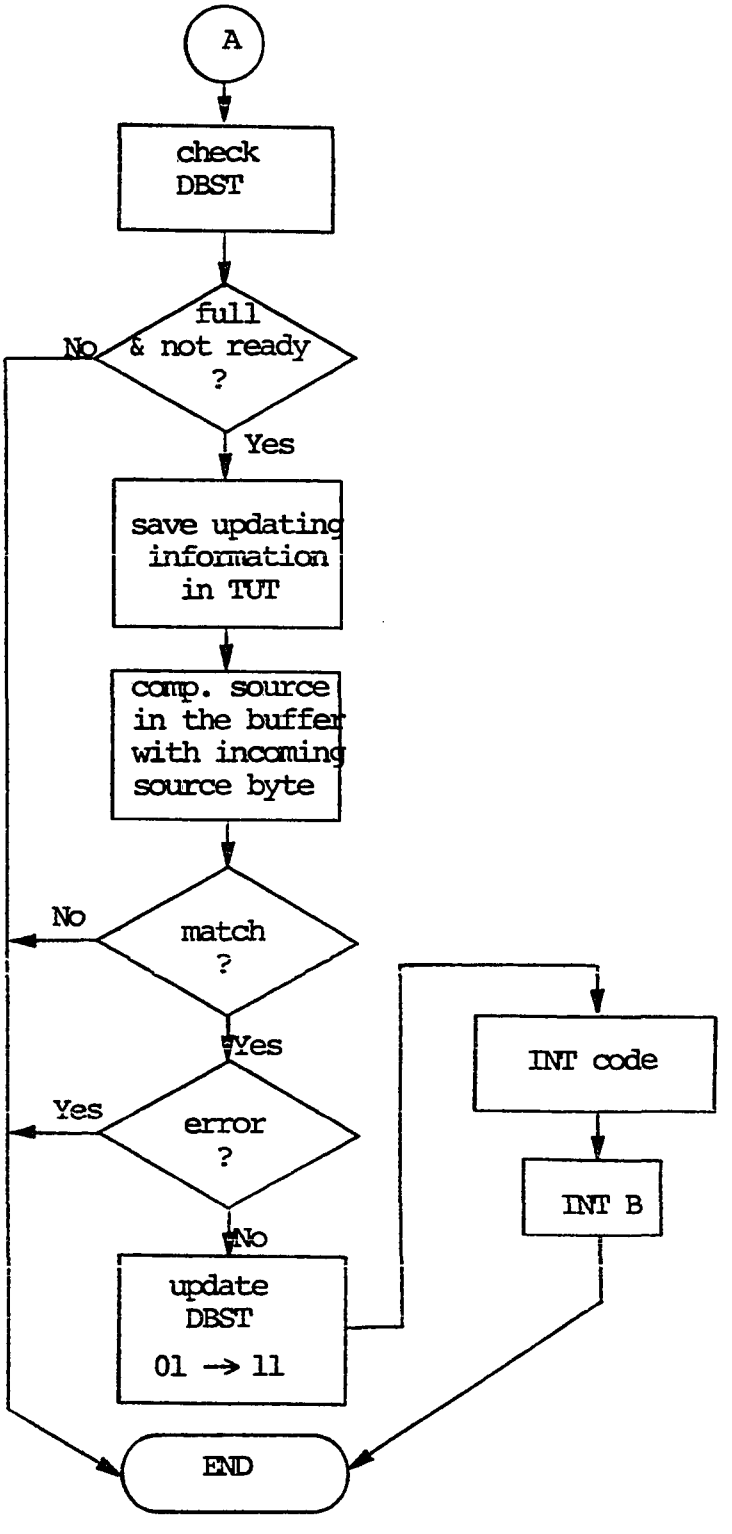


Figure A8. (continued)

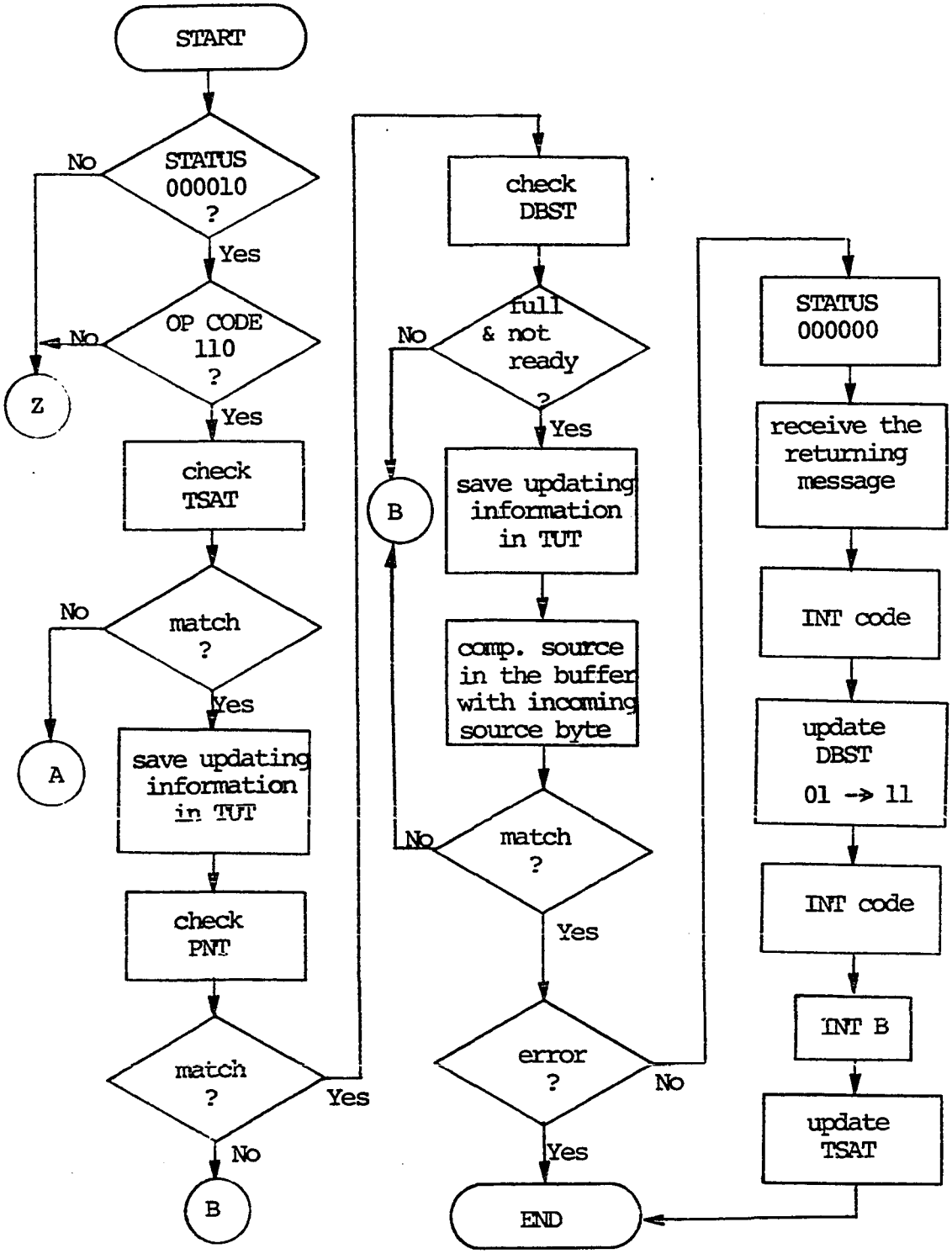


Figure A9. Special message S2.

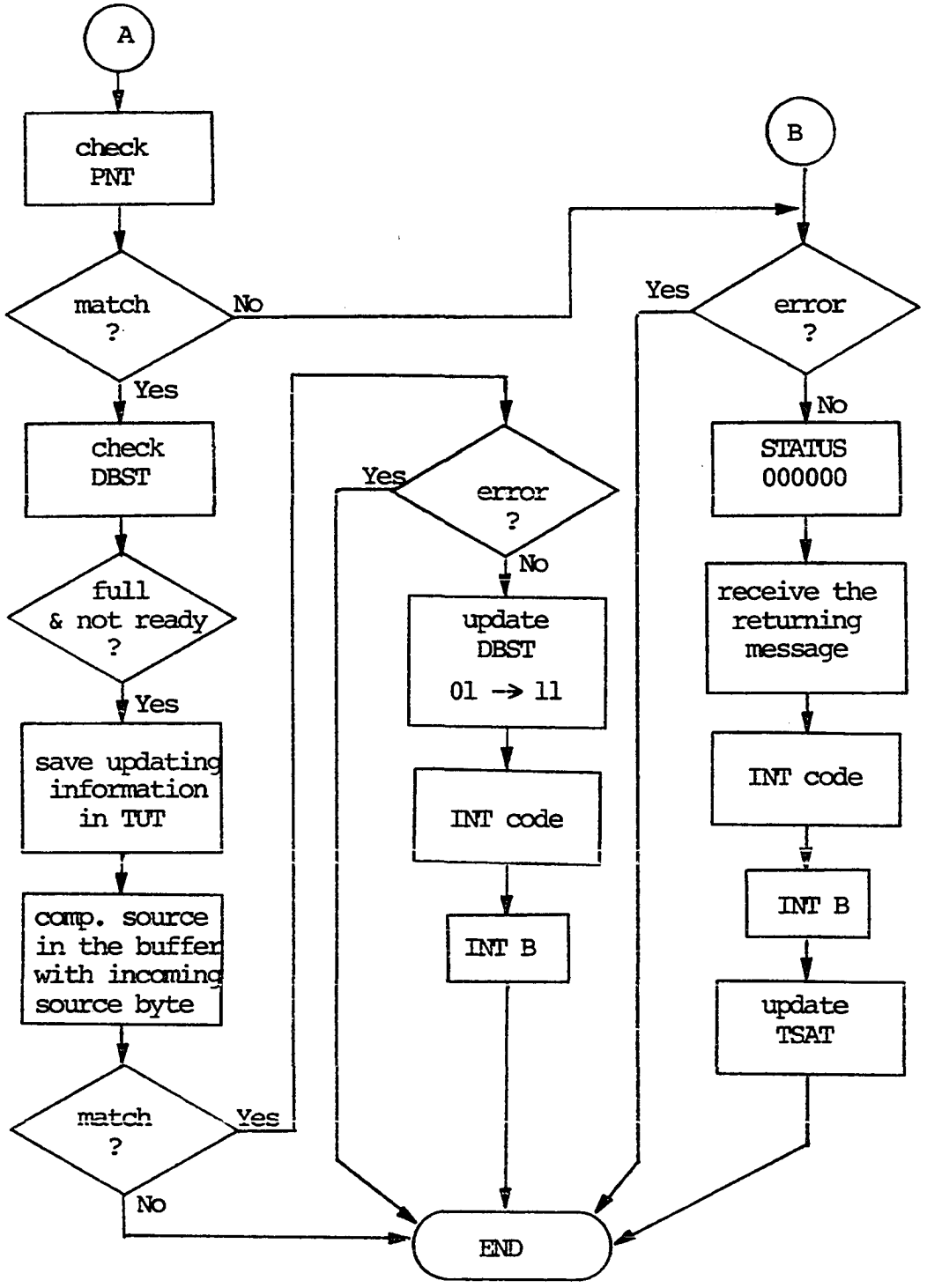


Figure A9. (continued)

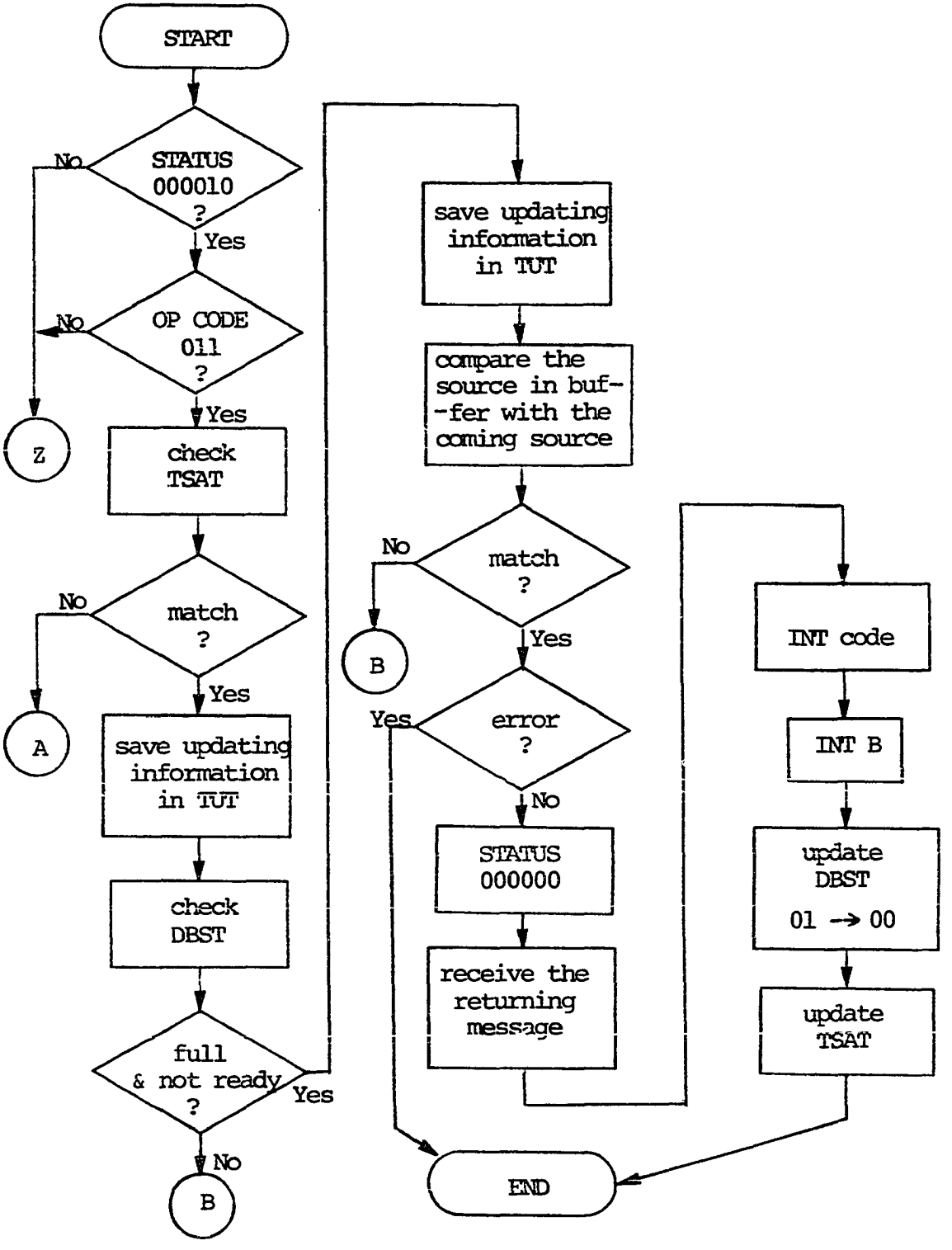


Figure A10. Special message S3.

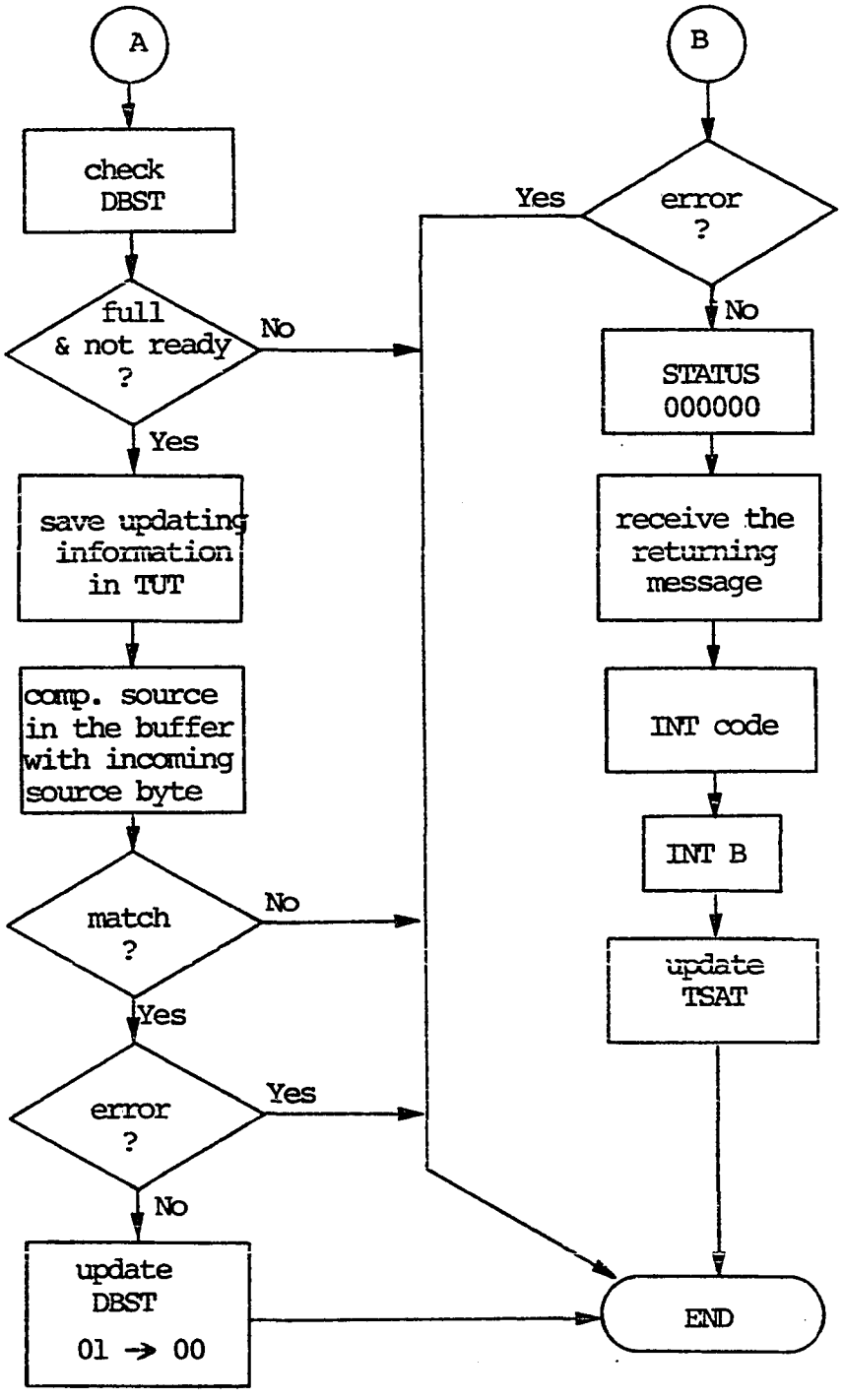


Figure A10. (continued)

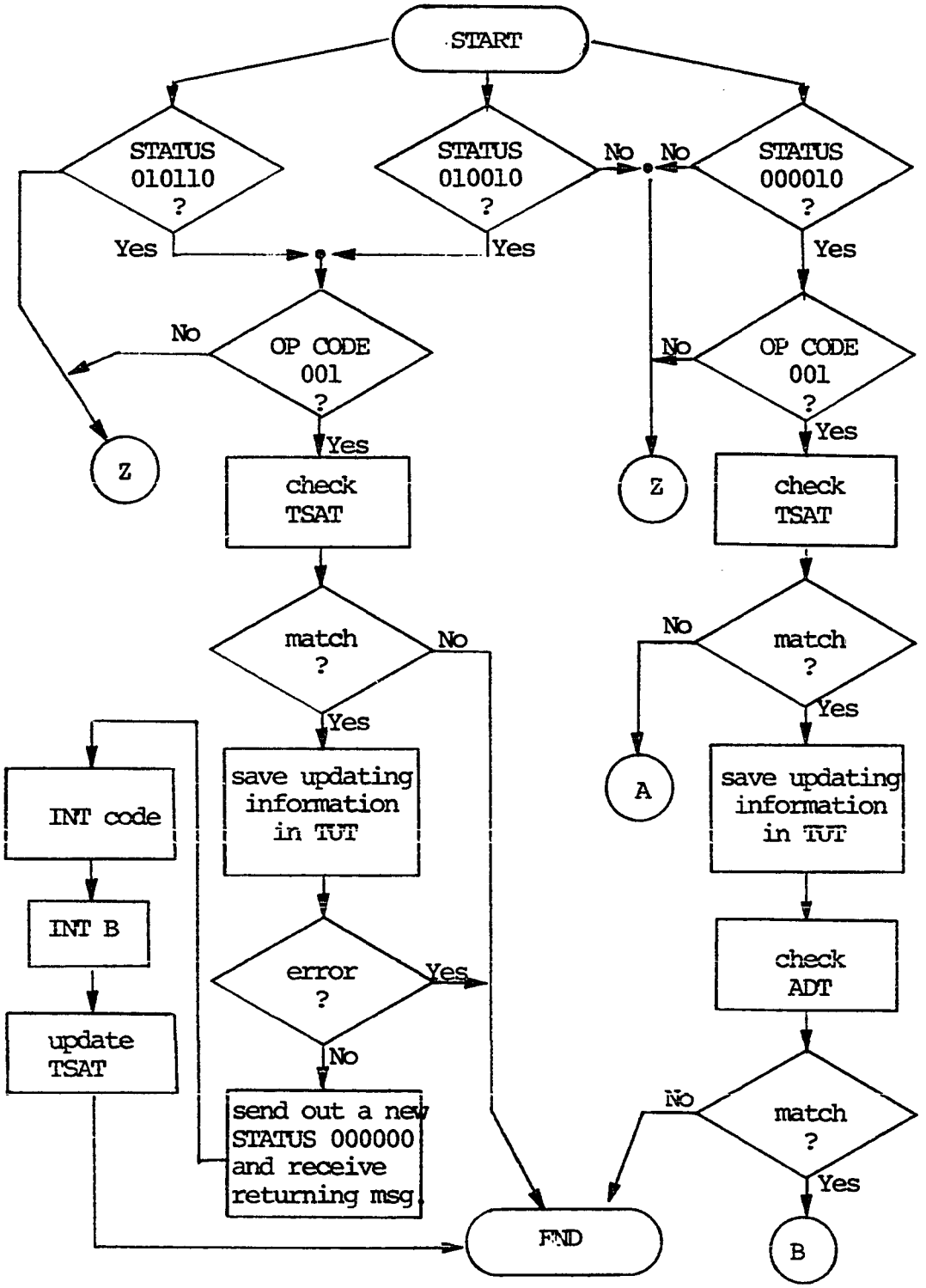


Figure A11. System message to a device address.

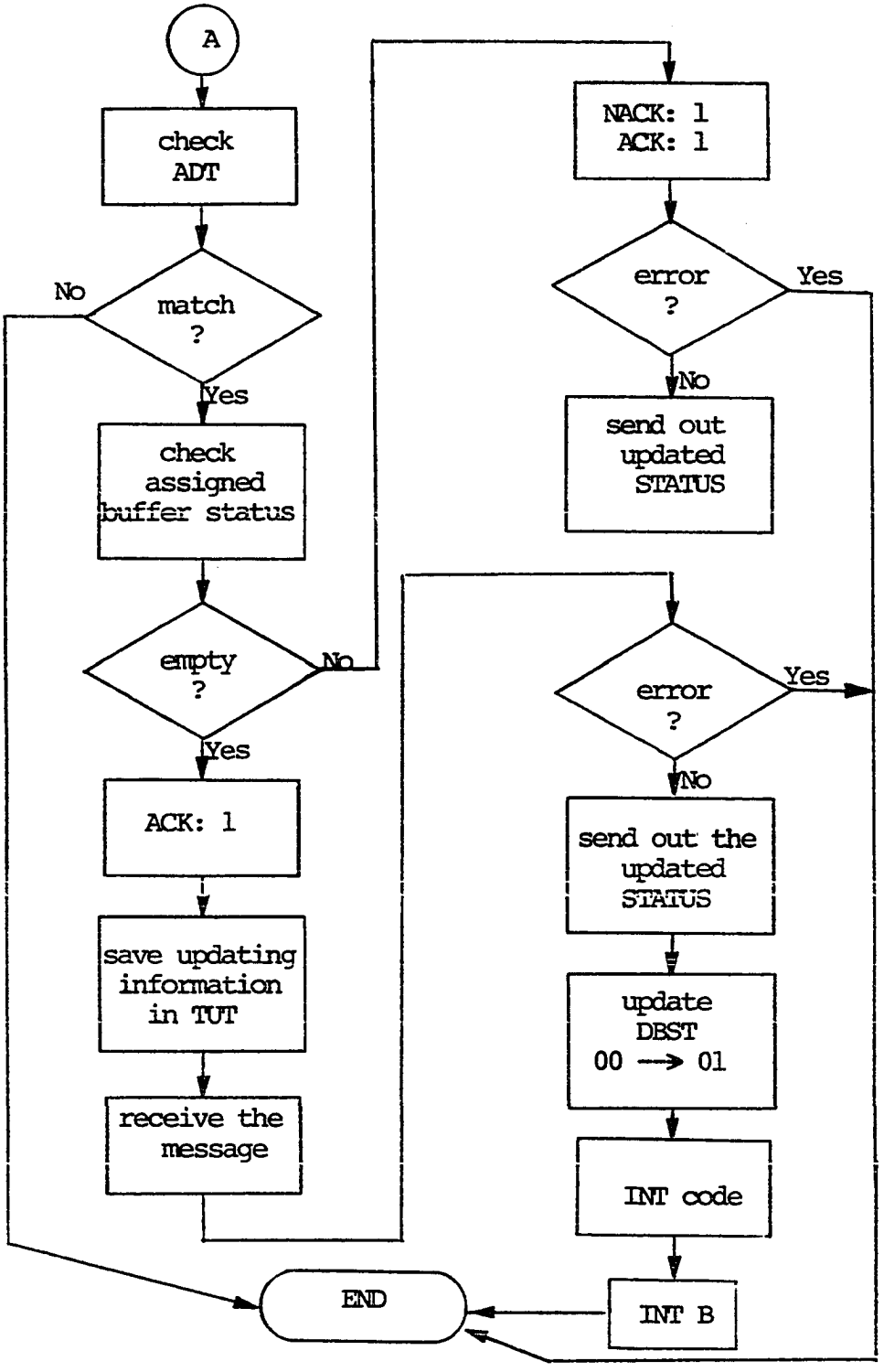


Figure A11. (continued)

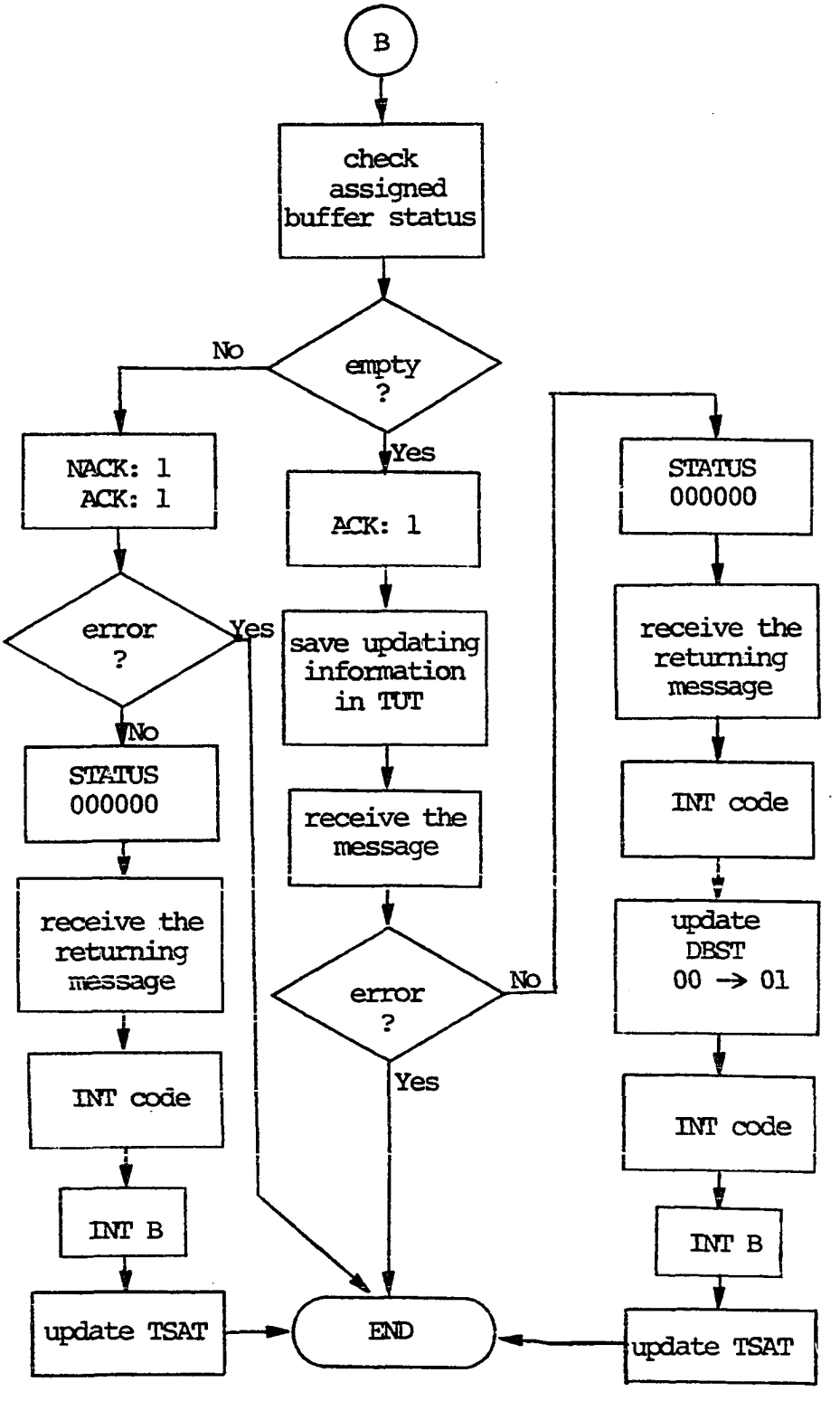


Figure A11. (continued)

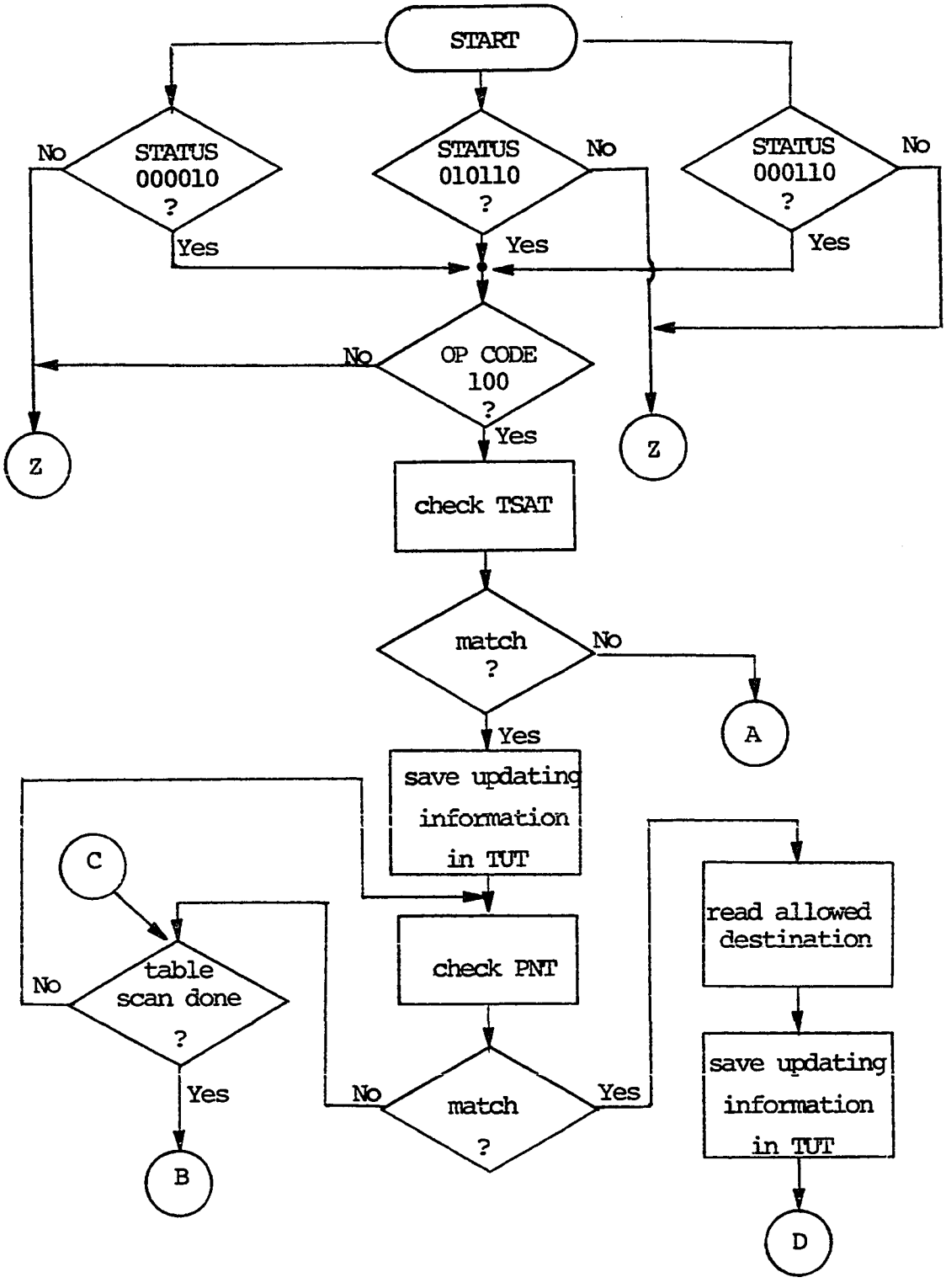


Figure A12. Regular message to a process.

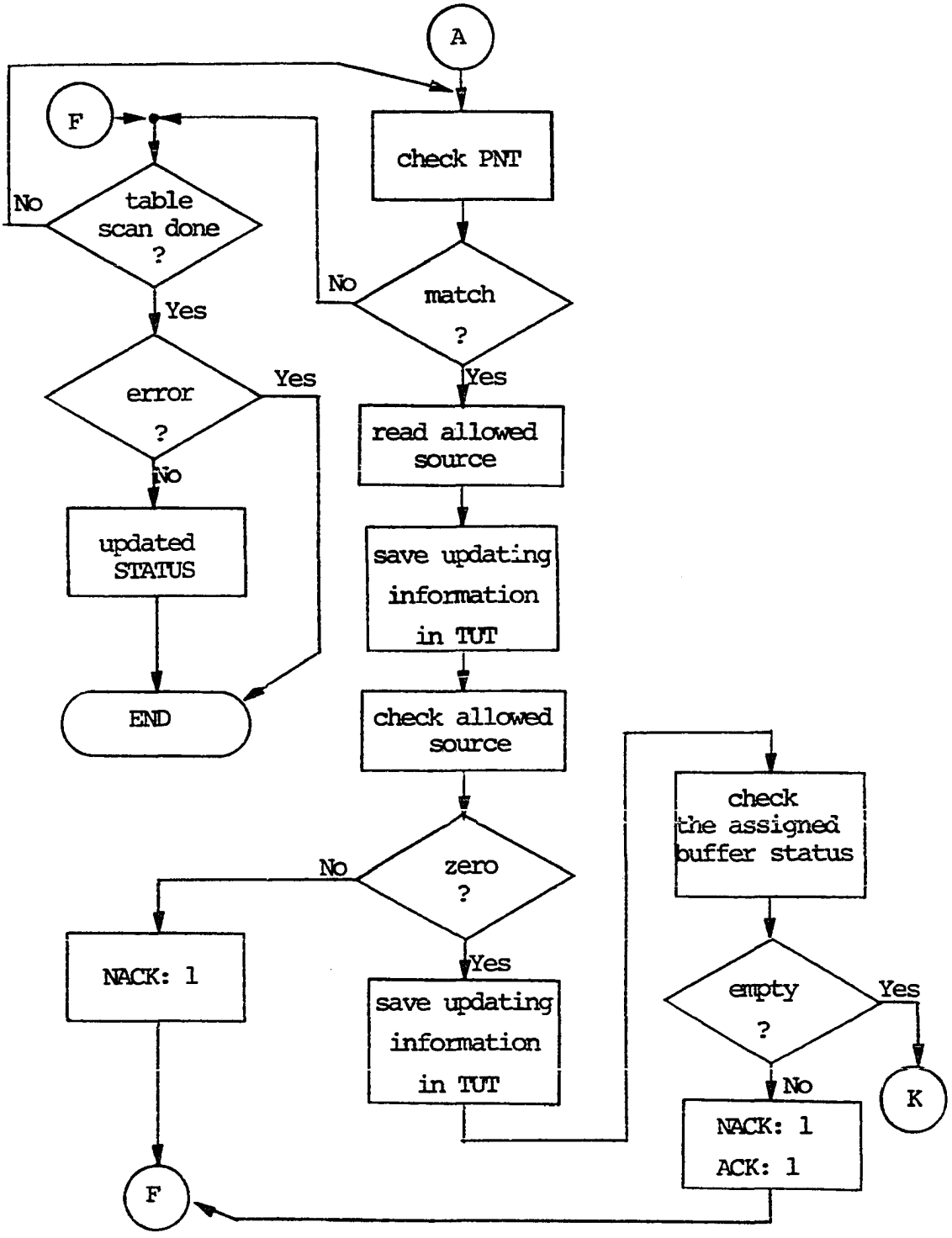


Figure A12. (continued)

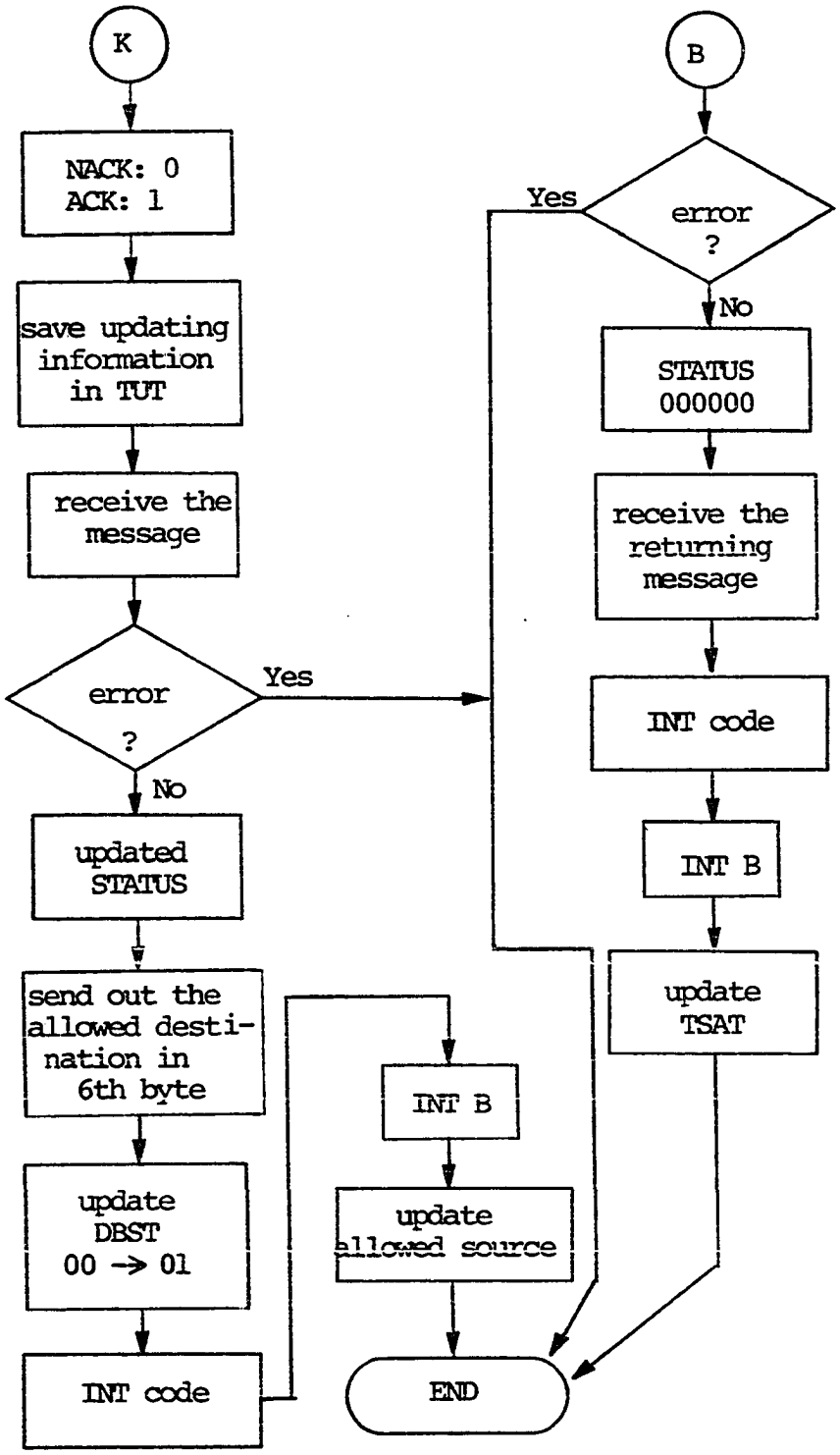


Figure A12. (continued)

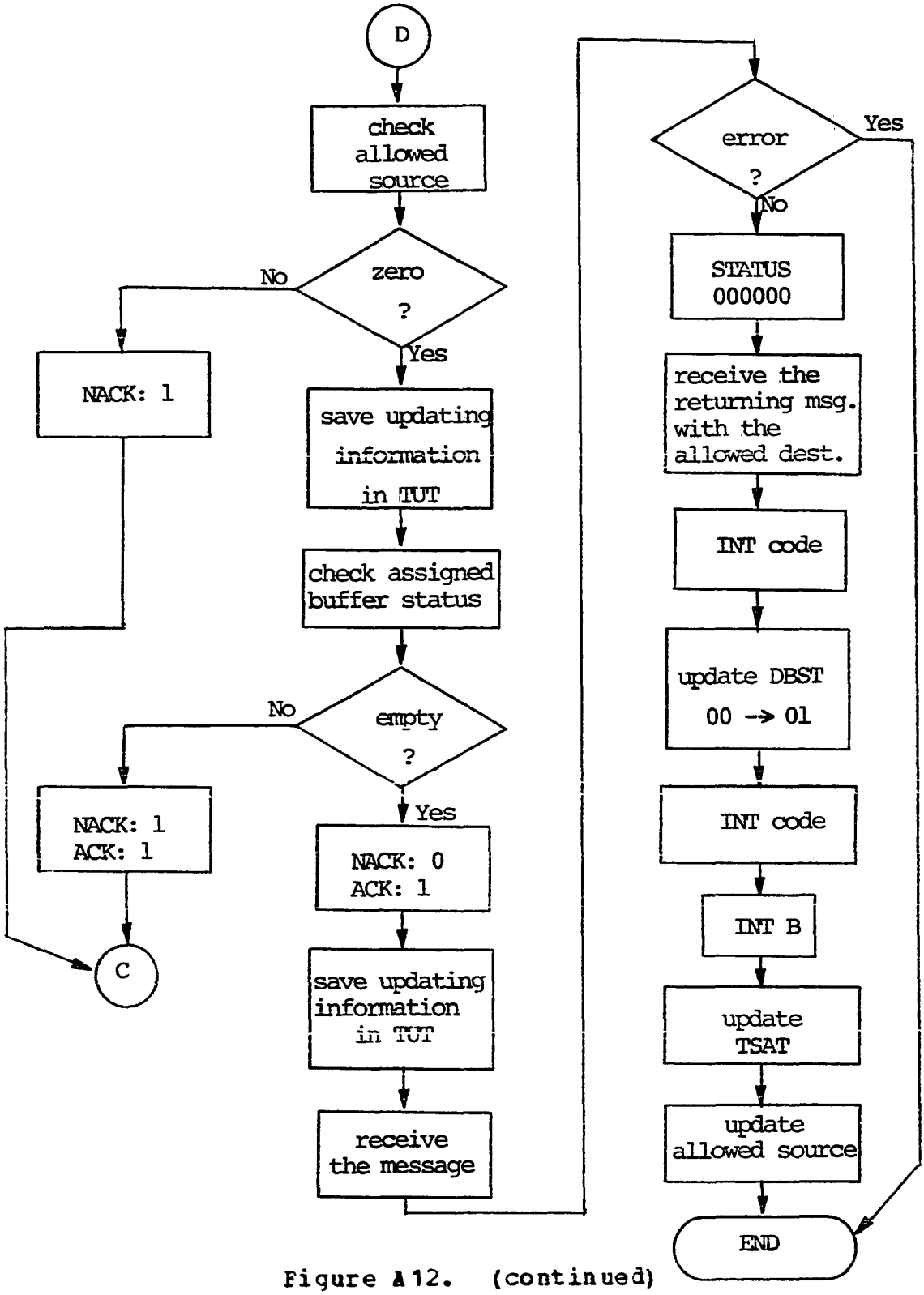


Figure A12. (continued)

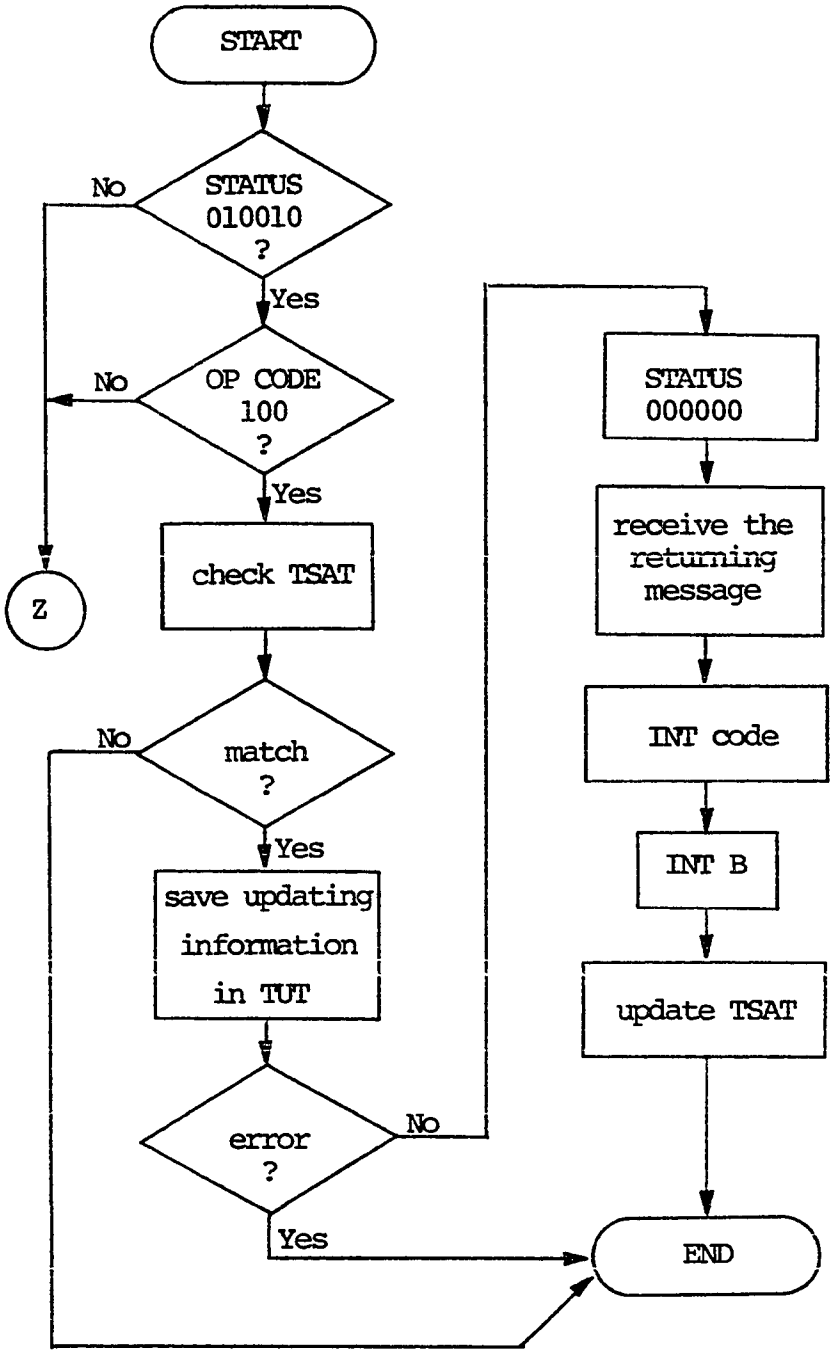


Figure A13. Returning regular message directed to a process after acknowledged.

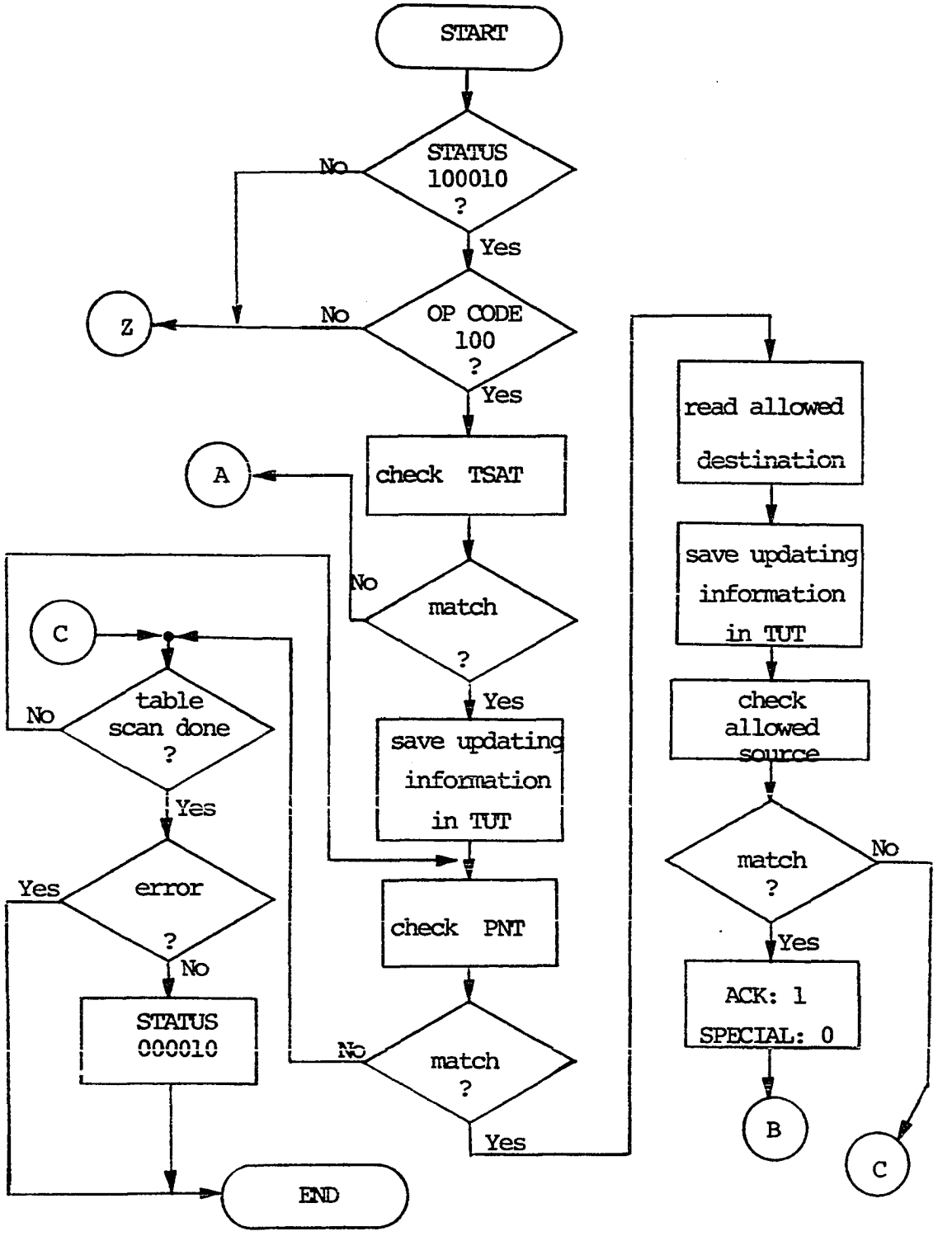


Figure A14. Retransmission of a regular message directed to a process.

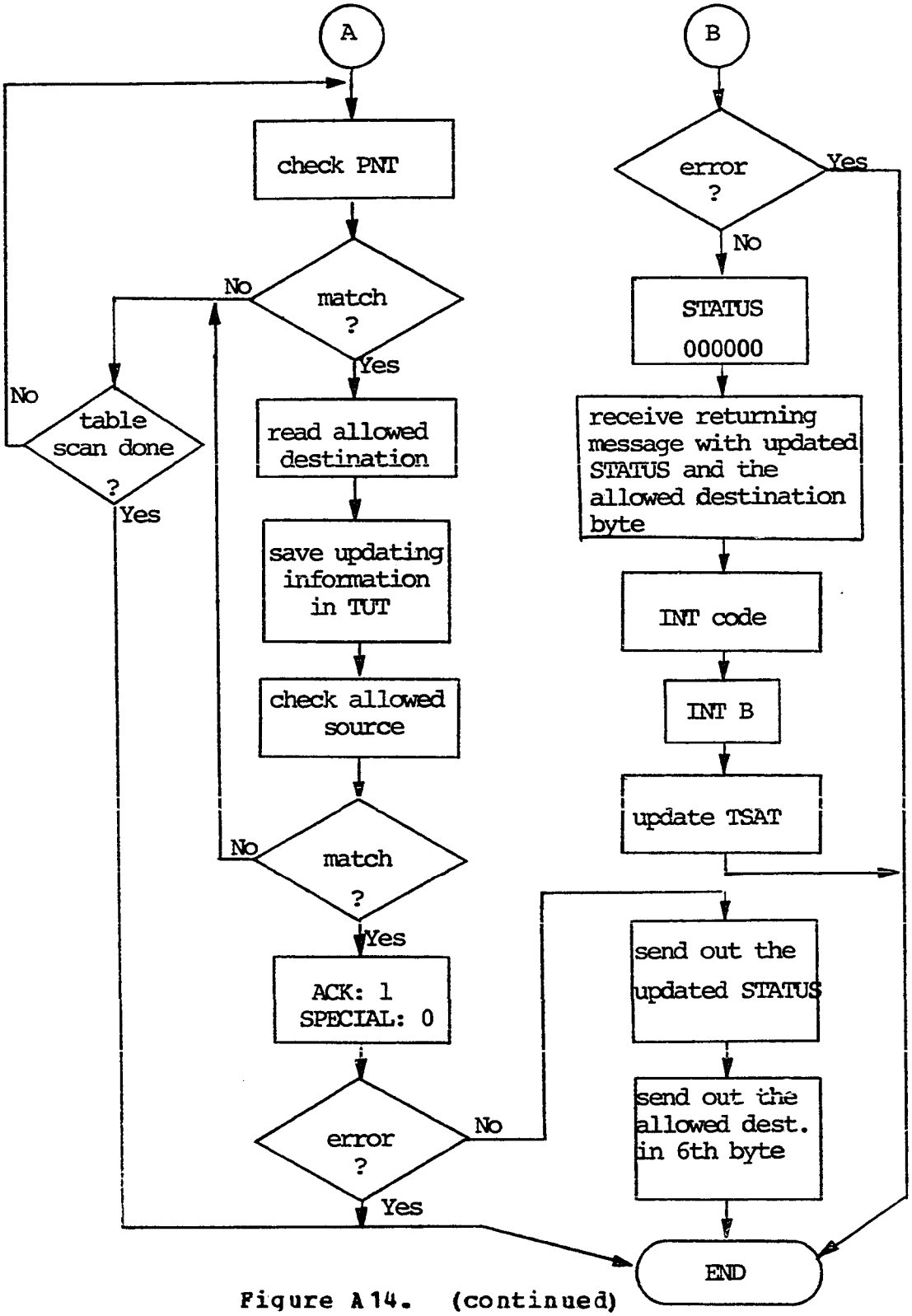


Figure A14. (continued)

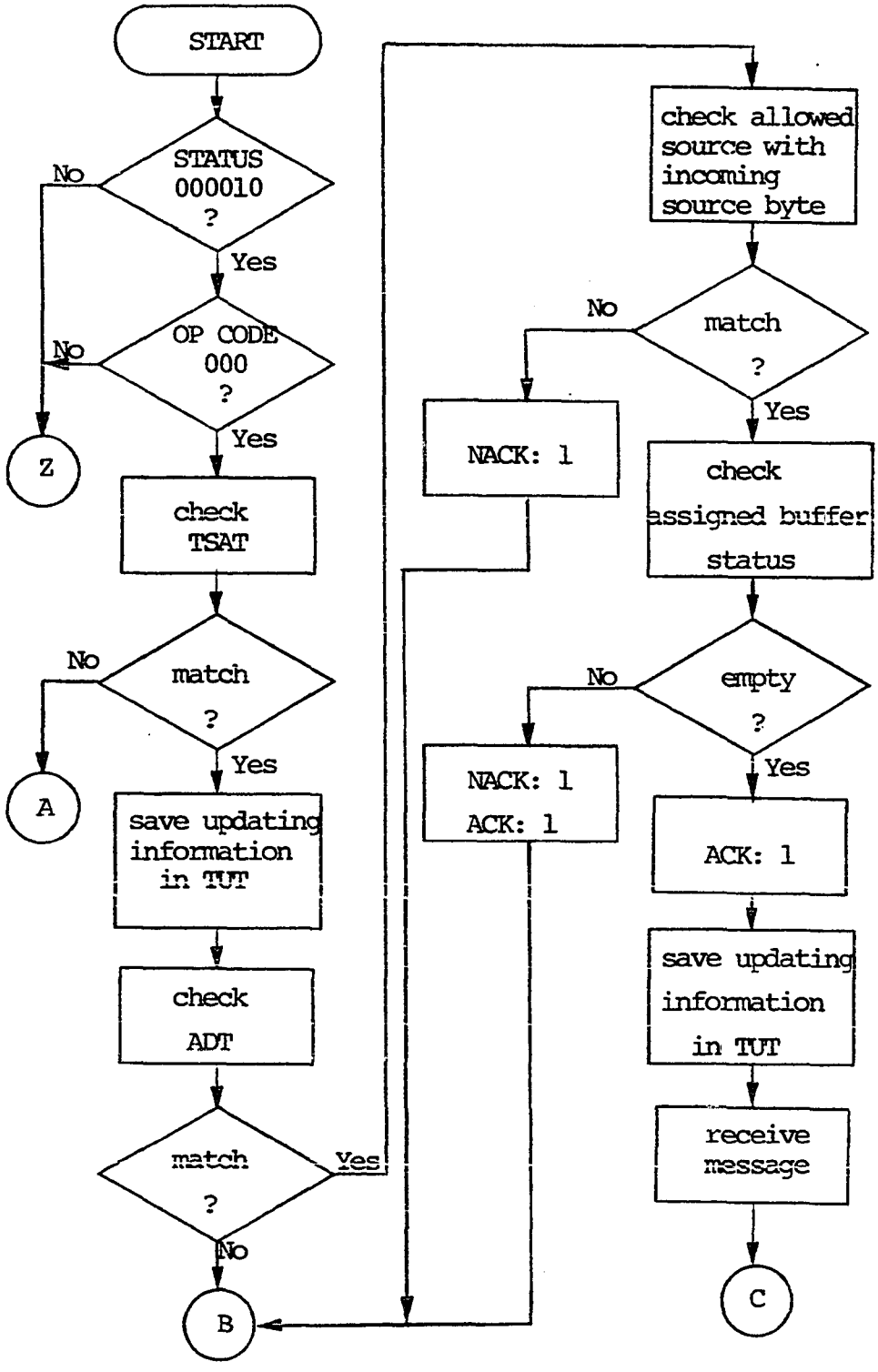


Figure A15. Regular message to a device address.

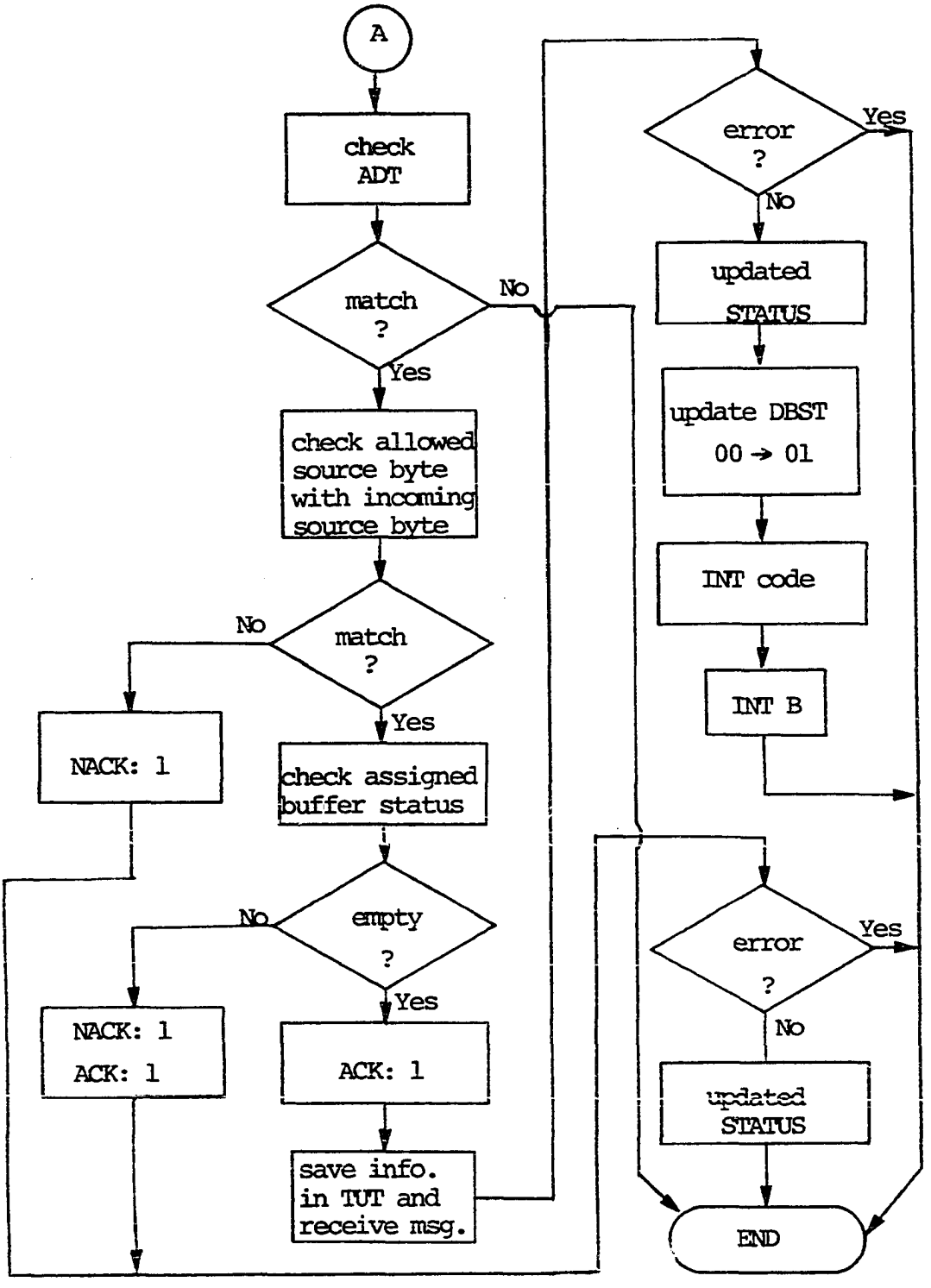


Figure A15. (continued)

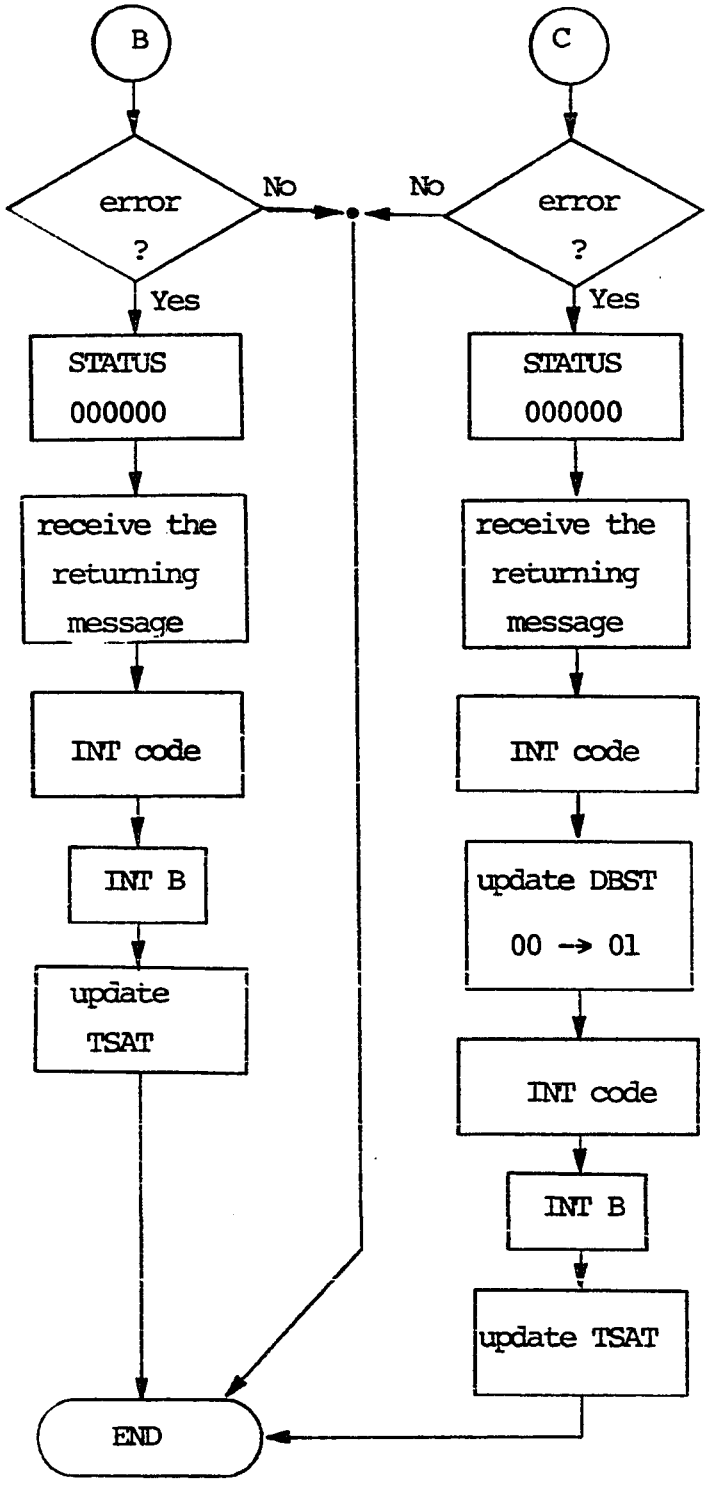


Figure A15. (continued)

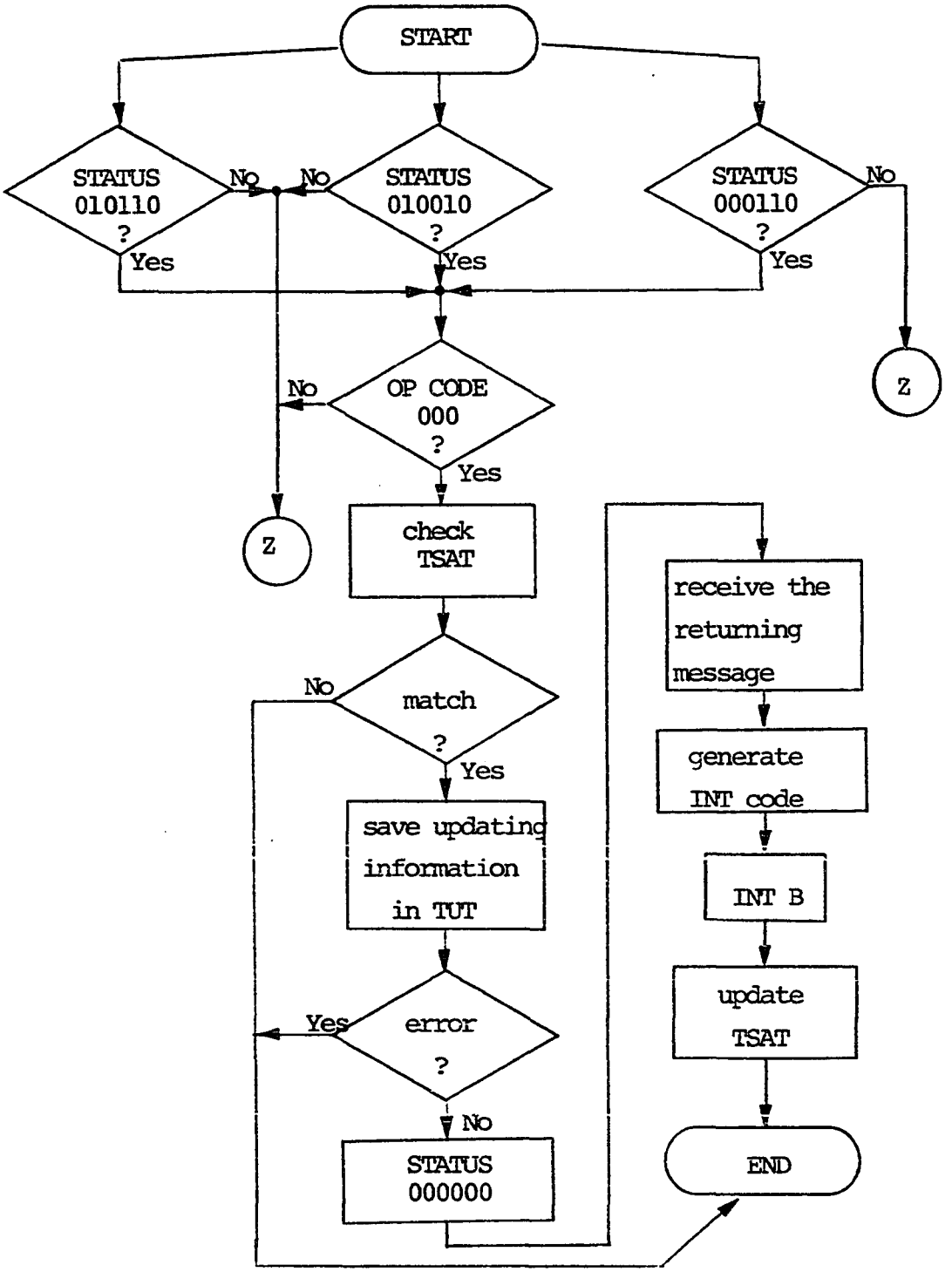


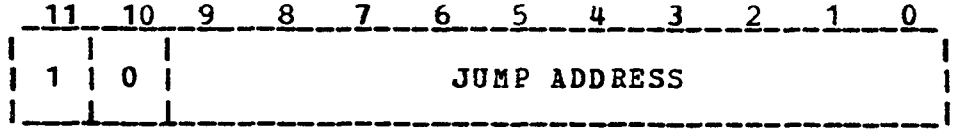
Figure A16. Returning regular message directed to a device address after acknowledged.

APPENDIX B. INSTRUCTION SET DESCRIPTIONS

The instructions described herein, each of which is 12 bit long, are those instructions which have been implemented in the actual interface processor. When appropriate, mnemonics are given for various instructions in addition to the binary code and the instruction execution times. Abbreviations which are used for various registers and counters are shown in Table B1.

JUMP

JMP 200 ns



The next instruction to be executed is in the memory location specified by the 10 JUMP ADDRESS bits. This requires that the contents of ROM AR1 is immediately replaced by the JUMP ADDRESS.

WAIT AND JUMP

WJMP 150 ns →

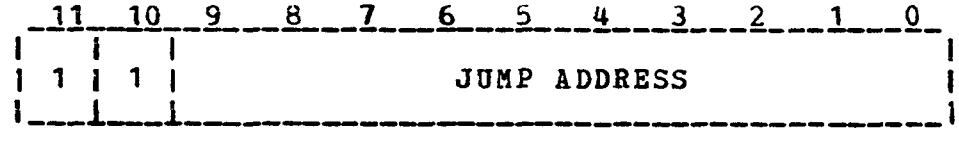


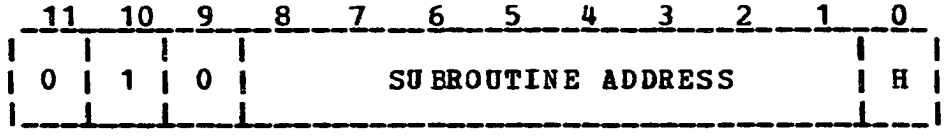
Table B1. Abbreviations for Registers and Counters

Abbreviation	Register/Counter
DA	Data Register A
DB	Data Register B
DC	Data Register C
RA	Address Register A
RB	Address Register B
RC	Address Register C
OSARA	Output Sequencer Address Register A
OSARB	Output Sequencer Address Register B
SBP	Source Buffer Pointer
XMITC	Transmit Counter
STSR	STATUS Register
ROM AR1	Read Only Memory Address Register 1
ROM AR2	Read Only Memory Address Register 2
OSCR	Output Sequencer Control Register
TS	Current Time Slot Number

Hold the normal operations of the interface until transmission error on the current time slot is checked by the associated repeater station. When this instruction is decoded, the Character Counter and the Bit Counter of the associated repeater station are to be checked to determine whether the Table Matcher can be used by the associated Micro-processor for a fast table checking operations or not. If no transmission errors are detected, the operations are same as those of JUMP.

CALL

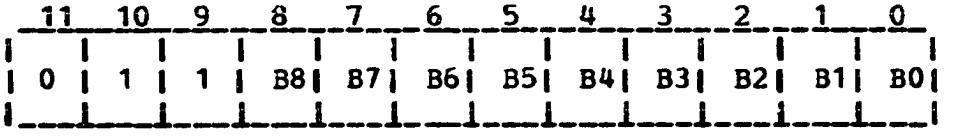
CALL 200 ns



Replace the contents of the ROM AR2 with the 8 SUBROUTINE ADDRESS bits. If the last bit (HALT bit) is set and execution of the subroutine initiated at the SUBROUTINE ADDRESS is done, hold the normal operations of the interface.

OUTPUT SEQUENCER CONTROL

OSC 150 ns



This instruction controls the Output Sequencer by giving the necessary information of expected operations to be done

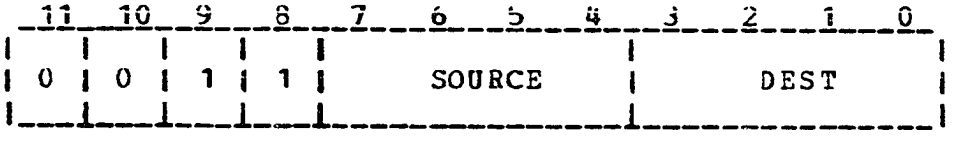
by the associated Output Sequencer. Only one bit out of B8, B7, B6 and B5 can be set at one time and only one bit out of B2, B1 and B0 can be set at one time. When this instruction is decoded, the Output Sequencer saves this control information in OSCR (Output Sequencer Control Register) until transmission error is checked on the time slot. The proper messages following this saved information are to be transmitted to the net by decoding the appropriate Character Counter and Bit Counter of the associated repeater, if no transmission error is detected. Otherwise, information saved in OSCR is invalid and therefore ignored. Each bit, B0 through B8, indicates a specific operation as shown in Table B2.

Table B2. Mnemonics and Operations

Bit	Mnemonics	Operations
B8	STF	Send out xx000010 for STATUS byte.
B7	STZ	Send out xx000000 for STATUS byte.
B6	STU	Send out the contents of the STSR for STATUS byte.
B5	STS	Send out xx100010 for STATUS byte.
B4	TAD	Send out the allowed destination for the requested process name for 6th byte.
B3	EOST	Enable Output Sequencer to transmit 15 bytes data in the selected source buffer whose beginning address is in OSARB and OSARA.
B2	S1	Send out xxxxx010 for OP CODE byte.
B1	S2	Send out xxxxx011 for OP CODE byte.
B0	S3	Send out xxxxx110 for OP CODE byte.

MOVE, SOURCE, DESTINATION

MOVE, SD 200 ns



Transfer the contents of the source register/counter into the destination register/counter. SOURCE and DEST are specified as hexadecimal codes for mnemonics. Specifications

of registers and counters for SOURCE and DEST are shown in Table B3.

Table B3. Register Specifications

Binary Code	Destination/Source
0000	DA
0001	DB
0010	DC
0011	RA
0100	RB
0101	OSARB
0110	OSARA
0111	TS
1000	SBP
1001	STSR

ADD AND READ

ADD/ADR 250 ns - 450 ns

11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	A1	A2	A3	A4	_	_	RD	RT

Add the required number (or contents of register) to the contents of RB and RA on the 12 bit binary full adder and store the sum back into the RB and RA. When RD (read) bit is

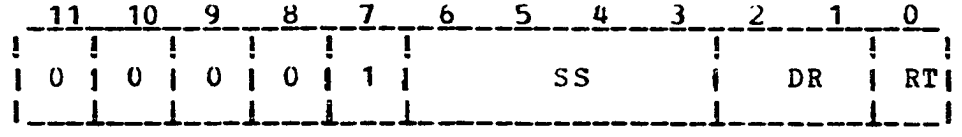
set, read one byte data from RAM location addressed by RB and RA and load the data into DA. The last bit is RETURN bit. The operations and mnemonics are shown in Table B4.

Table B4. Mnemonics and Operations

Bit	Mnemonics	Operations
A1	ADD.A1 or ADR.A1	Add one to the contents of RB and RA and store the sum back into the RB and RA.
A2	ADD.A2 or ADR.A2	Add three to the contents of RB and RA and store the sum back into the RB and RA.
A3	ADD.A3 or ADR.A3	Add the contents of DA to the contents of RB and RA and store the sum back into the RB and RA.
A4	ADD.A4 or ADR.A4	Add the high order three bits of RC to the contents of RB and RA and store the sum back into RB and RA.

LOAD, SPECIAL STORAGE, DESTINATION REGISTER

LOAD, SS, DR 200 ns



Read one byte data from RAM location addressed by the four SS bits and the two DR bits and load the data into the destination register specified by the two DR bits. Specifications for destination registers are shown in Table

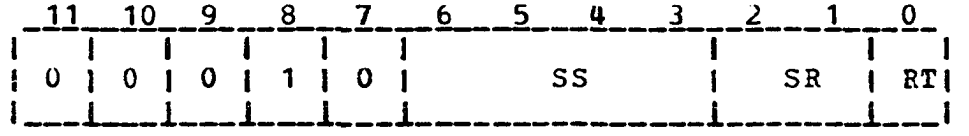
B5. If the upper most bit of the four SS bits is '1', then the high order four bits of actual RAM address are '1101'. Otherwise, the high order four bits of actual address are '0010'. Therefore nine bit actual memory address for the 512 bytes RAM can be specified in the 6 bit code space (4 SS bits plus 2 DR bits) in which the two DR bits can be also used to specified a destination register. For example, instruction 000011000110 says 'read one byte data from the memory location 110100011 and load the data into RC'.

Table B5. Specifications for DR/SR

DR/SR	Destination/Source register
00	RB
01	RA
10	DA
11	RC

STORE, SPECIAL STORAGE, SOURCE REGISTER

STORE, SS, SR 200 ns



Write the contents of the source register specified by the two SR bits into the RAM location addressed by SS and SR.

Specifications for source registers are same as LOAD and are also shown in Table B5. The upper most bit of the four SS bits is used in the same way as LOAD. For example, instruction code 000100111010 says 'write the contents of RA into the memory location 00101110 1'.

The reserved locations which can be accessed by these instructions, LOAD and STORE, are shown in Appendix D.

RECEIVE, n BYTES DATA

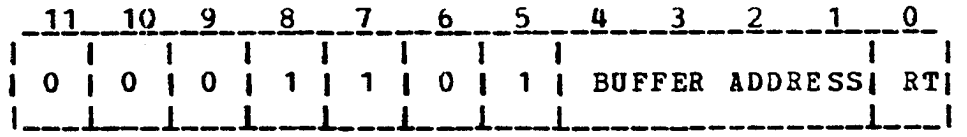
RCV n 350ns x n (n<16)

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	0	0	n				RT

Receive n bytes of data from the number two 16 bytes buffer and store these n bytes data into the n consecutive memory locations of the selected destination buffer. The initial address is currently stored in RB and RA. This instruction initiate a sequence of operations as follows: Read one byte data from the no. 2 16 bytes buffer, load the data into DA, write the contents of DA into the memory location addressed by RB and RA and increment the contents of RB and RA by one and repeat these operations until the required n bytes of data are all stored in the proper memory locations.

BDB, Nth BYTE

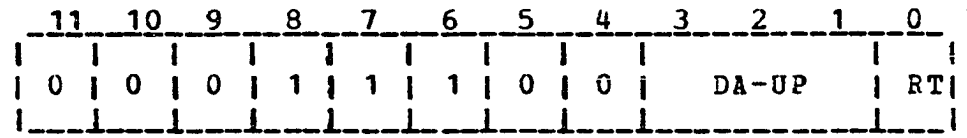
BCB N 200 ns



Read one byte data from the number 2 16 bytes buffer location addressed by the four BUFFER ADDRESS bits and load the data into DB.

UPDATE DA

150 ns



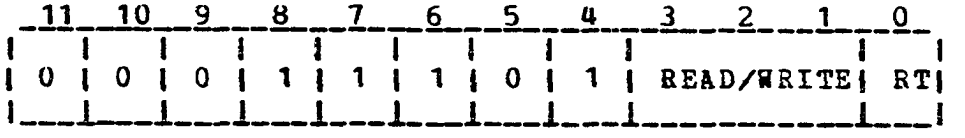
Update the contents of DA according to the three DA-UP bits as shown in Table B6.

Table B6. UPDATE DA Mnemonics and Operations

DA-UP	Mnemonics	Operations
000	DA(00)	xxxxxx00
001	DA(01)	xxxxxx01
010	DA(11)	xxxxxx11
011	DA(100)	100xxxxx
100	DA(010)	010xxxxx
101	DA(001)	001xxxxx
110	DA(SC)	Replace the low order 4 bits of DA with the low order 4 bits of DB.
111	DA(DT)	Replace the high order 4 bits of DA with the low order 4 bits of DB.

READ/WRITE

200 ns



WHEN THE READ/WRITE bits are 000, reading one byte data from memory is performed as follows: Read one byte data from the memory location addressed by RB and RA and load the data into DA.

When the READ/WRITE bits are 001, writing one byte data into memory is performed as follows: Write the contents of DA into the memory location addressed by RB and RA.

UPDATE STATUS REGISTER

200 ns

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	1	0	UPDATE			RT

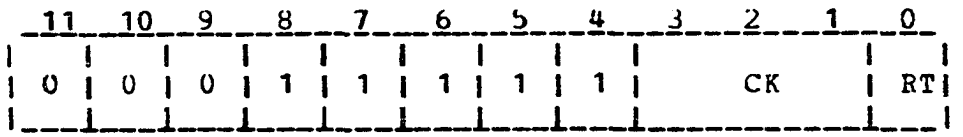
Update the contents of STSR according to the three UPDATE bits as shown in Table B7.

Table B7. UPDATE STATUS Mnemonics and Operations

UPDATE	Mnemonics	Operations
000	SU1	xxxxx1xx (NACK; 1)
001	SU2	xxx1xxxx (ACK; 1)
010	SU3	xxx1x1xx (NACK, ACK; 1)
011	SU4	xxx1x0xx (NACK; 0, ACK; 1)
100	SU5	xx01xxxx (SPECIAL; 0, ACK; 1)

CHECK

300 ns



Do checking operations according to the three CK bits as shown in Table B8. When the last bit (return bit) is set and any required match has been found, then SUBROUTINE ROM is disabled and the PROCEDURE ROM is activated.

Most of these checking instructions require two-way branch-out depending upon the match and non-match of the checking result. However, one instruction, CK2, requires three-way branch-out depending upon the information checked. For those two-way branch-out instructions, the next instruction to be executed is to be fetched from the memory address obtained by incrementing the current address by one or two depending upon the non-match or match of the checking respectively. For the three-way branch-out instruction, the next instruction to be executed is to be fetched from the memory address obtained by incrementing the current address by one, two or three depending upon the information checked.

For both cases, the memory address is automatically incremented to get the proper memory address of the next instruction to be executed depending upon the type of checking instruction and the result of checking.

Table B8. CHECK Mnemonics and Operations

CK	Mnemonics	Operations
000	CK1	Check match between DA and DB.
001	CK2	Check the low order two bits of DA for 00, 01, 11.
010	CKAB	Check the high order three bits of DA for 100, 010 and 001.
011	CKOP	Check the low order three bits of DA for 100.
100	CKXMITC	Check XMITC (Transmission counter) for 0100.

ENABLE TABLE MATCHER

400ns x table size

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	1	ETM			RT

Activate the Table Matcher for automatic table checking operations according to the three ETM bits as follows.

When ETM bits are 000 (ETM A), the following operations are performed: Activate the Table Matcher to check the match between DB and the data in RAM location addressed by RB and RA. The number of entries to be checked is stored in the low order 5 bits of RC and an entry increment for calculating the

next memory address is in the high order three bits of RC. In short, this instruction enables the Table Matcher to do automatic checking following read-check-increment sequence of operations until either a match is found or the total required number of checks are done.

When ETM bits are 001 (ETM B), following operations are performed: Activate the Table Matcher to find the low order two bits of DA for 01. The first data to be checked is stored in the RAM location addressed by RB and RA. The number of entries in the table to be checked and the entry increment are stored in RC in the same way as ETM A. In short, this instruction enables the Table Matcher to do automatic checking following read-check-increment sequence of operations until either 01 in the low order two bits of DA is found or the total required number of checks are done.

CLEAR REGISTER/COUNTER

150 ns

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0	CLR			RT

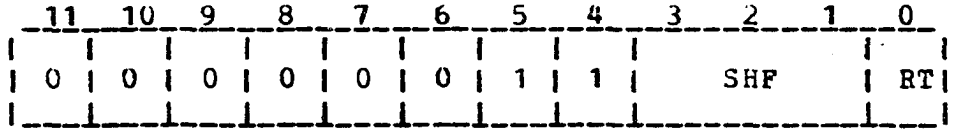
Replace the contents of registers or counters specified by the three CLR bits with zero as shown in Table B9.

Table B9. CLEAR Mnemonics and Operations

CLR	Mnemonics	Operations
000	CLR DA	Replace the contents of DA with 0.
001	CLR DB	Replace the contents of DB with 0.
010	CLR XMITC	Replace the contents of XMITC with 0.
011	CLR SBP	Replace the contents of SBP with 0.

SHIFT DA

150 ns



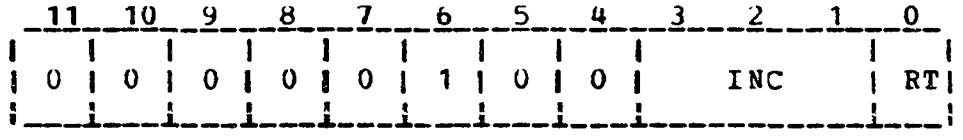
Shift the contents of DA according to the three SHF bits as shown in Table B10.

Table B10. SHIFT DA Mnemonics and Operations

SHF	Mnemonics	Operations
000	SHIFT A	Shift the contents of DA one bit to the left (low order bit to high order bit direction) with incoming carry '1'.
001	SHIFT B	Shift the contents of DA one bit to the left (low order bit to high order bit direction) with incoming carry '0'.

INCREMENT COUNTER

150 ns



Increment the contents of the counter specified by the three INC bits by one as shown in Table B11.

Table B11. INCREMENT COUNTER Mnemonics and Operations

INC	Mnemonics	Operations
000	INC SBP	Increment the contents of SBP by one.
001	INC XMITC	Increment the contents of XMITC by one.

INTERRUPT

150 ns

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1	INT			RT

Interrupt the Micro-processor according to the three INC bits as shown in Table B12.

Table B12. INTERRUPT Mnemonics and Operations

INT	Mnemonics	Operations
000	INT A	Interrupt the Micro-processor for either shut-down procedure or start-up procedure.
001	INT B	Interrupt the Micro-processor for either source or destination buffer or both.
010	INT C	Interrupt the Micro-processor to inform the results of a table checking when the Table Matcher has been used on Micro-processor's table checking operations.

HALT

HALT 50 ns

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	d	d	d	0

Stop the normal operations of the interface immediately in the same way as the case in which the last bit (halt bit) of CALL instruction is set when the accessed subroutine execution is done. When the interface is in a hold state by decoding this instruction, the Character Counter and Bit Counter in the associated repeater are to be checked to determine whether the Table Matcher can be used by the associ-

ated Micro-processor for a fast table checking operations. The 'd's in bits 3, 2, 1 are don't care bits.

RETURN

RTN 50 ns

11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	d	d	d	1

Activate the procedure ROM and disable the subroutine ROM same as the case in which the last bit (return bit) is set for the instructions whose last bit is return bit when the required instruction execution is done.

APPENDIX C. DESCRIPTIONS OF BASIC SUBROUTINES

Processing requirements of forty two basic subroutines stored in the 256 bytes (12 bit per byte) are described herein. The notations used for these basic subroutines are same as shown in the simplified flowcharts in Appendix E. For explicit descriptions, special storage locations in the reserved area are denoted as hexadecimal codes as specified in Appendix D if necessary.

X1: Save TSAT updating information in TUT after a match is found.

a) Store time slot number updating information in 80, 81 and 82.

b) Store assigned buffer updating information in 90, 91 and 92.

c) Store the source buffer number in 02.

X2: Save TSAT updating information in TUT after an empty location in TSAT is found.

a) Store time slot number updating information in 80, 81 and 82.

b) Store the selected source buffer number (contents of Source Buffer Pointer) updating information in 90, 91 and 92.

X3: Save SBST updating information in TUT after a ready source buffer is found.

a) Store the source buffer status updating information

in A0, A1 and A2.

b) Store the source buffer number in 02.

X4: Save DBST updating information in TUT after a required destination buffer status is found.

a) Store the destination buffer status updating information in B0, B1 and B2.

X5: Save the contents of registers temporarily.

a) Store the contents of RB, RA, DA and RC in F0, F1, F2 and F3 respectively.

b) Store Allowed Destination in 22.

X6: Save PNT updating information in TUT if the contents of the allowed source byte is zero.

a) Store the allowed source updating information in C0, C1 and C2.

Note: This routine is used only for a regular message to a process name.

X7: Save the destination buffer number assigned to the allowed destination in 12.

X8: Store system destination buffer number in 12.

X9: Set up the beginning address of the selected source buffer.

a) Load the source buffer number, in 02, into DA.

b) Load the beginning address of SBAT, in 50 and 51, into RB and RA respectively.

c) Calculate the source buffer address properly.

X11: Set up the beginning address of the selected destination buffer.

a) Load the destination buffer number, in 12, into DA.

b) Load the beginning address of DBAT, in 60 and 61, into RB and RA respectively.

c) Calculate the destination buffer address properly.

X12: Set up the SOURCE byte address of the selected destination buffer.

a) Load the destination buffer number, in 12, into DA.

b) Load the beginning address of DBAT, in 60 and 61, into RB and RA respectively.

c) Calculate the destination buffer address.

d) Calculate the SOURCE byte address of the buffer properly.

X13: Set up the OP CODE byte address of the selected source buffer.

a) Load the source buffer number, in 02, into DA.

b) Load the beginning address of SBAT, in 50 and 51, into RB and RA respectively.

c) Calculate the source buffer address.

d) Calculate the OP CODE byte address of the source buffer properly.

X14: Check the allowed source for zero.

X15: Check the allowed source for SOURCE byte of the incoming time slot.

X16: Check DBST.

a) Load the beginning address of DBST, in 40 and 41, into RB and RA respectively.

b) Load the destination buffer number, in 12, into DA.

c) Calculate the address of the required destination buffer status to be checked.

d) Check the destination buffer status.

X17: Load the contents of registers, which were saved temporarily, back into the associated registers.

a) Load the contents of F0, F1, F2 and F3 into RB, RA, DA and RC respectively.

b) Prepare for continuing the table checking operations.

X18a: Update time slot number of TSAT.

a) Load the contents of 80, 81 and 82 into RB, RA and DA respectively.

b) Write the contents of DA into the memory location addressed by RB and RA.

X18b: Update the assigned buffer of TSAT.

a) Load the contents of 90, 91 and 92 into RB, RA and DA respectively.

b) Update the high order three bits of DA to 100.

c) Write the contents of DA into the memory location addressed by RB and RA.

X18c: Update an assigned buffer of TSAT.

a) Load the contents of 90, 91 and 92 into RB, RA and DA respectively.

b) Update the high order three bits of DA to 010.

c) Write the contents of DA into the memory location addressed by RB and RA.

X18d: Update an assigned buffer in TSAT.

a) Load the contents of 90, 91 and 92 into RB, RA and DA respectively.

b) Update the high order three bits of DA to 001.

c) Write the contents of DA into the memory location addressed by RB and RA.

X18e: Update an assigned buffer in TSAT.

a) Load the contents of 90, 91 and 92 into RB, RA and DA respectively.

b) Update the high order three bits of DA to 000.

c) Write the contents of DA into the memory location addressed by RB and RA.

X18f: Update SBST.

a) Load the contents of A0, A1 and A2 into RB, RA and DA respectively.

b) Write the contents of DA into the memory location addressed by RB and RA.

X18g: Update DBST.

a) Load the contents of B0, B1 and B2 into RB, RA and DA respectively.

b) Write the contents of DA into the memory location addressed by RB and RA.

X18h: Update the allowed source byte in PNT.

a) Load the contents of C0, C1 and C2 into RB, RA and DA respectively.

b) Write the contents of DA into the memory location addressed by RB and RA.

X19: Store an interrupt code for a source buffer.

a) Load the contents of 02 into DA.

b) Generate a proper interrupt code.

c) Store the interrupt code into 32.

X20: Store an interrupt code for a destination buffer.

a) Load the contents of 12 into DA.

b) Generate a proper interrupt code.

c) Store the interrupt code into 32.

X21a: Store an interrupt code to inform that the Table Matcher found a match for the Micro-processor's table checking operations.

a) Generate a proper interrupt code.

b) Store the interrupt code into 52.

X21b: Store an interrupt code to inform that the Table Matcher could not find a match for the Micro-processor's table checking operations.

a) Generate a proper interrupt code.

b) Store the interrupt code into 52.

X21c: Store an interrupt code for activating the Microprocessor to initiate the shut-down procedure after detecting a power failure.

a) Generate a proper interrupt code.

b) Store the interrupt code into 52.

X21d: Store an interrupt code for activating the Microprocessor to initiate the start-up procedure when power becomes normal.

a) Generate a proper interrupt code.

b) Store the interrupt code into 52.

X23: Prepare for checking the acknowledged time slots in TSAT.

a) Load the contents of 00 and 01 into RB and RA respectively.

b) Load the contents of 03 into RC.

c) Replace the contents of DB with the current time slot number.

X24: Prepare for checking an empty time slot location in TSAT.

a) Load the contents of 00 and 01 into RB and RA respectively.

b) Load the contents of 03 into RC.

c) Clear the contents of DB.

X25: Prepare for checking process names in PNT.

a) Load the contents of 10 and 11 into RB and RA re-

spectively.

b) Load the contents of 13 into RC.

c) Load the DESTINATION byte of the time slot into DB.

X26: Prepare for checking allowed destinations in ADT.

a) Load the contents of 20, 21 and 23 into RB, RA and RC respectively.

b) Load the DESTINATION byte of the current time slot into DB.

X27: Prepare for checking SBST to find a ready source buffer to be transmitted.

a) Load the contents of 30, 31 and 33 into RB, RA and RC respectively.

X28: Access the argument list to obtain checking informations when the Table Matcher is used for checking a table on the Micro-processor's request.

a) Load the contents of F0, F1, F2 and F3 into RB, RA, DA and RC respectively.

b) Replace the contents of DB with the contents of DA.

X29: Load the updated STATUS byte and the following 5 bytes of returning message into the original source buffer.

X30: Load the updated STATUS byte, the following 4 bytes of returning message and the allowed destination into the original source buffer.

a) Write the updated STATUS byte into the RAM location addressed by RB and RA.

b) Receive 4 bytes of returning message (OP CODE, DESTINATION, SOURCE, SEQUENCE) into the consecutive 4 bytes of memory locations.

c) Load the contents of 22 into DA.

d) Write the contents of DA into the 6th byte of the original source buffer.

X31: Return memory location where the Table Matcher found a match for the Micro-processor's checking request.

a) Store the contents of RB and RA into the memory location D0 and D1 respectively.

X32: Load allowed destination into DA and replace the contents of DC with the contents of DA.

a) Load the contents of 22 into DA.

b) Replace the contents of DC with the contents of DA.

X33: Load the updated STATUS and the following 14 bytes incoming message into the selected destination buffer.

APPENDIX D. SPECIAL STORAGE

The reserved locations in RAM described herein are those which are being used as special storages. Each reserved location is used for specific purpose, mostly for the temporary data storage. Total 64 bytes out of 512 bytes RAM are reserved for this purpose. In this way, significant amount of hardware involved in designing the interface can be saved by reducing the number of scratchpad registers inside the interface.

Table D1. Reserved locations for TAT, TUT and IIT

TYPE	address	SS	register		Usage
TAT	001000000	0	0	RB	beginning address of TSAT
TAT	001000001	0	1	RA	
TUT	001000010	0	2	DA	source buffer number
TAT	001000011	0	3	RC	no. of entries and an entry increment in TSAT
TAT	001000100	1	0	RB	beginning address of PNT
TAT	001000101	1	1	RA	
TUT	001000110	1	2	DA	destination buffer number
TAT	001000111	1	3	RC	no. of entries and an entry increment in PNT
TAT	001001000	2	3	RB	beginning address of ADT
TAT	001001001	2	1	RA	
TUT	001001010	2	2	DA	allowed destination
TAT	001001011	2	3	RC	number of entries and an entry increment in ADT
TAT	001001100	3	0	RB	beginning address of SBST
TAT	001001101	3	1	RA	
IIT	001001110	3	2	DA	interrupt code for source/destination buffer
TAT	001001111	3	3	RC	no. of entries and an entry increment in SBST

Table D1. (continued)

TYPE	address	SS	register		Usage
TAT	001010000	4	0	RB	beginning address of DBST
TAT	001010001	4	1	RA	
	001010010	4	2		
	001010011	4	3		
TAT	001010100	5	0	RB	beginning address of SBAT
TAT	001010101	5	1	RA	
IIT	001010110	5	2	DA	interrupt code for start-up, shut-down procedures and allowing Table Matcher to Micro-processor
	001010111	5	3		
TAT	001011000	6	0	RB	beginning address of DBAT
TAT	001011001	6	1	RA	
	001011010	6	2		
	001011011	6	3		
IIT	001011100	7	0	RB	beginning address of a table
IIT	001011101	7	1	RA	
IIT	001011110	7	2	DA	data to be compared
IIT	001011111	7	3	RC	no. of entries and an entry increment
					argument list for Micro-processor

Table D1. (continued)

TYPE	address	SS	register		Usage
TUT	110100000	8	0	RB	time slot number updating
TUT	110100001	8	1	RA	information for TSAT
TUT	110100010	8	2	DA	
	110100011	8	3		
TUT	110100100	9	0	RB	assigned buffer updating
TUT	110100101	9	1	RA	information for TSAT
TUT	110100110	9	2	DA	
	110100111	9	3		
TUT	110101000	A	0	RB	source buffer status updating
TUT	110101001	A	1	RA	information for SBST
TUT	110101010	A	2	DA	
	110101011	A	3		
TUT	110101100	B	0	RB	destination buffer status
TUT	110101101	B	1	RA	updating information for DBST
TUT	110101110	B	2	DA	
	110101111	B	3		

Table D1. (continued)

TYPE	address	SS	register		Usage
TUT	110110000	C	0	RB	allowed source updating
TUT	110110001	C	1	RA	information for PNT
TUT	110110010	C	2	DA	
	110110011	C	3		
IIT	110110100	D	0	RB	interrupt data return
IIT	110110101	D	1	RA	for Micro-processor's job
	110110110	D	2		
	110110111	D	3		
	110111000	E	0		
	110111001	E	1		
	110111010	E	2		
	110111011	E	3		
TAT	110111100	F	0	RB	temporary saving area
TAT	110111101	F	1	RA	for RB, RA, DA and RC
TAT	110111110	F	2	DA	
TAT	110111111	F	3	RC	

APPENDIX E. SIMPLIFIED FLOWCHARTS

The flowcharts shown in Appendix A are simplified applying the basic subroutines and the instructions shown as mnemonics according to the STATUS and OP CODE byte described in Appendix C and Appendix B respectively. The processing requirements for different procedures are combinations in Table 3 and some other requirements.

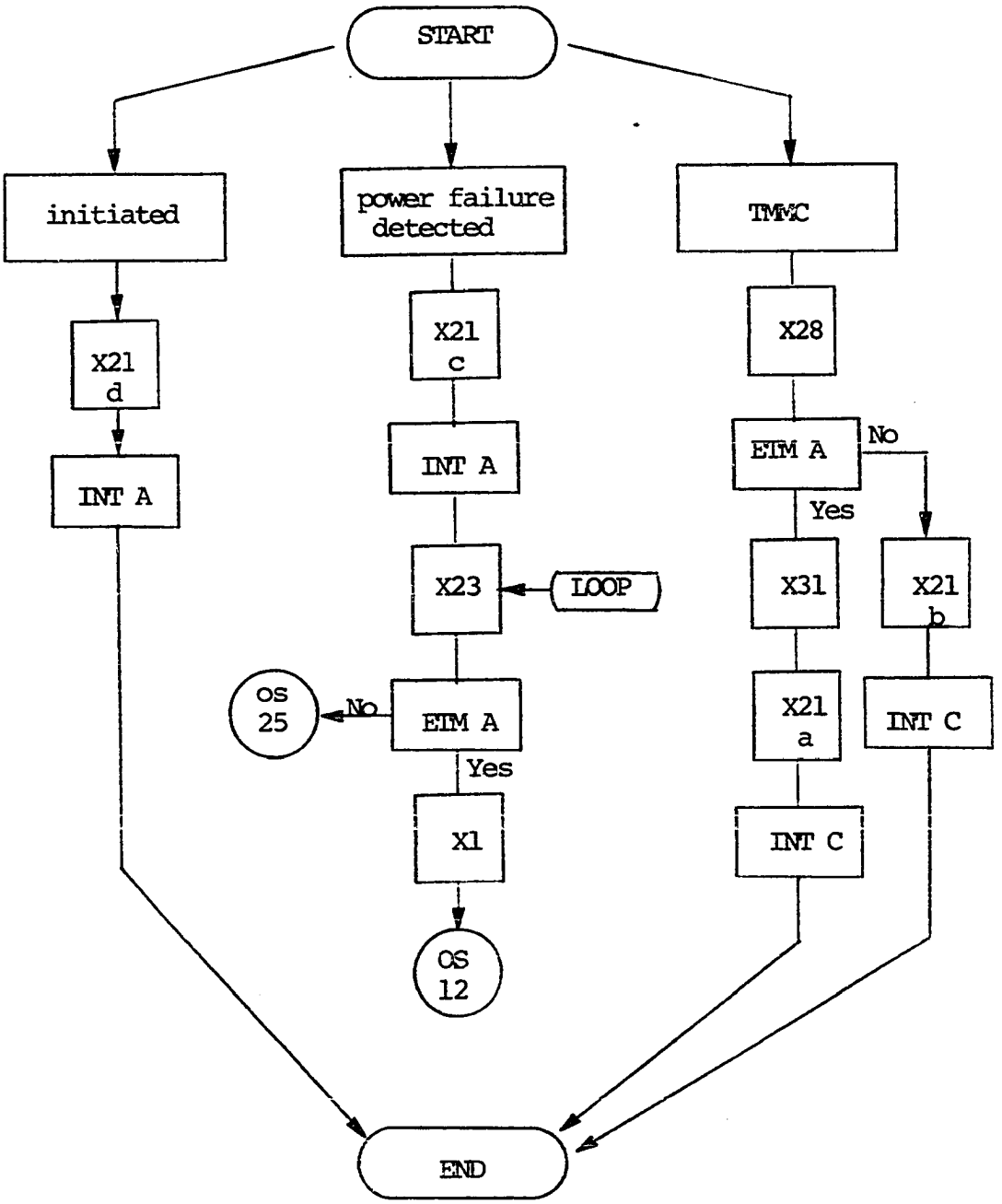


Figure E1. Start-up, shut-down and TMMC procedures.

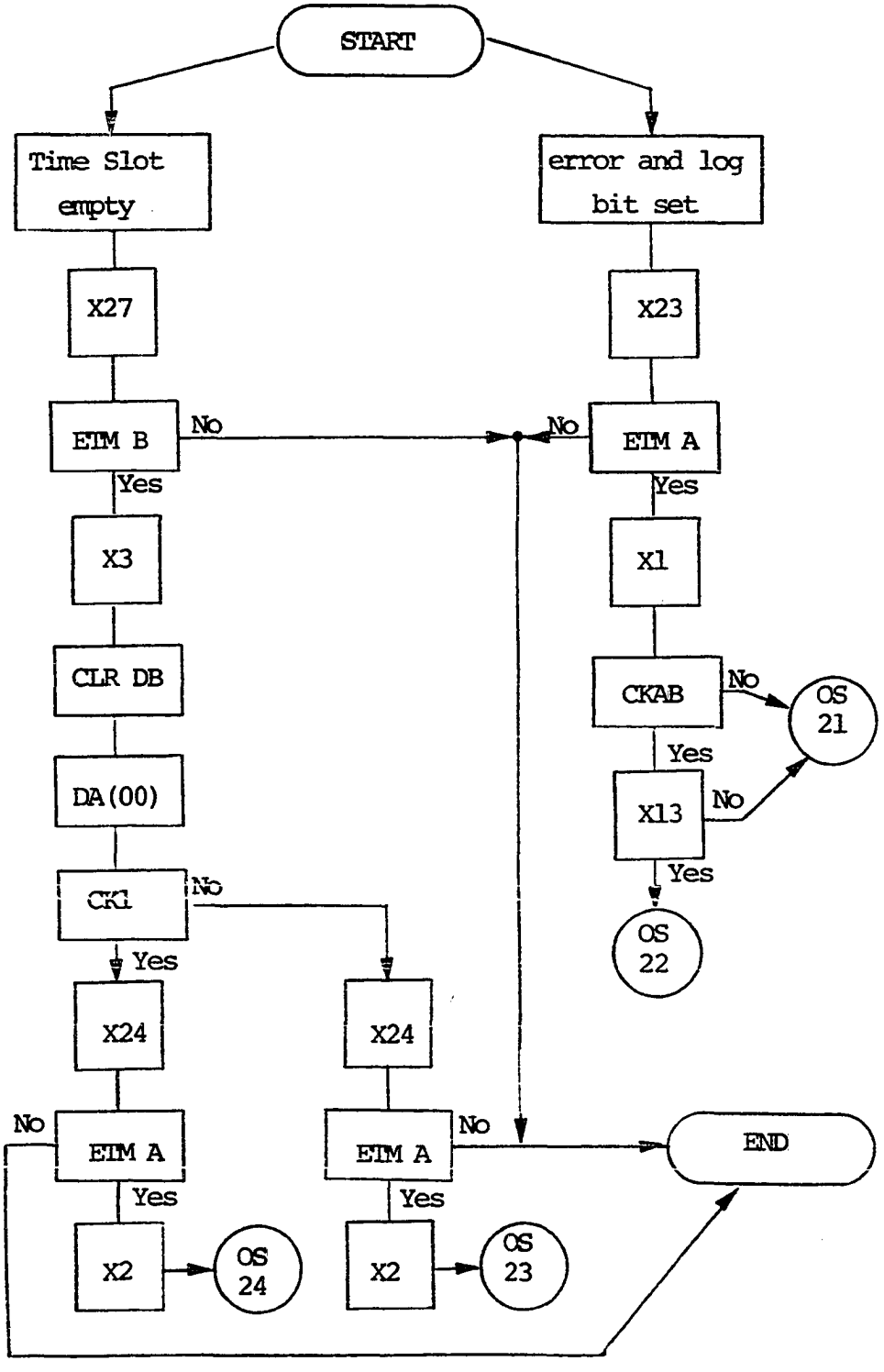


Figure E2. Data transmission and error procedures.

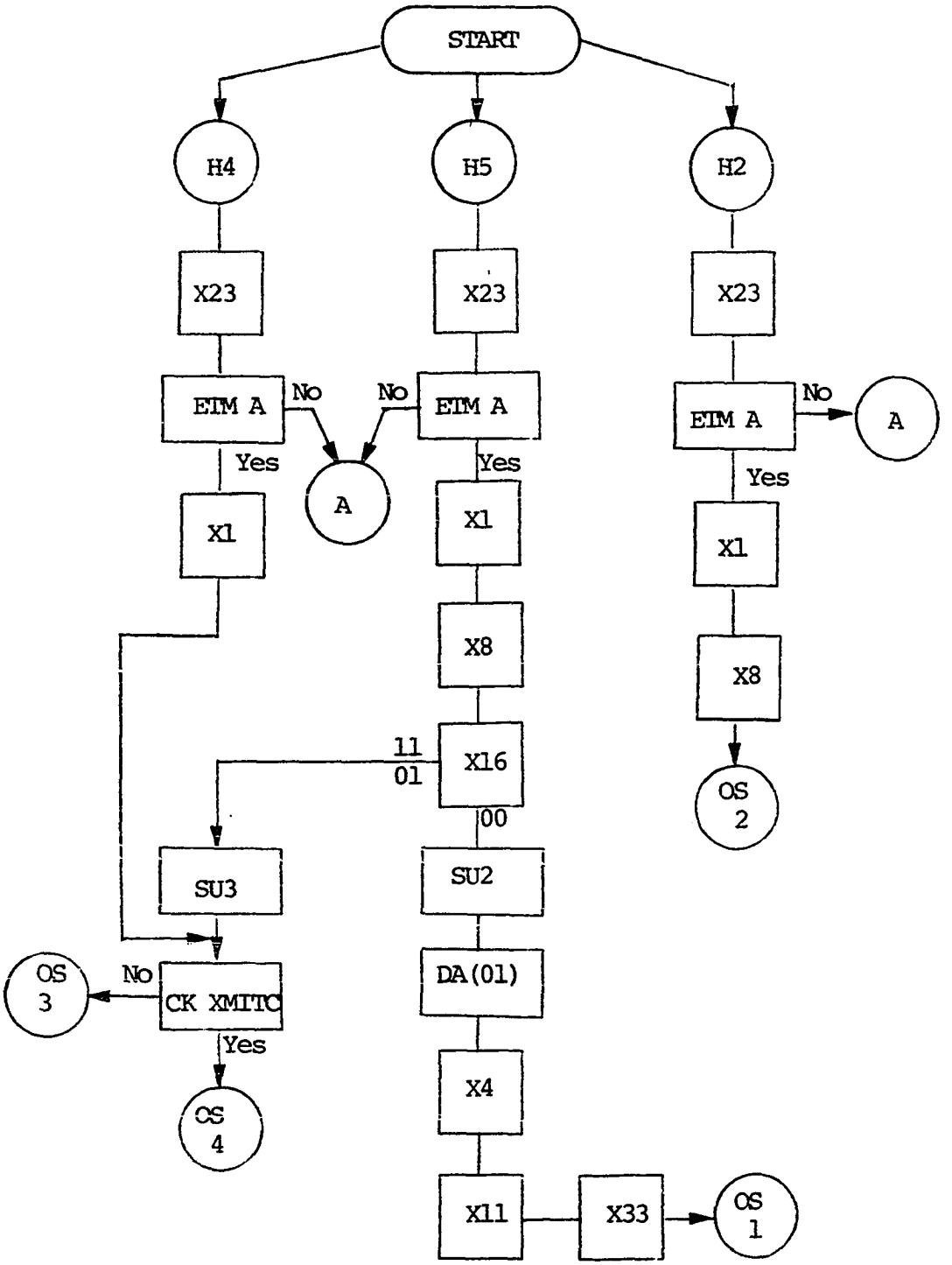


Figure E3. Broadcasting message.

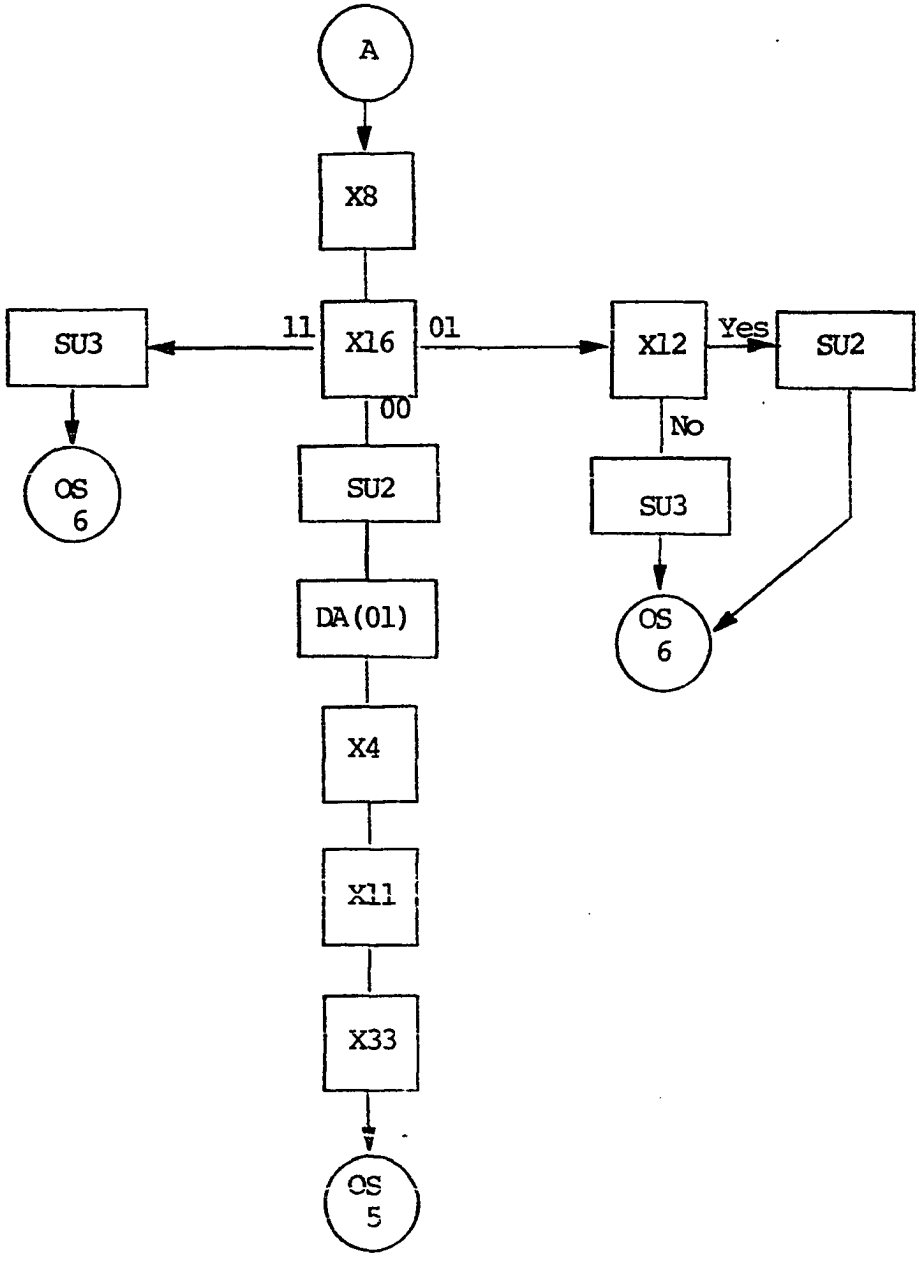


Figure E3. (continued)

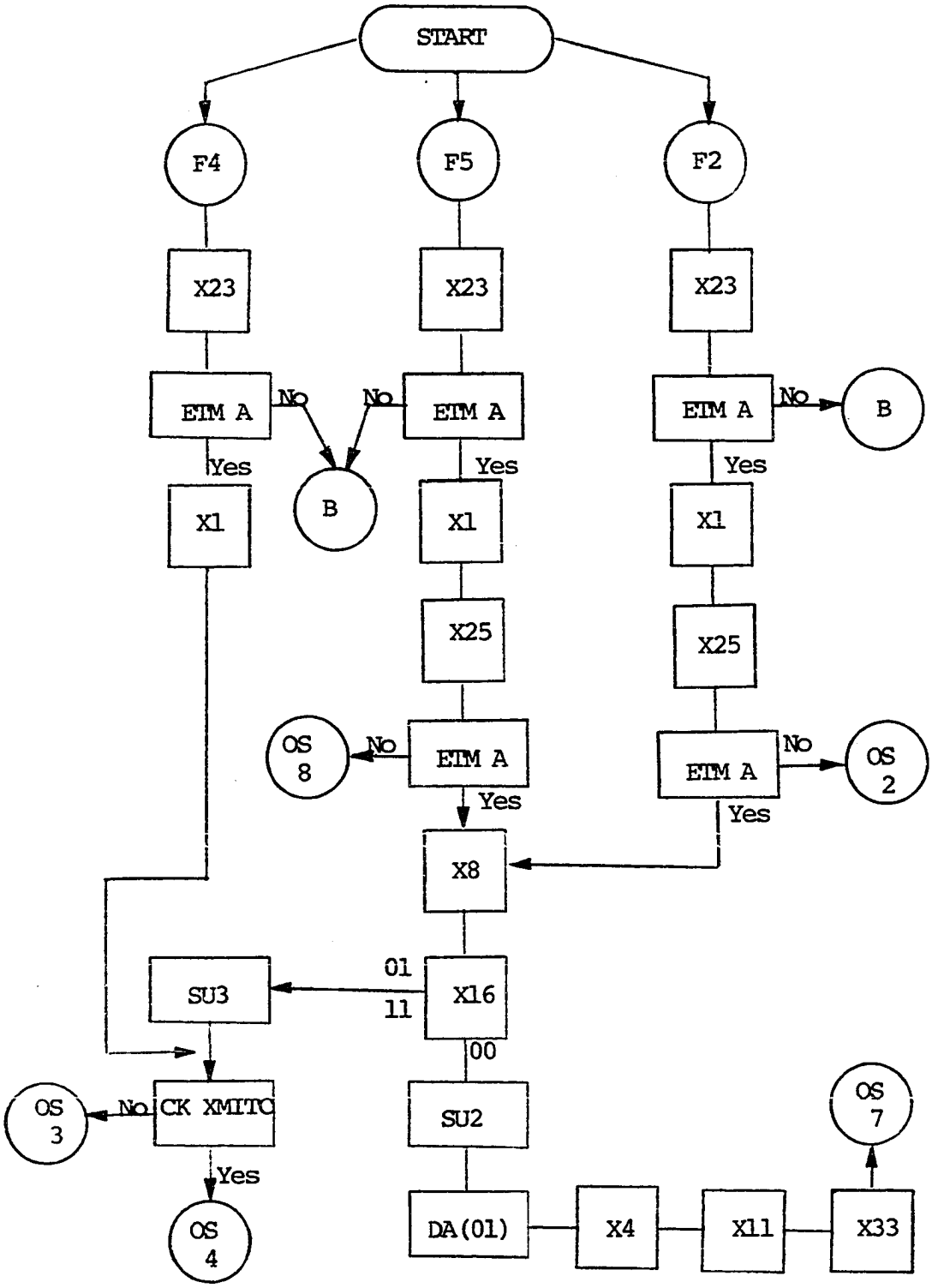


Figure E4. System message to a process.

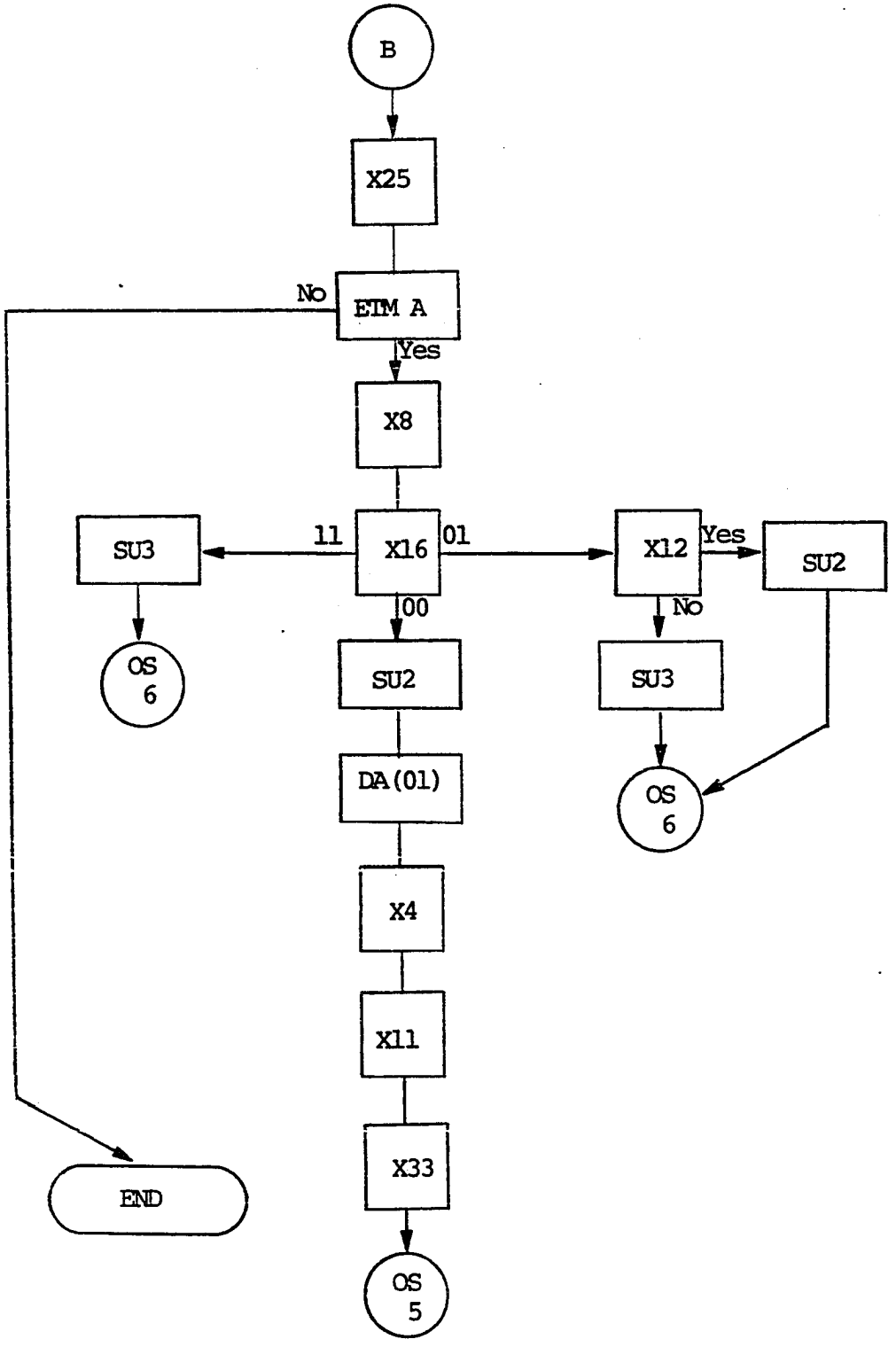


Figure E4. (continued)

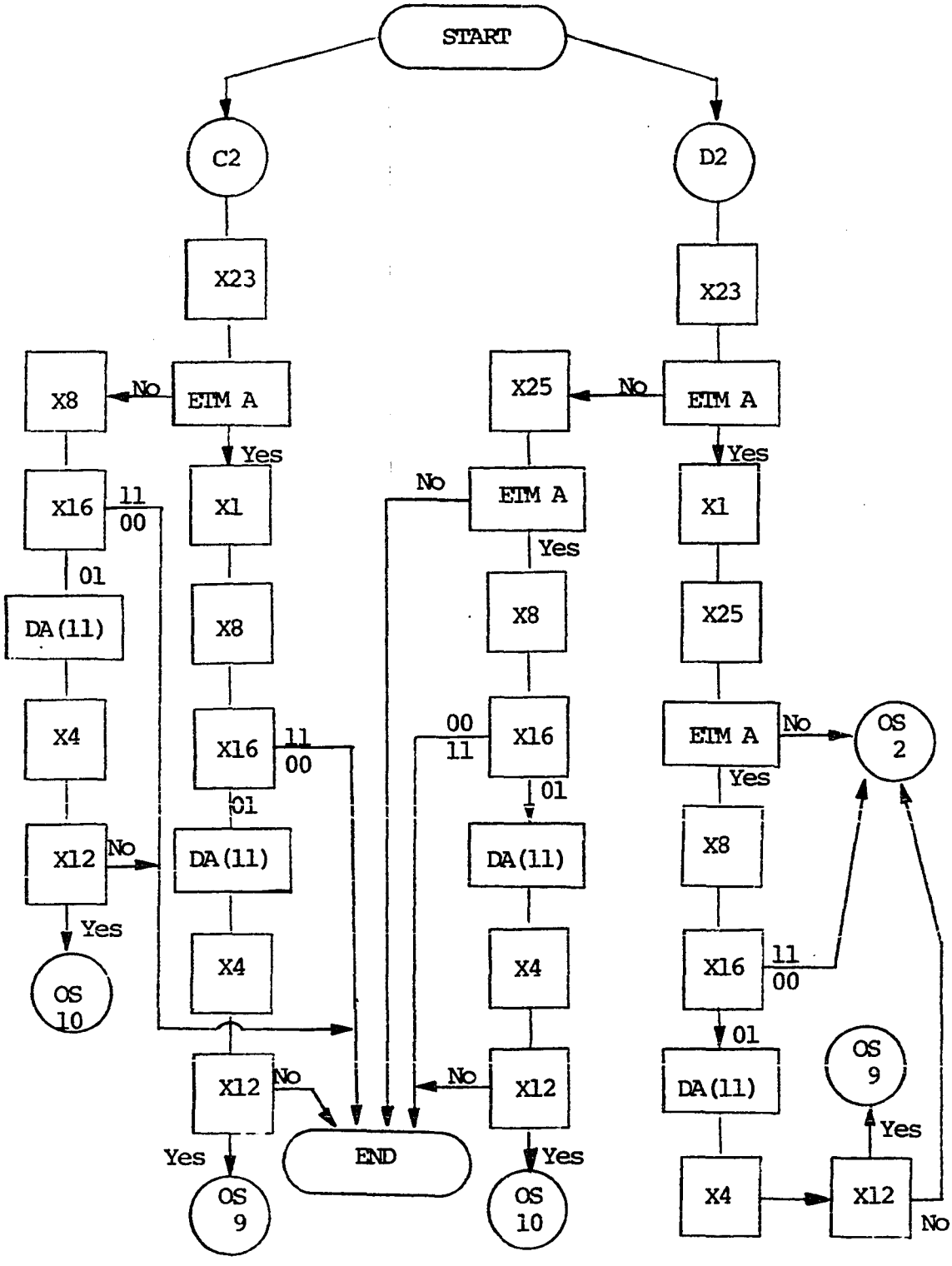


Figure E5. Special messages.

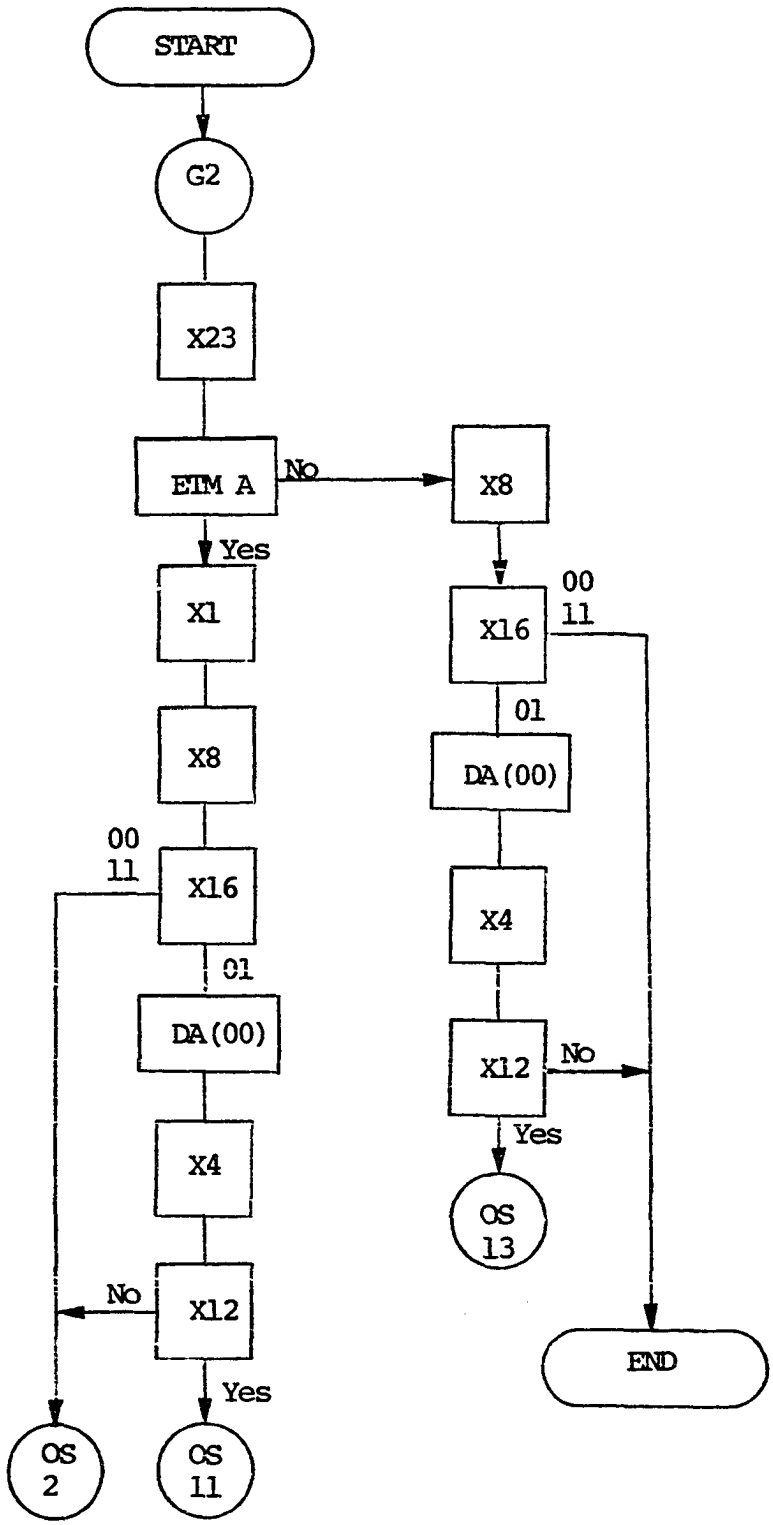


Figure E5. (continued)

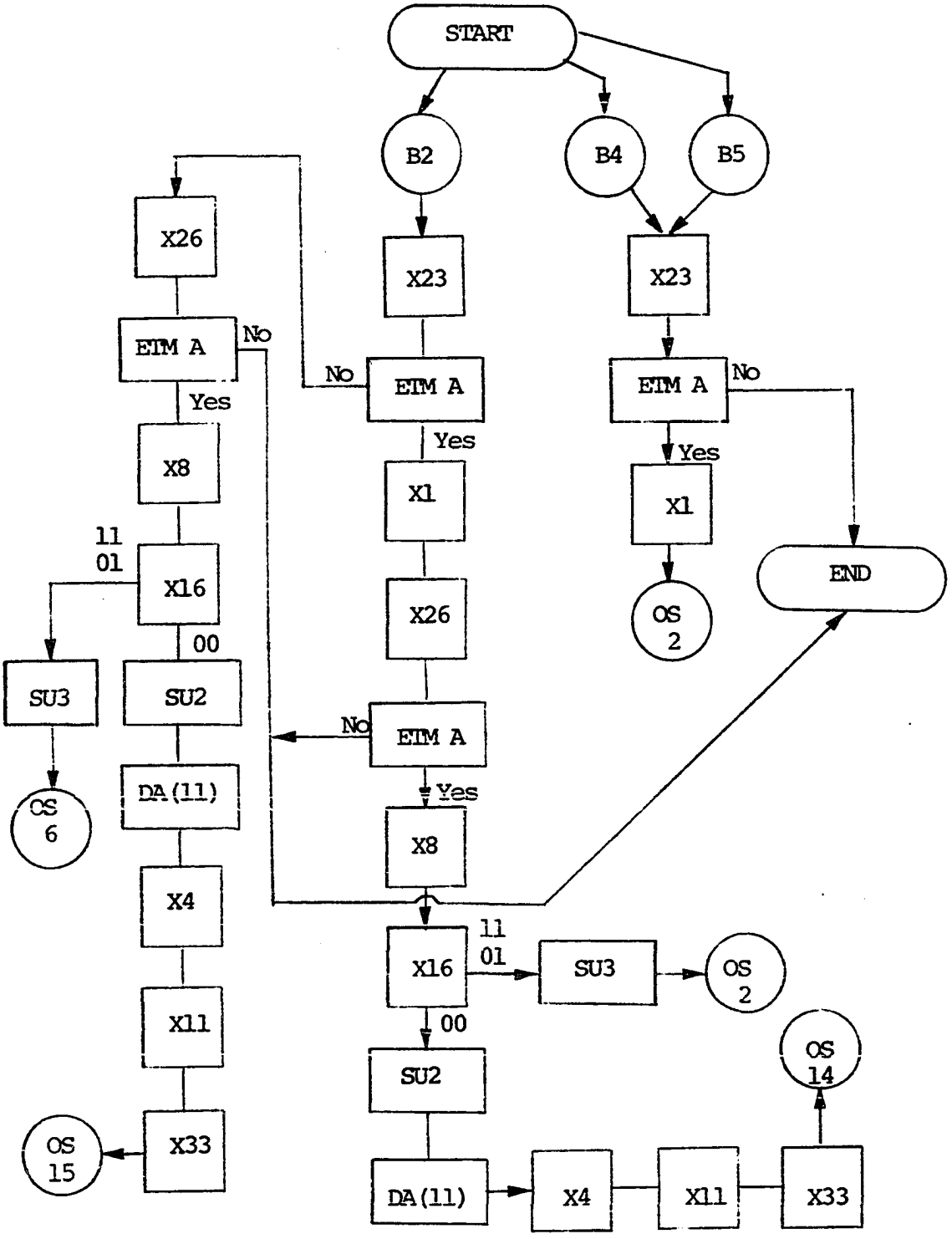


Figure E6. System message tc a device address.

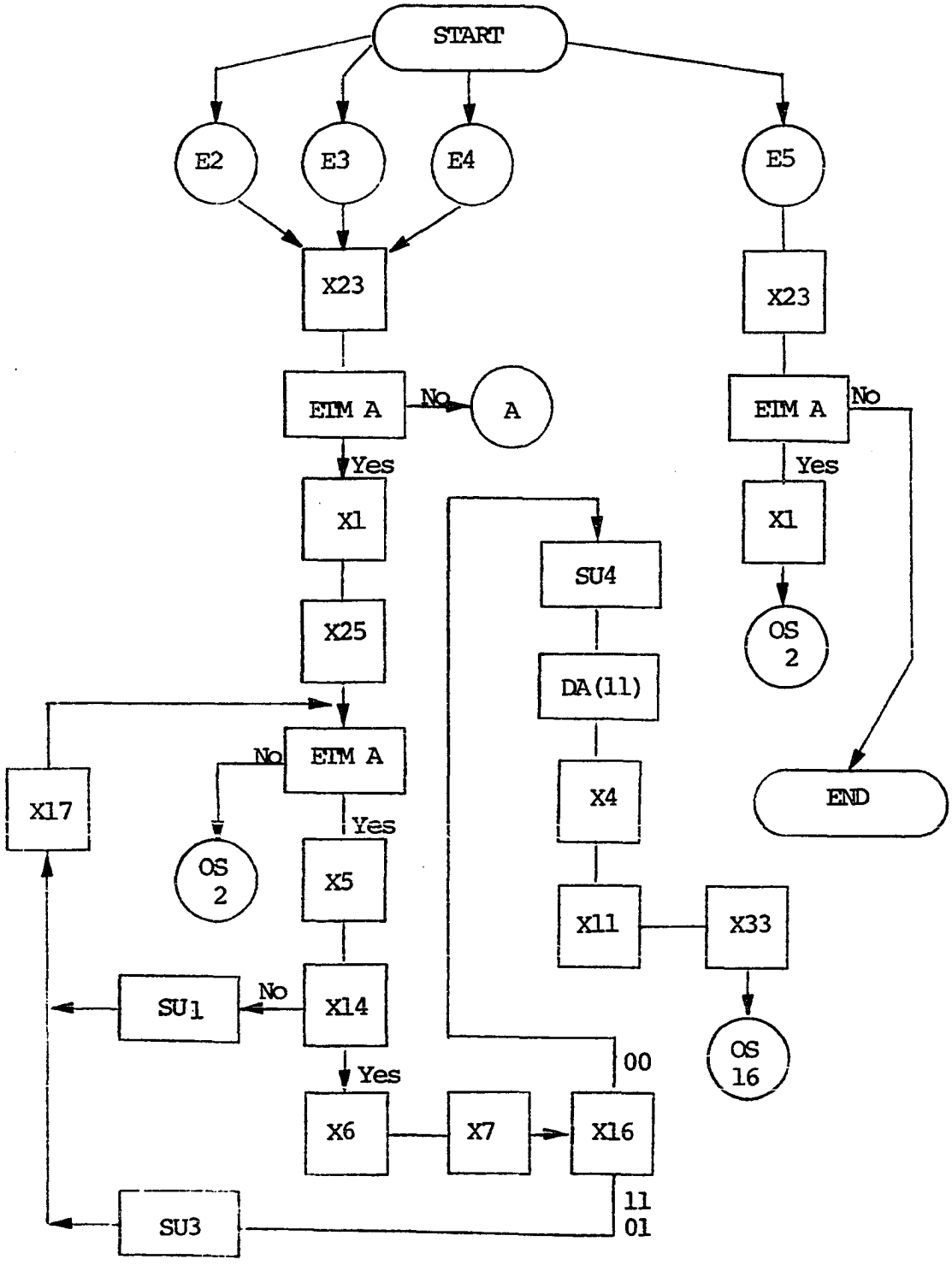


Figure E7. Regular message to a process.

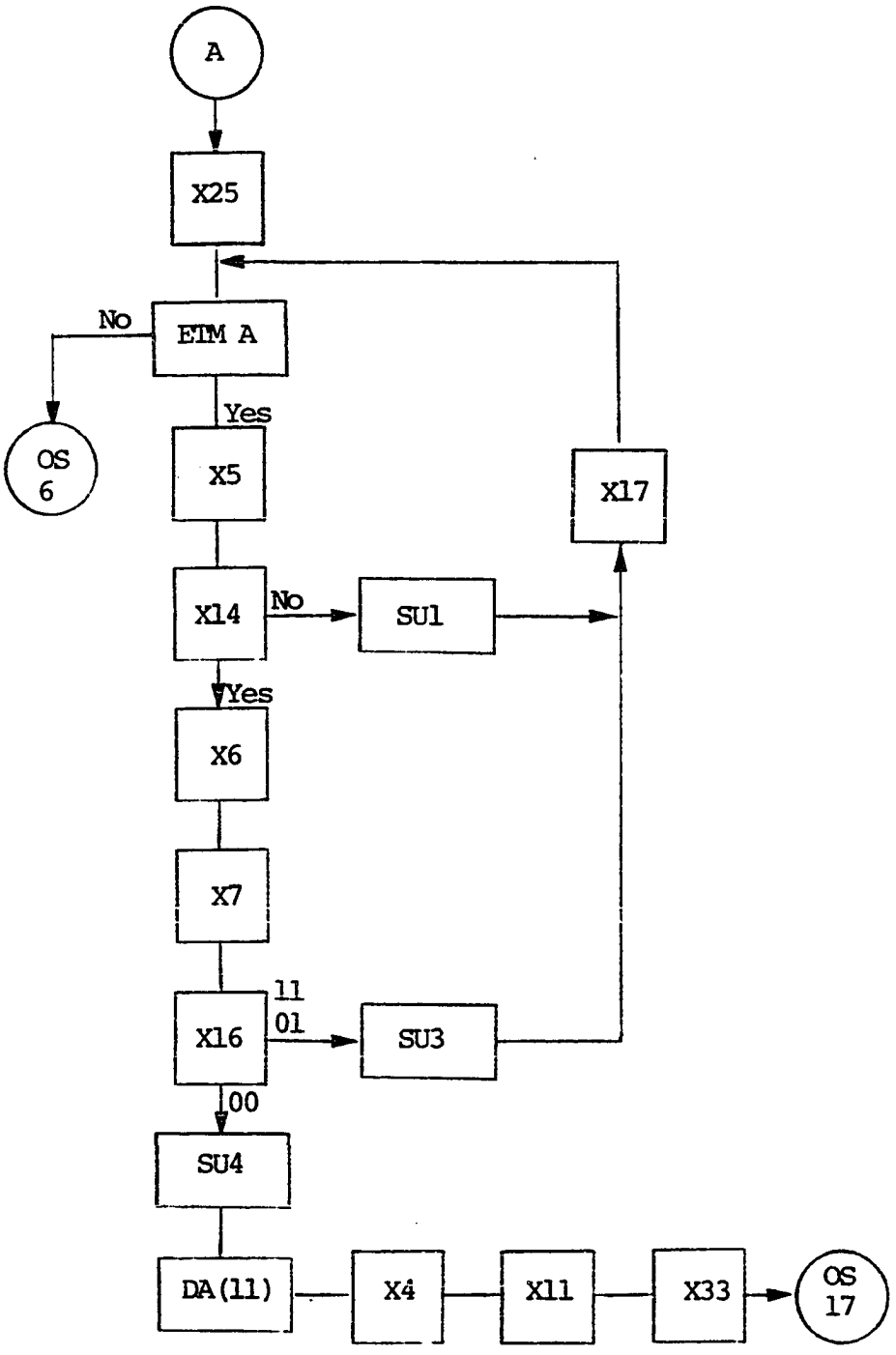


Figure E7. (continued)

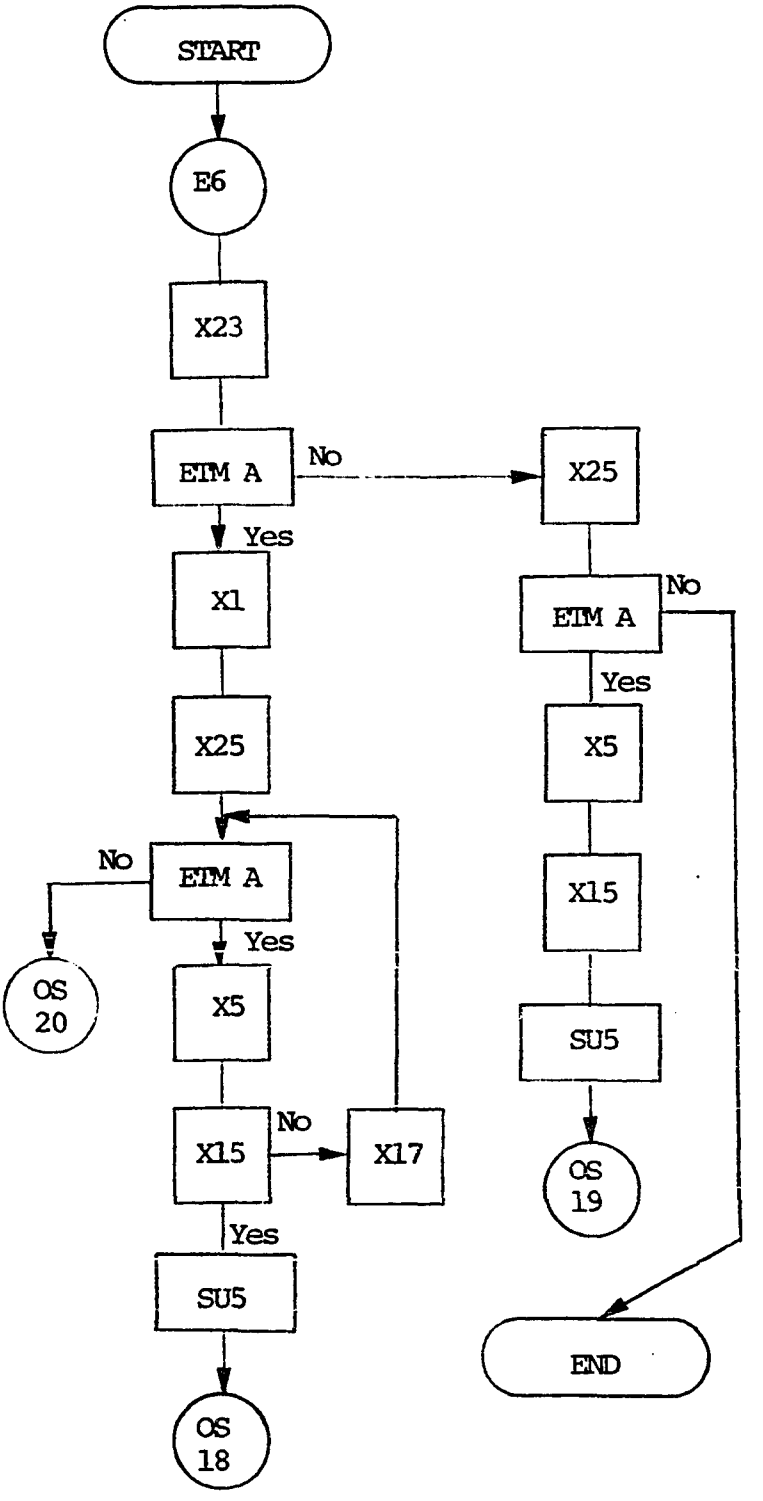


Figure E8. Retransmission of a regular message to a process.

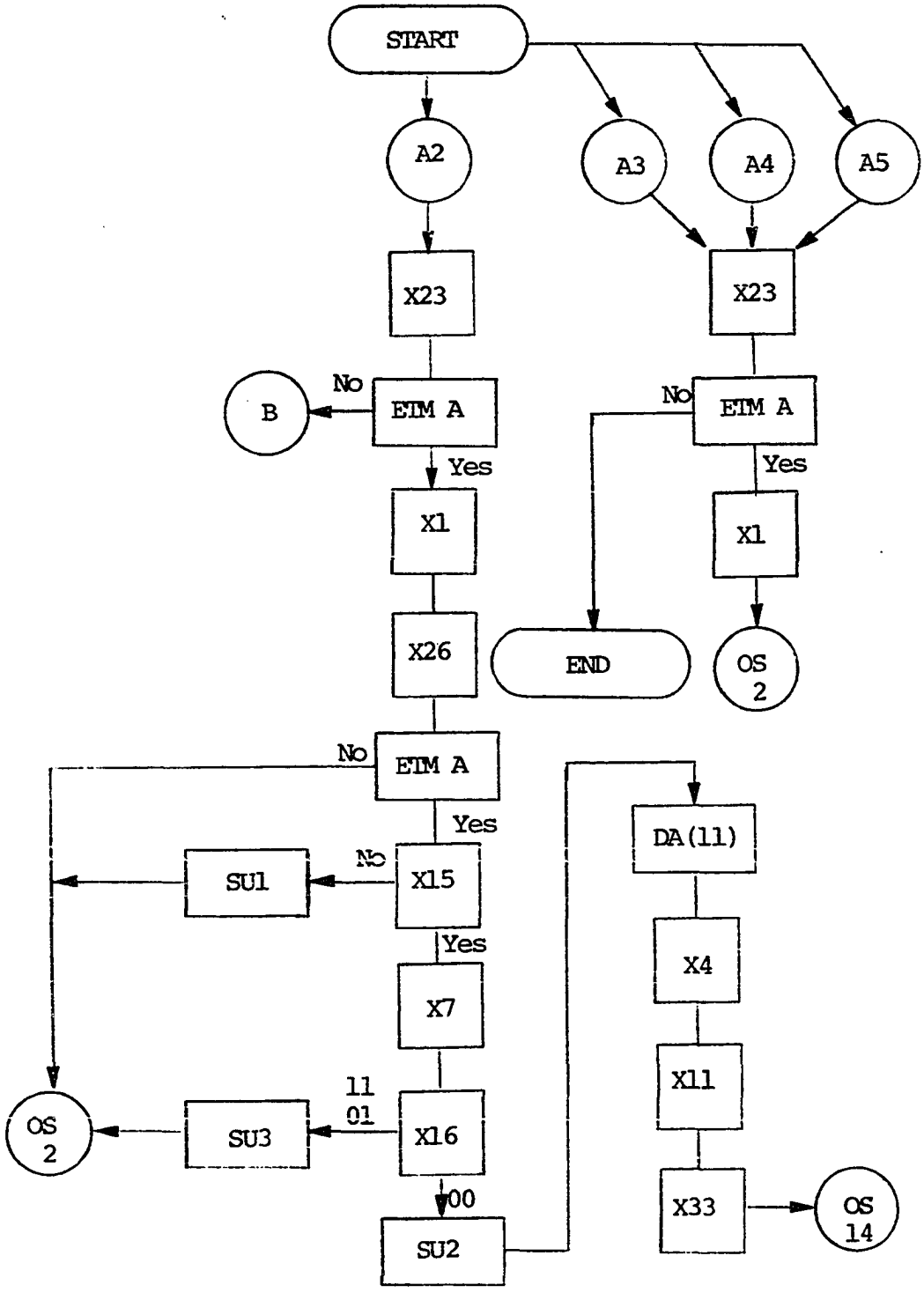


Figure E9. Regular message to a device address.

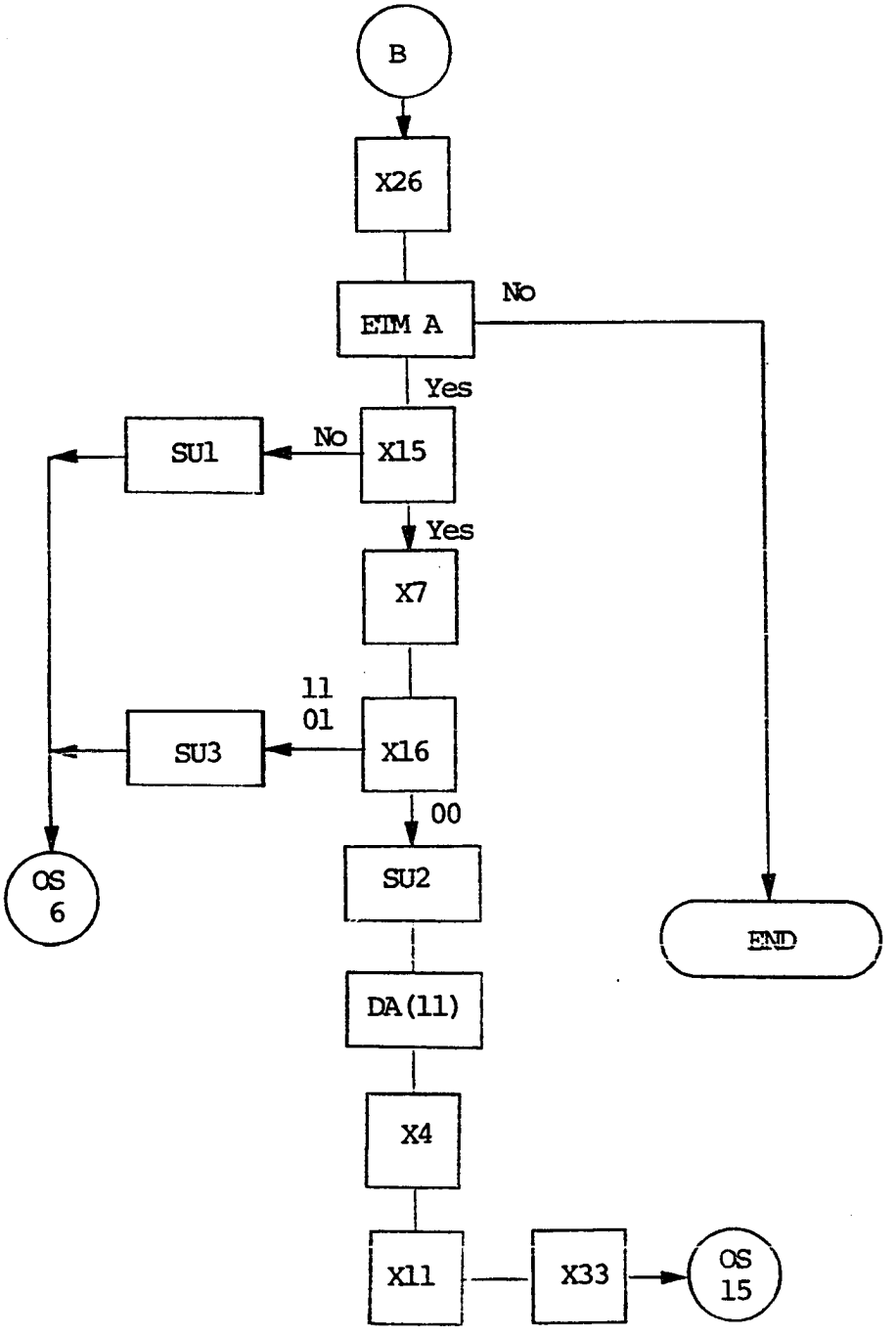


Figure E9. (continued)

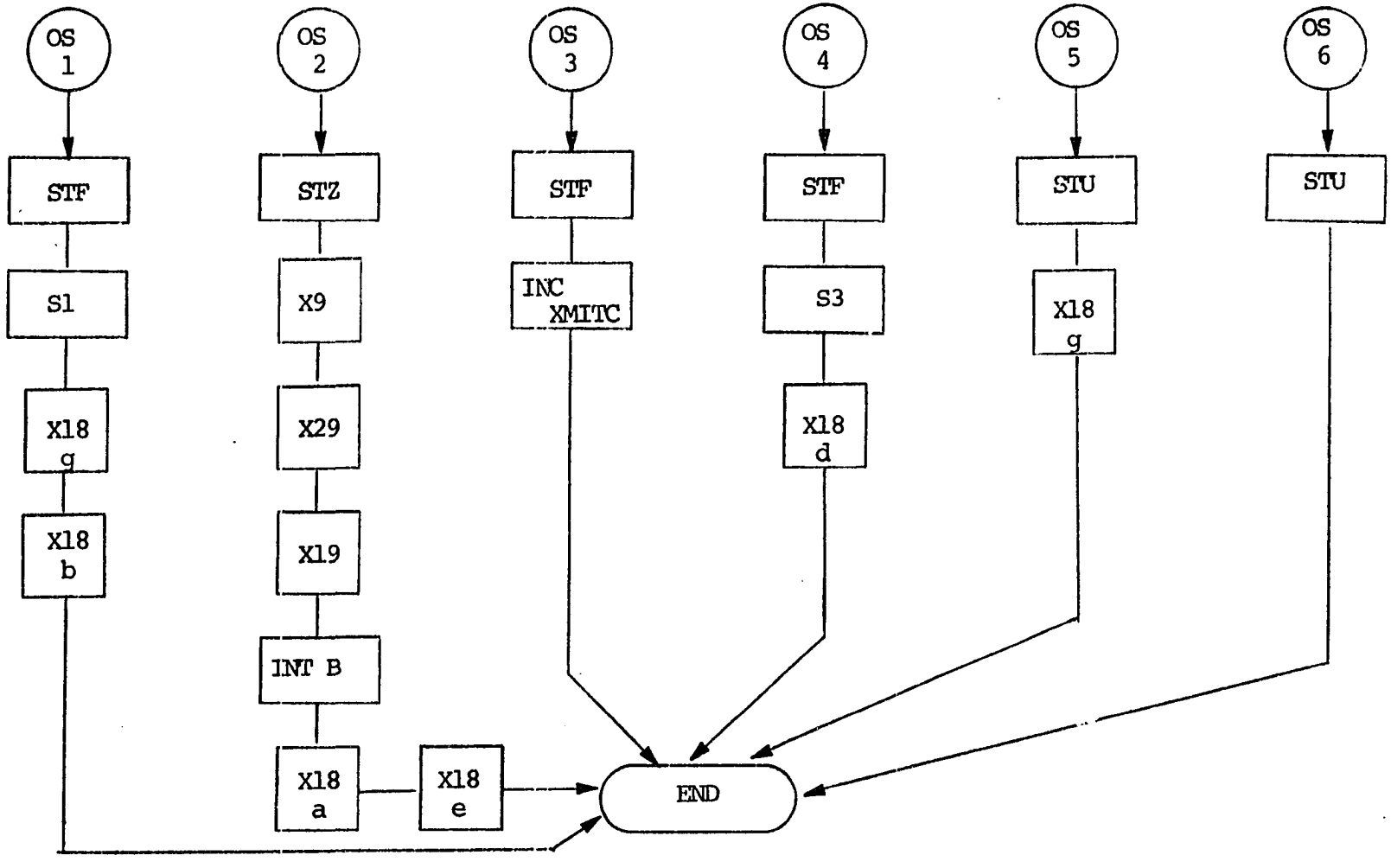


Figure E10. Continuing procedures after no transmission errors are detected.

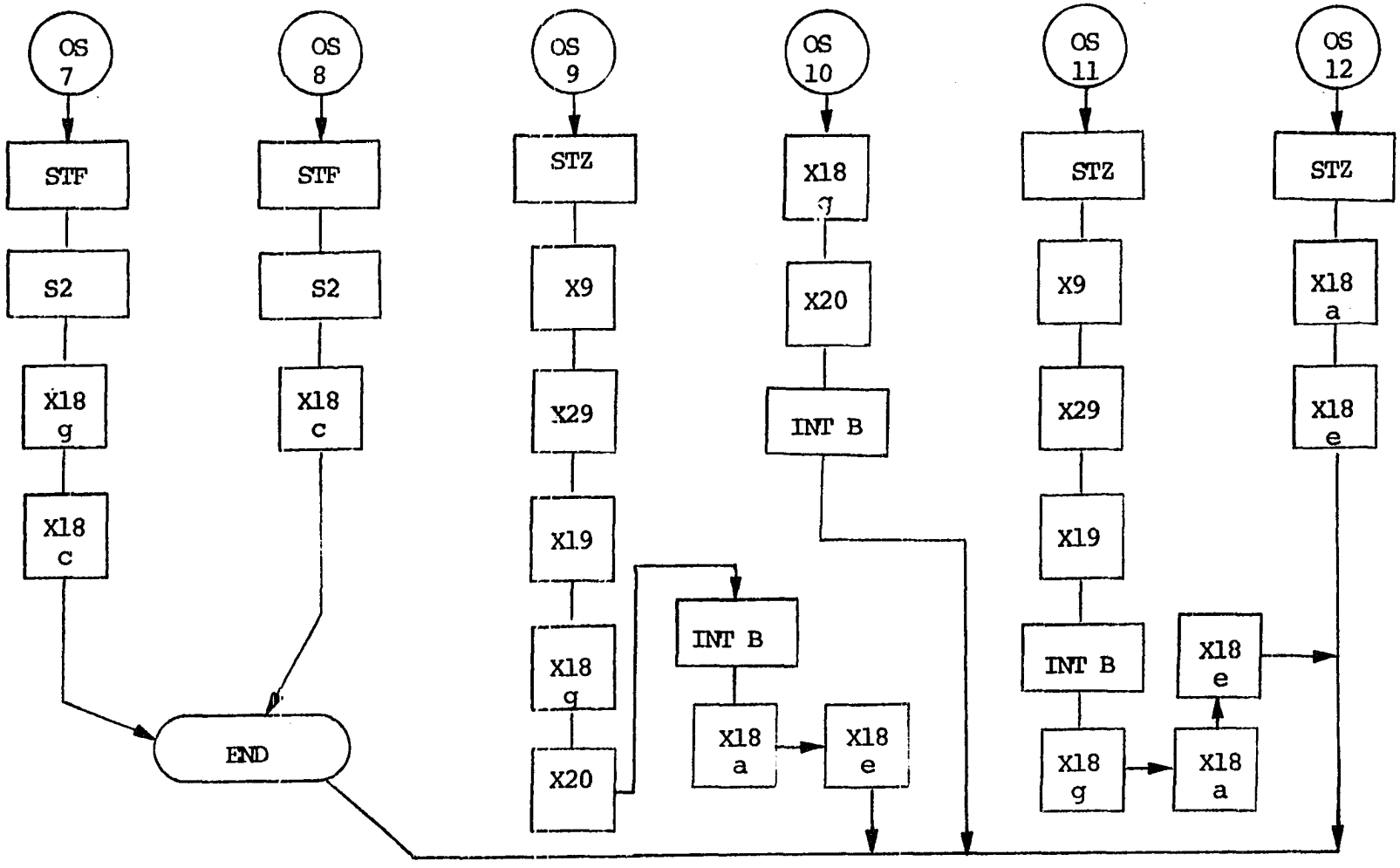


Figure E10. (continued)

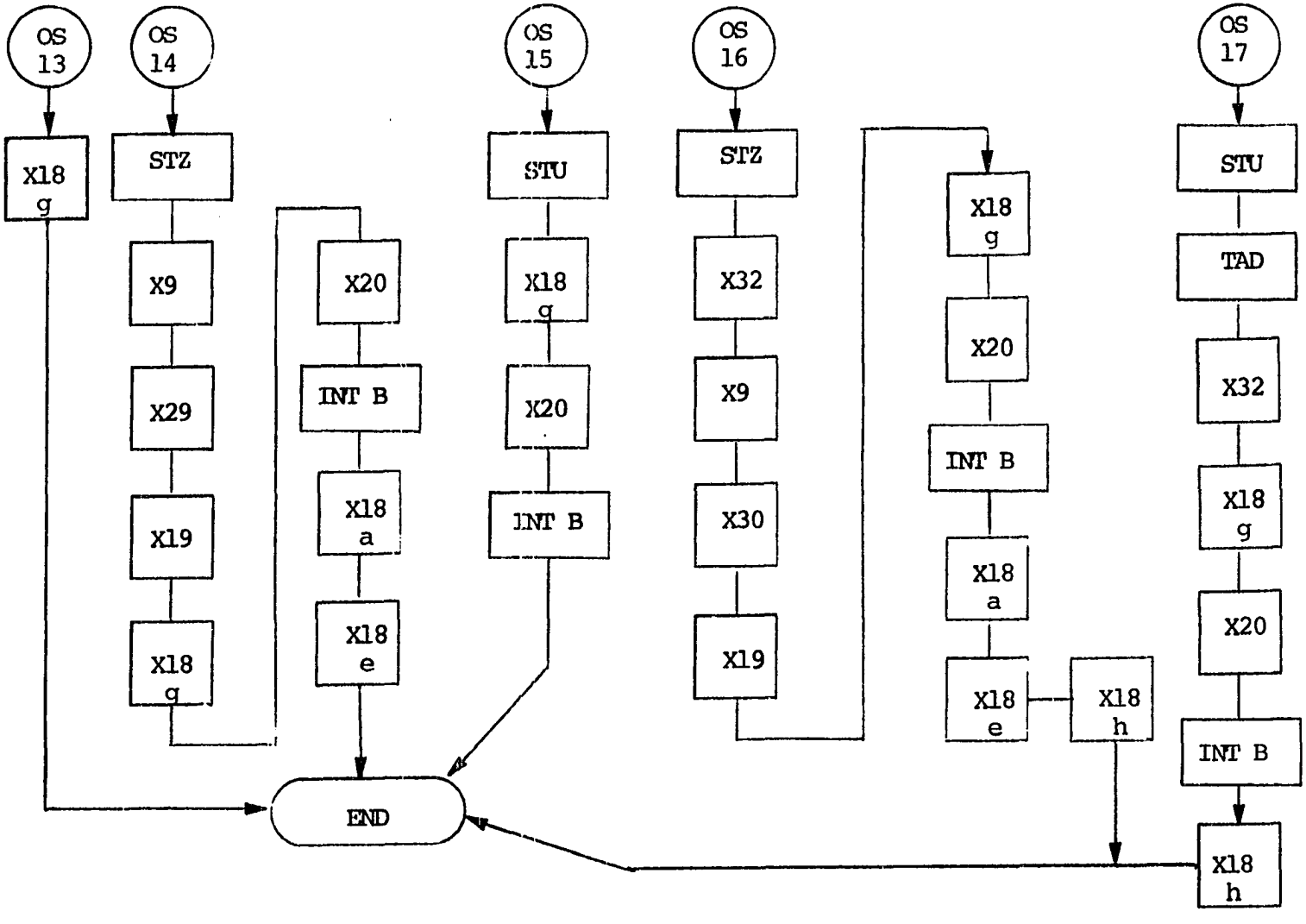


Figure E10. (continued)

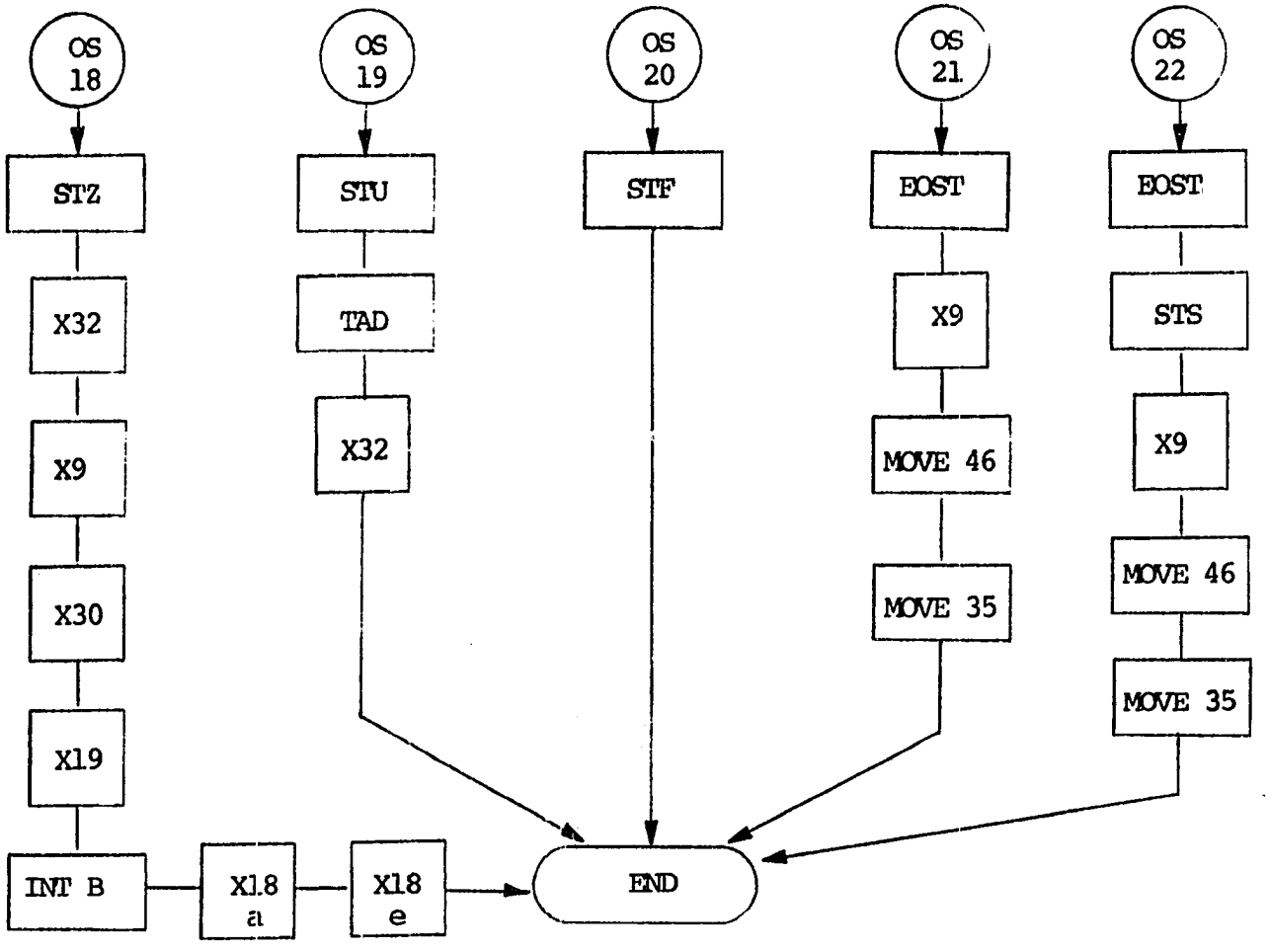


Figure E10. (continued)

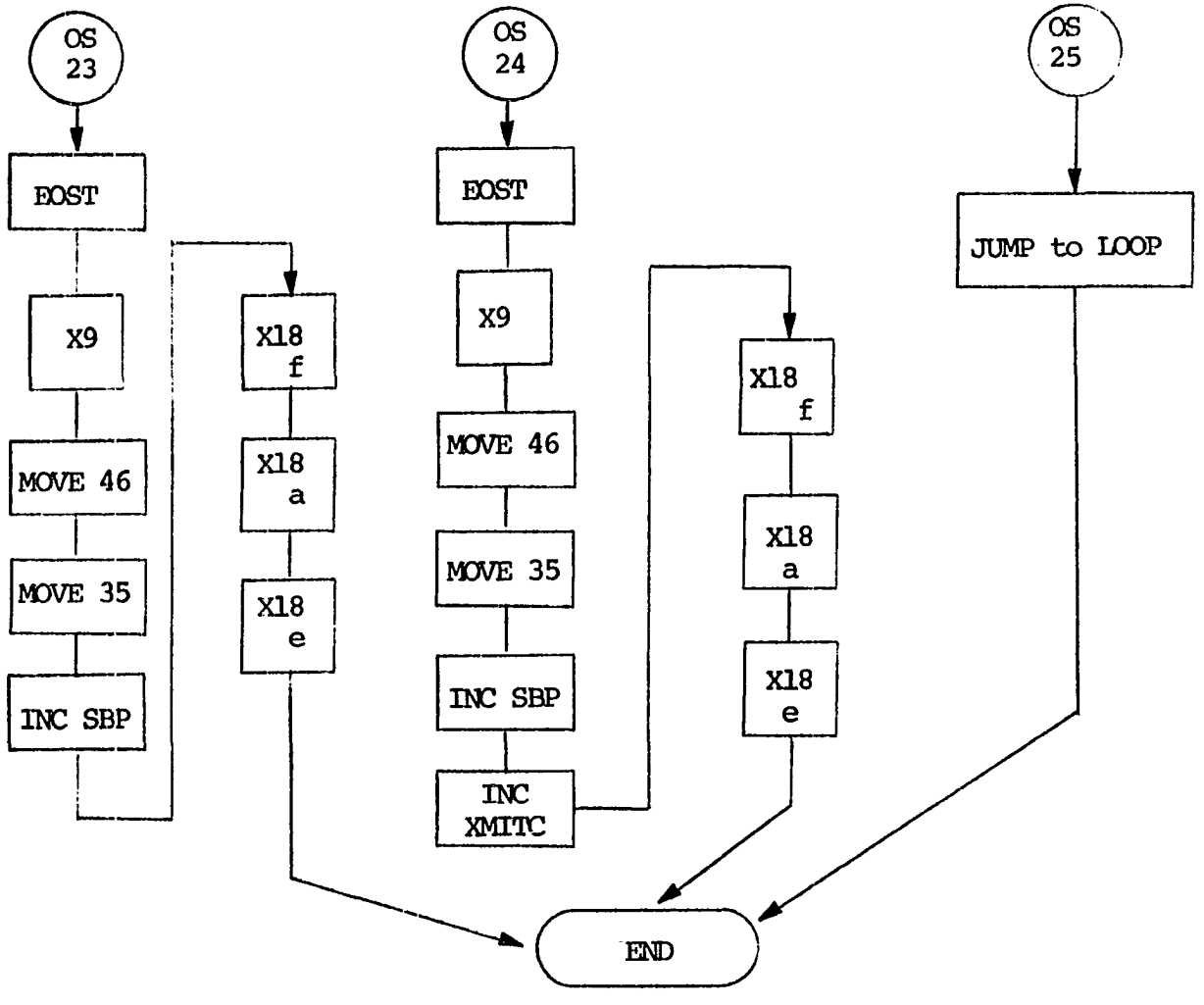


Figure E10. (continued)

APPENDIX F. CONTROL PROGRAMS

The control programs loaded into two separate programmable read only memories are obtained following the simplified flowcharts in Appendix E and the instructions described in Appendix B. These sequences of instructions govern the Table Matcher operations as well as all the major interface operations described earlier. Two different parts, one for procedures and the other for basic subroutines, are shown separately.

Procedures

0	JMP START	100000011000
1	JMP SD	100000011011
2	JMP TMMC	100000100010
3	JMP H5A	100000101100
4	JMP H4A	100000111110
5	JMP H2A	100001000011
6	JMP F5A	100001011011
7	JMP F4A	100001101100
8	JMP F2A	100001110001
9	JMP C2A	100001111101
10	JMP D2A	100010010101
11	JMP G2A	100010101010
12	JMP B54A	100011000010
13	JMP B2A	100011000111
14	JMP E5A	100011101001
15	JMP E234A	100011101110
16	JMP E6A	100100100000
17	JMP A2A	100100110110
18	JMP A345A	100101100000
19	JMP XMIT	100101100101
20	JMP ERROR	100101110111
21		
22		
23		

24	START	CALL X21D	010101101000
25		INT A	000001010000
26		HALT	000000000000
27	SD	CALL X21C	010101010010
28		INT A	000001010000
29	LOOP	CALL X23	010101110110
30		ETM A	000000010000
31		WJMP OS25	110111111010
32		CALL X1	010000000000
33		WJMP OS12	110110101111
34	TMMC	CALL X28	010110101010
35		ETM A	000000010000
36		JMP A9	100000101001
37		CALL X31	010111001000
38		CALL X21A	010101000100
39		INT C	000001010100
40		HALT	000000000000
41	A9	CALL X21B	010101001010
42		INT C	000001010100
43		HALT	000000000000
44	H5A	CALL X23	010101110110
45		ETM A	000000010000
46		JMP A1	100001001001
47		CALL X1	010000000000

48	CALL X8	010001011100
49	CALL X16	010011001100
50	JMP B1	100000111010
51	JMP B1	100000111010
52	SU2	000111100010
53	DA (01)	000111000010
54	CALL X4	010000111100
55	CALL X11	010001111000
56	CALL X33	010111010010
57	WJMP OS1	110110000010
58	B1 SU3	000111100100
59	C1 CK XMITC	000111111000
60	WJMP OS3	110110001100
61	WJMP OS4	110110001111
62	H4A CALL X23	010101110110
63	ETM A	000000010000
64	JMP A1	100001001001
65	CALL X1	010000000000
66	JMP C1	100000111011
67	H2A CALL X23	010101110110
68	ETM A	000000010000
69	JMP A1	100001001001
70	CALL X1	010000000000
71	CALL X8	010001011100

72		WJMP OS2	110110000101
73	A1	CALL X8	010001011100
74		CALL X16	010011001100
75		JMP D1	100001010011
76		JMP E1	100001010101
77		SU2	000111100010
78		DA (01)	000111000010
79		CALL X4	010000111100
80		CALL X11	010001111000
81		CALL X33	010111010010
82		WJMP OS5	110110010001
83	D1	SU3	000111100100
84		WJMP OS6	110110010011
85	E1	CALL X12	010010001100
86		JMP F1	100001011001
87		SU2	000111100010
88		WJMP OS6	110110010011
89	F1	SU3	000111100100
90		WJMP OS6	110110010011
91	F5A	CALL X23	010101110110
92		ETM A	000000010000
93		JMP A2	100001111001
94		CALL X1	010000000000
95		CALL X25	010110001010

96		ETM A	000000010000
97		WJMP OS8	110110011000
98	B2	CALL X8	010001011100
99		CALL X16	010011001100
100		JMP B1	100000111010
101		JMP B1	100000111010
102		SU2	000111100010
103		DA (01)	000111000010
104		CALL X4	010000111100
105		CALL X11	010001111000
106		CALL X33	010111010010
107		WJMP OS7	110110010101
108	F4A	CALL X23	010101110110
109		ETM A	000000010000
110		JMP A2	100001111001
111		CALL X1	010000000000
112		JMP C1	100000111011
113	F2A	CALL X23	010101110110
114		ETM A	000000010000
115		JMP A2	100001111001
116		CALL X1	010000000000
117		CALL X25	010110001010
118		ETM A	000000010000
119		WJMP OS2	110110000101

120	JMP B2	100001100010
121	A2 CALL X25	010110001010
122	ETM A	000000010000
123	HALT	000000000000
124	JMP A1	100001001001
125	C2A CALL X23	010101110110
126	ETM A	000000010000
127	JMP A3	100010001011
128	CALL X1	010000000000
129	CALL X8	010001011100
130	CALL X16	010011001100
131	HALT	000000000000
132	JMP B3	100010000110
133	HALT	000000000000
134	B3 DA (11)	000111000100
135	CALL X4	010000111100
136	CALL X12	010010001100
137	HALT	000000000000
138	WJMP OS9	110110011010
139	A3 CALL X8	010001011100
140	CALL X16	010011001100
141	HALT	000000000000
142	JMP C3	100010001111
143	HALT	000000000000

144		C3 DA (11)	000111000100
145		CALL X4	010000111100
146		CALL X12	010010001100
147		HALT	000000000000
148		WJMP OS10	110110100011
149	D2A	CALL X23	010101110110
150		ETM A	000000010000
151		JMP S1	100010100110
152		CALL X1	010000000000
153		CALL X25	010110001010
154		ETM A	000000010000
155		WJMP OS2	110110000101
156		CALL X8	010001011100
157		CALL X16	010011001100
158		WJMP OS2	110110000101
159		JMP D3	100010100001
160		WJMP OS2	110110000101
161	D3	DA (11)	000111000100
162		CALL X4	010000111100
163		CALL X12	010010001100
164		WJMP OS2	110110000101
165		WJMP OS9	110110011010
166	S1	CALL X25	010110001010
167		ETM A	000000010000

```
168          HALT          000000000000
169          JMP A3        100010001011
170      G2A CALL X23      010101110110
171          ETM A         000000010000
172          JMP E3        100010111000
173          CALL X1       010000000000
174          CALL X8       010001011100
175          CALL X16      010011001100
176          WJMP OS2      110110000101
177          JMP F3        100010110011
178          WJMP OS2      110110000101
179       F3 DA (00)       000111000000
180          CALL X4       010000111100
181          CALL X12      010010001100
182          WJMP OS2      110110000101
183          WJMP OS11     110110100111
184       E3 CALL X8       010001011100
185          CALL X16      010011001100
186          HALT          000000000000
187          JMP G3        100010111101
188          HALT          000000000000
189       G3 DA (00)       000111000000
190          CALL X4       010000111100
191          CALL X12      010010001100
```

192		HALT	000000000000
193		WJMP OS13	110110110011
194	B54A	CALL X23	010101110110
195		ETM A	000000010000
196		HALT	000000000000
197		CALL X1	010000000000
198		WJMP OS2	110110000101
199	B2A	CALL X23	010101110110
200		ETM A	000000010000
201		JMP A4	100011011010
202		CALL X1	010000000000
203		CALL X26	010110010010
204		ETM A	000000010000
205		HALT	000000000000
206		CALL X8	010001011100
207		CALL X16	010011001100
208		JMP B4	100011011000
209		JMP B4	100011011000
210		SU2	000111100010
211		DA (11)	000111000100
212		CALL X4	010000111100
213		CALL X11	010001111000
214		CALL X33	010111010010
215		WJMP OS14	110110110100

216	B4	SU3	000111100100
217		WJMP OS2	110110000101
218	A4	CALL X26	010110010010
219		ETM A	000000010000
220		HALT	000000000000
221		CALL X8	010001011100
222		CALL X16	010011001100
223		JMP C4	100011100111
224		JMP C4	100011100111
225		SU2	000111100010
226		DA (11)	000111000100
227		CALL X4	010000111100
228		CALL X11	010001111000
229		CALL X33	010111010010
230		WJMP OS15	110110111101
231	C4	SU3	000111100100
232		WJMP OS6	110110010011
233	E5A	CALL X23	010101110110
234		ETM A	000000010000
235		HALT	000000000000
236		CALL X1	010000000000
237		WJMP OS2	110110000101
238	E234A	CALL X23	010101110110
239		ETM A	000000010000

```
240          JMP A5                  100100001001
241          CALL X1                 010000000000
242          CALL X25                010110001010
243       D5 ETM A                   000000010000
244          WJMP OS2                110110000101
245          CALL X5                 010001000010
246          CALL X14                010011000000
247          JMP B5                  100100000011
248          CALL X6                 010001001110
249          CALL X7                 010001011000
250          CALL X16                010011001100
251          JMP C5                  100100000110
252          JMP C5                  100100000110
253          SU4                     000111100110
254          DA (11)                 000111000100
255          CALL X4                 010000111100
256          CALL X11                010001111000
257          CALL X33                010111010010
258          WJMP OS16               110111000010
259       B5 SU1                     000111100000
260          CALL X17                010011010110
261          JMP D5                  100011110011
262       C5 SU3                     000111100100
263          CALL X17                010011010110
```

```
264          JMP D5          100011110011
265       A5 CALL X25        010110001010
266       G5 ETM A           000000010000
267          WJMP OS6        110110010011
268          CALL X5         010001000010
269          CALL X14        010011000000
270          JMP E5          100100011010
271          CALL X6         010001001110
272          CALL X7         010001011000
273          CALL X16        010011001100
274          JMP F5          100100011101
275          JMP F5          100100011101
276          SU4             000111100110
277          DA (11)         000111000100
278          CALL X4         010000111100
279          CALL X11        010001111000
280          CALL X33        010111010010
281          WJMP OS17       110111001101
282       E5 SU1             000111100000
283          CALL X17        010011010110
284          JMP G5          100100001010
285       F5 SU3             000111100100
286          CALL X17        010011010110
287          JMP G5          100100001010
```

```
288          E6A CALL X23          010101110110
289              ETM A             000000010000
290              JMP H5            100100101110
291              CALL X1           010000000000
292              CALL X25          010110001010
293           J5 ETM A             000000010000
294              WJMP OS20         110111011101
295              CALL X5           010001000010
296              CALL X15          010011000110
297              JMP I5            100100101100
298              SU5               000111101000
299              WJMP OS18         110111010011
300           I5 CALL X17          010011010110
301              JMP J5            100100100101
302           H5 CALL X25          010110001010
303              ETM A             000000010000
304              HALT              000000000000
305              CALL X5           010001000010
306              CALL X15          010011000110
307              HALT              000000000000
308              SU5               000111101000
309              WJMP OS19         110111011011
310          A2A CALL X23          010101110110
311              ETM A             000000010000
```

```
312          JMP A7          100101001101
313          CALL X1         010000000000
314          CALL X26        010110010010
315          ETM A           000000010000
316          WJMP OS2        110110000101
317          CALL X15        010011000110
318          JMP B7          100101001001
319          CALL X7         010001011000
320          CALL X16        010011001100
321          JMP C7          100101001011
322          JMP C7          100101001011
323          SU2             000111100010
324          DA (11)         000111000100
325          CALL X4         010000111100
326          CALL X11        010001111000
327          CALL X33        010111010010
328          WJMP OS14       110110110100
329       B7 SU1             000111100000
330          WJMP OS2        110110000101
331       C7 SU3             000111100100
332          WJMP OS2        110110000101
333       A7 CALL X26        010110010010
334          ETM A           000000010000
335          HALT            000000000000
```

336		CALL X15	010011000110
337		JMP D7	100101011100
338		CALL X7	010001011000
339		CALL X16	010011001100
340		JMP E7	100101011110
341		JMP E7	100101011110
342		SU2	000111100010
343		DA (11)	000111000100
344		CALL X4	010000111100
345		CALL X11	010001111000
346		CALL X33	010111010010
347		WJMP OS15	110110111101
348	D7	SU1	000111100000
349		WJMP OS6	110110010011
350	E7	SU3	000111100100
351		WJMP OS6	110110010011
352	A345A	CALL X23	010101110110
353		ETM A	000000010000
354		HALT	000000000000
355		CALL X1	010000000000
356		WJMP OS2	110110000101
357	XMIT	CALL X27	010110100100
358		ETM B	000000010010
359		HALT	000000000000

```
360              CALL X3          010000100100
361              CLR DB           000000100010
362              MOVE 80          001110000000
363              CK1              000111110000
364              JMP L1           100101110010
365              CALL X24         010110000010
366              ETM A            000000010000
367              HALT             000000000000
368              CALL X2          010000010010
369              WJMP OS24        110111110001
370            L1 CALL X24        010110000010
371              ETM A            000000010000
372              HALT             000000000000
373              CALL X2          010000010010
374              WJMP OS23        110111101001
375        ERROR CALL X23         010101110110
376              ETM A            000000010000
377              HALT             000000000000
378              CALL X1          010000000000
379              CK AB            000111110100
380              JMP A8           100110000000
381              CALL X13         010010100100
382              JMP A8           100110000000
383              WJMP OS22        110111100100
```

```
384          A8 WJMP OS21                  110111011111
385
386        OS1 OSC 82                      011100000100
387            CALL X18g                   010100011110
388            CALL X18b HT                010011110001
389        OS2 OSC 7                       011010000000
390            CALL X9                     010001100000
391            CALL X29                    010110110110
392            CALL X19                    010100110100
393            INT B                       000010100010
394            CALL X18a                   010011100010
395            CALL X18e HT                010100001111
396        OS3 OSC 8                       011100000000
397            INC XMITC                   000001000010
398            HALT                        000000000000
399        OS4 OSC 80                      011100000001
400            CALL X18d HT                010100000101
401        OS5 OSC 6                       011001000000
402            CALL X18g HT                010100011111
403        OS6 OSC 6                       011001000000
404            HALT                        000000000000
405        OS7 OSC 81                      011100000010
406            CALL X18g                   010100011110
407            CALL X18c HT                010011111011
```

408	OS8	OSC 81	011100000010
409		CALL X18c HT	010011111011
410	OS9	OSC 7	011010000000
411		CALL X9	010001100000
412		CALL X29	010110110110
413		CALL X19	010100110100
414		CALL X18g	010100011110
415		CALL X20	010100111100
416		INT B	000001010010
417		CALL X18a	010011100010
418		CALL X18e HT	010100001111
419	OS10	CALL X18g	010100011110
420		CALL X20	010100111100
421		INT B	000001010010
422		HALT	000000000000
423	OS11	OSC 7	011010000000
424		CALL X9	010001100000
425		CALL X29	010110110110
426		CALL X19	010100110100
427		INT B	000001010010
428		CALL X18g	010100011110
429		CALL X18a	010011100010
430		CALL X18e HT	010100001111
431	OS12	OSC 7	011010000000

432		CALL X 18a	010011100010
433		CALL X 18e	010100001110
434		JMP LOOP	100000011101
435	OS13	CALL X 18g	010100011110
436	OS14	OSC 7	011010000000
437		CALL X 9	010001100000
438		CALL X29	010110110110
439		CALL X 19	010100110100
440		CALL X18g	010100011110
441		CALL X 20	010100111100
442		INT B	000001010010
443		CALL X 18a	010011100010
444		CALL X 18e HT	010100001111
445	OS15	OSC 6	011001000000
446		CALL X 18g	010100011110
447		CALL X20	010100111100
448		INT B	000001010010
449		HALT	000000000000
450	OS16	OSC 7	011010000000
451		CALL X 32	010111001100
452		CALL X 9	010001100000
453		CALL X 30	010110111100
454		CALL X 19	010100110100
455		CALL X 18g	010100011110

456		CALL X20	010100111100
457		INT B	000001010010
458		CALL X18a	010011100010
459		CALL X18e	010100001110
460		CALL X18h HT	010100101101
461	OS17	OSC 64	011001010000
462		CALL X32	010111001100
463		CALL X18g	010100011110
464		CALL X20	010100111100
465		INT B	000001010010
466		CALL X18h HT	010100101101
467	OS18	OSC 7	011010000000
468		CALL X32	010111001100
469		CALL X9	010001100000
470		CALL X30	010110111100
471		CALL X19	010100110100
472		INT B	000001010010
473		CALL X18a	010011100010
474		CALL X18e HT	010100001111
475	OS19	OSC 64	011001010000
476		CALL X32 HT	010111001101
477	OS20	OSC 8	011100000000
478		HALT	000000000000
479	OS21	OSC 3	011000001000

```
480             CALL X9                 010001100000
481             MOVE 46                 001101000110
482             MOVE 35                 001100110101
483             HALT                    000000000000
484       OS22  OSC 53                  011000101000
485             CALL X9                 010001100000
486             MOVE 46                 001101000110
487             MOVE 35                 001100110101
488             HALT                    000000000000
489       OS23  OSC 3                   011000001000
490             CALL X9                 010001100000
491             MOVE 46                 001101000110
492             MOVE 35                 001100110101
493             INC SBP                 000001000000
494             CALL X18f               010100010110
495             CALL X18a               010011100010
496             CALL X18e HT            010100001111
497       OS24  OSC 3                   011000001000
498             CALL X9                 010001100000
499             MOVE 46                 001101000110
500             MOVE 35                 001100110101
501             INC SBP                 000001000000
502             CLR XMITC               000000100100
503             CALL X18f               010100010110
```

```
504                CALL X18a              010011100010
505                CALL X18e HT           010100001111
506          OS25  JMP LOOP               100000011101
507
508
509
510
511
```

Subroutines

0	X1	STORE 80	000101000000
1		STORE 81	000101000010
2		CLR DA	000000100000
3		STORE 82	000101000100
4		ADR A1	001010000010
5		STORE 90	000101001000
6		STORE 91	000101001010
7		STORE 92	000101001100
8		STORE 02 RTN	000100000101
9	X2	STORE 80	000101000000
10		STORE 81	000101000010
11		MOVE 70	001101110000
12		STORE 82	000101000100
13		ADR A1	001010000010
14		STORE 90	000101001000
15		STORE 91	000101001010
16		MOVE 80	001110000000
17		STORE 92 RTN	000101001101
18	X3	MOVE 01	001100000001
19		MOVE 80	001110000000
20		ADD A3	001000100000
21		STORE A0	000101010000
22		STORE A1	000101010010
23		MOVE 10	001100010000

24		DA (11)	000111000100
25		STORE A2	000101010100
26		MOVE 80	001110000000
27		STORE 02 RTN	000100000101
28			
29			
30	X4	STORE B0	000101011000
31		STORE B1	000101011010
32		STORE B2 RTN	000101011101
33	X5	STORE F0	000101111000
34		STORE F1	000101111010
35		STORE F2	000101111100
36		STORE F3	000101111110
37		ADR A1	001010000010
38		STORE 22 RTN	000100010101
39	X6	STORE C0	000101100000
40		STORE C1	000101100010
41		BDB 3	000110100110
42		MOVE 10	001100010000
43		STORE C2 RTN	000101100101
44	X7	ADB A1	001010000010
45		STORE 12 RTN	000100001101
46	X8	CLR DA	000000100000
47		STORE 12 RTN	000100001101

48	X9	LOAD 02	000010000100
49		LOAD 50	000010101000
50		LOAD 51	000010101010
51		SHIFT B	000000110010
52		ADR A3	001000100010
53		MOVE 01	001100000001
54		ADR A1	001010000010
55		MOVE 14	001100010100
55		MOVE 03	001100000011
57		RTN	000000000001
58			
59			
60	X11	LOAD 12	000010001100
61		LOAD 60	000010110000
62		LOAD 61	000010110010
63		SHIFT B	000000110010
64		ADR A3	001000100010
65		MOVE 01	001100000001
66		ADR A1	001010000010
67		MOVE 14	001100010100
68		MOVE 03	001100000011
69		RTN	000000000001
70	X12	LOAD 12	000010001100
71		LOAD 60	000010110000

72		LOAD 61	000010110010
73		SHIFT B	000000110010
74		ADR A3	001000100010
75		MOVE 01	001100000001
76		ADR A1	001010000010
77		MOVE 14	001100010100
78		MOVE 03	001100000011
79		BDB 3	000110100110
80		ADR A2	001001000010
81		CK1 RTN	000111110001
82	X13	LOAD 02	000010000100
83		LOAD 50	000010101000
84		LOAD 51	000010101010
85		SHIFT B	000000110010
86		ADR A3	001000100010
87		MOVE 01	001100000001
88		ADR A1	001010000010
89		MOVE 14	001100010100
90		MOVE 03	001100000011
91		ADR A1	001010000010
92		CKOP RTN	000111110111
93			
94			
95			

96	X14	CLR DB	000000100010
97		ADR A1	001010000010
98		CK1 RTN	000111110001
99	X15	BDB 3	000110100110
100		ADR A1	001010000010
101		CK1 RTN	000111110001
102	X16	LOAD 40	000010100000
103		LOAD 41	000010100010
104		LOAD 12	000010001100
105		ADR A3	001000100010
106		CK2 RTN	000111110011
107	X17	LOAD F0	000011111000
108		LOAD F1	000011111010
109		LOAD F2	000011111100
110		LOAD F3	000011111110
111		MOVE 01	001100000001
112		ADD A4 RTN	001000010001
113	X18a	LOAD 80	000011000000
114		LOAD 81	000011000010
115		LOAD 82	000011000100
116		WRITE RTN	000111010011
117			
118			
119			

120	X18b	LOAD 90	000011001000
121		LOAD 91	000011001010
122		LOAD 92	000011001100
123		DA (100)	000111000110
124		WRITE RTN	000111010011
125	X18c	LOAD 90	000011001000
126		LOAD 91	000011001010
127		LOAD 92	000011001100
128		DA (010)	000111001000
129		WRITE RTN	000111010011
130	X18d	LOAD 90	000011001000
131		LOAD 91	000011001010
132		LOAD 92	000011001100
133		DA (001)	000111001010
134		WRITE RTN	000111010011
135	X18e	LOAD 90	000011001000
136		LOAD 91	000011001010
137		LOAD 92	000011001100
138		WRITE RTN	000111010011
139	X18f	LOAD A0	000011010000
140		LOAD A1	000011010010
141		LOAD A2	000011010100
142		WRITE RTN	000111010011
143	X18g	LOAD B0	000011011000

144		LOAD B1	000011011010
145		LOAD B2	000011011100
146		WRITE RTN	000111010011
147			
148			
149			
150	X18h	LOAD C0	000011100000
151		LOAD C1	000011100010
152		LOAD C2	000011100100
153		WRITE RTN	000111010011
154	X19	LOAD 02	000010000100
155		MOVE 01	001100000001
156		DA (SC)	000111001100
157		STORE 32 RTN	000100011101
158	X20	LOAD 12	000010001100
159		MOVE 01	001100000001
160		DA (DT)	000111001110
161		STORE 32 RTN	000100011101
162	X21a	CLR DA	000000100000
163		SHIFT A	000000110000
164		STORE 52 RTN	000100101101
165	X21b	CLR DA	000000100000
166		SHIFT A	000000110000
167		SHIFT A	000000110000

168		STORE 52 RTN	000100101101
169	X21c	CLR DA	000000100000
170		SHIFT A	000000110000
171		SHIFT A	000000110000
172		SHIFT B	000000110010
173		STORE 52 RTN	000100101101
174			
175			
176			
177			
178			
179			
180	X21d	CLR XMITC	000000100100
181		CLR SBP	000000100110
182		CLR DA	000000100000
183		SHIFT A	000000110000
184		SHIFT A	000000110000
185		SHIFT A	000000110[illegible]
186		STORE 52 RTN	000100101101
187	X23	LOAD 00	000010000000
188		LOAD 01	000010000010
189		LOAD 03	000010000110
190		MOVE 70	001101110000
191		MOVE 01	001100000001

192		RTN	000000000001
193	X24	LOAD 00	000010000000
194		LOAD 01	000010000010
195		LOAD 03	000010000110
196		CLR DB RTN	000000100011
197	X25	LOAD 10	000010001000
198		LOAD 11	000010001010
199		LOAD 13	000010001110
200		BDB 2 RTN	000110100101
201	X26	LOAD 20	000010010000
202		LOAD 21	000010010010
203		LOAD 23	000010010110
204		BDB 2 RTN	000110100101
205			
206			
207			
208			
209			
210	X27	LOAD 30	000010011000
211		LOAD 31	000010011010
212		LOAD 33 RTN	000010011111
213	X28	LOAD F0	000011111000
214		LOAD F1	000011111010
215		LOAD F2	000011111100

216		LOAD F3	000011111110
217		MOVE 01	001100000001
218		RTN	000000000001
219	X29	MOVE 90	001110010000
220		WRITE	000111010010
221		RCV 5 RTN	000110001011
222	X30	MOVE 90	001110010000
223		WRITE	000111010010
224		RCV 4	000110001000
225		ADD A1	001010000000
226		LOAD 22	000010010100
227		WRITE RTN	000111010011
228	X31	STORE D0	000101101000
229		STORE D1 RTN	000101101011
230	X32	LOAD 22	000010010100
231		MOVE 02	001100000010
232		RTN	000000000001
233	X33	MOVE 90	001110010000
234		WRITE	000111010010
235		RCV 14 RTN	000110011101
236			
237			
238			
239			

240

241

242

243

244

245

246

247

248

249

250

251

252

253

254

255