MASSACHUSETTS INSTITUTE OF TECHNOLOGY

APOLLO

GUIDANCE, NAVIGATION AND CONTROL

Approved: Eldon C. Hall Date: 2/2/67
ELDON C. HALL, DIRECTOR, DIGITAL DEVL.
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: David G. Hoag Date: 6 Feb 67
DAVID G. HOAG, DIRECTOR
APOLLO GUIDANCE AND NAVIGATION PROGRAM

Approved: Ralph R. Ragan Date: 7 Feb 67
RALPH R. RAGAN, DEPUTY DIRECTOR
INSTRUMENTATION LABORATORY

E-2065

[redacted]

BLOCK II AGC SELF-CHECK AND SHOW-BANKSUM

by

Edwin D. Smally

December 1966

[redacted]

INSTRUMENTATION LABORATORY

CAMBRIDGE 39, MASSACHUSETTS

COPY # 60

ACKNOWLEDGEMENT

This report was prepared under DSR Project 55-23850, sponsored by the Manned Spacecraft Center of the National Aeronautics and Space Administration through Contract NAS 9-4065.

ADDENDUM 1

E-2065

BLOCK II AGC SELF-CHECK AND SHOW-BANKSUM

by

E. D. Smally
December 1966

The SELF-CHECK routine as found in Sunburst 206 is essentially as described in E-2065 with the exceptions noted below.

A. Initialization of SELF-CHECK Reserved Erasable

Fresh Start does not clear SFAIL or ERCOUNT. ERESTORE is cleared by Fresh Start. Restart does not clear **SFAIL.**

It may be desirable to clear ERCOUNT, by keyboard, after doing Verb Fresh Start.

B. Alarm Display

A SELF-CHECK initiated program alarm turns on the program alarm light and displays the FAILREG set (Noun 50). The FAILREG set, (FAILREG, FAILREG +1, FAILREG +2), displays the alarm codes of the first, next to last, and last program failure. The alarm code for SELF-CHECK is still 01102. If additional information is desired, the operator may display Noun 31, the ALMCADR set, (ALMCADR, ALMCADR +1, ERCOUNT), The contents of ALMCADR, if SELF-CHECK was the last failure, is equal to 1 + address of the failure (the contents of SFAIL), ALMCADR +1 the contents of BBANK for SELF-CHECK (76002), and ERCOUNT the number of SELF-CHECK failures since ERCOUNT was last cleared. If the contents of ALMCADR +1 is not 76002, due to an intervening failure by a program other than SELF-CHECK, the contents of SFAIL (machine address 01357) could then be displayed directly.

C. ERASCHK

This part of SELF-CHECK makes sure that it is possible to read a "1" and a "0" into and out of each bit position of erasable memory.

In the event that a RESTART occurs in the midst of ERASCHK, the RESTART subroutine does a FRESH START if a set of erasable registers were being checked. The reason for the FRESH START is that ERASCHK has just contaminated two erasable registers, and the EBANK information needed to identify and restore them has been destroyed by the lead in to the RESTART program (this problem has been corrected in the 258 mission program). The RESTART program tests register ERESTORE and if ZERO, proceeds with RESTART, otherwise it goes to DOFSTART (program FRESH START).

The non-special erasable registers are checked for correct addressing and content by placing their own address in two successive registers and making sure there is a difference of -1 when the contents of the lower address register is added to the complement of the higher address register; if it is not, this subroutine performs a TC to the PRERRORS subroutine, The previous contents of the erasable registers had been preserved and are restored to the two registers by PRERRORS which then performs a TC to the Errors subroutine.

If the difference is -1, the contents of the two registers are complemented and the complement of the lower register added to the contents of the higher register; the result is checked for -1. If the result is not -1, TC to PRERRORS as noted above. If the result is -1, restore the previous contents to the two registers, and proceed to the next iteration. The higher address register of the past iteration becomes the lower address register of the next iteration. The erasable memory banks are checked from zero through seven with common erasable (60-1374) being checked after each erasable bank.

D. Future Reassemblies

Future reassemblies of Sunburst will not be protected by cuss if a bank is loaded to the point of not leaving room for the two TC selfs normally preceeding the Bugger word. The 206 version of SELF-CHECK requires the two TC selfs in order for ROPECHK or SHOWSUM to work with the resulting program. (The 258 version of SELF-CHECK does not require the two TC selfs for ROPECHK or SHOWSUM to work.)

E-2065

BLOCK II AGC SELF-CHECK AND SHOW-BANKSUM

ABSTRACT

This report is in two main sections. The first section contains the operating procedures to be utilized by persons using the SELF-CHECK or SHOW-BANKSUM routines. It also has block diagram flow charts which should help explain how the operating procedures of SELF-CHECK may be used for diagnostic purposes. The procedures for SELF-CHECK are slightly different in BLOCK I and BLOCK 11 while the procedures for SHOW-BANKSUM are the same.

The second sectionof this report goes into an explanation of SELF-CHECK and SHOW-BANKSUM. The explanation of SELF-CHECK consists of an explanation of the computer internal selfcheck and an explanation of the check of the DSKY electroluminescents. There is a separate description of each subroutine in SELF-CHECK and SHOW-BANKSUM. There is also a separate flow chart, located in the appendix, for each subroutine. This section should prove helpful to field engineers in locating the cause of malfunctions in the computer.

All numbers in this report are octal unless specifically mentioned otherwise.

The two subroutines that check the multiply and divide arithmetic functions of the computer will be removed from flight ropes. Figure 1 shows that placing a ±6 or ±7 in the SMODE register will allow the computer to loop in either the arithmetic multiply or arithmetic divide subroutines if these subroutines are part of SELF-CHECK. Placing a ±6 or ±7 in the SMODE register will exercise the internal computer self-check when these two subroutines are removed from SELF-CHECK. Thus, ±6, ±7, or ±10 all perform the same function when thearithmetic multiply and arithmetic divide are removed from SELF-CHECK.

by Edwin D. Smally

December 1966

TABLE OF CONTENTS

PART I

OPERATING PROCEDURES FOR SELF-CHECK AND SHOW-BANKSUM

CHAPTER 1

SELF-CHECK OPERATING PROCEDURES

There are 19 possible options in this BLOCK II version of SELF-CHECK. These options are explained farther on in this report. The first 18 options are used to check the internal operation of the computer (±0 to ±10) while the 19th option (±11) checks the electroluminescent displays on the DSKY. It is felt that most people will use the options associated with ±10 or -zero since all three of these options perform a complete internal self-check of the computer, however, these three options perform different diagnostic functions when an error is detected. The options associated with ±1 to ±7 check out various parts of the computer and will be useful for field engineers or other personnel interested in diagnostic testing of the computer.

The normal use of SELF-CHECK is as a backup routine to check the computer continuously when the computer is not busy with other routines. The ±10 or -zero options can be used for this purpose.

Options Available in SELF-CHECK

The different options of BLOCK II SELF-CHECK are controlled by putting different numbers in the SMODE register (normally during the SELF-CHECK start procedure); this is the same as BLOCK I. However, it should be noted that the options are not the same in the BLOCK I and BLOCK II computers.

Placing a +0 in the SMODE register forces the computer to go into the backup idle loop where it continuously looks for a new job.

Placing a ±NON-ZERO number below octal 12 or -0 number in the SMODE register starts one of the active options of SELF-CHECK. Below is a description of what part (s) of the computer the options check. A block diagram in Figure 1 on the next page shows the options available and indicates the number to put in the SMODE register for the desired option.

+1 octal:	checks all pulses possible by internal control of the computer.
±2 octal:	checks all the IN-OUT instruction pulses.
+3 octal:	checks SC registers and all bit combinations.

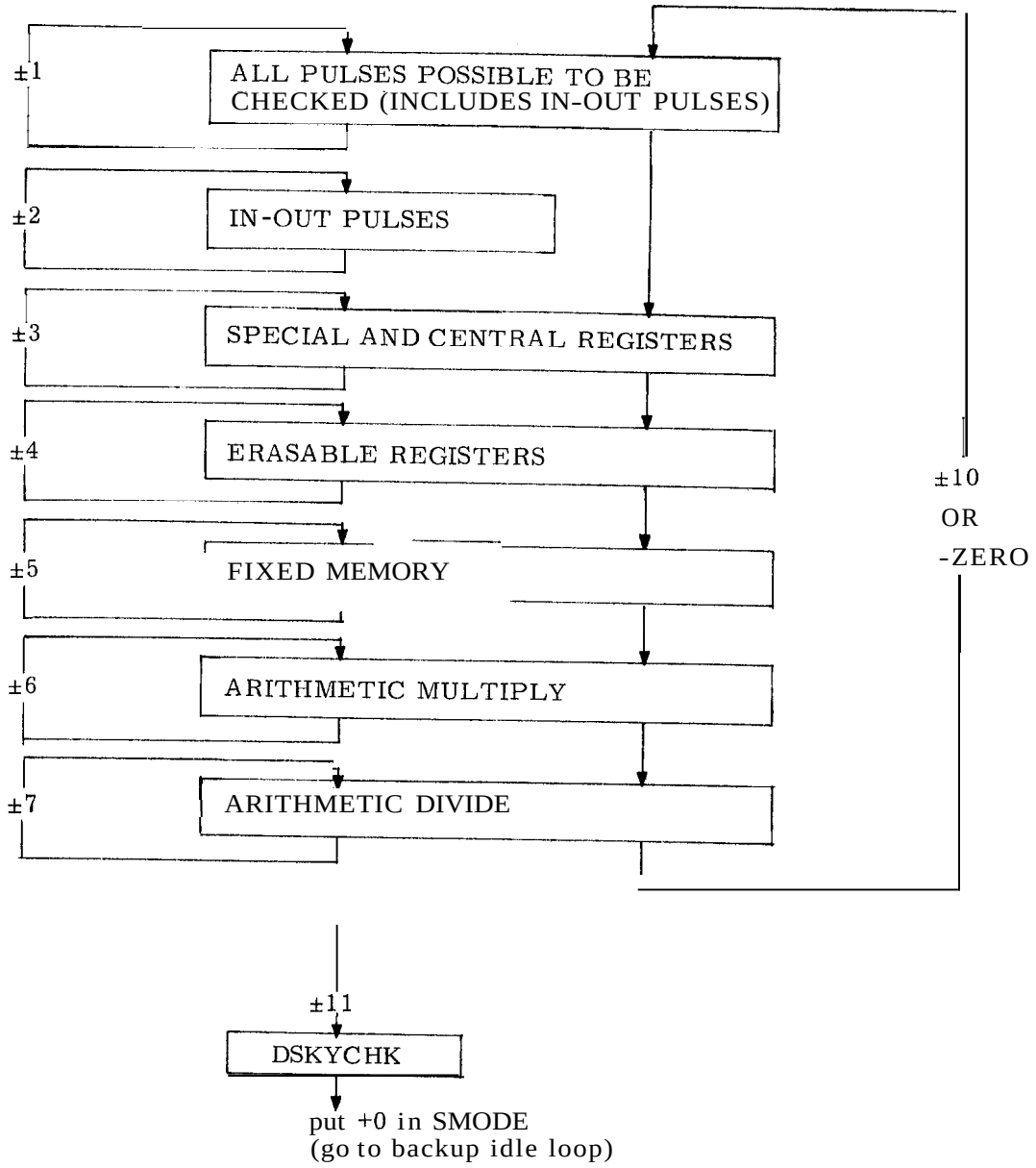


Fig. 1 OPTIONS OF SELF-CHECK

The numbers associated with the options represent the contents of the SMODE register.

The +0 option forces the computer to stay in the backup idle loop, a tight loop which looks for **a** new job from the EXECUTIVE.

±4 octal:	checks erasable memory.
±5 octal:	checks fixed memory.
+6 octal:	an extensive multiply arithmetic check.
+7 octal:	an extensive divide arithmetic check.
+10 octal:	checks everything in the previous seven options (internal self-check of the computer).
±11 octal:	turns on the electroluminescent displays in the DSKY.
-zero:	this option is the same as the ±10 options until an error is detected.
+zero:	does not purposely check any part of the computer.

Procedure to Start SELF-CHECK

SELF-CHECK has its own verb-noun combination that should be utilized when starting any of the options from the DSKY (verb 21 and noun 27).

V21N27E (±0 or ±NON-ZERO)E

This procedure puts the desired number in the SMODE register depending upon the option desired. The pressing of the second enter (E) button completes the procedure.

Report E-1905 by Alan I. Green is recommended for those not acquainted with the operation of the keyboard and display of the Apollo computer. A description of what the three symbols used stand for is given below:

V = Verb
N = Noun
E = Enter

Malfunction Indication

The block diagram in Figure 2 on page 10 is used as a reference for this discussion. If SELF-CHECK should locate a malfunction the following sequence of events will occur:

Step 1:	The contents of the Q register is put in the SFAIL register. This is the address +1 of where the error occurred.
Step 2:	The ERCOUNT register is incremented by one.
Step 3:	The program alarm light on the DSKY is turned on.
Step 4:	Octal 01102 is put in the FAILREG register,
Step 5:	(a) stop SELF-CHECK (if c(SMODE) is +NON-ZERO).
	(b) start at beginning again (if c(SMODE) is -NON-ZERO).
	(c) continue on with SELF-CHECK at the next address after the error (if c(SMODE) is -ZERO).

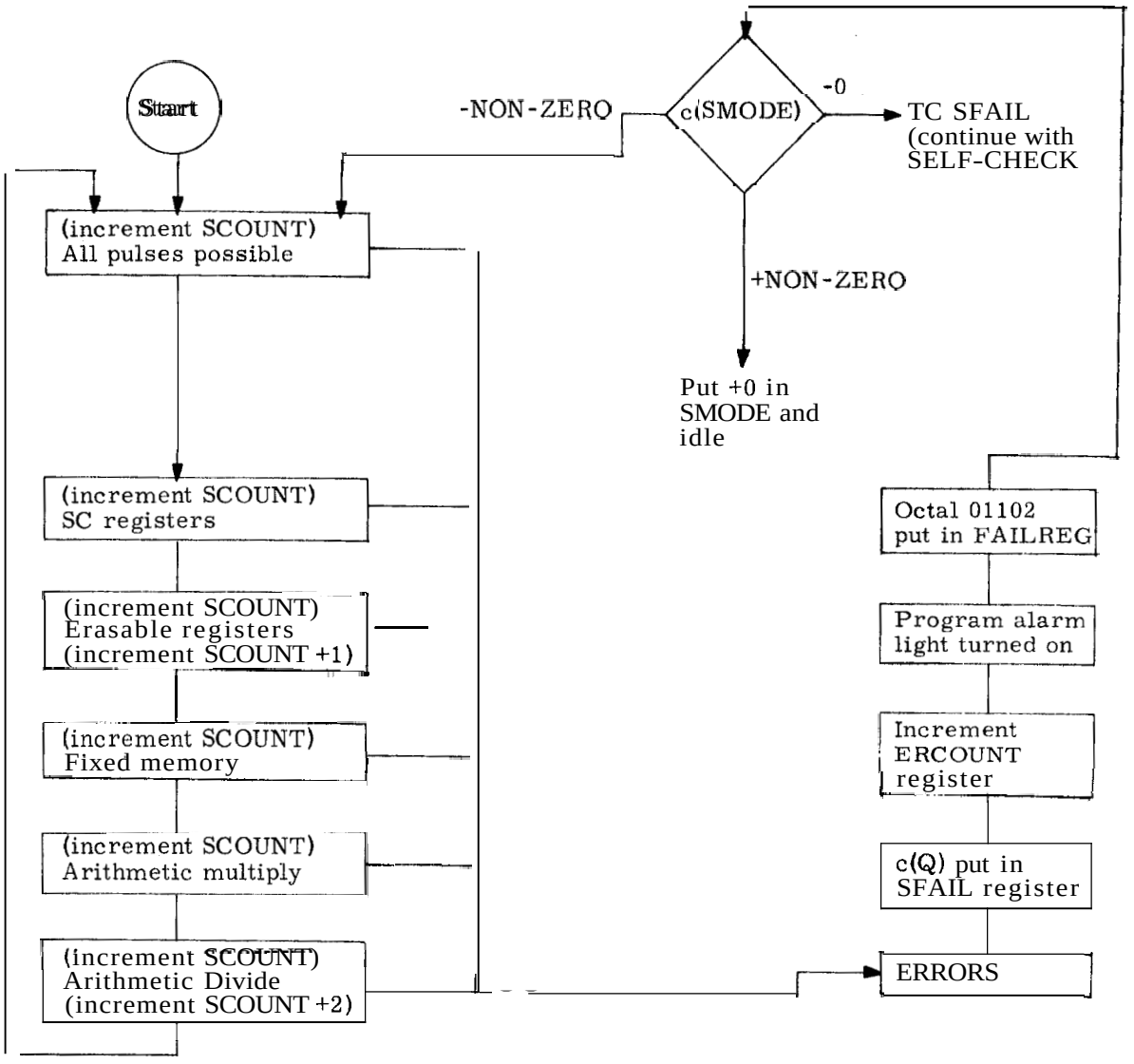


FIG. 2 COUNT REGISTERS AND MALFUNCTION INDICATORS

The above block diagram indicates the flow of SELF-CHECK when ±10 or -zero is put in the SMODE register.

* SCOUNT +2 is incremented after completion of the fixed memory check when arithmetic multiply and arithmetic divide are removed from SELF-CHECK.

Steps 3 and **4** will be omitted if the contents of the FAILREG register is not +zero. A computer "FRESH START" will set the SMODE, SFAIL, FAILREG, and ERCOUNT registers to +zero. A computer "RESTART" will set the SFAIL register to +zero.

If a second malfunction is located, 41102 is put in the FAILREG register but steps **3** and **4** are omitted, Steps **3** and 4 are omitted from all successive malfunctions until the FAILREG register is made +zero (normally by performing a "FRESH START").

It is possible to leave SELF-CHECK on for a long period and keep track of the number of malfunctions that have occurred by observing the ERCOUNT register, The SFAIL register will contain the error address +1 of the last malfunction.

The "program alarm" light on the DSKY is used by other programs beside SELF-CHECK. Therefore, the FAILREG register (1363) should be observed to verify what type of malfunction occurred should this light come on. An octal number 01102 in this register indicates a SELF-CHECK error. Registers SFAIL (1364) and ERCOUNT (1365) should be observed,, and probably recorded, if there has been a SELF-CHECK error because these registers contain the address +1 of where the last error occurred and the total number of errors.

Changes (if any) in SELF-CHECK Options When a Malufnction is Detected

Putting a ±11 in the SMODE register illuminates all possible electroluminescent displays on the DSKY. The subroutine puts a +zero in the SMODE register. This routine does not automatically check for a malfunction of the computer. It depends on an observer to watch **the** DSKY for the proper displays.

No useful function will be performed by putting a number larger than octal 11 in the SMODE register because no SELF-CHECK subroutines have been written for these numbers. If octal 12 or a larger number is put in the SMODE register a subroutine will change the contents of the SMODE to +zero, which forces the computer to go to the backup idle loop,

Figure 3a shows what happens to the option of SELF-CHECK you are in if an error is detected while ±1 to ±10 is in the SMODE register. First, the malfunction indications previously discribed are gone through if the number in the SMODE register is either ±NON-ZERO. However, the next step depends on the sign of the number in the SMODE register. If the number is plus, the contents of the SMODE register is changed to +zero which forces the computer into a backup idle loop. **If** the number in the SMODE is negative, the subroutine that is associated with that number is started at the beginning again and the contents of the SMODE register is not changed.

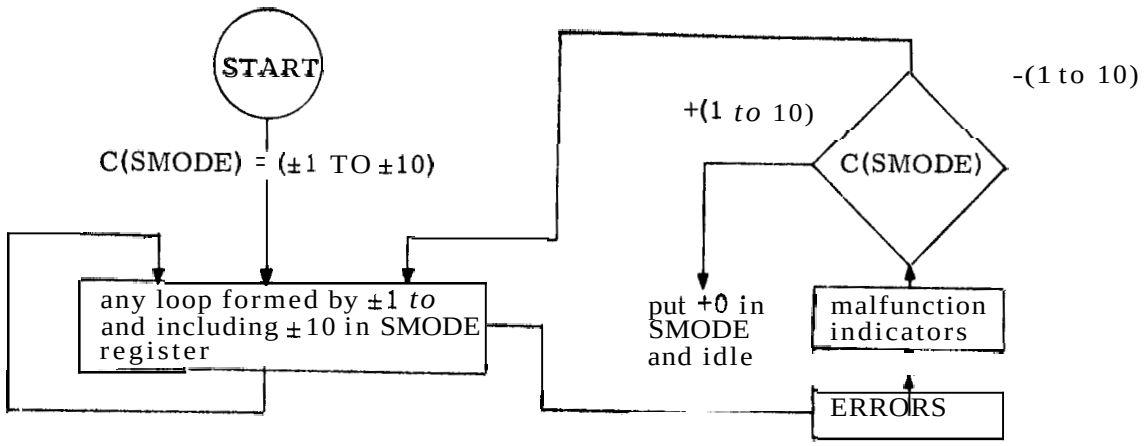


FIG 3a

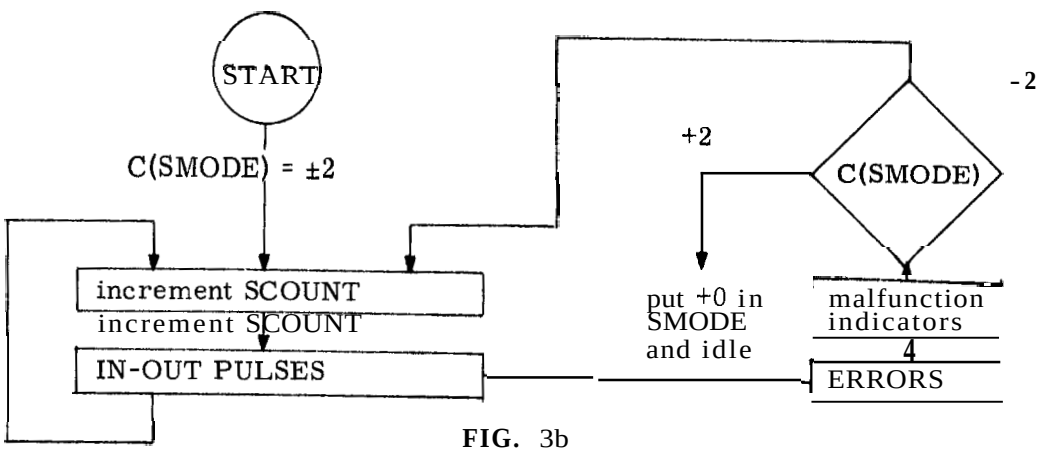
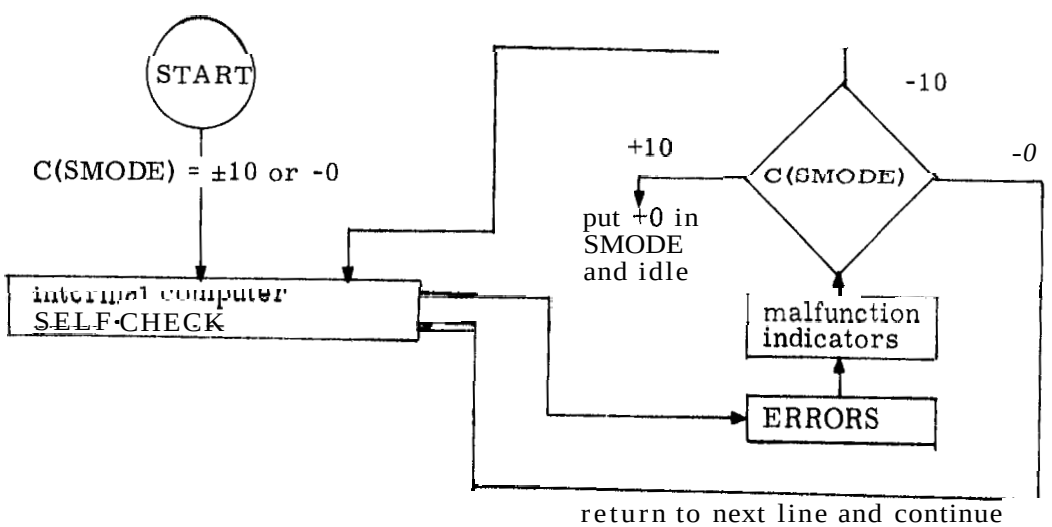


FIG. 3b



FIG, 3c

Fig. 3 ERROR OPTIONS OF SELF-CHECK

Figure 3b shows what happens when ±2 is in the SMODE register and an error is detected. The SCOUNT register is incremented at the beginning of each of the subroutines that make up the internal computer selfcheck, even if they are run through consecutively as they are when ±10 of -zero is in the SMODE register.

Figure 3c shows what happens to the options of SELF-CHECK controlled by ±10 or -zero being in the SMODE register, The reader should also look at Figure 2 to observe how the ±1 to ±7 options are run through consecutively when ±10 or -zero is in the SMODE register. If an error is detected while +10 is in the SMODE register, it is replaced by a +zero. If a -10 is in the SMODE register, the internal computer self-check is started at the beginning again. If a -zero is in the SMODE register, the computer goes back to continue checking the internal computer self-check at the nextline from where the error was detected. Of course the malfunction indicators are updated every time an error is detected,

How to Use the **DSKY** to Monitor SELF-CHECK

The block diagram in Figure 2 shows how the three SCOUNT registers may be utilized to monitor the operation of SELF-CHECK. Register SCOUNT (1366) is incremented at the start of each of the seven minor loops that make up the internal computer self-check. Register SCOUNT +1(1367) is incremented upon the completion of the erasable memory part of the internal computer self-check when +4, ±10 or -0 is in the SMODE register. Register SCOUNT +2(1370) is incremented upon *the* completion of the arithmetic divide part of the internal computer self-check when ±7, ±10, or -0 is in the SMODE register. The incrementing of the SCOUNT +2 register when ±10 or -0 is in the SMODE register indicates the successful completion of the internal self-check of the computer. If a V15N01E 1366E is performed on the DSKY, the contents of these three count registers will appear in R1, R2, and R3 of the DSKY.

It may be desirable, for information or diagnostic reasons, to set the three SCOUNT registers and the ERCOUNT register to zero before initiating one of the options of SELF-CHECK. If *so,* these four registers have to be set to zero from the DSKY. The following procedure will accomplish this:

Step 1:	V21N01E	1765E	00000E	(ERCOUNT register)
Step 2:	N15E	00000E		(SCOUNT register)
Step 3:	E	00000E		(SCOUNT +1 register)
Step 4:	E	00000E		(SCOUNT +2 register)

CHAPTER 2

SHOW-BANKSUM OPERATING PROCEDURES

The SHOW-BANKSUM routine shows the sum of the bank in R1 of the DSKY, the bank number in R2 of the DSKY (should be same number as in R1, but can be positive or negative), and the "bugger" word in R3 of the DSKY. The operating procedure consists of three steps: it is important to perform the last step to end this particular job.

Procedure to START SHOW-BANKSUM

This routine has its own Verb (56) *so* it is very easy to start. The information for bank 00 appears in R1, R2, and R3 of the DSKY immediately after starting SHOW-BANKSUM.*

STARTING PROCEDURE V56E

Procedure to Display Next Bank

The "proceed" verb is utilized to display the sum of the rest of the banks. Each time the proceed verb is entered from the DSKY, the information for the next higher bank appears in R1, R2, and R3 of the DSKY. If another "proceed verb enter" is performed after the last bank in a particular rope has been observed, the information for bank 00 will be displayed again, Continued proceed verb enters will allow you to observe all the banks a second time.

CONTINUE PROCEDURE V33E

Procedure to Stop BANK-SHOWSUM

The operator must punch in the "terminate" verb when he is through with SHOW-BANKSUM. This terminates the SHOW-BANKSUM routine in the EXECUTIVE.

TERMINATE PROCEDURE V34E

---

* Starting SHOW-BANKSUM puts +0 in the SMODE register, This forces SELF-CHECK to go into the backup idle loop.

PART II

EXPLANATION OF SELF-CHECK AND SHOW-BANKSUM

CHAPTER 3

EXPLANATION OF COMPUTER INTERNAL SELFCHECK

SELF-CHECK has been written so it is a check of the computer by the computer. The 19 options described in the "Operating Procedures" part of this report may be utilized for diagnostic purposes should the computer develop a malfunction. Eighteen options are related to the internal operation of the computer; the other option lights up DSKY electroluminescents. The fact that it is possible to successfully change the options of SELF-CHECK assures the basic operation of the EXECUTIVE and much of the DSKY.

There are seven major sections in this version of SELF-CHECK; a minor section (IN-OUT pulses) is also a part of one of the major sections (all pulses possible). The first major section exercises almost all of the control pulses used by the computer, The second major section checks the special and central registers. Erasable memory is checked thirdly. The fourth section checks for the correct contents of the rope and checks the computer circuitry associated with fixed rope memory. The fifth and sixth sections check the arithmetic operations of the multiply and divide instructions, The electroluminescent displays on the DSKY are checked in the seventh section.

Three of the options available in SELF-CHECK allow the computer to consecutively execute the first six major sections of SELF-CHECK. These six sections are considered the internal selfcheck of the computer. Following is a list containing all of the subroutines of these six sections in chronological order as they would be performed when performing the internal selfcheck.

TC+TCF
CCSCHK
BZMFCHK
RESTORE 1
RESTORE2
RESTORE3 — checks almost all pulses, 35 to 45 milliseconds.
BZFCHK
DXCH+DIM
DAS+INCR
MPCHK

The only non-programmable instruction that is checked is PINC, which is checked in the RUPTCHK subroutine. The fact that TIME3 interrupts 2 1/2 milliseconds after TIME4 assures the proper functioning of all the pulses in this instruction. It is not possible to check the pulses in the other non-programmable instructions.

No particular effort has been made to check the pulses associated with the S, Z, and SQ registers, Some of these pulses are used in every memory cycle and the fact that SELF-CHECK is successfully completed assures the existence of these pulses,

A short description of the pulses checked by each subroutine in the pulses section of this report will now be given:

TC+TCF — This subroutine checks all of the pulses of the TC and TCF instructions except the ability to TC to erasable memory. A CS fixed memory instruction is used for the first time and is checked by the next subroutine.

CCSCHK — The main purpose of this subroutine is to make sure the CCS instruction performs the four required branches correctly and that c(A) is correct after each branch. It was necessary to perform a fifth CCS to make sure the CI pulse forced the result of a ± 1 to be +0. All of the CCS pulses are checked except RB-WG,

This subroutine also checks pulses associated with CS fixed, erasable, and special and central memory. Also those associated with a TS to erasable memory.

RZMFCHK — All of the pulses used by the BZMF instruction are checked by the BZMFCHK subroutine. Also those pulses used by CA fixed memory,

The fact the BZMF instruction should jump when the $c(A) \leq 0$ and that it should not jump when c(A) = +NON-ZERO is checked. Also that is does not jump when c(A) is overflow with +0 (01-00000).

RESTORE1 — This subroutine checks the ability of the NDX, CCS, AD, MSU, SU, CA, and MASK instructions to read the original contents back into erasable memory. The normal operation of these instructions are not of primary importance.

The NDX erasable, CA erasable, and MASK erasable instructions are used and checked for the first time.

The fact that the MASK, MP, and DV instructions do not edit is also checked.

RESTORE2 This subroutine checks the ability of the extended NDX, DCA, and DCS instructions to read the original contents back into erasable memory. The normal operation of these instructions are not of primary importance.

The pulses used by the XCH erasable, extended NDX erasable, extended NDX fixed memory, DCS erasable, CA special and central, and the DCA erasable instructions are checked.

RESTORE3 The ability to restore instructions back into erasable memory is checked by this subroutine.

BZFCHK All of the pulses used by the BZF instruction are checked by the BZFCHK subroutine. The fact that the BZF instruction should jump when the c (A) = ±0 and should not jump when c (A) # ±0 is checked. It is also made sure that the BZF instruction will not jump with overflow (01-0000) and underflow (10-377777) in the A register.

DXCH+DIM DXCH+DIM checks all of the pulses used by the DXCH and the DIM instructions. It also checks the pulses used by the TS with overflow, TS special and central, CA special and central, and AD erasable instructions.

DAS+INCR This subroutine checks all of the pulses in the DAS and INCR instructions. It also checks the pulses used by the DCA fixed memory, DCS fixed memory, LXCH special and central, and XCH special and central memory instructions.

The pulses in the AD instruction are also checked thoroughly for the first time. The AD instruction has been used before but this is the first time the result of the addition has been checked.

MPCHK	The MPCHK subroutine checks all of the pulses used by the MP, AUG, and ADS instructions. The AUG and ADS instructions are utilized in the process of checking the four sign combinations possible in multiply.
DVCHK	All of the pulses of the DV and QXCH instructions are checked by this subroutine as well as the pulses used by the TS with underflow instruction. Six divides are used to thoroughly check out all the sign combinations and other features of this instruction.
MSUCHK	This subroutine checks all of the pulses of the MSU instruction except the RB-WG pulses, which are checked by the RESTORE1 subroutine.
MASKCHK	MASKCHK checks the pulses in the MASK that have not previously been checked.
NDX+SU	This subroutine finishes checking the pulses in both the index instructions. It also checks all of the pulses in the SU instruction except RB-WG, which are checked in the RESTORE1 subroutine.
D--SC	The D--SC subroutine checks that DCS, DXCH, and DCA can be performed on special and central registers. A DXCH and DCS is performed on the L register because the order sequence of pulses can be checked more thoroughly by using this register.
D--LCHK	This subroutine was written to check that the overflow bit disappeared when a word went into and out of the L register and to make sure that the Q register was capable of holding 16 bits.
ADDRCHK	ADDRCHK makes sure the overflow, underflow, end-around-carry features, and other features of the adder are functioning correctly. It also makes sure that the ADS special and central instruction is working satisfactorily when the result of the addition is overflow.

RUPTCHK — The main purpose of this subroutine is to make sure that an overflow-underflow condition in the A register will hold off an interrupt. It also checks that INHINT will also hold off an interrupt and that a waiting interrupt will interrupt immediately after the RESUME instruction. The basic operation of TIME3, TIME4, and the WAITLIST are also checked since they are all used by this subroutine.

IN-OUT1 — Checks all pulses of the WRITE and READ instructions.

IN-OUT2 — Checks all pulses of the ROR and WOR instructions.

IN-OUT3 — Checks all pulses of the RAND, WAND, and RXOR instructions.

Check of Special and Central Registers

This section of SELF-CHECK makes sure the A, B, C, G, and Q registers and the output of the adder have all 16 decimal bit combinations pass through them at least once. All 15 decimal bit combinations are put into and called out of the L register and erasable memory; thus the parity bit is generated and checked for each 15 bit combination. It is not possible to guarantee the parity register is working correctly if words come out of erasable memory correctly. However this part of SELF-CHECK will indicate an error if any bits are dropped or picked up. Therefore, if the parity register does not catch bits being dropped or picked up, this part of SELF-CHECK will indicate a malfunction.

Following is a short description of the subroutines in this part of SELF-CHECK:

COUNTCHK — Effectively counts down a 15 decimal bit number by one until zero is reached and checks that each successive number is actually one less than the number preceding it. Actually bit 15 is a sign bit so the countdown alternates between plus and minus numbers. In the process of counting down the 15 decimal bit number all the bit combinations are generated by the adder and are written in and out of the A, B, C, L, Q, and G registers as well as erasable memory. Also the parity bit is generated and checked internally by the computer for all 15 bit combinations.

O-UFLOW Checks that all overflow and underflow bit combinations are generated by the adder and are written into and out of A, B, C, and Q registers. The procedure used is to count down, by one, from maximum positive overflow and negative underflow conditions until the overflow-underflow condition does not exist. Again there is a check that each successive number is one less than the preceding number.

Check of Erasable Memory

This part of SELF-CHECK makes sure that is is possible to read a "1" and a "0" into and out of each bit position of erasable memory with the following exceptions. Registers 1377, 1376, and 1375 are not specifically checked in this part of SELF-CHECK because they have previously been thoroughly checked while checking the special and central registers, These three registers are required for storage while checking the rest of erasable memory. The special erasable registers from 61 down through 10 are only addressed to see if a parity error occurs. Finally the cycle and shift registers are checked by putting a combination of alternate zeros and ones in these registers and making sure the correct operation is performed.

Following is a short description of the subroutines in this part of SELF-CHECK:

ERASCHK The non-special erasable registers are checked for correct addressing and content by placing their own address in two successive registers and making sure there is a difference of -1 when the contents of the lower address register is added to the complement of the higher address register; if it is not, this subroutine performs a TC to the ERRORS subroutine. The contents of the two registers are complemented and the complement of the lower register added to the contents of the higher register; the result is checked for -1. The previous contents of the erasable registers are preserved and replaced after the registers have been checked. The higher address register of the previous iteration becomes the lower address register of the present iteration. The erasable memory banks are checked from zero through seven with common erasable (60-1374) being checked after each erasable bank.

CNTRCHK The CS instruction is performedon all erasable registers from octal 60 through octal 10. These include all counters and other special erasable registers. It is not feasible to put their own address in these registers and check their contents because of their special use.

CYCLSHFT The octal number **25252** is placed in the two cycle registers, the shift right register, and the EDOP register. The contents of these registers are then twice checked for correct contents.

Check of Rope Memory

The routine for checking the correct contents of a rope is called ROPECHK. Its purpose is twofold. First it is a check on the computer. It makes sure all current drivers, sense amplifiers, and associated circuitry used in connection with the fixed memory are operating properly. Secondly it is a check on the rope itself. It makes sure none of the sense or inhibit lines have become shorted or opened (essentially guarantees contents of rope is correct and can be read correctly by the computer).

The sum of each bank should be the same as its bank number in the low order bits of the computer. A special word, which is called a "bugger" word, is added to the normal sum of the bank as the last word to be added. This bugger word forces the sum of the bank to be plus or minus the Bank Number. As an example, the sum of bank *33* octal may be 00033 or 77744.

Two TC SELF words indicate the end of the summing process for each bank. The "bugger" word immediately follows the second TC SELF word. Of course all addresses in a bank up to and including the bugger word have to contain words of good parity.

Following is a short description of the ROPECHK subroutine:

ROPECHK Each bank in the rope is summed separately; from the lowest address to the highest address used in that bank. The contents of a higher address is added to the sum of the previous addresses. If this creates an overflow condition a +1 is added to the new sum; a -1 is added to the new sum if an underflow condition is created. The sum of each bank should be plus or minus its own bank number. If the sum of the bank is its bank number the subroutine proceeds on to checking the next bank. If the sum of the bank is not

its bank number SELF-CHECK goes to the error routine. The banks are checked in ascending order.

Check of Multiply Arithmetic Function

There are four multiply loops in the multiply subroutine. The two main purposes of this subroutine are to form all the different combinations of adds possible in the multiply instruction (1 to 14) and to change the value of the word to be added from minimum to maximum for each combination of add. The total time of the multiply routine takes approximately 20 seconds.

It is felt that the multiply and divide subroutines are a good arithmetic check of the computer. Therefore the long activity time of these subroutines may be utilized to check normal operation of the computer in conjunction with asynchronous and synchronous interface signals. The correct result of each multiply and each divide is verified before proceeding on. The procedure gone through if an error is found is described in the "operating procedures'' section of this report.

A description of the multiply subroutine is below:

MPNMBRS — The first multiply loop multiplies 37777 by (37777 through 00001). The contents of the A register counts down while the contents of the L register counts up. There is a check after each multiplication that these two registers add up to 37777. The second multiply loop multiplies 77776 by (37777 through 00001). There is a check in this loop that the c (A) is minus zero and the c (L) counts down by minus one after each multiplication. The third loop interchanges the multiplier and the multiplicand of the first loop. The contents of the A and L registers should be the same as in the first loop. The fourth loop interchanges the multiplier and multiplicand of the second loop. The contents of the A and L registers should be the same as the second loop.

Check of Divide Arithmetic Function

The four divide subroutines form different combinations of subtractions while varying the value of the word to be subtracted. It takes approximately 0.01 second to go through all the four divide subroutines. However SELF-CHECK keeps the computer in the divide subroutines for approximately 20 seconds.

Following is a description of the divide subroutines:

DVICHK — Divides ±/17777/±/37777/ by ±/20000/. The contents of the A register and L register have opposite signs before the division. The quotient is ±/37774/; the sign depends on the sign of c (A) and the sign of the divisor. The remainder is ±1 depending upon the sign of the contents of the A register before the division.

DV2CHK — Divides +17777+37777 by +20000. The quotient is +37777 with +17777 the remainder.

DV4CHK — Divides +37776+0 by +37776. The quotient is +37777 with a remainder of +37776.

DV5CHK — Divides ±0±0 by ±0. The contents of the A register and L register have opposite signs before each division. The quotient will be ±/37777/; the sign depends on the sign of c (L) and the sign of the divisor. The remainder is ±0; the sign depends on the sign of the L register before the division. This is not a useful dicision but it does help to make sure the computer is operating correctly.

CHAPTER 4

EXPLANATION OF DSKYCHK

The purpose of DSKYCHK is to light up all the DSKY electroluminescent elements. It puts a +0 in the SMODE register at the beginning of the routine, which forces the computer internal selfcheck to sleep. This is the only routine in SELF-CHECK that does not have to be terminated. It runs to completion once and then the computer falls into the backup idle loop. The routine has to be entered as one of the SELF-CHECK options every time it is to be exercised.

Each electroluminescent display lasts for 5.12 seconds to allow time to observe all the elements in the display. The sequence of the displays is described next:

DSKYCHK — First the digit "9" is displayed in the R1, R2, R3, Verb, Noun, and PROG positions of the DSKY. The digits 8 through 0 are then each displayed in all the possible displays on the DSKY. The next display leaves all zeros in the DSKY and turns on the "computer activity" light and the "verb" flash and the "noun" flash. The last display has only the "computer activity" light on, Finally the DSKY is left completely blank.

CHAPTER 5

EXPLANATION OF SHOW-BANKSUM

SHOW-BANKSUM consists of a routine called SHOWSUM. This routine essentially does the same thing that the routine ROPECHK does; that is, add up the sum of separate banks in the rope. After this the similarity ends. ROPECHK makes sure the sum of the bank is plus or minus its own bank number while SHOWSUM displays the sum of the bank in R1 of the DSKY irrespective of what the sum may be. SHOWSUM also displays the bank number and the bugger word in R2 and R3 of the DSKY at the same time. The sum of the bank and bank number in R1 and R2 are shown as the least significant bit instead of bits 11 - 15 (the actual bank bits in the computer). Again it is worthwhile mentioning that the sum of a bank may be plus or minus its bank number. This is, bank 5 may be 00005 or 77772.

Undoubtedly the greatest use of this routine will be in restoring the confidence of personnel in the computer and in verifying that the correct rope modules for a particular mission are actually the ones in the computer package. Following is a short description of the SHOWSUM subroutine:

SHOWSUM — Each bank in the rope is summed separately; from the lowest address to the highest address used in that bank. The contents of a higher address are added to the sum of the previous addresses. If this creates an overflow condition a +1 is added to the new sum; a -1 is added to the new sum if an underflow condition is created. The sum of each bank should be plus or minus its own bank number. The sum of the bank is displayed in R1 of the DSKY. The bank number (actual bank number used to sum the bank shifted 5 places left) is displayed in R2 and the bugger word is displayed in R3. Entering a proceed verb (33) from the DSKY will display the same information for the next higher bank, Entering a terminate verb (34) from the DSKY will end the SHOWSUM routine.

APPENDIX

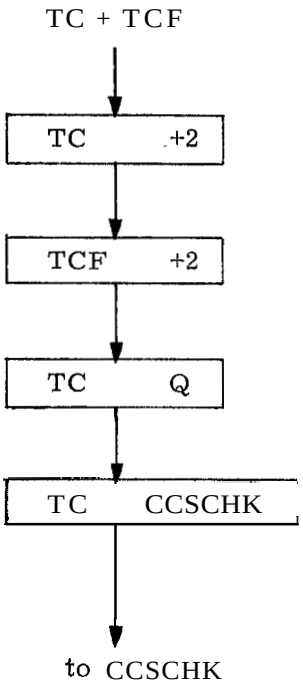
The flow charts in this appendix use both AGC instructions and word descriptions to explain the subroutines. The purpose was to use word descriptions most of the time and utilize instructions only where they were necessary to make the overall picture clear.

Some explanation of symbols:

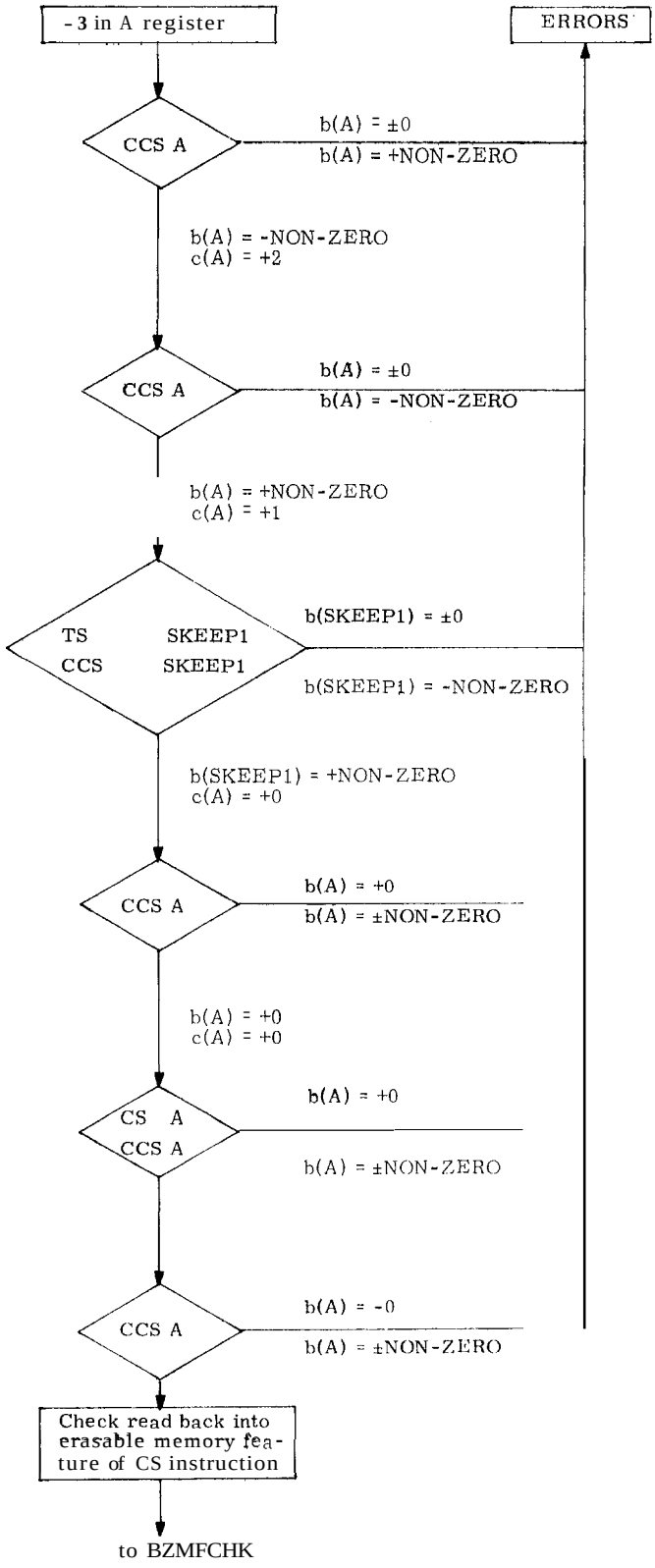
b(X) means before contents of X register

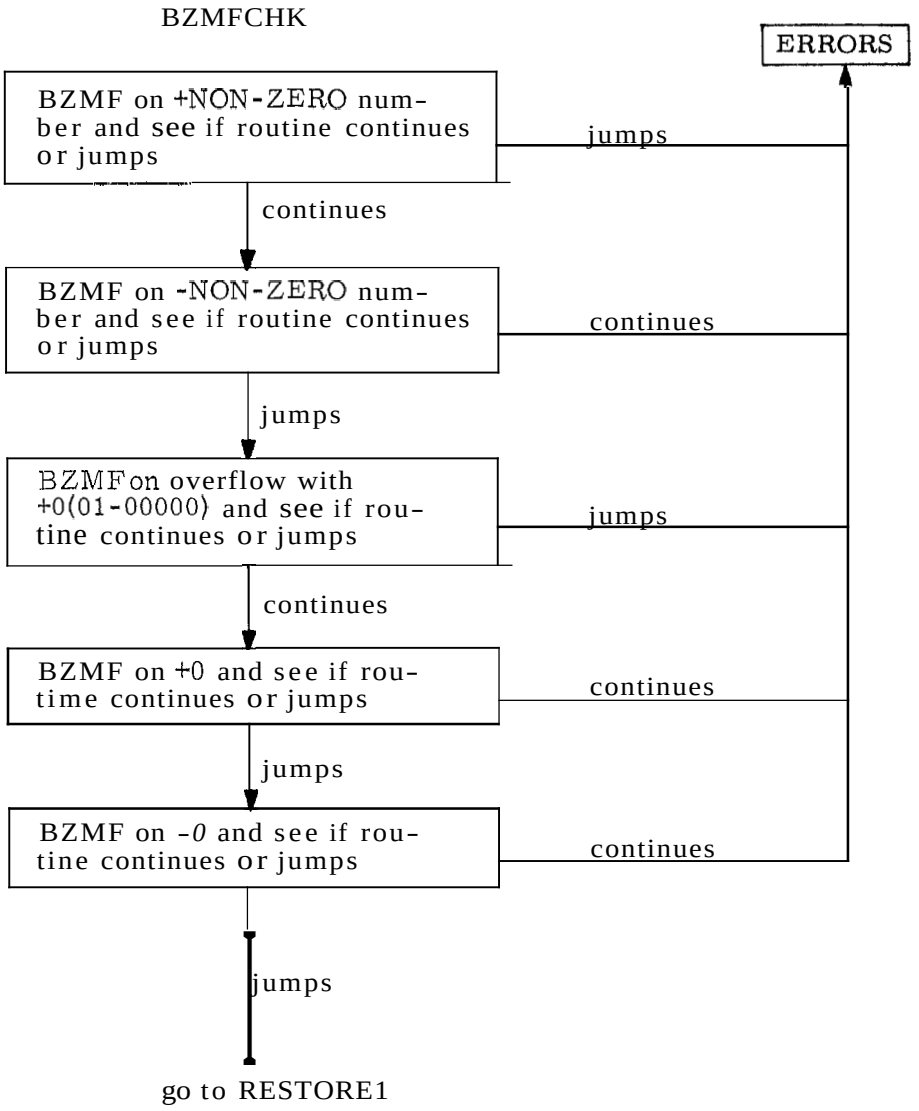
c(X) means contents of X register

SKEEPl through SKEEP7 are erasable registers,

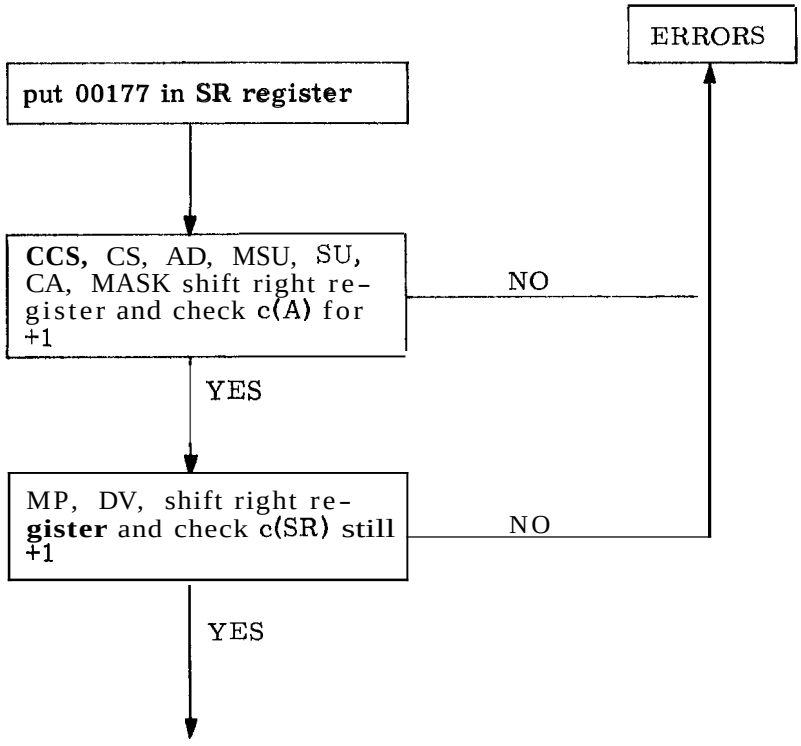

TC + TCF
TC +2
TCF +2
TC Q
TC CCSCHK
to CCSCHK

CCSCHK


-3 in A register
ERRORS
CCS A
b(A) = ±0
b(A) = +NON-ZERO
b(A) = -NON-ZERO
c(A) = +2
CCS A
b(A) = ±0
b(A) = -NON-ZERO
b(A) = +NON-ZERO
c(A) = +1
TS SKEEP1
CCS SKEEP1
b(SKEEP1) = ±0
b(SKEEP1) = -NON-ZERO
b(SKEEP1) = +NON-ZERO
c(A) = +0
CCS A
b(A) = +0
b(A) = ±NON-ZERO
b(A) = +0
c(A) = +0
CS A
CCS A
b(A) = +0
b(A) = ±NON-ZERO
CCS A
b(A) = -0
b(A) = ±NON-ZERO
Check read back into erasable memory feature of CS instruction
to BZMFCHK

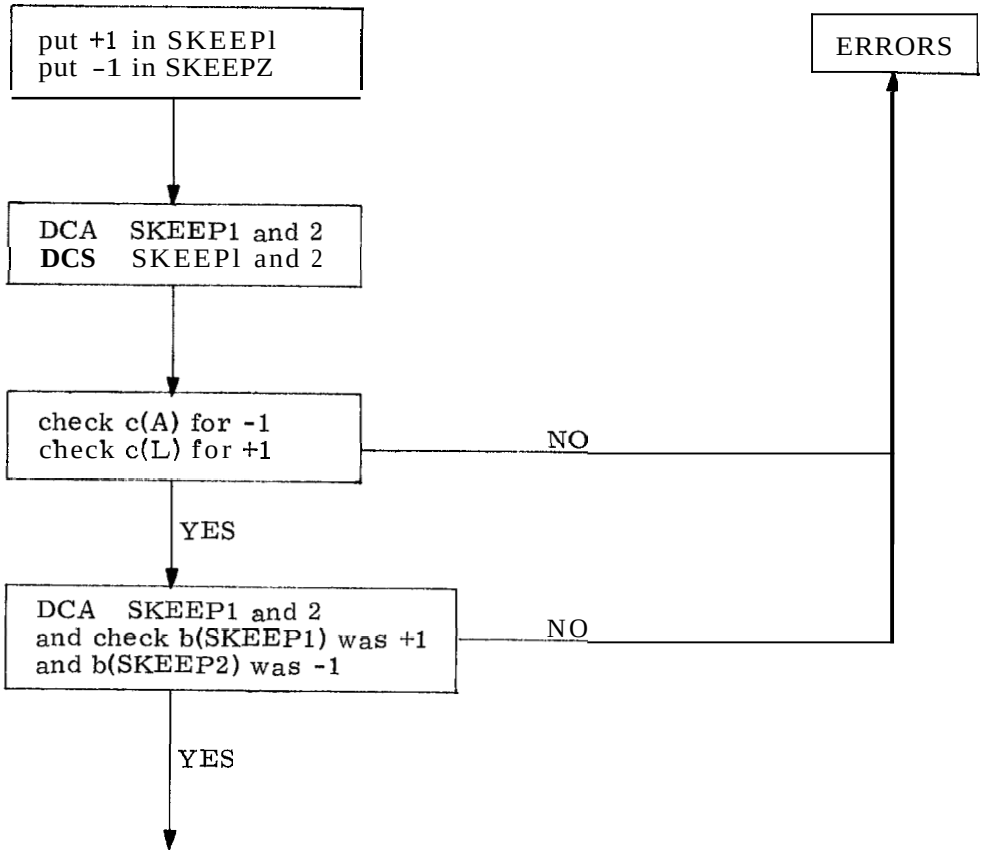

BZMFCHK
ERRORS
BZMF on +NON-ZERO number and see if routine continues or jumps
jumps
continues
BZMF on -NON-ZERO number and see if routine continues or jumps
continues
jumps
BZMF on overflow with +0(01-00000) and see if routine continues or jumps
jumps
continues
BZMF on +0 and see if routine continues or jumps
continues
jumps
BZMF on -0 and see if routine continues or jumps
continues
jumps
go to RESTORE1

RESTORE1

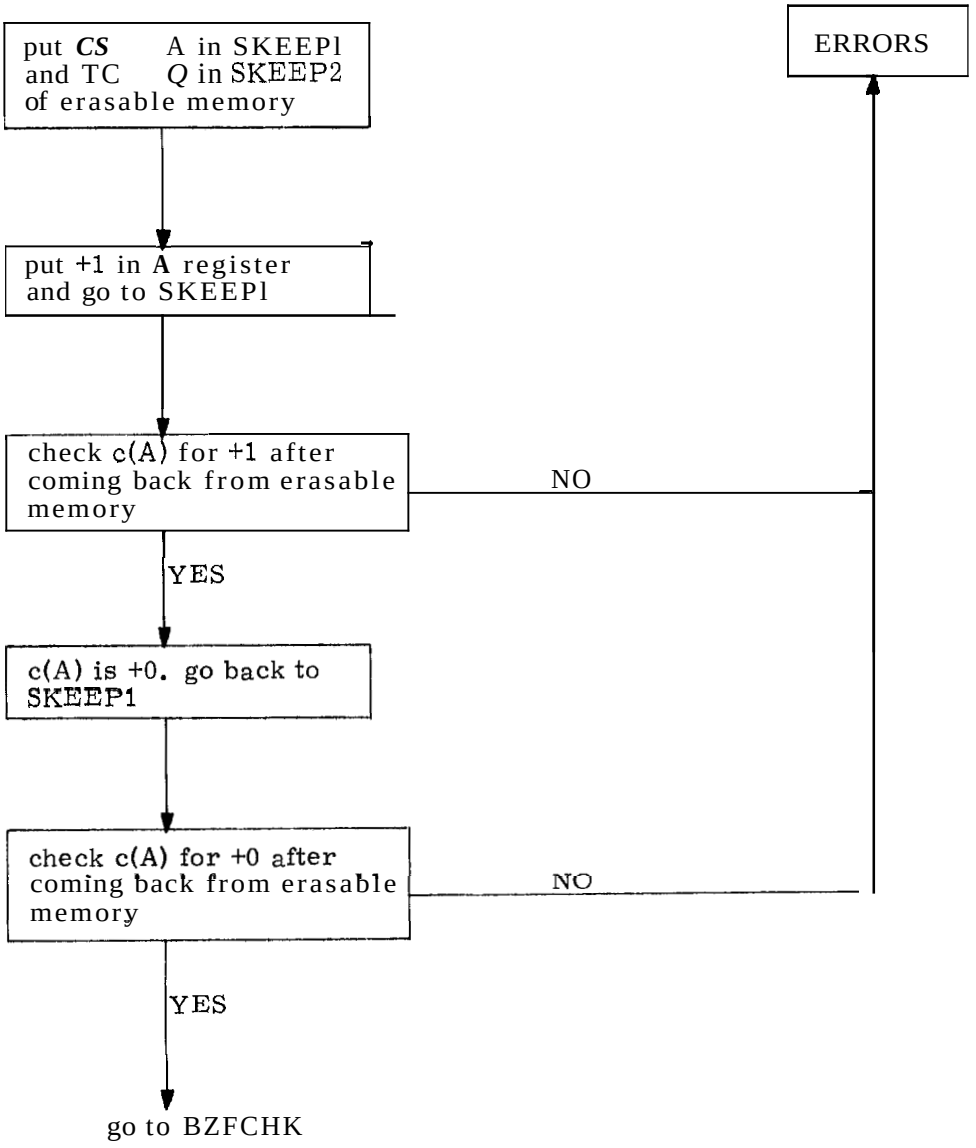


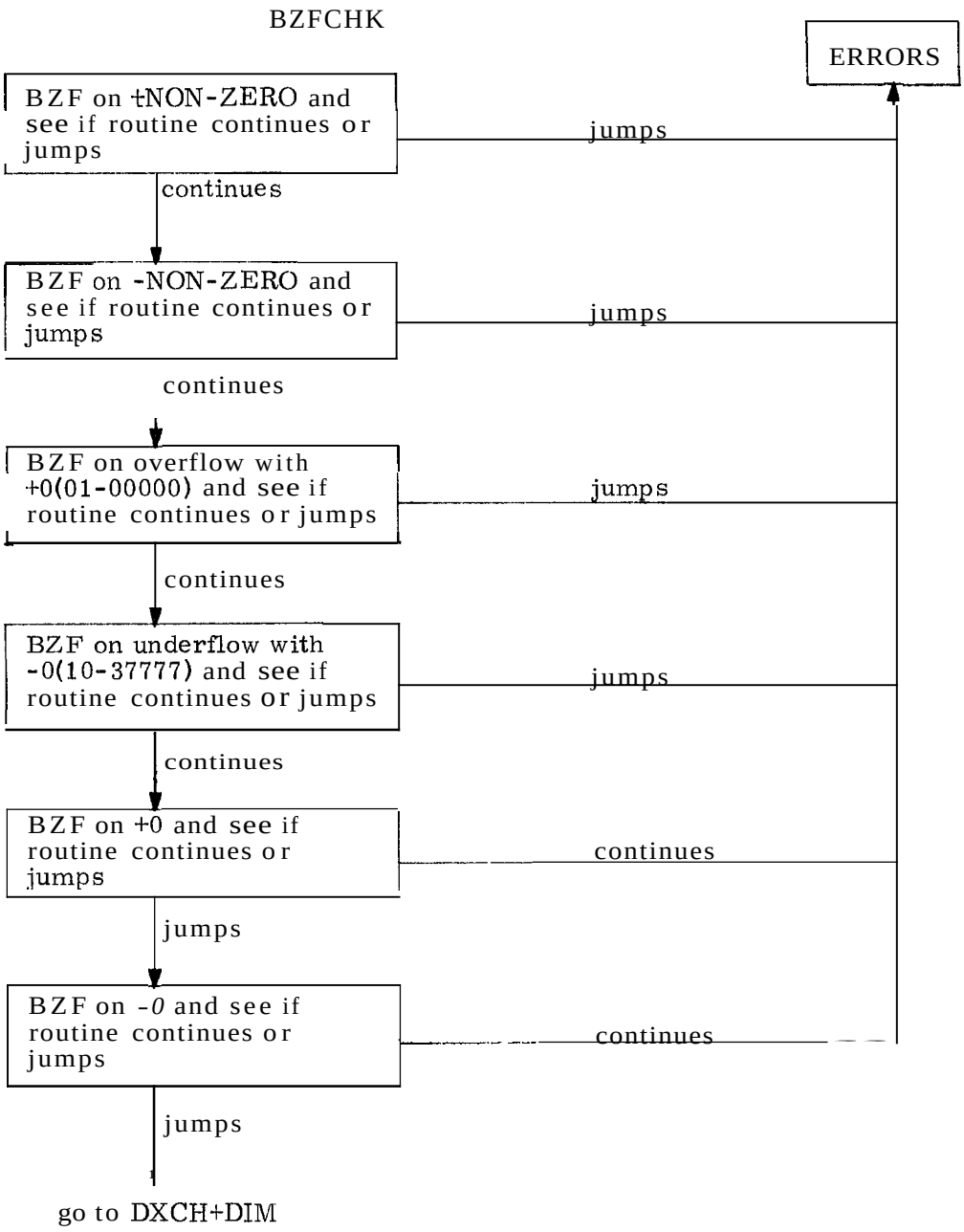
go to RESTORE2

RESTORE2

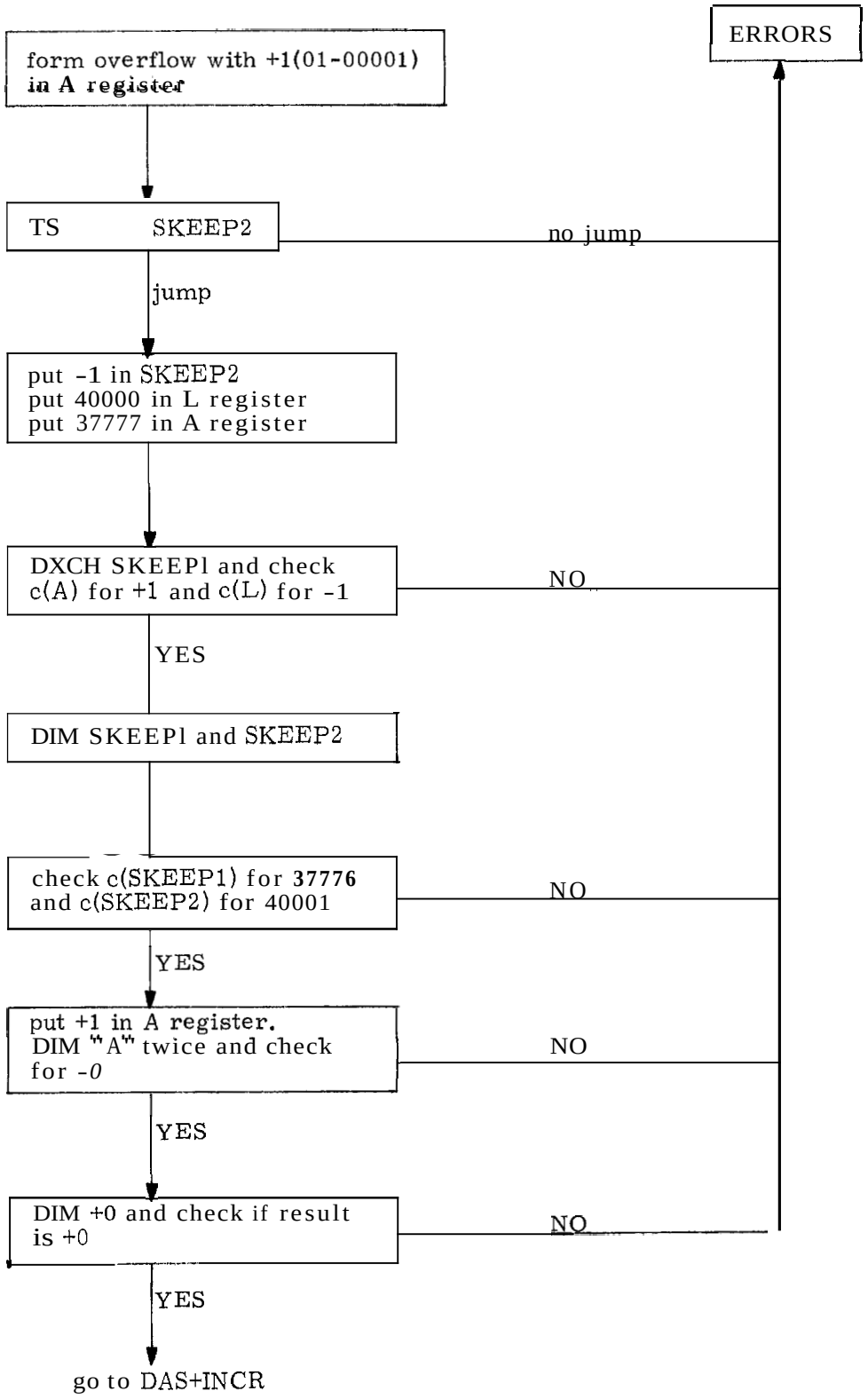

put +1 in SKEEPl
put -1 in SKEEPZ
ERRORS
DCA SKEEP1 and 2
DCS SKEEPl and 2
check c(A) for -1
check c(L) for +1
NO
YES
DCA SKEEP1 and 2
and check b(SKEEP1) was +1
and b(SKEEP2) was -1
NO
YES
go to RESTORE3

RESTORE3

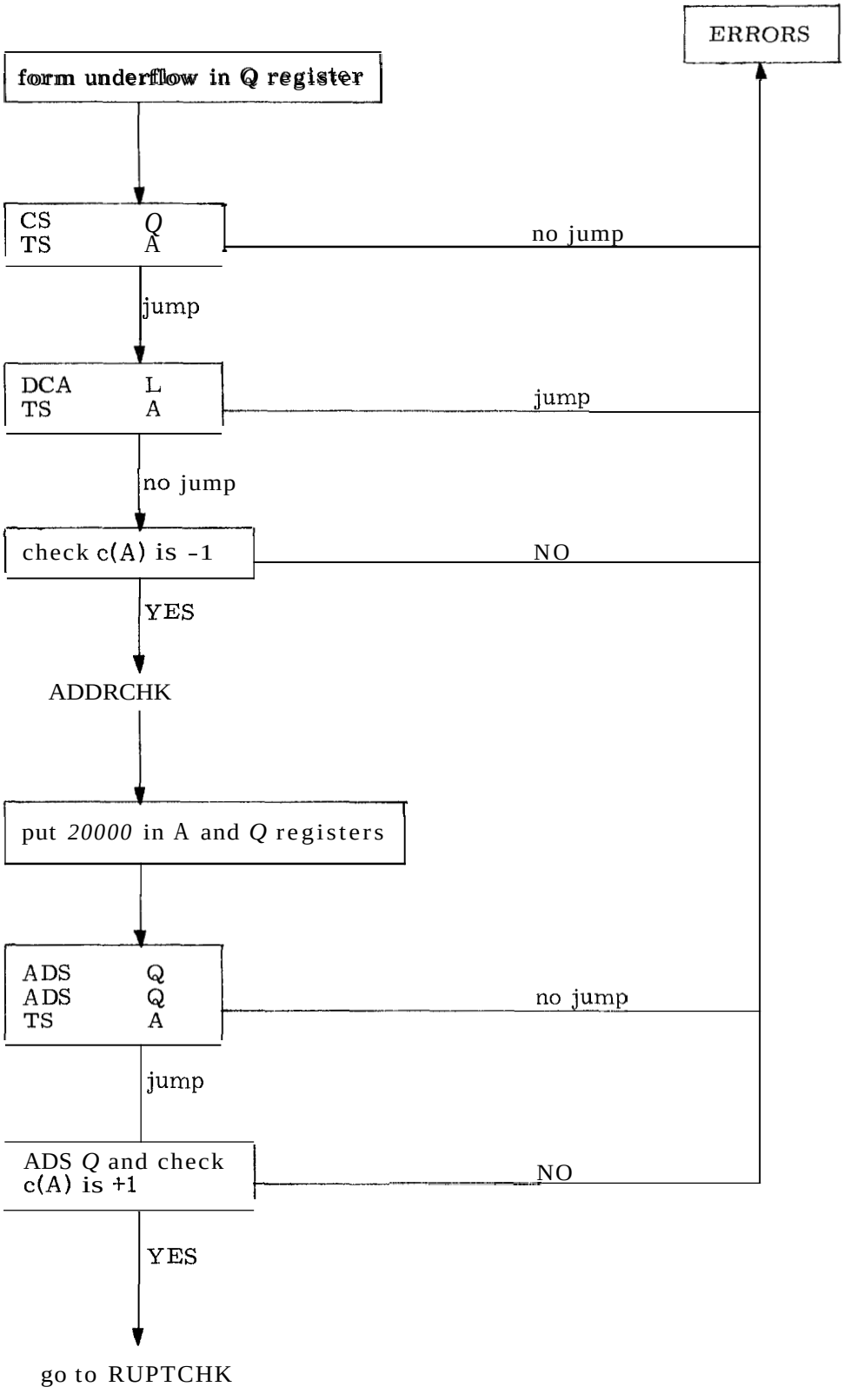

put CS A in SKEEP1
and TC Q in SKEEP2
of erasable memory
ERRORS
put +1 in A register
and go to SKEEP1
check c(A) for +1 after
coming back from erasable
memory
NO
YES
c(A) is +0. go back to
SKEEP1
check c(A) for +0 after
coming back from erasable
memory
NO
YES
go to BZFCHK


BZFCHK
ERRORS
BZF on +NON-ZERO and see if routine continues or jumps
jumps
continues
BZF on -NON-ZERO and see if routine continues or jumps
jumps
continues
BZF on overflow with +0(01-00000) and see if routine continues or jumps
jumps
continues
BZF on underflow with -0(10-37777) and see if routine continues or jumps
jumps
continues
BZF on +0 and see if routine continues or jumps
continues
jumps
BZF on -0 and see if routine continues or jumps
continues
jumps
go to DXCH+DIM

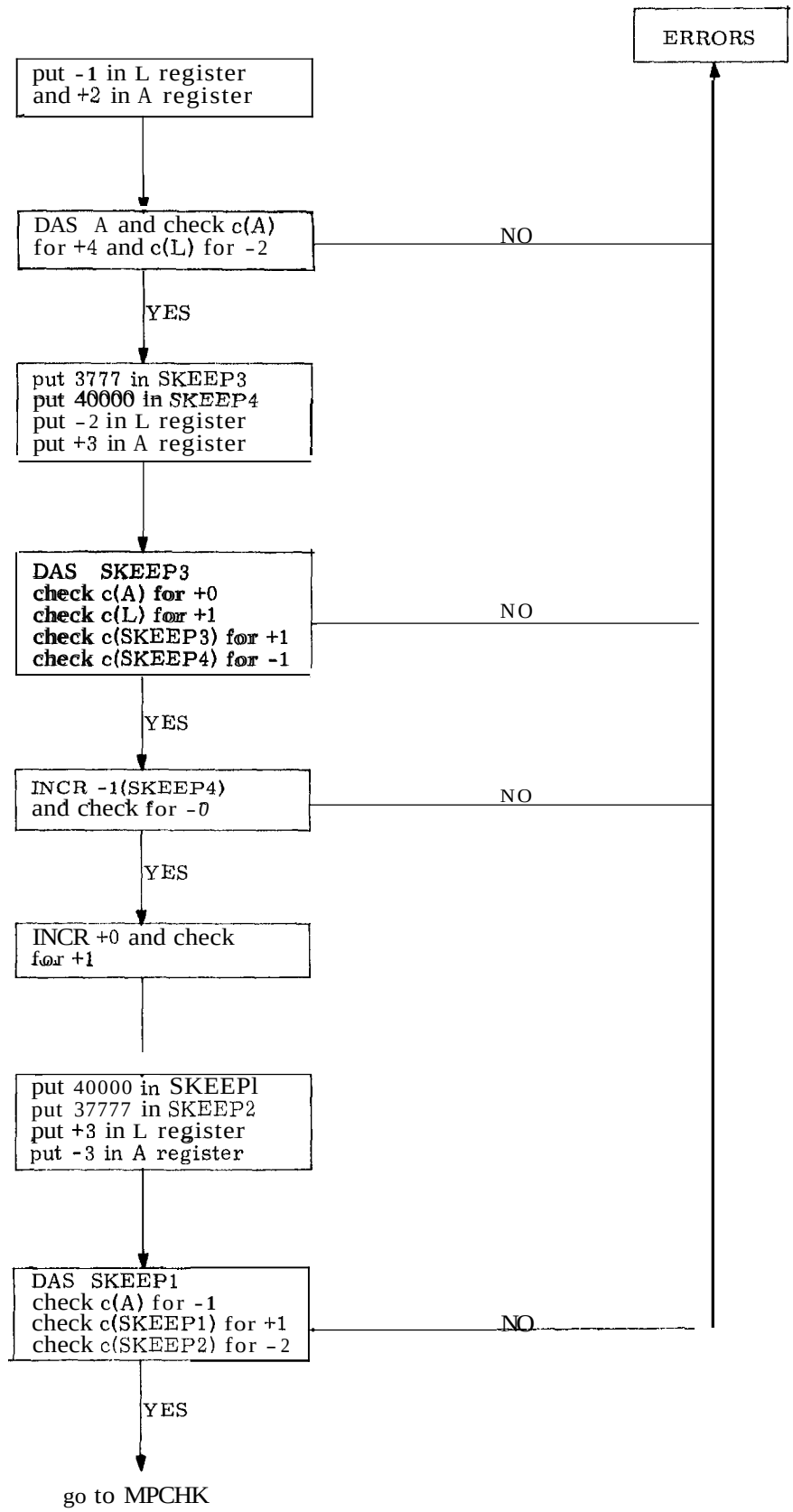
DXCH+DIM


ERRORS
form overflow with +1(01-00001) in A register
TS SKEEP2
no jump
jump
put -1 in SKEEP2
put 40000 in L register
put 37777 in A register
DXCH SKEEP1 and check c(A) for +1 and c(L) for -1
NO
YES
DIM SKEEP1 and SKEEP2
check c(SKEEP1) for 37776 and c(SKEEP2) for 40001
NO
YES
put +1 in A register. DIM "A" twice and check for -0
NO
YES
DIM +0 and check if result is +0
NO
YES
go to DAS+INCR

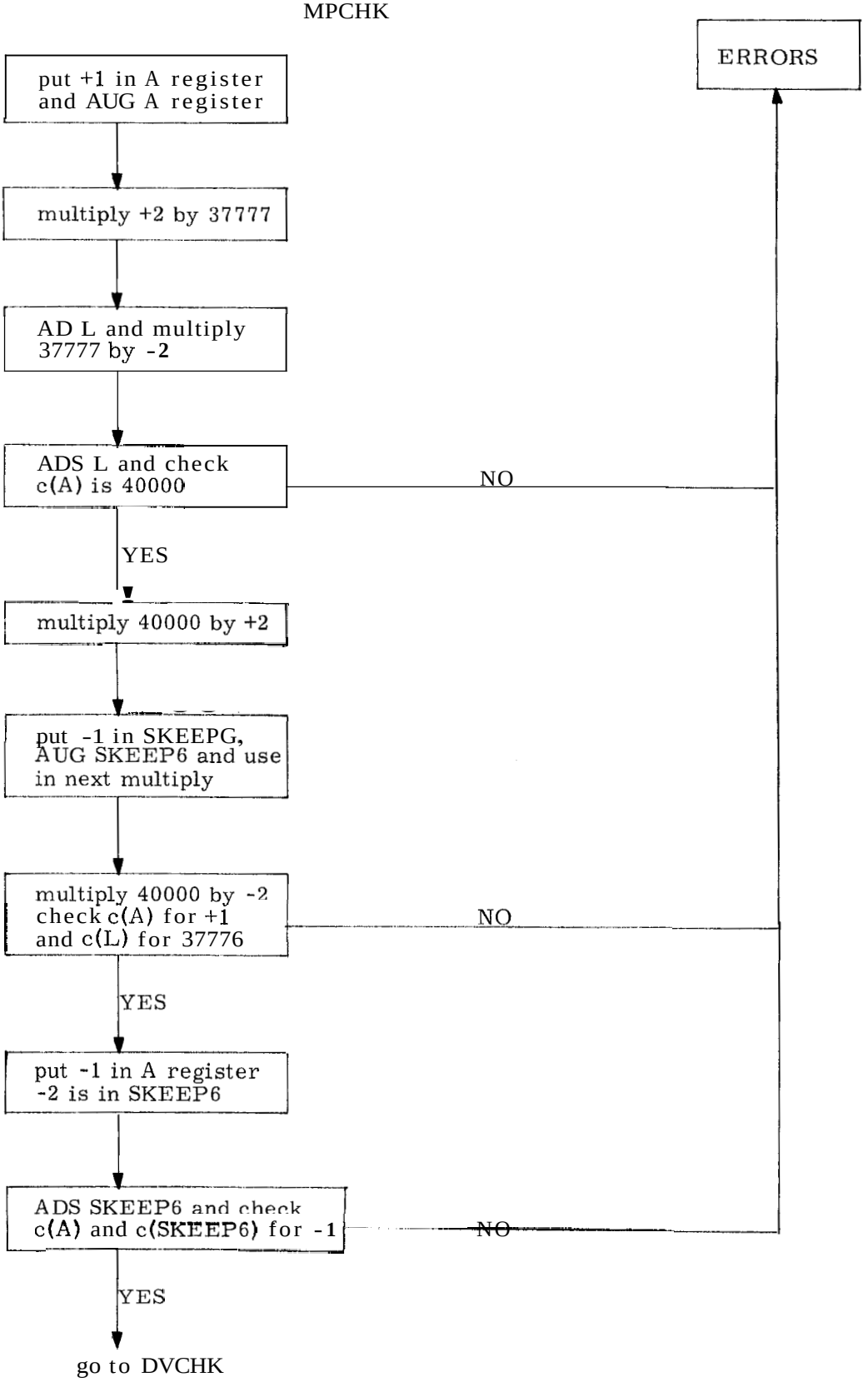
D--LCHK


ERRORS
form underflow in Q register
CS Q
TS A
no jump
jump
DCA L
TS A
jump
no jump
check c(A) is -1
NO
YES
ADDRCHK
put 20000 in A and Q registers
ADS Q
ADS Q
TS A
no jump
jump
ADS Q and check c(A) is +1
NO
YES
go to RUPTCHK

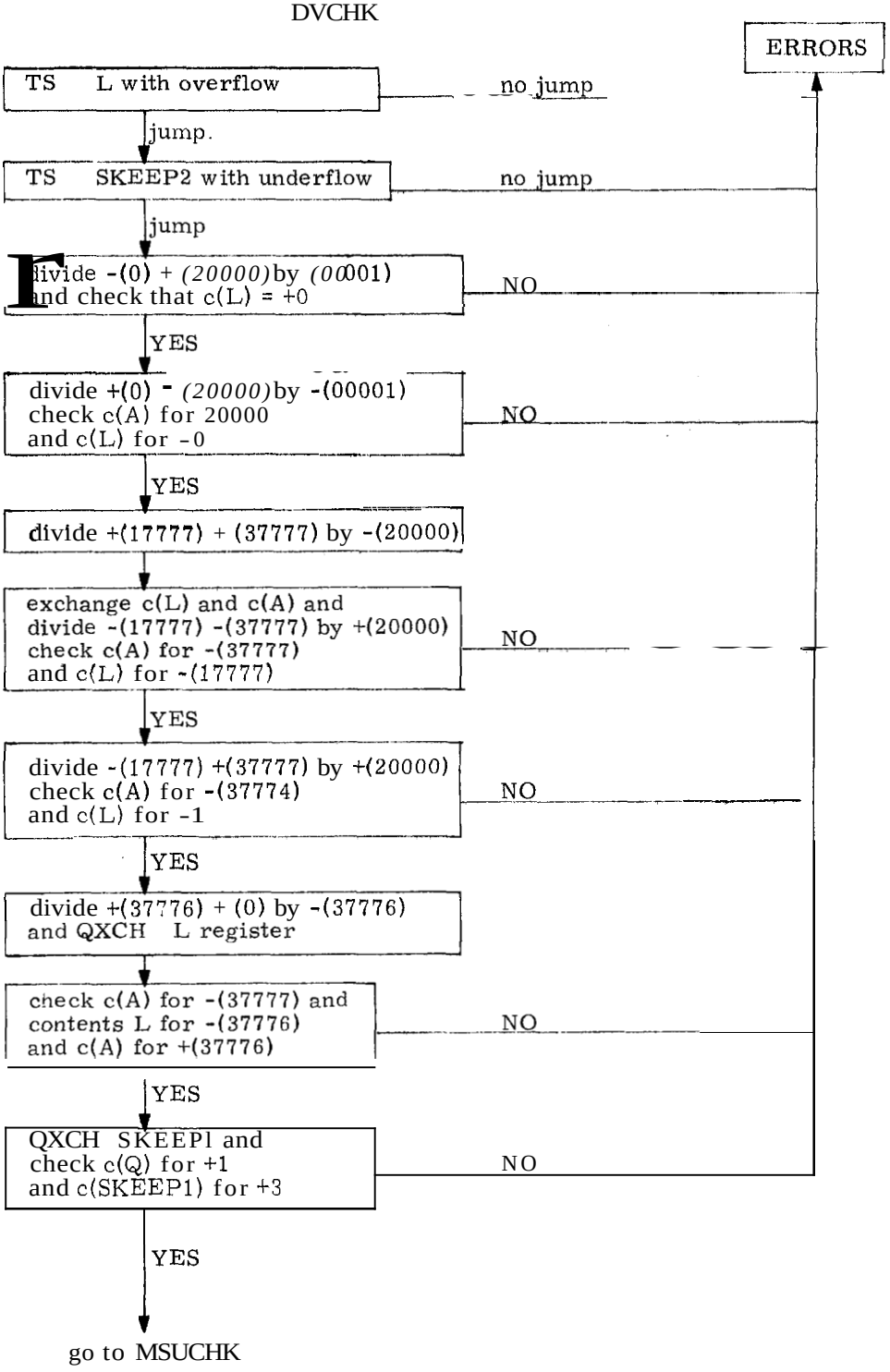
DAS+INCR


ERRORS
put -1 in L register
and +2 in A register
DAS A and check c(A)
for +4 and c(L) for -2
NO
YES
put 3777 in SKEEP3
put 40000 in SKEEP4
put -2 in L register
put +3 in A register
DAS SKEEP3
check c(A) for +0
check c(L) for +1
check c(SKEEP3) for +1
check c(SKEEP4) for -1
NO
YES
INCR -1(SKEEP4)
and check for -0
NO
YES
INCR +0 and check
for +1
put 40000 in SKEEP1
put 37777 in SKEEP2
put +3 in L register
put -3 in A register
DAS SKEEP1
check c(A) for -1
check c(SKEEP1) for +1
check c(SKEEP2) for -2
NO
YES
go to MPCHK

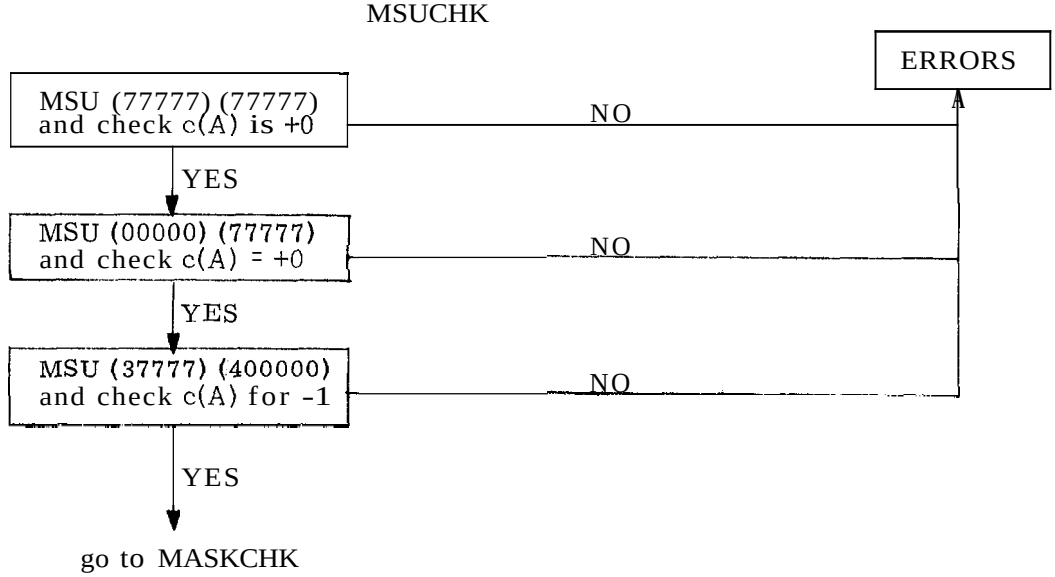
MPCHK


ERRORS
put +1 in A register and AUG A register
multiply +2 by 37777
AD L and multiply 37777 by -2
ADS L and check c(A) is 40000
NO
YES
multiply 40000 by +2
put -1 in SKEEPG, AUG SKEEP6 and use in next multiply
multiply 40000 by -2 check c(A) for +1 and c(L) for 37776
NO
YES
put -1 in A register -2 is in SKEEP6
ADS SKEEP6 and check c(A) and c(SKEEP6) for -1
NO
YES
go to DVCHK

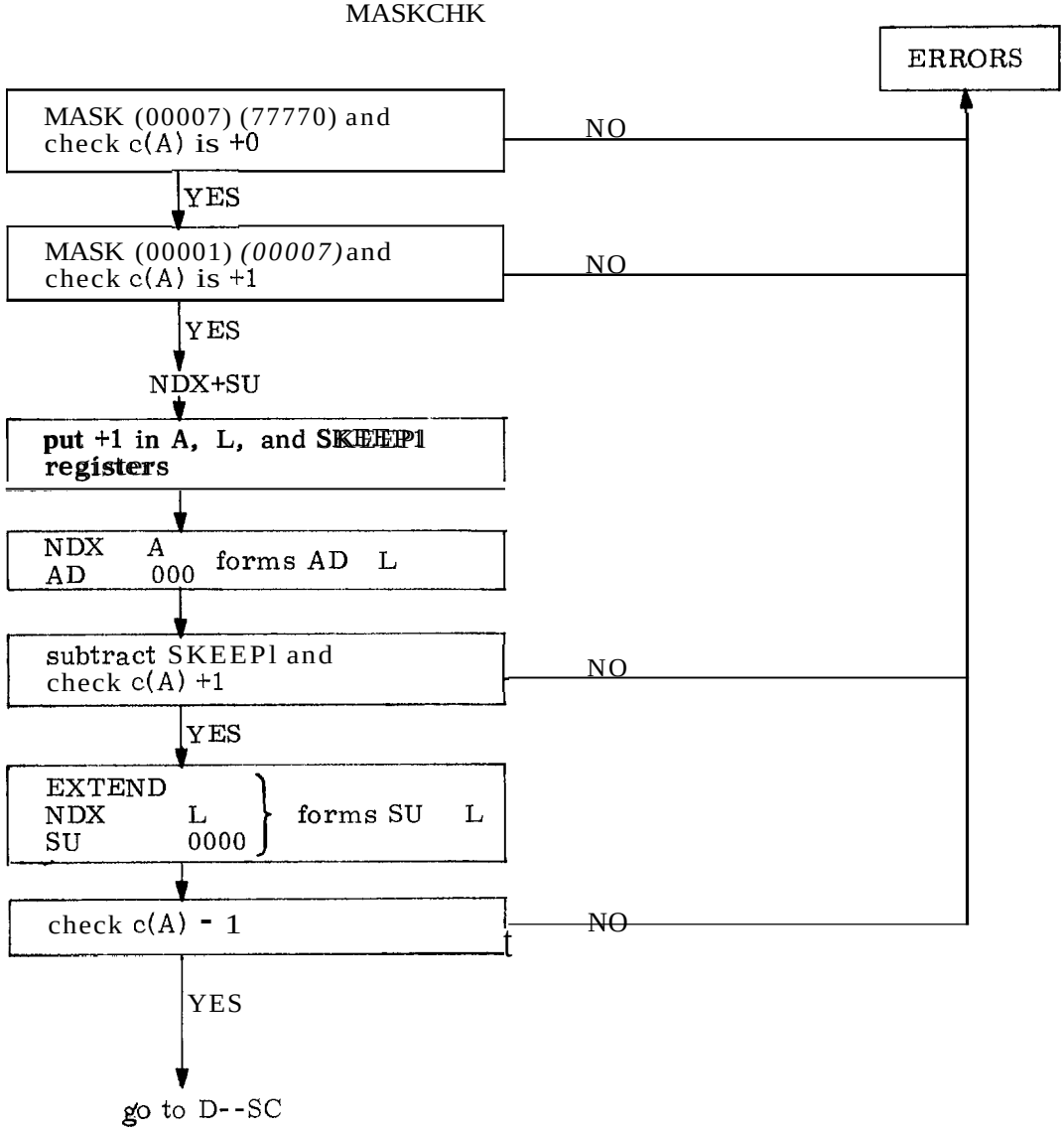
DVCHK


ERRORS
TS L with overflow
no jump
jump.
TS SKEEP2 with underflow
no jump
jump
Divide -(0) + (20000) by (00001)
and check that c(L) = +0
NO
YES
divide +(0) - (20000) by -(00001)
check c(A) for 20000
and c(L) for -0
NO
YES
divide +(17777) + (37777) by -(20000)
exchange c(L) and c(A) and
divide -(17777) -(37777) by +(20000)
check c(A) for -(37777)
and c(L) for -(17777)
NO
YES
divide -(17777) +(37777) by +(20000)
check c(A) for -(37774)
and c(L) for -1
NO
YES
divide +(37776) + (0) by -(37776)
and QXCH L register
check c(A) for -(37777) and
contents L for -(37776)
and c(A) for +(37776)
NO
YES
QXCH SKEEP1 and
check c(Q) for +1
and c(SKEEP1) for +3
NO
YES
go to MSUCHK

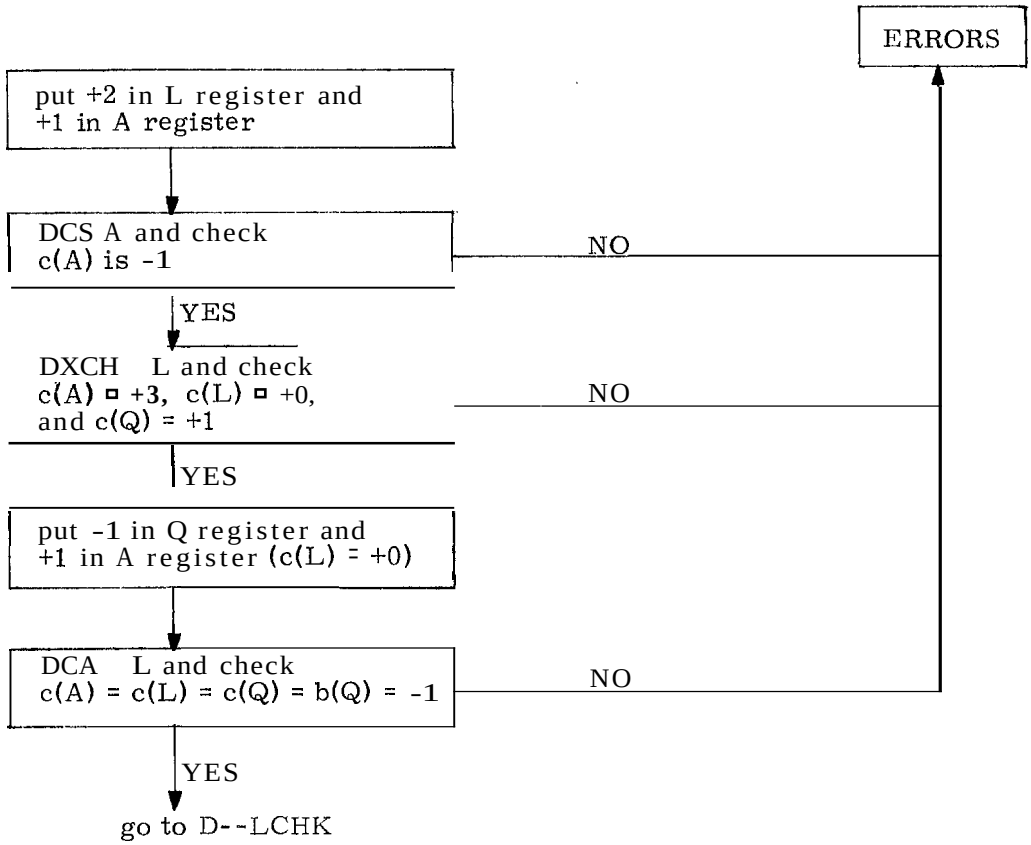
MSUCHK


ERRORS
MSU (77777) (77777)
and check c(A) is +0
NO
YES
MSU (00000) (77777)
and check c(A) = +0
NO
YES
MSU (37777) (400000)
and check c(A) for -1
NO
YES
go to MASKCHK

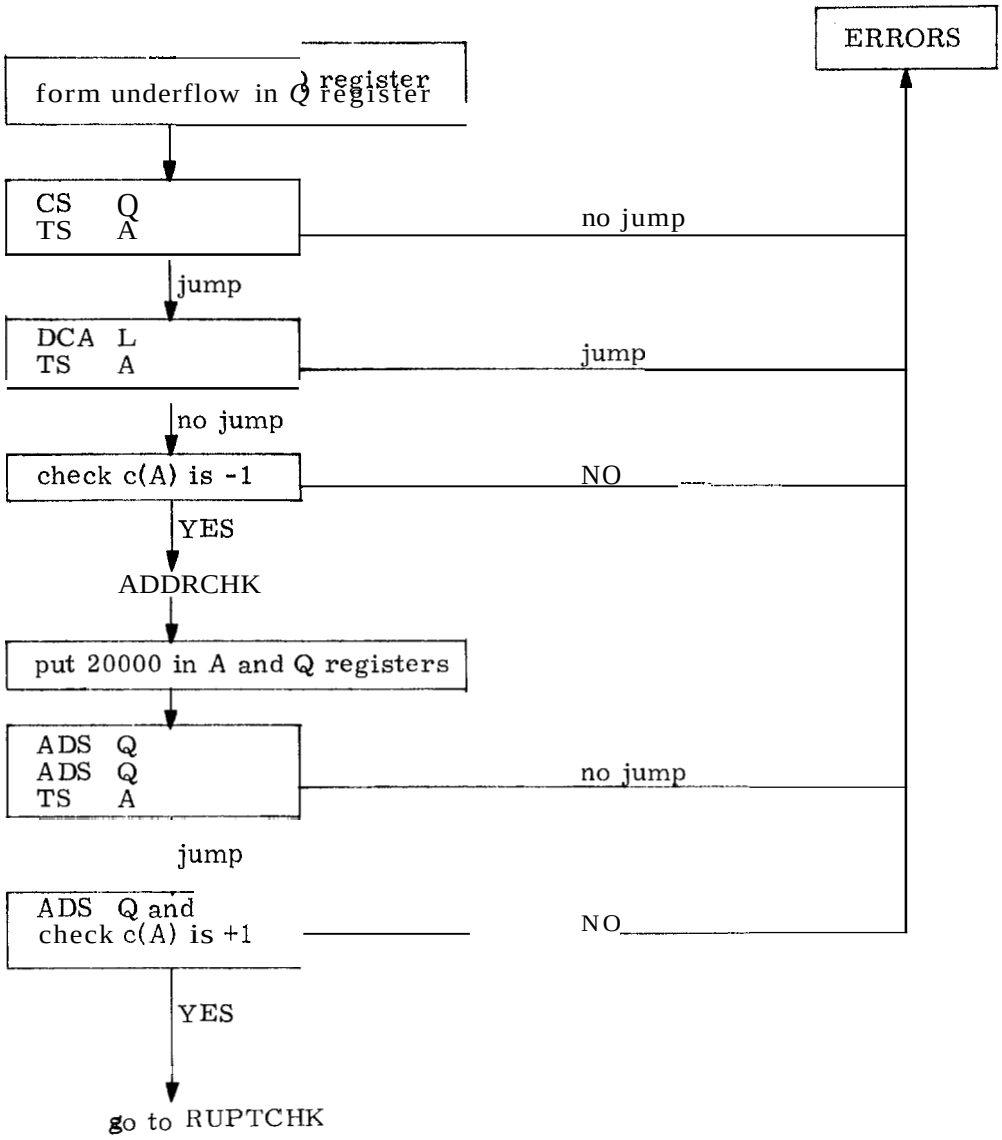
MASKCHK


ERRORS
MASK (00007) (77770) and check c(A) is +0
NO
YES
MASK (00001) (00007) and check c(A) is +1
NO
YES
NDX+SU
put +1 in A, L, and SKEEP1 registers
NDX A
AD 000
forms AD L
subtract SKEEP1 and check c(A) +1
NO
YES
EXTEND
NDX L
SU 0000
forms SU L
check c(A) - 1
NO
YES
go to D--SC

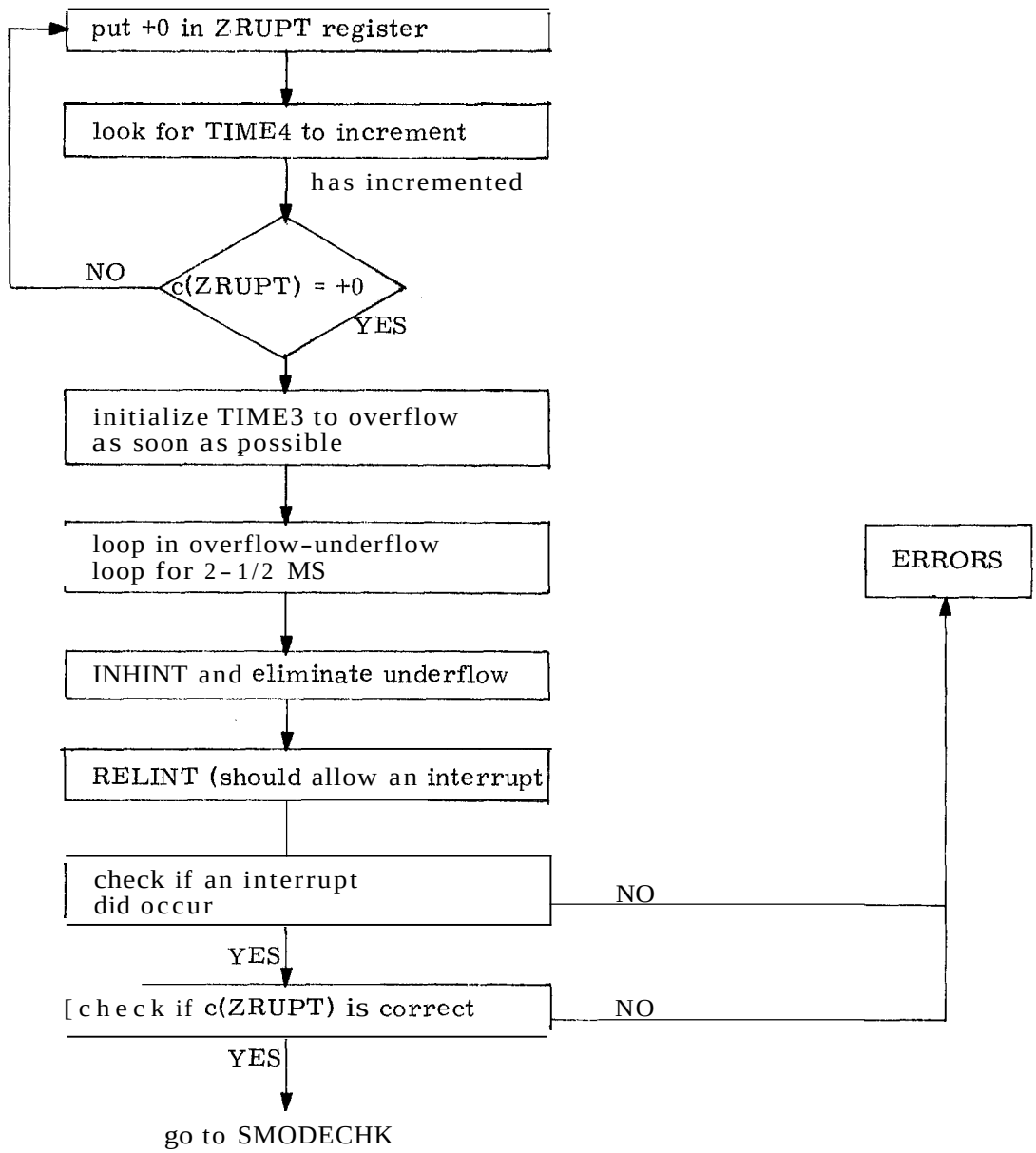
D--SC


ERRORS
put +2 in L register and
+1 in A register
DCS A and check
c(A) is -1
NO
YES
DXCH L and check
c(A) ¤ +3, c(L) ¤ +0,
and c(Q) = +1
NO
YES
put -1 in Q register and
+1 in A register (c(L) = +0)
DCA L and check
c(A) = c(L) = c(Q) = b(Q) = -1
NO
YES
go to D--LCHK

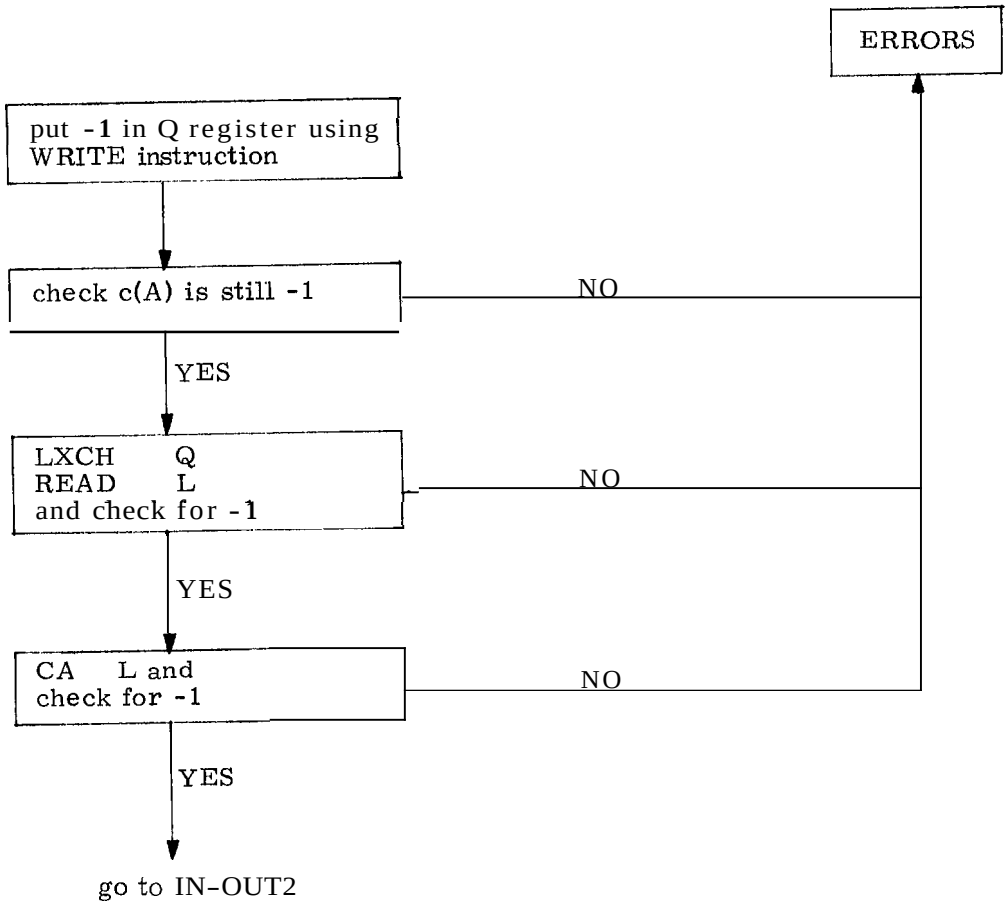
D--LCHK


ERRORS
form underflow in Q register
CS Q
TS A
no jump
jump
DCA L
TS A
jump
no jump
check c(A) is -1
NO
YES
ADDRCHK
put 20000 in A and Q registers
ADS Q
ADS Q
TS A
no jump
jump
ADS Q and
check c(A) is +1
NO
YES
go to RUPTCHK

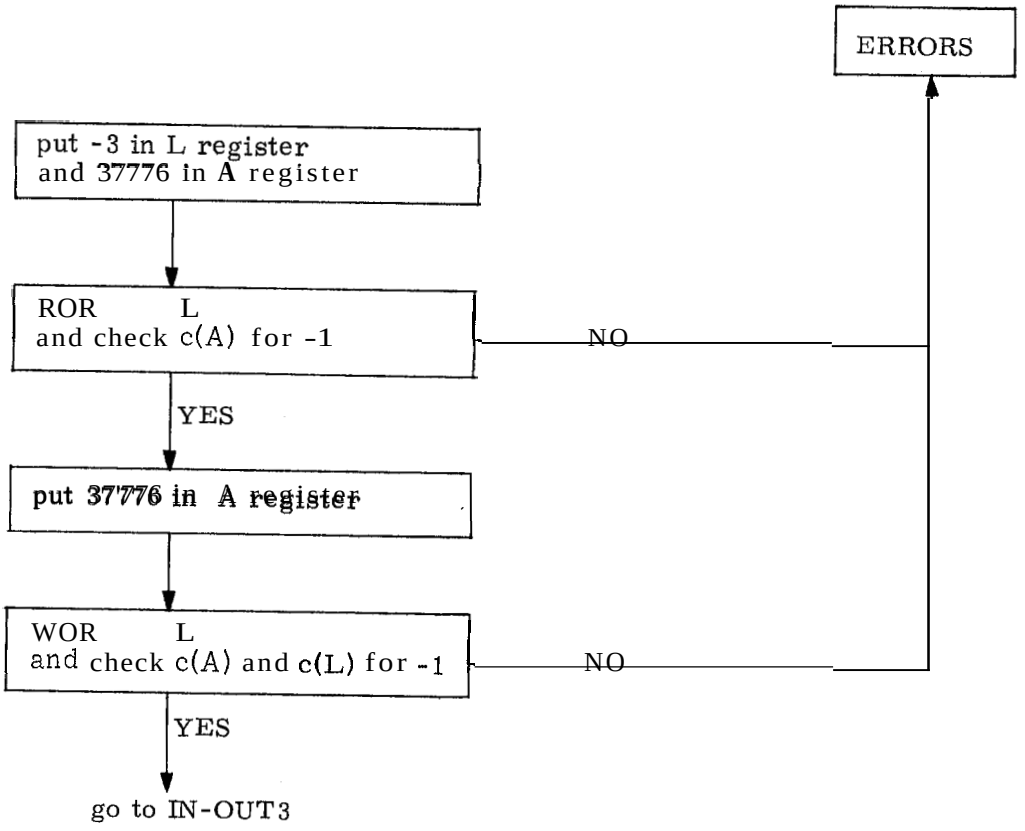
RUPTCHK


put +0 in ZRUPT register
look for TIME4 to increment
has incremented
NO
c(ZRUPT) = +0
YES
initialize TIME3 to overflow as soon as possible
loop in overflow-underflow loop for 2-1/2 MS
INHINT and eliminate underflow
RELINT (should allow an interrupt
check if an interrupt did occur
NO
YES
[check if c(ZRUPT) is correct
NO
YES
go to SMODECHK
ERRORS

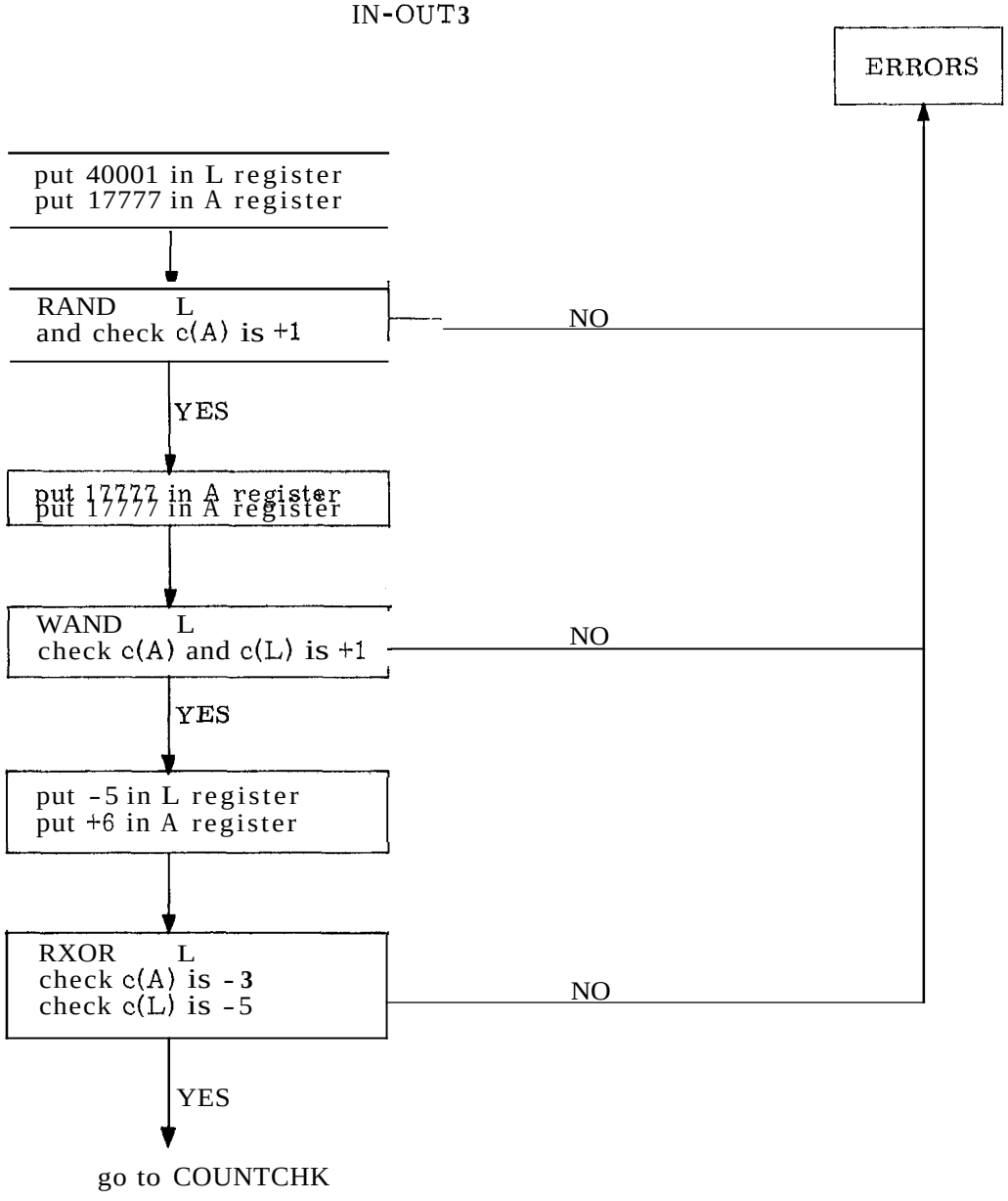
IN-OUT1


ERRORS
put -1 in Q register using WRITE instruction
check c(A) is still -1
NO
YES
LXCH Q
READ L
and check for -1
NO
YES
CA L and
check for -1
NO
YES
go to IN-OUT2

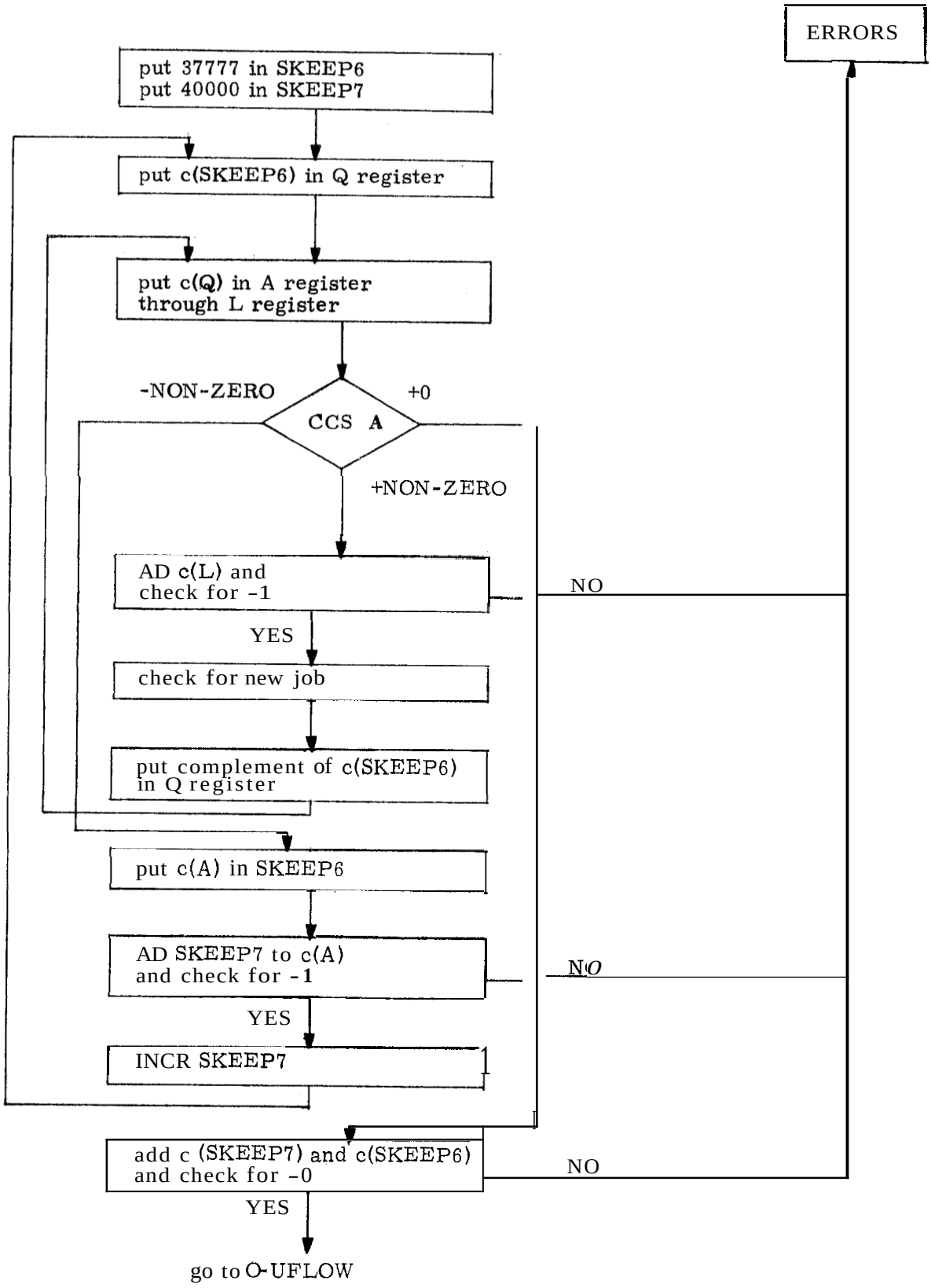
IN-OUT2

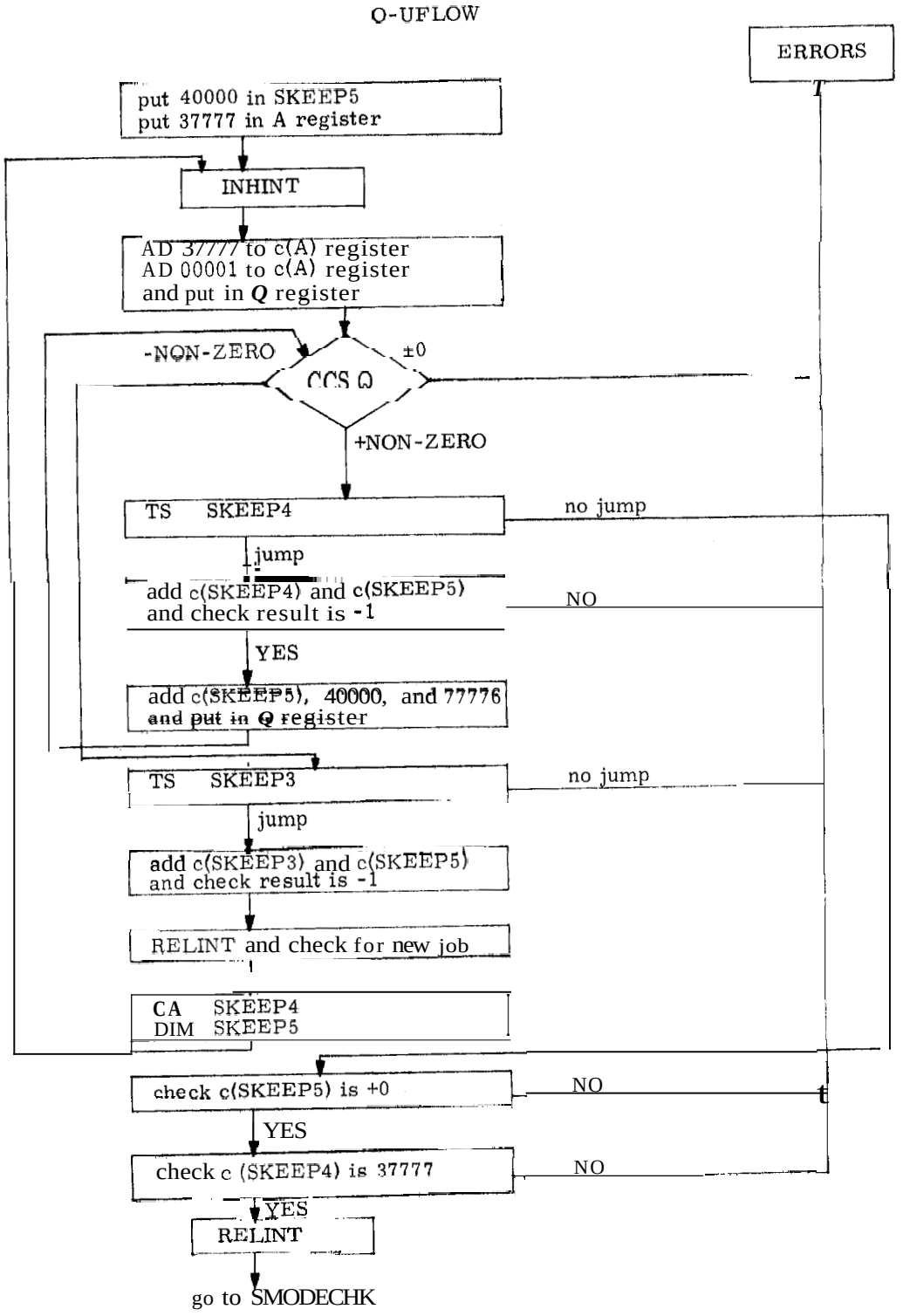

ERRORS
put -3 in L register
and 37776 in A register
ROR L
and check c(A) for -1
NO
YES
put 37776 in A register
WOR L
and check c(A) and c(L) for -1
NO
YES
go to IN-OUT3

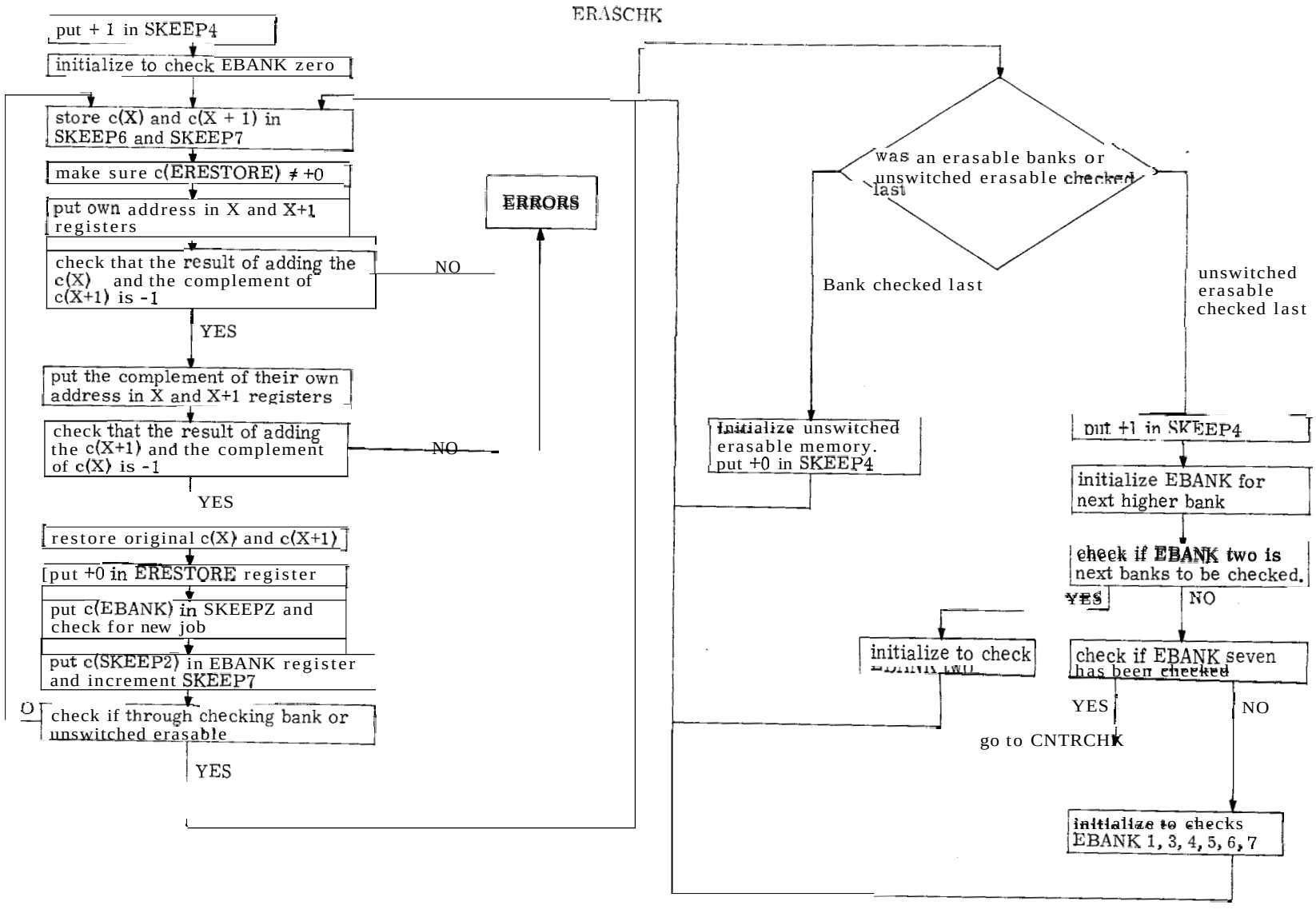
IN-OUT3


ERRORS
put 40001 in L register
put 17777 in A register
RAND L
and check c(A) is +1
NO
YES
put 17777 in A register
put 17777 in A register
WAND L
check c(A) and c(L) is +1
NO
YES
put -5 in L register
put +6 in A register
RXOR L
check c(A) is -3
check c(L) is -5
NO
YES
go to COUNTCHK

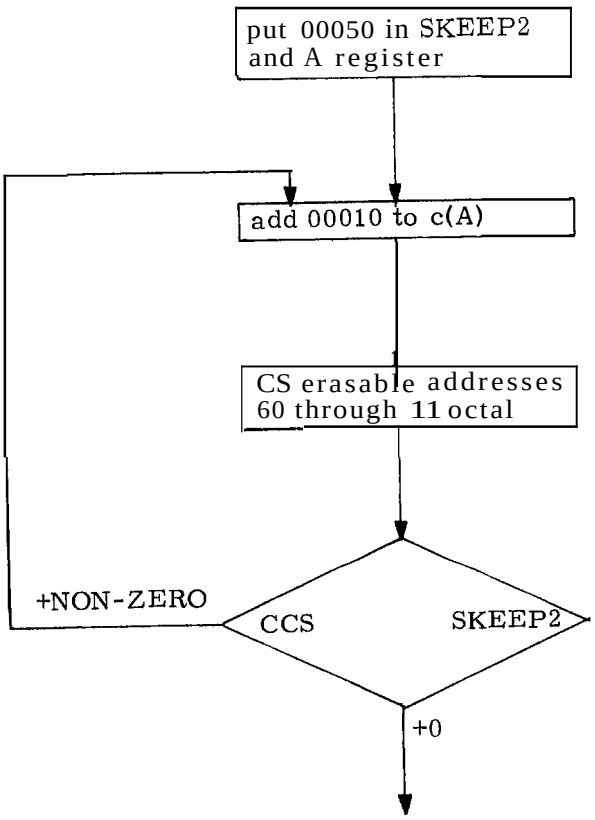
COUNTCHK


ERRORS
put 37777 in SKEEP6
put 40000 in SKEEP7
put c(SKEEP6) in Q register
put c(Q) in A register
through L register
-NON-ZERO
+0
CCS A
+NON-ZERO
AD c(L) and
check for -1
NO
YES
check for new job
put complement of c(SKEEP6)
in Q register
put c(A) in SKEEP6
AD SKEEP7 to c(A)
and check for -1
NO
YES
INCR SKEEP7
add c (SKEEP7) and c(SKEEP6)
and check for -0
NO
YES
go to O-UFLOW

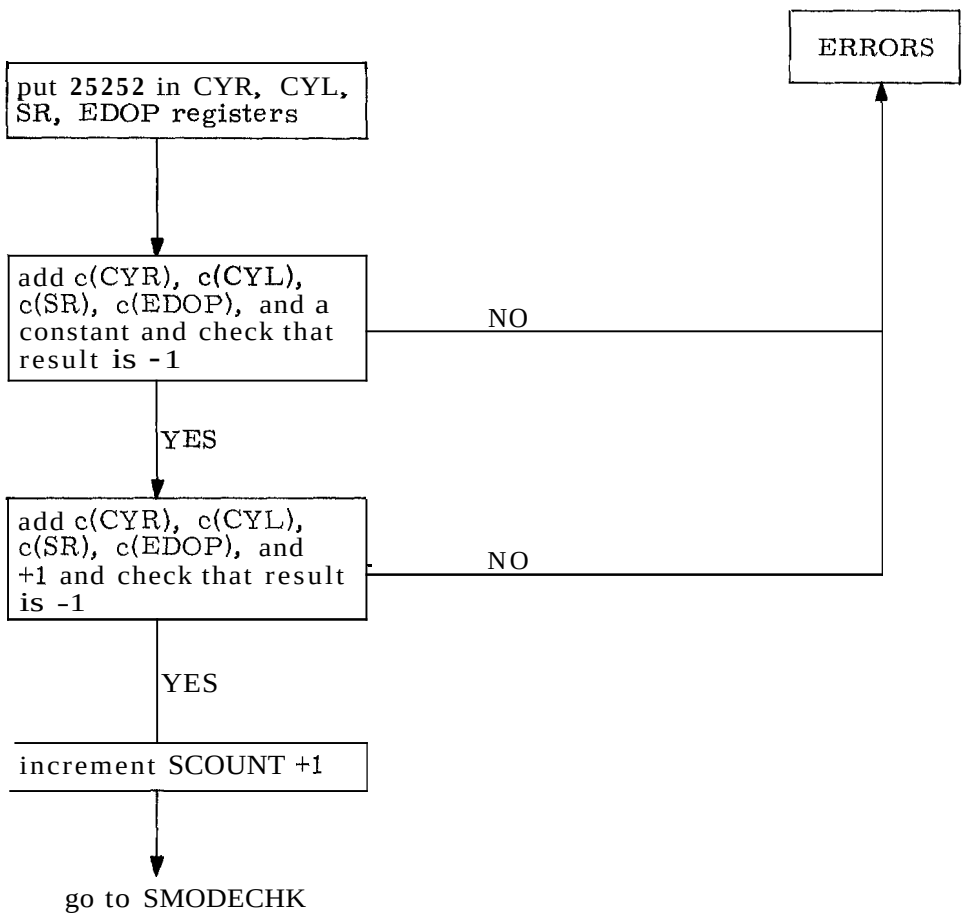

Q-UFLOW
ERRORS
put 40000 in SKEEP5
put 37777 in A register
INHINT
AD 37777 to c(A) register
AD 00001 to c(A) register
and put in Q register
-NON-ZERO
±0
CCS Q
+NON-ZERO
TS SKEEP4
no jump
jump
add c(SKEEP4) and c(SKEEP5)
and check result is -1
NO
YES
add c(SKEEP5), 40000, and 77776
and put in Q register
TS SKEEP3
no jump
jump
add c(SKEEP3) and c(SKEEP5)
and check result is -1
RELINT and check for new job
CA SKEEP4
DIM SKEEP5
check c(SKEEP5) is +0
NO
YES
check c (SKEEP4) is 37777
NO
YES
RELINT
go to SMODECHK

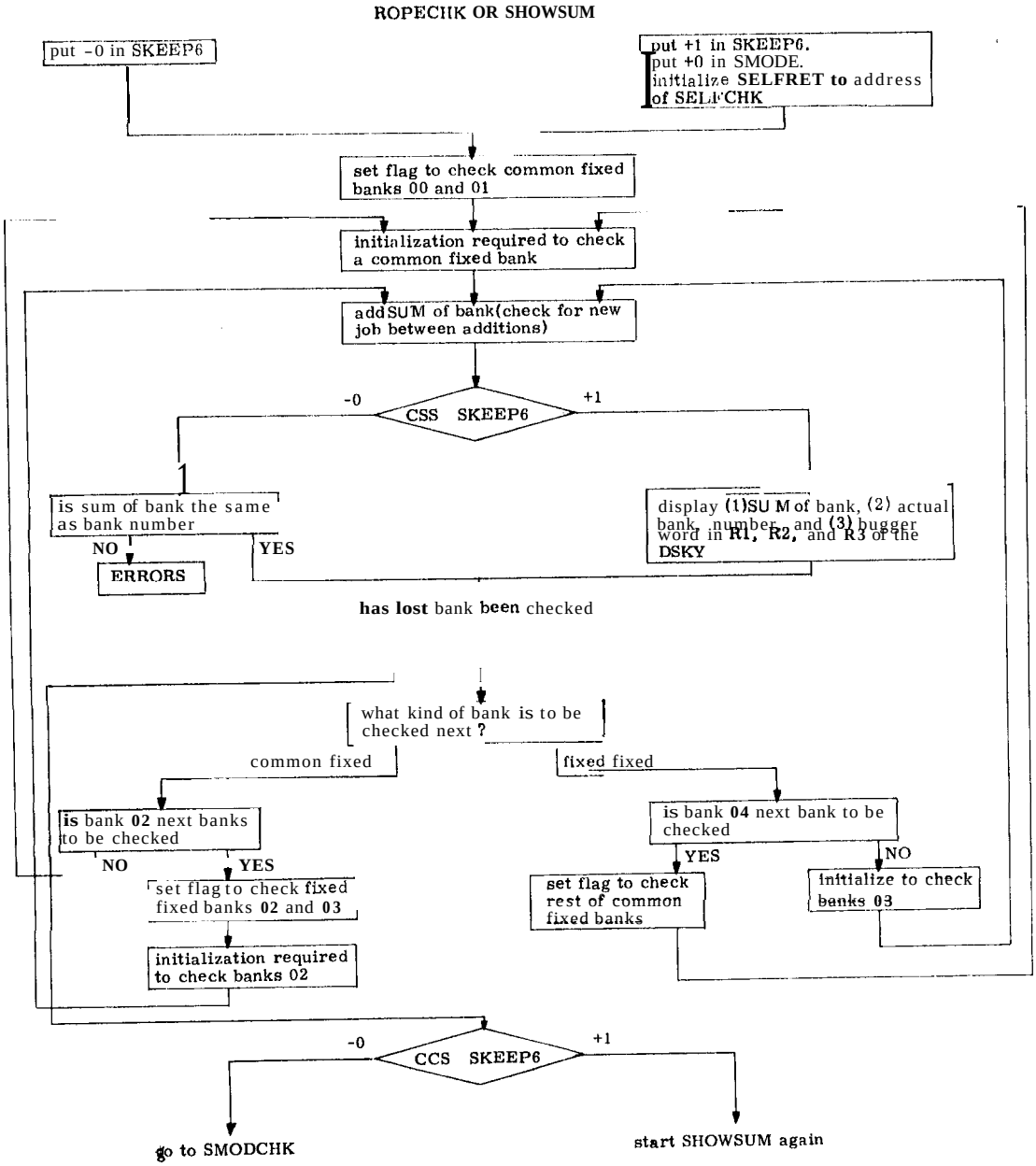

ERASCHK
put + 1 in SKEEP4
initialize to check EBANK zero
store c(X) and c(X + 1) in SKEEP6 and SKEEP7
make sure c(ERESTORE) ≠ +0
put own address in X and X+1 registers
check that the result of adding the c(X) and the complement of c(X+1) is -1
NO
ERRORS
YES
put the complement of their own address in X and X+1 registers
check that the result of adding the c(X+1) and the complement of c(X) is -1
NO
YES
restore original c(X) and c(X+1)
put +0 in ERESTORE register
put c(EBANK) in SKEEPZ and check for new job
put c(SKEEP2) in EBANK register and increment SKEEP7
check if through checking bank or unswitched erasable
YES
was an erasable banks or unswitched erasable checked last
Bank checked last
unswitched erasable checked last
initialize unswitched erasable memory. put +0 in SKEEP4
put +1 in SKEEP4
initialize EBANK for next higher bank
check if EBANK two is next banks to be checked.
YES
NO
initialize to check
check if EBANK seven has been checked
YES
NO
go to CNTRCHK
initialize to checks EBANK 1, 3, 4, 5, 6, 7

CNTRCHK


put 00050 in SKEEP2
and A register
add 00010 to c(A)
CS erasable addresses
60 through 11 octal
+NON-ZERO
CCS
SKEEP2
+0
go to CYCLSHFT

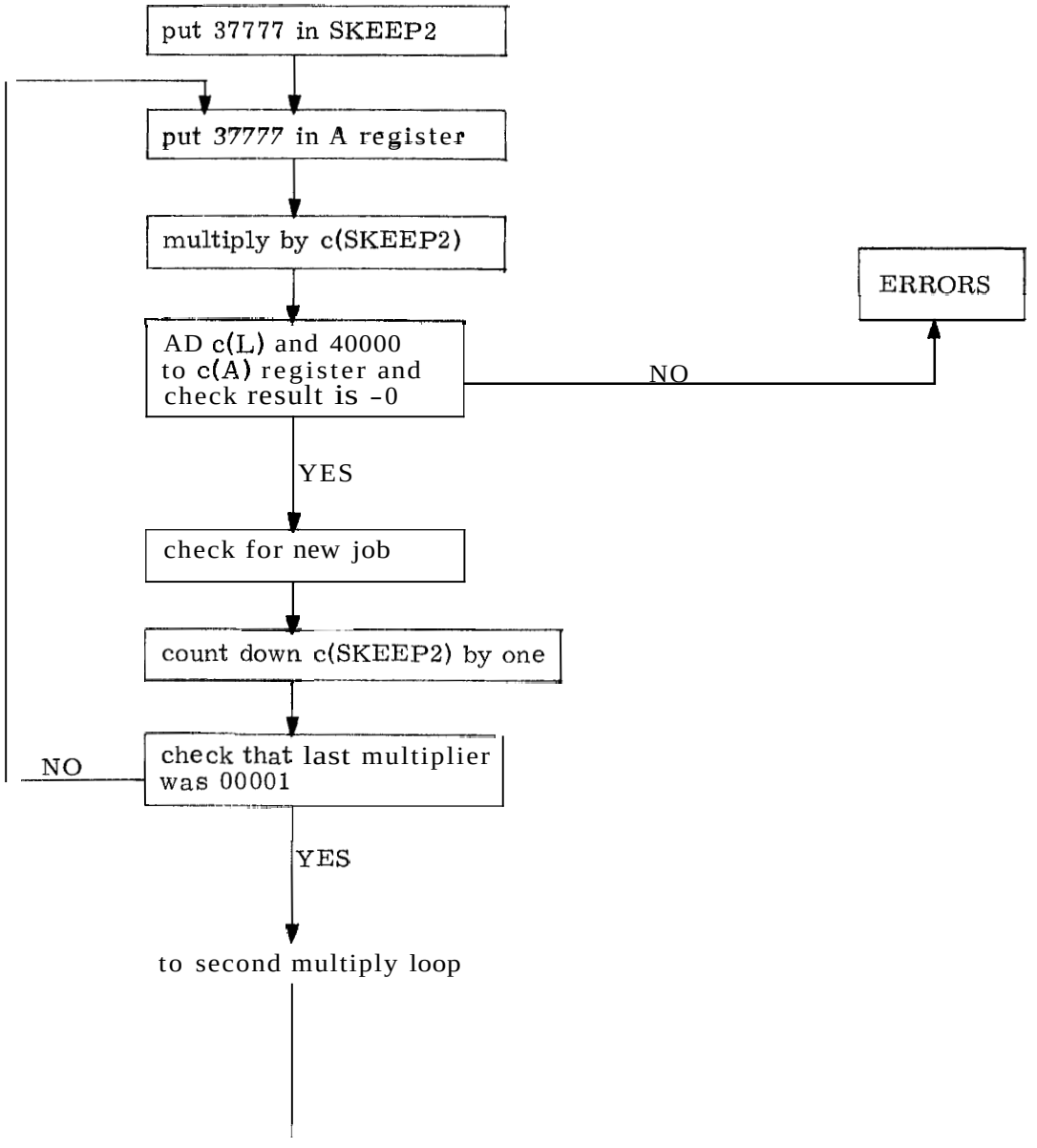
CYCLSHFT


ERRORS
put 25252 in CYR, CYL, SR, EDOP registers
add c(CYR), c(CYL), c(SR), c(EDOP), and a constant and check that result is -1
NO
YES
add c(CYR), c(CYL), c(SR), c(EDOP), and +1 and check that result is -1
NO
YES
increment SCOUNT +1
go to SMODECHK

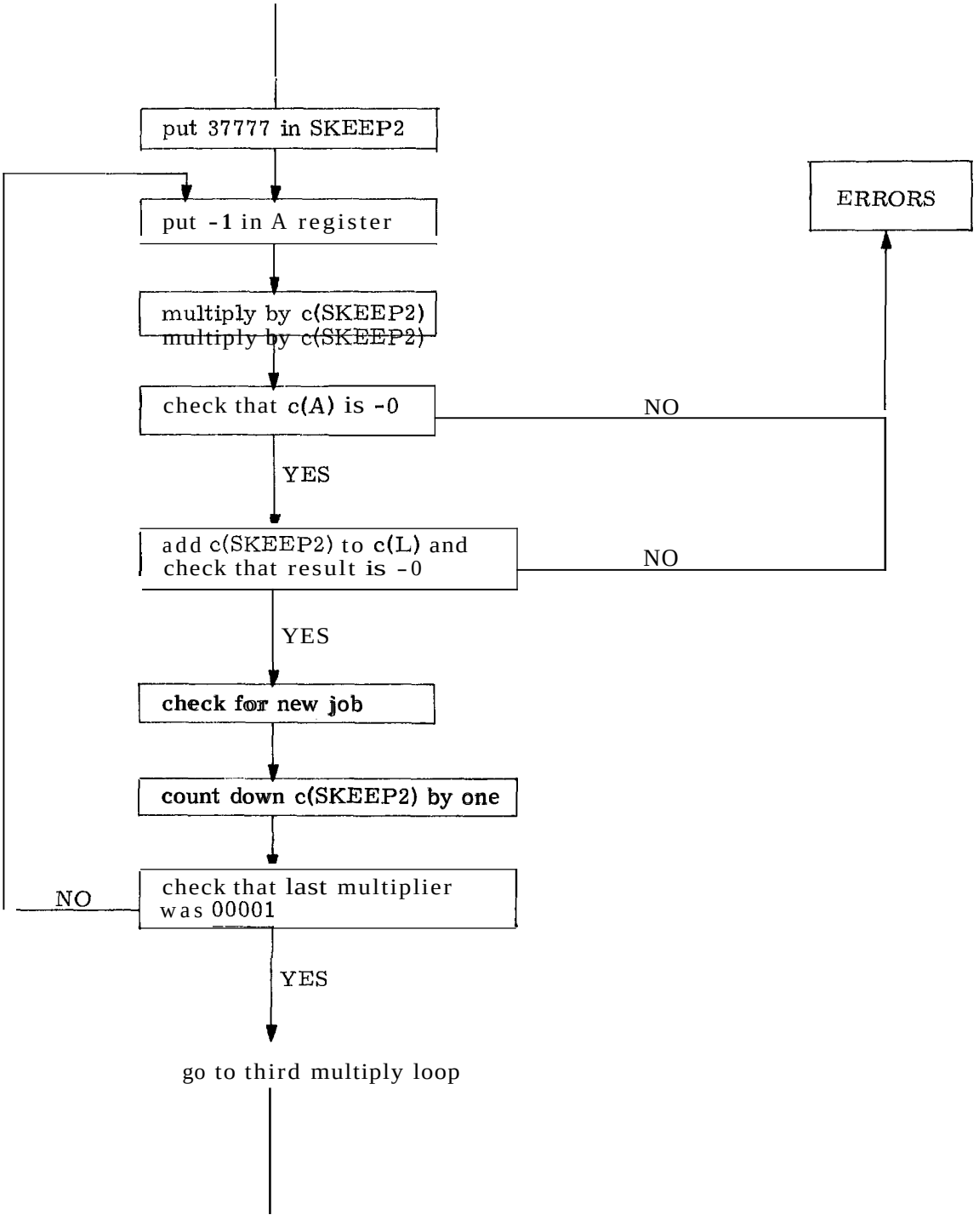

ROPECHK OR SHOWSUM
put -0 in SKEEP6
put +1 in SKEEP6.
put +0 in SMODE.
initialize SELFRET to address of SELFCHK
set flag to check common fixed banks 00 and 01
initialization required to check a common fixed bank
add SUM of bank (check for new job between additions)
-0
CSS SKEEP6
+1
is sum of bank the same as bank number
NO
YES
ERRORS
display (1) SUM of bank, (2) actual bank number, and (3) bugger word in R1, R2, and R3 of the DSKY
has lost bank been checked
what kind of bank is to be checked next ?
common fixed
fixed fixed
is bank 02 next banks to be checked
NO
YES
set flag to check fixed fixed banks 02 and 03
initialization required to check banks 02
is bank 04 next bank to be checked
YES
NO
set flag to check rest of common fixed banks
initialize to check banks 03
-0
CCS SKEEP6
+1
go to SMODCHK
start SHOWSUM again

MPNMBRS

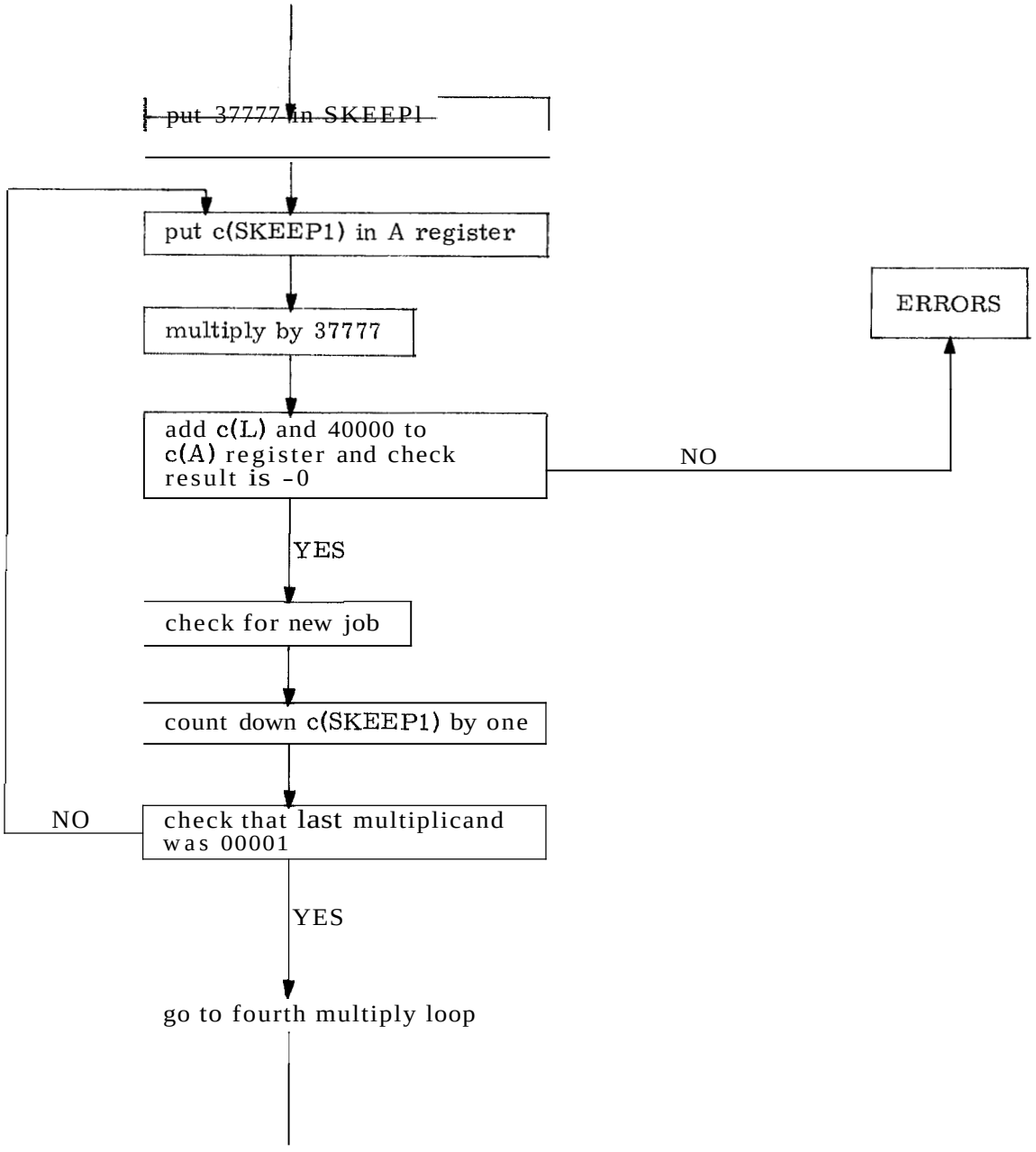
(20 second multiplier check)



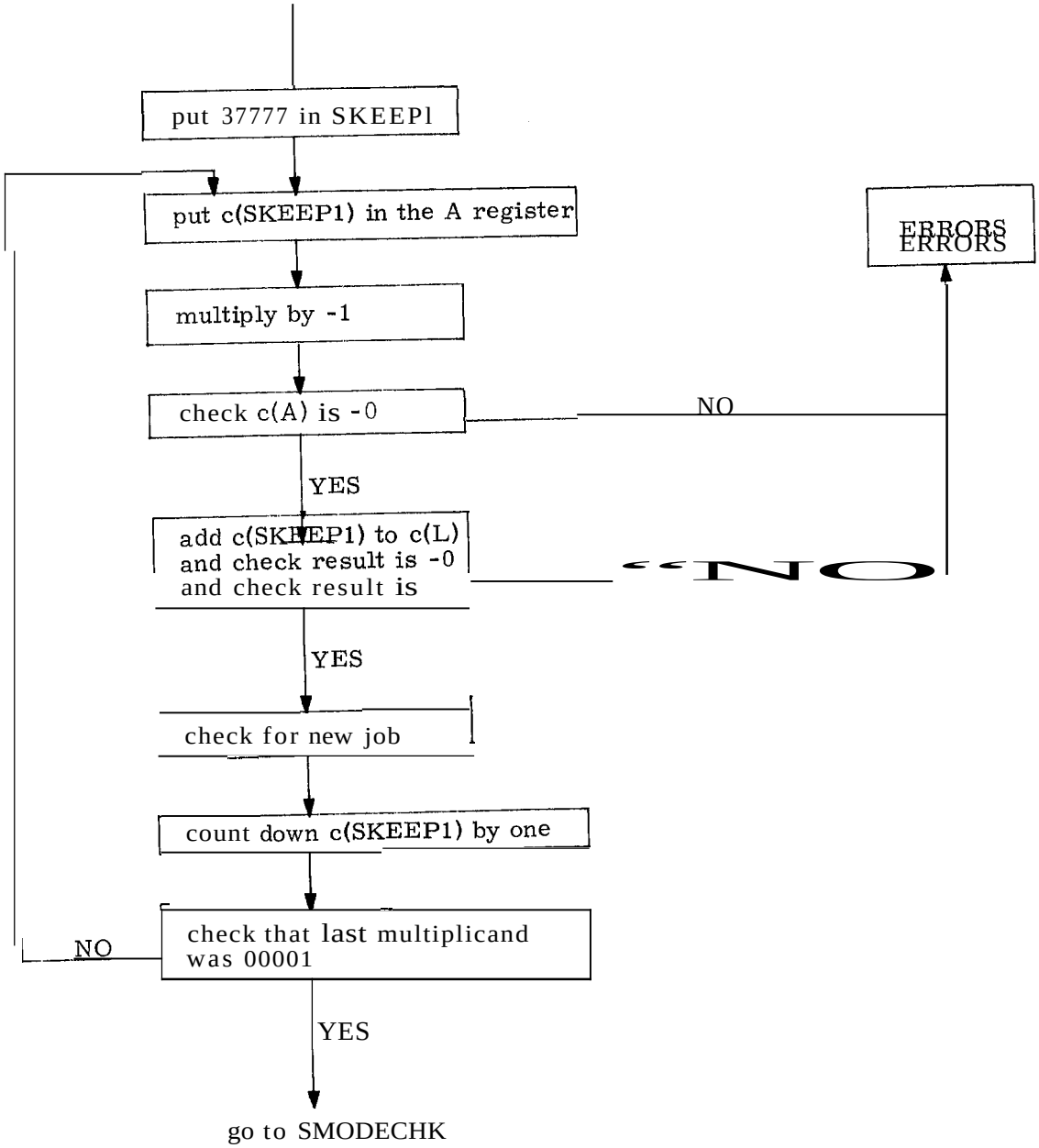
MPNMBRS (Cont'd)

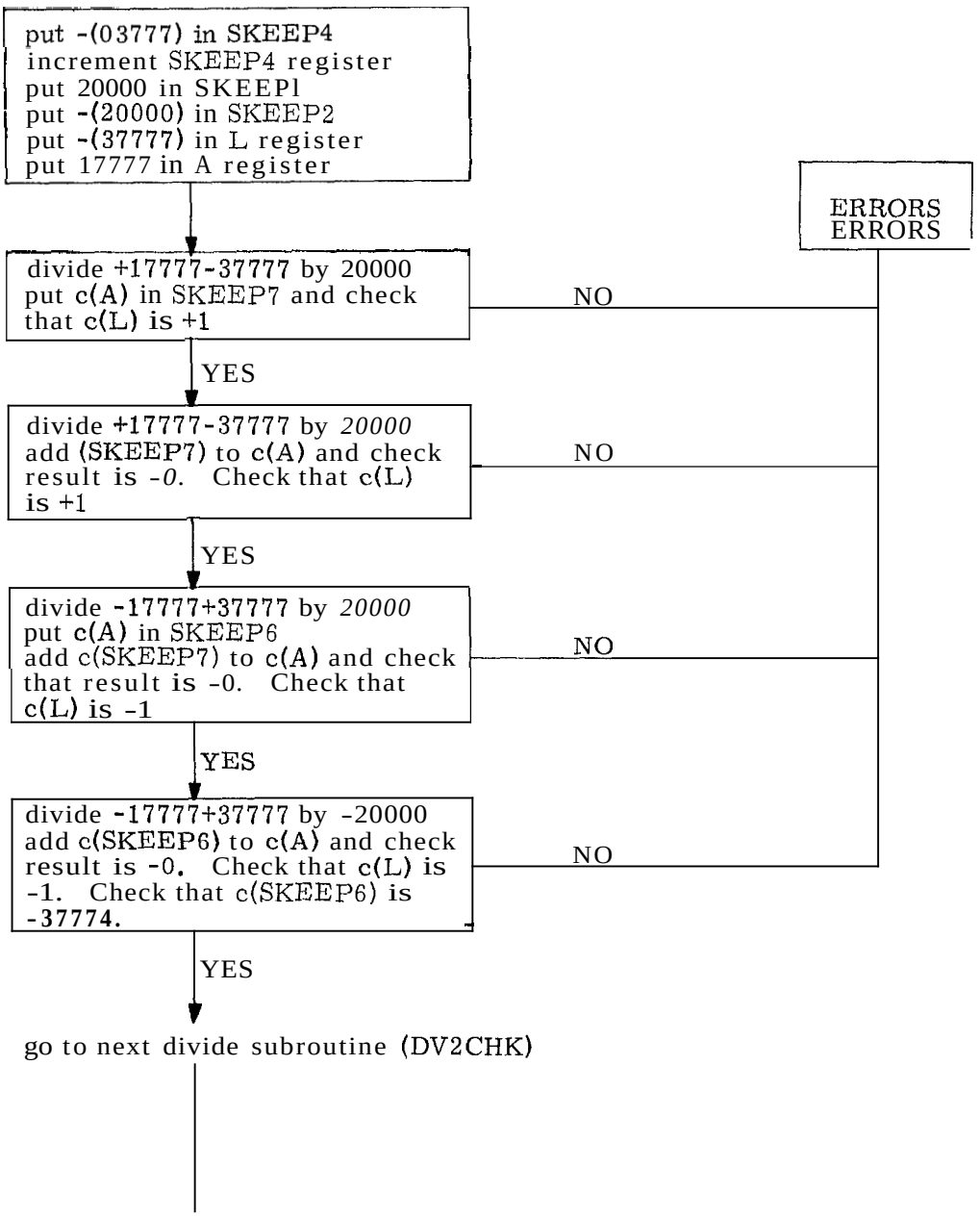


MPNMBRS (Cont'd)

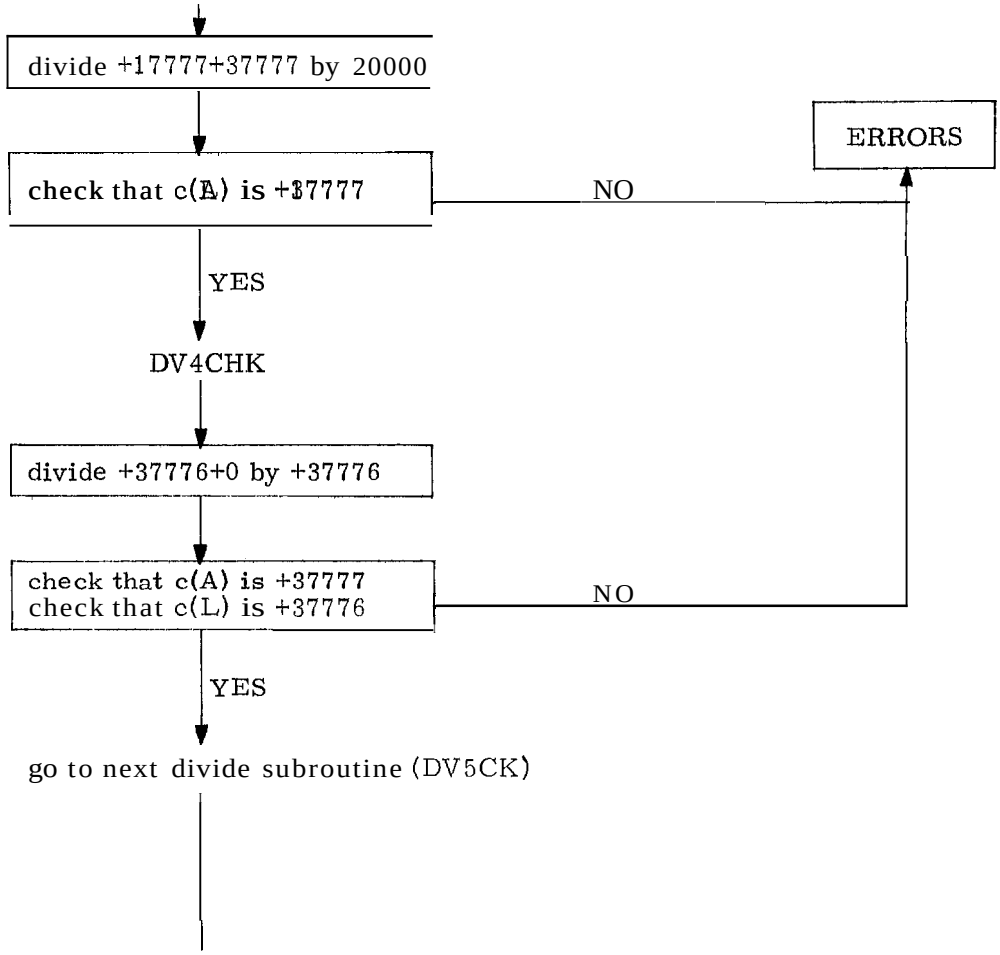


MPNMBRS (Cont'd)

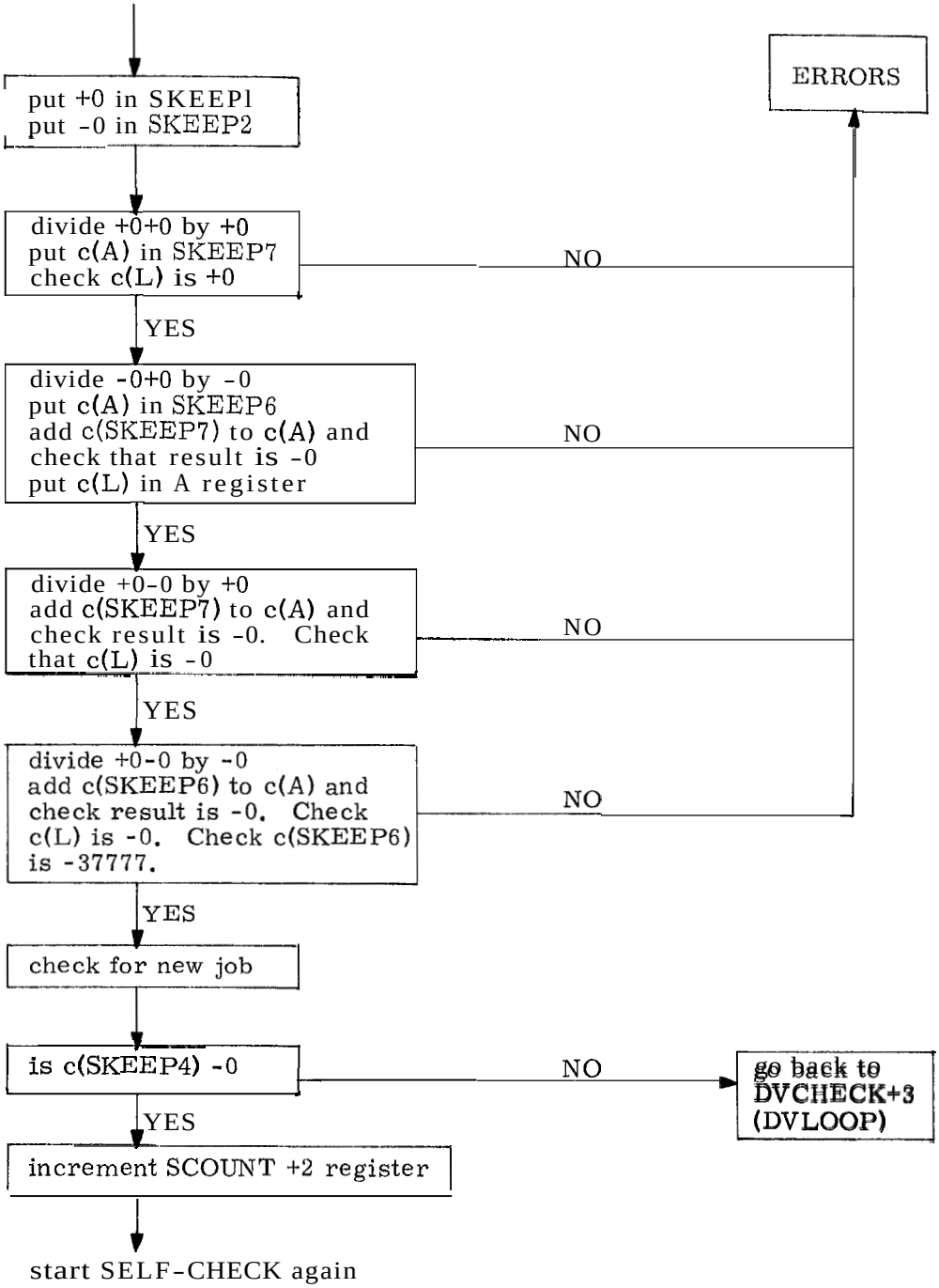



put -(03777) in SKEEP4
increment SKEEP4 register
put 20000 in SKEEPl
put -(20000) in SKEEP2
put -(37777) in L register
put 17777 in A register
ERRORS
ERRORS
divide +17777-37777 by 20000
put c(A) in SKEEP7 and check
that c(L) is +1
NO
YES
divide +17777-37777 by 20000
add (SKEEP7) to c(A) and check
result is -0. Check that c(L)
is +1
NO
YES
divide -17777+37777 by 20000
put c(A) in SKEEP6
add c(SKEEP7) to c(A) and check
that result is -0. Check that
c(L) is -1
NO
YES
divide -17777+37777 by -20000
add c(SKEEP6) to c(A) and check
result is -0. Check that c(L) is
-1. Check that c(SKEEP6) is
-37774.
NO
YES
go to next divide subroutine (DV2CHK)

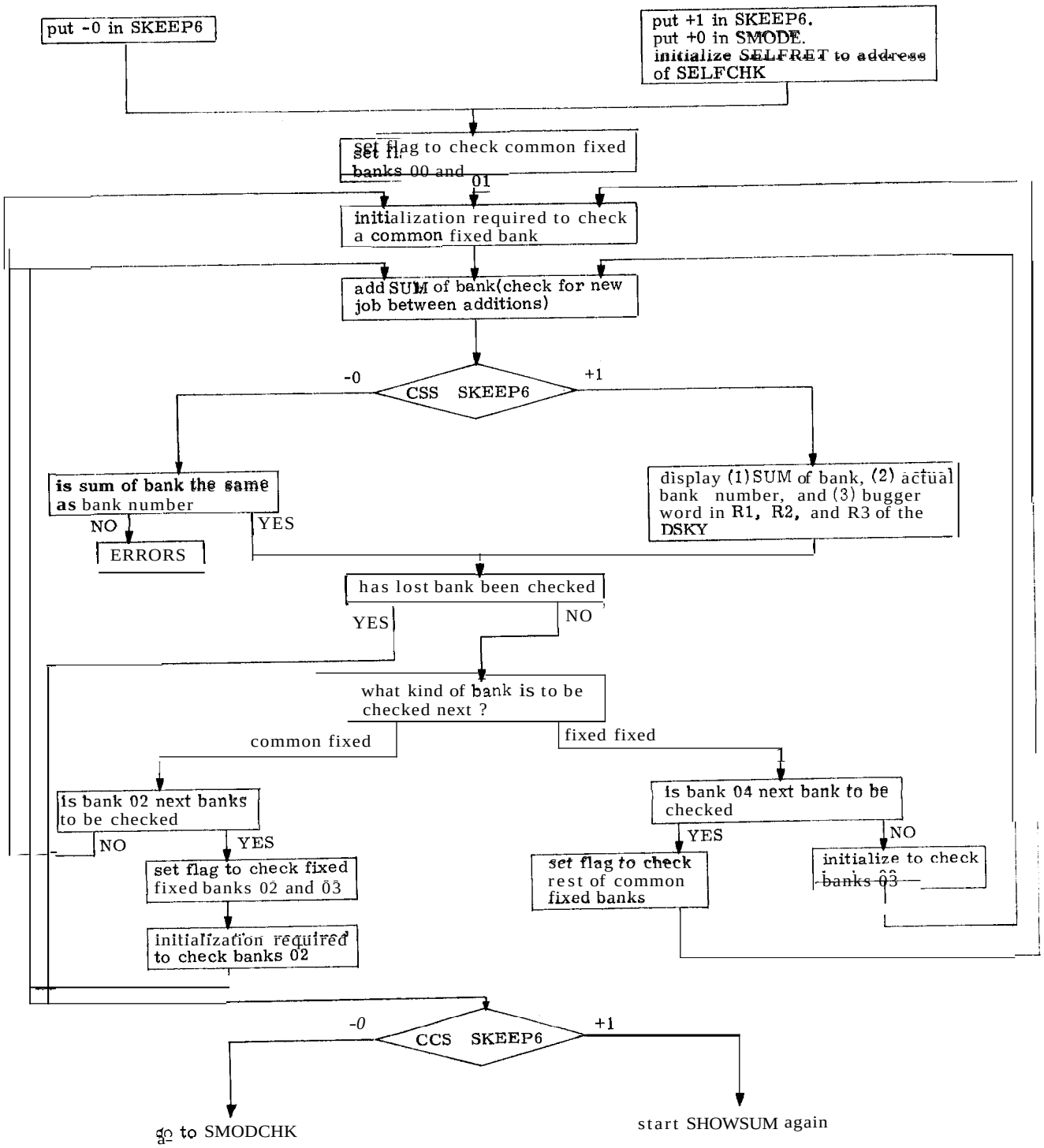
DVCHECK (Cont'd)



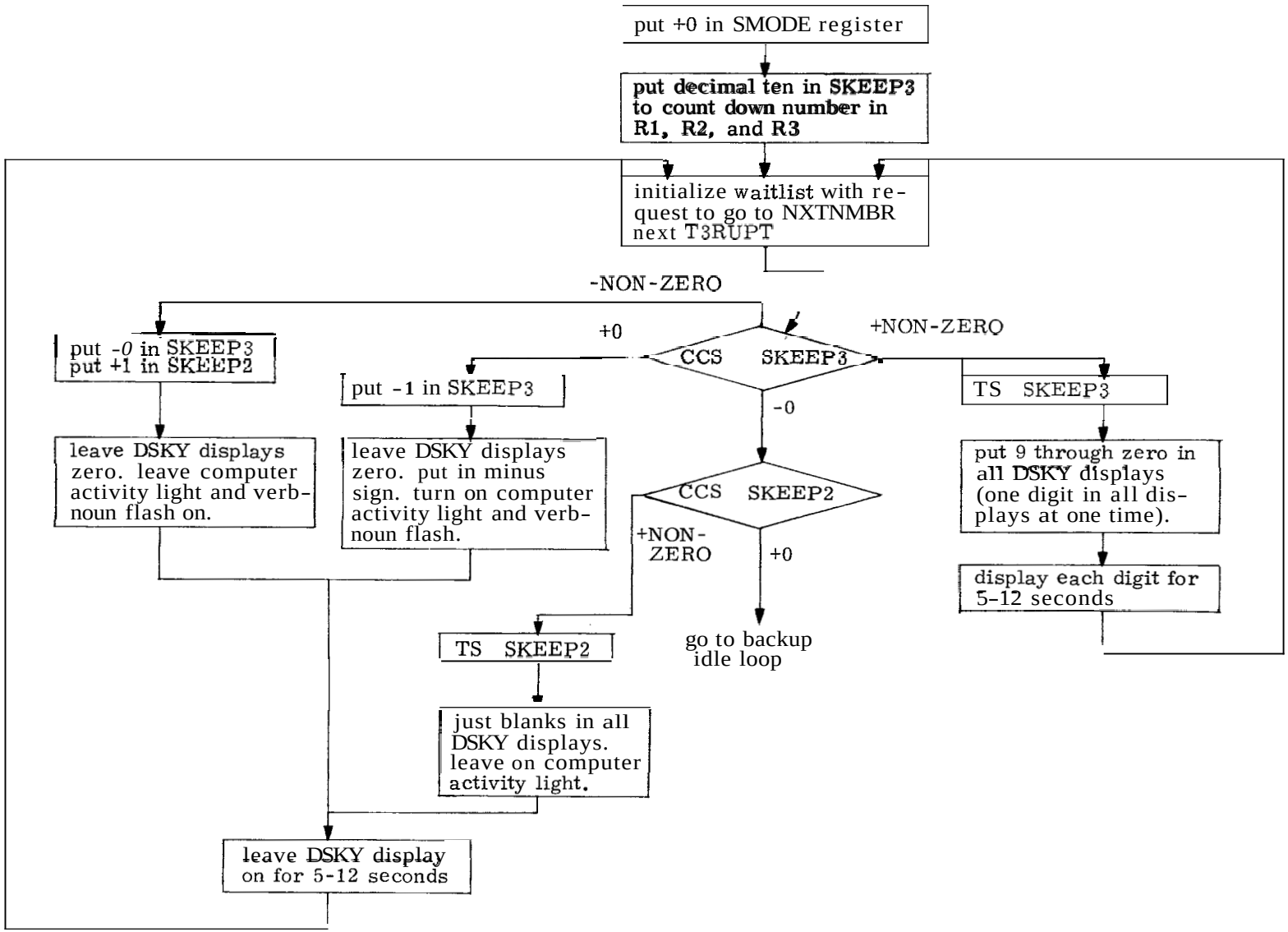
DVCHECK (Cont'd)



ROPECHK OR SHOWSUM



DSKYCHK


put +0 in SMODE register
put decimal ten in SKEEP3 to count down number in R1, R2, and R3
initialize waitlist with request to go to NXTNMBR next T3RUPT
-NON-ZERO
+0
+NON-ZERO
CCS SKEEP3
-0
put -0 in SKEEP3
put +1 in SKEEP2
put -1 in SKEEP3
TS SKEEP3
leave DSKY displays zero. leave computer activity light and verb-noun flash on.
leave DSKY displays zero. put in minus sign. turn on computer activity light and verb-noun flash.
CCS SKEEP2
put 9 through zero in all DSKY displays (one digit in all displays at one time).
+NON-ZERO
+0
display each digit for 5-12 seconds
TS SKEEP2
go to backup idle loop
just blanks in all DSKY displays. leave on computer activity light.
leave DSKY display on for 5-12 seconds

E-2065

DISTRIBUTION LIST

Internal

M. Adams (MIT/GAEC)
J. Alekshun
R. Alonso
R. Battin
H. Blair-Smith
P. Bowditch/F. Siraco
D. Bowler
R. Byers
G. Cherry
E. Copps
R. Crisp
J. Dahlen
J. DeLisle
G. Edmonds
J. B. Feldman
P. Felleman
S. Felix
J. Flanders
J. Fleming
F. Gaunt (3)
A. Green
F. Grant
Eldon Hall
T. Hemker (MIT/NAA)
D. Hoag
A. Hopkins
F. Houston
L. B. Johnson
M. Johnston
A. Kosmala
A. Laats
A. LaPointe
L. Larson
S. Laquideira
T. M. Lawton (MIT/MSC)
D. Lickly
G. Mayo
R. McKern
James Miller
John Miller
J. Nevins
J. Nugent
F. O'Glishen
M. Petersen
R. Ragan
G. Schmidt
R. Scholten
D. Scolamiero
N. Sears
J. Shillingford
E. Smally (7)
W. Stameris
M. Trageser
R. Weatherbee
R. White
R. Woodbury
W. Wrigley
Apollo Library (2)
MIT/IL Library (6)

External:

W. Rhine (NASA/MSC) (2)
NASA/RASPO (1)
AC Electronics (3)
Kollsman (2)
Raytheon (2)
Major H. Wheeler (AFSC/MIT) (1)

MSC: (25 + 1R)

National Aeronautics and Space Administration
Manned Spacecraft Center
Apollo Document Distribution Office (PA2)
Houston, Texas 77058

LRC: (2)

National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia
Attn: Mr. A. T. Mattson

GAEC: (3 + 1R)

Grumman Aircraft Engineering Corporation
Data Operations and Services, Plant 25
Bethpage, Long Island, New York
Attn: Mr. E. Stern

NAA: (18 + 1R)

North American Aviation, Inc.
Space and Information Systems Division
12214 Lakewood Boulevard
Downey, California
Attn: Apollo Data Requirements
Dept. 096-340, Bldg. 3, CA 99

NAA RASPO: (1)

NASA Resident Apollo Spacecraft Program Office
North American Aviation, Inc.
Space and Information Systems Division
Downey, California 90241

ACSP RASPO: (1)

National Aeronautics and Space Administration
Resident Apollo Spacecraft Program Officer
Dept. 32-31
AC Electronics Division of General Motors
Milwaukee 1, Wisconsin
Attn: Mr. W. Swingle

Defense Contract Administration (1)
Service Office, R
Raytheon Company
Hartwell Road
Bedford, Massachusetts 01730

Mr. S. Schwartz (1)
DOD, DCASD, Garden City
605 Stewart Avenue
Garden City, L. I., New York
Attn: Quality Assurance

Mr. D. F. Kohls (1)
AFPRO (CMRKKA)
AC Electronics Division of General Motors
Milwaukee 1, Wisconsin 53201