[redacted]

Technical Report 70-119
NGR-21-002-206

June 1970

Microprogramming the IOEX of the IBSYS

by

Ira L. Avrunin

UNIVERSITY OF MARYLAND
COMPUTER SCIENCE CENTER
COLLEGE PARK, MARYLAND

ABSTRACT

The IBSYS operating system which is presently implemented on the IBM 7090/7094 computer systems consists of a supervisor and an input-output executor.

The supervisor initializes certain control locations and changes these locations in response to control card input. The input-output executor, IOEX, initiates activity on a channel, maintains activity on that channel as long as activity has been requested and supervises traps.

The primary parts of IOEX are presented, in this paper, as logical flow charts in algorithmic form. From these flow charts a hardware configuration is developed and sequence charts presented. From the sequence charts a control program is developed to present as a microprogram.

This paper shows that to develop a microprogram control sequence from a software system program is conceptually possible.

INTRODUCTION

The IBSYS operating system, which is currently implemented on the IBM 7090/94 computer system, consists of 5 parts.

- The processing programs are Subsystems that are maintained by EDIT.
- The special debugging facility is provided by SYSDUMP.
- The supervisor and control card interpreter is IBSUP.
- The resident portion of the nucleus is IBSNUC.
- The input output executor is IOEX.

This paper presents the input output executor and develops a configuration to produce the same results as the software IOEX using Microprogramming.

The approach taken is that used by Dr. Yaohan Chu, O.R. Pardo and Jeffrey Yeh in "A Methodology for Unified Hardware-Software Design." The first step is to study the existing program, in this case IOEX, derive the algorithm and present the algorithm in flow chart form. After deriving the software algorithm, a sequence chart is constructed with a proposed hardware configuration. The final step is to develop a control configuration and develop the microprogram for this.

In this paper the algorithm of IOEX has been modified to leave out certain contingency conditions. The algorithms presented are those for activating channel activity, selecting a unit on the channel and supervising traps.

From this study of the basic parts of IOEX, it has been shown that the Input Output Executor could be replaced by a control configuration.

ACKNOWLEDGEMENTS

The author wishes to express his thanks to Dr. Chu of the University of Maryland for his helpful suggestions, to Robert W. Tinker of the Naval Ship Research and Development Center for his technical assistance, and to his wife, Miriam, for her encouragement and help in preparing this paper.

Table of Contents

List of Tables and Figures

1. Introduction to IBSYS

Contingent with the coming of the second generation of computers was the debut of the various operating systems. These operating systems controlled the coordination of jobs within the machine. "Operating Systems" refers to the whole complex environment of programming, debugging and operational aids with which the programmer deals.

An operating system is made up of three main components: the input output supervisor, the processing programs, and the supervisor. IBSYS which is currently implemented on the IBM 7090/94 computer system, was selected for the study.

The input-output system of IBSYS is the Input-Output Executor, IOEX. IOEX is used by the supervisory programs as well as the processing programs under IBSYS, for coordination of I/O activities.

The Processing Programs are in the form of subsystems which are maintained by another processing program, the Editor.

The Supervisor portion consists of an always resident nucleus, IBNUC, and a transient portion IBSUP.

1.1 The supervisor--IBNUC and IBSUP.

The system supervisor, IBSUP, provides control card interpretation and job coordination. Most of the control

cards read in effect a switch used by the supervisor, or cause constant areas in the NUCLEUS, IBNUC, to be varied. For each control card read in an action occurs. A key control card is the $EXECUTE card which causes one of the processing programs to be loaded and given control. The processing program uses the locations of IBNUC to determine unit assignments and other necessary information. Certain locations of IBNUC are used for communication with the supervisor.

The main parts of the nucleus are:

- The Communication Region, containing constants, control words, and transfer words required for intercommunication between the Supervisor and the Processing Programs.
- The System Unit Function Table which is used to keep track of all the system unit function assignments.
- The Unit Control Block (UCB) Table, consisting of a one word entry for each channel. The one word entry contains the address of the first Unit Control Block for that channel, the total number of input-output devices assigned to that channel, and the number of card units assigned to that channel. The UCB's are always resident and are a portion of IOEX.
- The System Loader, used to load records from the System Library unit.
- The Unit Availability Table, consisting of one word for each channel, which points to the beginning of a chain of units that are not assigned to system unit functions.

To coordinate the use of input and output devices between processing programs and the supervisor, IBSYS provides symbolic input output units called system unit functions. The address portion of each word in the

System Unit Function table contains the address of the first word of the Unit Control Block for the unit assigned to the particular System Unit Function.

The Unit Control Block (UCB) is used to control the usage of Input and Output devices. The UCB indicates such information as: availability of the unit, whether the unit is attached, the unit type, the channel type, the actual ECD address of the unit, whether an end of tape has been sensed, the tape density and the chain address if the unit is in the availability chain.

The cold start procedure for IBSYS consists of loading the supervisor which then initializes the nucleus and generates the Unit Control Blocks, which have the format shown in figure 1. The nucleus chaining is then done and the chaining structure of core at the end of cold start is as shown in figure 2.

1.2 The Input-Output Executor, IOEX.

The input-output Executor, (IOEX), which is always core resident, consists of a trap supervisor for data channel operations and a series of subroutines to perform specific functions that might be required by a processing program. The subroutines of IOEX control the movement of data, perform non data selct operations, convert data, handle on-line printing, reading and punching and provide temporary and permanent halts. The Trap supervisor handles all data channel traps and maintains activity on a channel

as long as some activity has been requested.

The parts of IOEX which were selected for this report are really the key parts for Input-Output activity and coordination. These parts are: ACTIVE, ISSEK, and TRAP.

When a problem program wishes to initiate activity on a unit of a channel the following steps are taken. First the user stores in the second word of the Unit Control Block the location of a routine which will start the input-output sequence and will handle the completion of the sequence. After storing this in the UCB, the user then calls on the ACTIVE subroutine with the following sequence:

```
TSX     .ACTV,4
P       A,T
```

Where P is a prefix of either MZE or PZE and A is the address of a pointer to the desired UCB. If P is PZE, the unit is to be started if the channel is free, if the channel is not free control is returned to the processing program. If the prefix is MZE, the unit is given priority and control is not returned until the unit is started.

The processing program is responsible for providing the routines for initiating the I/O. The routine for doing the I/O is called the select routine and will be entered two times. The first entry is with the sign of the accumulator + (SEL+). This entry is for starting the channel operations. The second time is with the sign of the accumulator - (SEL-) and at this time the routine does what it wants, knowing that the operation has been completed.

The Trap Supervisor of IOEX handles the interruptions

that occur on termination of the channel programs. The channel program constructed by SELF has to end in a condition causing a trap to occur. The trap supervisor gives control to the SELF routine and then initiates activity for any other unit that requires it on the channel.

ISSEK is a subroutine, used by ACTIVE and TRAP, which gives control to the SELF routine after actually determining which unit should get control.

2. IOEX

IOEX provides subroutines for printing messages on-line, converting data, performing non-data select operations and coordinating the I/O activity on the channels.

The part of IOEX to be studied here is the part that coordinates the use of the channels. The assumption was made that only tape drives would be used so the additional coding for hypertape and disk devices has been bypassed.

Channel activity is started by calling on ACTIVE. Once ACTIVE has done some housekeeping, it gives control to ISSEK to initiate the unit and give control to the SELECT routine. When the requested I/O operation finishes, the TRAP supervisor gets control and determines what channel and unit has just ended. A check is made for any errors and if possible recovery is attempted. After

the input-output operation is completed, the users Select routine is entered for the posting entry. Upon return from the Select routine, ISSEK is called to initiate the next unit on the channel if one has requested activity.

2.1 ACTIVE

The ACTIVE subroutine is used to initiate I/O operations on a channel for a specified unit. The flow chart is shown in the appendix as figures A1a and A1b.

After testing to be sure the unit is legal, the traps are disabled so that a trap cannot occur. The channel number is then extracted from the unit control word (UCW), which is the first word of the UCB, and used as an index for addressing certain arrays.

The CHXAC array is used to store the UCW address of the active unit on each channel--there is one entry for each channel. The CHXSP array is used to store the address of the UCW for the unit that has priority on that channel. If the call is made during TRAP time the unit is just given priority and exit made, i.e. CHXSP is set to point to the UCW.

The channel activity location CHXAC is then checked to see if the channel has an active unit. If not, the ISSEK subroutine is called to initiate the activity. Upon return from ISSEK the traps are enabled and return made.

If the channel was active on entry, the prefix in the calling sequence is picked up and tested. If the prefix is PZE, return is made to the calling program. If the prefix was MZE the channel priority location is set to point to the requested UCW, the interrupts are enabled and a loop is entered. The loop is terminated when TRAP processes a trap for the channel and the unit is given control of the channel.

2.2 TRAP

The trap supervisor, TRAP, is entered whenever a trap occurs on a channel. The flow chart for TRAP is shown in the appendix as figures A2a, b, c, d. The first thing done is to determine which channel trapped. The locations starting a 12_8 are for the storage of channel information when a trap occured. By checking for a non-zero location, TRAP can determine which channel has trapped. The hardware of the 7090 stores the information for each channel in even locations and executes the next odd instruction. For Channel A the IC is stored at location 12 and location 13 executed. Once the channel is determined, the channel activity cell is checked to find out which unit was active on that channel. Then the operation is checked for completion, and whether the operation was successful. In addition to I/O errors being checked, a check is done for End of File or noise records.

Having processed the error condition, the users SEI- routine is entered. On return from the SEL-, the subroutine ISSEK is called to initiate activity, if necessary, for another unit on that channel.

2.3 ISSEK

The subroutine, ISSEK, is called by TRAP and ACTIVE. ISSEK is shown in the flow chart A3 in the appendix.

ISSEK checks to be sure that priority has been assigned to the unit, before just initiating the same one, ISSEK also makes sure there is a select routine to go to. If no unit has priority, ISSEK searches the chain of UCBs to find a unit that has a select routine provided. Once a UCB is found, for which there is a Select routine, and the hardware of the channel is free for new activity, the users SEI+ routine is entered. Upon return from the Select routine, some housekeeping is done and return made to the calling program.

3. SEQUENCE CHARTS

Having extracted the algorithms for the desired functions of IOEX, it is now necessary to develop a configuration for the implementation using hardware. The sequence charts are then developed for each of the subroutines.

3.1 Configuration

The computer elements required for the implementation of IOEX are shown in the block diagram of figure 3, except for the control components which are discussed separately. The main memory is called MEM and it is the memory where a users program would reside and where the Unit Control Blocks and other communication information is stored. The auxiliary memories, CHXAC, CHXSP, URRX, UCBX, and CHAN are used as storage arrays with each memory having one location for each channel possible.

The memories are addressed by 15 bit registers having the names AR, AA, AP, ARR, AU, and ACC. For each memory there is a buffer register of 36 bits called SR, AB, PB, RB, UB, and CB, respectively. There are also several 36 bit work registers named TRAPSW, ENBSW, ISSSP, IOAC, IOMQ, and IOSI. There are also several 15 bit registers used for storage and counting--these are X1, X2, and X4. To provide save areas for the CPU registers when having to give control to select routines, there are save registers with a suffix of S attached to the CPU register. The names of these are ACS, MQS, SIS, XR1S, XR2S and CPUICS.

In order to provide for the waiting process during active, there are two save registers for saving the state of IOEX registers. These are SAVEX1, to save X1, and SAVEIOAC, to save the contents of IOAC.

In addition to the 15 and 36 bit registers, there is a series of one bit registers used for control purposes. These registers are ACTIVE, WAIT, ISSEK, TRAP, SAVE, RESTORE, IOCHECK, USER, and WAITING.

The configuration just given for the IOEX processor can be described using the following Computer Design Language (CDL) statements.

```
Comment, configuration of the IOEX Processor
Register,   AR(0-14),       $address register for main memory
            SR(0-35),       $memory buffer register
            TRAPSW(0-35),   $location of trapped UCW
            ENBSW(0-35),    $which traps can occur
            ISSSP(0-35),    $work register for ISSEK routine
            IOAC(0-35),     $IOEX storage register
            IOMQ(0-35),     $IOEX temporary register
            X1(0-14),       $channel number storage register
            X2(0-14),       $work register
            X4(0-14),       $parameter pointer
            UCWPT(0-14),    $pointer to first word of UCB(UCW)
            COUNT(0-14),    $counting register
            IOSI(0-35),     $IOEX sense indicator settings
            AP(0-14),       $address register for CHXSP memory
            AA(0-14),       $address register for CHXAC memory
            ARR(o-14),      $address register for URRX memory
            AU(0-14),       $address register for UCBX memory
            ACC(0-14),      $address register for CHAN memory
            PB(0-35),       $buffer register for CHXSP memory
            AB(0-35),       $buffer register for CHXAC memory
            RB(0-35),       $buffer register for URRX memory
            UB(0-35),       $buffer register for UCBX memory
            CB(0-35),       $buffer register for CHAN memory
            ACS(0-35),      $save register for CPU AC
            MQS(0-35)       $save register for CPU MQ
            SIS(0-35),      $save register for CPU SI
            XR1S(0-14),     $save register for CPU XR1
```

```
                XR2S(0-14),         $save register for CPU XR2
                XR4S(0-14),         $save register for CPU XR4
                CPUICS(0-14),       $save register for CPU IC
                SAVEXI(0-14)        $save register for IOEX-XI while waiting
                SAVEIOAC(0-35),     $save register for IOEX-IOAC while waiting
                ACTIVE,             $control register for ACTIVE sequence
                WAIT,               $control register for ACTIVE sub sequence
                ISSEK,              $control register for ISSEK sub sequence
                TRAP,               $control register for TRAP sub sequence
                SAVE,               $control register for SAVE sub sequence
                RESTORE,            $control register for RESTORE sub sequence
                IOCHECK,            $control register for IOCHECK sub sequence
                USER,               $control register for waiting during SEL(++)
                WAITING,            $control register for ACTIVE waiting
                AC(0-35),           $AC
                MQ(0-35),           $MQ
                SI(0-35),           $SI
                XR1(0-14),          $XR1
                XR2(0-14)           $XR2
                XR4(0-14)           $XR4
                CPUIC(0-14)         $IC
Subregister,    IOAC(ADDR)=IOAC(21-35),     $address part of IOAC
                IOAC(DEC)=IOAC(3-17),       $decrement part of IOAC
                IOAC(SEL)=IOAC(3-17),       $address of SEL routine
                AC(S)=AC(0)                 $sign of accumulator
Memory ,        MEM(AR)=MEM(0-32791,0-35),  $main memory
                CHXSP(AP)=CHXSP(1-8,0-35),  $channel priority memory
                CHXAC(AA)=CHXAC(1-8,0-35),  $channel activity memory
```

```
URRX(ARR)=URRX(1-8,0-35),  $redundancy channel memory
UCBX(AU)=UCBX(1-8,0-35),   $unit control block locator memory
CHAN(AC)=CHAN(1-9,0-35)    $channel status memory
```

3.2 ACTIVE Sequence Chart

The sequence chart for the ACTIVE sequence is shown in Figure 4. It is assumed that the unit has been checked to be sure it is valid and that CPU register 4 is pointing to the parameter list.

When the control element ACTIVE goes to 1 the coding is started. Registers are initialized and the address of the Unit Control Word picked up. The index of the channel is extracted from the ucw and the auxiliary memory addressing registers set to address the appropriate location. If the entry is during trap time, the unit is given priority by setting the UCW address in CHXSP(AP) and returning to the calling program.

If entry is not during trap time, the CPU registers are saved and the channel activity location is checked to see if the channel is active. If the channel is not active, ISSEK is set to 1 to initiate the ISSEK sequence. On return from ISSEK, control is given back to the CPU program. If there is an active unit, the value of the parameter is checked for its sign. If the sign of the parameter is 0 then the request was PZE and return is made. If the request was MZE the controls for waiting are set up. The unit is given priority, the registers of IOEX are stored and WAITING and WAIT are set to 1. The system then loops until WAIT is not 1. WAIT is made zero by TRAP when TRAP detects that WAITING is 1. When

the wait period is over, the status of the channel is checked again to see if the unit was given control by TRAP. If it was, then return is made to the calling program; otherwise the wait loop is reentered.

3.3 TRAP sequence chart

The sequence chart for the TRAP sequence is shown in Figure 5. The control Register TRAP is checked to see if it is 1. When TRAP becomes 1, the sequence begins. If the system was currently waiting, the status of all CPU registers is saved and then the work registers are initialized. Location 9 of the CHAN memory contains the total number of channels on the system at present. This value is used as a counter. The even locations of core are searched, starting at 12_8 for one which is non-zero indicating the channel which trapped.

When the channel that trapped is found, X1 will contain its index. The memory registers are then set to the value of X1 for reading out or writing into and IOAC and IOSI are initialized. If no activity is known of or there is no priority the transfer to a user select routine is skipped and ISSEK is given control.

If there is an active unit, IOCHECK is set to 1 to process the trap for any error conditions. The error processor will try to correct the condition and post the UCB as to the results. The record count field is

then fetched and updated and stored back into the UCB. The address of the select routine is then extracted and the CPUIC set to give control to the routine. The USER control register is set to 1 to hold up the control program while the Select processing is being done. When the SEL- is finished, USER will be set to 0 and IOEX can continue.

After the SEL- entry, there are bookkeeping functions performed and ISSEK is given control to select the next unit. On return from ISSEK, the location indicating the trap is set to 0 and WAITING is tested to see if a wait was in process. If a WAIT was in process, then WAIT is set to 0 to allow the checking to occur.

3.4 ISSEK sequence Chart

Figure 6 presents the ISSEK sequence. When ISSEK becomes 1 this sequence begins.. Initialization is performed and the priority location is checked to see if any unit has priority. If there is a unit with priority and there is a specified Select routine then the channel is reset and the users select routine given control when the channel operation has stopped. The Select routine is given control by setting the CPUIC to the address and setting USER to 1. If no unit has priority, or there is no specified select routine for the unit with priority, a search is initiated for a UCB with a select routine.

The chain of UCB's on the channel is searched for a specified select routine and this unit is given control of the channel. If no unit has a Select routine specified, the channel is allowed to go dormant and is inactive.

4. Microprogram

To convert the sequence charts into a Microprogram , a control configuration is first developed. The sequence charts are then expressed in CDL and the final Microprogram developed.

4.1 Microprogram Control Configuration

The control memory is a fast memory having a capacity of 64 words of 96 bits. There is a 6 bit address register CAR and a 96 bit buffer register F. Each location of the control memory contains a microinstruction.

The control configuration is shown in Figure 7. Register E is a 1 bit register used to control read out of microinstructions or their execution. When E=0 the memory is read and when E=1 the execution of the instruction occurs. To provide for 1 level of subroutine, 6 bit register CARRET has been provided. There is a four phase clock P. MC is a ring counter used to sequence the main memory through four cycles.

The control signals are developed as a combination of P, MC and F. This configuration can be described by the following CDL statements.

```
Comment, control configuration for the IOEX processor
Register,     MC(0-3),              $main & auxiliary memory control
              CAR(1-6),             $control memory address register
              F(0-95),              $control memory buffer register
              E,                    $control memory read control
              GO,                   $control flip flop
              CARRET(1-6)           $subroutine return
Terminal,     RUN=ACTIVE.OR.TRAP.OR.ISSEK.OR(WAITING.AND WAIT')
Subregister,  F(ADS)=F(90-95)       $address field
Memory,       CM(0-63;0-95)         $control memory
Switch,       START(ON)             $start switch
Clock,        P(0-3)                $four phase clock
/P(3)*RUN/    MC <-CIRMC            $sequence MC
```

4.2 Statement Description

The format of the control word is shown in Table 1, with each bit performing the specified operation.

The timing signals are such that each clock cycle is a control memory cycle and 4 control memory cycles make up the main memory cycle. There are 4 steps in each control memory cycle and 16 steps in each main memory cycle.

The reading of the main memory occurs at the 5th clock step (i/e. MC(1)*P(1)*RUN). The writing into main memory occurs at the 10th step (MC(2)*P(1)*RUN).

When E is reset to 0 the following sequence of labels occurs to read another microinstruction.

```
/P(0)*RUN*E'/          F ←-CM(CAR)
                       E ←-1

/P(1)*RUN*E'/

/P(2)*RUN*E'/

/P(3)*RUN*E'/          CAR ←-countup CAR
```

CAR is set depending on which action has been requested (i.e. ACTIVE or TRAP) and run is a terminal defined for execution control purposes. The state of WAIT and WAITING determines the ACTIVE wait loop.

The following CDL statements can be used to describe the sequence charts.

```
Comment, description of active sequence
/START(ON)/                 GO ←-1, ISSEK ←-0, ACTIVE ←-0, TRAP ←-0
/ACTIVE*GO*ISSEK'/          GO ←-0, CAR ←-0, E ←-0
/TRAP*GO*ISSEK'/            GO ←-0, CAR ←-15, E ←-0
Comment, control memory fetch cycle
/P(0)*RUN*E'/               F ←-CM(CAR), E ←-1
/P(3)*RUN*E'/               CAR ←-countup CAR
Comment, execute microinstruction at location 0
/P(1)*RUN*F(48)/            WAIT ←-0
/P(1)*RUN*F(49)/            TRPSW ←-0
/P(1)*RUN*F(87)/            X4 ←-XR4
/P(2)*RUN*F(10)/            AR ←-X4.ADD.1
/MC(1)*P(1)*RUN*F(1)/       SR ←-MEM(AR)
/MC(3)*P(2)*RUN/            E ←-0
Comment, fetch and execute microinstruction at location 1
/P(3)*RUN*E'/               CAR ←-countup CAR
/P(0)*RUN*E'/               F ←-CM(CAR), E ←-1
/P(1)*RUN*F(11)/            AR ←-SR(21-35)
/MC(1)*P(1)*RUN*F(1)/       SR ←-MEM(AR)
/MC(3)*P(2)*RUN/            E ←-0
Comment, fetch and execute the microinstruction at location 2
/P(3)*RUN*E'/               CAR ←-countup CAR
/P(0)*RUN*E'/               F ←-CM(CAR), E ←-1
/P(1)*RUN*F(26)/            IOAC ←-SR
/P(1)*RUN*F(85)/            UCWPT ←-SR
/P(1)*RUN*F(11)/            AR ←-SR(21-35)
/MC(1)*P(1)*RUN*F(1)/       SR ←-MEM(AR)
```

```
/MC(3)*P(2)*RUN/          E <-0

Comment, fetch and execute the microinstruction at location 3

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(32)/          X1 <-SR(3-17).AND.017

/F(1)*RUN*F(69)/          IF(TRAP=0) THEN CAR<-F(ADS)          $XFER to 5

/F(1)*RUN*F(15)/          AP <-X1,AA <-X1,ARR <-X1,AU <-X1,ACC <-X1

/P(2)*RUN*F(0)/           E <-0

Comment fetch and execute microinstruction at location 4

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(16)/          PB <-IOAC

/P(1)*RUN*F(39)/          CPUIC <-X4.ADD.1

/MC(2)*F(1)*RUN*F(16)/    CHXSP(AP) <-PB

/MC(3)*P(3)*RUN*F(66)/    GO <-1,ACTIVE <-0

Comment, fetch and execute microinstruction at location 5

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(42)/          CPUICS <-CPUIC,ACS <-AC,XR1S <-XR1,XR2S <-XR2,
                          XR4S <-XR4,MQS <-MQ,SIS <-SI

/P(2)*RUN*F(10)/          AR <-X4.ADD.1

/MC(1)*P(1)RUN*F(1)/      SR <-MEM(AR)

/MC(3)*P(2)*RUN/          E <-0

Comment, fetch and execute the microinstruction at location 6

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(26)/          IOAC <-SR

/MC(1)*P(1)*RUN*F(2)/     AB <-CHXAC(AA)
```

```
/MC(3)*P(2)*RUN/          E <-0

Comment, fetch and execute the microinstruction at location 7

/P(3)*RUN*E'/             CAR <-countup (CAR)

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(69)/          IF (AB=0) THEN CAR <-F(ADS)          $XFER to 10

/P(2)*RUN*F(0)/           E <-0

Comment, fetch and execute the microinstruction at location 8

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(70)/          IF(IOAC(S)=1) THEN CAR<-F(ADS)       $XFER to 12

/P(2)*RUN*F(0)/           E <-0

Comment, fetch and execute the microinstruction at location 9

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(40)/          AC <-ACS, XR1 <-XR1S, XR2 <-XR2S, XR4 <-XR4S,
                          MQ <-MQS, SI <-SIS

/P91)*RUN*F(39)/          CPUIC <-X4.ADD.1

/MC(3)*P(3)*RUN*F(66)/    GO <-1, ACTIVE <-0

Comment, fetch and execute the microinstruction at location 10

/P(3)*RUN*E'/             CAR <-countupCAR

/P(10)*RUN*E'/            F <-CM(CAR),E <-1

/P(1)*RUN*F(43)/          IOMQ <-IOAC

/P(1)*RUN*F(17)/          RB <-0

/P(1)*RUN*F(50)/          TRPSW <-UCWPT

/P(2)*RUN*F(27)/          IOAC <-UCWPT

/MC(2)*P(1)*F(7)/         URRX(ARR) <-RB, CARRET <-CAR

/MC(2)*P(2)*F(47)/        CAR <-F(ADS)                         $XFER to 32

/MC(2)*P(2)*F(51)/        ISSEK <-1
```

```
/MC(3)*P(2)*RUN/          E <-0

Comment, fetch and execute the microinstruction at location 11

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(49)/          TRPSW <-0

/P(1)*RUN*F(52)/          ENBSW (3-17) <-0

/P(1)*RUN*F(40)/          AC <-ACS,XR1 <-XR1S,XR2 <-XR2S,XR4<-XR4S,
                          MQ <-MQS,SI <-SIS

/P(1)*RUN*F(39)/          CPUIC <-X4.ADD.1

/MC(3)*P(3)*RUN*F(66)/    GO <-1,ACTIVE <-0

Comment, fetch and execute the microinstruction at location 12

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(16)/          PB <-IOAC

/P(1)*RUN*F(53)/          WAITING <-1

/P(1)*RUN*F(54)/          SAVEX1 <-X1

/P(1)*RUN*F(55)/          SAVEIOAC <-IOAC

/P(1)*RUN*F(56)/          WAIT <-1

/MC(1)*P(1)*RUN*F(6)/     CHXSP(AP) <-PB

/MC(3)*P(3)*RUN*F(66)/    GO <-1,ACTIVE <-0

Comment, fetch and execute the microinstruction at location 13

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR),E <-1

/P(1)*RUN*F(33)/          X1 <-SAVEX1

/P(1)*RUN*F(28)/          IOAC <-SAVEIOAC

/P(1)*RUN*F(15)/          AP <-X1,AA <-X1,ARR <-X1,AU <-X1,ACC <-X1

/MC(1)*P(1)*RUN*F(3)/     PB <-CHXSP(AP)

/MC(1)*P(1)*RUN*F(2)/     AB <-CHXAC(AA)
```

```
/MC(3)*P(2)*F(0)/          E <-0

Comment, fetch and execute microinstruction at location 14

/P(3)*RUN*E'/              CAR <-countup CAR

/P(0)*RUN*E'/              F <-CM(CAR),E <-1

/P(1)*RUN*F(71)/           IF(PB=0.OR.IOAC=AB) THEN (WAITING <-0,
                           CAR <-F(ADS) ELSE (WAIT <-1,GO <-1,ACTIVE <-0)

Comment, description of the trap sequence

Comment, fetch and execute the microinstruction at location 15

/P(0)*RUN*E'/              F <-CM(CAR),E <-1

/P(1)*RUN*F(76)/           IF(WAITING=1)THEN CPUICS <-CPUIC,ACS <-AC,MQS<-MQ,
                           XR1S <-XR1,XR2S <-XR2,XR4S <-XR4,SIS <-SI

/P(1)*RUN*F(14)/           AR <-10,ACC <-9,X1 <-1

/MC(1)*P(1)*RUN*F(4)/      CB <-CHAN(ACC)

/MC(3)*P(2)*RUN/           E <-0

Comment, fetch and execute the microinstruction at location 16

/P(3)*RUN*E'/              CAR <-countup CAR

/P(0)*RUN*E'/              F <-CM(CAR),E <-1

/P(1)*RUN*F(44)/           COUNT <-CB(21-35)

/P(2)*RUN*F(0)/            E <-0

Comment, fetch and execute the microinstruction at location 17

/P(3)*RUN*E'/              CAR <-countup CAR

/P(0)*RUN*E'/              F <-CM(CAR),E <-1

/MC(1)*P(1)*RUN*F(1)/      SR <-MEM(AR)

/MC(3)*P(2)*RUN/           E <-0

Comment, fetch and execute the microinstruction at location 18

/P(3)*RUN*E'/              CAR <-countup CAR

/P(0)*RUN*E'/              F <-CM(CAR),E <-1

/P(1)*RUN*F(72)/           IF(SR=0) THEN CAR <-F(ADS) ELSE X1<-countup X1,
                           COUNT <-countdown COUNT, AP <-AR.ADD.2
                                                        $XFER to 21

/P(2)*RUN*F(0)/            E --0
```

```
Comment, fetch and execute the miroinstruction at location 19
/P(3)*RUN*E'/            CAR <-countupCAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
/P(1)*RUN*F(73)/         IF(COUNT=0) THEN CAR <-F(ADS)        $XFER to 17
/P(2)*RUN*F(0)/          E <-0
Comment, fetch and execute the microinstruction at location 20
/P(3)*RUN*E'/            CAR <-countupCAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
/P(1)*RUN*F(78)/         IF(WAITING=1) THEN (WAIT <-0,CAR <-F(ADS))
                                                      $XFER to 13
/P(2)*RUN*F(86)/         GO <-1, TRAP <-0
Comment, fetch and execute the microinstruction at location 21
/P(3)*RUN*E'/            CAR <-countupCAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
/P(1)*RUN*F(26)/         IOAC <-SR
/P(1)*RUN*F(57)/         IOSI <-SR
/(P(1)*RUN*F(15)/        AP <-X1,AA <-X1,ARR <-X1,AU<-X1,ACC <-X1
/MC(1)*P(1)*RUN*F(2)/    AB <-CHXAC(AA)
/MC(1)*P(1)*RUN*F(3)/    PB <-CHXSP(AP)
/MC(2)*P(2)*RUN/         E <-0
Comment, fetch and execute microinstruction at location 22
/P(3)*RUN*E'/            CAR <-countupCAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
/P(1)*RUN*F(25)/          IF(AB=0) THEN CAR <-F(ADS)          $XFER to 25
/P(2)*RUN*F(0)/          E <-0
Comment, fetch and execute microinstruction at location 23
/P(3)*RUN*E'/            CAR <-countup CAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
```

```
/P(1)*RUN*F(74)/          IF (PB=0) THEN CAR <-F(ADS)          $XFER to 29
/P(2)*RUN*F(0)/           E <-0
Comment, fetch and execute the microinstruction at location 24
/P(3)*RUN*E'/             CAR <-countup CAR
/P(0)*RUN*E'/             F <- CM(CAR),E <-1
/P(1)*RUN*F(18)/          AB <-PB
/MC(2)*P(1)*RUN*F(9)/     CHXAC(AA) <-AB
/MC(2)*P(2)*RUN/          E <-0
Comment, fetch and execute the microinstruction at location 25
/P(3)*RUN*E'/             CAR <-countup(CAR)
/P(0)*RUN*E'/             F <-CM(CAR),E <-1
/P(1)*RUN*F(58)/          IOCHECK <-1
/P(1)*RUN*F(46)/          CARRET <-CAR
/P(2)*RUN*F(47)/          CAR <-F(ADS)                         $XFER TO IOCHECK
/P(2)*RUN*F(0)/           E <-0
Comment, fetch and execute microinstruction at location 26
/P(3)*RUN*E'/             CAR <-countup (CAR)
/P(0)*RUN*E'/             F <-CM(CAR),E <-1
/P(1)*RUN*F(35)/          X2 <-AR
/P(2)*RUN*F(12)/          AR <-TRPSW.ADD.1
/MC(1)*P(1)*RUN*F(1)/     SR <-MEM(AR)
/MC(2)*P(2)*RUN/          E <-0
Comment, fetch and execute the microinstruction at location 27
/P(3)*RUN*E'/             CAR <-countupCAR
/P(0)*RUN*E'/             F<-CM(CAR),E <-1
/P(1)*RUN*F(19)/          SR(COUNT) <-countup SR(COUNT)
/MC(2)*P(1)*RUN*F(8)/     MEM(AR) <-SR
/MC(3)*P(2)*RUN/          E <-0
```

```
Comment, fetch and execute the microinstruction at location 28
/P(3)*RUN*E'/            CAR <-countup CAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
/P(1)*RUN*F(41)/         CPUIC <-SR(ADDR)
/P(1)*RUN*F(59)/         AC(S) <-1
/P(1)*RUN*F(60)/         USER <-1
/P(2)*RUN*F(0)/          E <-0
Comment, fetch and execute the microinstruction at location 29
/P(3)*RUN*E'/            CAR <-countup CAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
/P(1)*RUN*F(77)/         IF (USER=1) THEN CAR <-F(ADS)      $XFER to 29
/P(2)*RUN*F(0)/          E <-0
Comment, fetch and execute the microinstruction at location 30
/P(3)*RUN*E'/            CAR <-countup CAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
/P(1)*RUN*F(17)/         RB <-0
/P(1)*RUN*F(29)/         IOAC <-AB
/P(1)*RUN*F(20)/         PB <-AB
/P(1)*RUN*F(61)/         UCWPT <-TRPSW
/P(1)*RUN*F(46)/         CARRET <-CAR
/P(2)*RUN*F(65)/         IOAC(S) <-0                        $XFER TO 32
/P(2)*RUN*F(47)/         CAR <-F(ADS)
/MC(2)*P(1)*RUN*F(7)/    URRX(ARR) <-RB
/MC(2)*P(1)*F(6)/        CHXSP(AP) <-PB
/MC(2)*P(1)*F(51)/       ISSEK <-1
Comment, fetch and execute the microinstruction at location 31
/P(3)*RUN*E'/            CAR <-countup CAR
/P(0)*RUN*E'/            F <-CM(CAR),E <-1
```

```
/P(1)*RUN*F(13)/          AR <-X2
/P(1)*RUN*F(21)/          SR <-0
/P(2)*RUN*F(47)/          CAR <-F(ADS)
/MC(2)*P(1)*RUN*F(8)/     MEM(AR) <-SR
/MC(3)*P(2)*RUN/          E <-0
Comment, ISSEK subroutine sequence
Comment, fetch and execute the microinstruction at location 32
/P(3)*RUN*E'/             CAR <-countup CAR
/P(0)*RUN*E'/             F <-CM(CAR),E <-1
/P(1)*RUN*F(36)/          X2<-UCWPT
/P(1)*RUN*F(62)/          ISSSP <-UCWPT
/MC(1)*P(1)*RUN*F(3)/     PB <-CHXSP(AP)
/MC(3)*P(2)*RUN/          E <-0
Comment, fetch and execute the microinstruction at location 33
/P(3)*RUN*E'/             CAR <-countup CAR
/P(0)*RUN*E'/             F <-CM(CAR),E <-1
/P(1)*RUN*F(74)/          IF(PB=0) THEN CAR <-F(ADS)        $XFER to 37
/P(2)*RUN*F(0)/           E <-0
Comment, fetch and execute the microinstruction at location 34
/P(3)*RUN*E'/             CAR <-countup CAR
/P(0)*RUN*E'/             F <-CM(CAR),E <-1
/P(1)*RUN*F(10)/          AR <-X4.ADD.1
/MC(1)*P(1)*RUN*F(1)/     SR <-MEM(AR)
/MC(3)*P(2)*RUN/          E <-0
Comment, fetch and execute the microinstruction at location 35
/P(3)*RUN*E'/             CAR <-countup CAR
/P(0)*RUN*E'/             F <-CM(CAR),E <-1
/P(1)*RUN*F(79)/          IF(SR(3-17)=0) THEN CAR <-F(ADS)   $XFER to 42
/P(2)*RUN*F(0)/           E <-0
```

```
Comment, fetch and execute the microinstruction at location 36
/P(3)*RUN*E'/              CAR <-countup CAR
/P(0)*RUN*E'/              F <-CM(CAR),E <-1
/P(1)*RUN*F(21)/           SR <-0
/P(1)*RUN*F(63)/           ISSSP <-0
/MC(2)*P(1)*RUN*F(8)/      MEM(AR) <-SR
/MC(3)*P(1)*RUN/           E <-0
Comment, fetch and execute the microinstruction at location 37
/P(3)*RUN*E'/              CAR <-countup CAR
/P(0)*RUN*E'/              F <-CM(CAR),E <-1
/MC(1)*P(1)*RUN*F(5)/      UB <-UCBX(AU)
/MC(3)*P(2)*RUN/           E <-0
Comment, fetch and execute the microinstruction at location 38
/P(3)*RUN*E'/              CAR <-countup CAR
/P(0)*RUN*E'/              F <-CM(CAR),E --1
/P(1)*RUN*F(31)/           IOAC <-UB
/P(1)*RUN*F(37)/           X2 <-UB(21-35)
/P(1)*RUN*F(45)/           COUNT <-UB(3-17)
/P(2)*RUN*F(0)/            E <-0
Comment, fetch and execute the microinstruction at location 39
/P(3)*RUN*E'/              CAR <-countup CAR
/P(0)*RUN*E'/              F <-CM(CAR),E --1
/P(2)*RUN*F(22)/           AR <-X2.ADD.1
/MC(1)*P(2)*RUN*F(1)/      SR <-MEM(AR)
/MC(3)*P(2)*RUN/           E <-0
Comment, fetch and execute the microinstruction at location 40
/P(3)*RUN*E'/              CAR <-countup CAR
/P(0)*RUN*E'/              F <-CM(CAR),E --1
```

```
/P(1)*RUN*F(80)/          IF (SR=0) THEN (ISSSP <-X2, XR2 <-X2,
                          CAR <-F(ADS)) ELSE X2 <-X2.ADD.4,
                          COUNT <-countdown COUNT          $XFER to 43

/P(2)*RUN*F(81)/          IF (COUNT=0) THEN CAR <-CAR.ADD.2
                          ELSE AB <-PB

/P(2)*RUN*F(0)/           E <-0

Comment, fetch and execute the microinstruction at location 41

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR), E <-1

/MC(2)*P(1)*RUN*F(6)/     CHXSP(AP) <-PB

/MC(2)*P(1)*RUN*F(9)/     CHXAC(AA) <-AB

/MC(3)*P(2)*RUN/          E <-0

Comment, fetch and execute the microinstruction at location 42

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR), E <-1

/P(1)*RUN*F(67)/          ISSEK <-0

/P(1)*RUN*F(84)/          CAR <-CARRET

/P(2)*RUN*F(0)/           E <-0

Comment, fetch and execute the microinstruction at location 43

/P(3)*RUN*E'/             CAR <-countup CAR

/P(0)*RUN*E'/             F <-CM(CAR), E <-1

/P(1)*RUN*F(64)/          RES CHAN X1

/P(1)*RUN*F(25)/          AC <-X2, AB <-X2, PB <-X2

/P(2)*RUN*F(65)/          AC(S) <-0

/MC(2)*P(1)*RUN*F(90/     CHXAC(AA) <-AB

/MC(2)*P(1)*RUN*F(6)/     CHXSP(AP) <-PB

/MC(3)*P(1)*RUN/          E <-0

Comment, fetch and execute the microinstruction at location 44

/P(3)*RUN*E'/             CAR <-countup CAR
```

```
/P(0)*RUN*E'/          F <- CM(CAR),E <-1

/P(1)*RUN*F(82)/       IF (CHANX1=ACTIVE) THEN CAR <-F(ADS)
                       ELSE (CPUIC <-SR(21-35),USER <-1)  $XFER to 41

/P(2)*RUN*F(0)/        E <-0

Comment, fetch and execute the microinstruction at location 45

/P(3)*RUN*E'/          CAR <-countup CAR

/P(0)*RUN*E'/          F <-CM(CAR),E <-1

/P(1)*RUN*F(83)/       IF (USER=1) THEN CAR <-F(ADS)       $XFER to 45

/P(2)*RUN*F(0)/        E <-0

Comment, fetch and execute the microinstruction at location 46

/P(3)*RUN*E'/          CAR <-countup CAR

/P(0)*RUN*E'/          F <-CM(CAR),E <-0

/P(1)*RUN*F(47)/       CAR <-F(ADS)

/P(2)*RUN*F(0)/        E <-0
```

4.3 Microprogram

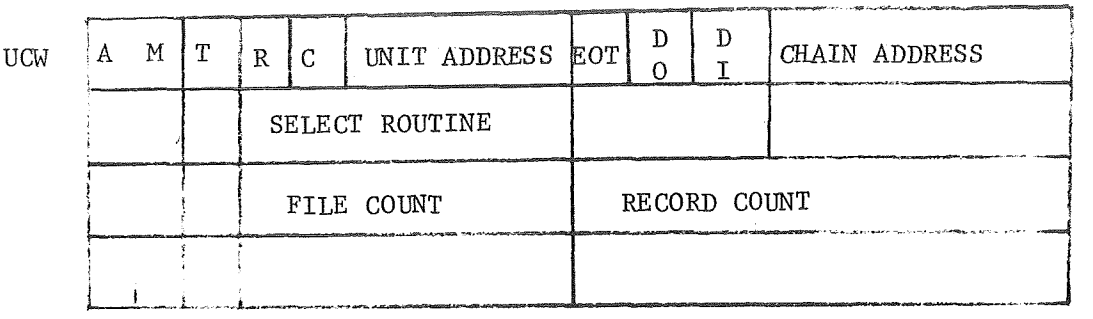
Figure 8 shows the octal microprogram representation to do the functions of IOEX.

The microprogram was obtained by seeing what bits of F are 1 and which are 0 during each control memory cycle.

5. Conclusions

From this study of a part of IOEX, it can be seen that micro-programming of a supervisory system is practical and desirable.

By adding just a little additional hardware to the CPU, communication can be achieved with the supervisor. In this manner, the problem programs will function the same with microprogramming as without. Where necessary, control is given to the CPU for further processing, thereby achieving the flexibility of software control and the speed of microprogramming.



A - Availability Flag

M - Attachment Flag

T - Unit Type

R - Reserve Status Flag

C - Channel Type

EOT - End of Tape Indicator

DO - Density at Load Point

DI - Density at Current Position

S - Select Type

R - Redundancy Message

N - Noise Record Flag

Figure 1 - Unit Control Block

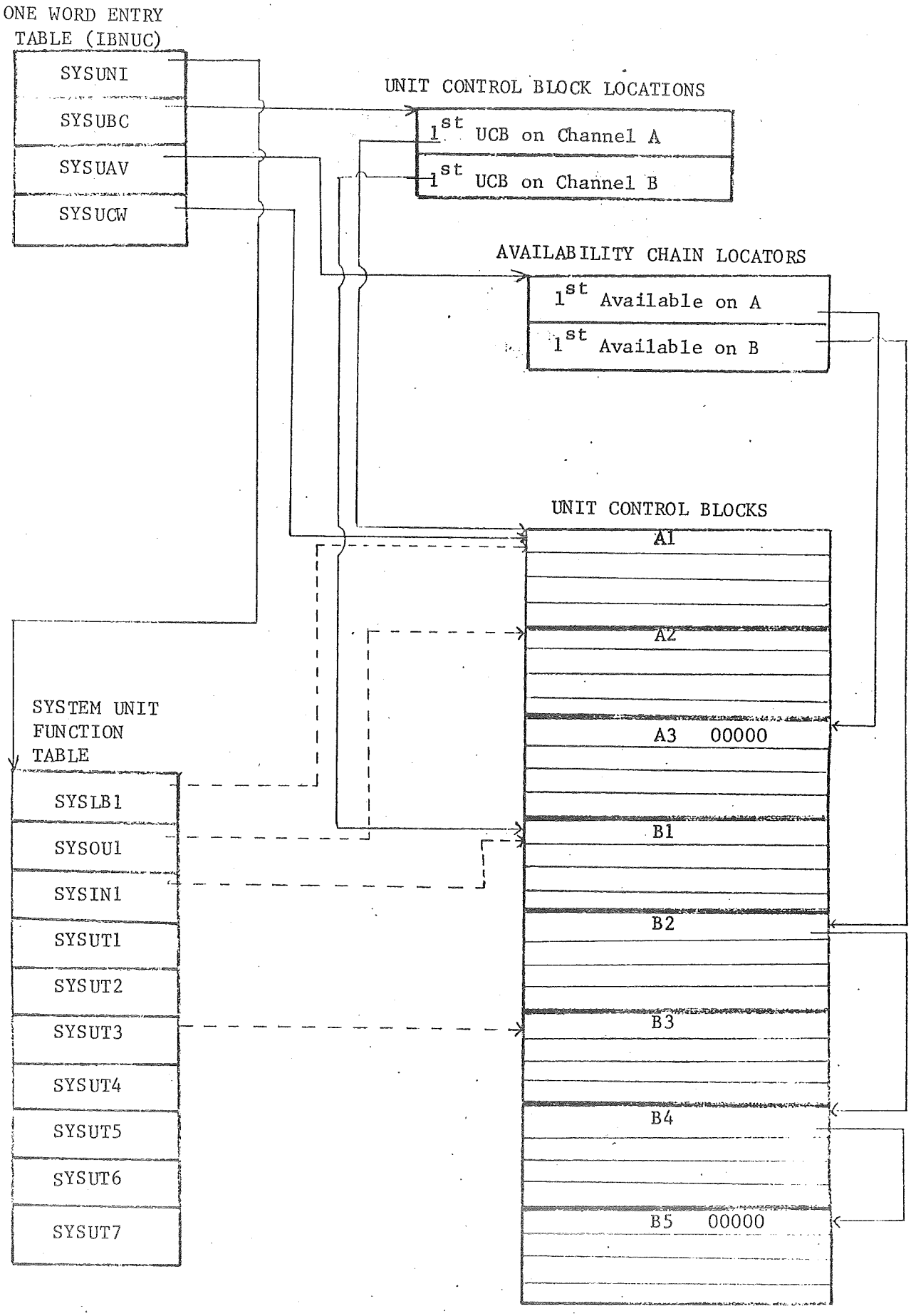


Figure 2 - Sample Nucleus Chaining After Cold Start

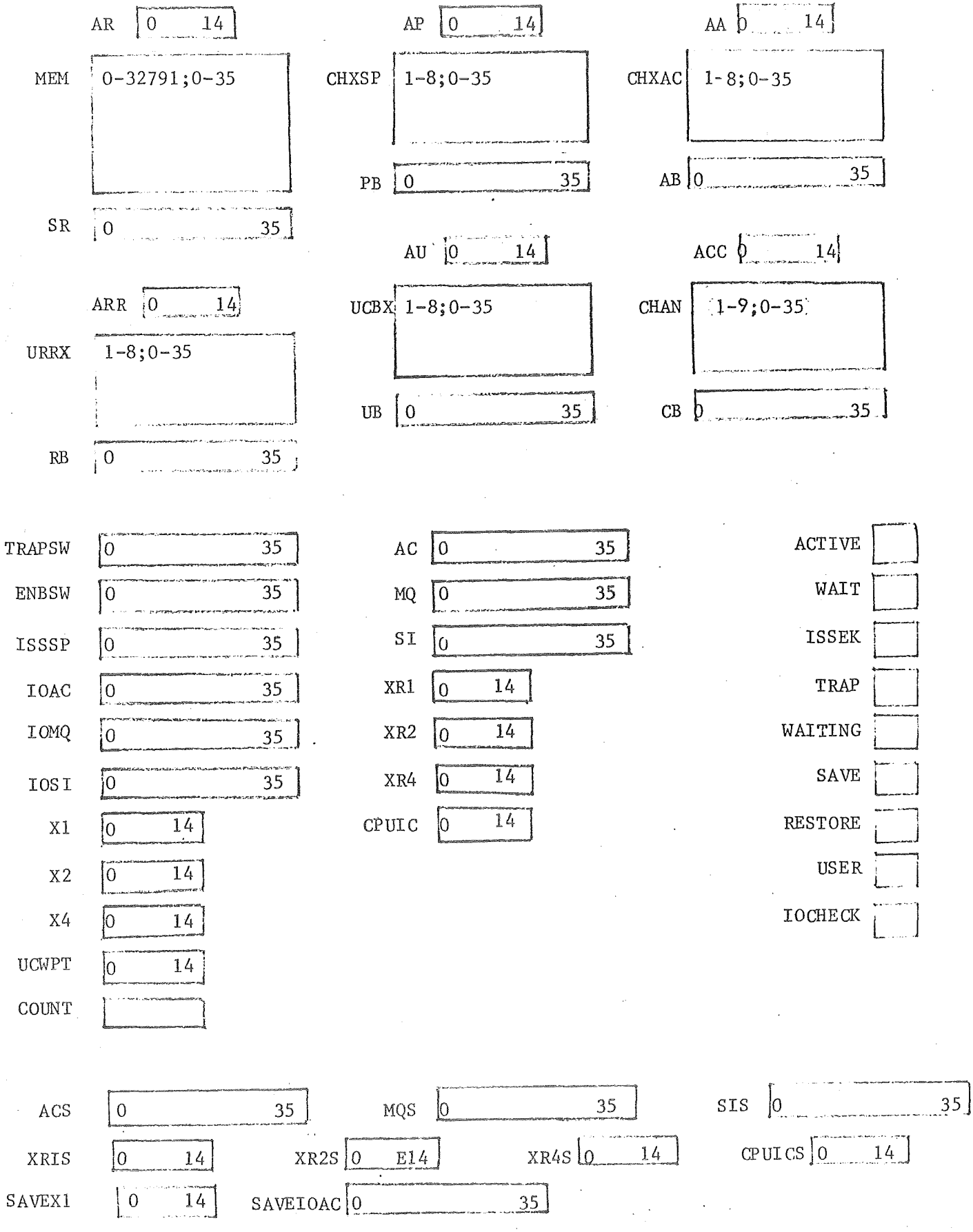


Figure 3 - IOEX Configuration

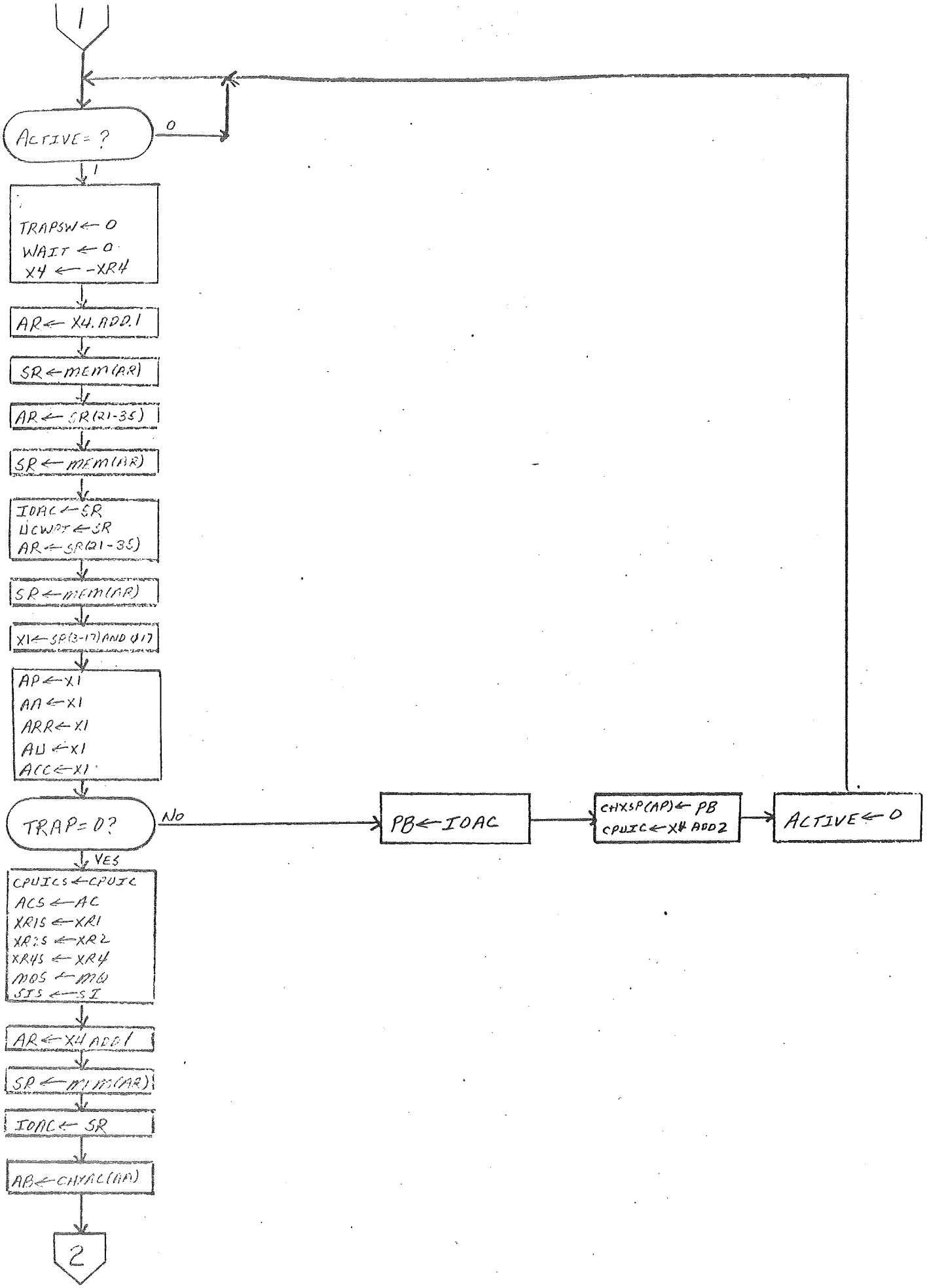


Figure 4a - Active Sequence Chart

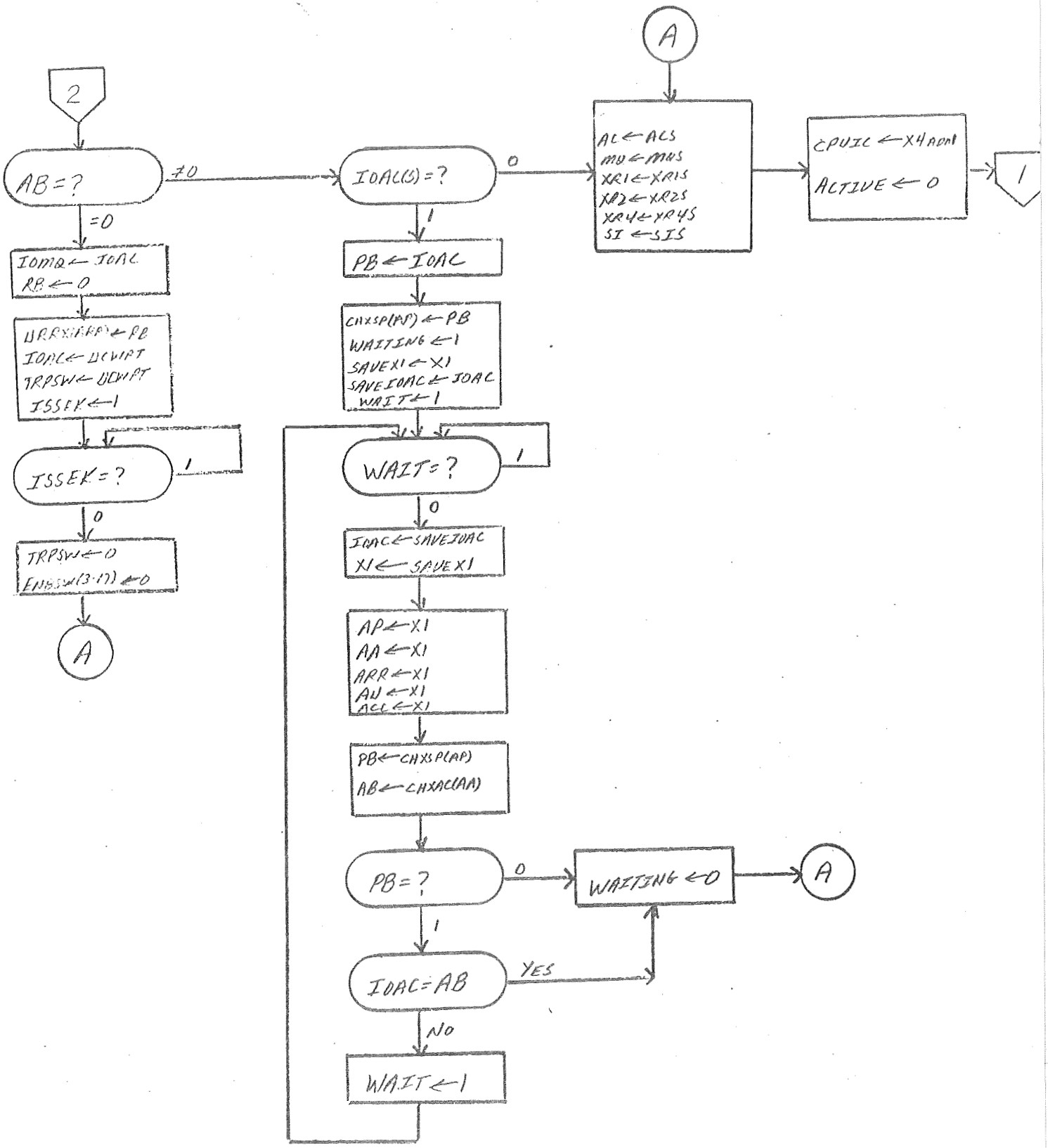


Figure 4b - Active Sequence Chart

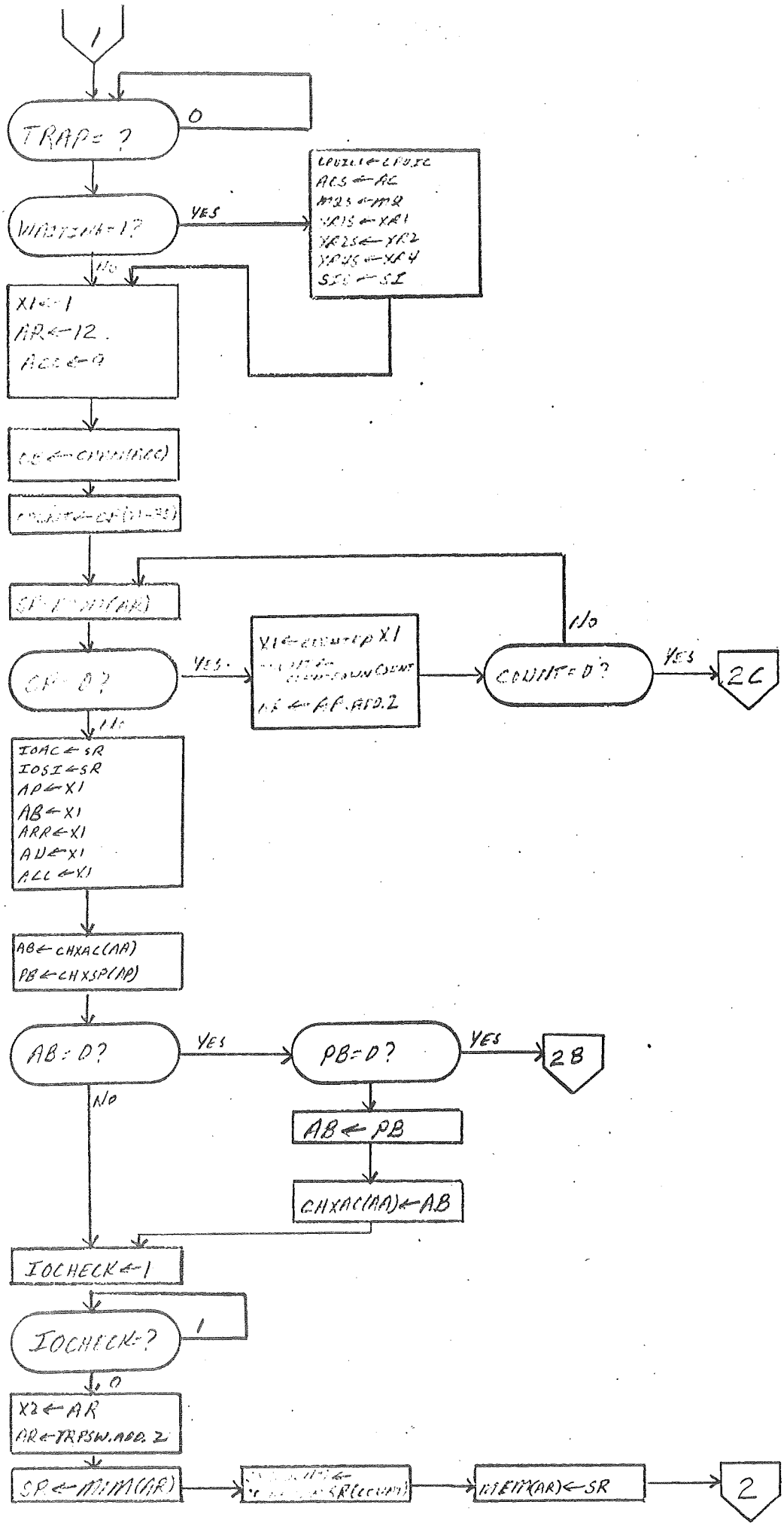


Figure 5a - Trap Sequence Chart

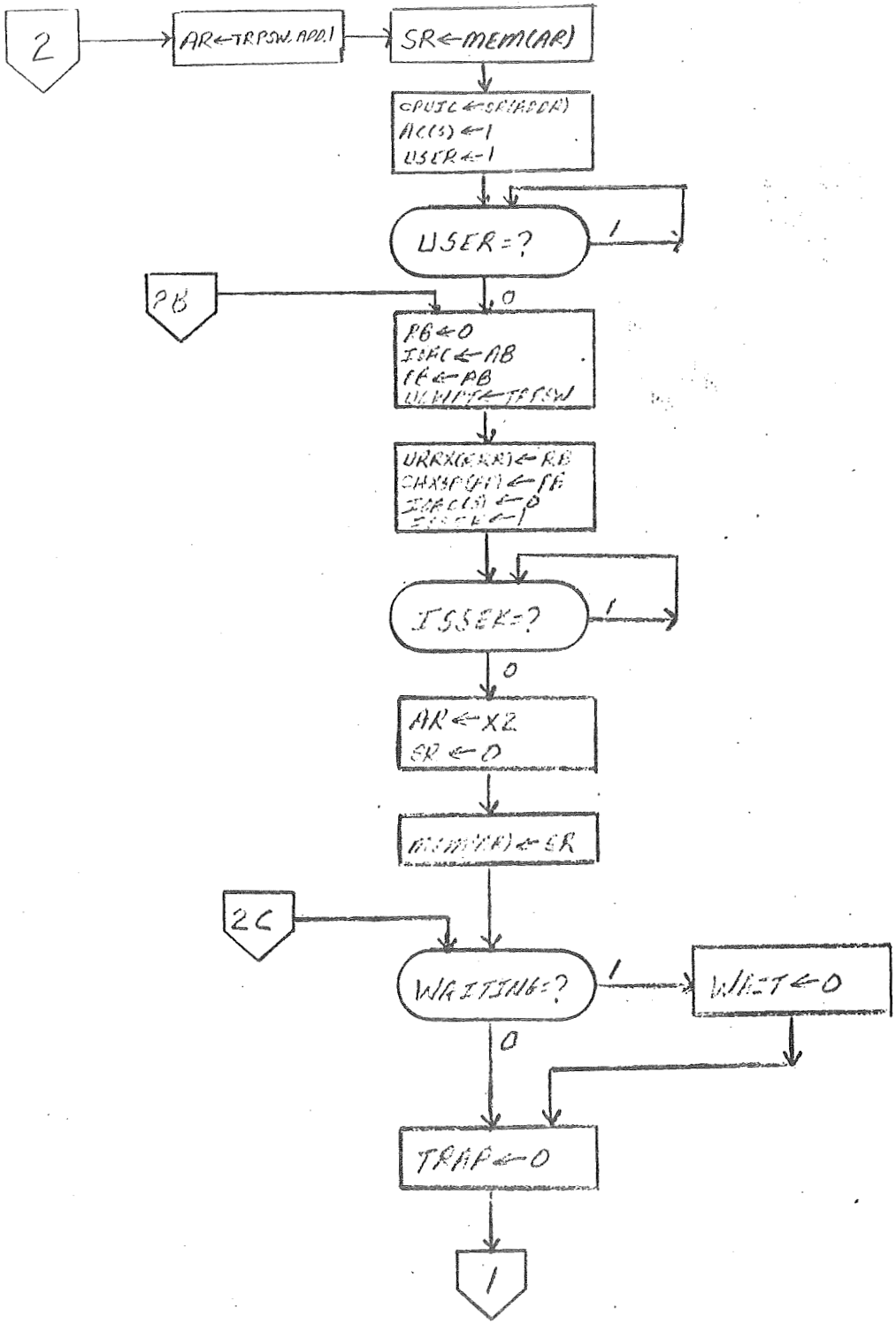


Figure 5b - Trap Sequence Chart

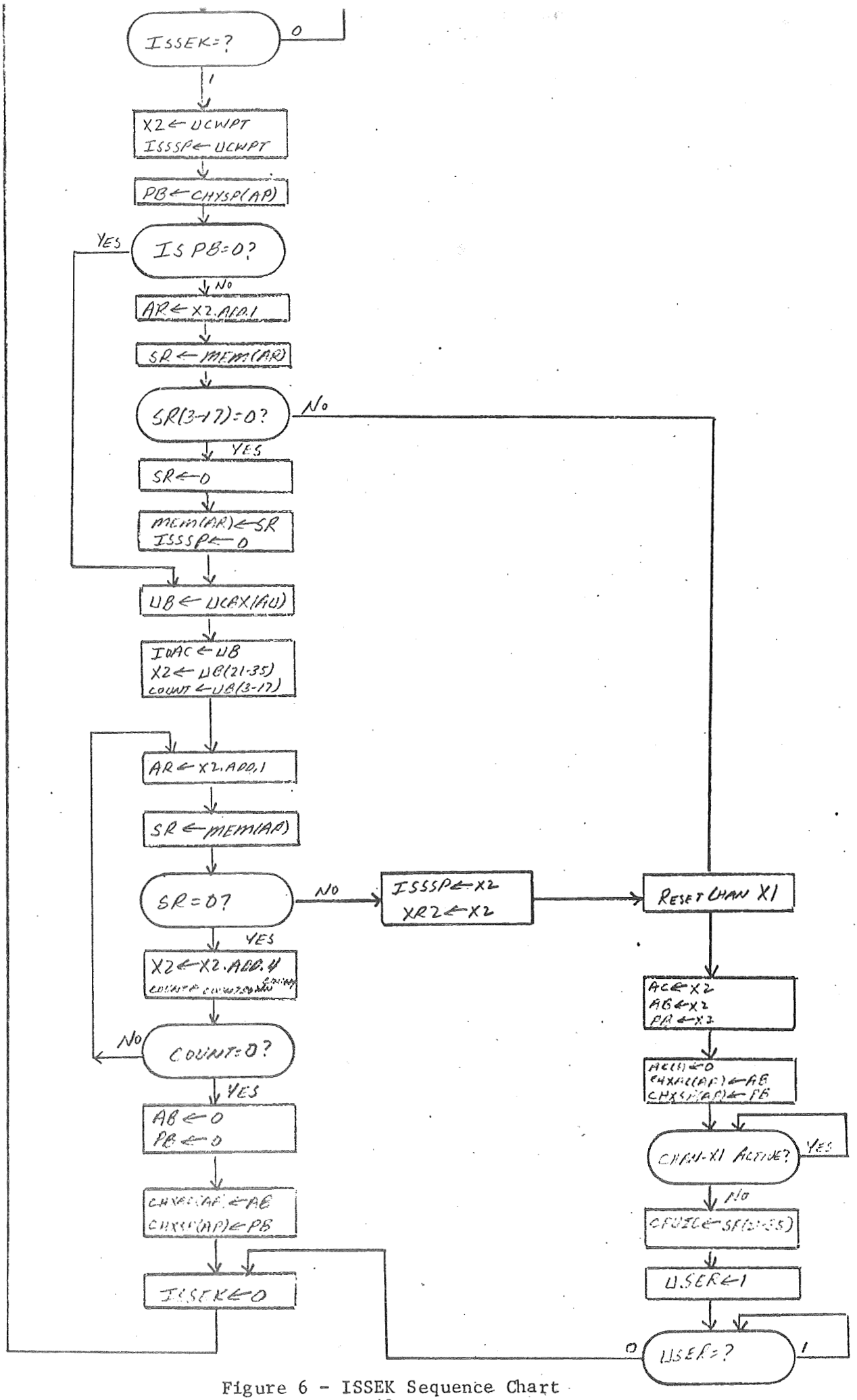


Figure 6 - ISSEK Sequence Chart

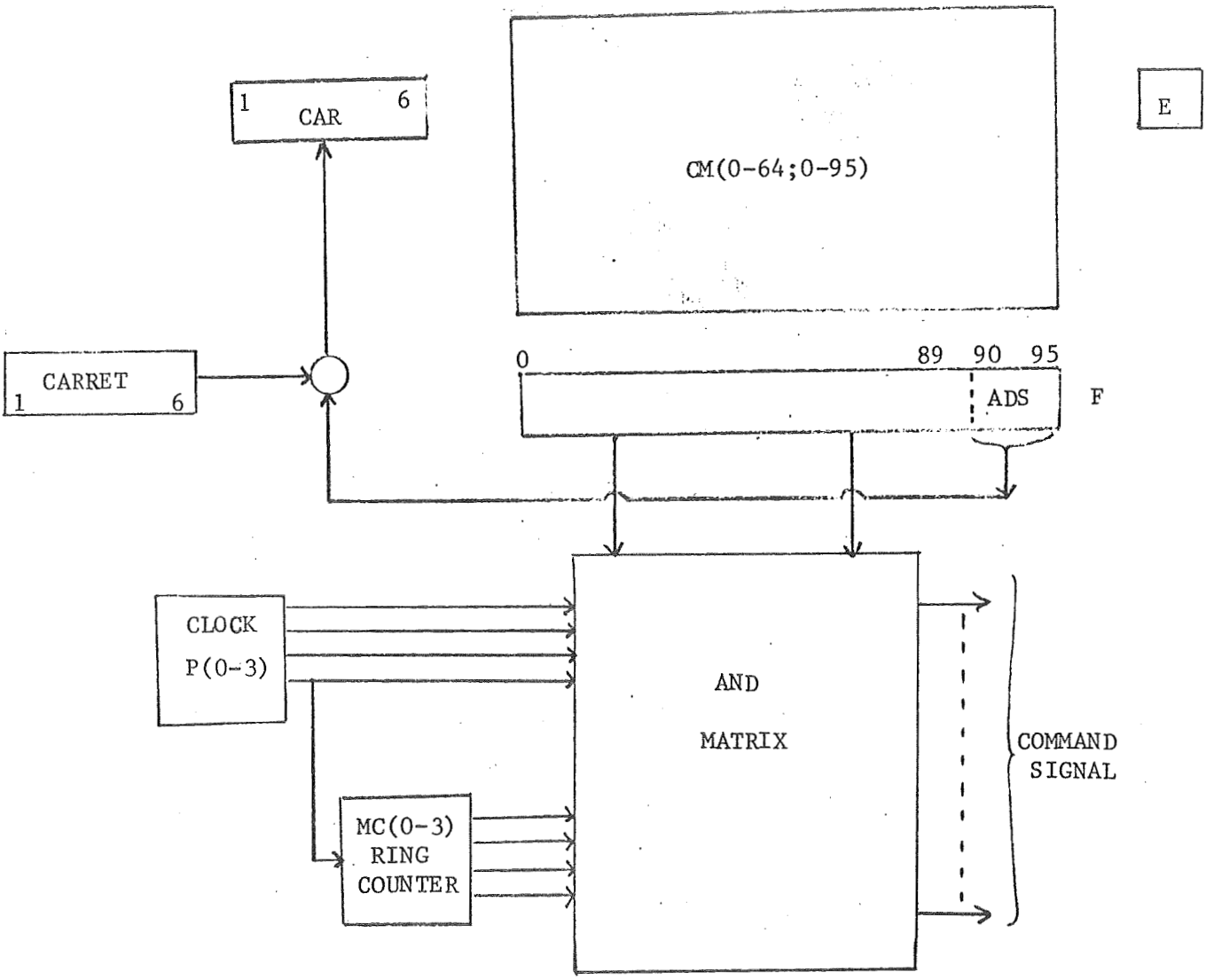


Figure 7 - Control Configuration

Control Memory Address	Microinstruction (Octal number)
0	20020000000000060000000004000000
1	20000000000100000000000000000000
2	20410000000000006000000000020000
3	40100400001000000000010000000004
4	00400200000004000000040000000000
5	20020000000000400000000000000000
6	10000000100000000000000000000000
7	40000000000000000000004000000011
8	40000000000000000000002000000013
9	00000000000006000000040000000000
10	00200100040000211400000000000037
11	00000000000006002200004000000000
12	00400200000000001700040000000000
13	54000400020400000000000000000010
14	00000000000000000000001000000000
15	02010000000000000000000020000000
16	40000000000001000000000000000000
17	20000000000000000000000000000000
18	40000000000000000000000400000024
19	40000000000000000000000200000020
20	00000090000000000000000004010014
21	14000400100000000004000000000000
22	40000000000000000000000040000030
23	40000000000000000000000100000034
24	00040040000000000000000000000000
25	40000000000000030002000000000000
26	20004000000100000000000000000000
27	00100020000000000000000000000000
28	40000000000001000001400000000000
29	40000000000000000000000001000034
30	00200110010000304002100000000037
31	00102004000000010000000000000000
32	04000000000040000000100000000000
33	40000000000000000000000100000045
34	20020000000000000000000000000000
35	40000000000000000000000002000052
36	00100004000000000000400000000000
37	10000000000000000000000000000000
38	40000000002020040000000000000000
39	20000002000000000000000001400052
40	40000000000000000000000000000000
41	00440000000000000000000000000000
42	40000000000000000000002000040000
43	00440000200000000000300000000000
44	40000000000000000000000002000050
45	40000000000000000000000010000054
46	00000000000000001000000000000000

Figure 8- The Microprogram

CONTROL WORD FORMAT
TABLE 1

BIT	MICROINSTRUCTION
0	E ←-0
1	SR ←-MEM(AR)
2	AB ←-CHXAC(AA)
3	PB ←-CHXSP(AP)
4	CB ←-CHAN(ACC)
5	UB ←-UCBX(AU)
6	CHXSP(AP) ←-PB
7	URRX(ARR) ←-RB
8	MEM(AR) ←-SR
9	CHXAC(AA) ←-AB
10	AR ←-X4.ADD.1
11	AR ←-SR(21-35)
12	AR ←-TRPSW.ADD.1
13	AR ←-X2
14	AR ←-10, ACC ←-9, X1 ←-1
15	AP ←-X1, AA ←-X1, ARR ←-X1, AU ←-X1, ACC ←-X1
16	PB ←-IOAC
17	RB ←-0
18	AB ←-PB
19	SR(COUNT) ←-countupSR(COUNT)
20	PB ←-AB
21	SR ←-0
22	AR ←-X2.ADD.1
23	AB ←-0
24	PB ←-0
25	AC ←-X2, AB ←-X2, PB ←-X2
26	IOAC ←-SR
27	IOAC ←-UCWPT
28	IOAC ←-SAVEIOAC
29	IOAC ←-AB
30	IOAC(S) ←-0
31	IOAC ←-UB
32	X1 ←-SR(3-17).AND.017
33	X1 ←-SAVEX1
34	X1 ←-countupX1, COUNT ←-countdownCOUNT, AR ←-AR.ADD.2
35	X2 ←-AR
36	X2 ←-UCWPT
37	X2 ←-UB(21-35)
38	X2 ←-X2.ADD.4
39	CPUIC ←-X4.ADD.1
40	AC ←-ACS, XR1 ←-XR1S, XR2 ←-XR2S, XR4 ←-XR4S, SI ←-SIS, MQ ←-MQS
41	CPUIC ←-SR(ADDR)
42	CPUICS ←-CPUIC, ACS ←-AC, XR1S ←-XR1, XR2S ←-XR2, XR4S ←-XR4, SIS ←-SI, MQS ←-MQ
43	IOMQ ←-IOAC
44	COUNT ←-CB(21-35)
45	COUNT ←-UB(3-17)
46	CARRET ←-CAR
47	CAR ←-F(ADS)
48	WAIT ←-0

BIT	MICROINSTRUCTION
49	TRPSW ←-0
50	TRPSW ←-UCWPT
51	ISSEK ←-1
52	ENBSW(3-17) ←-0
53	WAITING ←-1
54	SAVEX1 ←-X1
55	SAVEIOAC ←-IOAC
56	WAIT ←-1
57	IOSI ←-SR
58	IOCHECK ←-1
59	AC(S) ←-1
60	USER ←-1
61	UCWPT ←-TRPSW
62	ISSSP ←-UCWPT
63	ISSSP ←-0
64	RESET CHAN-X1
65	AC(S) ←-0
66	GO ←-1, ACTIVE ←-0
67	ISSEK ←-0
68	IF(TRAP=0) THEN CAR←F(ADS)
69	IF (AB=0) THEN CAR←F(ADS)
70	IF (IOAC(S)=1) THEN CAR←F(ADS)
71	IF(PB=0.OR.AB=IOAC)THEN(WAITING ←-0, CAR ←-F(ADS))ELSE (WAIT ←-1, GO ←-1, ACTIVE ←-0)
72	IF(SR=0)THEN CAR ←-F(ADS) ELSE X1 ←-countupX1, COUNT ←-countdownCOUNT, AR ←-AR.ADD.2
73	IF(COUNT=0)THEN CAR ←-F(ADS), E ←-0
74	IF(PB=0) THEN CAR ←-F(ADS)
75	IF(AB=0) THEN CAR ←-F(ADS)
76	IF(WAITING=1)THEN(CPUICS←-CPUIC, ACS ←-AC, MQS ←-MQ, XR1S ←-XR1, XR2S ←-XR2, XR4S ←-XR4, SIS ←-SI)
77	IF(USER=1)THEN CAR←F(ADS)
78	IF(WAITING=1)THEN(WAIT ←-0, CAR ←-F(ADS))
79	IF(SR(3-17)=0)THEN CAR ←-F(ADS)
80	IF(SR=0)THEN(ISSSP ←-X2, XR2 ←-X2, CAR ←-F(ADS))ELSE (X2 ←-X2.ADD.4, COUNT ←-countdown COUNT)
81	IF(COUNT=0) THEN CAR ←-F(ADS)ELSE AB ←-PB
82	IF (CHANX1=ACT) THEN CAR ←-F(ADS)ELSE(CPUIC ←-SP(21-35), USER ←-1)
83	IF(USER=1) THEN CAR ←-F(ADS)
84	CAR ←-CARRET
85	UCWPT ←-SR
86	GO ←-1, TRAP ←-0
87	X4 ←-XR4

REFERENCES

IBM 7090/7094 IBSYS Operating System-System Monitor, Form C28-6248.

IBM 7090/7094 IBSYS-IOEX Programming Systems Analysis Guide, form C28-6299.

IBM 7090 Principles of Operation, form A22-6528-6.

Rosen, Saul ed. Programming Systems and Languages. McGraw Hill, 1967.

Y. Chu, O. Pardo, J. Yeh, "A Methodology for Unified Hardware Software Design", Tech Report 70-107, Computer Science Center, U. of Maryland, January 1970.

Y. Chu, "A Higher-order Language for Describing Microprogrammed Computers", Tech Report 68-78, Computer Science Center, U. of Maryland, September 1968.

Appendix

This appendix presents a set of three flow charts which describe the ACTIVE, TRAP and ISSEK subroutines of the IOEX.

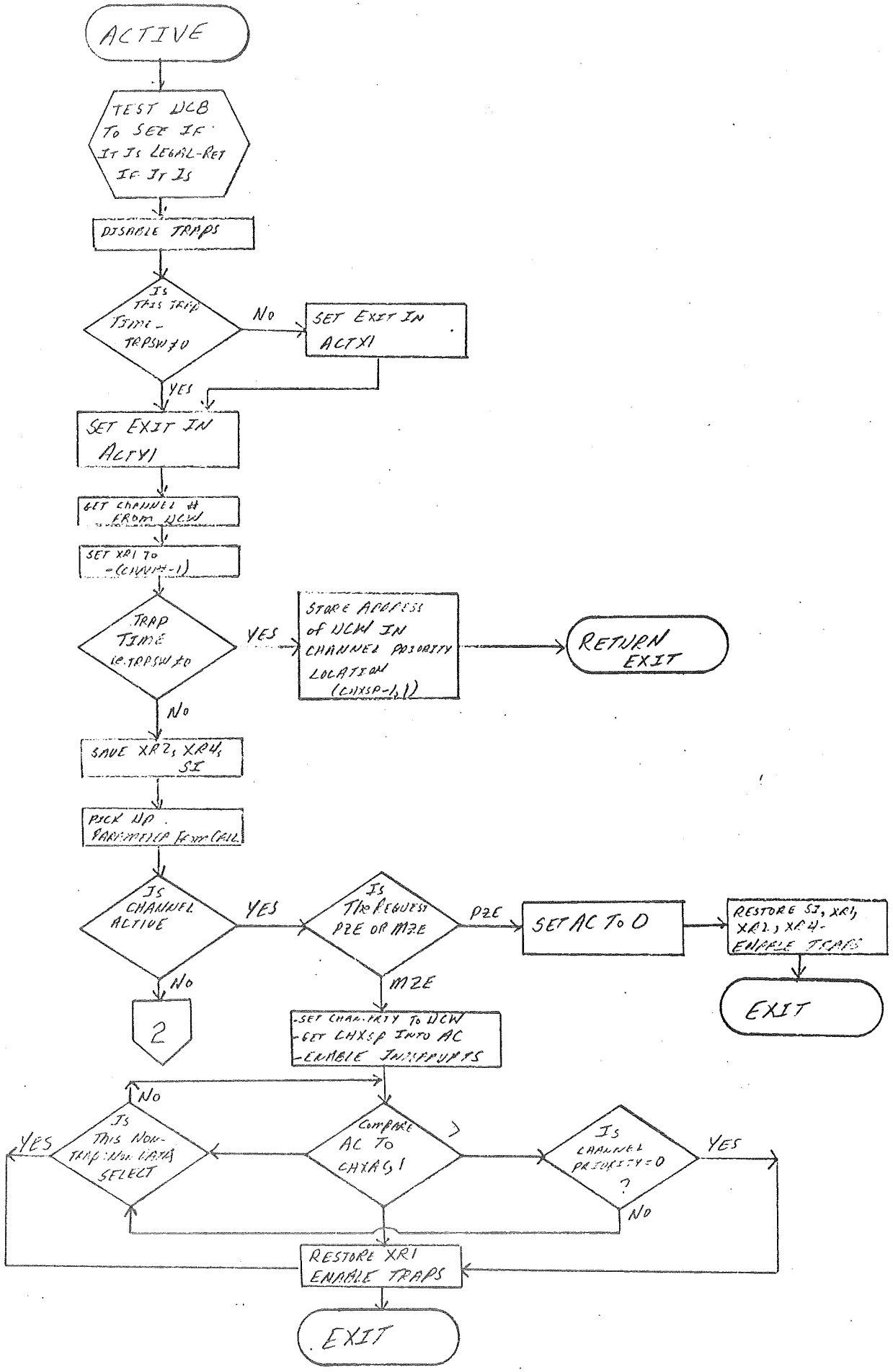


Figure A1a-Active Subroutine, Logical Flow Chart

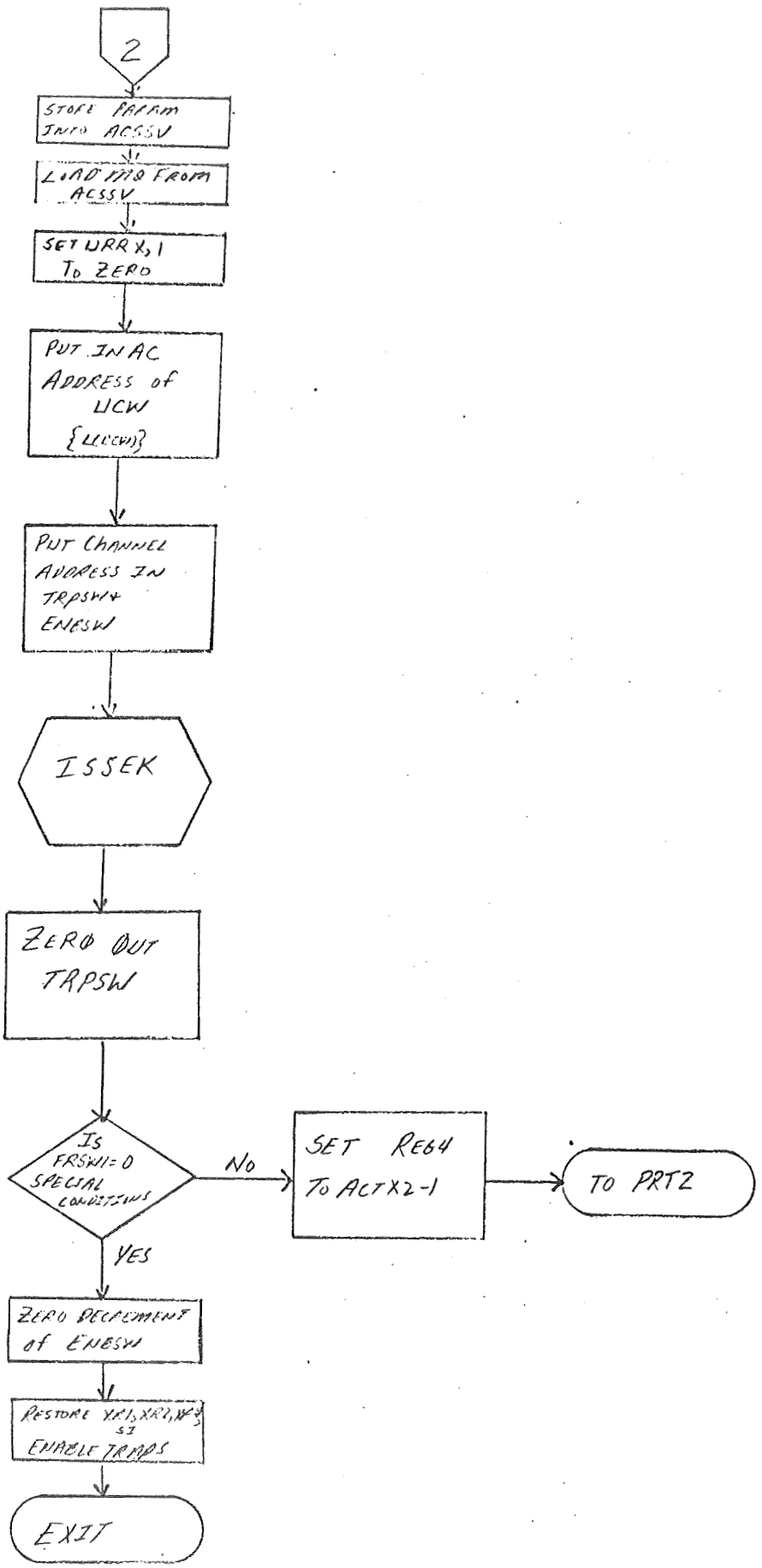


Figure A1b-Active Subroutine, Logical Flow Chart

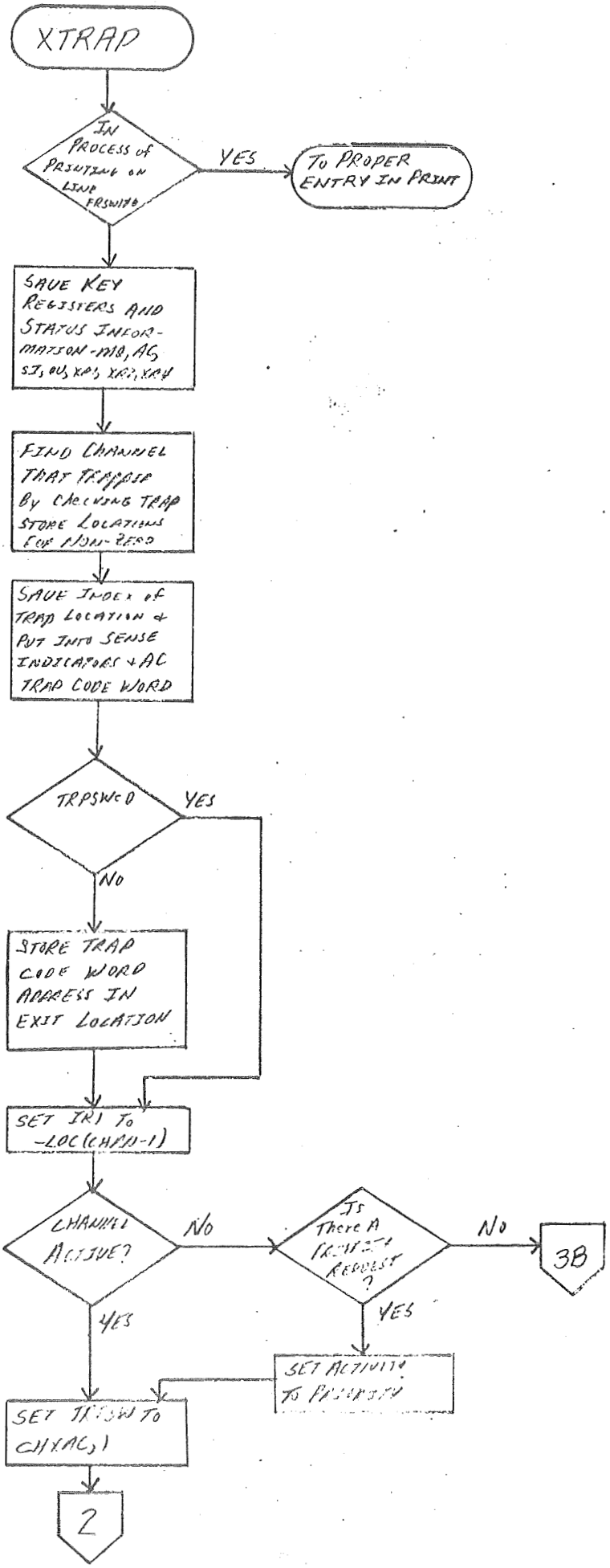


Figure A2a-Trap Supervisor, Logical Flow Chart

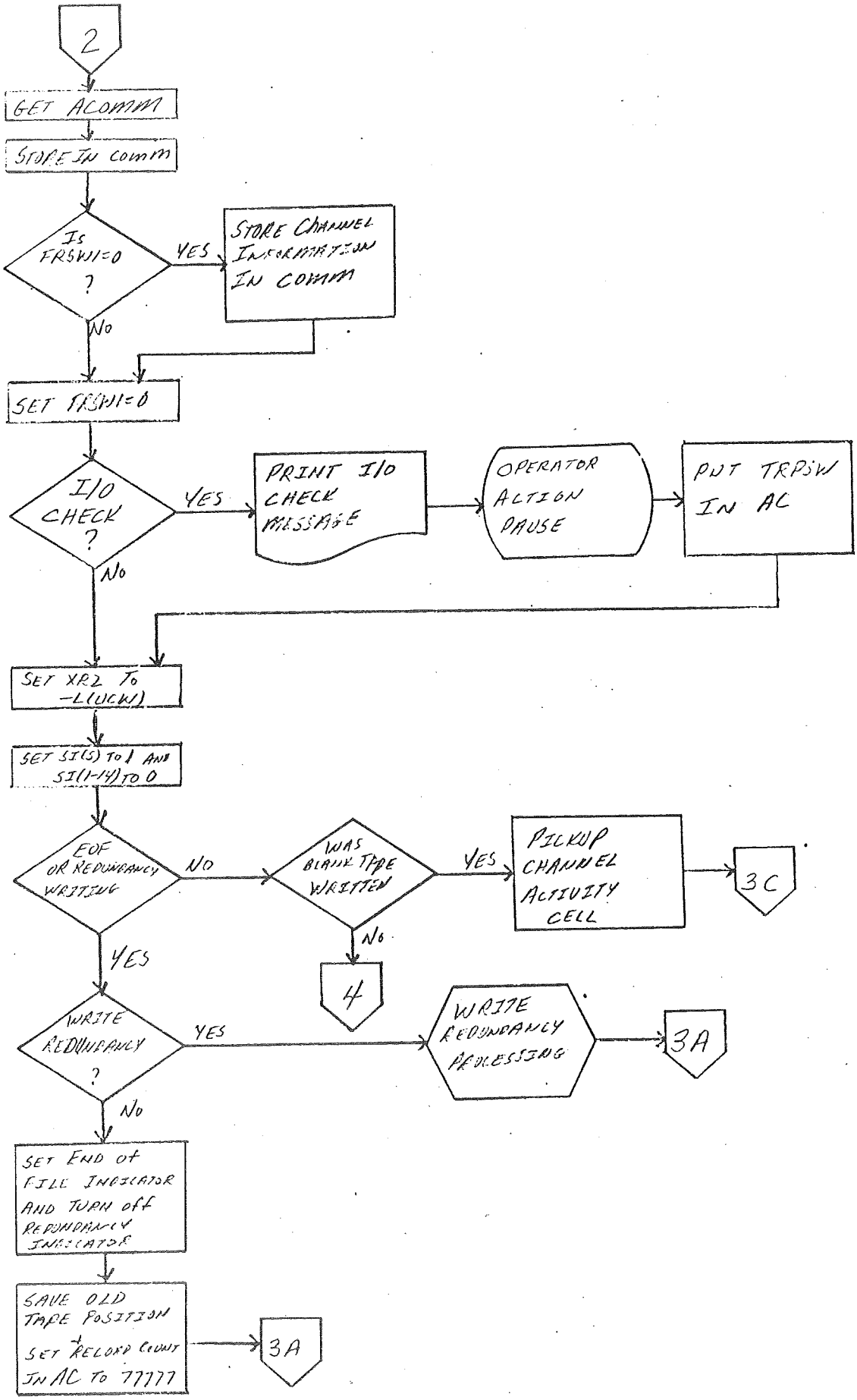


Figure A2b-Trap Supervisor, Logical Flow Chart

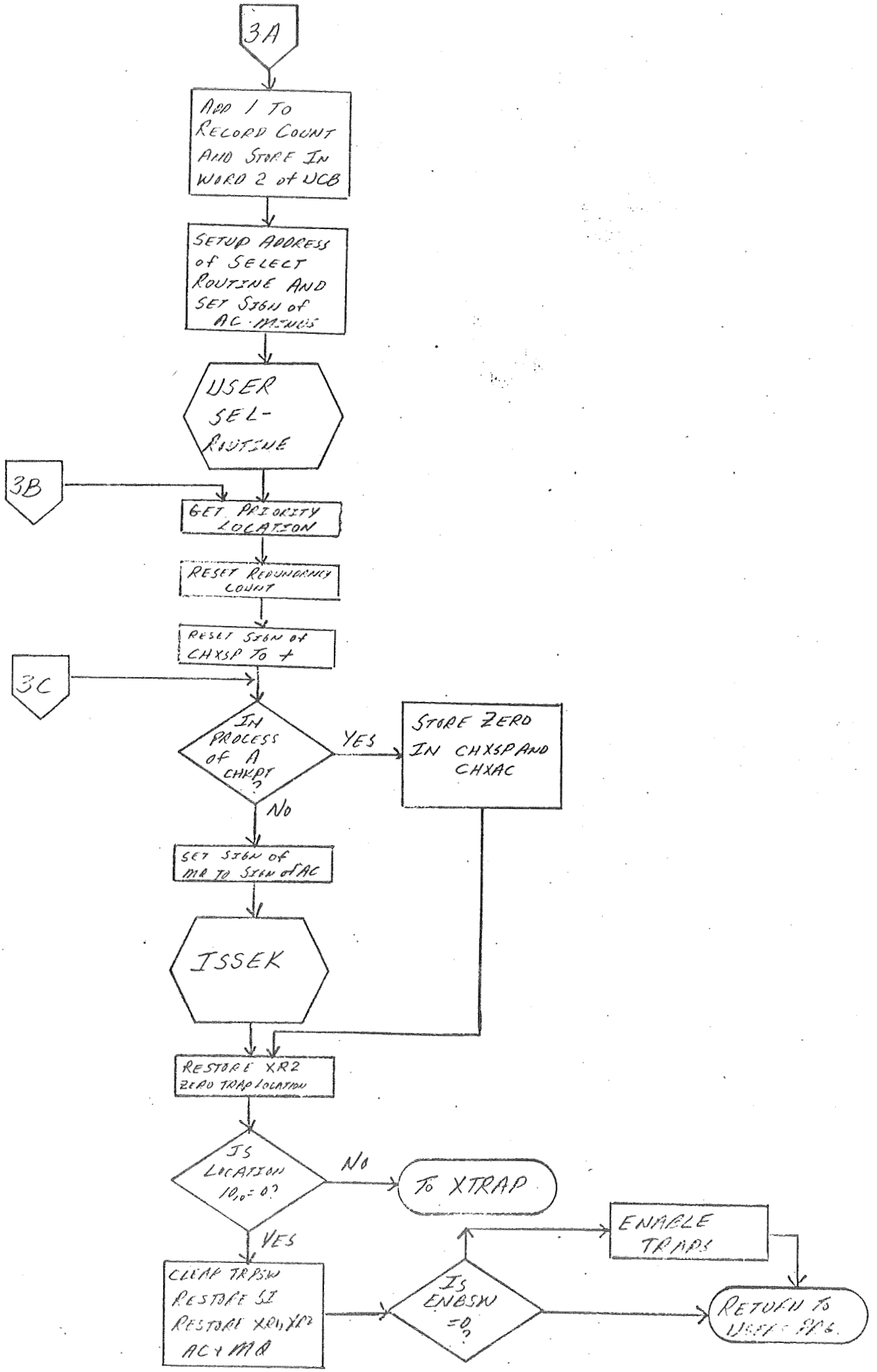


Figure A2c-Trap Supervisor, Logical Flow Chart

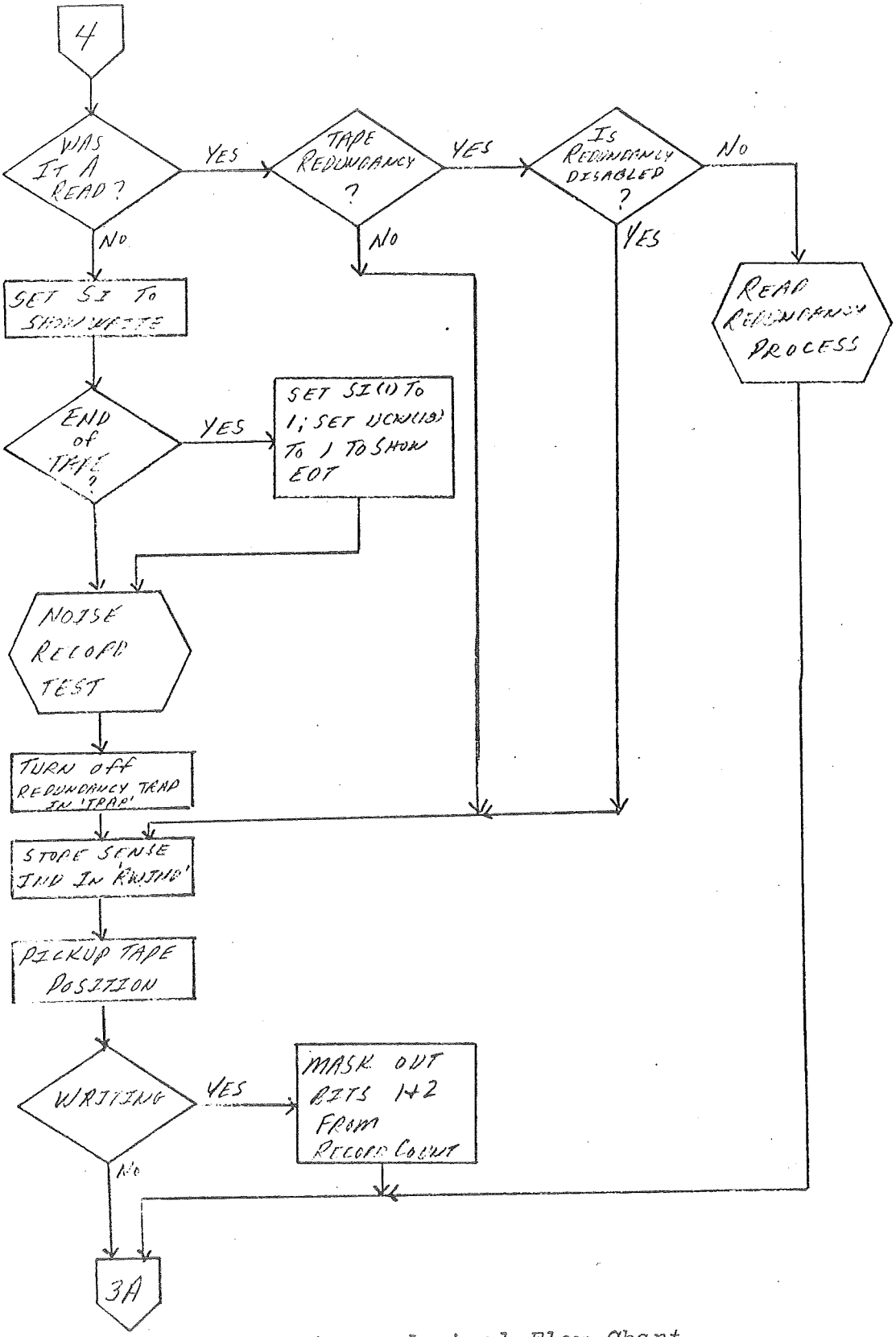


Figure A2d-Trap Supervisor, Logical Flow Chart

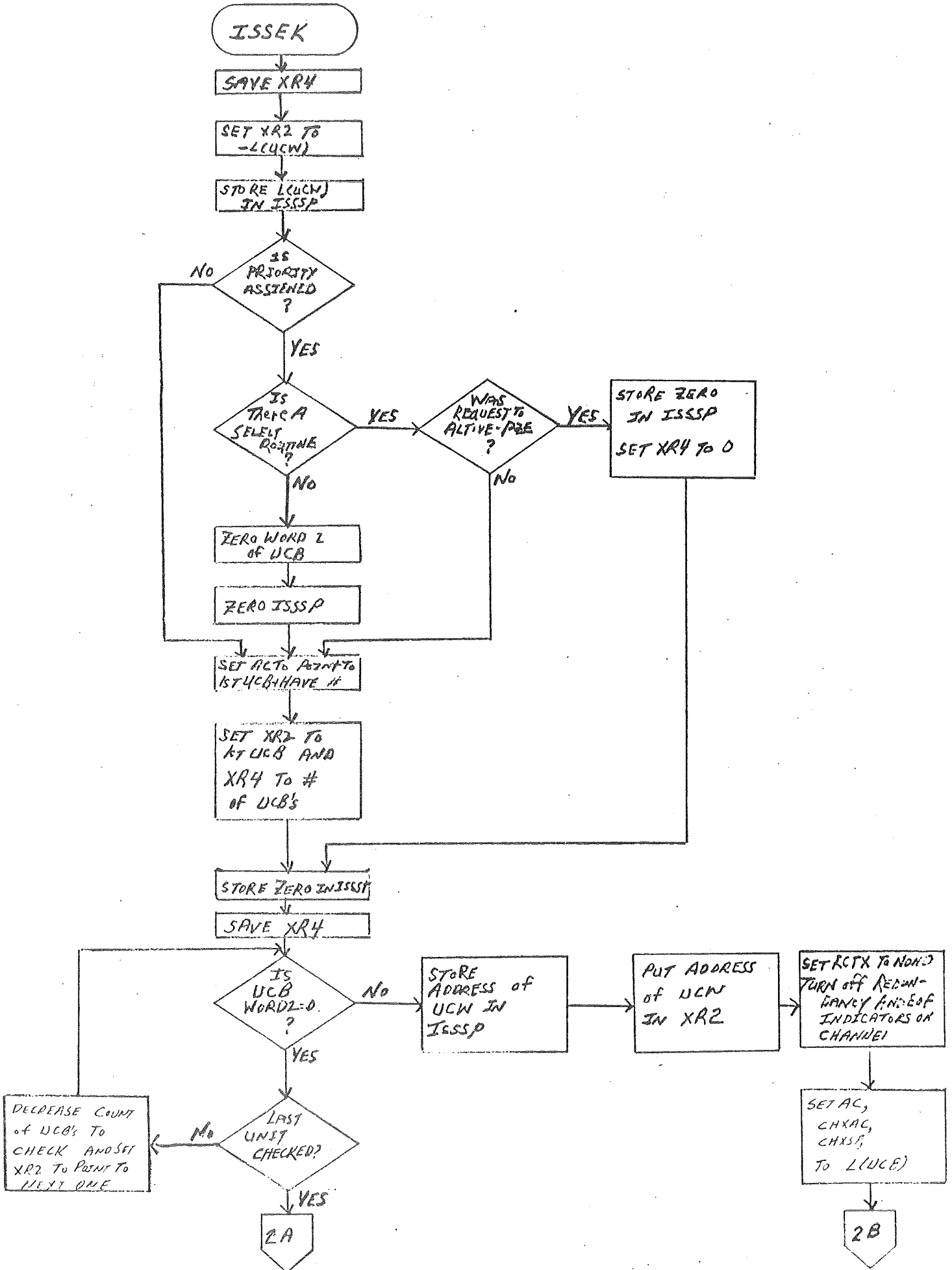


Figure A3a-Issek Subroutine, Logical Flow Chart

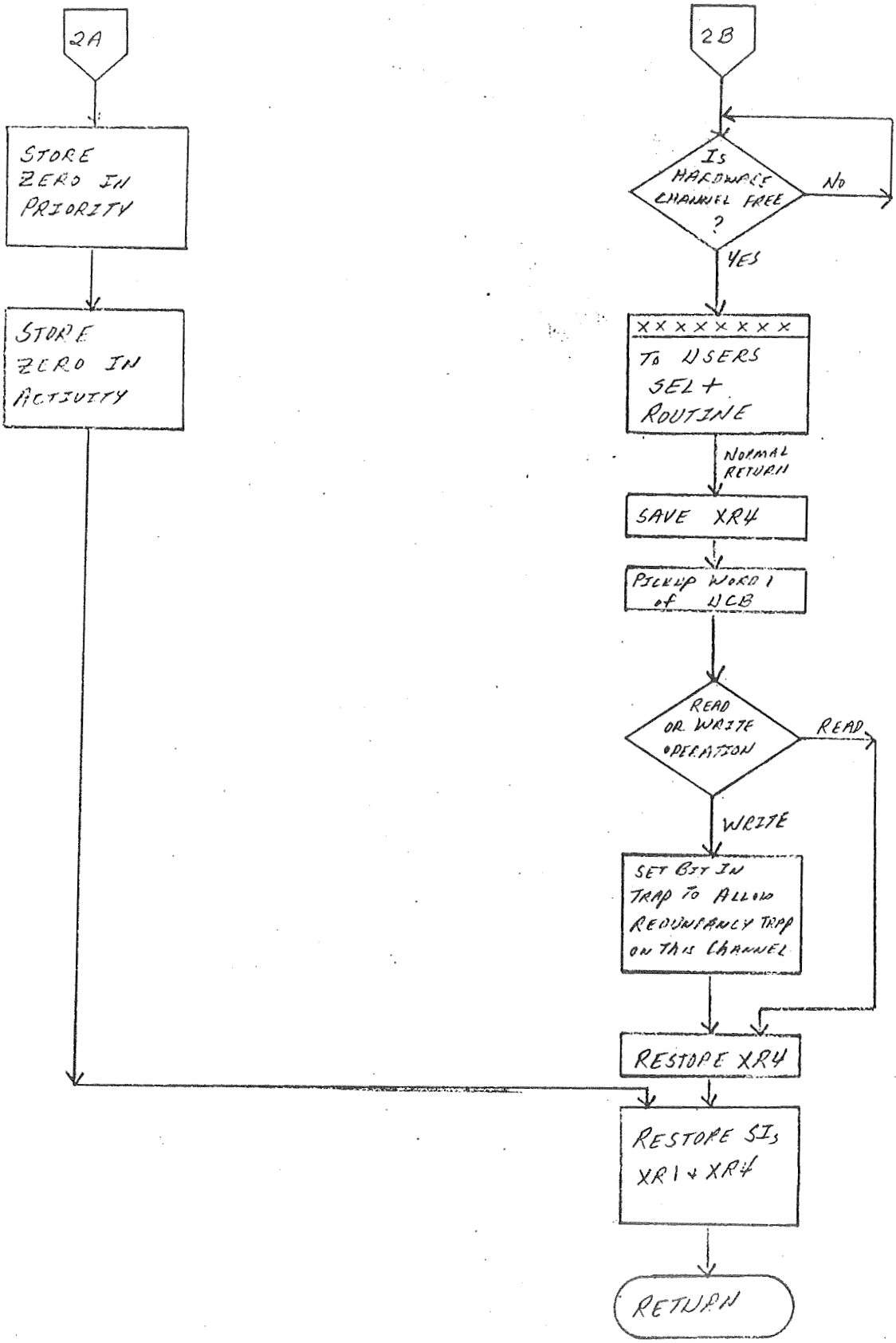


Figure A3b-Issek Subroutine, Logical Flow Chart