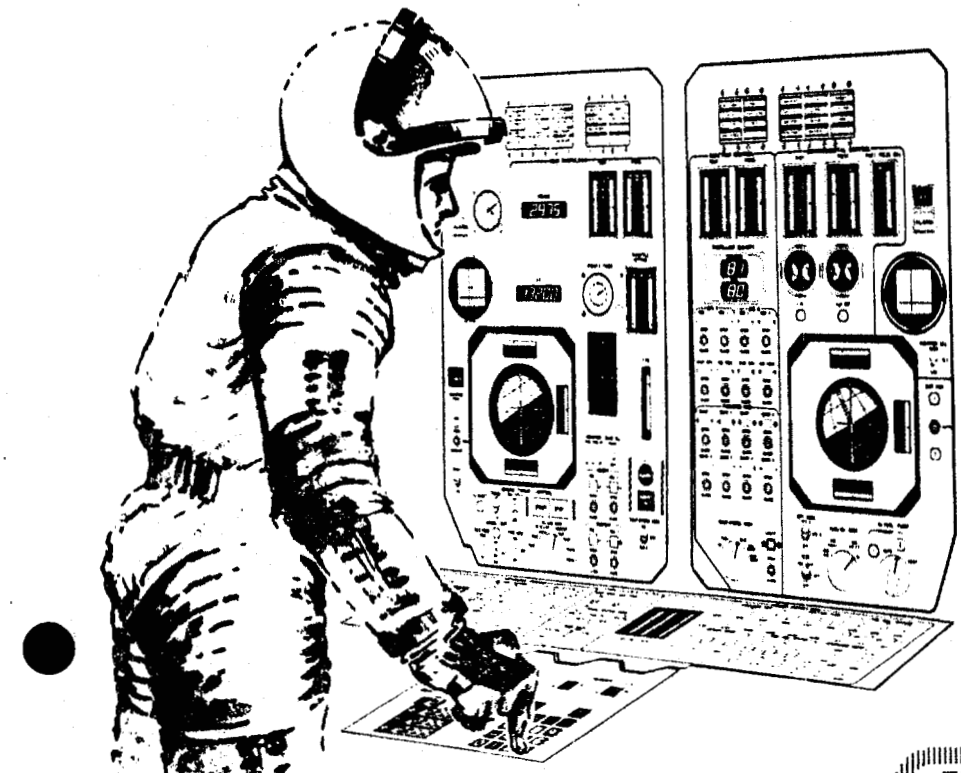
APOLLO GUIDANCE and NAVIGATION SYSTEM

STUDY GUIDE

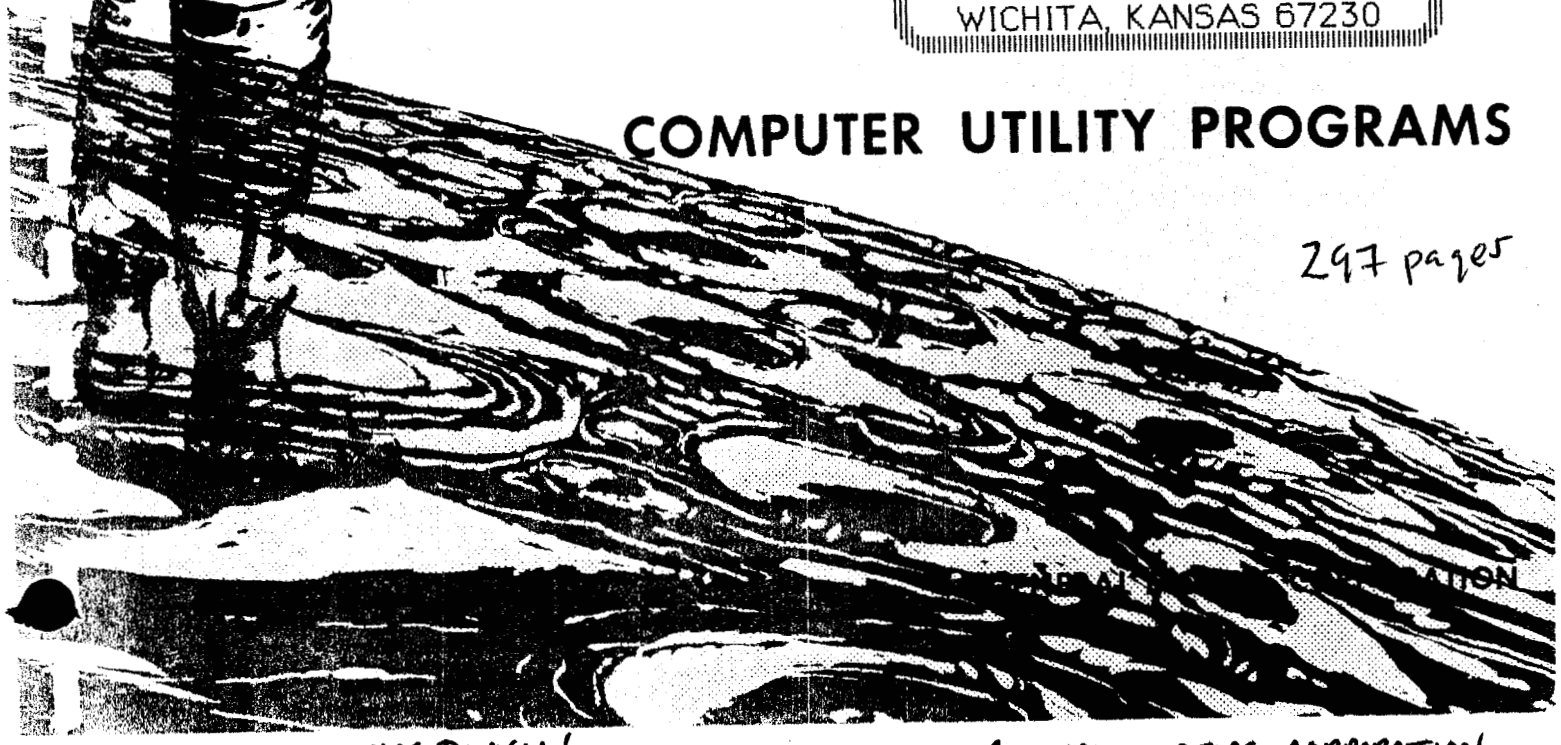
LM PRIMARY GUIDANCE NAVIGATION AND CONTROL SYSTEM

(15 January 1967)



COMPUTER UTILITY PROGRAMS

297 pages



AC ELECTRONICS DIVISION

GENERAL MOTORS CORPORATION

APOLLO

GUIDANCE AND NAVIGATION SYSTEM

LUNAR MODULE

STUDENT STUDY GUIDE

PRIMARY GUIDANCE, NAVIGATION AND CONTROL

SYSTEM COURSE 3100

COMPUTER UTILITY PROGRAMS C3100

PREPARED BY

AC ELECTRONICS

DIVISION OF GENERAL MOTORS

MILWAUKEE, WISCONSIN

15 JANUARY 1967

PREFACE

This student study guide has been prepared by AC Electronics in responee to:

Contract NAS 9-497

for

System Assembly and Test, Inertial

Measurement Unit, Coupling Display Unit

Power and Servo Assembly - Project APOLLO

This study guide contains training material and should be used for instruction purposes only.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

TABLE OF CONTENTS (cont)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (cont)

LIST OF TABLES

OBJECTIVES

The intent of this study guide is to give the student an understanding of the basic utility programming concepts associated with the LM computer. The programs described in this study guide are utility programs which, for the most part, are used in conjunction with all computer operations **and** include ***the*** basic executive routines, input/output routines and miscellaneous service routines along with basic programming techniques.

This study guide **is** organized in the sequence of instruction of the course and is divided into four major sections. Each of these sections is associated with the LM peculiar programs.

The objectives of this study guide are to provide the student with:

a. Course materials organized in the sequence **of** classroom presentation for self-study.

b. **A** familiarity of the overall utility programs associated with the LM computer.

REFERENCES

The following documents were used in preparation of this study guide:

Digital Development Report **9**	Operating Procedures for AGC Block II Self Check and Show-Banksum Revision 14 of Program SUNBURST Dated **9** September 1966
ND-1021042	**Apollo** Lunar Excursion Module Primary Guidance, Navigation, and Control System
Digital Development Memo # **254, Rev. B**	Block II Channel Assignments

SECTION I

GENERAL COMPUTER PROGRAMMING CONCEPTS

INTRODUCTION

This section presents the general programming concepts as used in the Apollo computer. Included is **(1)** a discussion of the development process of the computer program, *(2)* the real-time environment in which the computer operates, **(3)** time sharing of the computer among its processing functions, **(4)** the scheduling and implementation of program controlled processing functions, **(5)** the scheduling and implementation of time dependent program controlled processing functions, and *(6)* a general discussion of the computer's relationship with its hardware and human environment. Also included is (7) a brief discussion **of** the interpretive programming technique used in the computer.

1.1 THE DEVELOPMENT OF THE COMPUTER PROGRAM

The development **of** the total program capability for the Apollo computer, has been, and will continue to be an evolutionary process. This is true because changes in hardware design, mission, interface, etc., which must be reflected in the program of the computer and also the magnitude **of** the job of programming the computer.

In this evolutionary process, several groups of programs have been released. Each group of programs **has** superseded the previous group and has contained more of the required programs or routines. Also, each of the groups of programs have had many revisions to the programs contained within the group.

The first major series of programs which have been released for the **Block** II Command Module and LM computers (CMC and LGC), was called RETREAD. Basically, RETREAD converted Block I computer utility programs to Block II language and updated the various routines. RETREAD was followed by the **AURORA** (LM) and SUNDIAL (Command Module) series **of** programs. These two programs built on the foundation set by RETREAD and branched out in there respective directions to encompass programs associated only with the Command Module **or** LM. This study guide **is** based on the Sunburst Computer Program for LM.

Future programs will be based on the foundation developed by this series of programs. Each new group **of** programs will add to those programs contained in its immediate predecessor reflecting changes to previous programs deemed necessary by equipment changes in the PGNCS, mission, etc. Through this progression, the final computer program will be obtained and will afford the designers, programmers and users of the PGNCS a high degree of confidence in the computer programs.

1.2 THE COMPUTER'S REAL TIME ENVIRONMENT

The computer operates within the spacecraft and specifically within the **PGNCS.** Various systems on board the spacecraft interact with the computer to enable the required functions to be performed, thereby enabling the mission to be accomplished. All systems within the spacecraft operate on a real time basis, and therefore, the computer must also operate on a real time basis keeping cognizant **of** the happenings in this environment. Based on these happenings **or** conditions which exist at any particular time, the computer must determine if an action is required, and if it is, what must be done.

Inputs to the computer are derived from the **PGNCS** Inertial Subsystem, Optical Subsystem and Computer Subsystem. Also, inputs to the computer are derived from the Stabilization Control **System,** the Communication and Instrumentation System, etc. The inputs from these systems and subsystems may change at any time and the computer must be able to cope with them within a reasonable period of time.

Outputs from the computer are routed to the three subsystems of the **PGNCS** along with direct outputs to the Central Timing Equipment, Communications and Instrumentation System and to the 'Stabilization and Control System. The computer is also capable of controlling the outputs **of** the inertial subsystem of the **PGNCS** to the Stabilization and Control System. The outputs to these systems must correspond to the happenings or conditions which exist at any particular time in order that the computer effectively copes with the situation and fulfills **its** role in the spacecraft.

1.3 TIME SHARING THE COMPUTER HARDWARE

The computer operates in an environment in which many parameters and conditions change in a continuous manner. The computer, however, operates in a discrete, incremental manner, operating on only one item at any instant in time. Therefore, in order for the computer to process the many parameters and conditions, and perform its function in the PGNCS and spacecraft, the computer hardware must be time shared. The time sharing of the computer hardware is accomplished by assigning priorities to the various processing functions required **of** the computer. These priorities are used by the computer so that it processes the highest priority processing function required at any particular time.

1.4 IMPLEMENTING THE TIME SHARING OF THE COMPUTER

As previously stated, the basis for the time sharing **of** the computer is the priority of the processing functions requiring processing. The implementation of the time sharing is accomplished through one of three methods which are:

a. **A** pure hardware function. (Counter interrupts)

b. **A** hardware and program control function. (Program interrupts)

c. A pure program control function. (Program controlled processing)

Each **of** these three groups has **a** relative priority with respect to the other groups; also, within each **of** the groups there are a number of processing functions, each having a priority level relative to the other functions within the group. The majority of the processing performed **by** the computer falls into a pure program control processing category. In this category the computer hardware is controlled by the program stored in the computer's memory.

1.4.1 COUNTER INTERRUPTS. The processing performed by the computer which **is** accomplished under control of the computer hardware **is** referred to as a counter interrupt. This processing handles items such as **AV** pulses from the PIPA's, $\Delta\theta$ pulses from the CDU's, Δ time pulses from the computer timing circuitry, and control pulse outputs from the computer used to position the stable member of the IMU.

Whenever one of these pulse inputs is present, any other processing being performed by the computer is temporarily suspended or interrupted. Then the input pulse is processed under control of the computer hardware. After the input pulse is processed, control of the computer hardware is returned to the program controlled processing which was suspended. (See figure 1-1.) The processing of one of these input pulses requires approximately 12 microseconds.

Through the processing of the counter interrupts, the computer accumulates data such as velocity, IMU gimbal angles, radar angles and the number of computer pulse outputs developed to position the stable member.

1.4.2 PROGRAM INTERRUPTS. The processing performed by the computer which is controlled through both circuit and program controlled processing functions is referred to as program interrupt. This type of processing is performed whenever a particular condition exists, either internal or external to the computer. The conditions which cause a program interrupt are:

a. Timing of reaction control system.

b. Time to process a routine scheduled to be processed at a particular time.

c. Time to process a routine which performs computer input/output functions.

d. An input from the DSKY or MARK pushbuttons.

e. Time to load a new DOWNLINK telemetry word.

f. Time to process an UPLINK word.

g. Time to process an attitude or tranelation controller input.

h. Time to process a radar input.

The processing of one of these conditions is initiated whenever the condition exists. The initiation of the processing is accomplished by a circuit function which forces control of the computer hardware to a particular program controlled processing routine. The program controlled processing function being processed at the time when a program interrupt occurs is suspended and control of the computer is forced to the routine corresponding to the program interrupt condition which exists. (See figure 1-2.) The program interrupt routine then processes whatever is required, depending on the program interrupt condition present. After completing the required processing for the program interrupt, control of the computer hardware is returned to the suspended program.

1.4.3 PROGRAM CONTROLLED PROCESSING, Most of the time, the computer's hardware is controlled by the program stored in its memory. Of the many routines or processing functions that the computer is capable of processing, some means must be employed to enable the computer to process the routines required at any one time, and to process the most important required routine first.

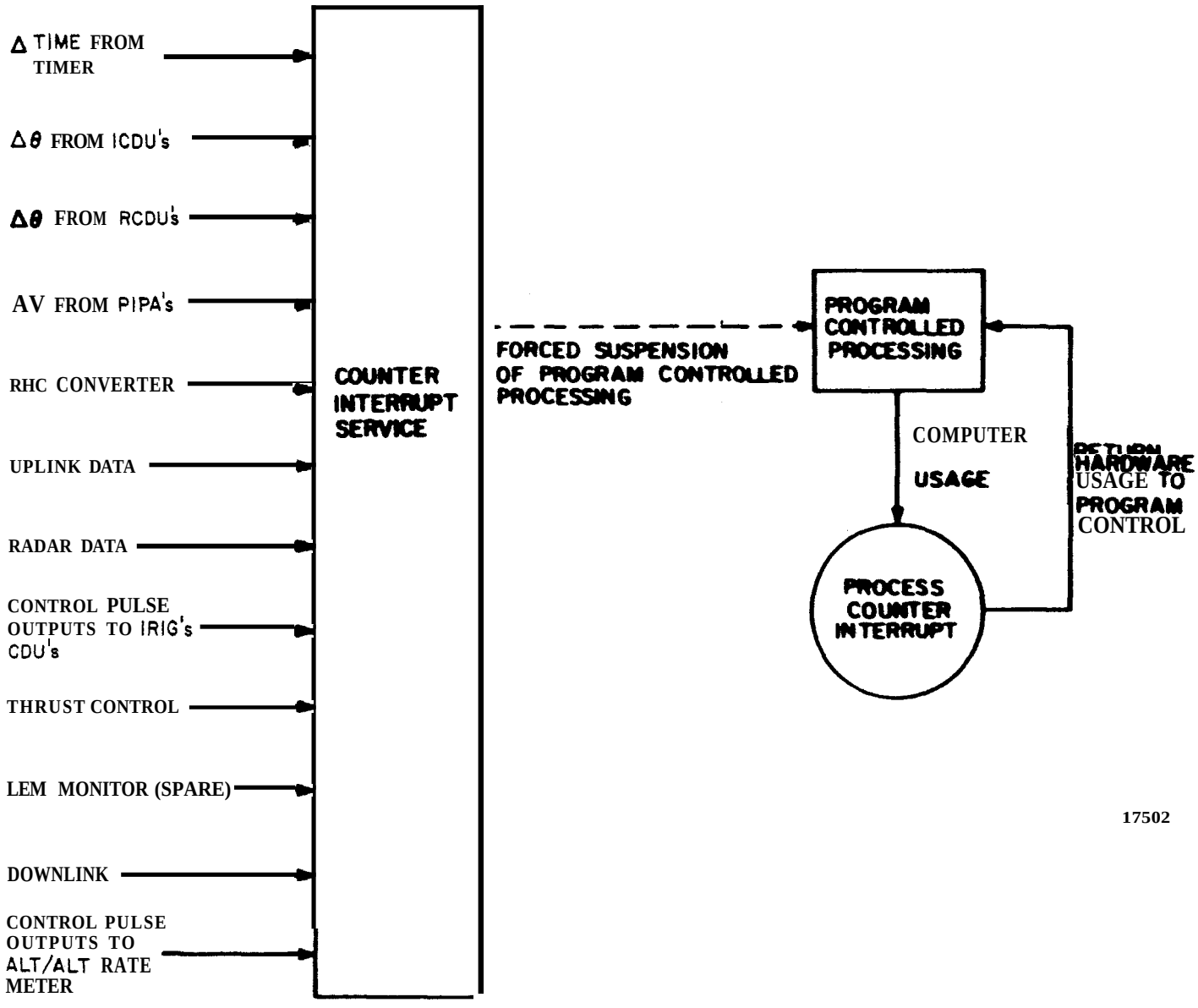


Figure 1-1. Counter Interrupt Processing

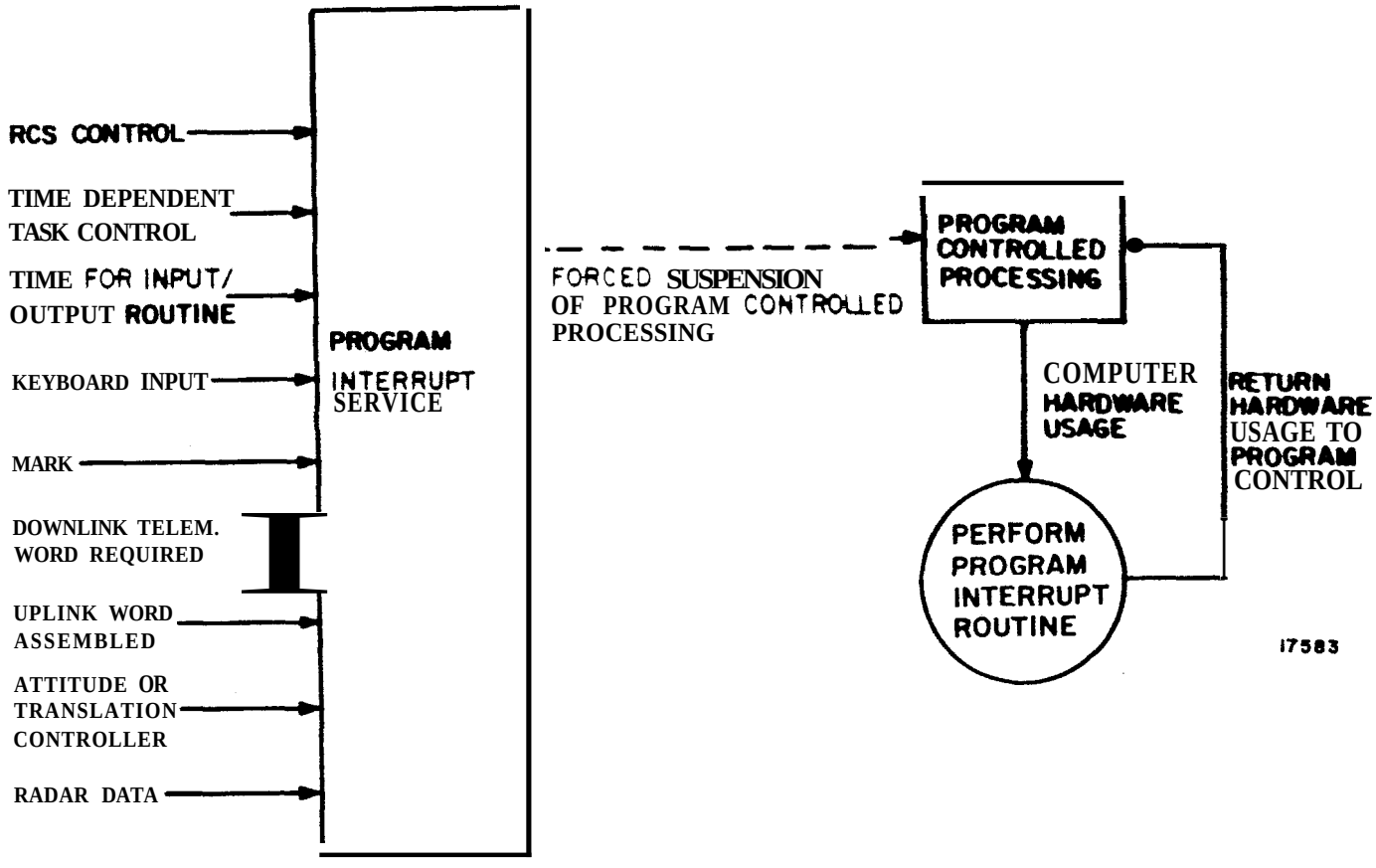


Figure 1–2. Program Interrupt Processing

In order for a routine or program controlled processing function to be performed, it must first be scheduled. The scheduling of a particular routine or processing function is a function of another routine or processing function. The scheduling also can be initiated through the **DSKY.** At the present time, the computer is capable of having up to seven routines, usually referred to as "jobs", scheduled to be done at one time. The job which is processed out of the possible seven scheduled jobs is determined by the priority numbers assigned to the **jobs.** If a **job** is scheduled having a priority higher than the job being processed, the computer suspends the processing of the lower priority **job** and processes the higher priority job. When the higher priority job is completed, the control of the complter hardware returns to the lower priority job at the point where it was suspended. Using the scheduling of jobs and the priority assigned to the various jobs, the most important program controlled processing function is performed at any time.

1.5 RELATIVE PRIORITIES OF THE **TYPES** OF **PROCESSING**

As previously stated, each of the three types of processing (counter interrupt, program interrupt and program controlled processing) have relative priorities. Of the three types, the counter interrupt processing is the highest priority processing function. A counter interrupt input, which requires processing, causes the processing of either a program controlled function **or** program interrupt to be suspended. After processing the counter interrupt, control is returned to the processing which was suspended. (See figure **1-3.)**

Program interrupts are the next highest priority type of processing. This type of processing causes the suspension of any program controlled processing. A program interrupt cannot interrupt or suspend the processing of a counter interrupt or the processing of another program interrupt. However, through program action, an inhibit can be set so that the program interrupt processing cannot interrupt the program controlled processing.

The program controlled processing is the lowest priority type of processing. Any counter interrupt or program interrupt processing causes the program controlled processing to be suspended. The exception to this, as stated above, is when the honoring of a program interrupt is inhibited through program action. The program interrupts would be inhibited if some fairly critical function was being performed through program controlled processing.

1.6 SCHEDULING AND EXECUTION OF PROGRAM **CONTROLLED PROCESSING** ON **THE** BASIS **OF** PROGRAM PRIORITY

1.6.1 INTRODUCTION. The processing of program controlled processing functions, as previously stated, is controlled on the basis of the priority assigned to the processing functions. However, a program controlled processing function cannot be processed unless it has been scheduled. In the following paragraphs, the scheduling and control **of** program controlled processing functions is discussed.

1.6.2 TERMINOLOGY. Program controlled processing function is a term which has been used up to this point in the study guide to refer to program routines, subroutines, etc., which control the processing of various functions. Any one of these categories can be scheduled to be processed on a priority basis. These programs, routines, .etc. , which require scheduling in order to be processed are referred to as JOBS.

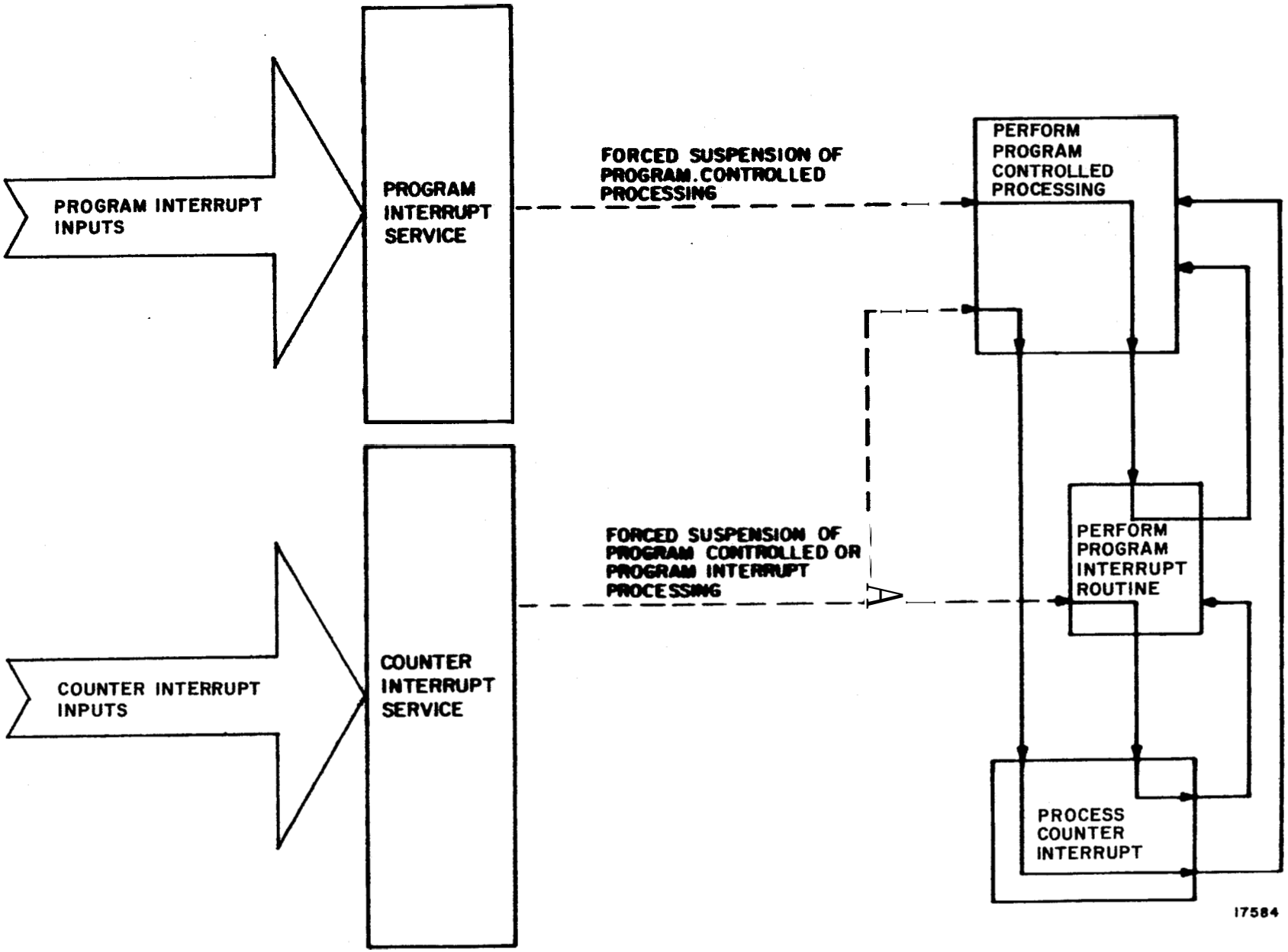


Figure 1-3. Counter and Program Interrupt Processing

1.6.3 SCHEDULING. Scheduling **of** a job must be performed under control of another job or program interrupt routine. Whichever type of processing function schedules a job, the scheduling process is the same.

Whenever a job or routine wishes to schedule a job, it uses a routine of the computer program called the EXECUTIVE. The scheduling job or routine, referred to as the calling program, must supply *the* Executive routine with the priority number to be assigned and the starting address of the job being scheduled. The Executive routine uses these two quantities to schedule the job. The priority number is then used to control when'the **job is** processed and the starting address **is** used to route control of the computer to the starting point of the job. The actual scheduling of the job is accomplished by the Executive routine inserting the priority number and starting address into a position on the core set list which the Executive maintains. At the present time, the Executive's core set list provides for scheduling up to seven jobs at any one time. The core set list **is** used by the Executive routine to control which of the scheduled jobs **is** processed based on their priority numbers. The highest priority scheduled job is processed at any one time.

1.6.4 EXECUTION **CONTROL** Whenever a job **is** scheduled the priority number assigned to the job can be equal to, higher than or lower than the priority number of the job presently being processed. If the newly scheduled job's priority is equal to **or** lower than the priority of the job presently being processed, the processing **of** this job continues. When this job has been completed, *it* is removed from the core set list. At this time, the core set list **is** scanned to find the scheduled job with the highest priority. After finding the highest priority job, processing of the job **is** initiated if no processing of the job has been accomplished, or **is** resumed if a protion of the job had already been processed. (See figure 1-4,)

However, if a job **is** scheduled which has a priority higher than **the** job presently being processed, processing of the newly scheduled higher priority job must begin. The job presently being processed is suspended and left uncompleted when the processing of the higher priority job initiated. After all of the jobs of higher priority than the one just suspended are completed, processing of this lower priority job is resumed. Processing is resumedat the point in the job where processing was suspended. No reprocessing is required when the processing of a job is resumed.

Most jobs, **in** the course **of** being processed, must wait for information to be loaded into the complter or for an action external to the computer to occur. In general, the job cannot continue until the required information **is** loaded ***or*** the action has occurred. When instances as these occur during the processing **of** a job, it is desirable to deactivate the job while it **is** waiting rather than having **a** reiterative waiting loop as part of the job. By deactivating a job, the processing of lower priority jobs can be accomplished. The act of deactivating a job does not remove the job from the core set list and therefore remains scheduled. This process of deactivating a job is termed "putting a job to sleep".

When the required information has been loaded or the action has occurred, the job is re-activated. The job can then use the information that was loaded or continue on since the required action has occurred. The process of reactivating a job **is** termed "waking up a job". It should be noted that a job can put itself to sleep but another job or routine is required to wake it up. **(This is** the same relationship which exists between you and your alarm clock.)

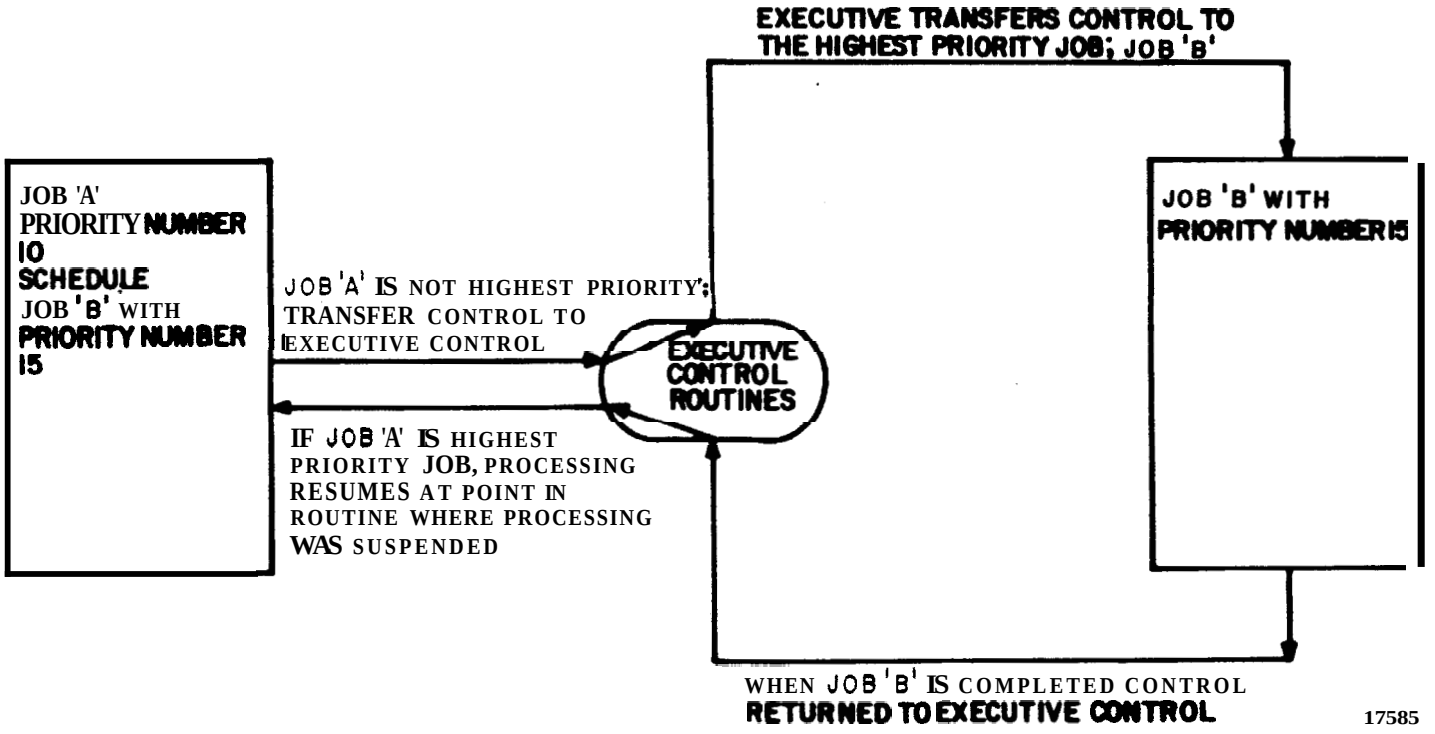


Figure 1-4. Control of Program Controlled Processing on Basis of Program Priority Numbers

1.6.5 CORE SET AREAS AND VAC AREAS. The capability of the computer to suspend and deactivate jobs and resume processing at the point where they were suspended or deactivated is made possible by the storage provided by the CORE SET AREAS and VAC AREAS (Vector Accumulator Area). These two areas provide storage for the information of a job while it is being processed. A CORE SET AREA or both a CORE SET AREA and a VAC AREA are reserved for use for every scheduled job. An area reserved for a particular scheduled job cannot be used by another job. Therefore, with all of the information being processed by a job stored in the reserved CORE SET and VAC areas, a job can be suspended or deactivated and the processing can be resumed at the point in the job where it was suspended or deactivated.

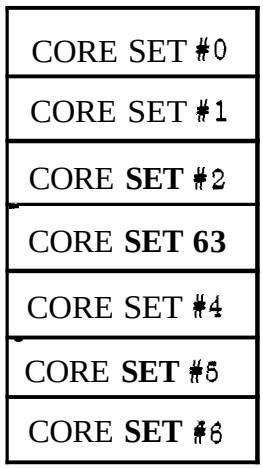
1.6.5.1 Core Set Areas. The CORE SET AREAS of the computer program are an integral part of the core set list. Each of the seven **CORE** SET **AREAS** consists of twelve sequential memory registers. One of the twelve is used for storing the priority number and the **VAC** address associated with the job. The priority in this register signifies that the CORE SET **AREA** is reserved. **A** memory register used for storing the starting address of the job, when the job **is** originally scheduled, will store the resumption address of a **job** which has been suspended or deactivated. The other memory registers are used to store information concerned with the **job** using the **CORE SET AREA.** ***(See*** figure 1-5.)

1.6.5.2 VAC Areas. Some jobs or programs of the computer require more storage capability than is provided by the core set area. One type of job which requires more storage are those jobs involved with the processing of vector quantities. Double or triple precision, three component representation is used for most vector quantities in the computer. (Double and triple precision representation uses two or three computer words to represent one quantity.) Therefore, vector quantities require up to nine memory registers for storage. Besides the vector quantity, other information associated with a particular job must be stored. In order to meet the storage capacity required for these type of jobs, an additional block **of** memory registers can be reserved for a job. The computer program now provides for five blocks, with each block containing **44** memory registers. These blocks of memory registers are called VECTOR ACCUMULATOR AREAS or, in short, **VAC** AREAS. If a VAC AREA is required by a job, one of the five is reserved for the **job** along with one of the CORE SET AREAS. (See figure **1-6.**)

Again, since a particular VAC AREA is reserved for use by a particular job, and since information being processed by the job is stored in the VAC AREA and **CORE SET AREA,** the processing of a job **can** be resumed at the point in the processing where it was suspended or deactivated. It should also be noted that with the seven **CORE SET** AREAS and the five **VAC AREAS,** sufficient scheduling and storage capacity is provided to handle the processing loads imposed on the computer.

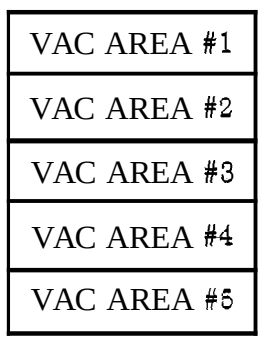
1.7 SCHEDULING AND EXECUTION OF TIME DEPENDENT PROCESSING

1.7.1 INTRODUCTION. Some processing functions and corresponding output functions of the computer require a rather stringent consideration of time. In order to accommodate this consideration, **a** job or routine which controls a processing or output function must be initiated at a specific time. This could be accomplished by having incorporated,



- EACH CORE SET AREA CONSISTS OF TWELVE (DECIMAL) MEMORY REGISTERS, SEVEN OF WHICH ARE USED FOR STORAGE OF QUANTITIES PERTAINING TO THE SCHEDULED JOB USING THE CORE SET AREA.
- THE REMAINING FIVE MEMORY REGISTERS ARE USED TO STORE INFORMATION ABOUT THE JOB; PRIORITY, STARTING ADDRESS, ETC.

Figure 1-5. Core Set Areas of the Computer Program (Core Set List)



- A VAC AREA (VECTOR ACCUMULATOR) PROVIDES 44 (DECIMAL) MEMORY REGISTERS FOR STORAGE OR INFORMATION PERTAINING TO THE JOB FOR WHICH IT WAS RESERVED.
- JOB, ESPECIALLY THOSE INVOLVING VECTOR QUANTITIES REQUIRE MORE STORAGE CAPACITY THAN IS AFFORDED BY THE CORE SET AREAS
- IF A VAC AREA IS REQUIRED FOR A JOB, BOTH A VAC AREA AND A CORE SET AREA MUST BE RESERVED FOR THE JOB.

Figure 1-6. VAC **Areas of** the Computer **Program**

as part of a job, a waiting loop which would continuously look at the computer's real time reference, TIME 1 and TIME 2 counters. If this method was used, a considerable amount of computer time would be wasted. Another method of time scheduling and execution of functions as used in the computer is discussed in general terms in this portion of the study guide.

1.7.2 IMPLEMENTING TIME DEPENDENT FUNCTIONS. The computer, in implementing the time dependent initiation of various processing functions, utilizes the TIME 3 counter. This counter is incremented at 10 m. *s.* intervals through the counter interrupt priority control circuitry of the computer. By setting the TIME 3 counter to overflow at the time a specific function is to be performed, the overflow condition of the TIME 3 counter indicates when a function is to be performed. When the TIME 3 counter is in an overflow state, the T3RUPT program interrupt routine is initiated. This routine uses the starting address stored by the scheduling job to transfer control of the computer's processing to the routine which controls the processing or output function which is required at this specific time.

Implementing the initiation of a time dependent function in this manner allows the scheduling routine to set up the TIME 3 counter and the address of the routine to be initiated at a specific time. Then it need not be concerned with the timing of the initiation of the routine. The scheduling job can continue to be processed or be put to sleep, thereby conserving time.

1.7.3 SCHEDULING OF TIME DEPENDENT FUNCTIONS. A job scheduled through the Executive core set list or a program interrupt routine can schedule a time dependent routine. The scheduling process is performed by a routine of the computer program called the WAITLIST. This routine maintains a scheduling list of time dependent routines, referred to as TASKS, to be done. For each of the nine possible entries at any one time of this waiting list, the WAITLIST routine requires two quantities, the time till the function should occur in increments of 10 m. *s.* and the starting address of the routine or TASK that is to be initiated at the specified time. The scheduling routine must provide the Waitlist routine with these two quantities.

1.7.4 EXECUTION OF TIME DEPENDENT FUNCTIONS. The control over the execution of a task is provided by the T3RUPT routine whenever the TIME 3 counter overflows. The T3RUPT routine uses the stored starting address to route control of the computer to process the task. After the task has been completed, it returns control to the T3RUPT routine which in turn, returns control to the job which was interrupted.

In summary, a job or task can schedule a task to be done by providing the Waitlist routine with the time till the task is to be performed and the starting address of the task. Then, through counter interrupt processing, the TIME 3 counter, which was set to overflow minus the time till the desired function is to be executed, is incremented until an overflow condition exists in the counter. Whenever overflow exists in the TIME 3 counter, the desired amount of time has elapsed and the TBRUPT routine is initiated which uses the starting address of the task to transfer the control of computer processing to the task. After the task has been completed, control is returned to the T3RUPT routine which returns control of computer processing to the job that was being processed when the TIME 3 counter overflowed. By implementing the execution of tasks in this manner, the computer's jobs do not have to concern themselves with the task it has scheduled. This saves considerable computer time.

1.8 LGC INPUT AND OUTPUT CHANNEL INTERFACE

In addition to the counter interrupt and the program interrupts previously described, the LGC has a number of other inputs derived from its interfacing hardware. These inputs are a result of the functioning of the hardware or an action by the operator of the spacecraft. The counter interrupts in most cases enable the LGC to process inputs representative of data parameters such as changes in velocity. The program interrupt inputs to the LGC are used to initiate processing of functions which must be processed a relatively short time after a particular function is present. The other inputs to the LGC, in general, enable the LGC to be cognizant of "conditions" which exist in its environment. These inputs are routed to, and are available to the LGC's programs through the LGC's input registers.

The outputs of the LGC fall in one of the following categories: (1) data., (2) control, (3) condition indications. Some of these outputs are controllable through the LGC's program while others are present as a function of the LGC circuitry. All of the outputs which are controlled by the LGC's programs are developed through the LGC's output registers. The bit breakdown per channel is shown in table 1-1.

1.8.1 CHANNEL 01. This channel is used as the L register of the central processor.

1.8.2 CHANNEL 02. This channel is used as the Q register of the central processor.

1.8.3 **CHANNEL 03 HIGH-ORDER SCALER.** This channel furnishes a 14-bit positive number whose least significant bit has a weight of 5. 12 seconds. The maximum content of the register is 23.3 hours.

1.8.4 **CHANNEL 04 LOW-ORDER** SCALER. This channel furnishes a 14-bit positive number whose least significant bit has as a weight of 1/3200 second. The maximum content of the register is 5.12 seconds.

1.8.5 OUTPUT CHANNEL 05. This channel has eight bit positions and is associated with the reaction control system jets. The channel outputs are used for translational and rotational motion of the LM. The RCS jet commands from the channel are fed to the preamplifiers of the jet drivers in the CES. The driver amplifier outputs are then fed to the RC subsystem to provide the required control.

The first number contained in the bit positions indicates which of the 16 thrusters is controlled by that bit. Four clusters is used. The letter indicates the direction of thrust such as U for up and D for down.

1.8.6 **OUTPUT CHANNEL 06.** This channel has eight bit positions and is also associated with the reaction control system jets. A logic one in any of the bit positions will cause the appropriate reaction control jet to be fired.

1.8.7 OUTPUT CHANNEL 07. This channel is the F EXT register. It is associated with the selection of word locations in fixed memory as shown in figure 1-7. This channel has three bit positions.

1.8.8 OUTPUT CHANNEL 10. The information contained in this channel is routed the DSKY's. The different configurations light various displays on the DSKY's. In Section IV, it will be seen that there is a basic difference between the information in bit positions 1 through 11 and the information in bits 12 through 15.

Table 1-1. Channel Assignments LM

CHANNEL	NAME	BIT 15	BIT 14	BIT 13	BIT 12	BIT 11	BIT 10	BIT 9	BIT 8	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	CHANNEL
1	L	CP REGISTER L, BITS 16-1															1
2	Q	CP REGISTER Q, BITS 16-1															2
3	SCALER2	HIGH - ORDER SCALER CHANNEL, BITS 14-1															3
4	SCALER1	LOW - ORDER SCALER CHANNEL, BITS 14-1															4
5	PITCH YAW								#14 RCS ID	#13 RCS 1U	#10 RCS 2D	#9 RCS 2U	#6 RCS 3D	#5 RCS 3U	#2 RCS 4D	#1 RCS 4U	5
6	ROLL								#16 RCS 1S	#4 RCS 4S	#8 RCS 3S	#12 RCS 2S	#11 RCS 2F	#15 RCS 1F	#3 RCS 4F	#7 RCS 3F	6
7	FEXT									FE7	FE6	FE5					7
10	DSKY R & C	RELAY ADRS 4	RELAY ADRS 3	RELAY ADRS 2	RELAY ADRS 1	RELAY BIT 11	RELAY BIT 10	RELAY BIT 9	RELAY BIT 8	RELAY BIT 7	RELAY BIT 6	RELAY BIT 5	RELAY BIT 4	RELAY BIT 3	RELAY BIT 2	RELAY BIT 1	10
11	DSKY A		ENGINE OFF	ENGINE ON			CAUTION RESET	TEST CONNEC OUTBIT		OP ERROR LAMP	VN FLASH	KEY RLSE LAMP	TEMP CAUTION LAMP	UPLINK ACTY LAMP	COMP ACTY LAMP	ISS WARNING	11
12	GN&C	ISS TURNON DELAY COMPLETE	RR ENAB AUTO TRK	LR POS COMM	-ROLL GIMBAL TRIM	+ROLL GIMBAL TRIM	-PITCH GIMBAL TRIM	+PITCH GIMBAL TRIM	DISPLAY INERTIAL DATA		ENABLE IMU ERR CTR	ZERO IMU CDU	COARS ALIGN ENABLE	HORIZ VEL LO SCALE	ENABLE RR ERR CTR	ZERO RRCDU	12
13	LGC	ENABLE T6RUPT	RESET TRAP 32	RESET TRAP 31B	RESET TRAP 31A	ENABLE STANDBY	TEST ALARMS	RHC READ	ENABLE RHC CTR	DNLNK WD ORD	BLOCK INLINK	INHIBIT UPLINK	RADAR ACTY	RADAR a	RADAR b	RADAR c	13
14	IMU	DRIVE CDUX	DRIVE CDUY	DRIVE CDUZ	DRIVE CDU T	DRIVE CDU S	GYRO ACTY	GYRO MINUS	GYRO a	GYRO b	GYRO ENABLE		THRUST DRIVE	ALT METER	ALT RATE	OUTLINK ACTY	14
15	MAIN DSKY											KEY 5M	KEY 4M	KEY 3M	KEY 2M	KEY 1M	15
16	NAV DSKY									DESCENT -	DESCENT+	MARK REJECT	MARK Y	MARK X			16
30	GN&C *	TEMP IN LIM	ISS TURNON REQUEST	IMU FAIL	ICDU FAIL	IMU CAGE	G.N CONT OF S/C	IMU OPER		RRCDU FAIL	DISPLAY INERTIAL DATA	AUTO THROTL	ABORT STAGE	ENGINE ARMED	STAGE VERIFY	ABORT	30
31	TRANS & * ROT	ATT CONT OUT OF DETENT	AUTO STABILI-ZATION	ATT HOLD	-Z TRANS	+Z TRANS	-Y TRANS	+Y TRANS	-X TRANS	+X TRANS	-AZ (LPD) -RMI	+AZ (LPD) +RMI	-YMI	+YMI	-EL (LPD) -PMI	+EL (LPD) +PMI	31
32	IMPULSE *						ROLL GIMBOFF	PITCH GIMBOFF	T 10-11 FAIL	T 9-12 FAIL	T 13-15 FAIL	T 14-16 FAIL	T 6-7 FAIL	T 1-3 FAIL	T 5-8 FAIL	T 2-4 FAIL	32
33	OPTICS * LGC	OSC ALARM	WARNING	PIPA FAIL	DNLNK TOO FAST	UPLNK TOO FAST	BLOCK UPLINK	LR RANG LO SCALE	LR VEL DAT GOOD	LR POS 2	LR POS 1	LR DATA GOOD	RR DATA GOOD	RR RANGE LO SCALE	RR PWR ON AUTO		33
34	DNLNK1	FIRST OF TWO WORDS															34
35	DNLNK2	SECOND OF TWO WORDS															35
		15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	

* INVERTED LOGIC USED

1.8.9 OUTPUT CHANNEL 11. The information contained in this channel is also routed to the **DSKY.** The function of the information in the various bit positions is detailed in Section **1-9-3.**

1.8.10 OUTPUT **CHANNEL 12.** This channel consists of **15** bit positions, **14** of which are presently used. The outputs are dc signals sent to the spacecraft systems and the **PGNCS. Bits 13** and **14** are sent through the DSKY's but light no indicator.

1.8.11 OUTPUT CHANNEL 13. The first four bite **of** this channel are associated with the landing and rendezvous radar. The content of bits positions **1** through **3** defines which data **is** to be supplied **by** the radar **and** can select one of the six inputs as shown in figure **1-8.**

When a one has been entered into bit position **4** together with the necessary selection bits in bit positions **1** through **3,** the LGC starts to transmit one **of** the six control signals. While the control signal **is** being transmitted, **a** sync pulse **is** also transmitted. When the radar receives the **sync** pulses, it sends data pulses to the **LGC.**

FE7	FE6	FE5	HIGH BANKS
0	X	X	30-37
1	0	0	40-43
1	0	1	EMPTY
1	1	0	EMPTY
1	1	1	EMPTY

Figure **1-7.** Channel **07 Fix** Extension Bits

a	b	c	Function
0	0	0	-
0	0	1	**RR** Range
0	0	0	**RR** range rate
0	1	1	-
1	0	0	**LR X** Velocity
1	0	1	**LR Y** Velocity
1	1	0	**LR Z** Velocity
1	1	1	**LR** Range

Figure **1-8.** Radar Selection

Bit positions 12 through 14 have been covered under program interrupt priority control.

1.8.12 OUTPUT CHANNEL 14. The altitude meter control is controlled by bit positions 2 and 3 of output channel 14.

Bit positions 11 through 15 are associated with the CDU drive control. The CDU drive control enters the following dc signals into the counter priority control to request the execution of a DINC instruction: X IMU CDU, Y IMU CDU, Z IMU CDU, S RR CDU and T RR CDU.

Signal X IMU CDU is generated when bit position 15 contains a logic one, signal Y IMU CDU is generated when bit position 14 contains a logic one, signal Z IMU CDU when bit position 13 contains a logic one, signal T RR CDU when bit position 12 contains a logic one, and signal S RR CDU when bit position 11 contains a logic one. More than one of these signals can be generated simultaneously.

The gyro drive control selects a gyro to be torqued positively or negatively as shown in figure 1-9 and then applies a 3200 cps to the appropriate gyro to aocomplish this function. The appropriate signal is determined by the bit configuration of bits 7 through 9 of output channel 14. If bit positions 6 and 10 are a logic one, a 3200 cps pulse train is routed to the gyro electronics specified by bit positions 7 through 9, and a dc signal is entered into the counter priority control which commands the sequence generator to perform a DINC instruction.

a	b	Gyro
0	0	-
0	1	X
1	0	Y
1	1	Z

Figure 1-9. Gyro Selection

1.8.13 INPUT CHANNEL 15. This channel consists of five bit positions. Whenever a key on the DSKY is pressed, a unique five bit code is entered into this channel. The RUPT 5 interrupt routine is also developed whenever a key is depressed.

1.8.14 INPUT CHANNEL 16. This channel consists of seven bit positions. If the MARK pushbutton has been depressed, a logic one is entered into bit position 3 or 4. This would cause a KEYRUPT 2 (RUPT 6) interrupt routine. If the MARK REJECT pushbutton has been depressed, a logic one is entered into bit position 5 of this channel. This will also cause a KEYRUPT 2 interrupt routine to be performed. Bits 6 and 7 receive discretes from the crew station commanding an increase or decrease in the rate of descent.

1.8.15 INPUT CHANNELS 17 THROUGH 27. Spares.

1.8.16 INPUT CHANNEL **30.** This channel consists of 15 bit positions and uses inverted logic. These positions are utilized as follows:

a. Bit Position **1** (ABORT) This Signal informs the LGC that an abort using the descent engine has been commanded and is initiated by the crew from the control panel.

b. Bit Position **2** (STAGE VERIFY) This signal informs the LGC that staging has occurred and originates with the explosive devices.

c. Bit Position **3** (ENGINE ARMED) This signal informs the LGC that the crew has armed either the ascent or descent engine and is initiated from the control panel.

d. Bit Position **4** (ABORT STAGE) This signal informs the LGC that an abort which requires use of ascent engine has been commanded.

e. Bit Positions **5** (AUTO THROTTLE) Informs LGC that it is in command of descent engine throttle.

f. Bit Position **6** (DISPLAY INERTIAL DATA) This signal informs the LGC to supply forward and lateral velocity to the display panel.

g. Bit Position 7 (RR CDU FAIL) This input is generated when a failure has occurred in one of the radar CDU channels.

h. Bit Position **9** (IMU OPERATE) A binary one in this bit position indicates that the IMU is turned on and is operating with no malfunctions.

i. Bit Position **10** (G&N CONTROL OF S/C) This signal informs LGC that PGNCS (as opposed to abort guidance) is in control of the LM.

j. Bit Position **11** (IMU CAGE) This input indicates that the IMU cage condition exists in the **ISS.**

k. Bit Position **12** (IMU CDU FAIL) This input indicates that a failure has occurred in one of the inertial CDU channels.

1. Bit Position **13** (IMU **FAIL)** This input indicates that a malfunction has occurred in the IMU stab loops.

m. Bit Position **14 (ISS** TURN-ON REQUEST) This input indicates when the ISS has been turned on or commanded to be turned on.

n. Bit Position **15** (TEMP IN LIMITS) This input indicates when the stable member temperature has not exceeded its design limits.

1.8.17 INPUT CHANNEL **31.** This channel consists of **15** bit positions and uses inverted logic.

a. Bit Positions 1 and 2 (±PMI) These signals indicate ±pitch manual input commands from the attitude controller. Bit positions are utilized for landing point designator elevation changes.

b. Bit Positions 3 and 4 (±YMI) These signals indicate ±yaw manual input commands from the attitude controller.

c. Bit Positions 5 and 6 (±RMI) These signals indicate roll manual input commands from the attitude controller. These bit positions are utilized for landing point designator azimuth changes.

d. Bit Positions 7 through 12 (±X, Y, Z TRANS) These signals from the translation controller command LM translation by ON/OFF firing of the RCS jets under LGC control.

e. Bit Position 13 (ATTITUDE HOLD) This signal indicates the SCS is operating in the attitude hold mode.

f. Bit Position 14 (AUTO STABILIZATION) This signal informs the LGC that the SCS is operating in the automatic mode.

g. Bit Position 15 (ATTITUDE CONTROLLER OUT OF DETENT) This signal informs the LGC that the attitude controller is not in the neutral position.

1.8.18 INPUT CHANNEL 32. This channel consists of 15 bit positions and uses inverted logic.

a. Bit Positions 1 through8 (THRUSTER FAIL) These eight signals inform the LGC of thruster pair shutoff so that the LGC immediately ceases to command these jets on and compensates for their loss.

b. Bit Positions 9 and 10 (PITCH OR ROLL GIMBAL OFF) This signal informs the LGC that the descent engine pitch or roll gimbal drive amplifier has been shut off by automatic failure detection circuitry.

1.8.19 INPUT CHANNEL 33. This channel consists of 15 bit positions and uses inverted logic.

a. Bit Position 2 (RR POWER ON/AUTO) This signal indicates that the RR power is on and the mode switch is in the automatic (computer) position.

b. Bit Position 3 (RR RANGE LOW SCALE) This signal is implemented automatically by the rendezvous radar at a range of approximately 50 nautical miles and indicates that the RR scale factor is on low scale.

c. Bit Positions 4 and 5 (RR AND LR DATA GOOD) These signals indicate that the RR and LR range trackers have locked on.

d. Bit Positions 6 and 7 (LR POSITIONS 1 AND 2) These signals indicate the position of the landing radar antenna.

e. Bit Position **8** (LANDING VEL DATA GOOD) This signal indicates that the LR velocity trackers have locked on.

f. Bit Position **9** (LR RANGE LOW SCALE) This signal is implemented automatically by the landing radar at approximately **2500** feet range and supplied to the LGC to indicate a change in scale factor.

g. Bit Position **10** (BLOCK UPLINK SWITCH) This signal is generated by a switch closure to inhibit reception of data via uplink. (Uplink capability not presently **on** LM).

h. Bit Positions **11** and **12** (UPLINK AND DOWNLINK TOO FAST) These signals are generated by the telemetry system indicating PGNCS telemetry rate is too high.

i. Bit Position **13** (PIPA FAIL) This signal by the computer when an accelerometer loop failure occurs.

j. Bit Position **14** (COMPUTER WARNING) This signal is generated by the computer if one of the following items occur:

1) Restart

2) Counter fail

3) Voltage fail in standby mode

4) Alarm test

5) Scaler double alarm

k. Bit Position **15** (OSC ALARM) This signal occurs if the computer oscillator stops.

1.8.20 OUTPUT CHANNELS **34** AND **35.** These channels provide **16** bit words including a parity bit for downlink telemetry transmission.

1.9 COMPUTER/DSKY - HARDWARE/ASTRONAUT RELATIONSHIP

The DSKY serves an important interface function in the PGNCS. Through the DSKY the computer controls the mode of operation of the ISS and radar, keeps the astronaut cognizant of the operational condition of certain portions of the PGNCS equipment, displays pertinent information to the astronaut and makes requests of the astronaut to perform various actions. The astronaut, in turn, is capable of loading data into the computer, requesting the display of data, commanding system modes of operation and commanding other miscellaneous functions to be performed by the computer.

1.9.1 KEYBOARD. The keyboard consists of ten numerical keys (pushbuttons) labeled *0* through 9, two **sign** keys (+ or –) and seven instruction keys: **VERB,** NOUN, CLR (clear), **STBY** (standby), KEY REL (key release), ENTR (enter) and RSET (reset), Table **1-2** lists these keys (pushbuttons) and their functions.

Table 1-2. **DSKY** Pushbuttons

Pushbutton	Function
0 through **9** pushbuttons	Enter numerical data, noun codes and verb codes into the computer.
+ and − pushbuttons	Inform the computer that the following numerical data is decimal and indicate the sign of the data.
NOUN pushbutton	Conditions the computer to interpret the next two numerical characters as a noun code and causes the noun display to be blanked.
CLEAR pushbutton	Clears data contained in the data displays. Depressing this key clears the data display currently being used. Successive depressions clear the other two data displays.
STBY pushbutton	Commands the computer to the standby mode when depressed the first time. **An** additional depression commands the computer to resume regular operation.
KEY REL pushbutton	Releases the DSKY displays initiated **by** keyboard action so that information supplied by the computer program may be displayed.
ENTR pushbutton	Informs the computer that the assembled data is complete and that the requested function is to be executed
RSET pushbutton	Extinguishes the lamps that are controlled by the computer.
VERB pushbutton	Conditions the computer to interpret the next two numerical characters as a verb code and causes the verb display to be blanked.

Whenever **a** key is depressed, a unique five bit code associated with that key is generated, There **is,** however, no five bit code associated with the **STBY** key. If a key on the DSKY is pressed, the five bit code associated with that key is entered into bit positions **1** through **5 of** input channel **15** of the computer. Note that this input will cause a request for the KEYRUPT **1** program interrupt.

The switches for the keys are wired in series to insure that only one input at a time is presented to the diode encoder and, consequently, only one code at a time to the input channel. Trap reset signals are associated with the **DSKY.** When a key is released on the computer **DSKY,** signal TRAP **15** RESET is sent to trap circuitry in the computer associated with the **KEYRUPT 1** program priority interrupt.

1.9.2 DISPLAY INDICATORS. There are **24** display indicators on the DSKY: **21** digit display indicators and three sign display indicators. The digit display indicators are as follows:

a. M1 and **M2** which comprise the program display

b. V1 and **V2** which comprise the verb display

c. N1 and **N2** which comprise the noun display

d. **R1D1** through R1D5 which comprise the numerical portion of data display R1

e. R2D1 through R2D5 which comprise the numerical portion of data display **R2**

f. R3D1 through R3D5 which comprise the numerical portion of data display **R3**

The sign display indicators are as follows:

a. R1S which is the sign portion of data display R1

b. **R2S** which is the sign portion of data display **R2**

c. **R3S** which is the sign portion of data display **R3**

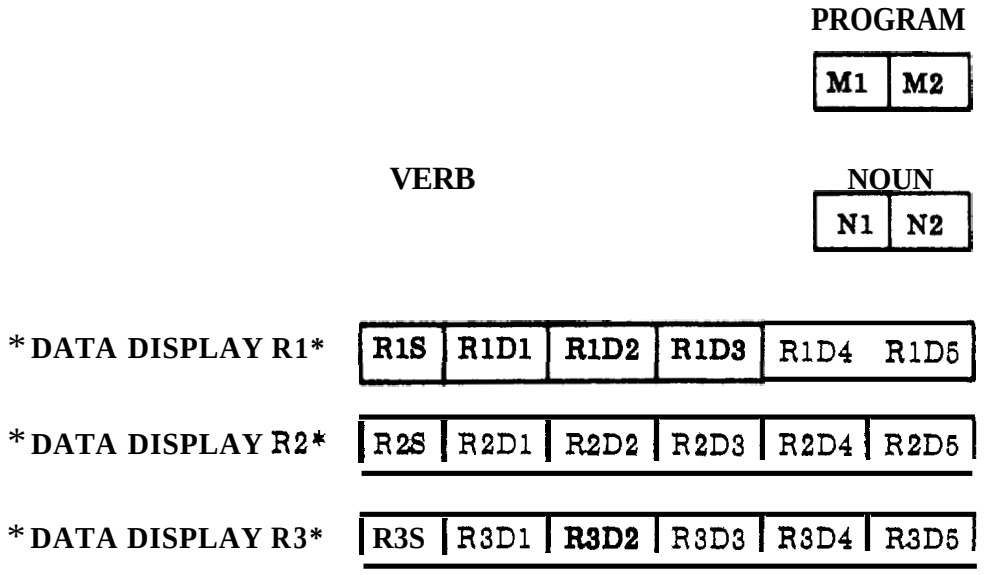
Figures **1-10** and **1-10A** show the displays and their locations.

Table **1-3** Lists the display indicators and their functions.

The relays shown in **figures 1-11** and 1-11A are used in conjunction with the display indications and **some** condition indicators. These relays are controlled Output Channel 10.

Table **1-3.** Display Indicators and Functions

Display Indicator	Function
PROGRAM indicators	Indicate program being processed by the computer.
VERB indicators	Indicate verb code entered at keyboard or commanded **by** the computer.
NOUN indicators	Indicate noun code entered at keyboard or commanded by the computer.
DATA DISPLAY indicators	Indicate numerical data entered at keyboard or commanded by the computer and sign associated with this numerical data if it is in decimal.



NOT INCLUDED ON FACE OF DSKY

Figure 1-10. Display Indicators

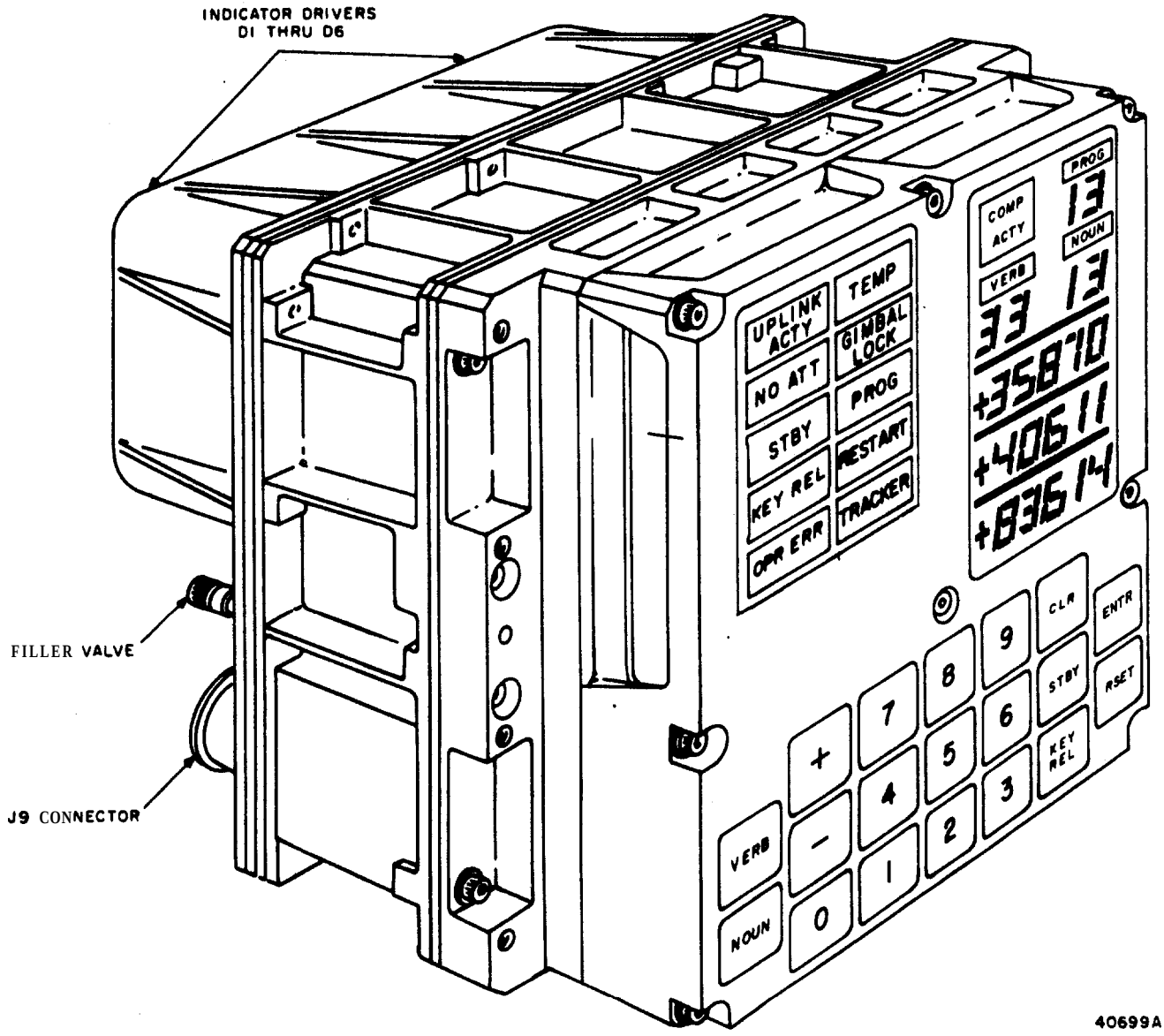


Figure 1-10A. Display and Keyboard

1.9.3 DSKY CONDITION INDICATORS. There are fourteen condition indicators displayed on the DSKY. Table **1-4** lists the indicators and their functions.

The UPLINK ACTY indicator will not be used on the LM DSKY.

The TEMP indicator **will** light if bit position 15 of input channel **30** contains a logic 0. This indicator can be lit during the standby mode,

The GIMBAL LOCK indicator will light if bit position *6* of output channel 10 contains a logic **1** and bit position **15** through **12** of the same channel are 1, 1, *0*, *0* respectively.

The PROG indicator will light if bit position 9 of output channel 10 contains a logic **1** and bit position **15** through **12** of the same channel are 1, 1, 0, *0* respectively.

The NO ATT indicator will light if bit position **4** of output channel 10 contains a logic **1** and bit positions **15** through **12** of the same channel are **1, 1,** 0, 0 respectively.

The TRACKER indicator will light if bit position **8** of output channel 10 contains a logic **1** and bit positions **15** through 12 of the same channel are 1, 1, 0, 0 respectively.

The STANDBY indicator will light if the STANDBY circuit **is** enabled. The indicator **will** also light if a light test **is** performed.

The KEY REL indicator will light if bit position 5 of output channel **11** is a logic **1.** This indicator is modulated by the flash signal.

The **OPR** ERR indicator will light if bit position **7** of output channel **11** is a logic **1.** This indicator is also modulated by the flash signal.

The COMP ACTY indicator will light if bit position 2 of output channel **11** is a logic 1.

1.9.4 DSKY OPERATION. The operator of the **DSKY** can communicate with the computer by the depression **of** a sequence of keys on the **DSKY** keyboard. Each depression of a **key** inserts a five bit code into the computer. The computer responds by returning a code to the DSKY which controls the display on a particular display panel or initiating an operation by the computer. The computer is also capable of initiating a display of information or a request for some action to the operator through the processing of its program.

1.9.4.1 Verb-Noun. The basic communication language used in the interchange of information **is** a pair of words **known** as the VERB **and** NOUN. Each of these words **is** represented by a two-digit octal code. The VERB code specifies **that** an action **is to be** performed. The NOUN code specifies on what the action is to be performed. An example of a VERB-NOUN code combination **is** given below.

VERB **16** -- MONITOR **IN DECIMAL ALL** COMPONENTS OF --

NOUN **21** -- PIPAS

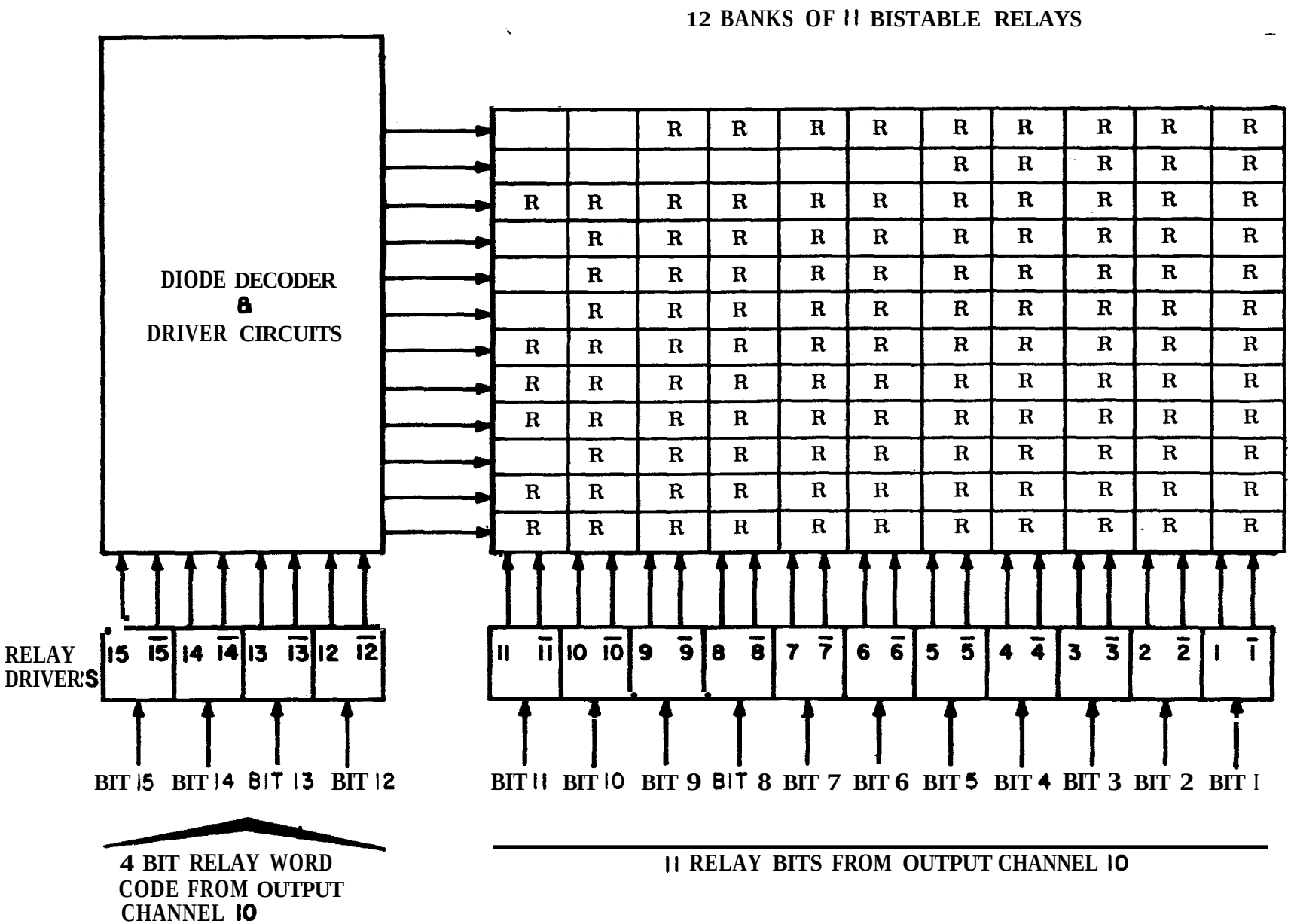


Figure **1-11.** **DSKY** Display Relay Circuitry

Resulting Action		Contents of Channel 10														
Row	Display	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
12	Program	1	1	0	0	0	0	1	0	0	0	0	0	0	0	0
12	Tracker	1	1	0	0	0	0	0	1	0	0	0	0	0	0	0
12	Spare	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0
12	Gimbal Lock	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0
12	Spare	1	1	0	0	0	0	0	0	0	0	1	0	0	0	0
12	No Att	1	1	0	0	0	0	0	0	0	0	0	1	0	0	0
12	Spare	1	1	0	0	0	0	0	0	0	0	0	0	1	0	0
12	Spare	1	1	0	0	0	0	0	0	0	0	0	0	0	1	0
12	Spare	1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
11	Reg 1 Pos 5	1	0	0	0	0	0	0	0	0	0	x	x	x	x	x
10	Reg 1 (+)	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0
10	Reg 1 Poa 4	0	1	1	1	0	x	x	x	x	x	0	0	0	0	0
10	Reg 1 Pos 3	0	1	1	1	0	0	0	0	0	0	x	x	x	x	x
9	Prog. Pos 1	1	0	1	1	0	x	x	x	x	x	0	0	0	0	0
9	Prog. Pos 2	1	0	1	1	0	0	0	0	0	0	x	x	x	x	x
8	Verb Pos 2	1	0	1	0	0	x	x	x	x	x	0	0	0	0	0
8	Verb Pos 1	1	0	1	0	0	0	0	0	0	0	x	x	x	x	x
7	Noun Pos 2	1	0	0	1	0	x	x	x	x	x	0	0	0	0	0
7	Noun Pos 1	1	0	0	1	0	0	0	0	0	0	x	x	x	x	x
6	Reg 1 (-)	0	1	1	0	1	0	0	0	0	0	0	0	0	0	0
6	Reg 1 Pos 2	0	1	1	0	0	x	x	x	x	x	0	0	0	0	0
6	Reg 1 Pos 1	0	1	1	0	0	0	0	0	0	0	x	x	x	x	x
5	Reg 2 (+)	0	1	0	1	1	0	0	0	0	0	0	0	0	0	0
5	Reg 2 Pos 5	0	1	0	1	0	x	x	x	x	x	0	0	0	0	0
5	Reg 2 Pos 4	0	1	0	1	0	0	0	0	0	0	x	x	x	x	x
4	Reg 2 (-)	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
4	Reg 2 Pos 3	0	1	0	0	0	x	x	x	x	x	0	0	0	0	0
4	Reg 2 Pos 2	0	1	0	0	0	0	0	0	0	0	x	x	x	x	x
3	Reg 2 Pos 1	0	0	1	1	0	x	x	x	x	x	0	0	0	0	0
3	Reg 3 Pos 5	0	0	1	1	0	0	0	0	0	0	x	x	x	x	x
2	Reg 3 (+)	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0
2	Reg 3 Pos 4	0	0	1	0	0	x	x	x	x	x	0	0	0	0	0
2	Reg 3 Pos 3	0	0	1	0	0	0	0	0	0	0	x	x	x	x	x
1	Reg 3 (-)	0	0	0	1	1	0	0	0	0	0	0	0	0	0	0
1	Reg 3 Pos 2	0	0	0	1	0	x	x	x	x	x	0	0	0	0	0
1	Reg 3 Pos 1	0	0	0	1	0	0	0	0	0	0	x	x	x	x	x

Figure 1-1 A. DSKY **Display Indications**

Table 1-4. DSKY Condition Indicators

Indication	Function
UPLINK ACTY	Indicates that Information is being received via uplink.
TEMP	Indicates that the stable member temperature has exceeded its design limits by ±5°F.
GIMBAL LOCK	Indicates that the middle gimbal has driven through an angle greater than 70° from its zero position.
PROG	Indicates that a program check has failed. This indicator is controlled by a computer program.
RESTART	Indicates: 1. That a word has been incorrectly transferred from memory - Parity fail 2. That the computer is in an endless control loop - TC Trap 3. That the computer has been interrupted for 30 milliseconds - RUPT lock. 4. That the computer has not accomplished a CSS new job within 1.28 sec. (Night watchman) 5. That a test alarm has been generated by program control.
TRACKER	Indicates rendezvous radar CDU failure or improper data from rendezvous radar.
OPR ERR	Indicates that the Keyboard and Display program has encountered some improper operating conditions.
KEY REL	Indicates that the internal program has attempted to use the Keyboard and Display System and found it busy.
STBY	Indicates that the computer is in the standby condition.
NO ATT	Indicates to the astronaut that the ISS is not suitable for use as an attitude reference.
COMP ACTY	Indicates that the computer is in a program other than dummy job and that the computer is not in standby mode.

This combination of VERB-NOUN codes causes the accumulation of PIPA counts (as accumulated by the computer) from each of the PIPA's to be displayed in R1 **(X** PIPA), R2 **(Y PIPA),** and **R3** (Z PIPA).

The standard procedure **of** inserting the VERB-NOUN codes via the keyboard is the depression of seven keys in a sequence. Using the VERB-NOUN codes previously discussed, the sequence of key depressions would be as follows:

a. VERB

b. 1

c. *6*

d. NOUN

e. **2**

f. **1**

g. ENTER

The ENTER key depression indicates to the computer that it should perform the operation indicated by the VERB-NOUN codes.

An alternate sequence of key depressions which would accomplish the same insertion of information would be as shown below:

a. NOUN .

b. **2**

c. **1**

d. VERB

e. **1**

f. **6**

g. ENTER

Whenever the VERB key is depressed, the two VERB **display** panels are blanked. Then as **the** digits **of** the VERB **code** are keyed in, the digits are displayed in the two VERB display **panels. For** example:

VERB KEY DEPRESSED -- VERB DISPLAY PANELS V1 AND V2 BLANKED

1 KEY DEPRESSED -- 1 DISPLAYED IN V1

6 KEY DEPRESSED -- 6 DISPLAYED IN V2

Whenever the NOUN key is depressed, the two NOUN display panels, N1 and N2, are blanked. As the two digits of the NOUN code are keyed in, the NOUN display panels display the digits of the NOUN code.

If the VERB–NOUN codes displayed in the VERB–NOUN display panels are those desired for the next entry of information, the VERB–NOUN codes need not be keyed in again. All that is required is the depression of the ENTER key. This indicates to the computer to use these codes again.

Prior to depressing the ENTER key, after entering the proper VERB–NOUN codes, the VERB–NOUN codes should be verified. If they are not the desired codes, the wrong action would be initiated which might cause damage to the system.

1.9.4.2 Data Loading. Some VERB–NOUN codes require more information to be keyed in other than the VERB–NOUN codes. If more data is required, after the depression of the ENTER key following the keying in of the VERB–NOUN codes, the VERB–NOUN display panels will flash on and off at a 1.5 cps rate. These display panels will continue to flash until all of the information associated with the VERB NOUN code has been keyed in. For example, using VERB 21 (WRITE 1ST COMPONENT INTO) NOUN 16 (TIME IN SECONDS), the entry sequence would be as follows:

a. VERB

b. 2

c. 1

d. NOUN

e. 1

f. 6

g. ENTER

After the ENTER key is depressed, the VERB–NOUN display panels will flash 21 and 16, respectively. This indicates that more information is required. In this case, it is a time in seconds. Assuming that the time to be entered is +75.25 seconds, the entry procedure would be as follows:

a. +

b. 0

c. 7

d. 5

e. 2

f. 5

g. ENTER

After the ENTER key is depressed, the VERB–NOUN display panels will stop flashing and remain on displaying VERB 21, NOUN 16. As the various keys are depressed while inserting the data, the digits are displayed in positions of one of the display registers corresponding to the order in which they were entered. For instance, when +75.25 seconds is being entered, the + key is depressed first and + is displayed in the R1S position. The 0 key is depressed and 0 is displayed in R1D1. The 7 key is depressed and 7 is displayed in R1D2. This continues until the information is completely keyed in. The ENTER key depression after keying the desired information not only stops the flashing of the VERB–NOUN display but indicates to the computer that it should proceed and perform the operation specified.

VERB 21 (WRITE 1ST COMPONENT INTO) 22 (WRITE 2ND COMPONENT INTO) 23 (WRITE 3RD COMPONENT INTO) 24 (WRITE 1ST AND 2ND COMPONENTS INTO) and 25 (WRITE 1ST, 2ND AND 3RD COMPONENTS INTO) are used to enter one, two or three components or portions of data into the computer. If VERB 25 (WRITE 1ST, 2ND AND 3RD COMPONENTS INTO) is entered, the VERB display will illuminate and display 25. When the ENTER key is depressed after keying in the VERB–NOUN code, the VERB will display 21 (WRITE 1ST COMPONENT INTO) flashing. After the first portion of data has been keyed in, displayed in R1 and the ENTER key depressed, the VERB display will illuminate 22 (WRITE 2ND COMPONENT INTO) flashing. After the second component or portion of the data is keyed in, displayed in R2 and the ENTER key depressed, the VERB display illuminates 23 (WRITE 3RD COMPONENT INTO) flashing. The third component of data is then entered, displayed in R3, and the ENTER key is depressed. The VERB display stops flashing and the computer proceeds to utilize the information entered.

1.9.4.3 Correcting Erroneous Data. Any time prior to depressing the last ENTER in the loading sequence, i.e., the ENTER after the third component was inserted in the previous paragraph, erroneous information can be corrected. To correct erroneous data, the CLEAR key is used. This key causes the display register, R1, R2, and R3, last loaded to be cleared and also clears the corresponding information loaded into the computer. For example, if a three component load is being keyed in and it is discovered that an error exists in the first component of data in R1, after R3 has been loaded but prior to the last ENTER, the following must be done to correct the data:

DEPRESS CLEAR KEY -- R3 BLANKED -- VERB 23 DISPLAYED

DEPRESS CLEAR KEY -- R2 BLANKED -- VERB 22 DISPLAYED

DEPRESS CLEAR KEY -- R1 BLANKED -- VERB 23 DISPLAYED

RELOAD R1, R2 and R3

The CLEAR key is not used to clear the VERB, NOUN or PROGRAM displays.

1.9.4.4 Decimal and Octal Display and Loading. Decimal and octal displays or loadings are distinguished by use **of** the + and - displays or key inputs. Whenever decimal data is to be loaded, the + or - key must be depressed prior to keying in the digits of the data to be Loaded. If the sign keys are not used, the data is assumed to be in octal **form by** the computer. Whenever data is displayed using a sign, + or -, the displayed data is **in** decimal. Otherwise, when the sign is not used and **R1S, R2S** or **R3S** are blanked, the data displayed is in octal.

1.9.4.5 Monitor vs. Display. Whenever a display type **VERB** is used, the requested data is transferred to the **DSKY** panels once each time the data is requested.

Monitoring type **VERBS,** in contrast, are periodically updated and the display of the requested data changes as the requested data in the computer changes. The updating **of** the displayed data for a monitor type **VERB** is accomplished approximately every **1** second.

1.9.4.6 Changing of Major Mode. The major mode refers to system operations in the various phases **of** a flight or while operating on the ground. Examples of major modes are:

PRELAUNCH ALIGNMENT

GUIDANCE REFERENCE RELEASE AND BOOST

ETC.

In order to request that the system initiate one or more major modes of operation, *a* different sequence **of** entering information through the **DSKY** is required. The procedure would be as follows using **VERB 37 (CHANGE MAJOR** MODE **TO).**

a. **VERB**

b. **3**

c. **7**

d. ENTER

When the ENTER key is depressed, after keying in VERB **37,** the VERB display panels flash and the NOUN display panels are blanked. Now the two-digit octal code for the desired major mode can be entered through the keyboard. **As** the appropriate keys are depressed, the digits of the code are displayed in the NOUN display panels. When the ENTER key **is** depressed after keying in the two code digits, the major mode code is displayed in the two PROGRAM display panels M1 and **M2.** If the operator wants to initiate the major mode PRELAUNCH ALIGNMENT which use the program number 01, the following keying sequence must be used:

a. VERB

b. **3**

c. **7**

d. ENTER

e. 0 Entry for Prelaunch Alignment mode request

f. **4** Entry indicating phase to enter Prelaunch Alignment

g. ENTER

The two program display panels would now display 01 and the NOUN panels would be blanked.

1.9.4.7 Mode initiation. Another group of VERBS enable the operation to initiate system mode6 **of** operation. Examples of these are:

COARSE ALIGN -- VERB **41**

FINE ALIGN IMU -- VERB **42**

ZERO -- VERB **40**

Some **of** these VERBS **do** not require an associated NOUN code. For example, **if** the change major mode is to be initiated, the procedure would be:

a. VERB

b. 3

c. **7**

d. ENTER

This would cause the system to change major mode. Other **VERBS** do require **NOUN** codes such as **VERB 40 (ZERO).** This **VERB** refers to CDU's and the NOUN code required with this **VERB** code specifies either the inertial or radar CDU's (NOUN **20,** inertial **CDU;** NOUN **40,** rendezvous radar angles). If it is desired to ZERO the inertial CDU's, the keying procedure would be:

a. **VERB**

b. **4**

c. **0**

d. NOUN

e. **2**

f. 0

g. ENTER

1.9.4.8 Computer Control of the **DSKY.** Display and monitoring of various data can be accomplished **by** the computer through its own initiative without requests for the data by the operator. The appropriate **VERB-NOUN** codes are displayed with the data *so* that it can be properly identified and used by the operator. Whenever the computer has initiated the display or monitoring of some data, the data will be displayed for at least 10 seconds. After this time duration, the computer is free to change the data displayed if it *so* desires.

The computer is also capable of requesting the operator to perform **an** action. The action that is requested is usually specified by a combination of **VERB-NOUN** codes and additional information displayed in one or more of the display registers, R1, R2 and R3. For example, if **VERB 50 (PLEASE** PERFORM) **NOUN 25 (CHECKLIST)** is displayed in the **VERB-NOUN** display panels, R1 will display a numerically coded checklist item. When the operator has performed the requested action, the ENTER key should **be** depressed. This indicates to the computer that the operation has been completed. **If** the operator does not wish to perform the action requested, he may **use VERB** 33 (PROCEED WITHOUT **DATA)** or **VERB 34 (TERMINATE).** These **VERB** codes indicate to the computer to continue on without the data or requested action as best it can or to terminate the function it is performing.

1.9.4.9 DSKY/Computer/Operator Interlocks. While the operator of the **DSKY** is using the DSKY to load, display, etc., the computer cannot interrupt this process. **An** interlock is set up by the computer inhibiting itself from using the **DSKY.** Therefore, the DSKY operator should remove this interlock when he is finished using it. This is accomplished **by** depressing the **KEY RELEASE** key, This action removes the DSKY-OPERATOR interlock and enables the computer to use the **DSKY.**

The computer is capable of requesting that the **DSKY** operator release the **DSKY** *so* the computer may use it. Illuminating the **KEYRLSE** panel on the **DSKY FAILURE** INDICATOR **PANEL** indicates that the computer has some data to display to the operator. The operator **is** not obligated to release control of the **DSKY** if he wishes to continue to use it.

As previously mentioned, when the computer has initiated a display of data, the data will be displayed for at least 10 seconds before the computer is able to display different data. This is because of an interlock the computer imposes on itself to enable the operator time enough to read the data displayed. After 10 seconds have elapsed, the computer drops the interlock and is free to display different data to the operator.

1.9.5 VERB-NOUN LIST. Contained in this section of the study guide is a complete listing of the VERB and NOUN codes which are used with the Sunburst Rev. 14 computer program. A brief description is also given for each of the VERB and NOUN codes along with the scaling of the data converted and displayed on the DSKY as a result of NOUN code usage. Keep in mind that many combinations of these codes exist; however, some of the combinations are non-sensical or illegal. Some VERB codes do not require a NOUN code to completely specify the desired action.

1.9.5.1 Verb Codes. The VERB codes are divided into two groups - Ordinary an Extended. The ordinary verbs generally are involved in the manipulation (loading display, etc.) of data. The extended verbs, in general, are used for initiation of actions (moding requests, equipment operation, etc.).

ORDINARY VERBS

Verb Code	Function	Display Location
00	Illegal	
01	Display (in octal) 1st component of:	R1
02	Display (in octal) 2nd component of:	R1
03	Display (in octal) 3rd component of:	R1
04	Display (in octal) 1st and 2nd components of:	R1, R2
05	Display (in octal) 1st, 2nd and 3rd components of:	R1, R2, R3
06	Display (in decimal) all component's of:	As appropriate
07	Double Precision decimal display	R1, R2
10	Spare	
11	Monitor (in octal) 1st component of:	R1
12	Monitor (in octal) 2nd component of:	R1
13	Monitor (in octal) 3rd component of:	R1

Verb Code	Function	Display Location
14	Monitor (in octal) 1st and 2nd component of:	R1, R2
15	Monitor (in octal) 1st, 2nd and 3rd component of:	R1, R2, R3
16	Monitor (in decimal) all component(s) of:	As appropriate
17	Monitor Double Precision decimal	R1, R2
20	Spare	
21	Load 1st component into:	R1
22	Load 2nd component into:	R2
23	Load 3rd component into:	R3
24	Load 1st and 2nd components into:	R1, R2
25	Load 1st, 2nd and 3rd components into:	R1, R2, R3
26	Spare	
27	Fixed Memory Display	
30	Request Executive	
31	Request Waitlist	R1
32	Bump Displays c(R2) into R3, c(R1) into R2	
33	Proceed without Data	
34	Terminate Current test or Load Request	
35	Test lights	
36	Fresh Start	
37	Change **Major** Mode to:	

EXTENDED **VERBS**

40	Zero (used with NOUN 20, ICDU; NOUN 40, RR angles, NOUN *70;* Optical Tracker Angles; only)
41	Coarse Align (used with NOUNS *20,* 40 and *70* only)
42	Fine Align IMU

Verb Code	Function	Display Location
43	Load IMU Attitude Error Meters	
44	Illegal Verb	
45	Command LR to Position 2	
46	Sample Radar Once per Second	
47	Perform LM FCS TEST	
50	Please Perform	
51	Please Mark	
52	Please Mark Y	
53	Please Mark X or Y	
54	Pulse Torque GYRO's	
55	Align Time	
56	Perform Banksum	
57	Perform System Test	
60	Illegal Verb	
61	Illegal Verb	
62	Scan LM Inbits	
63	Initialize AGS	
64	Illegal Verb	
65	Illegal Verb	
66	Illegal Verb	
67	Illegal Verb	
70	Illegal Verb	
71	Illegal Verb	
72	Illegal Verb	

Verb Code	Function
73	RHC Used For Minimum Impulse
74	RHC Used For Rate Command
*75	**DAP** Wide Deadband
*76	**DAP** Narrow Deadband
77	Illegal Verb

1.9.5.2 Verb Descriptions.

ORDINARY VERBS

Verbs 01 - 05	Perform octal displays of data.
Verb 06	Performs decimal display of data, The scale factors, types of scale factor routines, and component information are stored within the computer for each Noun which is required to display in decimal.
Verb 07	Performs a double precision decimal display of data. It does no scale factoring. It merely performs a **10** character fractional decimal conversion of two consecutive erasable registers using R1 and R2 (the sign is placed in the R1 sign position; the R2 sign position is blank). It cannot be used with Mixed Nouns. Its intended use is primarily with "Machine Address to be Specified" Nouns. If this verb is used with nouns that are inherently not double precision, the display will be meaningless.
Verbs **11 - 17**	The monitor verbs allow other keyboard activity. It is ended by terminate, **VERB 34,** any noun-verb subroutine that passes the **DSKY** block or another monitor. Monitor action is suspended but not ended, by any keyboard action except error reset and begins again when the KEY **RELEASE** is initiated.
Verbs **21 - 25**	Perform data load. Octal quantities are unsigned. Decimal quantities are preceded by a + or - sign.
Verb **27**	Bank Display. This Verb is included to permit displaying the contents of fixed memory in any bank. Its intended use is for checking program ropes and the **BANK** position of program ropes.

* Not included in Sunburst Rev **14** Listing

ORDINARY **VERBS** (Cont'd)

Verb **30** — Enters request to Executive Routine for any machine address **with priority. This Verb is** used **with the** Noun "Machine Address to be Specified". This Verb assumes that Noun **26** has been preloaded with

Component **1** — Priority (bits 10-14), bit **1** = 0 for NOVAC and **1** for **FINDVAC.**

Component **2** — Job address **(12** bits)

Component **3** — Both Bank Constants.

The End of Job subroutine is performed after the request is entered. The display system is also released.

Verb **31** — Enters request to Waitlist Routine for any machine address with any delay. This Verb is used with the "Machine Address to be Specified" Noun. This Verb assumes that Noun **26** has been preloaded with

Component **1** — Delay (the desired number of 10 millisecond units of delay in the low bits)

Component **2** — Task Address **(12** bits)

Component **3** — Both **Bank** Constants

The End of Job subroutine is performed after the request is entered. The Display system is also released.

Verb **32** — Display Shift. Useful for preserving an existing display of a quantity while displaying another quantity.

Verb **33** — Proceed without Data. Informs routine requesting data to be loaded that the operator chooses not to load fresh data, but wishes the routine to continue as best it can with old data. Final decision for what action should be taken is left to requesting routine.

Verb **34** — Terminate. Informs routine requesting data to be loaded that the operator chooses not to load fresh data, and wishes the routine to terminate. Final decision for what action should be taken is left to requesting routine. If Monitor is one, it is turned off.

Verb **35** — The Test **Lamps** routine checks all of the DSKY lamps. After **5** seconds, the caution and status lamps are returned to their original setting.

ORDINARY VERBS (cont'd)

Verb 36	Initializes the program control software and Keyboard and Display System Program.
Verb **37**	This verb changes the major mode. This is accomplished by inserting VERB **37** ENTER, MAJOR MODE, ENTER. The new major mode number is in the noun display until ENTER is push. At this time the new major mode number will be in the Program display.

EXTENDED VERBS

Verb **40**	Must be used with Noun **20** (ICDU), Noun 40 **(RR** Angles) or Noun 70 (Optical Tracker Angle) only.
Verb **41**	Must be used with Noun 20, Noun **40** or Noun 70 only.
Verb **42-43**	Call programs that perform the indicated PGNCS procedure.
Verb **45-46**	Call programs to perform the indicated radar procedure.
Verb **47**	Call program to perform the digital autopilot test.
Verb **50**	This verb **is** used only by internal routines that wish the operator to perform a certain task. It should never be keyed in by the operator. It is usually used with Noun **25** (Checklist). The coded number for the Checklist Item to be performed is displayed in register R1 by the requesting routine. Once the operator has performed the requested action, he should press ENTER to indicate that the Checklist Item has been performed. If he wishes not to perform the requested action, he should key in the Verb "Proceed Without Data".
Verbs **51-53**	Verbs **51, 52** and **53** are used only by internal routines that wish the operator to MARK. They should never be keyed in by the operator. It is usually used with Noun **30** (Star Numbers). The numbers of the stars to be marked are displayed in registers **R1, R2,** R3 by the requesting routine. He should never press ENTER with Verbs **51, 52** or **53.**
Verb 54	Call program that performs the indicated PGNCS procedure.
Verb **55**	Used to update the computer clock.
Verb *56*	Check the sum of the fixed memory bank as a cursory check of the validity of the memory.

EXTENDED VERBS (cont'd)

Verb 57	Call program that will perform the selected system test, The test is selected by VERB 57 ENTER, CODE ENTER. The codes are listed below in table 1-5.
Verb 62	Call program to scan channel 30 through 32.
Verb 63	Call program that performs the indicated PGNCS operation.
Verbs 73 and 74	Indicates in what mode of operation the rotation hand controller is to be used.
*Verbs 75 and 76	Indicates to the computer the deadband being used for attitude control.

* **Not included in** Sunburst Rev **14 Listing**

1.9.5.3 Noun Codes. The Noun Codes refer to a computer memory register or registers. These codes are divided into two groups - Normal Nouns and Mixed Nouns. The Normal Nouns refer to data stored in sequential memory registers and the data contained in or to be loaded into these registers must use the same scaling. For example, Noun **21** refers to the PIPA counters which are three registers sequentially located in the computer's memory. All three of the quantities associated with these registers require the same scaling for display purposes.

Code	Nomenclature
0	Illegal
1	Gyro Drift Test
2	Repeat of IMU Test
3	IMU Alignment Test
4	**M U** Check
5	Gyro Torquing Test
6	Gyro Compassing
7	DSKY Check
10	Semi-Automatic Moding Check
11	Semi-Automatic Interface Test
12	**AOT** Angle Check
13	RR/Antenna Tracking
14	High Speed Radar Sampling
15	Zero **All** Erasable Memory Banks
16	Display Inertial Data Test

Table **1-5.** System Test Codes **(VERB 57)**

The other type of noun code, the Mixed Noun, refers to data which is not necessarily located in sequential memory registers nor necessarily use the same scaling. For example, Noun 60 when used with a display verb causes the display of the contents of the landing radar velocity Z and the computer's real time reference. These two quantities are not stored in sequential memory registers nor do they require the same scaling or conversion techniques for their display.

NORMAL NOUNS

Noun Code	Function
00	Not in use.
01	Specify Machine Address (.XXXXX)
02	Specify Machine Address (XXXXX.)
03	Specify Machine Address (XXX.XX Degrees)
04	Specify Machine Address (XXX.XX Hours)
05	Specify Machine Address (XXX.XX Seconds)
06	Specify Machine Address (XX.XXX Gyro Degrees)
07	Spare
10	Channel to be specified
11	Spare
12	Spare
13	Spare
14	Spare
15	Increment Machine Address (octal only)
16	Time (XXX.XX seconds)
17	Time (XXX.XX **Hours)**
20	ICDU **(XXX.XX** Degrees)
21	PIPA's (XXXXX. Pulses)
22	New Angles I (XXX.XX Degrees)
23	Delta Angles I (XXX.XX Degrees)
24	Delta Time (XXX.XX Seconds)

NORMAL NOUNS (Cont'd.)

Noun Code	Function
25	Checklist (XXXXX.)
26	Prio/Delay, Address, **BBCON** (Octal Only)
27	Self Test On/Off Switch (XXXXX.)
30	Star Numbers (XXXXX.)
31	Failreg, SFAIL, ERCOUNT [R1, R2, R3 (octal only)]
32	Midcourse Decision Time [XXX. XX Hours (Internal Units = Weeks)]
33	Midcourse Ephemeris Time [XXX. XX Hours (Internal Units = Weeks)]
34	Midcourse Measured Quantity (XXXX. X Kilometers)
35	Inbit Message (Octal Only)
36	Landmark Data 1 (Octal Only)
37	Landmark Data 2 (Octal Only)
40	RR Trunnion and Shaft Angles (XXX. XX Degrees)
41	New **RR** Trunnion and Shaft Angles (XXX. XX Degrees)
42	AOT Rotation Angles (XXX. XX Degrees)
43	AOT Detent Code (XXXXX.)
44	Forward Velocity Lateral Velocity (XXXXX. feet/second)
45	Rotational Hand Controller Angle Rates (XXXXX. Degrees/second)
46	Spare
47	Spare
50	Spare
51	Spare

NORMAL **NOUNS** (Cont'd.)

Noun Code	Function
52	Gyro Bias Drift (.BBXXXXX millirad/second)
53	Gyro Input Axis Acceleration Drift (.BBXXXXX $\frac{\text{millirad/sec}}{\text{cm/sec}^2}$)
54	Gyro Spin Axis Acceleration Drift (.BBXXXXX $\frac{\text{millirad/sec}}{\text{cm/sec}^2}$)
55	LR Altitude, Time (**XXXXX.** Feet, XXX.XX sec)
56	LR V_X, Time (XXXXX. Feet/Sec., XXX.XX Seconds)
57	LR V_Y, Time (XXXXX. Feet/Sec., XXX.XX Seconds)
60	LR V_Z, Time (**XXXXX.** Feet/Sec., XXX.XX Seconds)
61	Target Azimuth and Elevation (XXX.XX Degrees, XX.XXX **Deg.)**
62	**RR** Range, Shaft, Trunnion (**XXXXXB.** Feet, XXX.XX Degrees, XXX.XX Degrees)
63	RR Range Rate, Shaft, Trunnion (XXXXX. Feet, **XXX.XX** Deg., XXX.XX Deg.)
64	Initial Altitude, Final Altitude, Altitude Rate (XXXXX. Feet, **XXXXX.** Feet, XXXXX. Feet/Second)
65	Sampled Time (**XXX.XX** Hours, XXX.XX Seconds)
66	System Test Results (XXXXX., .XXXXX, **XXXXX.**)
67	Delta GYRO Angles (XX.XXX Degrees for Each)
70	Optical Tracker Azimuth **and** Elevation Angles (**XXX.XX** Degree, XXX.XX Degree)
71	Desired Optical Tracker Azimuth and Elevation Angles (XXX.XX Degrees, XXX.XX Degrees)
72	Delta Position (**XXXX.X** Kilometers **for** Each)
73	Delta Velocity (XXXX.X Meters/Second **for** Each)
74	Midcourse Measurement Data (XXX.XX Hours, XXXX.X Kilometers, XXXXX.)

NORMAL NOUNS (Cont'd.)

Noun Code	Function
75	Midcourse Measurement Deviations (XXXX.X Kilometers, XXXX.X Meters/second, XXXX.X Kilometers)
76	Position Vector (XXXX.X Kilometers for Each)
77	Velocity Vector (XXXX.X Meters/Second for Each)

1.10 INTERRELATIONSHIP OF PROCESSING FUNCTIONS

The interrelationship of processing functions in the computer becomes quite involved especially when all of the possible combinations of the processing functions are considered. The processing of counter interrupt inputs enabling the accumulation of incremental data is performed as required. The processing of program interrupts occurs as required handling the major portion of the input and output functions for other programs and routines, and timing the execution of various tasks. The processing of program controlled processing functions is carried on with the priority of the processing routine determining when a particular job is to be processed. The scheduling, terminating, changing, etc. of jobs and tasks is continuously being performed. Considerable interchange of data between various jobs and tasks is continuously in process.

At this point in the study guide, an example is given showing the interrelationships of some of these processing functions. The example used is the complter controlled IMU ZERO mode switching routine. In this example, it is assumed that the request to perform the mode switching routine is made through DSKY entries, although it is used by other mission programs and forms a part of these programs.

In order to request the IMU ZERO mode switching routine to be performed, the astronaut enters through the DSKY keyboard this sequence of key depressions:

a. VERB

b. 4

c. 0

d. NOUN

e. 2

f. 0

g. ENTER

This sequence of entries indicates to the computer that is should zero the inertial CDU channels, VERB 40 states that something *should* be zeroed while NOUN 20 specifies the inertial CDU channels.

As each of the keys are depressed for the VERB-NOUN codes, the control of the computer hardware is forced to the KEYRUPT routine. (See figure 1-12.) The KEYRUPT routine processes the keycode input and uses the Executive routine to schedule the PINBALL program to be processed on a program priority basis. After the scheduling is completed, the Executive routine returns control to the KEYRUPT routine which returns control back to the processing function Interrupted by the KEYRUPT.

Within 20 m.s. of the time the Pinball program was scheduled, the processing of one of the routines of PINBALL is initiated under Executive control, if PINBALL has the highest priority of the scheduled jobs. PINBALL processee the input keycodes and furnishes the T4RUPT routine with the data required for display of the keycode inputs. The T4RUPT routine drives the DSKY displays with this information. The DOWNRUPT routine provides the same Information for transmission by the DOWNLINK telemetry system. After PINBALL has completed the processing of each keycode input, the PINBALL job is terminated and control is returned to the next lower priority scheduled job under executive control.

Finally, when the ENTER key is depressed, the KEYRUPT routine is again initiated which again schedules the PINBALL program through the Executive routine and returns control to the processing function which was interrupted. Again, within 20 m.s., PINBALL program processing is initiated if it is the highest priority scheduled job. With the ENTER keycode input, the PINBALL routine uses the assembled VERB and NOUN codes to transfer control via BANKCALL to the IMU ZERO mode switching routine. This routine is executed under control of the scheduled PINBALL routine which is, in turn, executed under control of the Executive routines.

The IMU ZERO mode switching routine checks to determine if the IMU is being caged. If the IMU CAGE signal is present, IMU ZERO is terminated, through Executive action. If the IMU CAGE signal is not present, the inertial CDU and IMU fail indications are inhibited. A command is issued to zero the inertial CDU channels. After the inertial CDU channels are commanded to zero, the IMU ZERO2 task is scheduled on the WAITLIST to be executed in 320 milliseconds. After scheduling the IMU ZERO2 a check is made to insure that the IMU is operating, The IMU ZERO job is now terminated through Executive action. The 320 millisecond time delay in the execution of this moding operation allows sufficient time for the inertial CDU channels to zero.

After 320 milliseconds have elapsed from the time that IMU ZERO2 task was scheduled on the WAITLIST, the T3RUPT routine will result as a function of overflow. The TIME 3 counter and the IMU ZERO2 task will be executed. If the IMU CAGE signal is present, IMU ZERO2 is terminated through TSRUPT and the interrupted job will be resumed. If the IMU CAGE signal is not present, the computer's inertial CDU counters are ret to zero and the inertial CDU zero command is removed. A four second delay is now scheduled on the WAITLIST. At the end of this delay, the task IMU ZERO3 is performed. After IMU ZERO3 is scheduled on the WAITLIST, processing control is returned to the interrupted job through the TSRUPT routine. This ends the IMUZERO routine.

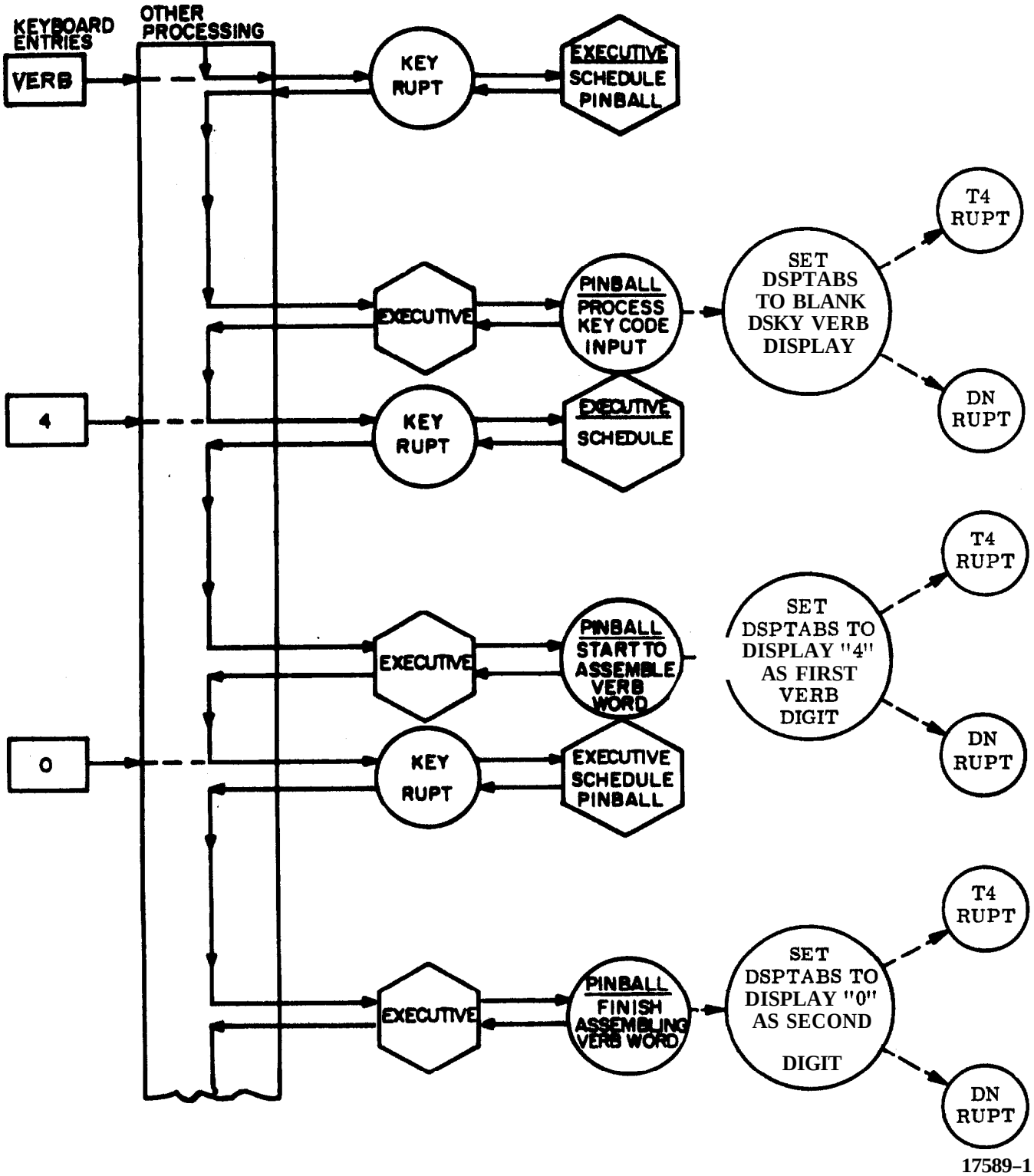


Figure 1-12. Simplified Processing for Zero IMU - CDU Routine (Sheet 1 of 4)

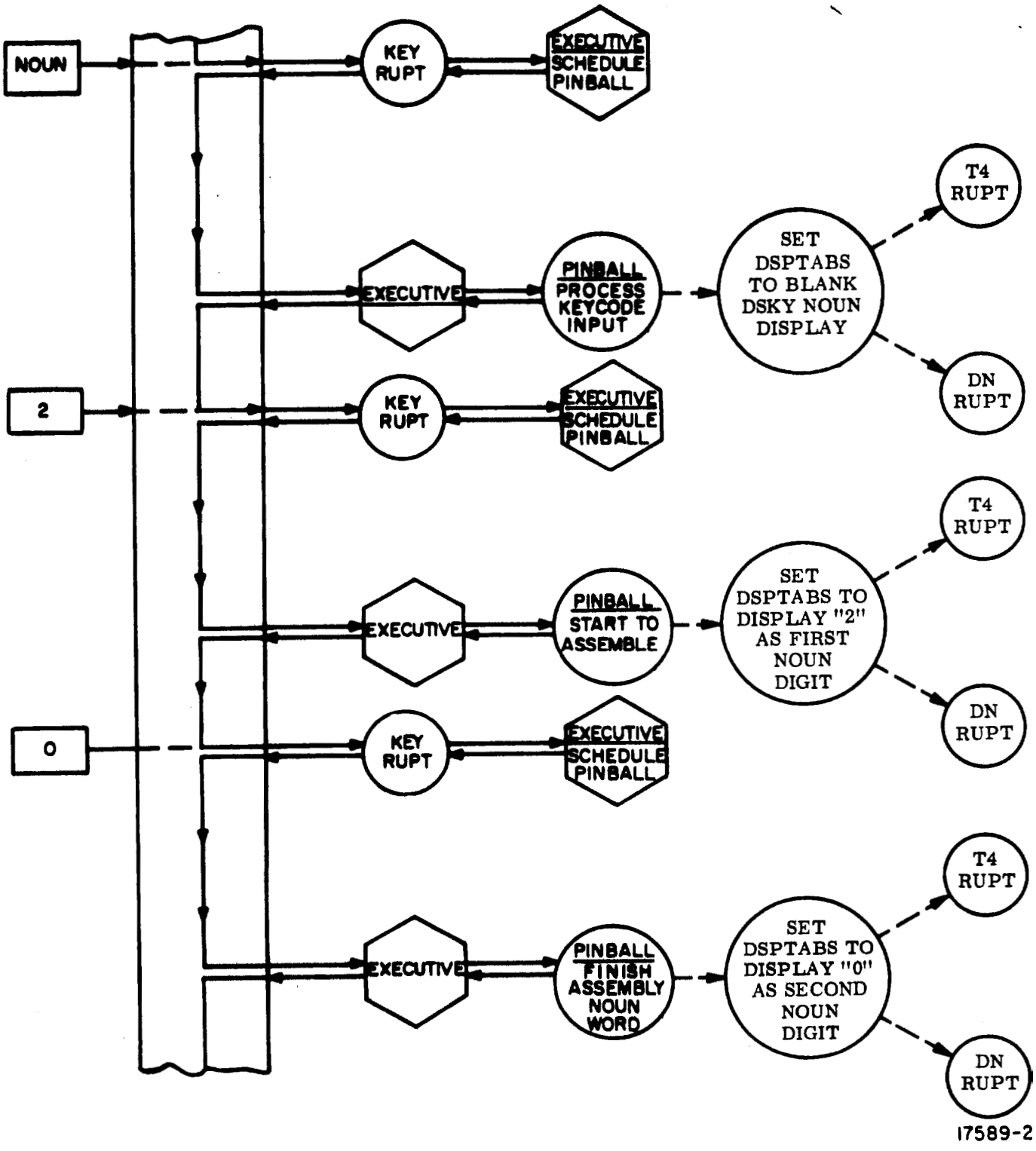


Figure 1-12. Simplified Processing for Zero N U - CDU Routine (Sheet 2 of 4)

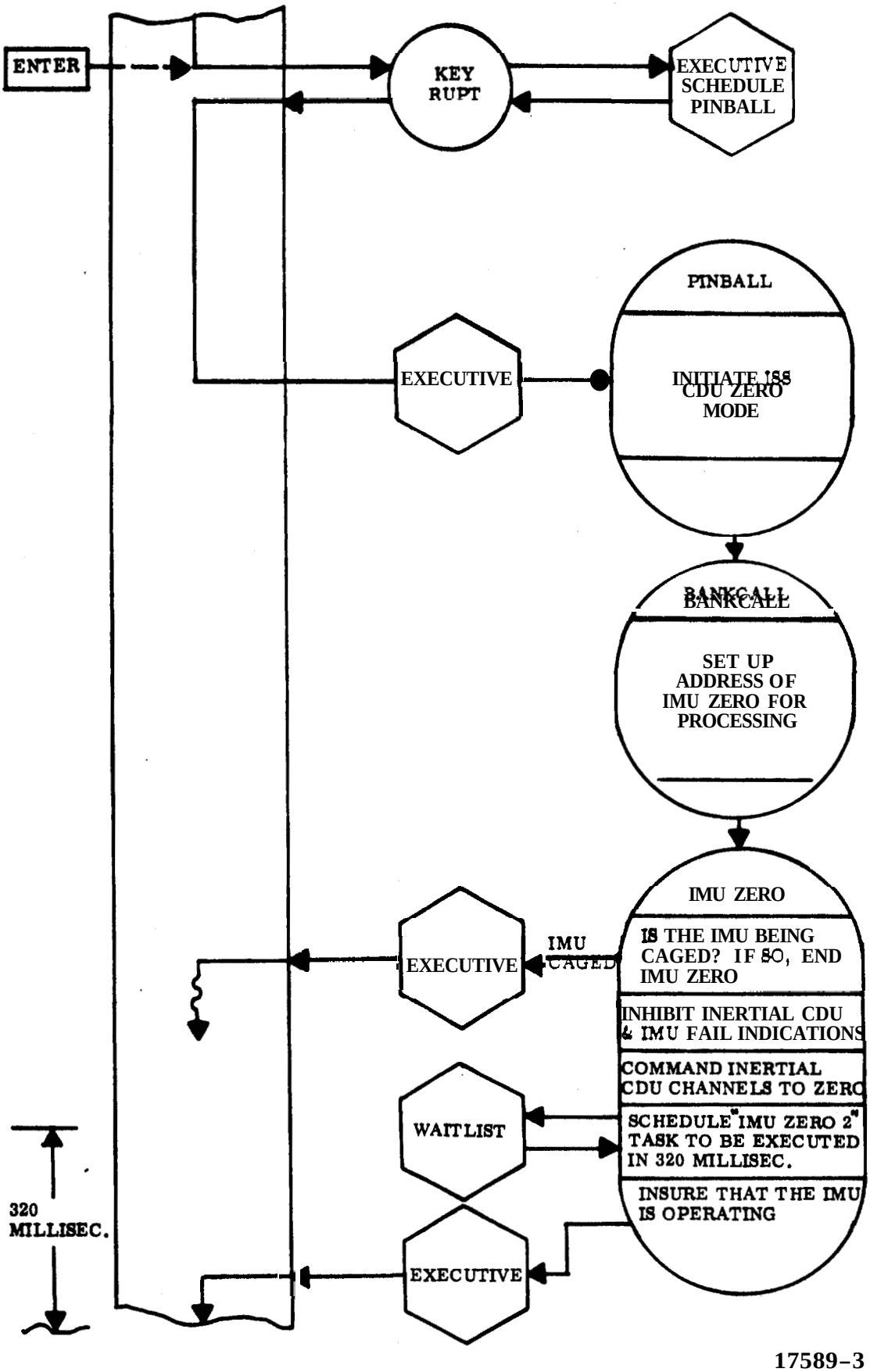


Figure 1-12. Simplified Processing for Zero IMU - CDU Routine (Sheet 3 of 4)

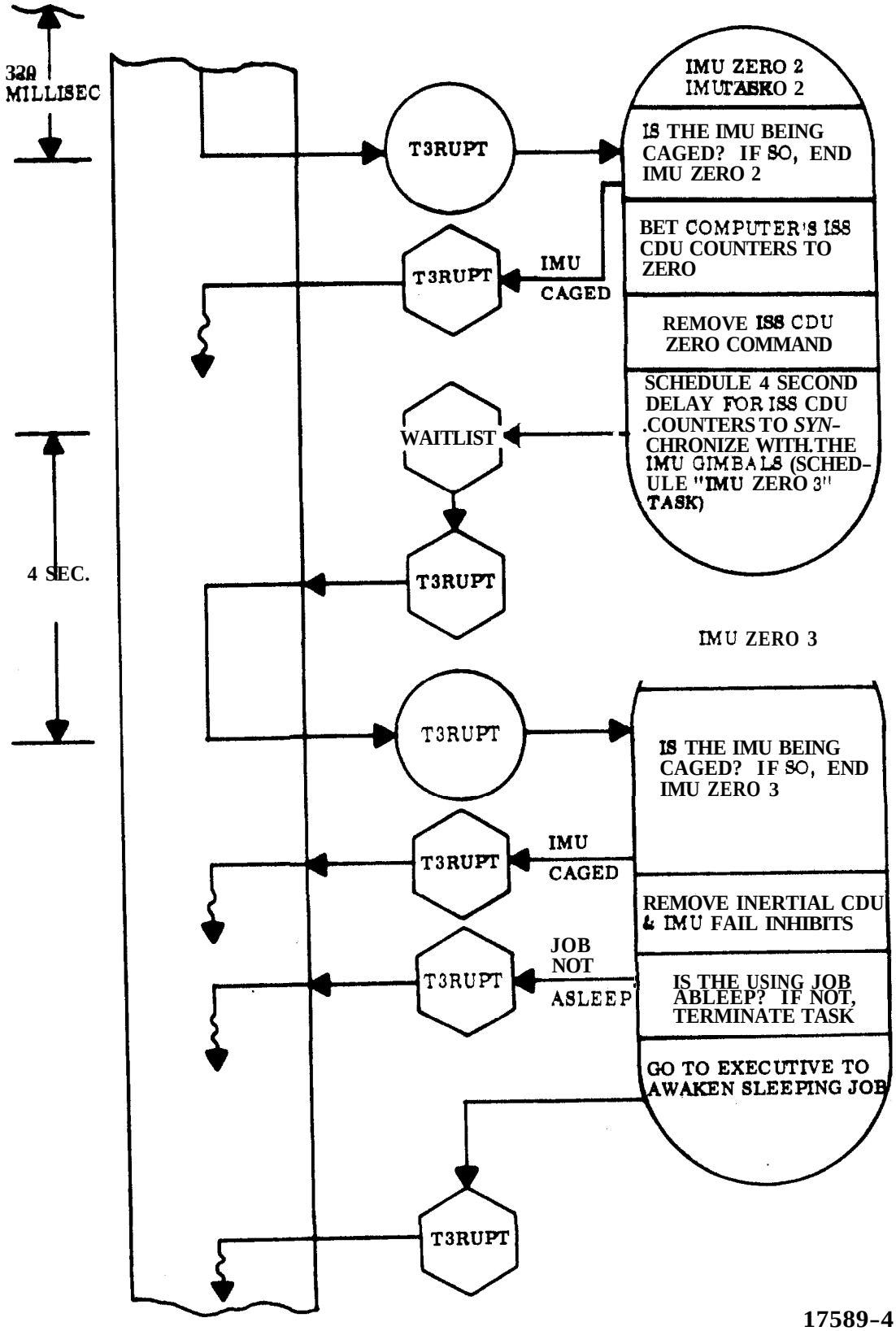


Figure 1-12. Simplified Processing for Zero IMU - CDU Routine (Sheet 4 of 4)

SECTION II

EXECUTIVE CONTROL OF COMPUTER PROCESSING

INTRODUCTION

This section of the study guide presents the Executive Control routines of the computer which includes the Executive, Waitlist, TSRUPT, Phase Table Maintenance routines. These routines provide control over the execution of all the processing performed by the computer with the exception of processing performed by the T4RUPT, KEYRUPT 1 & 2, UPRUPT, RADAR RUPT, CONTROLLER RUPT and DOWNRUPT. Also, the processing resulting from a hardware detected computer malfunction is not controlled by the Executive Control routines.

2.1 THE EXECUTIVE ROUTINE

The Executive routine of the computer controls all processing performed by the computer on the basis of program priorities. It provides for the scheduling of jobs, a means for changing and terminating jobs, and the capability of deactivating and reactivating jobs. The Executive routine consists of eight subroutines which are used by jobs and tasks to perform the functions mentioned above. The eight subroutines which are a part of the Executive routines are:

a. FINDVAC

b. NOVAC

c. CHANGE JOB

d. END OF JOB

e. JOBSLEEP

f. PRIORITY CHANGE

g. DUMMY JOB

h. WAKE JOB

2.1.1 FINDVAC AND NOVAC SUBROUTINES. The FINDVAC and NOVAC subroutines of the Executive routine provide for the scheduling of jobs. The FINDVAC subroutine is used to schedule a job which requires a fairly large amount of temporary storage for the variables involved in the job. This routine reserves a VAC (vector accumulator) area for use with a job and places the job on the Executive's core set list.

By placing the job on the core set list, the job is scheduled and a core set area is reserved for the job. Therefore, when a job is scheduled through the FINDVAC subroutine, a VAC area consisting of 44 memory registers and core set area consisting of twelve memory registers are reserved for use by the job. Figures 2-1 and 2-2 show the core set and VAC areas used by the computer.

	Memory Address	Use
Core *Set* Area #0	140 ↓ 146	MPAC (Multi-Purpose Accumulator)
	147	MODE (+1 for TP, +0 for DP, or -1 for Vectors)
	150	LOC (Location Associated with Job)
	151	BANKSET (Usually contains BBANK Setting)
	152	PUSHLOC (Word of Packed Interpretive Parameters)
	153	**PRIORITY** (Priority of Present Job and VAC Area Address, if required)
Core Set Area #1	154 ↓ 167	(See Note)
Core **Set** Area #2	170 ↓ 203	(See Note)
Core *Set* Area #3	204 ↓ 217	(See Note)
Core Set Area #4	220 ↓ 233	(See Note)
Core **Set** Area #5	234 ↓ 247	(See Note)
Core Set Area #6	250 ↓ 263	(See Note)

NOTE: The 12 memory locations in each core set area are used as those in Core Set Area # 0 shown above.

Figure 2-1. Executive's Core Set List

	Memory Address	Use
VAC Area #1	0431 0432 ↓ 0504	VAC1USE - Used to indicate if VAC area is in use.
VAC Area #2	0505 ↓ 0560	VAC2USE
VAC Area #3	0561 1 0634	VAC3USE
VAC Area #4	0635 1 0710	VAC4USE
VAC Area #5	0711 ↓ 0764	VAC5USE

Figure 2-2. Executive's Vac Areas

Whenever a job doesn't require a large amount of storage capacity and its requirements are satisfied by the storage provided by the core set area, the NOVAC subroutine is used to schedule the job. This subroutine places the job on the core set list, thereby scheduling the job and reserving a core set area for use by the job.

In order for a job to be scheduled, the job or task desiring to schedule the job supplies either of these two subroutines, FINDVAC or NOVAC, with the priority number and the starting address of the job to be scheduled. The job or task scheduling the job loads the Accumulator register of the computer with the priority number of the job prior to transferring control to the Executive's FINDVAC or NOVAC routines, The two memory registers immediately following the instruction which transfers control to either of these routines of the Executive, contains the complete starting address of the job.

The first thing done by either of these routines, as shown by the flow chart in Figure 2-3, is to temporarily store the priority number contained in the Accumulator register. Next the routine temporarily stores the complete address of the job being scheduled which is contained in the two memory registers following the transfer of control instruction which routed control to either FINDVAC or NOVAC. The complete address of the exit point of the scheduling routine is temporarily stored *so* control can be returned to the job or task at this point after completing the scheduling.

If the NOVAC subroutine is being used, the core set areas are scanned to find one which is not in use. This is accomplished by "looking" at the contents of the priority register in each of the core set areas. The priority register contains the priority number of the job for which the core set area is reserved if it is reserved. If the core set is not reserved for a job, the priority register will contain a *-0*.

If an unreserved core set area is not found, the Program Caution indicator is lighted. Also, the information for the display of VERB *05*, NOUN 31 and 01202 in R1 is provlded to the T4RUPT routine. This Verb, Noun and failure number display means: (VERB 05) Display octal component 1, 2, 3 of (NOUN 31) FAIL REGISTER and (01202 in R1) EXECUTIVE OVERFLOW - NO CORE SETS AVAILABLE. After this is done, a TC TRAP condition is forced which causes the processing of the RESTART routine.

If the FINDVAC subroutine is being used, after storing the priority number, starting address and return address, the five VAC areas are scanned to find one which is available. When one is found available, it is reserved for the job and the address of the VAC area is stored. Then, the core set list is scanned to find an available core set work area as was previously discussed for the NOVAC subroutine. The remainder of the FINDVAC subroutine is identical to that of NOVAC except that the address of the reserved VAC area is also stored in the core set area.

If a VAC area is not found available when the VAC areas are scanned, the Program Caution indicator is illuminated. *Also,* the information for the display of VERB 05, NOUN 31 and 01201 in R1 is provided to the T4RUPT routine. The display in R1 of the DSKY (01201) means EXECUTIVE OVERFLOW - NO VAC AREAS AVAILABLE. After this is done, a TC TRAP condition is forced which causes the processing of the RESTART routine,

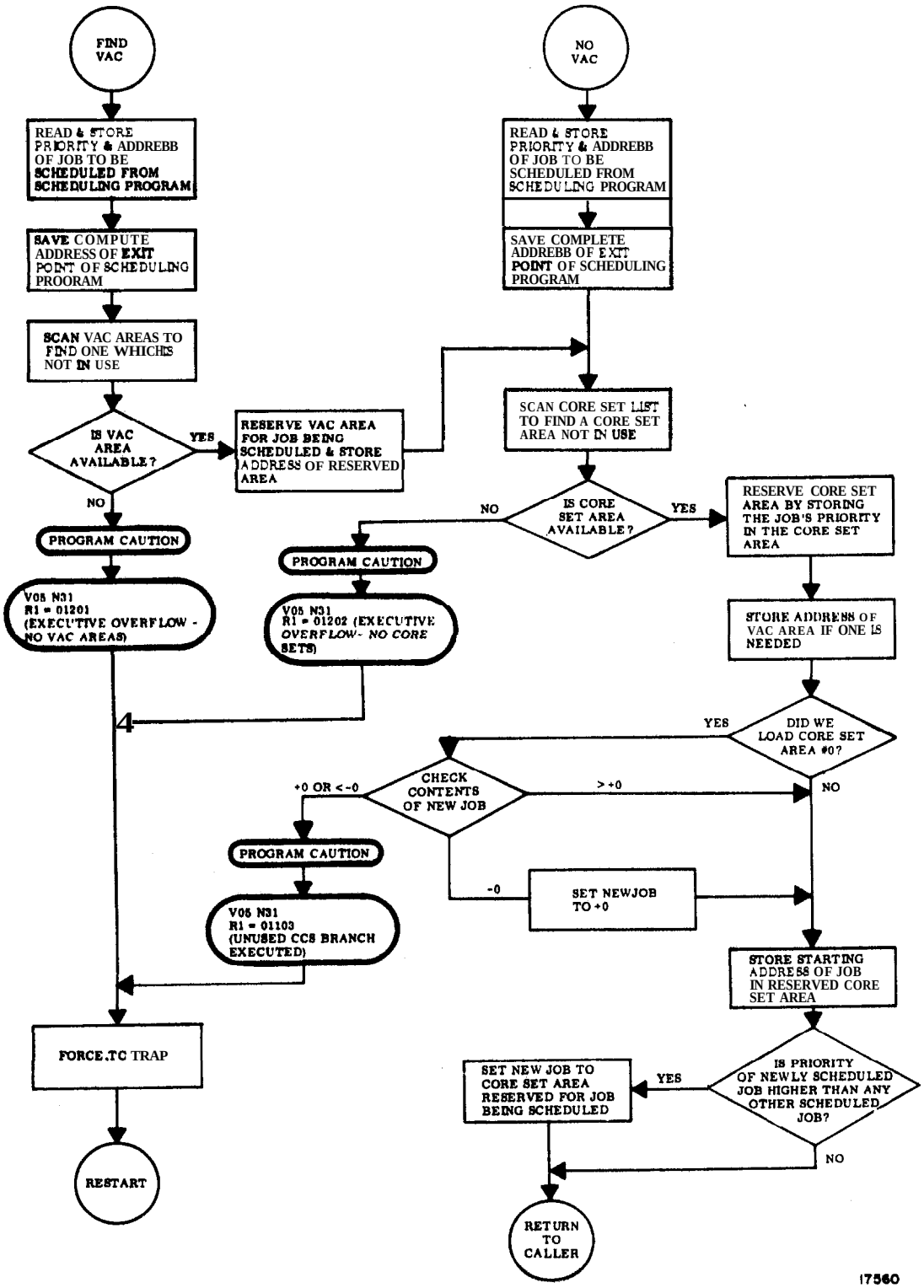


Figure 2-3. Executive's Findvac and Novac

After finding an unreserved core set, the priority number and associated VAC area address of the job being scheduled are stored in the core set area. Figure 2-1 shows which memory registers of the core set are used for the storage of the priority number and VAC area address.

If core set area #0 was loaded, a check is made of the contents of the NEWJOB register. If NEWJOB contains $+0$ or <-0, a program abort is initiated. The Program Caution indicator is illuminated, VERB *05,* NOUN 31 are displayed and 01103 (UNUSED CCS BRANCH EXECUTED) is displayed in Regieter 1 of the DSKY. A TC TRAP is forced and a RESTART occurs. If NEWJOB contained -0, it is now set to $+0$. If NEWJOB was $>+0$, it was not changed.

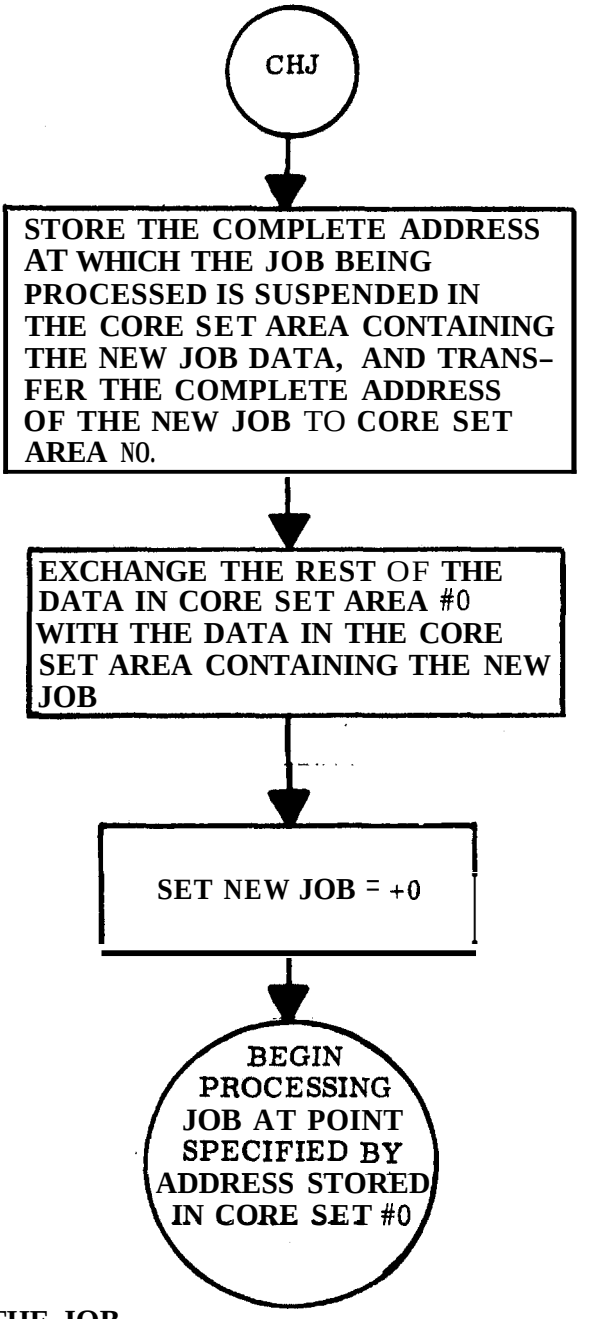
A check is made to see if the priority number of the job being scheduled is higher than the priority number of the job presently being processed. If the priority of the job being scheduled is higher than the one presently being processed, the number of the core set reserved for the job being scheduled is placed into a memory register called NEWJOB. If the priority of the job is equal to or less than that *of the* job presently being processed, the register NEWJOB *is* not changed. In either case, the stored return address of the job or task performing the scheduling is read and used to return control to the scheduling job or task.

The register NEWJOB is checked at least every 20 ms by every job. If the register contains + *0,* there has not been a job scheduled which has a higher priority than the one being processed, *so* the processing of the job presently being processed continues. If the contents of NEWJOB is greater than *+0,* a positive quantity, a job has been scheduled which has a priority higher than the job being processed. This positive quantity in NEWJOB is the core set number of the core set reserved for the higher priority job which was placed in NEWJOB by the Executive routine when the job was scheduled. Whenever this condition exists, control is transferred to the higher priority job under control of the Executive's CHANGE JOB subroutine.

Both of the subroutines, FINDVAC and NOVAC, are very similar, The only real difference is that FINDVAC reserves a VAC area while the NOVAC subroutine does not.

2.1.2 CHANGE **JOB SUBROUTINE.** The CHANGE JOB subroutine of the Executive provides the capability of changing the processing control from one job to another. The changingfrom one job to another is done whenever the job presently being processed is no longer the highest priority scheduled, active job. Control is routed to this subroutine whenever the NEWJOB register is interrogated and is found to contain a core set area number. The only time that NEWJOB will contain a core set area number is. when it *is* placed there by the FINDVAC or NOVAC subroutines if the job scheduled has a higher priority than the job presently being processed. At all other times, with the exception of Dummy Job and Self Check, NEWJOB will contain +0.

Figure 2-4 is a flow chart of the Change Job subroutine. This subroutine can be entered from a basic or an interpretive job. The first thing that is done is to store the complete address of the exit point of the job being suspended in the core set area associated with the job that is to be processed next. This address is stored *so* that later, when the job being suspended again becomes the highest priority scheduled job, processing can be resumed at the point where the processing was suspended. If the Change Job subroutine was entered from an interpretive job, the location address stored will be complemented.



NOTE: CORE SET AREA #0 IS USED FOR THE JOB CURRENTLY BEING PROCESSED.

17515

EXECUTIVE'S CHANGE JOB

Figure 2-4. Executive's Change Job

After this has been done, the Information stored in core set area *#0,* associated with the job being suspended, is exchanged with the information contained in the core set area specified by the number contained in NEWJOB. NEWJOB contains the relative address of the core set where the highest priority job's information is stored. By moving this information to core set #0, the processing of this job is enabled.

After the data is exchanged, NEWJOB is set to +0. Then, the starting or resumption address (whichever is applicable) of the highest priority scheduled job is read from core set area #0 and is used to transfer control to the job. If the job is interpretive, the location address will be complemented prior to transferring control.

2.1.3 END OF JOB, JOB SLEEP, AND PRIORITY CHANGE SUBROUTINES. Since a major portion of these subroutines is identical, the subroutines are presented together.

The **END** OF **JOB** subroutine is used to remove a job from Executive consideration. This subroutine is **"called"** by the job which is to be terminated. Therefore, the job must be processed at the time it is terminated.

The **JOBSLEEP** subroutine is used to deactivate a job or willfully suspend the processing of the job. This subroutine is used by a job to deactivate itself or put itself to sleep whenever it must wait for data, use of a piece of equipment or for a particular condition to exist. When a job is put to sleep, the processing of lower priority scheduled jobs can be accomplished.

The **PRIORITY CHANGE** subroutine is used to change the priority of the job presently under execution. This subroutine is used by a job to change its own priority. This job will return to the caller as soon as its priority is again the highest.

The flow diagram for these three subroutines is shown in Figure **2-5.** If the **END** OF JOB subroutine is used to terminate a job, the job is removed from the core set list by setting the priority register of core set area #0 to negative zero. By doing this, the job is not only removed from the scheduling list, but this makes the core set area available for use by other jobs. After making the core set area available, a check is made to see if a **VAC** area was used by the job being terminated. If a VAC area was used, it is made available for use by other **jobs** by setting the address of the VAC **USE** register in that **VAC USE** register. **This** is an indication that **VAC** area **is** unreserved.

The remaining core set areas are scanned to find the highest priority, nonsleeping job. After the core set areas are scanned, the highest active priority is checked to see **if** this job is **DUMMY JOB. If** it **is,** control is transferred **to DUMMY JOB.**

After the check **is** made for **DUMMY JOB,** an additional test is made to see if a job change is required. A job change will be required when the entry to this subroutine was from the **END OF JOB** subroutine subroutine and **DUMMY** JOB was not highest priority.

Control **is** transferred to the **CHANGE** JOB subroutine with the relative address of the core set area which contains the highest priority active **job.** The **CHANGE** JOB subroutine then uses this information to exchange the information in core set #0 with the appropriate core set area and transfers control to the highest priority job.

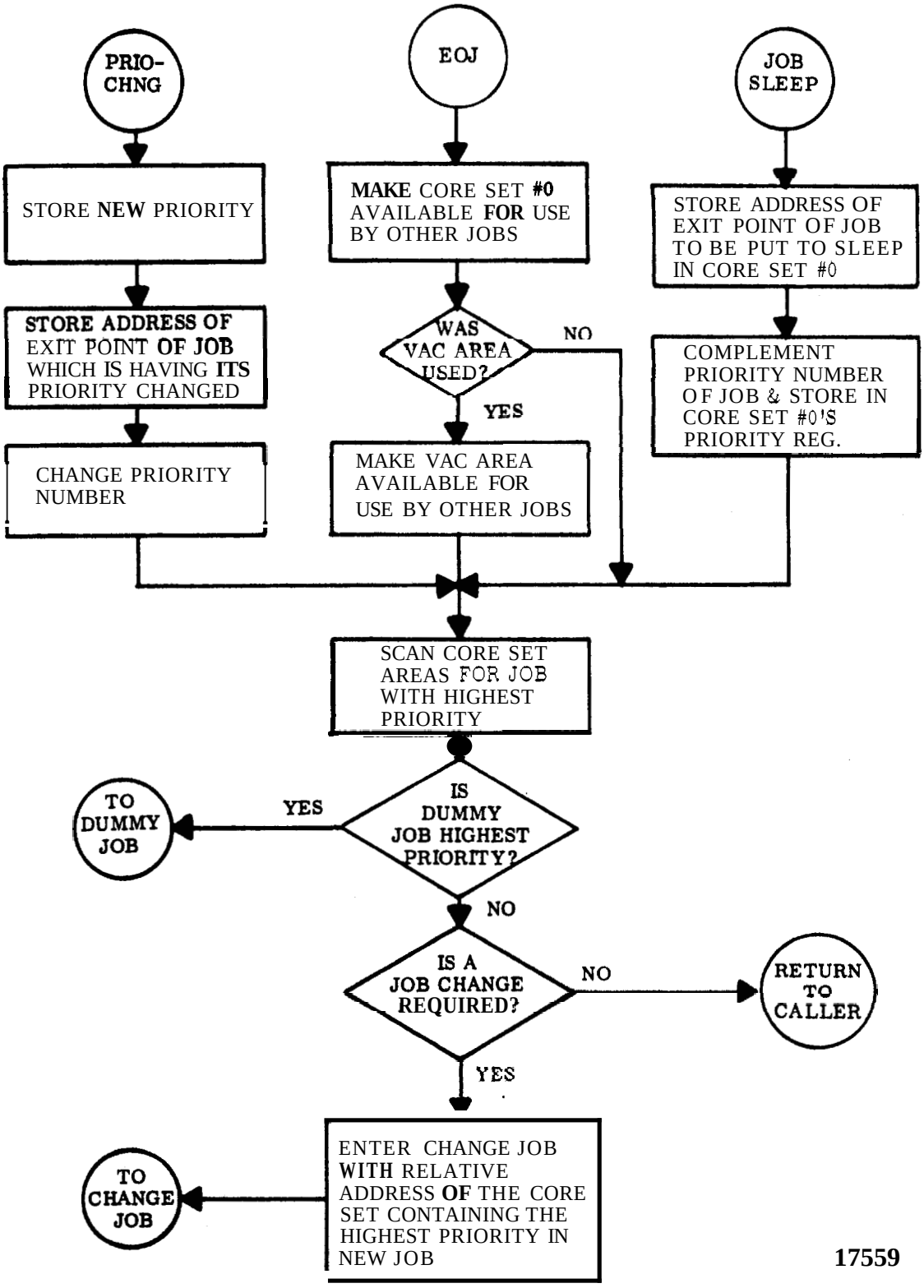


Figure 2-5. Executive's Priority Change, End of Job and Job Sleep

The JOBSLEEP subroutine, see Figure 2-5, stores the complete address of the point in the job where it should be awakened. After this has been accomplished, the priority number stored in core set area #0, (which is the priority of the job being put to sleep) is complemented. This causes the contents of the register to become a negative quantity which signifies a sleeping job. Note that the job is still scheduled on the core set list but it has been put into an inactive state by complementing its priority register.

When the job has been put to sleep, the remaining core set areas are scanned to find the highest priority active job. After the core set areas are scanned, a check is made to determine if the highest priority active job is DUMMY JOB. If it is, control is transferred to DUMMY JOB. If DUMMY JOB is not the highest priority active job, a test is made to determine if a job change will be required. A job change will be required when the entry to this subroutine was JOBSLEEP.

Control is transferred to the CHANGE JOB subroutine with the relative address of the core set area's highest priority, active job. The CHANGE JOB subroutine again exchanges the information of the iobs contained in the core set areas *so* that the information for the job to be performed is in core set area #0. Then control is transferred to the job using the address stored in core set area #0.

The PRIOCHNG (Priority Change) subroutine, also shown in Figure 2-5, stores the new priority to be assigned to the job being processed. After this has been accomplished, the complete address of the point that this job was at the time control was transferred to PRIOCHNG is stored. The priority number contained in core set area #0 is now changed to the new priority.

The remaining core set areas are now scanned to find the highest priority active job. After the scanning of the core set areas has been accomplished, a check is made to see if DUMMY JOB is the highest active priority. When PRIOCHNG subroutine is used, DUMMY JOB will not be the highest priority active job.

A check is now made to see if a job change is necessary. If the new priority assigned by the calling job to itself is still the highest, control is transferred to the caller. If it is not the highest, a job change will be required. The CHANGE JOB subroutine will exchange the information in the core set area which has its relative address in NEWJOB with the information in core eet area #0. Control is transferred to the new highest priority active job using the address stored in core set area #0.

2.1.4 DUMMY JOB SUBROUTINE. (See figure 2-6) The Dummy Job subroutine provides the computer with something to do if no other jobs require processing. It is performed under control of the Executive routine, is always scheduled to be processed, and has the lowest priority of any job. Therefore, any time there are no other active jobs scheduled on the job list, DUMMY JOB is processed.

Whenever Dummy Job is processed, the COMPUTER ACTIVITY indicator is not illuminated. Any time another job is being processed under control of the Executive routine, the indicator is illuminated. This indicator is extinguished whenever the DUMMY JOB is entered and is illuminated when the DUMMY JOB is left.

The flow diagram of the Dummy Job *is* shown in figure 2-6. Control is transferred to thie subroutine through the Executive Change Job subroutine when the Dummy Job's priority is the "highest" of all scheduled jobs. When control is routed to Dummy Job, the COMPUTER ACTIVITY indicator is extinguished by setting bit 2 of output channel 11 to a binary *0*.

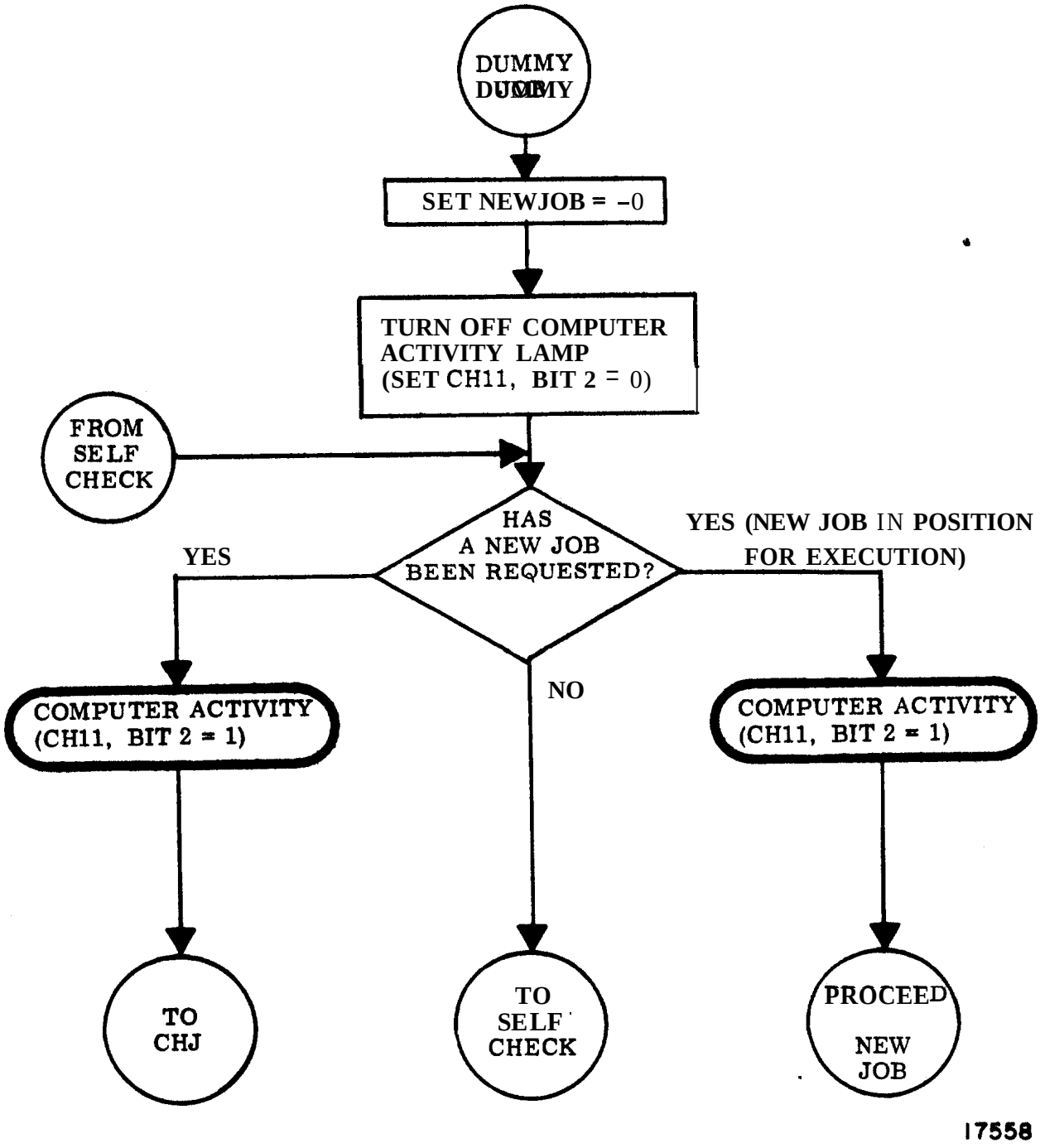


Figure 2-6. Executive's Dummy Job

After servicing the COMPUTER ACTIVITY indicator, a check is made to determine if a new job of higher priority has been scheduled. This is accomplished by checking the contents of the NEWJOB register. The NEWJOB register is set to the relative address of the core set area reserved for a scheduled job if its priority is higher than the job presently being processed. If NEWJOB contains a core set number greater than +0, the COMPUTER ACTIVITY indicator is illuminated by setting bit 2 of output channel 11 to a binary one. Control is then routed to the Change Job subroutine which routes control to the highest priority job.

If the NEWJOB register contains a -0, control is transferred to the self check routine. Periodically during the self check routine, the NEWJOB register is checked to see if a job with a priority higher than Dummy Job is to be performed.

The self check routine has the capability to insert a +0 into NEWJOB and to set up a special test as a part of the self test routine. In this special test case, the COMPUTER ACTIVITY indicator is illuminated and the address of the new job which had previously been inserted into the A and L registers is inserted into the Z and BB registers, respectively.

2.1.5 JOB WAKE SUBROUTINE. The JOB WAKE subroutine is used to wake up or to reactivate a sleeping or deactivated job. A job or task other than the sleeping job must awaken the sleeping job. The job or task wishing to awaken a job must furnish the JOB WAKE subroutine with the awakening address of the sleeping job. This address was stored in the core set area of the sleeping job when it put itself to sleep. The awakening address supplied to the JOB WAKE subroutine is used to find the sleeping job.

The flow chart of the JOB WAKE subroutine is shown in Figure 2-7. On entry to this routine, the awakening address is available and is stored temporarily for use in this subroutine. After storing this address, the complete address of the return point of the job or task using this routine is stored. This address is used to return control to the job or task after awakening the job. When this has been accomplished, a scan of the job areas is initiated to find a sleeping job. When a sleeping job is found, the awakening address stored in the core set area is checked against the one supplied by the job or task using the JOB WAKE subroutine. If the job is not the correct job, the scan continues. If the sleeping job is not found, control is returned to the job or task which called for the use of the subroutine. However, if the sleeping job is found, the priority number is complemented, thereby reactivating or awakening the job.

The rest of the flow chart is identical to the end of NOVAC FINDVAC.

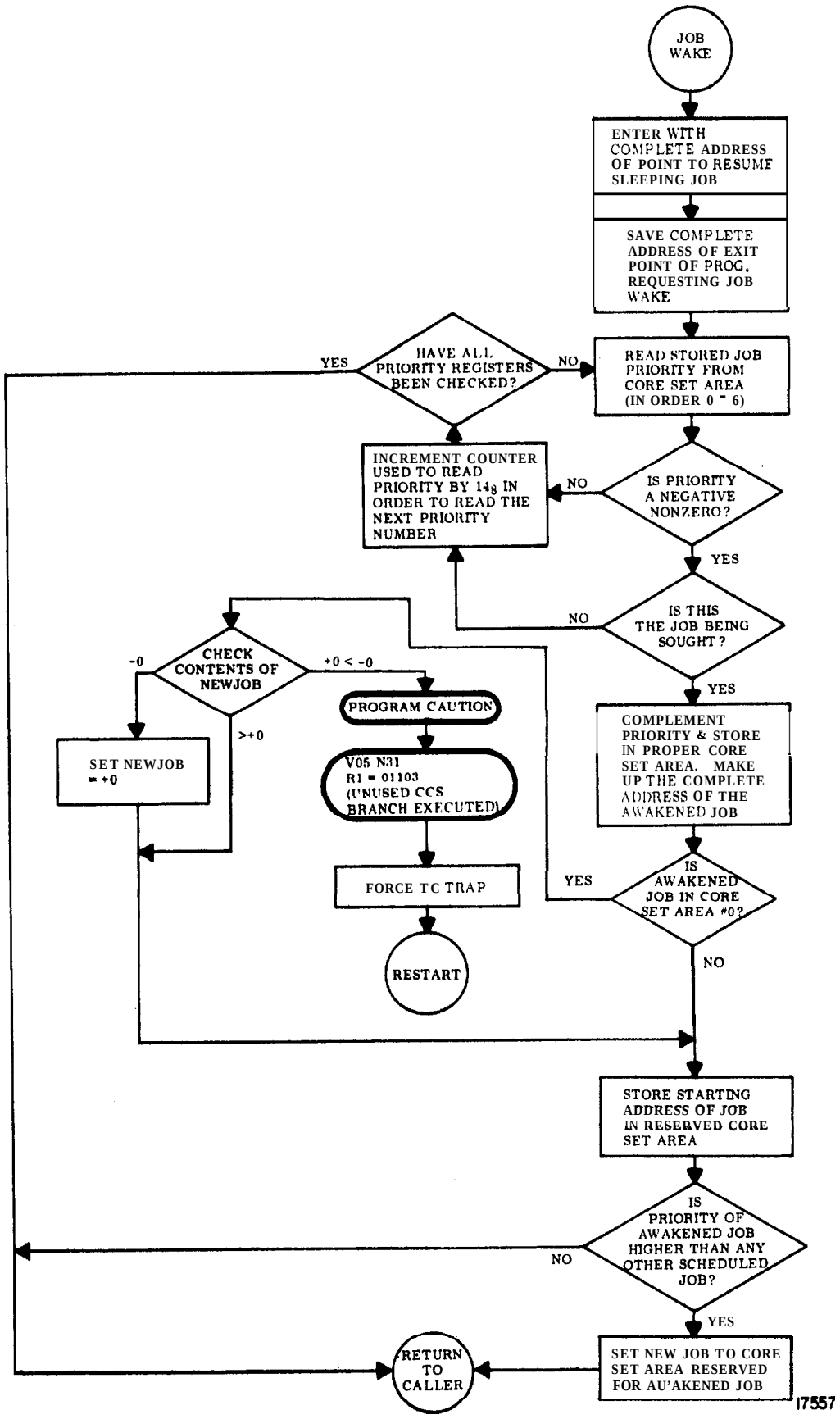


Figure 2-7. Executive's Job Wake

2.2 WAITLIST ROUTINE

The Waitlist routine performs a scheduling function for processing required at specific times within the next two minutes after scheduling occurs. This time dependent-processing is referred to as a TASK as opposed to jobs which are processed according to the priorities assigned to the job.

Any job or task can call upon the Waitlist routine to schedule a task. The scheduling is accomplished by the job or task supplying to the Waitlist the time from the present time that the task should be executed and the starting address of the task. With this information available, the control of the computer is transferred *to* the Waitlist routine which performs the scheduling. After scheduling the task, control is returned to the job or task which called for the use of the Waitlist routine.

The computer programming provides the capability of scheduling up to nine tasks at any one time. The Waitlist routine, therefore, maintains two lists as shown in Figure 2-8. One of the lists (LIST 1) has nine entries, including the TIME 3 counter, and is used to store the time values for the tasks. The other list (LIST 2) stores the task addresses or starting addresses of the scheduled tasks and has eighteen entries. The various tasks (their times and addresses) which are scheduled are maintained on these two lists in chronological order.

The overflow of the TIME 3 counter is used to initiate T3RUPT, a program interrupt. The overflow implies that it is time to process the task address in LIST 2 and LIST 2 + 1.

The TIME 3 counter, as shown in Figure 2-8, is the topmost entry of LIST 1. It will at all times contain the time remaining till it is time to process the task which should be processed next, of those tasks which are scheduled. This time value in the TIME 3 counter is actually OVERFLOW minus TIME TILL TASK EXECUTION. If it is assumed that the TIME till TASK EXECUTION, or Δ T is .2 minutes, the contents of the TIME 3 counter will be equivalent to overflow -. 2 minutes. (NOTE: Remember that tasks are executed when the TIME 3 counter overflows).

The TIME 3 counter is then incremented towards overflow every 10 ms. When it does overflow, it is time *to* execute the task whose starting address is stored in the corresponding LIST 2 entry.

The time values in the remaining entries of LIST 1 are basically stored as the complement of the time between a particular task and the immediately preceding task. The actual values have 00001_8 added *to* them *so* that when they are processed into the TIME 3 counter, the overflow will occur at the correct time. The time entry in LST 1 position of LIST 1 is expressed as:

$$\Delta T_{LST\ 1} = -(\Delta T_2 - \Delta T_1) + 00001_8$$

The LST 1 + 1 time entry is expressed as:

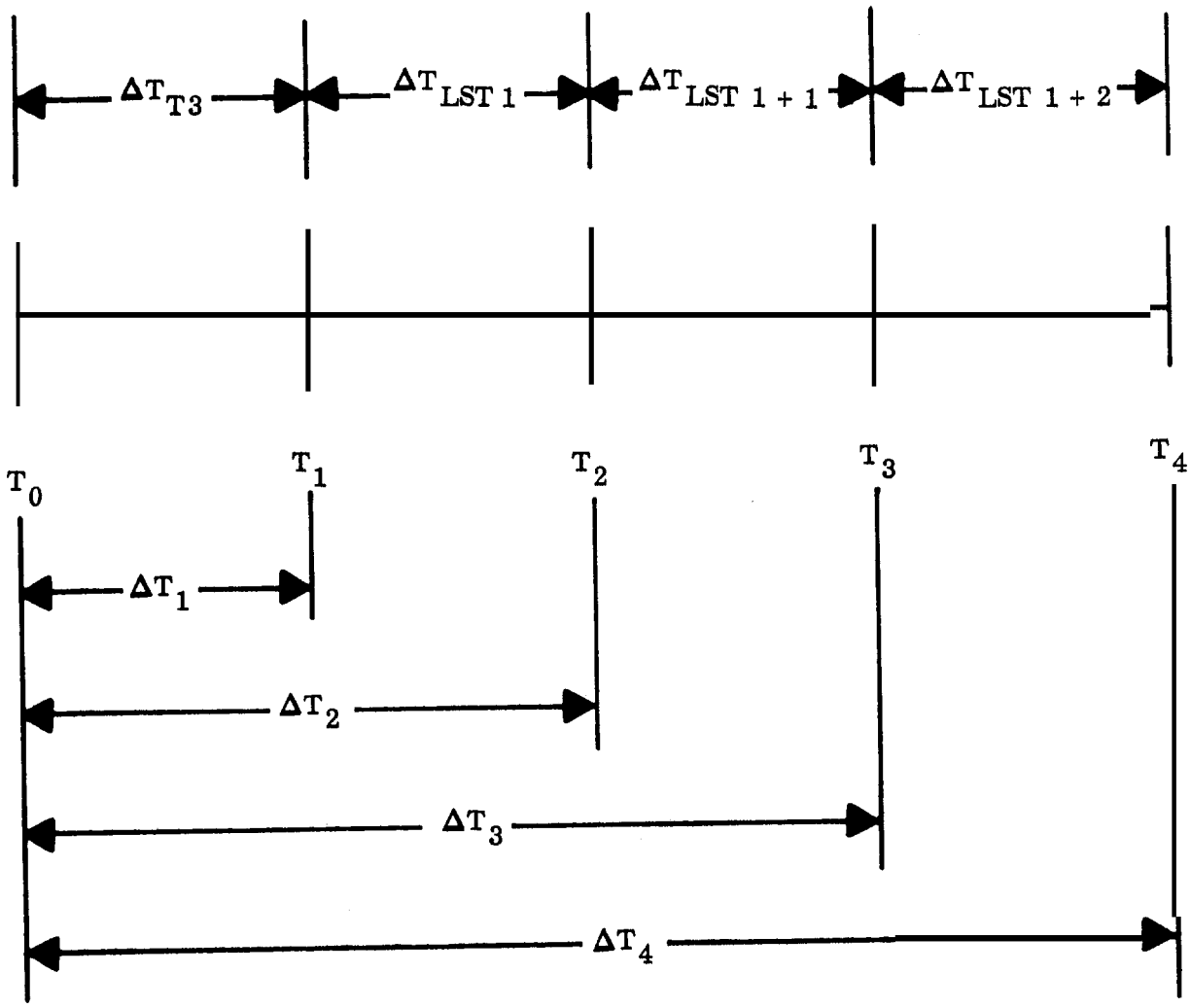
$$\Delta T_{LST\ 1+1} = -(AT, -\Delta T_2) + 00001_8$$

where $A\,T_{LST\ 1}$ and $A\,T_{LST\ 1+1}$ are the values stored in the corresponding LIST 1 entries and ΔT_1, ΔT_2, and AT3 are the actual times till the corresponding task is to be executed. Figure 2-9 shows this on a time line diagram.

TASK TIMES - LIST 1		TASK ADDRESSES - LIST 2 (2 CADR)			
		GEN ADDR		BBCON	
TIME 3	0026	LST 2	E3, 1410	LST 2 + 1	E3, 1411
LST 1	E3, 1400	LST 2 + 2	E3, 1412	LST 2 + 3	E3, 1413
LST 1 + 1	E3, 1401	LST 2 + 4	E3, 1414	LST 2 + 5	E3, 1415
LST 1 + 2	E3, 1402	LST 2 + 6	E3, 1416	LST 2 + 7	E3, 1417
LST 1 + 3	E3, 1403	LST 2 + 8	E3, 1420	LST 2 + 9	E3, 1421
LST 1 + 4	E3, 1404	LST 2 + 10	E3, 1422	LST 2 + 11	E3, 1423
LST 1 + 5	E3, 1405	LST 2 + 12	E3, 1424	LST 2 + 13	E3, 1425
LST 1 + 6	E3, 1406	LST 2 + 14	E3, 1426	LST 2 + 15	E3, 1427
LST 1 + 7	E3, 1407	LST 2 + 16	E3, 1430	LST 2 + 17	E3, 1431

NOTE:
1. TIME 3 is associated with LST 2 and LST 2 + 1, LST 1 is associated with LST 2 + 2 and LST 2 + 3, etc.
2. LST 1 + 7 should contain the time for ENDTASK and LST 2 + 16 and LST 2 + 17 should contain the starting address of ENDTASK. The Waitlist routine will transfer control to the abort routine which initiates the failure displays and the restart routine when the next entry is made if ENDTASK is not as indicated above.
3. All scheduled tasks are tabulated in chronological order in LIST 1 and LIST 2. If two tasks are scheduled to be done at the same time, the task scheduled first will be done first.
4. The eraeable memory address of each register used as a part of the list is shown following the register name as 0026 or E3, 1400, etc. These numbers are in octal.

Figure 2-8. Waitlist's Waiting List



T_O = PRESENT TIME

$T_{1,\ 2,\ 3,\ 4}$ = TIME AT WHICH TASKS ARE PROCESSED

$\Delta T_{1,\ 2,\ 3,\ 4}$ = TIME FROM PRESENT TO PROCESS TASK

ΔT_{T3} = TIME VALUE FOR TIME 3 COUNTER (OVERFLOW - ΔT_1)

$\Delta T_{LST\ 1}$ = TIME VALUE FOR LIST 1 = $[-(\Delta T_2 - \Delta T_1) + 00001_8]$

$\Delta T_{LST\ 1+1}$ = TIME VALUE FOR LIST 1 + 1 = $[-(\Delta T_3 - AT,) + 00001_8]$

Figure 2-9. Time Values Stored in List 1

By storing the time values in this manner in **LIST 1,** the values can be directly moved up the list. When a task time is moved from **LST 1** into the **TIME 3** counter, POXMAS **(377778)** and the contents of the **TIME 3** counter are added to the contents of LST 1 and the result is inserted into **TIME 3,** Whenever a task **is** to be processed next in chronological order, the time value inserted into the **TIME 3** counter specifies OVERFLOW minus the time remaining till the task is to be processed. OVERFLOW is **400008.** No additional computations must be performed on the time value prior to inserting it into the **TIME 3** counter.

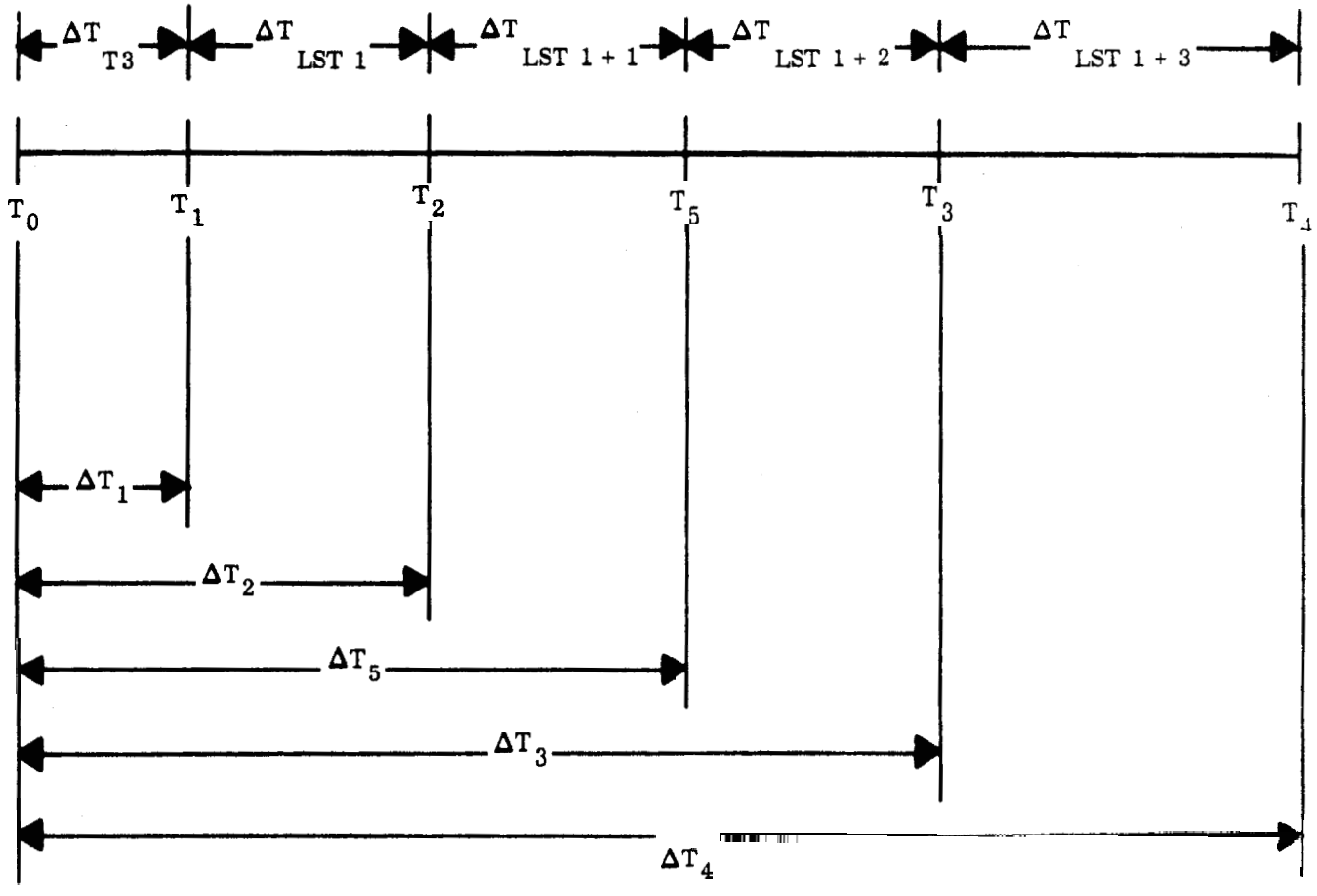
When a new task **is** being scheduled, the time list, **LIST 1,** must be searched in order to find the appropriate chronological position to insert it on the list. When the position is found, the remaining tasks which follow chronologically must be displaced one position on the list. Also, the first following time value must be modified so that it will still be executed at the proper time. **An** example of this is shown in Figure **2-10** where a new task is scheduled to be processed at time T_5. T_5 is greater than T_2 but less than T_3. Note that the time values for task **3** and **4** have been displaced one position on the list and that a new time value has been calculated for task **3.** The task addresses corresponding to tasks **3** and **4** are also displaced one position in **LIST 2** with the task address of task **5** replacing the task address for task **3.**

A simplified flow chart for the Waitlist routine is shown in Figure **2-11.** The first action performed by this routine is to temporarily store the task address and the time from now till the task is to be performed which is supplied by the job or task which called for the use of the Waitlist routine. Then, the complete address **of** the calling **job** or task **is** stored *so* that control can be returned to this job or task after the scheduling has been completed.

After this has been accomplished, the **TIME 3** counter ***is*** read and a check is made to determine if the **TIME 3** counter has overflowed. Another check is made to determine if the time till new task **is** greater than the time till **T3** counter overflow. If not, a new value **is** calculated and inserted in the **TIME *3*** counter. The former contents of the TIME **3** counter are appropriately modified and inserted in LST **1** of LIST **1.** All the remaining task times are displaced down one position in LIST **1.** The new task addresses are inserted in LST **2** and LST **2** + **1** and the remaining task addresses are displaced two registers down the address list (LIST **2).**

If the time till new task **is** less than time till TIME **3** counter overflow, the remaining task times in **LIST 1** are compared to the new task time. This **is** done in chronological order until the proper position **is** determined. When the position **in** determined, the task times following the new task time chronologically are displaced one position after the new task time expressed as **a** Δ T between it and the sum **of** the preceding **A T** task times **is** ineerted into the located position in LIST **1.** A new **A** T value **is** calculated for the first displaced task following the new task which was inserted on the **list.** After **LIST 1 is** rearranged, the addresses for the new task **is** inserted **in** ita proper location **in LIST 2** and **the** remnining contents of LIST **2 are** shifted two registers down the list.

In either case, as discussed above, after placing the new task on the two lists, a check is made to make sure that too many tasks have not been scheduled. **This** is accomplished by checking to see that the complement of the **ENDTASK** task address was located in **LST 2** + **16** and **LST 2** + **17. ENDTASK** is a task which gives the computer something to do when nothing else has been time scheduled. If the complement of **ENDTASK was** in these positions, the stored complete return **address of** the scheduling task or **job** is used to return control to the job or task, If it was not present in these positions, too many tasks have been scheduled and



NOTE: Since ΔT_5, which was supplied by the scheduling routine, is larger than the present ΔT_2 and smaller than the present ΔT_3, the contents of LST 1 + 1 is set to - (AT5 - ΔT_2) + 00001_8 and - (AT3 - ΔT_5) + 00001_8 is set into LST 1 + 2. The former contents of LST 1 + 2 and the other LIST 1 registers are moved down one register.

Figure 2-10. Maintaining Chronological Waiting List

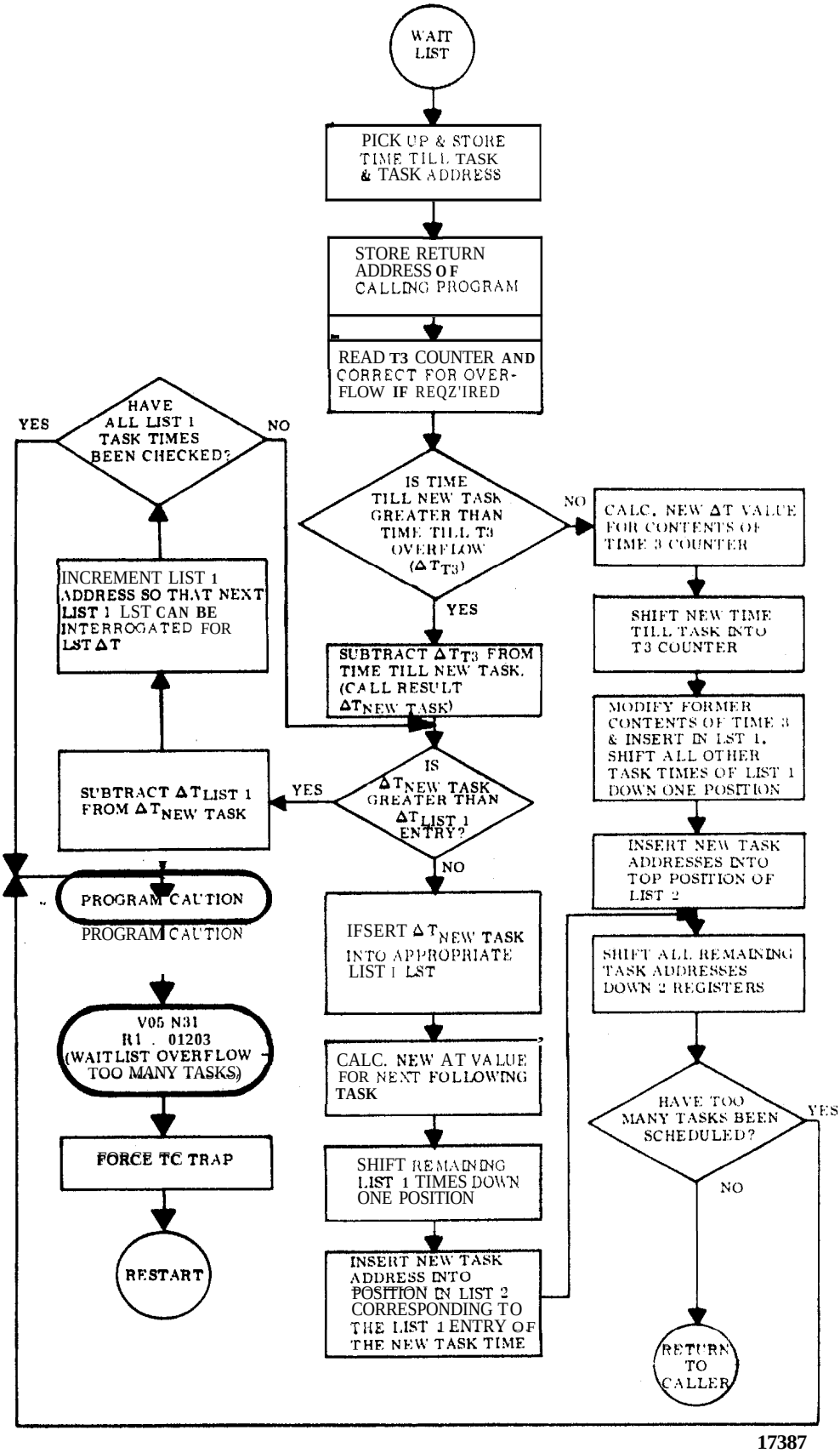


Figure 2-11. Waitlist

the **PROGRAM CAUTION** indicator **is** illuminated. Also, VERB **05,** NOUN **31** are displayed along with **01203** in the DSKY register **R1.** The number **01203** indicates that there has been a Waitlist Overflow or that too many tasks have been scheduled. Then a TC TRAP condition **is** forced causing the execution of the **RESTART** routine,

2.3 TIME 3 PROGRAM INTERRUPT ROUTINE (T3RUPT)

The T3RUPT routine **is** initiated by the overflow state of the **TIME 3** through a circuit forcing function. The **TIME 3** counter **is** set to overflow at specific times according to the information supplied to, and the scheduling performed by, the Waitlist routine. Therefore, whenever the TIME **3** counter overflows, it is time to process a particular task.

The T3RUPT routine **is** used to initiate the processing of the task which was scheduled to be processed at the time of the **TIME 3** counter overflow. The routine also moves **all** the task times and addresses up one register position in **LIST 1** and two registers in LIST **2.** The result of this **is** the loading of the **TIME 3** counter with a new time value to increment towards overflow.

Figure **2-12** contains the flow chart for the T3RUPT routine. Entry to this routine is forced whenever the **TIME 3** counter overflows through the program interrupt priority control circuit of the computer. The first action performed by this routine is to enter a value of 57777_8 on the bottom of **LIST 1. This** is the time value for the **ENDTASK** task which corresponds to a time to overflow, when inserted into the **TIME 3** counter, of approximately 82 seconds. After this has been accomplished, all of the time values of LIST **1** are moved up one position in the list. The contents of the **TIME 3** counter are added to the time value from LST **1** which is to be inserted into *the* **TIME 3** counter and **POSMAX (37777$_8$).** The contents of the TIME **3** counter **is** added to the contents to be inserted into TIME **3** counter because the counter could have been incremented since overflow because of delays in initiating the processing caused by the overflow condition. **This** could be caused by a combination of inhibiting interrupts and the scheduling of more than one task to be processed at a particular time. By adding the contents of the **TIME 3** counter to the next time value, the correct times are maintained for the remaining tasks if the delay in processing occurs. Note **that** if the counter has not been incremented since overflow, this addition has no effect, POSMAX **is** added to prepare the value for overflow when inserted into the **TIME 3** counter.

A flag is set *so* that if the value inserted in **TIME 3** was in overflow, the overflow wlll not be lost during the following operation. **This is** accomplished by setting **RUPTAGN** to **+1** if the value inserted into TIME **3 was** in overflow. **If** the value set into **TIME 3** was not in overflow, **RUPTAGN** will be set to *-0.*

Having completed the manipulations on LIST **1, LIST 2** must be serviced. The complement of the contents of **ENDTASK** is inserted in **LST 2 + 16** and **LST 2 + 17.** After this has been accomplished, all of the task addresses stored in **LIST 2** are moved up two registers. The address moved out of **LST 2** and LST **2 + 1** registers, as a result of **this operation,** is used to transfer control to the desired task. When control is returned to **the** T3RUPT routine, a check is made to determine if the **TIME 3** counter is again in the overflow state by checking **RUPTAGN.** If the **TIME 3** counter has overflowed, control is routed to the beginning of the T3RUPT routine which **is** processed again. If it is not in the overflow state, control is returned to the job which was interrupted.

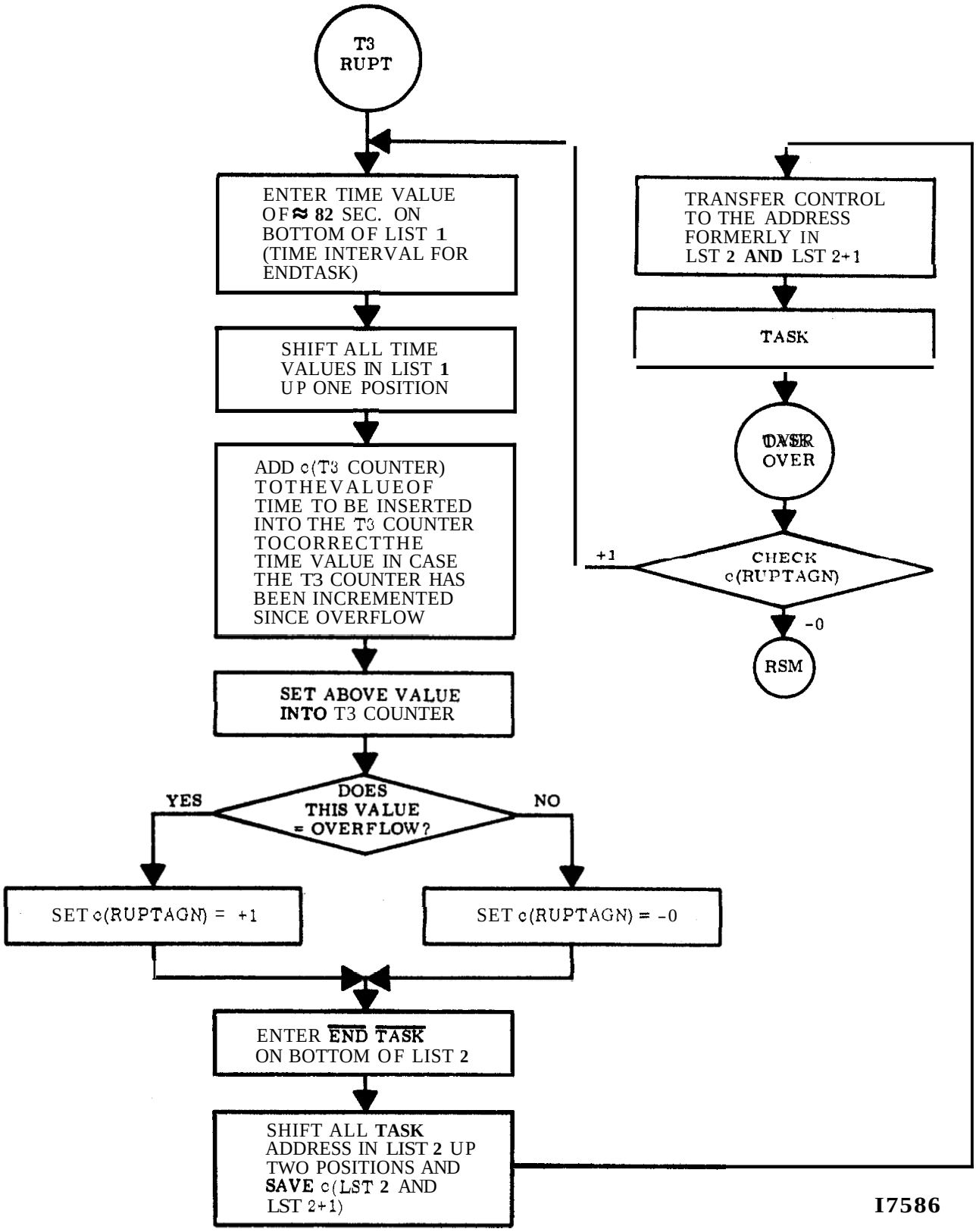


Figure 2-12, Time 3 Interrupt Routine

2.4 PHASE TABLE MAINTENANCE ROUTINE

The PHASE TABLE MAINTENANCE Routine consists of a group of subroutines which provide the initiation, termination, and progression through the mission or testing routines of the computer. The mission or testing routines are sometimes referred to as mission programs, testing programs or major modes. Throughout the description of the PHASE TABLE MAINTENANCE Routine, these routines will be termed major modes programs. The major mode programs supply the information required for the display of the program number on the DSKY's. Each of the major mode programs is assigned a program number which is displayed while the program is being processed.

Each of the major mode programs is divided into a number of different phases. The PHASE TABLE MAINTENANCE Routine maintains a table for phase numbers for the major mode programs. The phase numbers are used to control the routing and progression through a major mode program. *Also,* if a failure occurs and requires a restart, the phase numbers stored in the phase table are used to control the restarting of the major mode program at a particular phase of the program. The processing of the program would not necessarily begin at the beginning of the program nor at the phase specified by the phase number.

The phase table, where the phase numbers of the various major mode programs are stored, is actually maintained in duplicate. This is done to assure that the correct phase number is obtained if a failure occurs. The two copies of the table are called -PHASE and PHASE. -PHASE stores the complement of the phase number while PHASE stores the actual phase number. The phase tables consist of twelve registers.

2.4.1 PHASE CHANGE AND NEW PHASE SUBROUTINES. The PHASE CHANGE and NEW PHASE subroutines of the PHASE TABLE MAINTENANCE Routine are used by the major mode programs to change its phase number which is stored in duplicate in the phase tables or to initialize the phase tables for some major mode program operation. A flow chart of these subroutines is shown in figure 2-13. Except for the "lead-in" operation, the two subroutines are identical and perform the same function.

The PHASE CHANGE (PHASCHNG) subroutine is called by

```
L       TC      PHASCHNG
L+1     OCT     PPP GG
```

When transferring control to the PHASCHNG subroutine, the contents of the address in the Q register contain the phase number PPP in the three most significant octal digits. It also contains the group number in the two least significant digits. The contents of the Q register are masked to acquire the group number. This number is doubled and becomes the phase table relative address for use in indexing. The phase number is shifted two octal digits to the right and is stored in the A register.

The NEWPHASE subroutine is called by

```
L-1     CA      PPPPP
L       TC      NEWPHASE
L+1     OCT     OOOGG
```

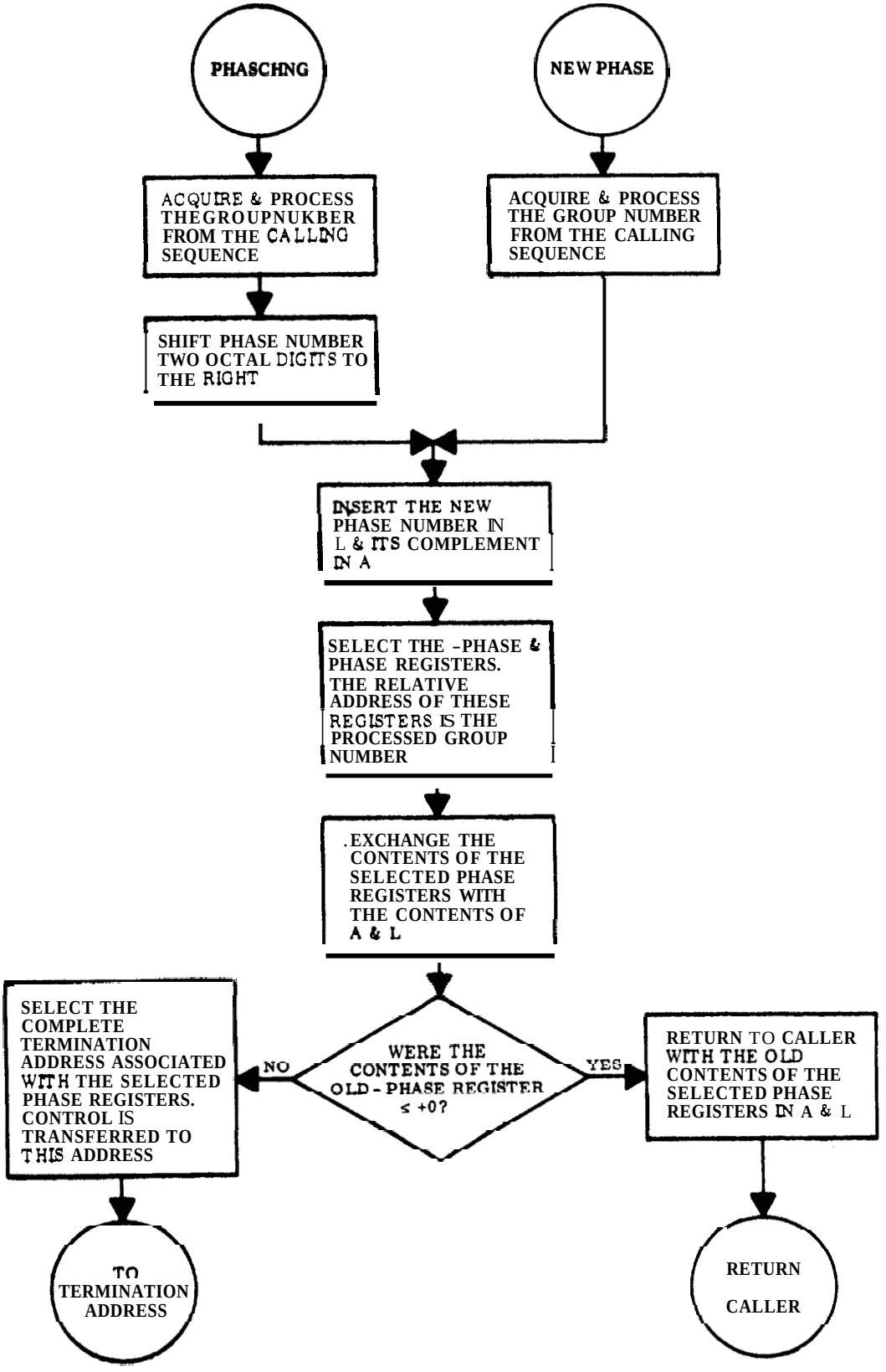


Figure 2-13. Phare Change and New Phase

Therefore, when control is transferred to NEWPHASE, the phase number is in the A register. The contents of the address in Q contains the group number. The group number is doubled and becomes the relative address of phase table.

From this point on, PHASCHNG and NEWPHASE are identical, The new phase number, which is in the A register is stored in the L register and the contents of the A register are complemented. The group number is used to index the desired phase table registers, a -PHASE and PHASE. The contents of the selected -PHASE and PHASE registers are exchanged with the contents of the A and L registers, respectively, The former contents of the -PHASE register are checked to determine if they were equal to or less than positive zero. If the answer is yes, the subroutines return control to the caller with the old contents of -PHASE and PHASE in the A and L registers, respectively. If the answer is no, control is transferred to the termination address associated with the selected phase table registers.

2.4.2 NEW MODE EXCHANGE SUBROUTINE. The NEW MODE EXCHANGE (NEWMODEX) Subroutine of the PHASE TABLE MAINTENANCE Routine is used by various major mode programs to set up the major mode program number which will be displayed on the DSKY by the T4RUPT routine. If the new major mode program number is the same as the present number, no change in the program number display is made by the T4RUPT routine. (See Figure 2-14.)

Control is transferred to NEWMODEX by the caller with the address of the major mode program number in the Q register. The program number is stored in MODREG. MODREG contains the present or new program number. The old contents of MODREG are compared to the new program number. If the two program numbers are the same, control is transferred to the caller. If the old and new program numbers are not in agreement, the new program number has to be set up for DSKY display. The relay code for the most significant of the two octal digits is acquired from RELTAB (Relay Code Table). The contents of DSPTAB+10D are saved and the relay code is inserted into bits 6 through 10. The relay code for the least significant digit is acquired from RELTAB and inserted into bits 1 through 5 of DSPTAB+10D. Control is returned to the calling routine. The T4RUPT routine will cause the program number on the DSKY to be changed to the new number.

2.4.3 CHECK MAJOR MODE SUBROUTINE. The CHECK MAJOR MODE subroutine of the PHASE TABLE MAINTENANCE Routine is used by various major mode programs to sample the program number of the major mode programs in process. The data is used for routing purposes by the major mode program which uses this subroutine. (See Figure 2-15.)

Control is transferred to the Check Major Mode (CHECK") subroutine with the address of the check number in the return address (Q) register. The check number is complemented and added to the contents of MODREG. The contents of MODREG will be equivalent to the program number displayed on the DSKY. If the contents of MODREG and the check number are in agreement, the contents of the accumulator will be negative zero. Branching to the next subroutine is dependent upon the contents of the accumulator and the contents of the two addresses following the address in Q. The exact action resulting from these two return points is dependent on the major mode program which called this subroutine.

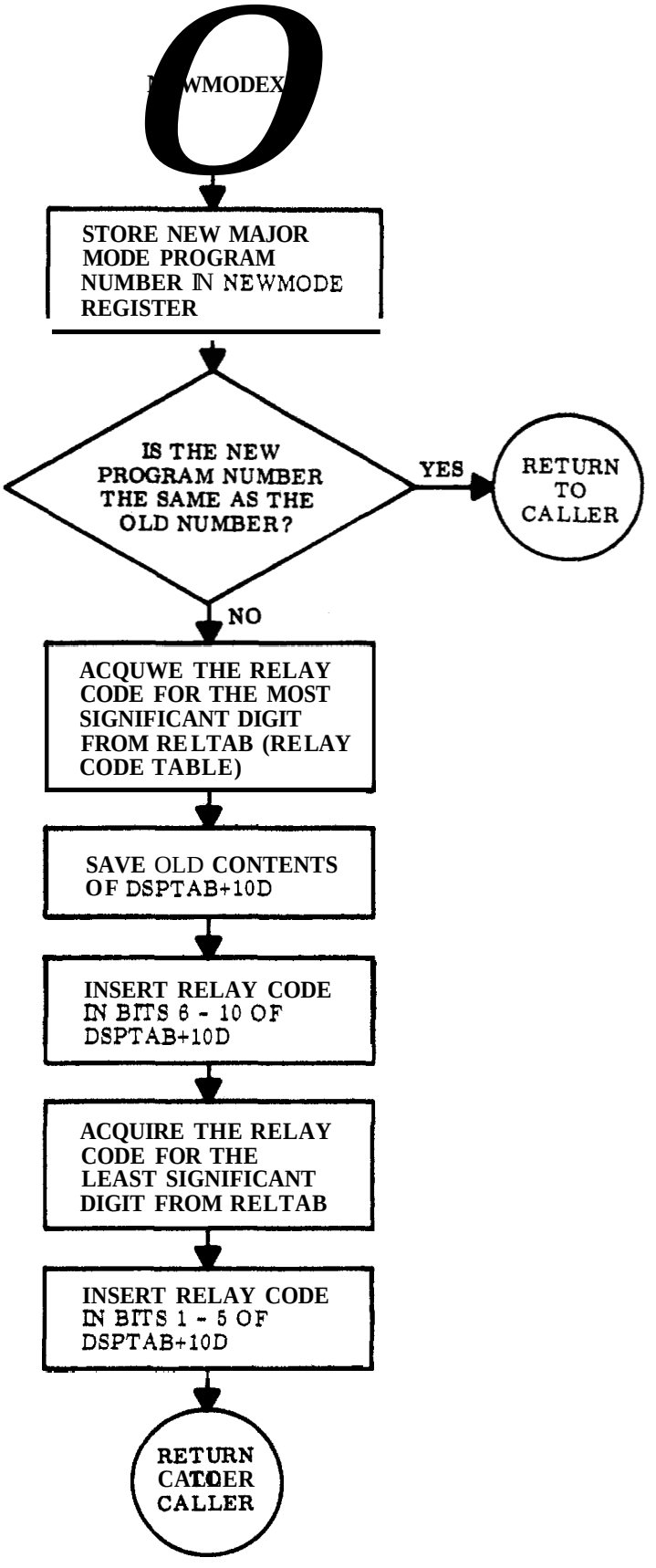


Figure 2-14. New Mode Exchange

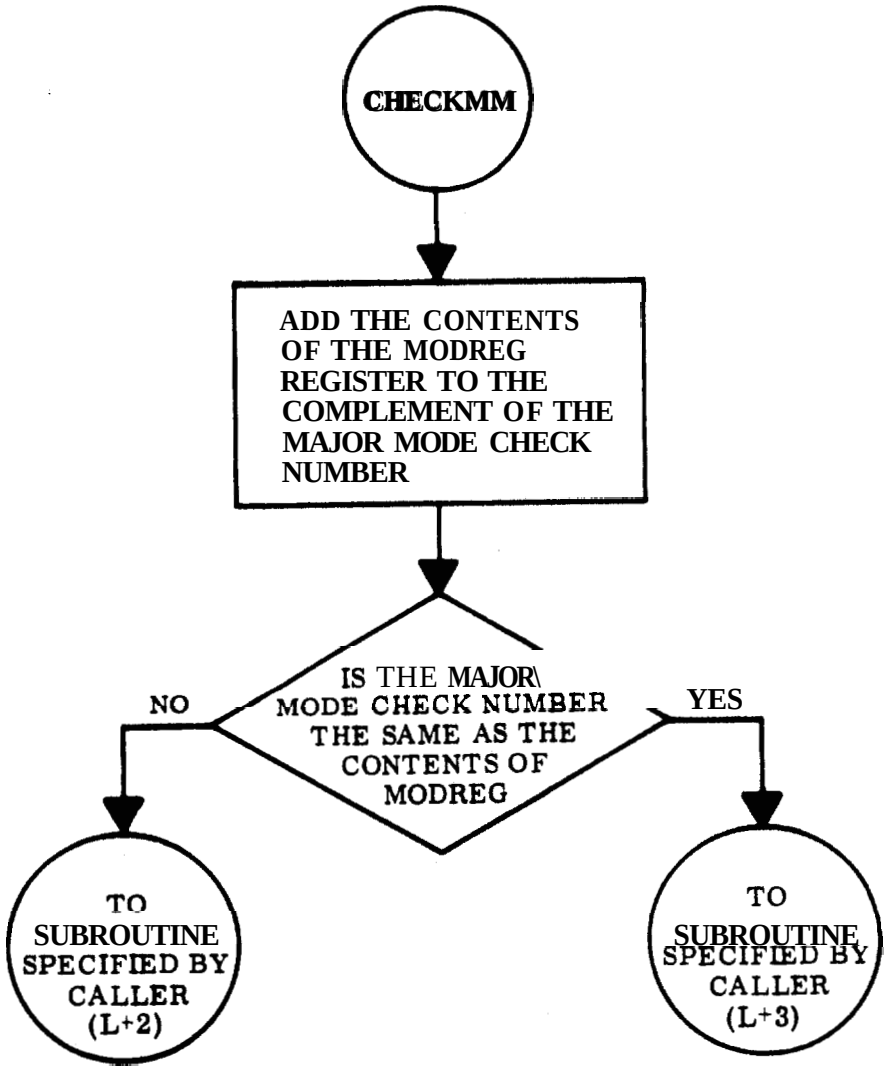


Figure 2-15. Check Major Mode

SECTION III

INPUT/OUTPUT CONTROL ROUTINES

INTRODUCTION

This section of the study guide presents the routines used to perform input and output functions of the **LGC.** The routines are ueed by most of the programs of the LGC to perform the required input and output functions. Through these routines, the LGC **is** capable of commanding spacecraft system modes, displaying and accepting information from the DSKY's and Radar providing for telemetry inputs and outputs, controlling the positions of the RR antenna and the stable member, and remaining cognizant of the **PGNCS** and other spacecraft system operatione.

3.1 TIME 4 COUNTER PROGRAM INTERRUPT ROUTINE (T4RUPT)

The TIME 4 COUNTER program interrupt routine (T4RUPT routine) is initiated whenever the TIME 4 counter overflows. Normally this time counter is set so that it will overflow every **120** ms. Everytime it overflows, the T4RUPT routine is initiated and one or more of the following functions are performed:

a. Updating the forward and lateral velocity meters and altitude meter.

b. Sampling and verification of the ISS mode of operation including turn-on.

c. Sampling and verification of the Radar mode of operation.

d. Monitoring the telemetry rates.

e. Sampling of malfunction indications from the ISS.

f. Control of the relays of the DSKY's for display of information, for commanding ISS, and other spacecraft modes, and for control of indicator panel'illumination.

g. Servicing the **RR** mode requests and RR CDU fail inbit.

h. Update the gimbal to pilot matrix.

All of the programs which desire to perform any of the functions listed, provide the information to the T4RUPT routine. The T4RUPT routine then uses the supplied information to perform the appropriate input or output function. In return, the T4RUPT routine furnishes information to the other routines or processing functions indicating the results of a desired action.

A deviation from the normal 120 ms rate at which the T4RUPT routine occurs is if the relays are manipulated. Whenever relays are driven, the **TIME 4** counter **is** set to overflow in 20 or **30** ms. This **is** done so that the driving of the relay coils is terminated after a sufficent amount of time for the relays to latch into the proper configuration. This also prevents ***ex***cessive heat build up and power consumption. After a **20** or **30** ms rupt has occurred, a 100 or 90 ms rupt **is** scheduled respectively, so that the nominal **120** ms interrupt rate is maintained.

Figure 3-1 *is* a general flow diagram of the T4RUPT routine. Following this a more detailed flow chart of the routine is presented. The general flow chart is presented to enable the student to obtain an overall knowledge of what is accomplished by this routine and in what order it is accomplished.

Referring *to* figure 3-1, entry is forced whenever the TIME 4 counter overflows. This occurs every 120 msec (with the exception of when DSKY or LMP relays have been driven). The display tables are serviced every 120 msec. This implies the capability of displaying new information or change in information every 120 msec, thus providing the operation "continuously" updated information.

After servicing the DSPTAB's, a check is made to determine the pass through the T4RUPT routine. This is determined by looking at a counter which is initially set to 7 and is decremented every time a 120 ms interrupt routine is processed. Therefore, the counter expresses eight states; 7, 6, 5, 4, 3, 2, 1 and 0. Whenever the counter goes to 0, it is set to 7. This counter is used to determine which of the 5 subroutines of the T4RUPT routine is to be processed for this interrupt.

Noting the numbers adjacent to the lines one sees that when the counter is in the 3 or 7 state the GP matrix is updated. If the counter is in state 0 the forward and lateral velocity meters along with the altitude meter receive updating. The counter in the 2 or *6* state routes servicing to the ISS failure, turn-on, and gimbal lock monitoring. Counter state 1 or 5 forces the T4RUPT into monitor the RR CDU fail inbit in addition to servicing RR moding and positioning. Counter state 4 forces an update of the forward and lateral velocity meters and an update of the altitude rate meter.

The following sections present detailed flow diagrams (figure 3-3 is located at the end of this chapter) illustrating how the program executes the T4RUPT routine,

3.1.1 T4RUPT LEAD IN, 20, 30 MSEC RUPT, SERVICE DSPTABS. The overflow of the TIME 4 counter forces the program interrupt T4. The first operation after transferring control to the T4RUPT program is to store the address of the exit point of the interrupted program to enable a return to the interrupted program at the completion of the specific T4RUPT routine, Setting the output channel 10 to zero removes the drive current to the DSKY relays. This is done regardless of whether DSKY relays were set or not on the previous pass through T4RUPT.

The contents of memory register T4LOC is checked to identify what type of interrupt this is. If it was either a 20 or 30 msec rupt approximate routing is accomplished to keep the total normal time equal to 120 msec. Thus for a 20 msec rupt, 100 msec is added prior to going through the normal T4 routing, for a 30 msec rupt, 90 sec are added in real time.

Then, the contents of memory register, DSRUPTSW (DISPLAY RUPT), is checked. Whenever the normal 120 ms interrupt occurs, the DSRUPTSW will be positive or positive zero. If it is positive, the contents of DSRUPTSW is decremented or if it is positive zero, it is set to 7. The decrementing and setting to 7 of the DSRUPTSW is used to control the routing to the various subloops of the T4RUPT routine. The contents of DSRUPTSW is used in this manner later in the routine.

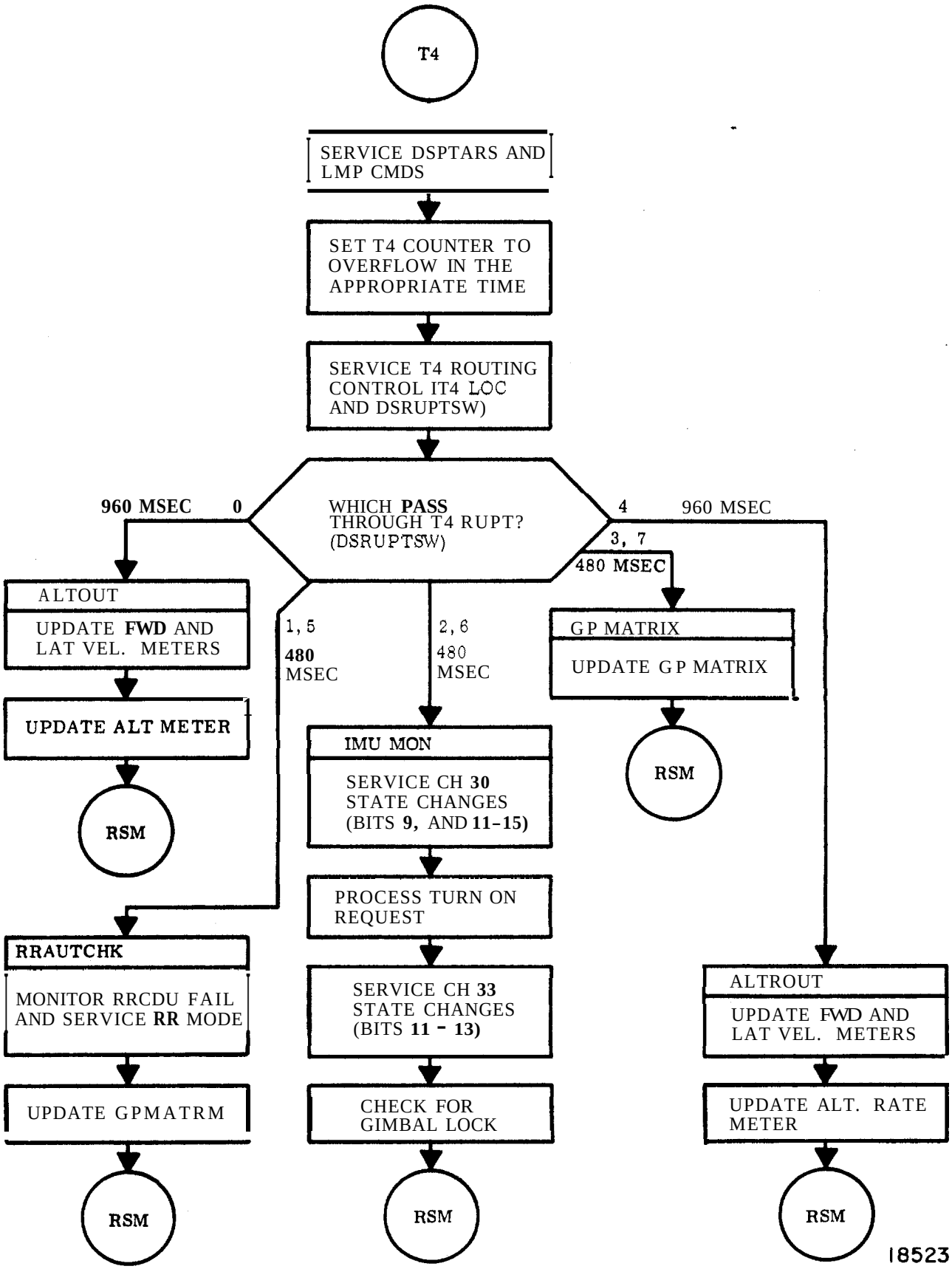


Figure 3-1. General T4RUPT

A check of the word in memory called LMPCMD is made. This word acts as a pseudo DSPTAB and is used for setting particular relays associated with LM interfacing. A unique requirement calling for a 30 msec "power on time" rather than the DSKY relay 20 msec on time. Assuming that a LMPCMD is required, flag bit 15 is reset *so* that this route won't be forced next pass. A special relay code of 74000_8 is added to the marked low order eleven bits that were in LMPCMD and the relay bits and code are placed in output channel 10. After the data is in channel 10 the time 4 counter is set to overflow in 30 msec and the appropriate routing is placed in T4LOC. Now the interrupted program is resumed.

Assuming a LMPCMD is not required a check of DSPTAB 11 is made. If it is negative some other program has made the request for a display of one or more of the items listed for DSPTAB 11. Note table 3-1 which depicts the contents of all display tables. At this point a brief discussion on the mechanism of displaying information will be presented.

Referring to figure 3-2, note that there is an interrelationship between a calling program, the T4RUPT routine, and the output channel 10. The calling program (which desires a display) supplies the appropriate code for the display to the appropriate DSPTAB. If it is a character display for R_1, R_2, R_3, a verb-noun or program number, the low order 9 DSPTAB's are used. The calling program places the code in the correct DSPTAB and sets the sign of that DSPTAB negative (i.e., bit 15 = 1). This bit 15 will identify to the T4RUPT program that the contents of that particular display table requires processing. Notice bits 12 - 14 are not used in the DSPTAB word. Only the low order 11 bits (and bit 15) have any significance. By virtue of the address of the particular DSPTAB a 4 bit code is assigned. This 4 bit code is a relay code whose sole function is to apply ground to the appropriate relay bank associated with the desired display.

T4RUPT scans the DSPTAB's every 120 msec starting at DSPTAB 11. If a particular DSPTAB is negative, T4RUPT "attaches" the particular relay code to the low order relay bits and places the resulting 15 bits into output channel 10 which, in turn, results in the activation of the appropriate relays causing the desired display,

If a requirement to drive a C relay exiets, reset bit 15 *so* that on the next pass a redisplay of identical data will not occur. Then, extract low order 11 bits (relay bits) and attach associated 4 bit relay code.

Transfer the entire 15 bit display word into output channel 10 and identify 20 msec RUPT. This allows sufficient time for the latching relays to pull in.

If there is not display requirements in DSPTAB 11 each of the DSPTAB's 10 - 0 are checked twice, If a display is required, bit 15 is reset and its code and command are set into output channel 10.

If no displays are required the normal 120 msec RUPT is set up and the contents of DSRUPTSW is checked to identify the pass #.

Bit	11	10	9	8	7	6	5	4	3	2	1
DSPTAB+11D			Program Caution	Tracker Warning		Gimbal Lock		No Att			
DSPTAB+10D	MD1 (5 bit Relay Code)							MD2 (right hand character)			
DSPTAB+9D	VD1							VD2			
DSPTAB+8D	ND1							ND2			
DSPTAB+7								R1D1 (left hand character)			
DSPTAB+6	+R1S		R1D2					R1D3			
DSPTAB+5	-R1S		R1D4					R1D5 (right hand character)			
DSPTAB+4	+R2S		R2D1					R2D2			
DSPTAB+3	-R2S		R2D3					R2D4			
DSPTAB+2			R2D5					R3D1			
DSPTAB+1	+R3S		R3D2					R3D3			
DSPTAB	-R3S		R3D4					R3D5			

Table 3-1. The 12-Word Display Table Bit Assignments

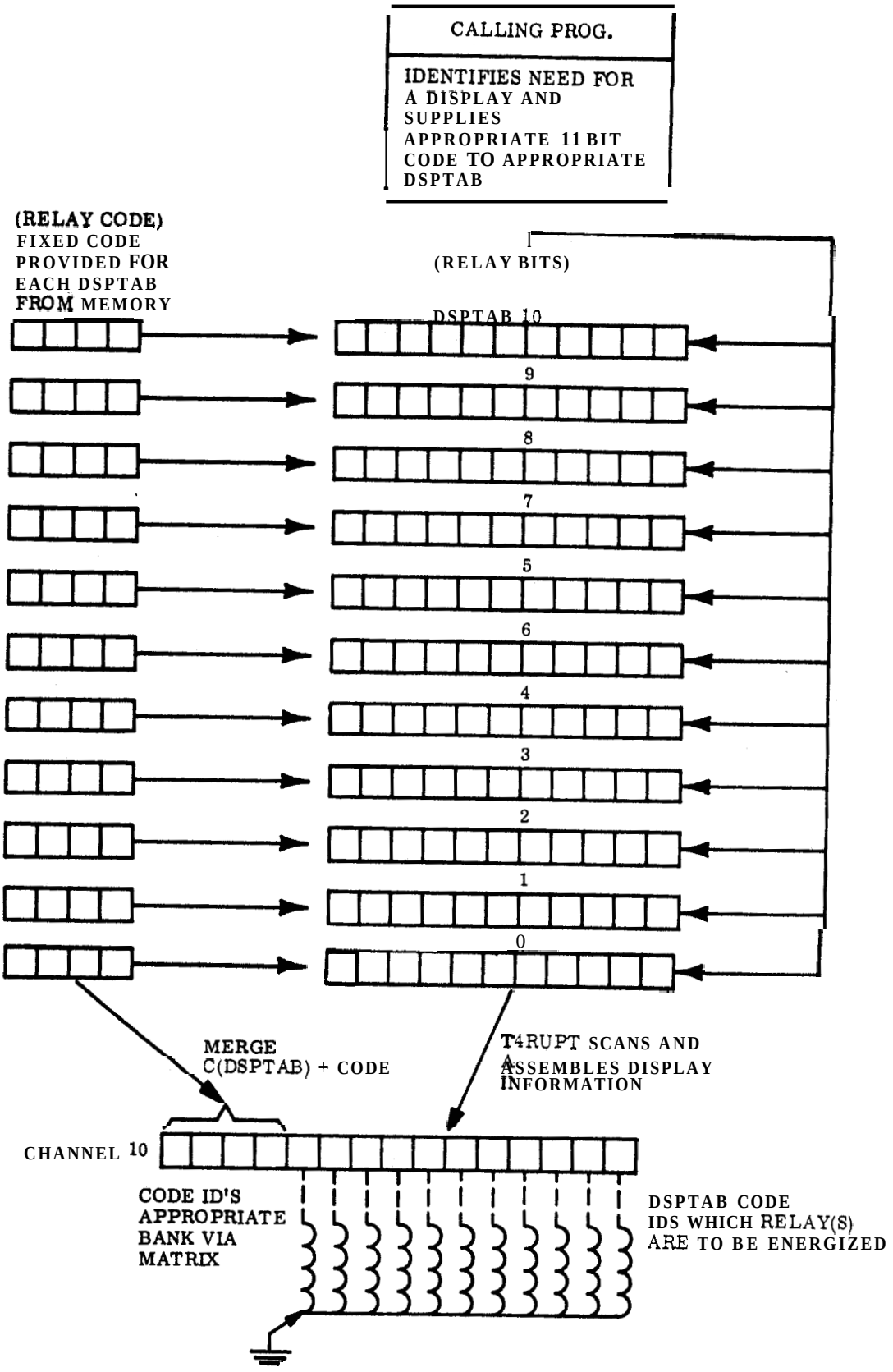


Figure 3-2. DSPTAB Code

Assuming we are on pass 0, control is rerouted to the ALTOUT portion of the T4RUPT routine.

3.1.2 ALTOUT. This routine is entered every 960 msec or every eighth pass through the T4RUPT routine when the contents of **DSRUPTSW** is equal to zero. The purpose of this subroutine is to update the altitude and forward and lateral velocity meters. The **ALTROUT** subroutine uses a portion of the **ALTOUT** subroutine and also updates the forward and lateral velocity meters. The forward and lateral velocity meters are updated every **480** msec.

The computer receives altitude, altitude rate and forward and lateral velocity information from the landing radar; this information is used in developing the various display drive signals. The altitude display data is transferred into the **ALTM** output counter and gated out to the altitude meter, The forward and lateral velocity display data is transferred into the **OPTXCMD** and **OPTYCMD** output counters and gated out to their respective meters. The actual gating out of information does not begin until output channel **14** bits **3, 11** and **12** are set to equal **1.** Forward and lateral velocity information is applied to the **RR CDU** error counters where it is converted from digital to analog information. The information is then routed to the altitude meter for display, The altitude information is gated directly to the altitude meter, the altitude meter converts the information from digital to a visual readout. Altitude information can be calculated by multiplying the altitude rate by the loop time (. **96** sec) to develop the change in altitude, and by subtracting the altitude change from the last altitude, the latest altitude can be generated.

3.1.3 ALTROUT. The **ALTROUT** subroutine is entered every 960 msec or every time the contents of the **DSRUPTSW** is equal to **4.** The purpose of this subroutine is to update the altitude rate meters display. The **ALTROUT** subroutine updates the forward and lateral velocity meters as well as the altitude rate meter.

3.1.4 RR AUT CHK (RENDEZVOUS RADAR AUTOMATIC CHECK). The T4RUPT program does this routine every **480** msec. **RRAUTCHK** services the **RR** inbits and drives the antenna. Erasable memory location **RADMODES** is updated to the latest **RR** condition every time the **RR** inbits change. *See* table 3-2.

If the **RR AUTO MODE** bit of channel **33** changes, a check is made to determine whether it just came on or just went off. The **RR AUTO MODE** bit just going off while a program is using the radar causes a **PROGRAM CAUTION** and the failure is displayed in R1 of the **DSKY.** If the **RR AUTO MODE** bit just came on while no other program was using the radar, the RR TURN-ON is initiated by scheduling the RR TURN-ON task.

The **RR TURN-ON task:** zeroes the radar **CDU** channels, sets the computer **RR CDU** counters to zero, sets **RADMODES** to agree with the antenna angles, and lights the **TRACKER WARNING** lamp if there are any tracker fails present.

The **RRCDU CHK** is performed if the **RR CDU** fail bit of channel **30** changes. If the **RR** is in the **AUTO** mode and the RR **CDU FAIL, LR FAIL,** or **RR DATA FAIL** is present, the **TRACKER WARNING** lamp is lit.

The **RRGIMON** routine monitors the **RR** antenna angles and initiates a reposition of the antenna is needed **(DORREPOS)** if the antenna angles exceed their limits specified by the particular mode.

The DORREPOS task selects the proper zero reference for the RR antenna mode and repositions the antenna to these reference positions. The antenna is driven one axis at a time; trunnion first and then shaft to within 1^{o} of the reference position. The drive pulses are limited to 384 pulses maximum for each CDU load. Every 1/2 second the CDU is driven until the particular axis angle is within one degree of the zero reference position. After driving the antenna to the zero reference position a check is made for the designate request. If the designate request is present, a check is made for a remode request and if not present, the START DES routine is performed.

The START DES routine schedules the job DODES every 1/2 second and monitors the amount of time required to achieve lock-on. If more than 30 seconds is required, the job is terminated and the PROGRAM CAUTION lamp is lit along with a display of the failure in R1 of the DSKY.

The job DODES calculates the shaft and trunnion $\Delta\theta$ angles to the desired target, scales the $\Delta\theta$ angles, develops and limits the necessary drive commands to a maximum of 384 pulses and terminates DODES when the antenna is within $.7^{o}$ of the desired angle.

If a REMODE request is present after a reposition, the antenna is driven as follows: (1) trunnion to 0^{o} or 1800 (mode 1 or 2), (2) shaft to -450, and (3) then trunnion to -120^{o} or -600 (mode 1 or mode 2).

3.1.5 IMU MONITOR, This portion of the T4RUPT program is entered every 480 msec when the contents of DSRUPTSW = 2 or 6. The purpose of this subroutine is to process changes in the status of the IMU and its associated moding or failures. See tables 3-3 and 3-4. Upon entering IMU monitor, channel 30 bits 9, 11 - 15 are checked to see if any bits changed. This is done by comparing c(IMODES 30 bits 9, 11 - 15) with channel30 bits 9, 11 - 15. IMODES 30 contains the last configuration of channel 30 and if any bit changed the appropriate action is initiated. The bits are scanned from 15 - 9 in order.

If any bit changed, IMODES 30 is updated to reflect latest change and the appropriate action is initiated.

If bit 15 changed, the IMU temperature status changed and a check is made to see if the temperature just went in limits or out of limits. If it went out of limits, the TEMP CAUTION lamp is lit. If the temperature just went in limits, the TEMP CAUTION lamp is turned off if lamp test is not in progress. After servicing bit 15 a check is made of bit 14.

If bit 14 changed, it means that there has been a change in the TURN-ON REQUEST discrete associated with ISS TURN-ON. If the request just came on and there is not a fail bit present, a flag is set to indicate first sample and nothing is done until the second sample. If the request just went off and the 90 second delay was not completed, a fail flag is set and the PROGRAM CAUTION lamp is lit and the failure indicated.

If bit 13 or 12 changed either the IMU or the CDU FAIL status changed, If the failure was not inhibited, the ISS WARNING lamp is lit. If the failure was inhibited and lamp test was not in process, the ISS WARNING lamp is turned off if on.

If bit 10 changed (the caging indication), a check is made to determine whether it came on or off. If it came on, all ISS driving is terminated, the NO ATTITUDE lamp is lit, and PIPA, ICDU and IMU failure inhibit bits of IMODES 30 are set. If the caging indication was just removed, the next channel 30 bit is checked.

If bit 9 (IMUOPERATE) changed, a check is made to determine if it just came on, If it is just on and no turn on request fail present, the first sample bit is set. If bit 9 was just removed and a program *was* using the IMU, the PROGRAM CAUTION lamp is lit and the failure displayed in R1.

After servicing the channel 30 bits, the turn on test is entered. If the first turn on sample bit is present, the second turn on sample bit is set and the next time through T4 the turn on test will be completed. Assume that the second sample bit has been set and we enter turn on test. The turn on sample bits are reset and if the turn on request and IMU operate bits are present turn on is initiated. The coarse align, zero ICDU discretes are issued, the NO ATTITUDE lamp is lit, the IMU failure inhibit bits are set and a 90 second time delay is initiated. After 90 seconds the ENDTNON task is initiated. If the turn on request bit had been present and no IMU operate present, the PROGRAM CAUTION lamp is lit and the failure displayed in R1. If the TURN ON REQUEST bit is not present and coarse align not present and no program using IMU, the IMU fail inhibit bits are set, zero ICDU is issued and the UNZ2 task is scheduled for 300 msec. The normal exit from the turn on test is to the channel 33 test (C33 TEST). After 300 msec, the task UNZ2 is performed which removes coarse align and zero ICDU discretes, and after a delay of 4 seconds removes ICDU and IMU failure inhibits, and checks for these failures. If either is present, the ISS WARNING lamp is lit. If neither is present, the ISS WARNING lamp is turned off if on and not in lamp test. The ISS turn on delay complete discrete is removed and task PFAIL OK is scheduled for 10 msec. After 10 msec the PIPA FAIL bits are RESET and the caging indications are reset. If any uninhibited failures are present, the ISS WARNING lamp is turned on.

The C33 test monitors channel 33 bits 13 - 11 for failure indications. If no bits changed the GLOCKMON check is made.

If bit 13 (PIPA FAIL) changed, IMODES 33 is updated to reflect the latest and if any uninhibited failures are present, the ISS WARNING lamp is lit. If a PIPA FAIL and its inhibit bit are present with the IMU in operate after initialization, the PROGRAM CAUTION lamp is lit and the failure displayed.

If bit 12 changed, the PROGRAM CAUTION lamp is lit, if bit 12 just came on and the failure (downlink too fast) is displayed in R1.

If bit 11 changed, the PROGRAM CAUTION lamp is lit if bit 11 just came on and the failure (uplink too fast) is displayed in R1.

After servicing channel 33 the gimbal lock monitor routine is initiated (GLOCKMON). The GLOCKMON routine monitors the middle gimbal angle. If the angle is $> 70^{\circ}$, the gimbal lock lamp is lit. If the middle gimbal angle is $> 85^{\circ}$ and the ISS is not in coarse align; the coarse align discrete is issued, the NO ATTITUDE lamp is lit and the IMUFAIL INHIBIT bit is set.

If the gimbal angle is $< 70^{\circ}$, the GIMBAL LOCK lamp is turned off if not in lamp test.

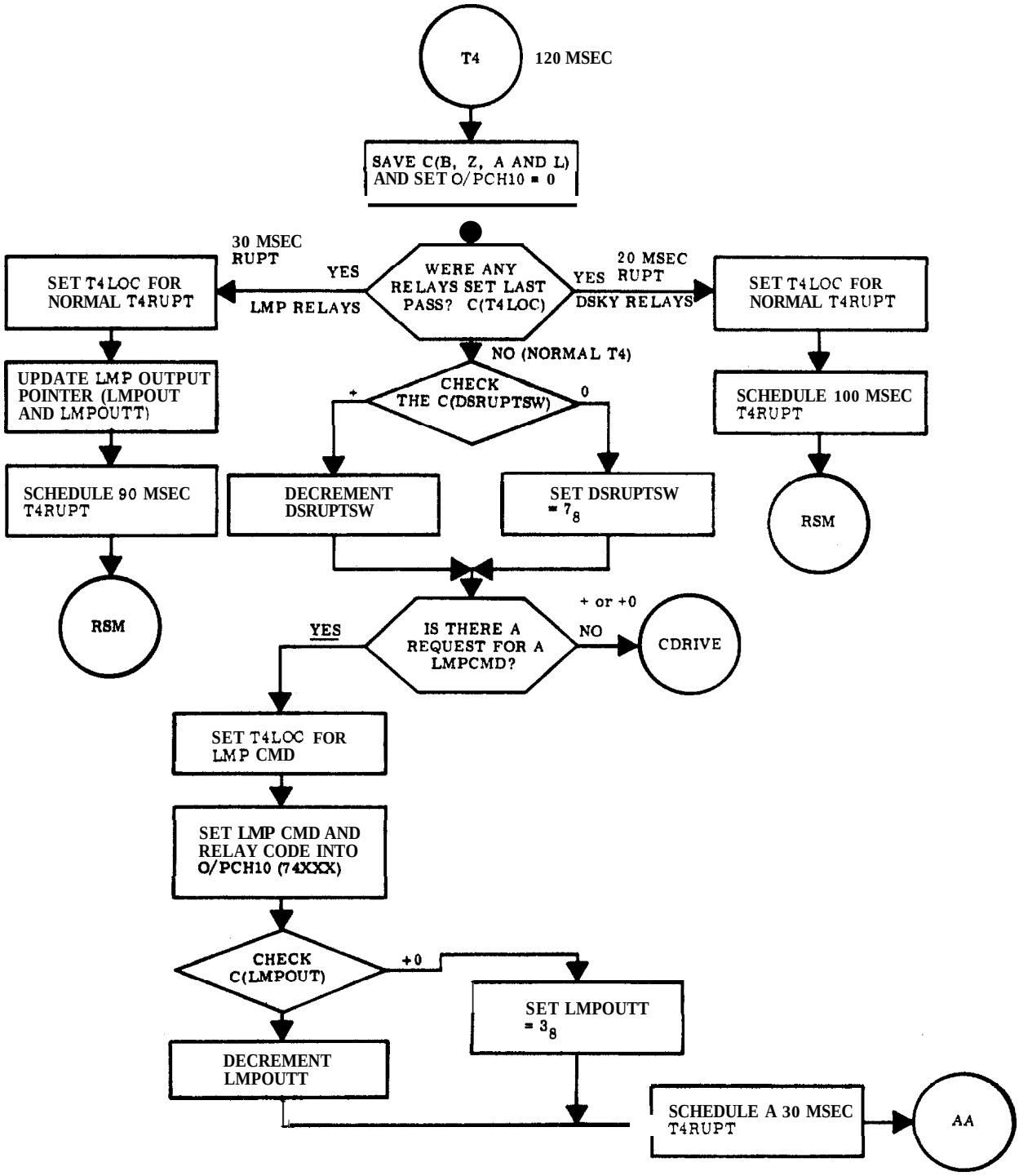


Figure 3-3. Detailed T4RUPT (Sheet 1 of 28)

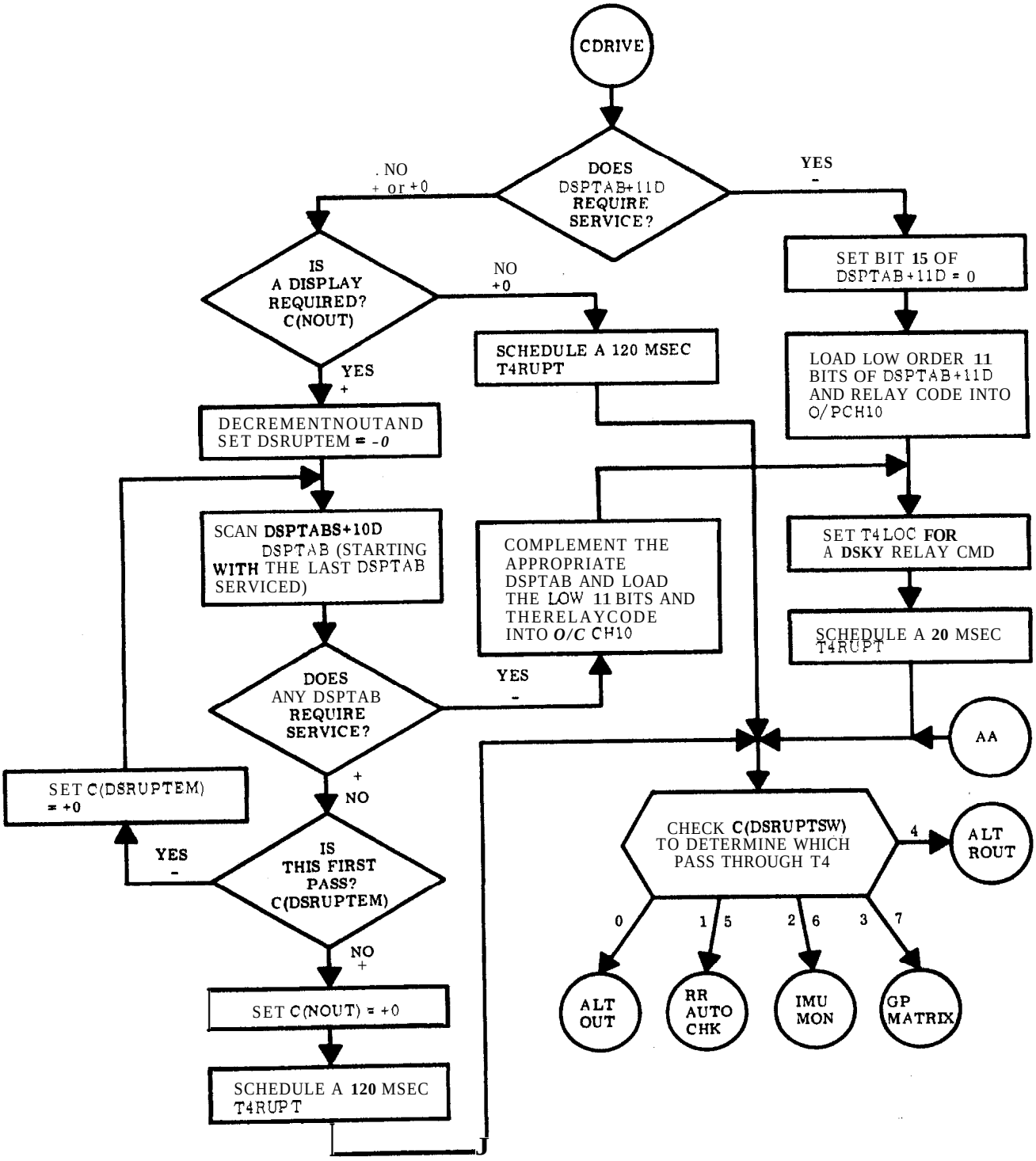


Figure 3-3. Detailed T4RUPT (Sheet 2 of 28)

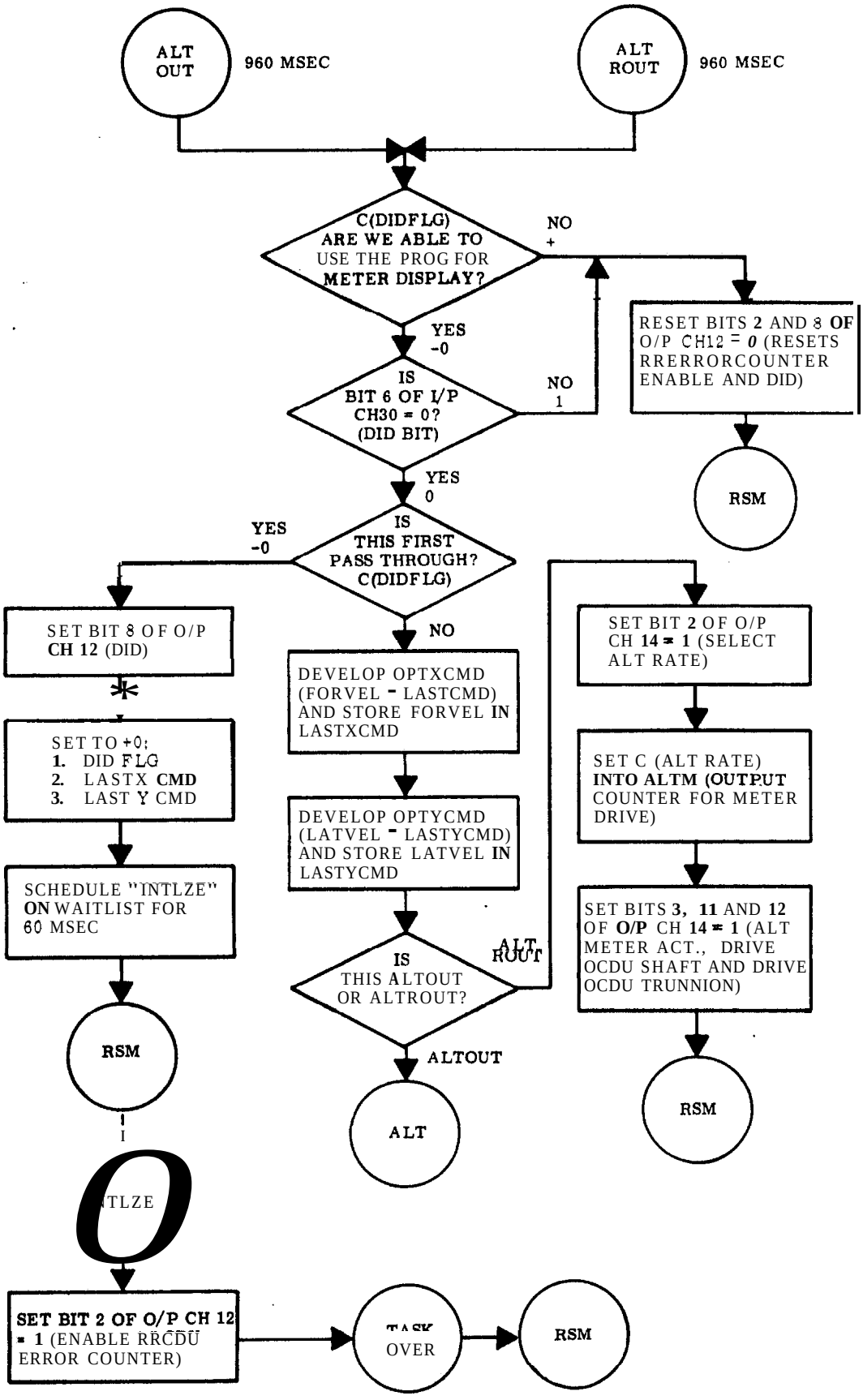


Figure 3-3. Detailed T4RUPT (Sheet **3 of** 28)

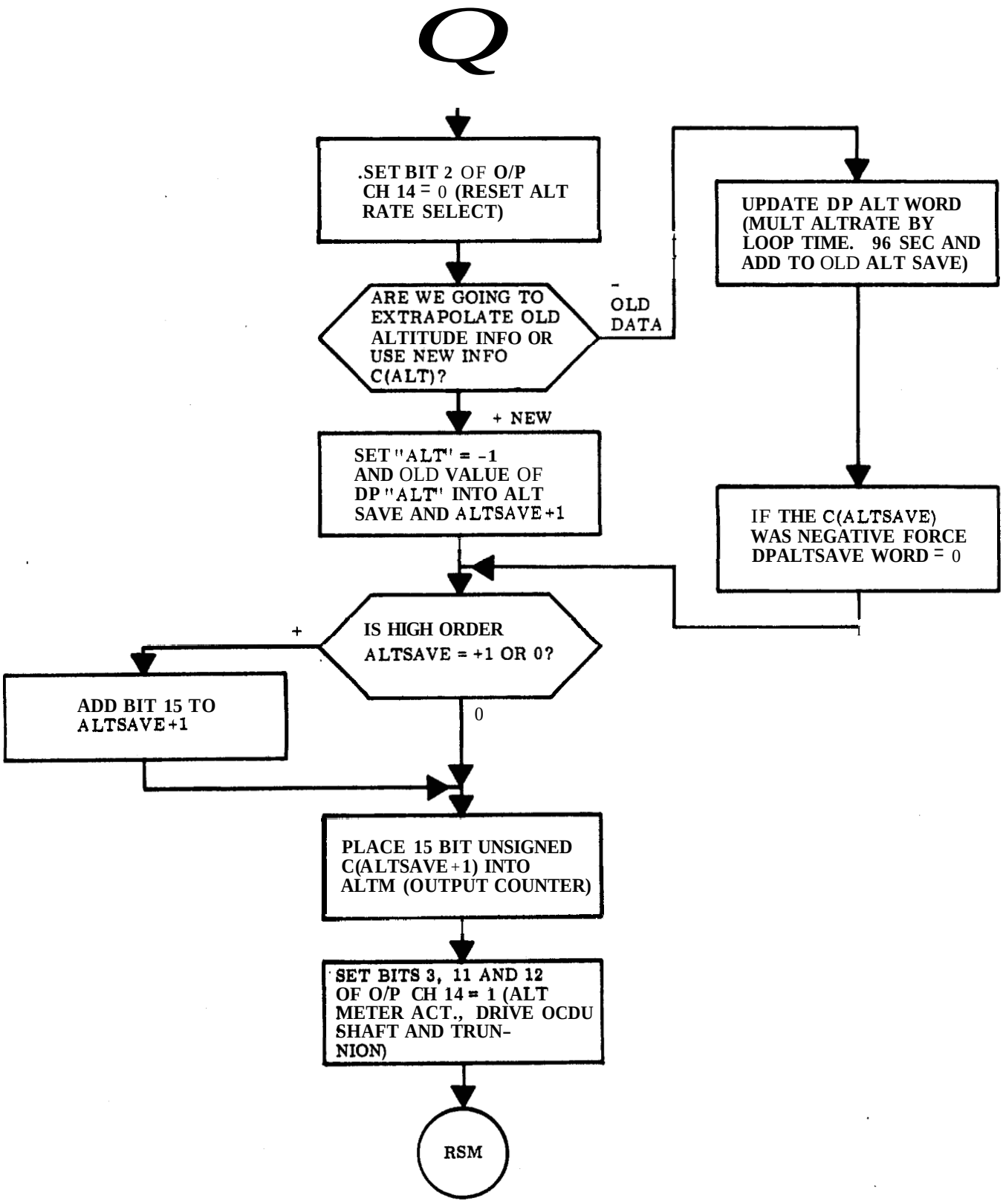


Figure 3-3. Detailed T4RUPT (Sheet 4 of 28)

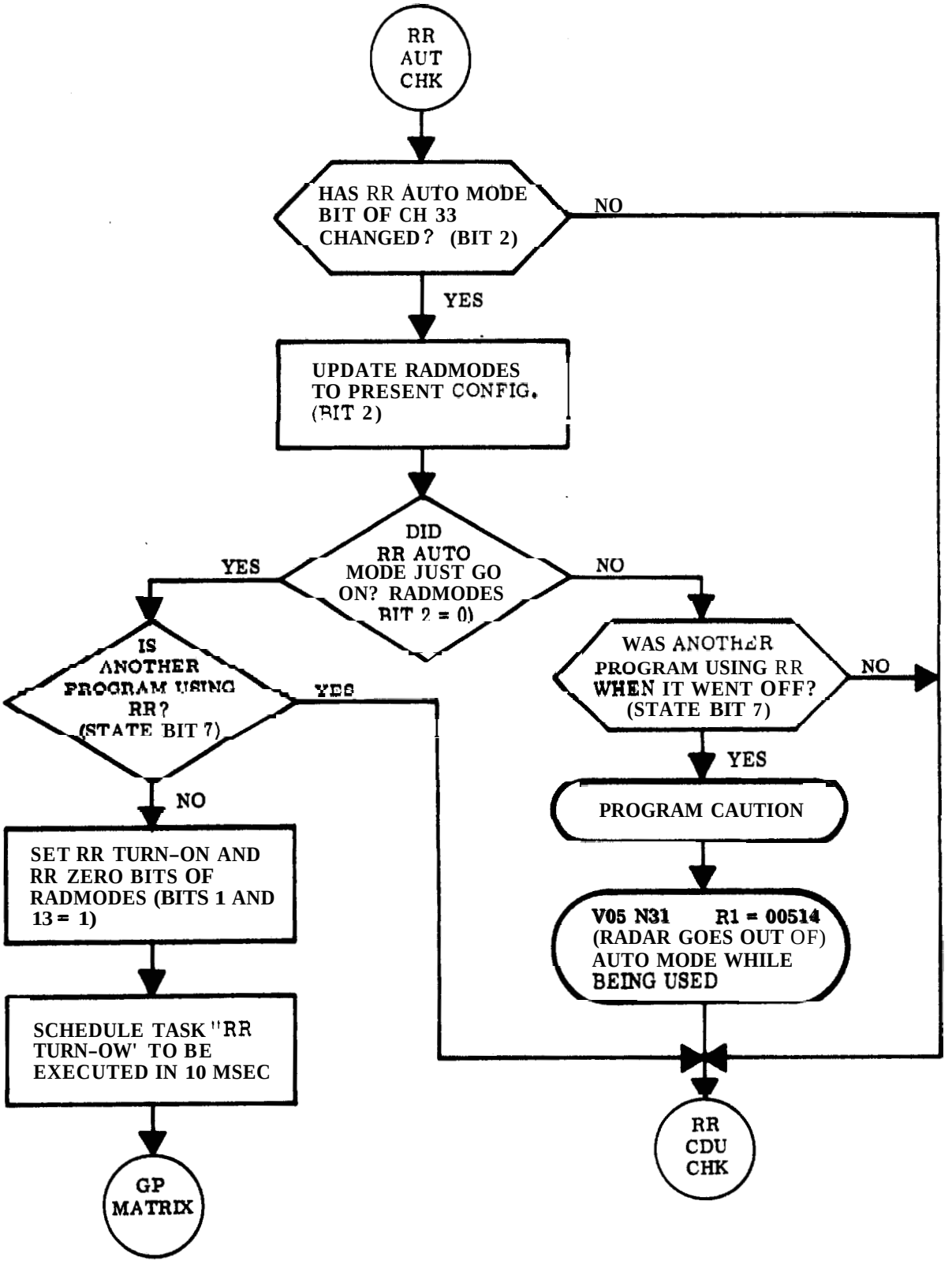


Figure 3-3. Detailed T4RUPT (Sheet 5 of 28)

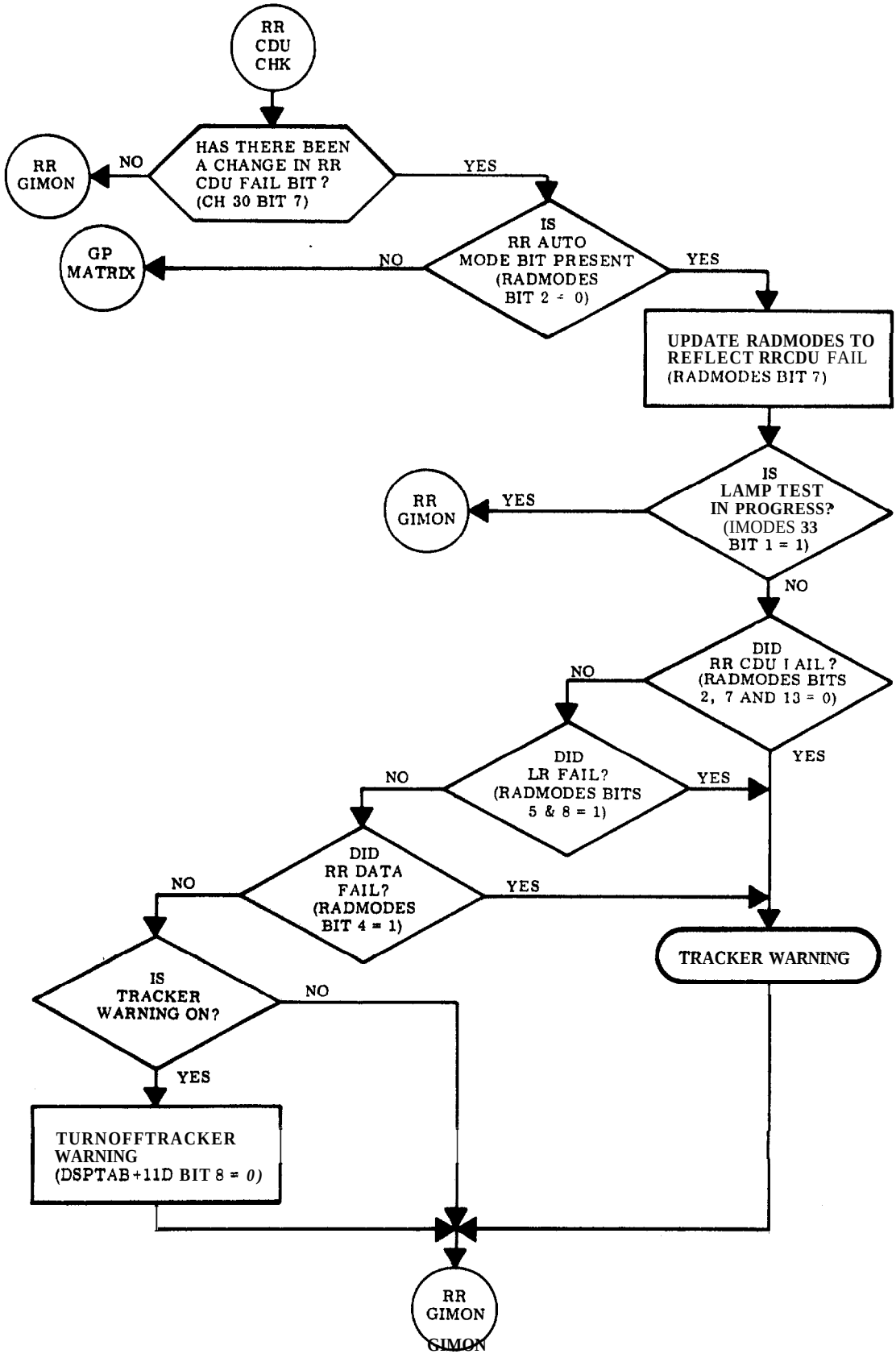


Figure 3-3. Detailed T4RUPT (Sheet 6 of 28)

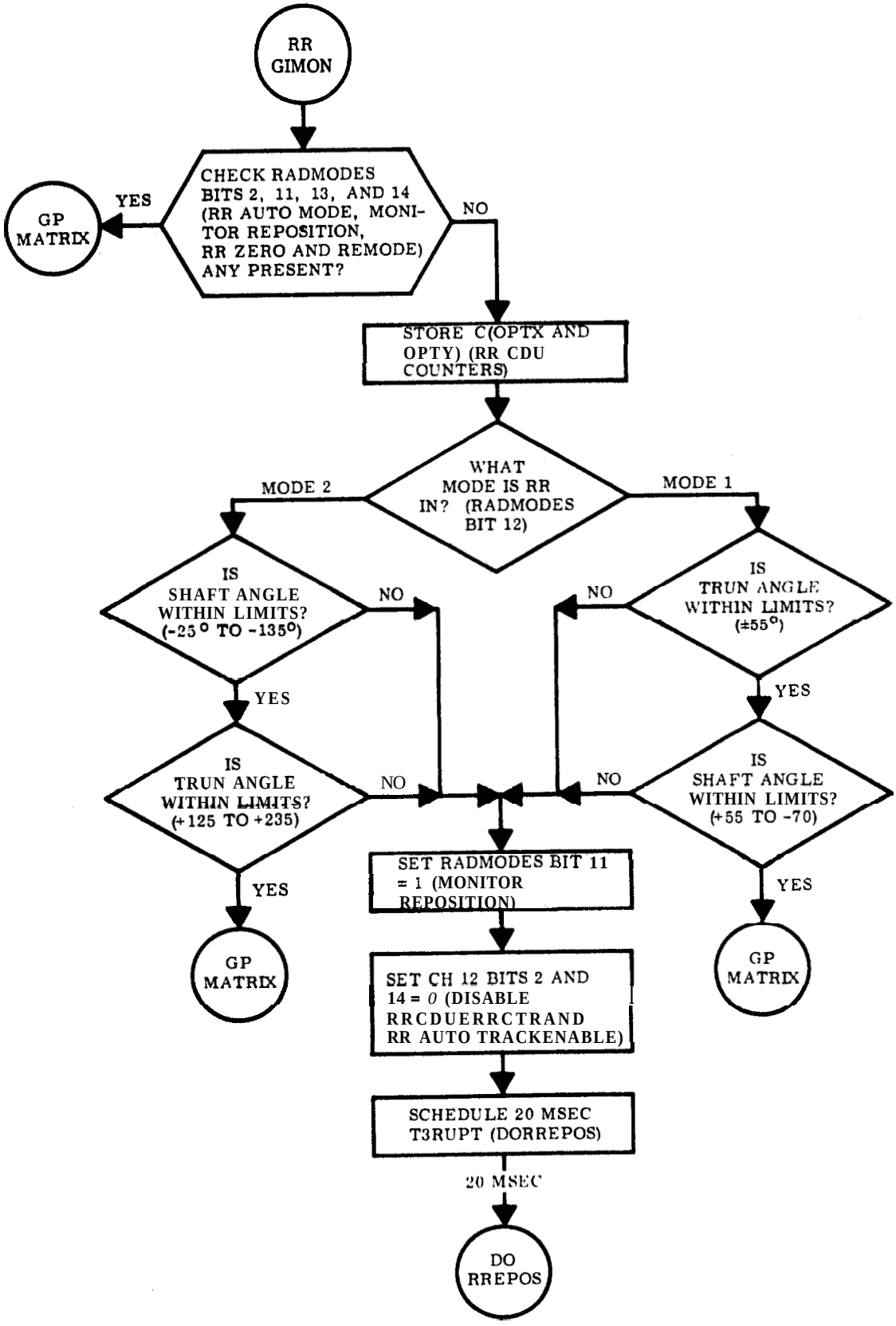


Figure 3-3. Detailed T4RUPT (Sheet 7 of 28)

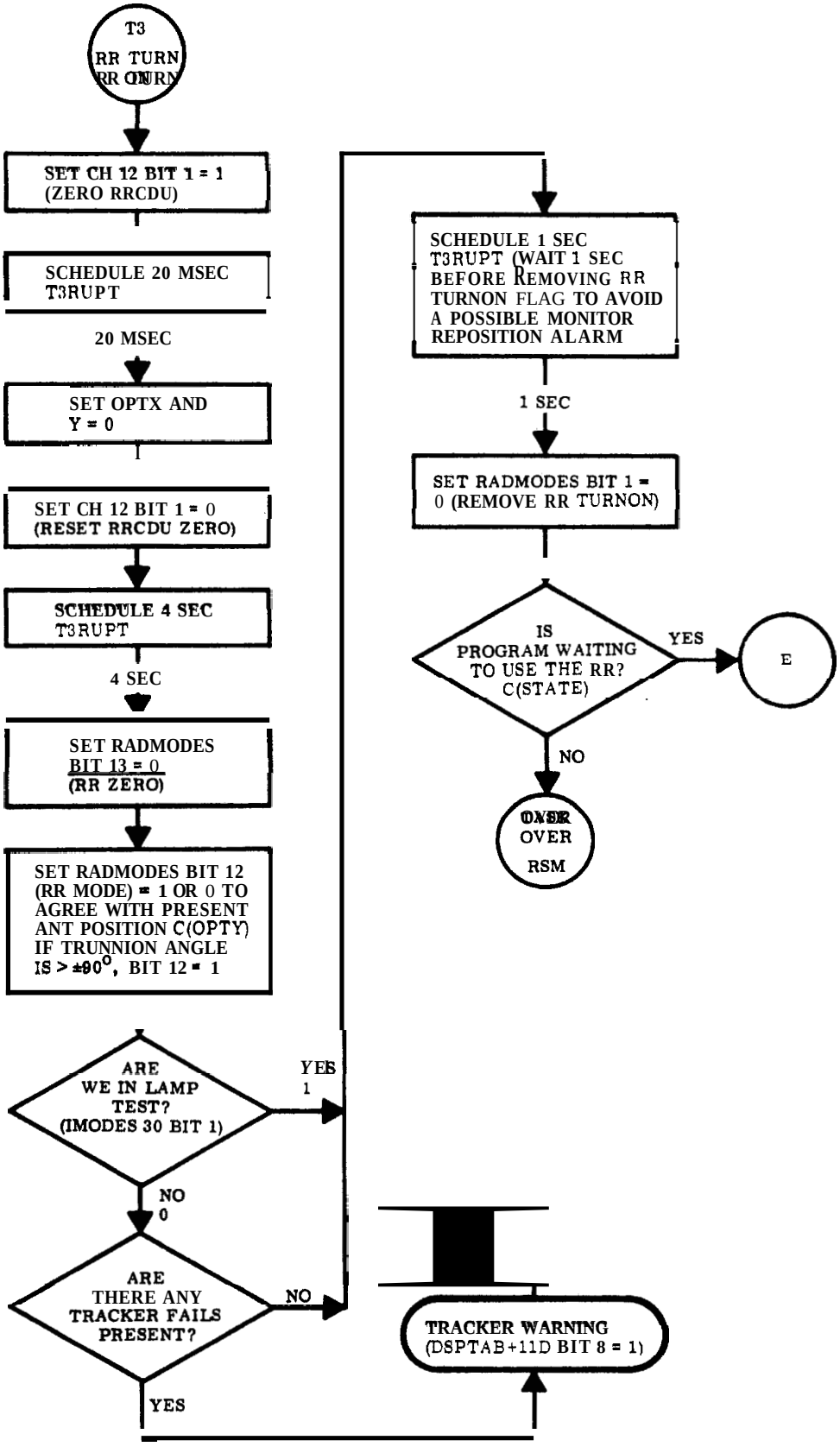


Figure 3-3. Detailed T4RUPT (Sheet 8 of 28)

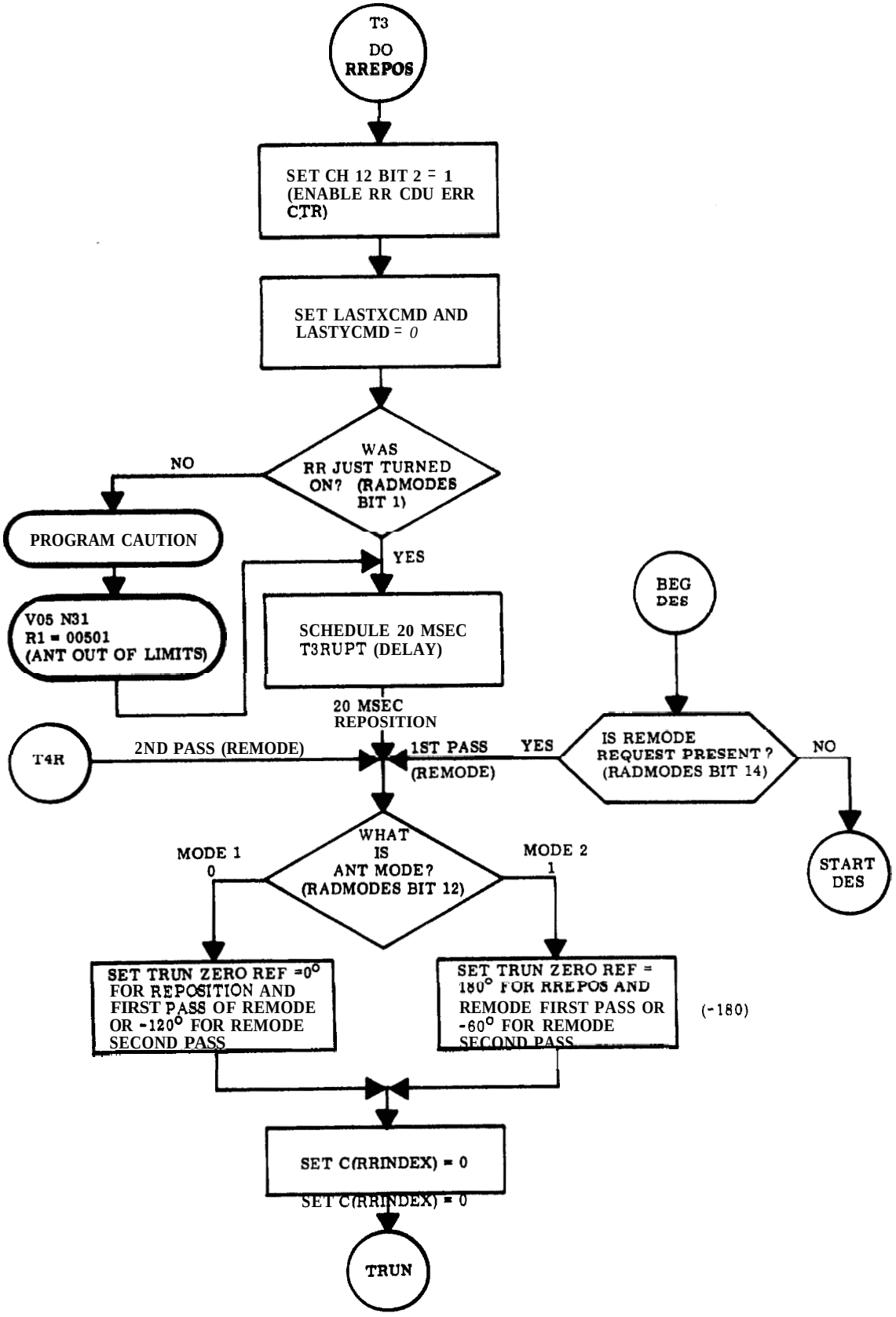


Figure 3-3. Detailed T4RUPT (Sheet 9 of 28)

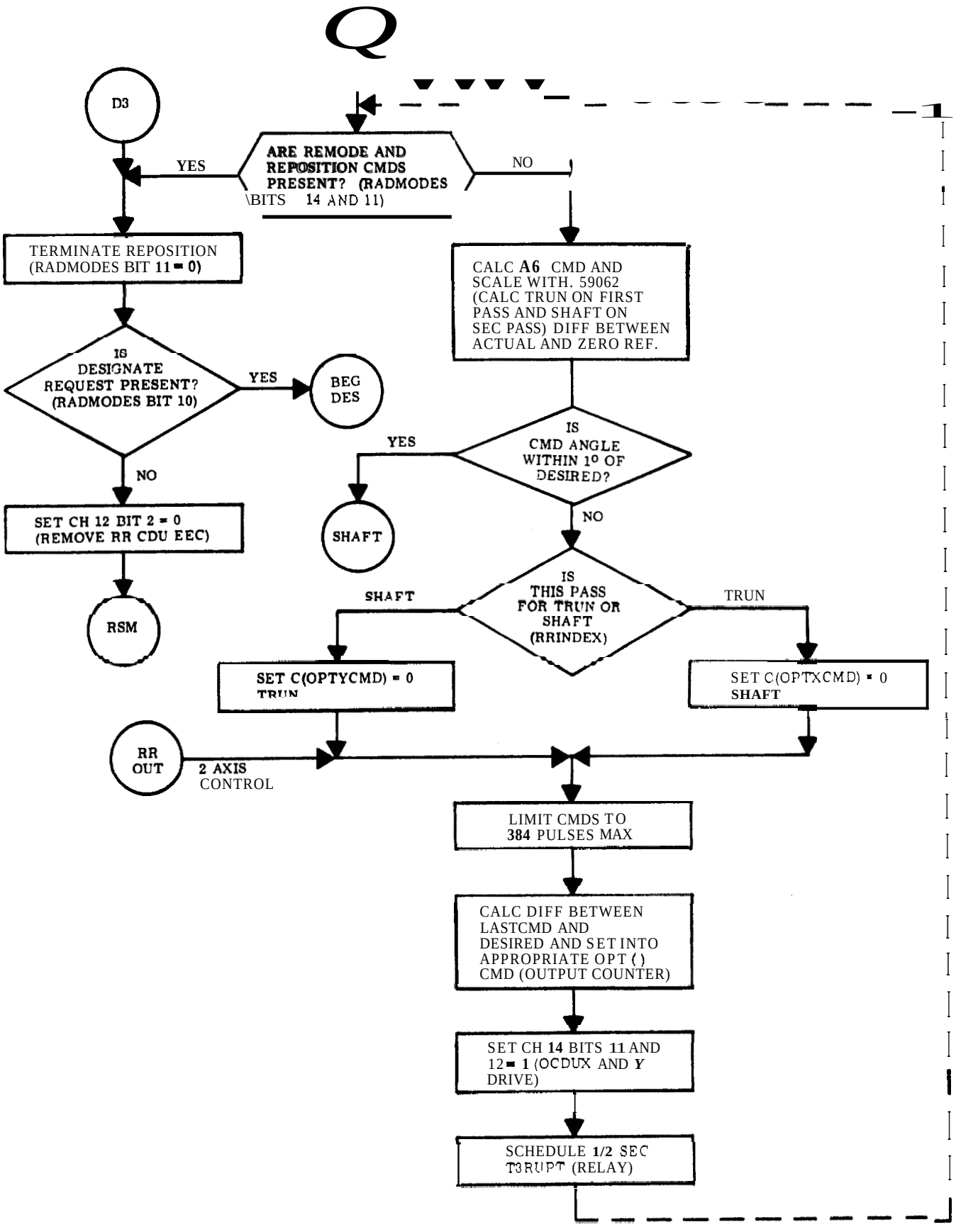


Figure 3-3. Detailed T4RUPT (Sheet 10 of 28)

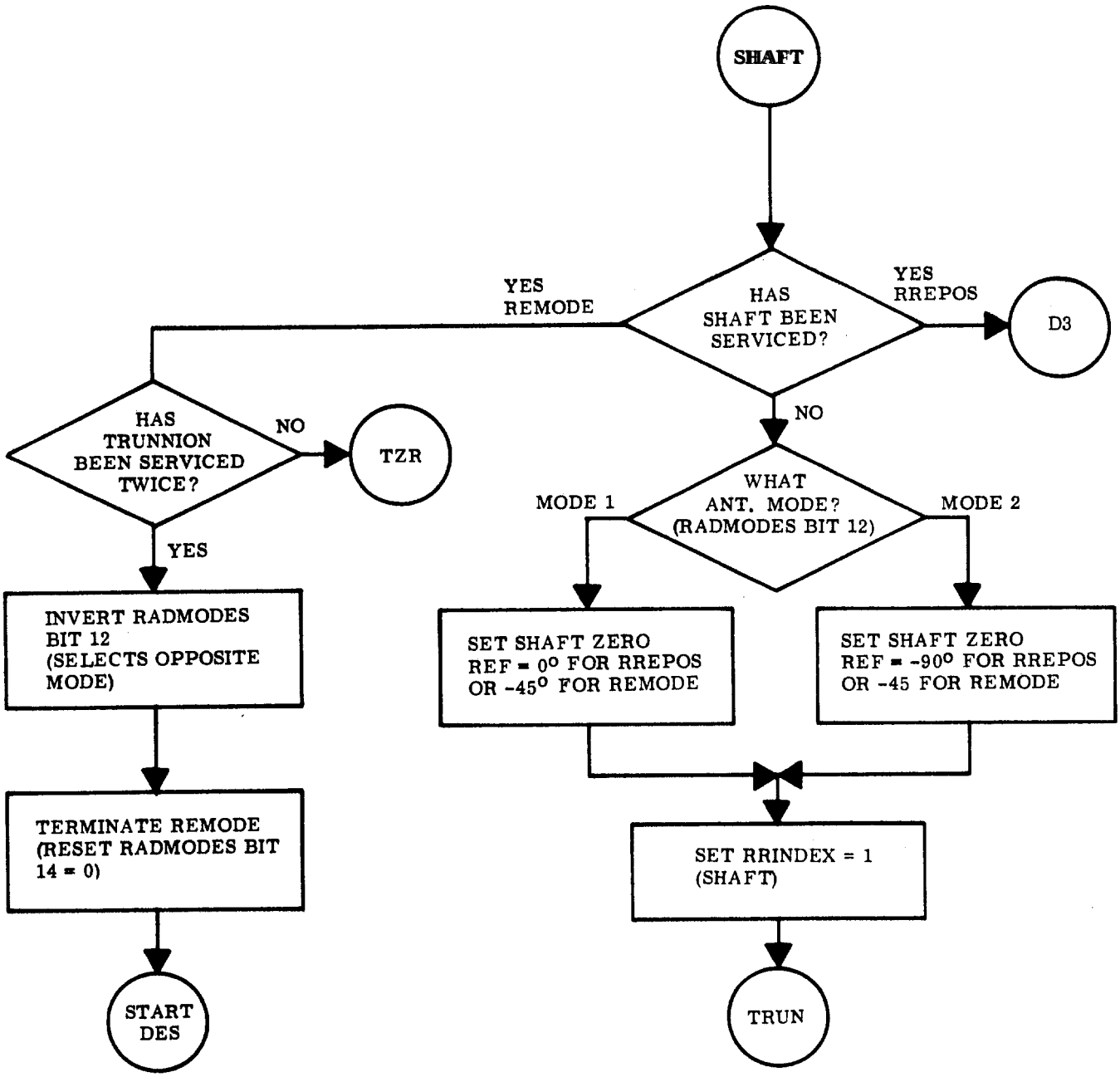


Figure 3-3. Detailed T4RUPT (Sheet 11 of 28)

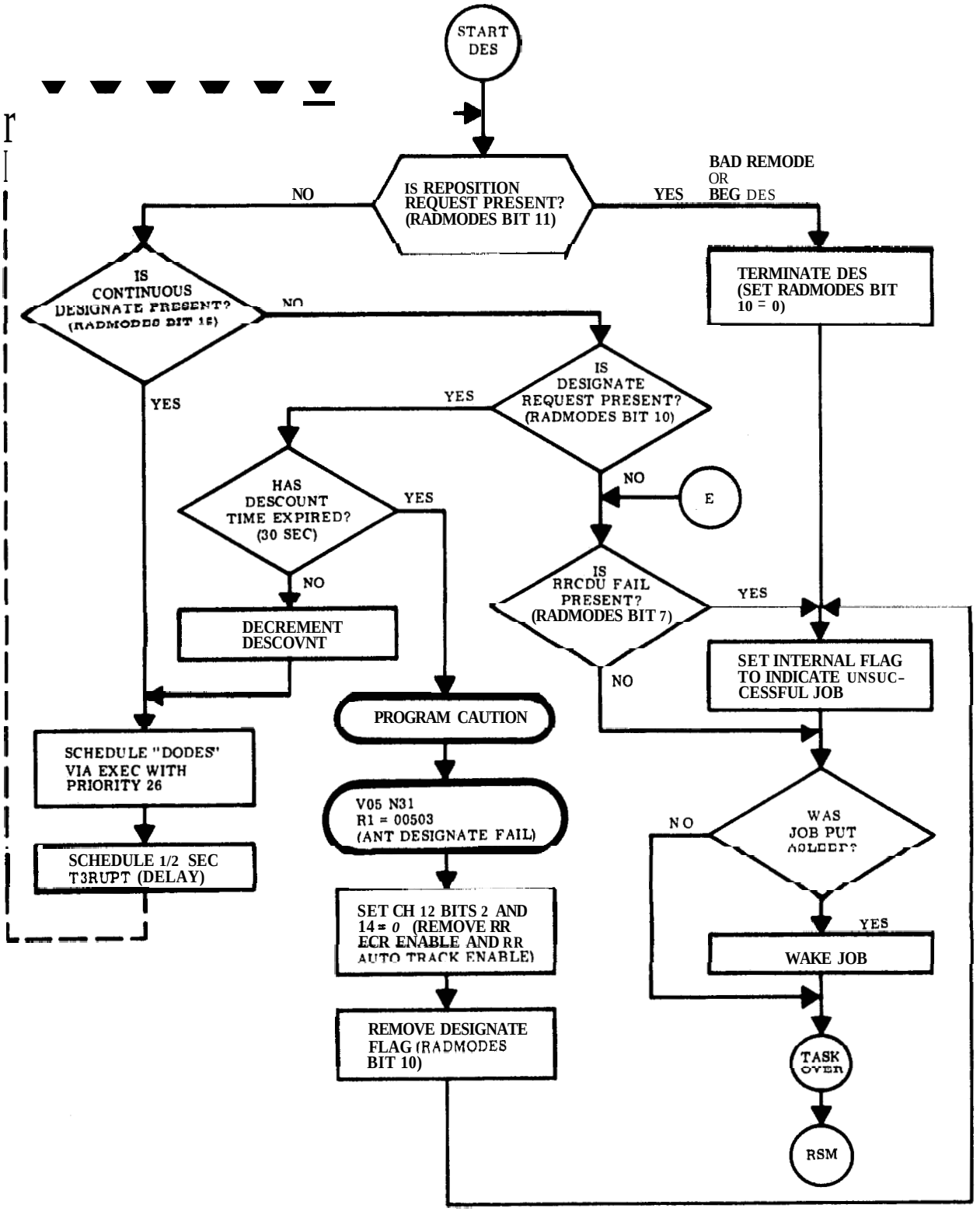


Figure 3-3. Detailed T4RUPT (Sheet 12 of 28)

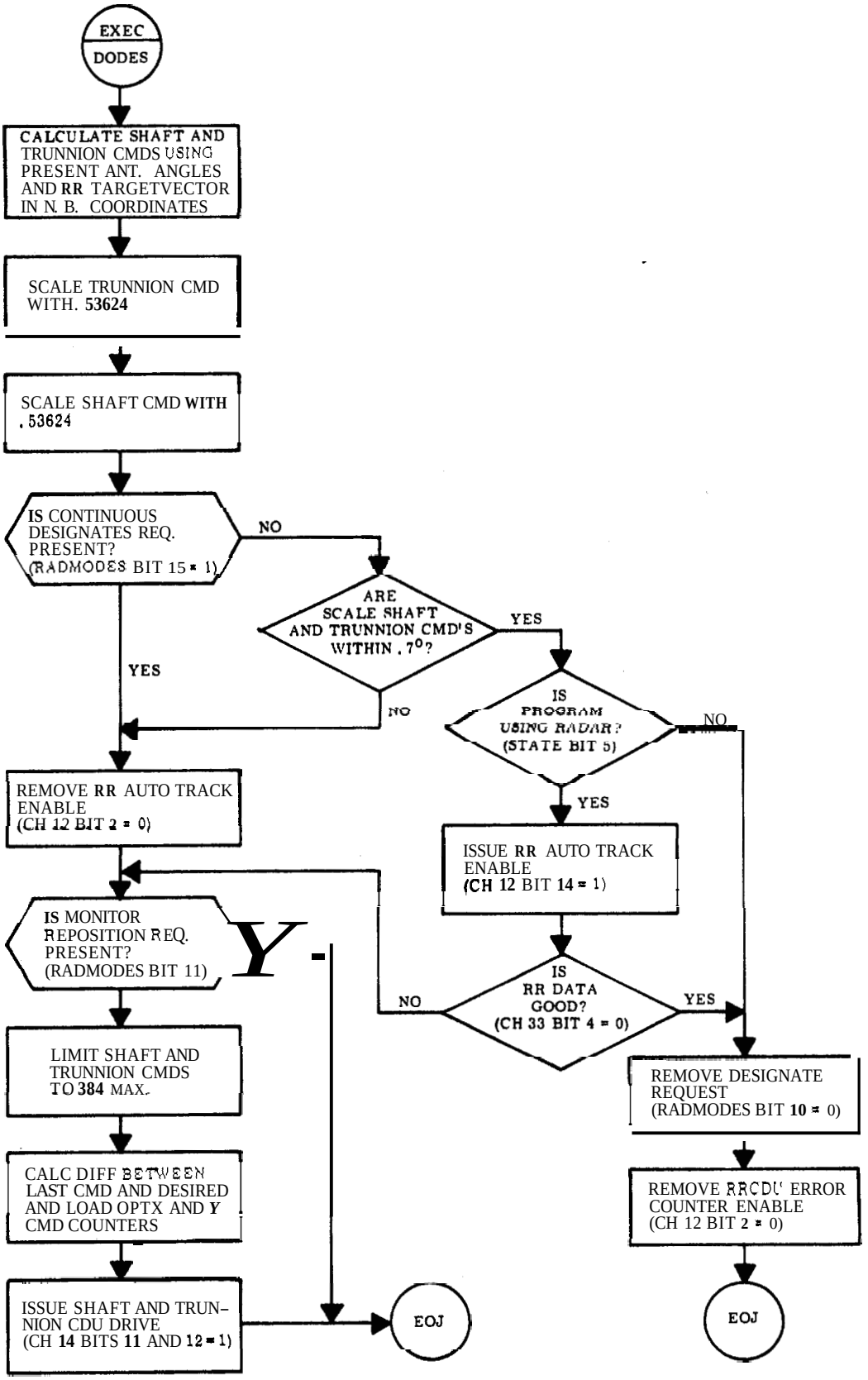


Figure 3-3. Detailed T4RUPT (Sheet 13 of 28)

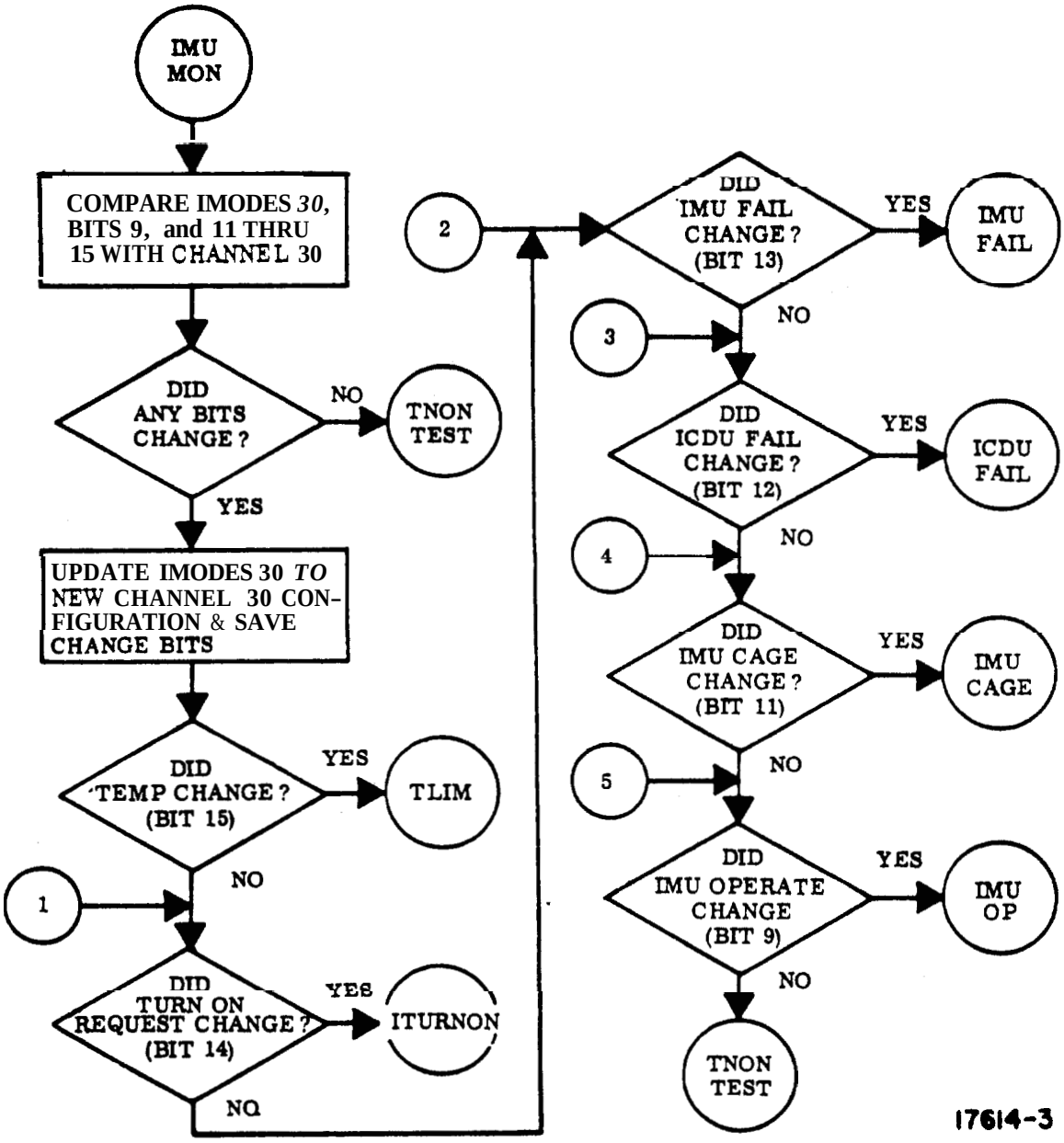


Figure 3-3. Detailed T4RUPT (Sheet 14 of 28)

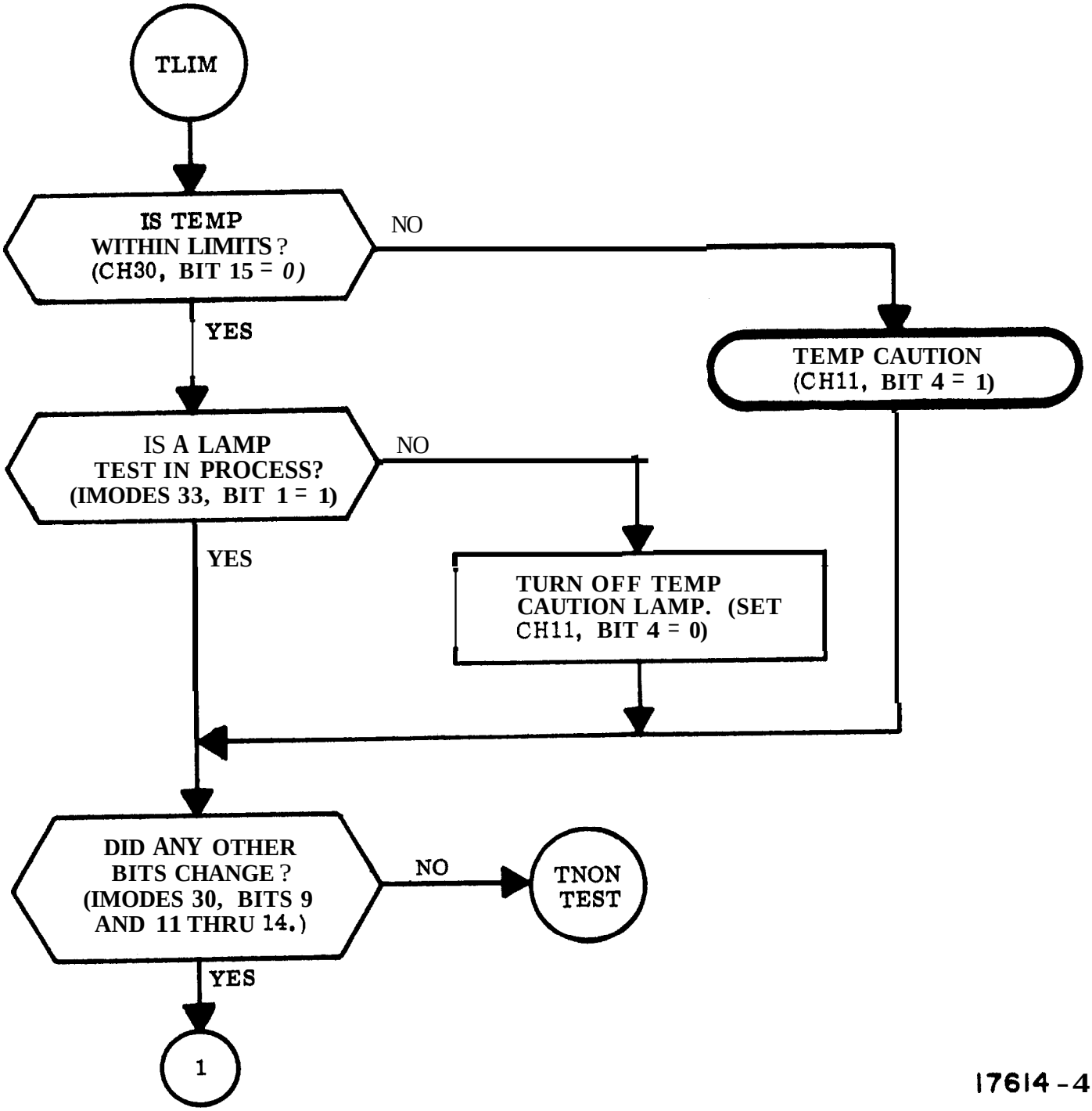


Figure 3-3. Detailed T4RUPT (Sheet 15 of 28)

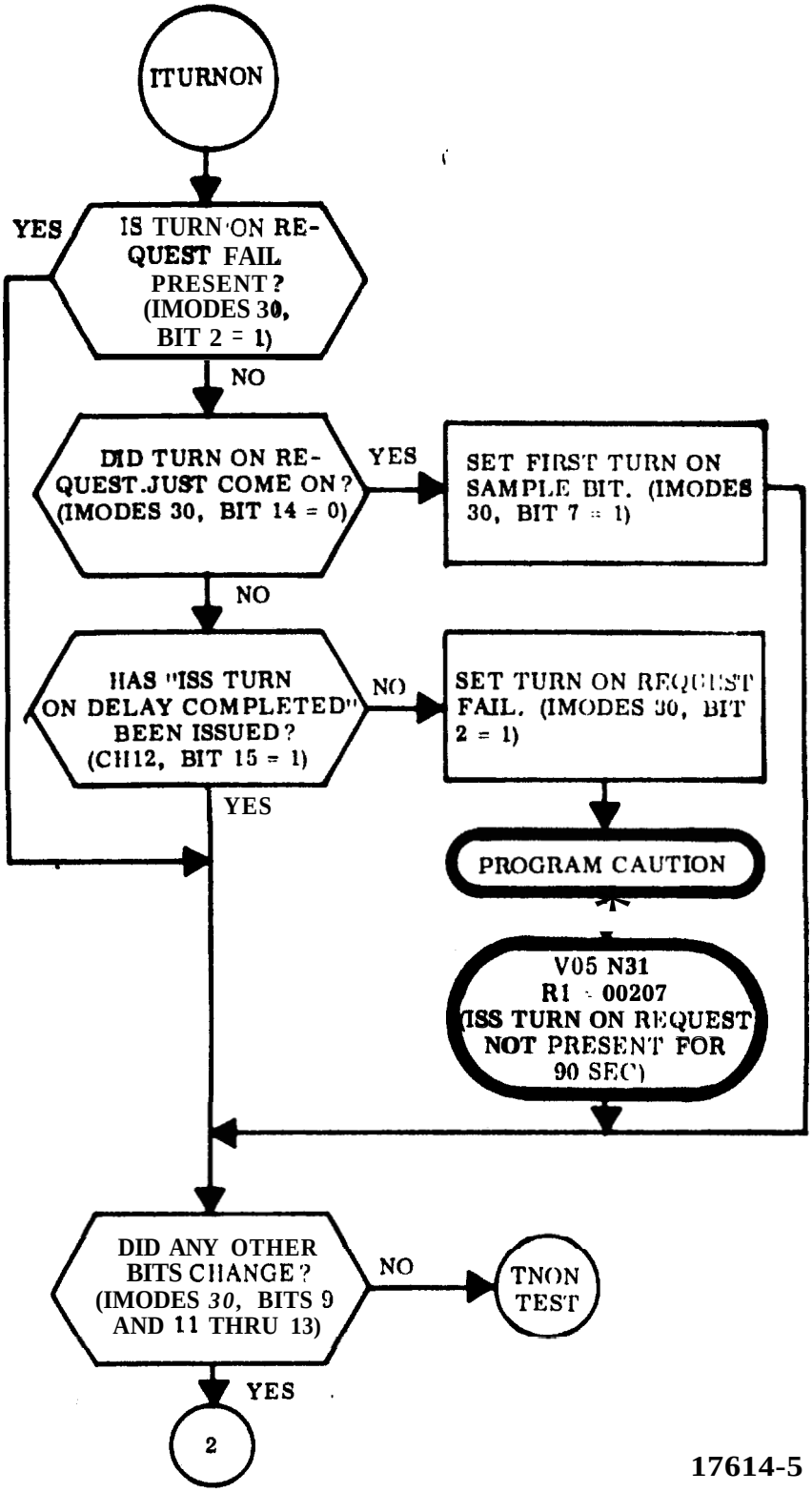


Figure 3-3. Detailed T4RUPT (Sheet 16 of 28)

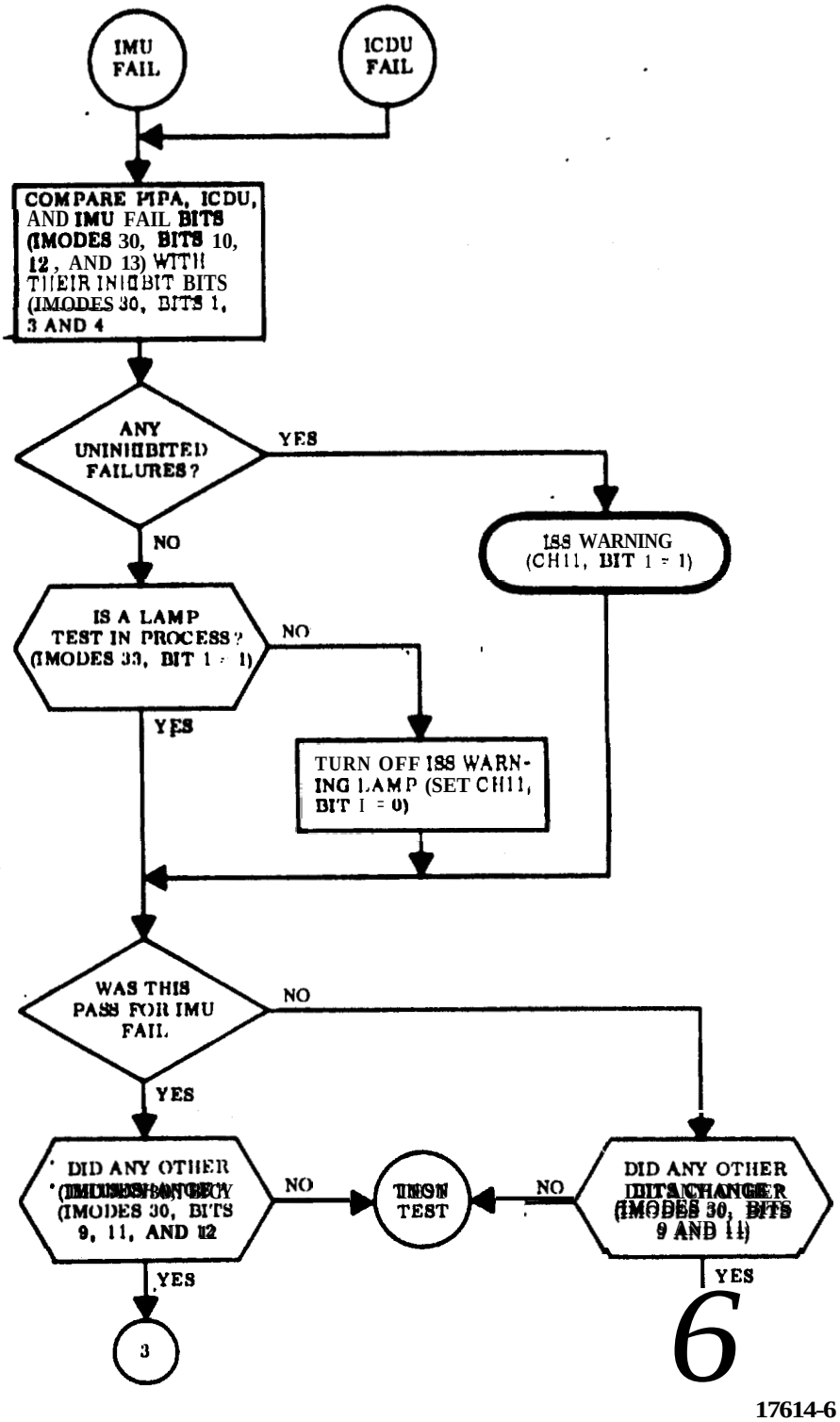


Figure 3-3. Detailed T4RUPT (Sheet 17 of 28)

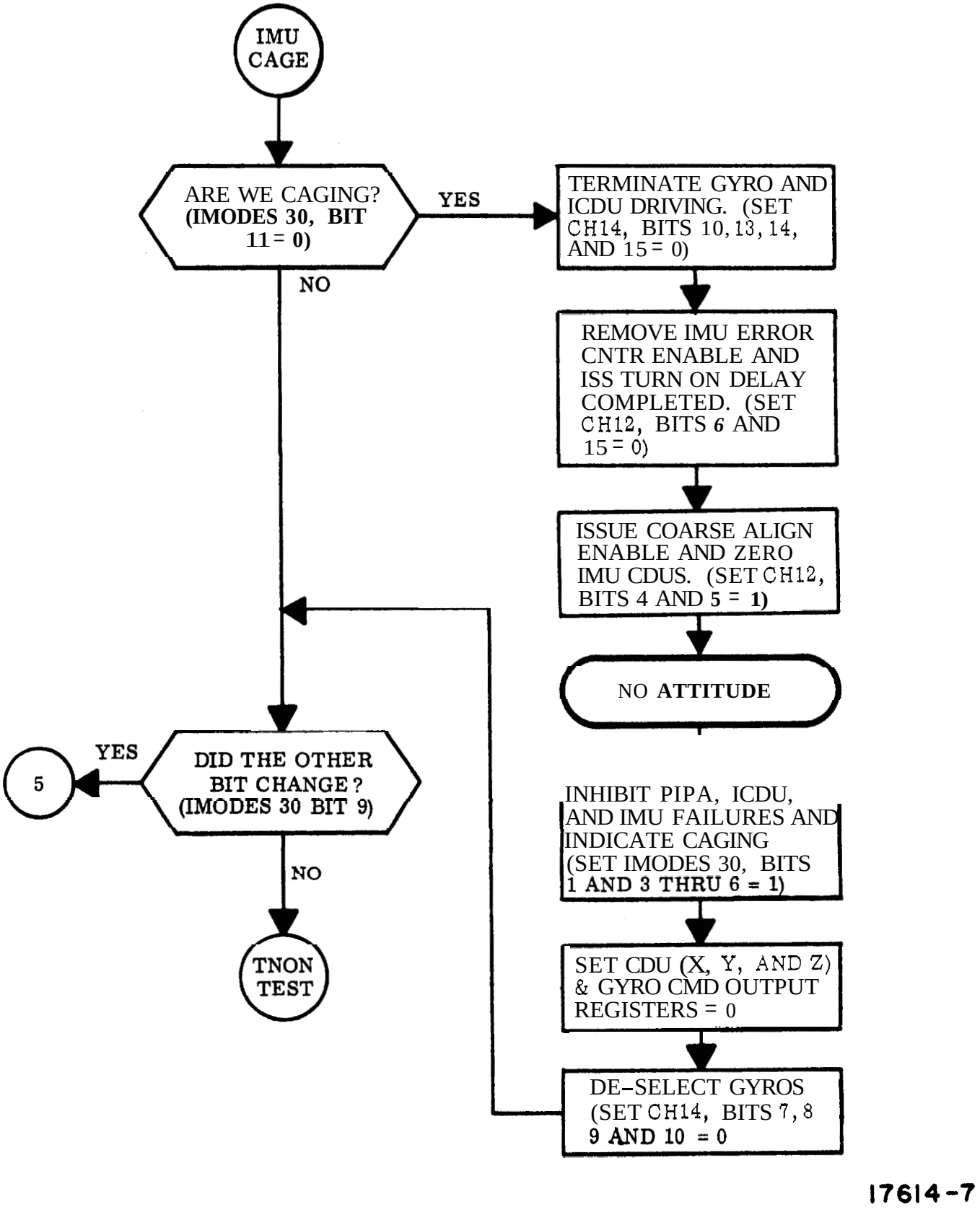


Figure 3-3. Detailed T4RUPT (Sheet 18 of 28)

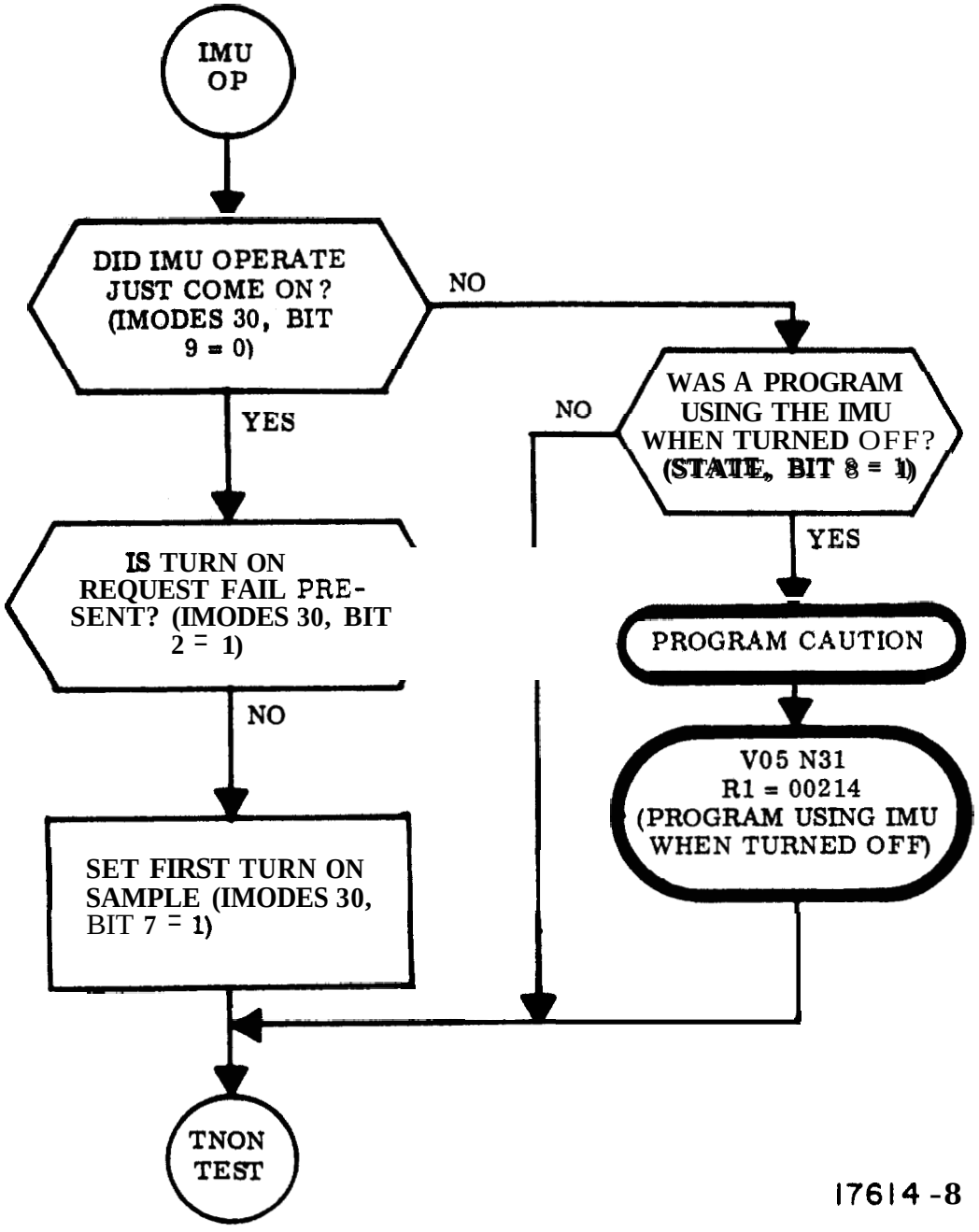


Figure 3-3. Detailed TBRUPT (Sheet 19 of 28)

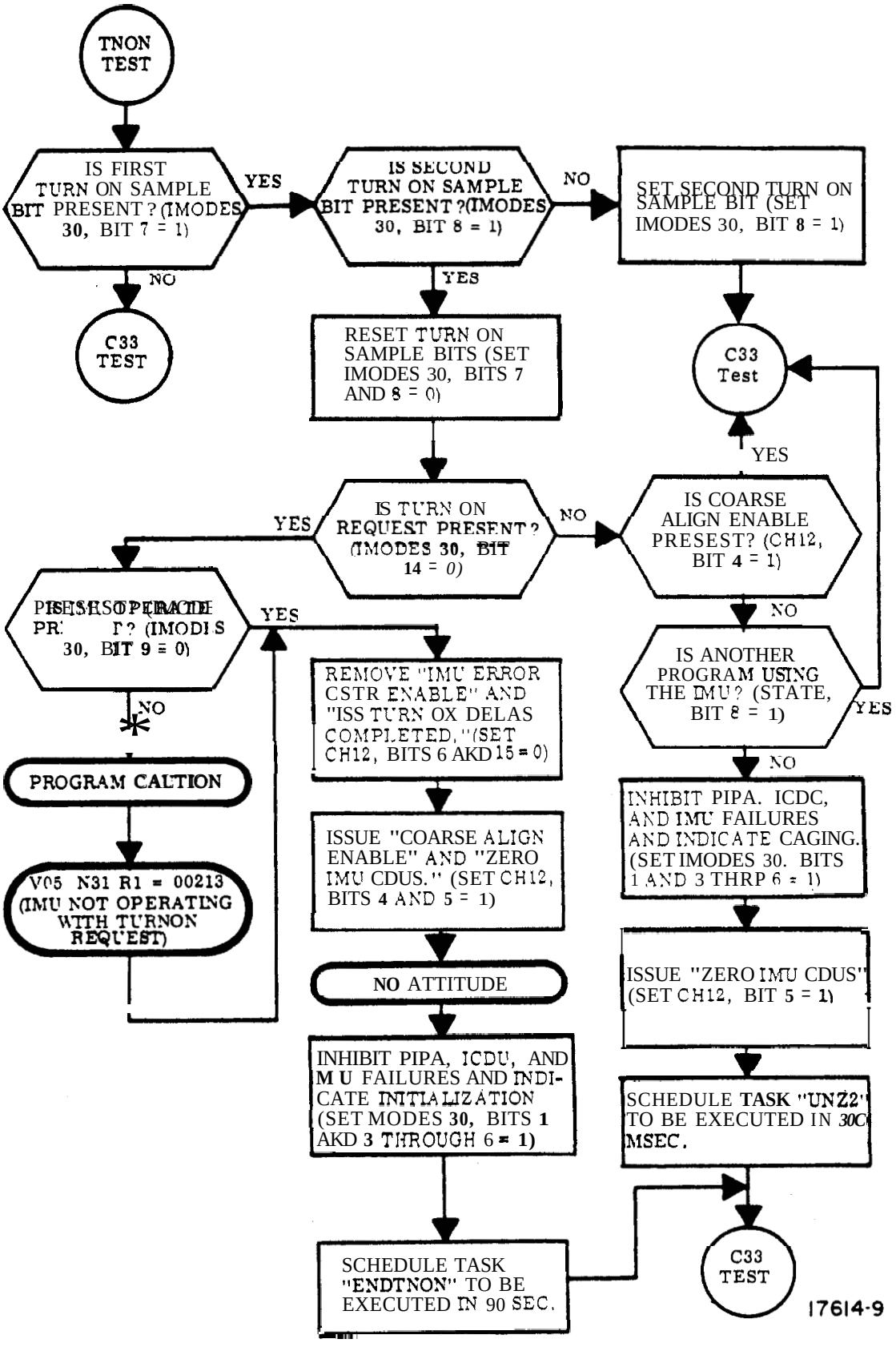


Figure 3-3. Detailed T4RUPT (Sheet 20 of 28)

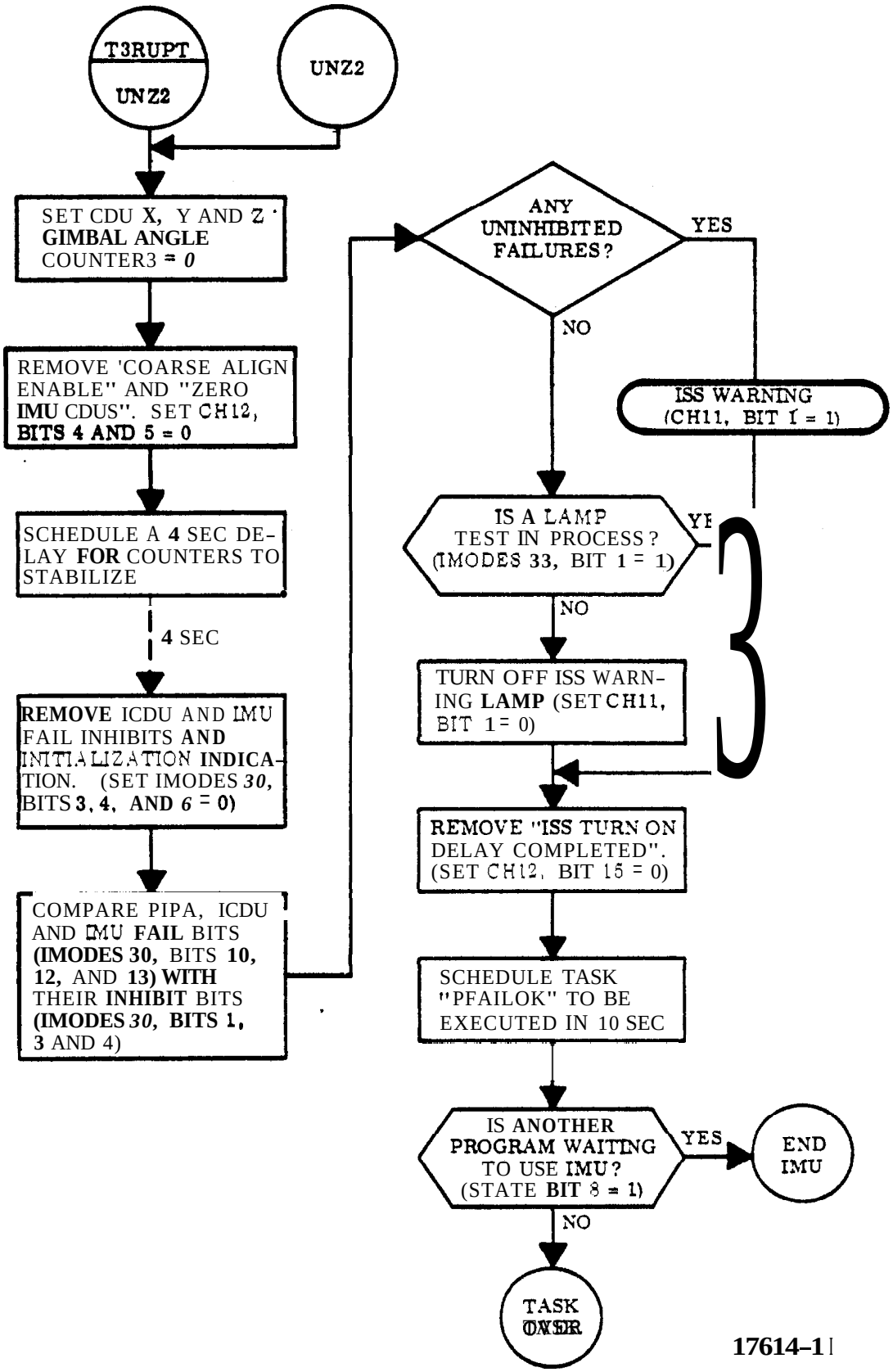


Figure 3-3. Detailed T4RUPT (Sheet **21 of 28)**

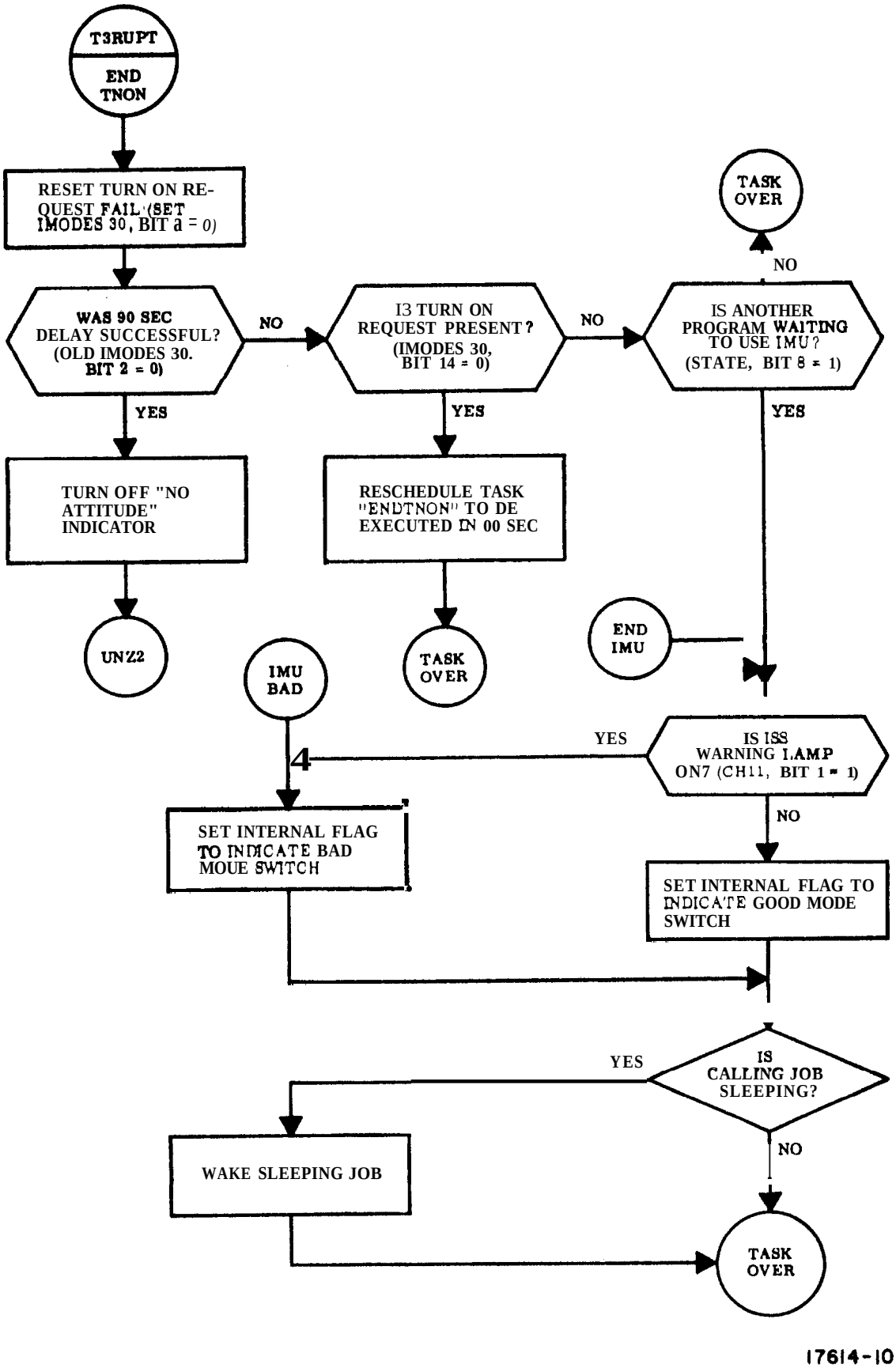


Figure 3-3. Detailed T4RUPT (Sheet 22 of 28)

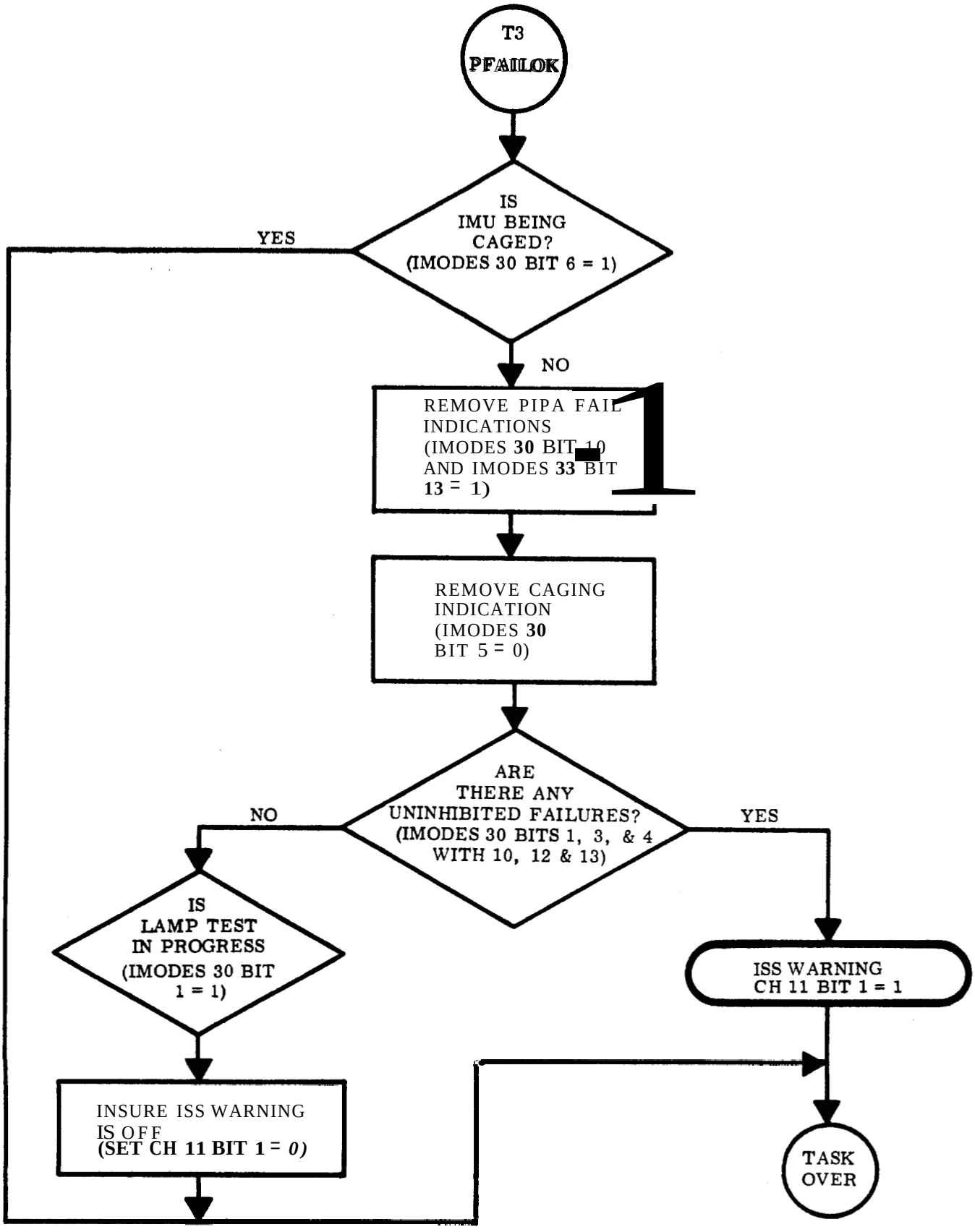


Figure 3-3. Detailed T4RUPT (Sheet 23 of 28)

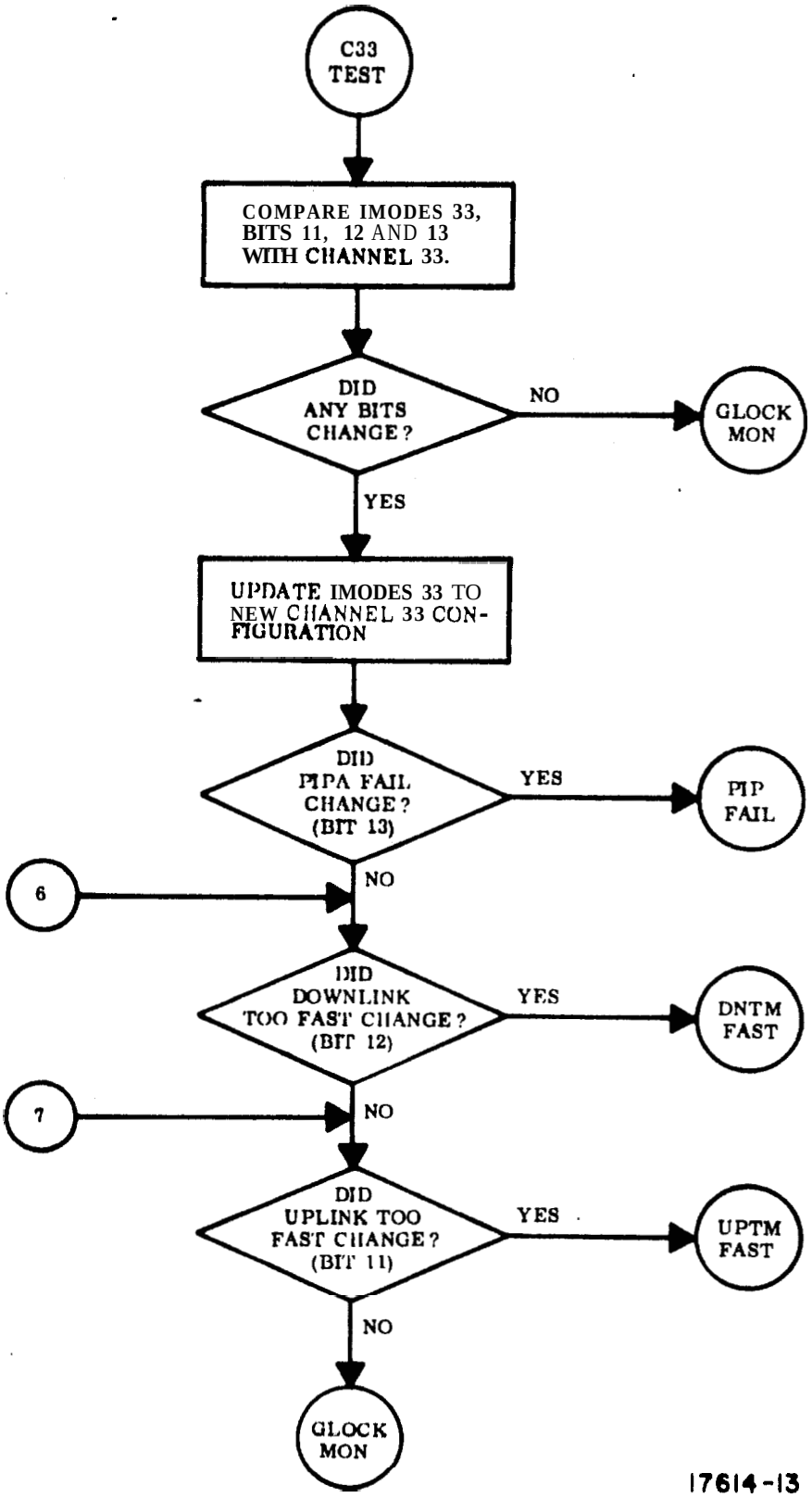


Figure 3-3. Detailed T4RUPT (Sheet 24 of 28)

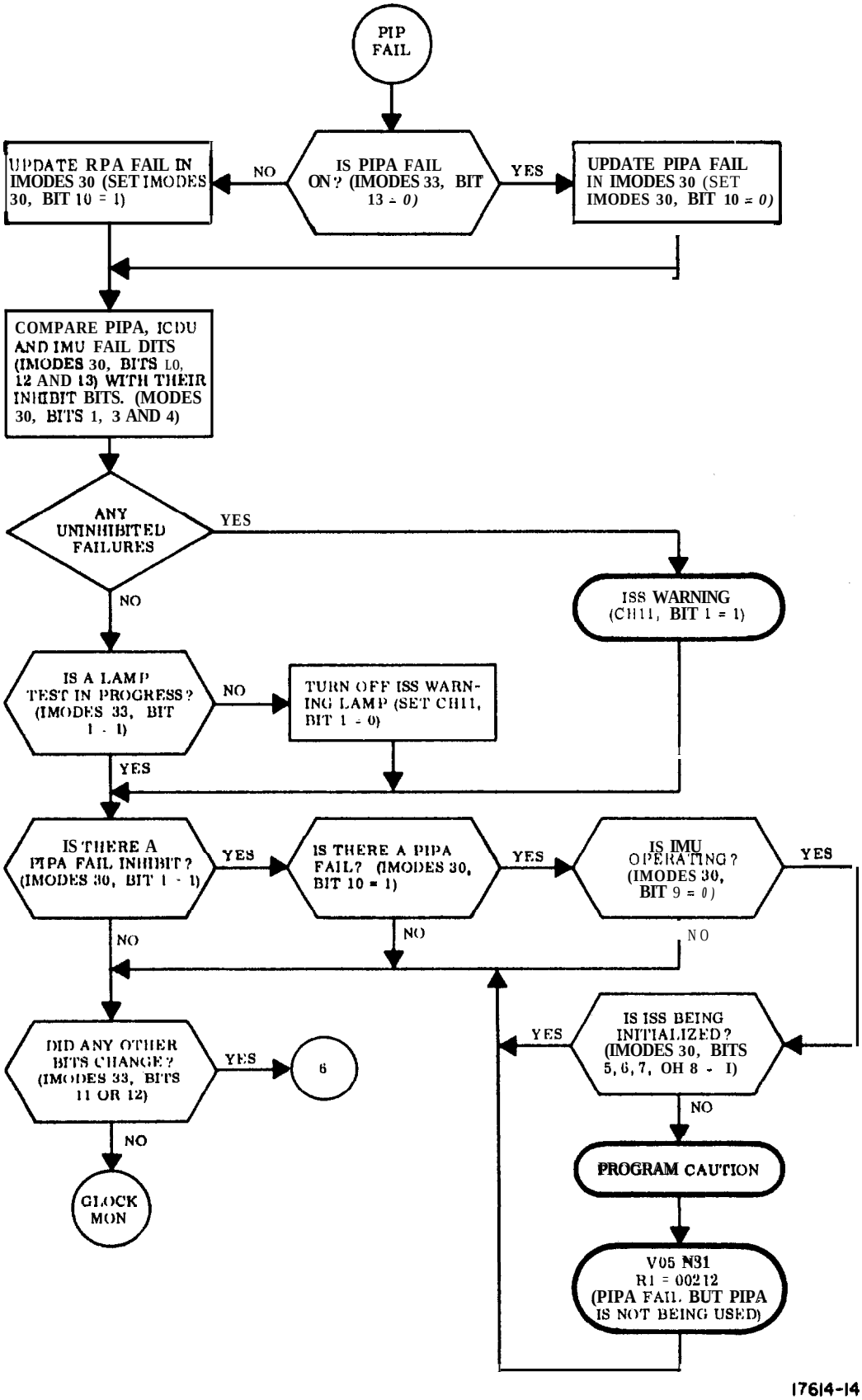


Figure 3-3. Detailed T4RUPT (Sheet 25 of 28)

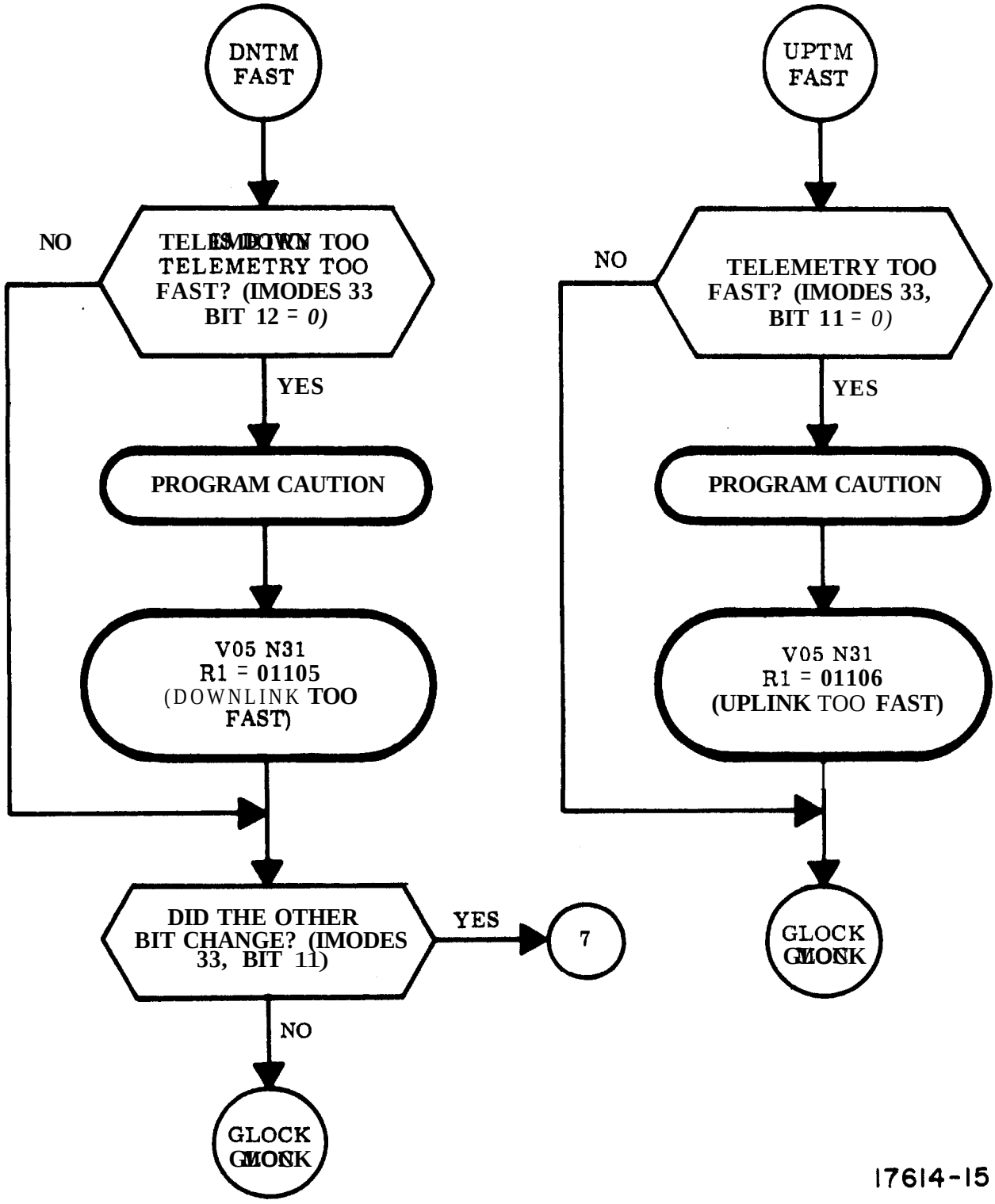


Figure 3-3. Detailed T4RUPT (Sheet 26 of 28)

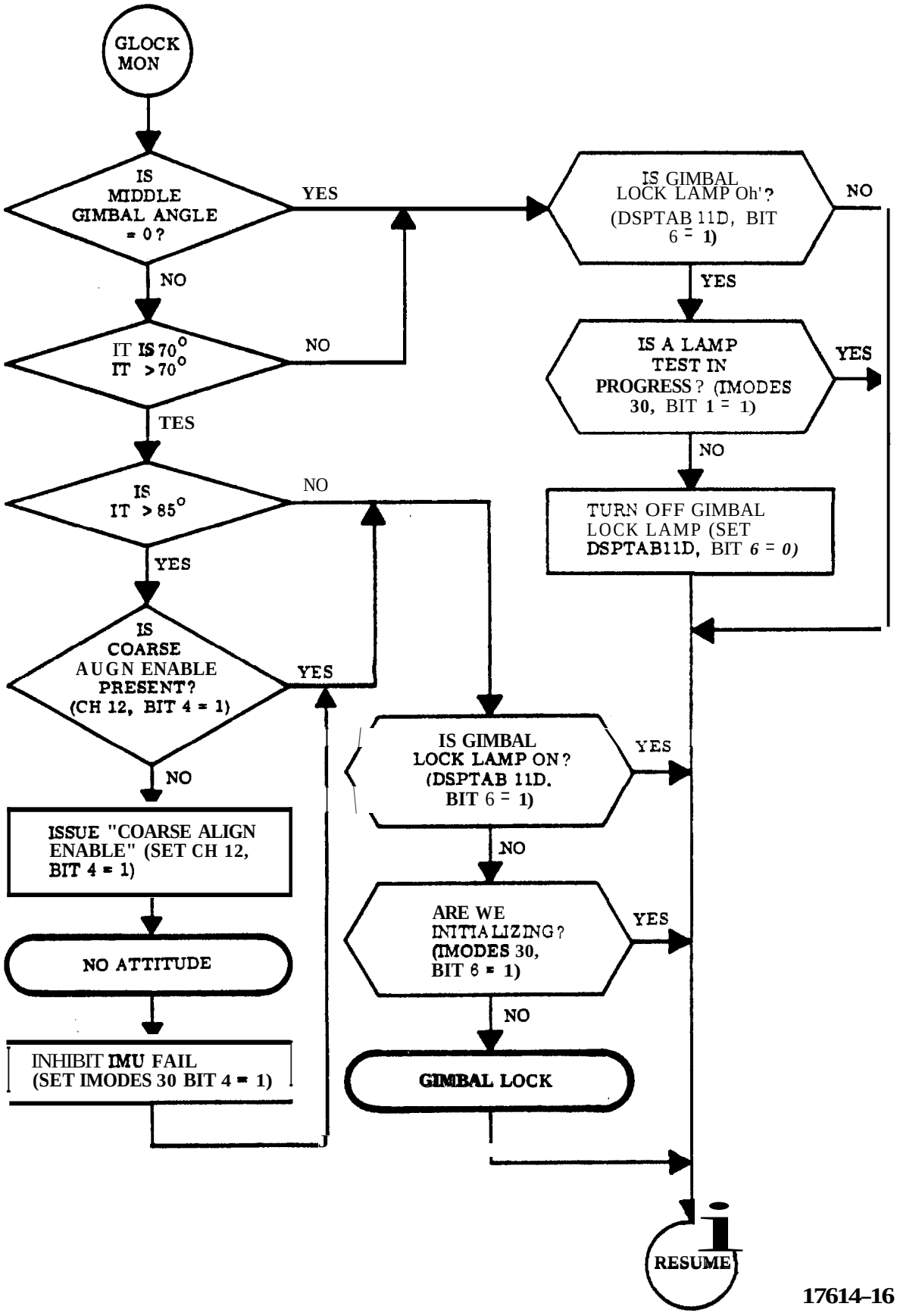


Figure 3-3. Detailed T4RUPT (Sheet 27 of 28)

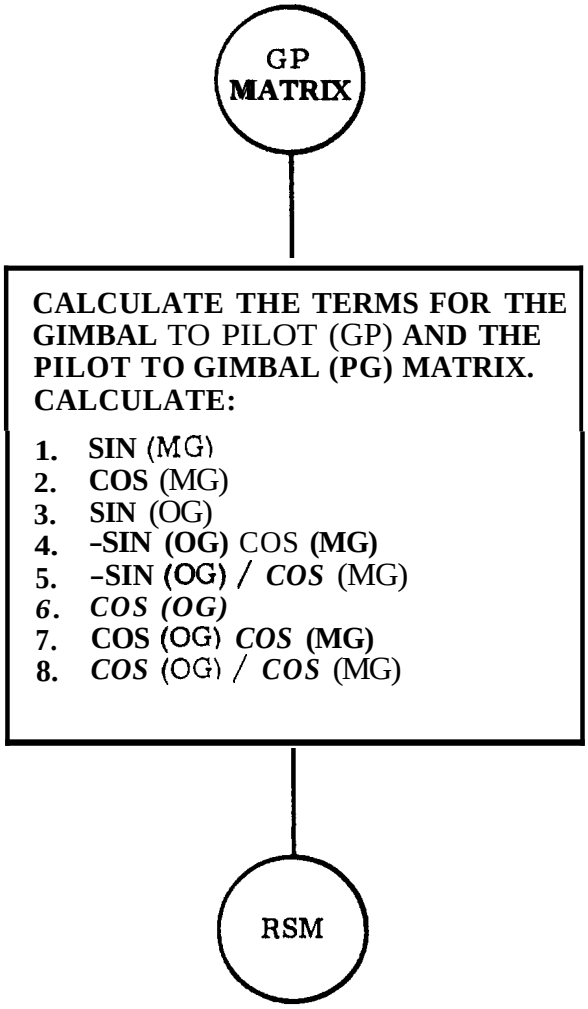


Figure 3-3. Detailed T4RUPT (Sheet 28 of 28)

Table 3-2. RADMODES - Channel Correlation

CHANNEL *30*	RADMODES	CHANNEL **33**
BIT / DESCRIPTION	DESCRIPTION	DESCRIPTION
1	RR TURN-ON	
2	RR AUTO MODE	RR POWER ON/AUTO
3	RR HI SCALE	RR RANGE LOW SCALE
4	RR DATA	RR DATA GOOD
5	LR POSITION DATA	LR DATA GOOD
6	LR ANT. POSITION # **2**	LR ANT. POSITION # **1**
7 **RR** CDU FAIL	RR CDU FAIL	
8	LR VEL DATA FAIL	LR VEL DATA GOOD
9	LR HI SCALE	LR RANGE LOW SCALE
10	DESIGNATE	
11	MONITOR REPOSITION	
12	RR MODE (ANT)	
13	RR ZERO	
14	REMODE	
15	CONTINUOUS DESIGNATE	

MARK **STAT**

BIT	DESCRIPTION
1	VAC ADDRESS
2	
3	
4	
5	
6	
7	
8	
9	
10	X MARK MADE
11	Y MARK MADE
12	MARKS ACCEPTED
13	MARK REJECT
14	INFLIGHT
15	

Table 3-3. IMODES 30 - Channel 30 Correlation

IMODES 30		Channel 30 (Inverted Logic)	
Bit	Description	Bit	Description
1	PIPA FAIL INHIBIT (ISS WARNING)		
2	TURN ON REQUEST FAIL		
3	ICDU FAIL INHIBIT		
4	IMU FAIL INHIBIT		
5	PIPA FAIL INHIBIT (PROG. CAUTION)		
6	IMU BEING INITIALIZED		
7	ISS INITIALIZATION REQUEST		
8	ISS INITIALIZATION WAIT 1 SAMPLE		
9	$\overline{\text{IMU OPERATING}}$	9	IMU OPERATE
10	$\overline{\text{PIPA FAIL}}$		
11	$\overline{\text{IMU CAGE}}$	11	IMU CAGE
12	$\overline{\text{ICDU FAIL}}$	12	ICDU FAIL
13	$\overline{\text{IMU FAIL}}$	13	IMU FAIL
14	$\overline{\text{ISS TURN ON REQUEST}}$	14	ISS TURN ON REQUEST
15	$\overline{\text{ISS TEMP IN LIMITS}}$	15	ISS TEMP IN LIMITS

Table 3-4. IMODES 33 - Channel 33 Correlation

IMODES 33		Channel 33 (Inverted Logic)	
Bit	Description	Bit	Description
1	LAMP TEST IN PROGRESS		
2	GYRO SCALE FACTOR TEST		
11	$\overline{\text{UPLINK TOO FAST}}$	11	UPLINK TOO FAST
12	$\overline{\text{DOWNLINK TOO FAST}}$	12	DOWNLINK TOO FAST
13	$\overline{\text{PIPA FAIL}}$	13	PIPA FAIL

3.2 DOWNTELEMETRY ROUTINE (DNRUPT)

The Downtelemetry Program is used to select the appropriate computer words to be transmitted via the Downlink Telemetry System and setting the selected words into channels **34** and **35.**

The overall synchronization of the downlink is obtained from the external telemetry programmer. In order to understand the sequence of events occurring during this program, one must know terminology and control pulse rates. The following is a synopsis of terms and data required for a discussion of this program:

Telemetry Word = **8** bits

Two telemetry words = **1** computer word

1 Prime Frame = **128** telmetry words

5 Telemetry Words = **1** DP Computer Data Word

*BIT RATE = **51.2 KC** or **1.6 KC**

*TL START PULSE = **50** pps or **10** pps

*TL END = **50** pps or **10** pps

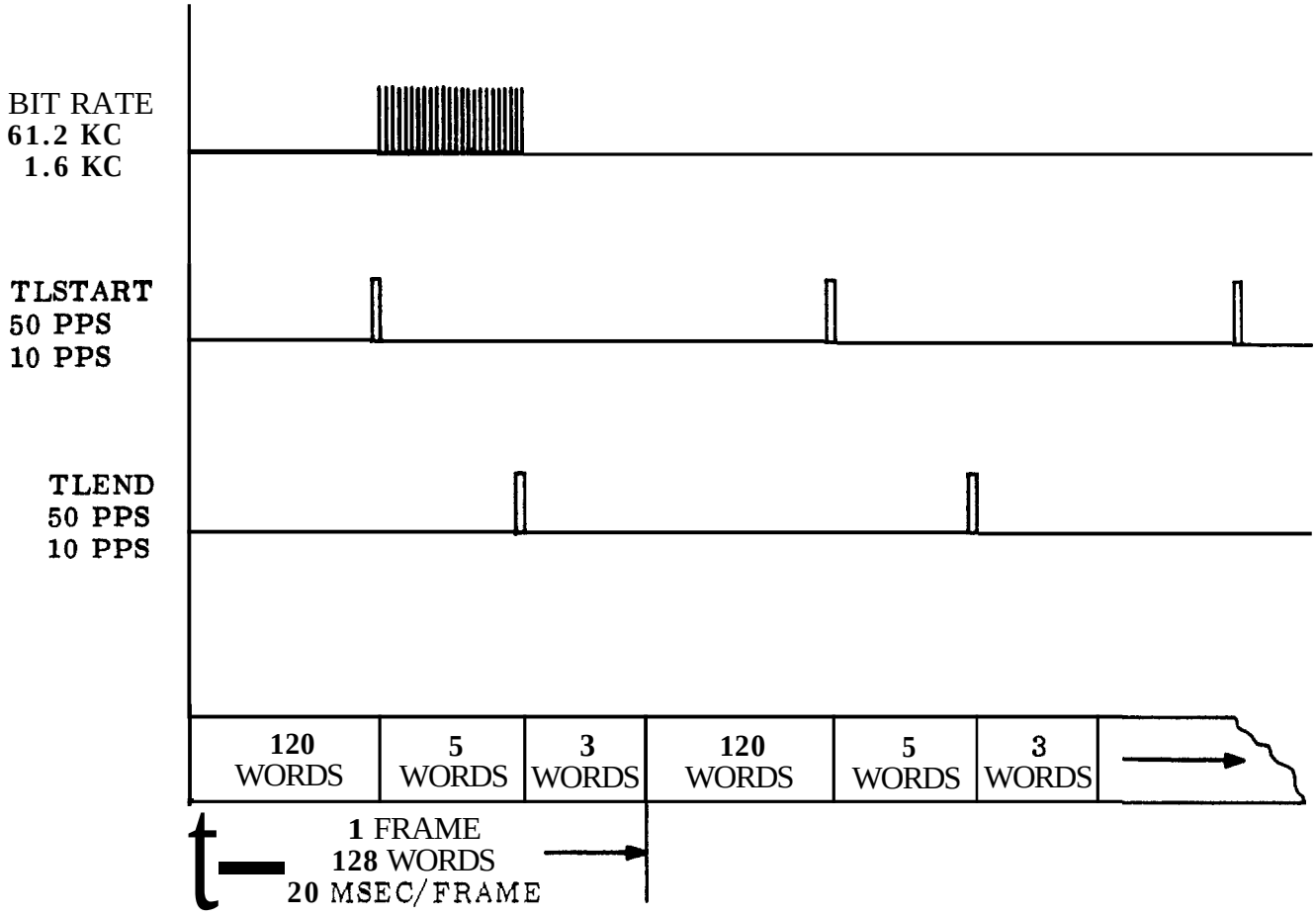
*NOTE: The *two* rates given are basic rates, the lower of which is used to transmit data when the spacecraft is located a considerable distance from the ground tracking station. Our discussion will be based on the higher **of** the two rates.

The telemetry system transmits data to the ground at the rate of **50** frames per sec. A frame consists of **128** eight bit telemetry words, five of which are allotted for computer data. **Thus,** every *20* msec computer data is sent out (figure **3-4).** At the high rate *50* DP computer words are transmitted each second, with some single precision words grouped together and sent two at a time as double precision pairs,

If one considers the five telemetry words (figure **3-5)** to be repeated **50** times per second and each transmission to contain different data, one would have a picture of what the down telemetry program must accomplish. A general format is shown in figure **3-6.**

The nominal downlink list for Sunburst, Rev **14 is** provided as figure **3-7.**

The downrupt **flow** diagram **is** shown in figure **3-8.** The arrival of the TELEND pulse will cause the program interrupt service to route control **of** the computer to the DNRUPT routine. Each pass through the DNRUPT routine will load output channels **34** and **35** with the next two words to be outputed via the telemetry system. These two words will picked from these channels upon occurrence **of** the next TELSTART pulse. The FRESHSTART/ RESTART routine initializes the contents of DNTMGOTO to the starting address of DNPHASE **1** *so* that the **first** TELEND pulse will cause a DNRUPT which will transfer the control of the computer to the routine defined **as** DNPHASE **1.**



The arrival **of** the TLEND pulse forces a DNRUPT which transfers control to the DODOWNTM program. This program loads the next double-precision data scheduled *to* be sent down into channels **34** and **35,** and sets bit **7** of channel **13** (Word Order Bit) *to* a **ZERO** for transmission **of** the other **49** DP words of the **50** DP word format. Channel **13,** bit **7,** controls the logical state **of** the leading bit (word order bit) in each **40** bit transmission ***so*** that the first **40** bit word transmitted is distinct from the remaining **49.** Bits **2-33** consist of the contents **of** channels **34** and **35;** bits **34-40** are all zeros.

Figure **3-4.** Computer Interface with Telemetry

Note: Time is increasing to the left and that the computer words are outputted serially from high to low order bit position.

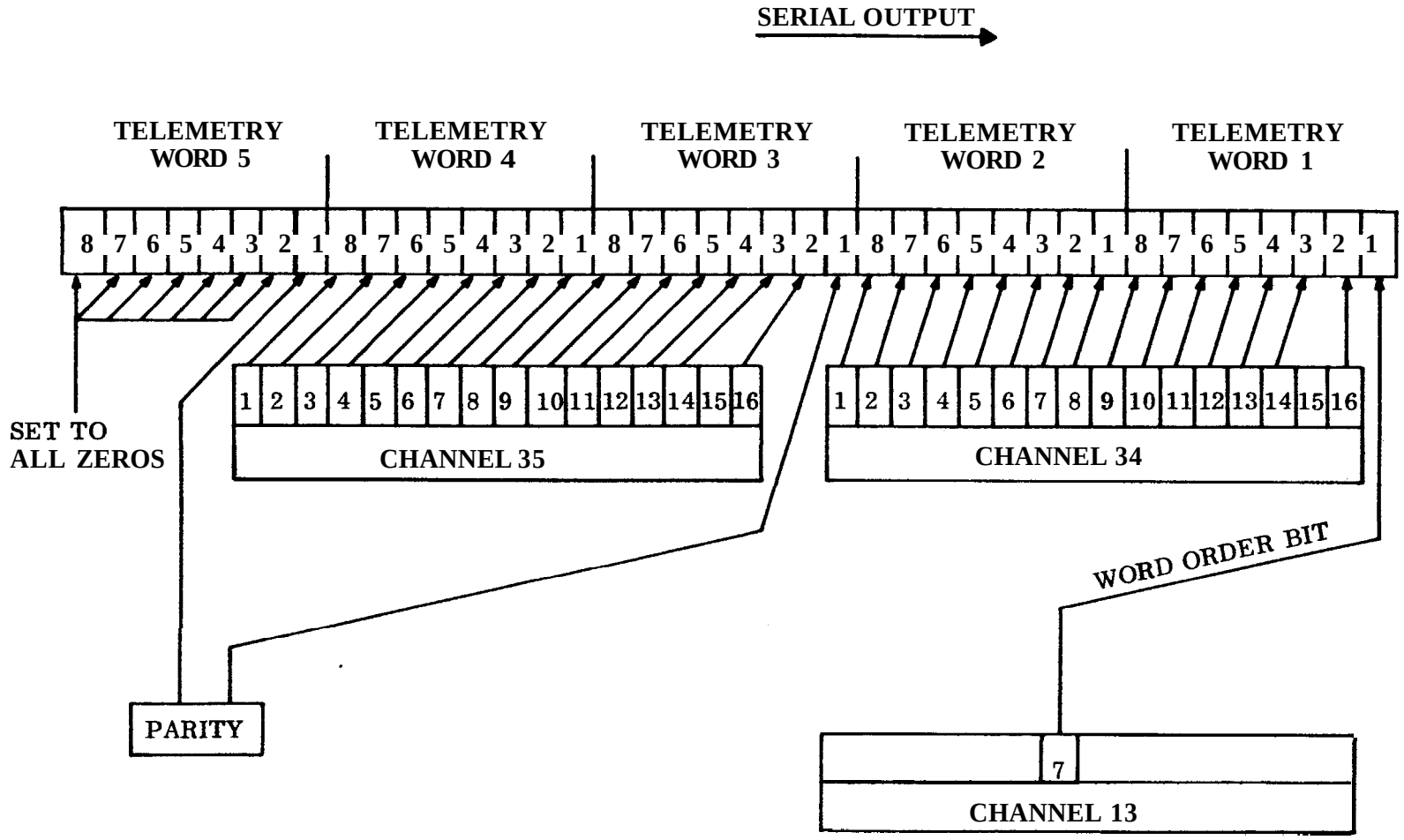


Figure 3-5. Downtelemetry Transfer

	LAST 7 BITS	CHANNEL 35	CHANNEL 34	WORD ORDER BIT
WORD #1	*0000000*	00437,	00000 (LIST START ADDRESS)	0
2	*0000000*	38_{10} COMPUTER WORDS DEFINED BY DATA **LIST**	38_{10} COMPUTER WORDS DEFINED BY DATA LIST	**1** **1** **1** **1** **1** **1** **1** **1** **1** **1** **1** **1** **1** **1** **1** **1**
39				
40	*0000000*	DSPTAB+1	DSPTAB	**1**
41		DSPTAB+3	DSPTAB+2	**1**
42		DSPTAB+5	DSPTAB+4	**1**
43		DSPTAB+7	DSPTAB+6	**1**
44		DSPTAB+9	DSPTAB+8	**1**
45		DSPTAB+11D	DSPTAB+10D	**1**
46		TIME **1**	TIME 2	**1**
47	*0000000*	CHANNEL **12**	CHANNEL **11**	**1**
48		CHANNEL 14	CHANNEL 13	**1**
49		CHANNEL **31**	CHANNEL **30**	**1**
50		CHANNEL 33	CHANNEL 32	**1**
		R E P E A T		

Figure **3-6.** Downtelemetry General Computer Format

		CHANNEL 34	CHANNEL 35
DNPHS 1	1	00000 (LIST ADDRESS)	00437_8
	2	PIPA X	PIPA *Y*
DNPHS X AND DNPHS 2 FROM SNAPSHOT BUFFER	3	PIPA Z	RHCP
	4	RADMODES	SAMPLIM
	5	OLDATAGD	DESCOUNT
	6	SAMPLSUM	SAMPLSUM+ 1
	7	OPTYHOLD	OPTYHOLD+ 1
	8	TIMEHOLD	TIMEHOLD+ 1
	9	ALT	ALT+ 1
	10	ALTRATE	FMALT
	11	ALTSAVE	ALTSAVE + 1
	12	FINALT	FINALT+ 1
	13	FOHVEL	LATVEL
DNPHS 3	14	CDU X	CDU *Y*
	15	CDU Z	OPT *Y*
	16	OPT *Y*	OPT X
	17	STATE	STATE + 1
	18	STATE+ 2	STATE+3
	19	REDOCTR	FAILREG
	20	LMPCMD	LMP CMD +1
	21	LASTYCMD	LASTXCMD
	22	TANG	TANG+ 1
	23	THETAD	THETAD+ 1
	24	THETAD+ 2	DELVX
	25	MARKSTAT	XYMARK

Figure 3-7. Nominal Downlink List, Sunburst, Rev 14 (Sheet 1 of 2)

		CHANNEL 34	CHANNEL 35
DNPHS 3 (cont)	26	DRIFT O	DRIFT O + 1
	27	DRIFT I	DRIFT I+ 1
	28	DRIFT T	DRIFT T+ 1
	29	INTY	INT Y+l
	30	ANG Z	ANG Z + l
	31	ANG Y	ANG Y+ 1
	32	ANC X	ANG X+ 1
	33	VLAUN	VLAUN + 1
	34	VLAUN+4	VLAUN+5
	35	DATA PL	DATA PL+1
	36	DATA PL+2	DATA PL+3
	37	DATA PL+4	DATA PL+5
	38	DATA PL+6	DATA PL+7
	39	UPLOCK	T6LOC
DNPHS 4	40	DSPTAB	DSPTAB+1
	41	DSPTAB+2	DSPTAB+3
	42	DSPTAB+4	DSPTAB+5
	43	DSPTAB+6	DSPTAB+7
	44	DSPTAB+8D	DSPTAB+9
	45	DSPTAB+10D	DSPTAB+11
	46	TIME 2	TIME 1
DNPHS 5	47	c(CHAN 11)	c(CHAN 12)
	48	c(CHAN 13)	c(CHAN 14)
	49	c(CHAN 30)	c(CHAN 31)
	50	c(CHAN 32)	c(CHAN 33)

Figure 3-7. Nominal Downlink List, Sunburst, Rev 14 (Sheet 2 of 2)

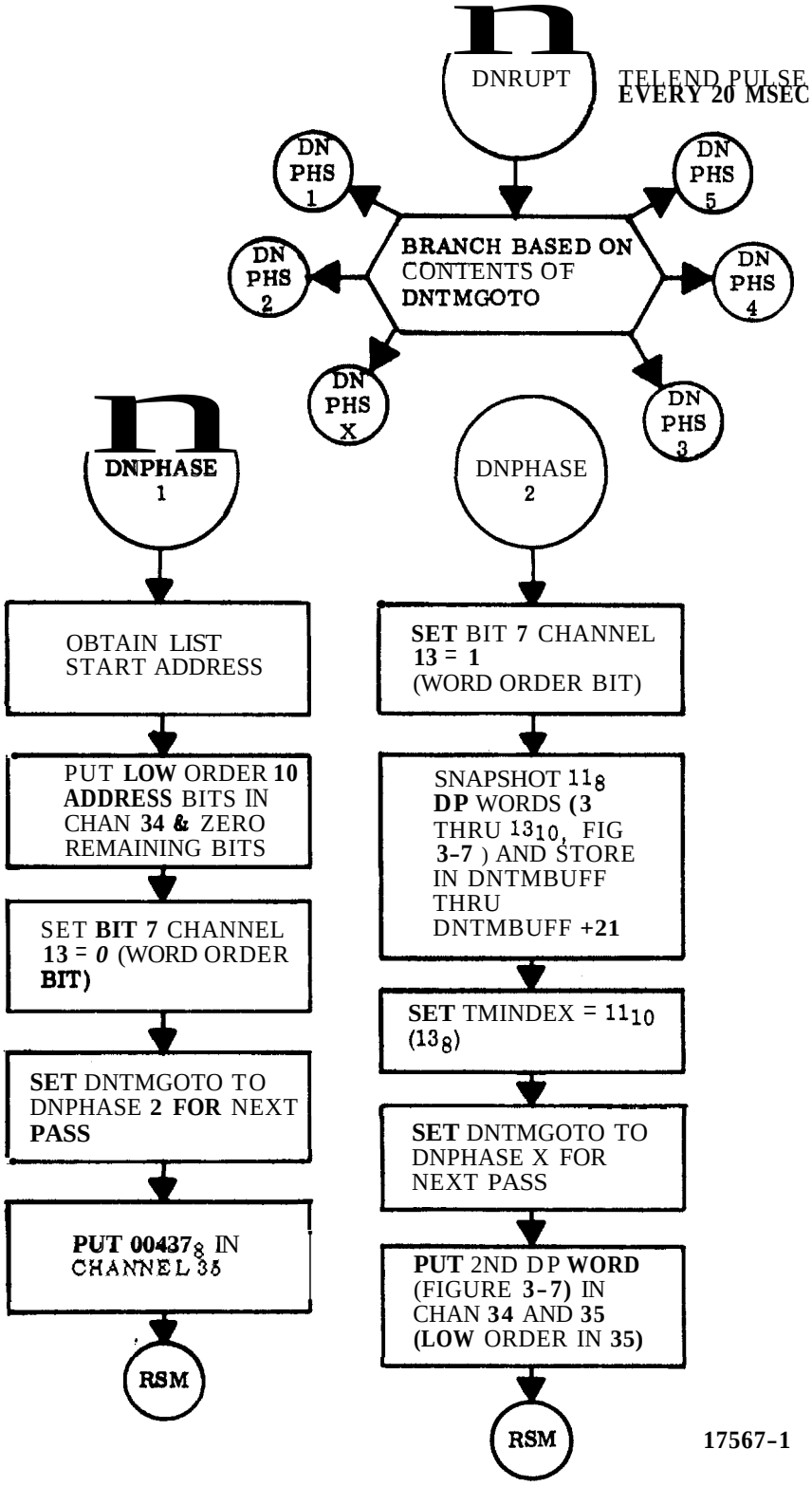


Figure 3-8. Downrupt (Sheet 1 of 2)

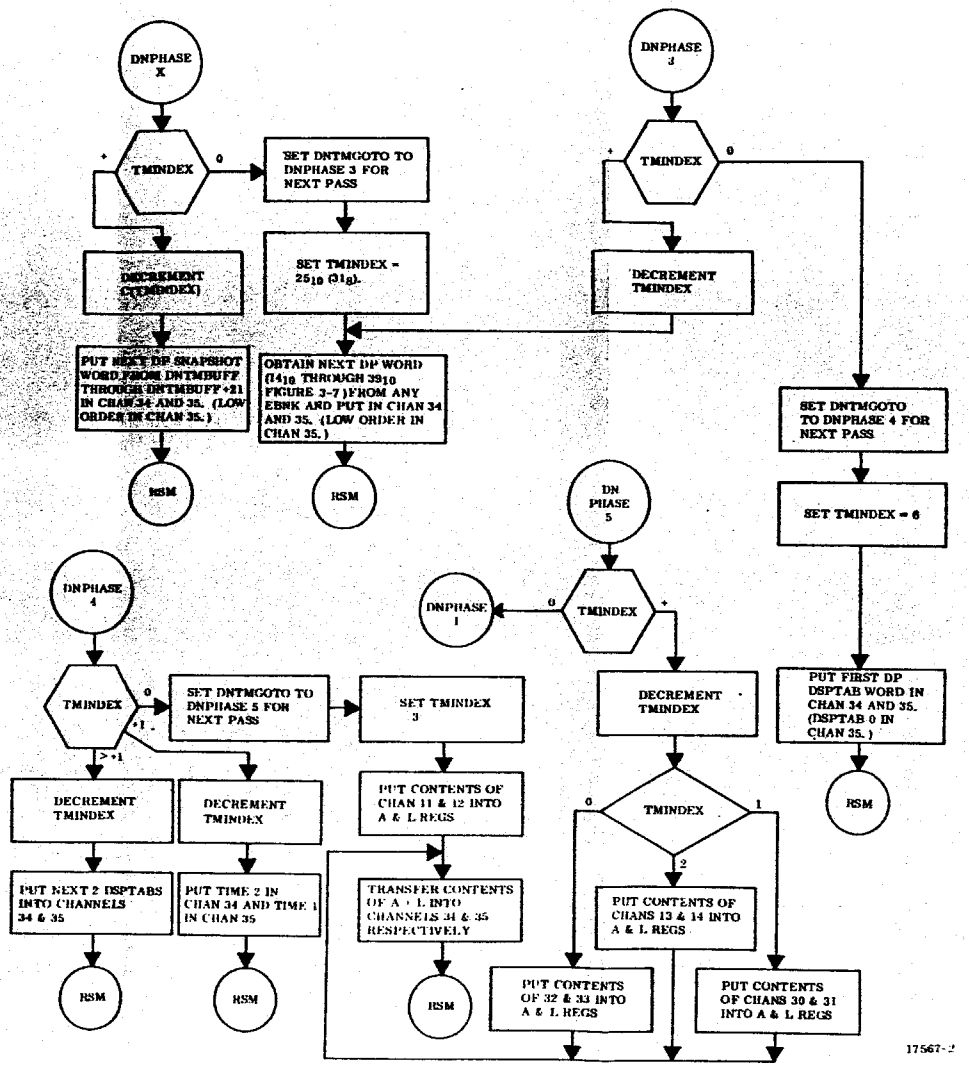


Figure 3-8. Downrupt (Sheet 2 of 2)

DNPHASE 1 - DNPHASE 1 will set the ID word into channels 34 and 35. This ID word is composed of the downlink list starting address in channel 34 and octal 00437 in channel 35. DNPHASE 1 will also set the word order bit to zero to indicate that the word in channels 34 and 35 is the ID word. DNTMGOTO is set up so that the next DNRUPT will transfer control to DNPHASE 2.

DNPHASE 2 - DNPHASE 2 sets the word order bit to 1 to indicate that the contents of channels 34 and 35 is ID WORD. DNTMGOTO is set to route computer control to DNPHASE X on the next DNRUPT. DP words 3 through 13 are obtained and stored in buffer registers and DP word 2 is inserted into channels 34 and 35. The words stored in the buffer will be loaded into channels 34 and 35 by the next eleven DNRUPT routines. They are sampled and stored at this time so that when interpreted on the ground they can be compared to each other with reference to the same time frame.

DNPHASE X - DNPHASE X merely obtains the next sequential snapshot word and puts it in channels 34 and 35 to be plucked upon receipt of the next TELSTART pulse.

DNPHASE 3 - DNPHASE 3 sets up DNTMGOTO for DNPHASE 4 on the next DNRUPT and sets the first DP DSPTAB word in channels 34 and 35.

DNPHASE 4 - DNPHASE 4 sets the remaining DSPTABS, TIME 2, and TIME 1 and channels 11 and 12 into channels 34 and 35.

DNPHASE 5 - DNPHASE 5 sets the remaining words of the downlink list into channels 34 and 35 in the following order: channels 13 and 14; channels 30 and 31; and channels 32 and 33. The next entry into DNPHASE 5 will route control of the computer to DNPHASE 1 to begin the second transmission of the downlink list.

3.3 KEYBOARD AND UPLINK TELEMETRY INPUT PROCESSING PROGRAM

The keyboard and uplink telemetry processing program includes the KEYRUPT routine, the UPRUPT routine and the routines of the PINBALL program. The KEYRUPT and UPRUPT routines accept the input keycodes from the keyboard and uplink telemetry system. The PINBALL program assebles the accepted inputs into meaningful information and controls the execution of that which is indicated by the assembled information. Besides performing the functions mentioned above, the PINBALL program can be used by internal programs to perform the various functions which can be performed through keyboard or uplink inputs.

The KEYRUPT and UPRUPT routines operate under control of the program interrupt circuitry of the computer and the processing specified by them is initiated whenever the appropriate input is present, The PINBALL program is executed under control of the Executive program and is scheduled by the KEYRUPT or UPRUPT routines to process the keycodes that they accept. When an internal program uses the PINBALL program, it is processed under the scheduling of the internal program using one or more of its routines.

Basically, the PINBALL program is made up of a group of routines with a routine for each of the types of keys on the DSKY's. There is a NUM, NOUN, VERB, CLEAR, SIGN, ERROR RESET, KEYRELEASE, and an ENTER routine. In addition to these routines, there are numerous other routines which control is transferred to by the ENTER routine which perform the requested action.

Figure 3-9 is a general flow diagram for the PINBALL program and the related KEYRUPT and UPRUPT routines. Inputs are accepted by the KEYRUPT routine from the computers DSKY or by the UPRUPT routine from a ground based keyboard via the uplink telemetry system. Both of these routines accept the keycode input and schedule the CHARIN (CHARACTER INPUT) routine of the PINBALL program to be executed through the Executive routine.

Because of the relatively high priority assigned to the Job CHARIN, the processsing of the input keycode begins quickly. CHARIN routes control to the proper routine depending on the keycode input. If the NOUN key is depressed, control is routed to the NOUN routine, etc. After the processing required for a keycode input is completed, control is routed to the Executive End Job routine to terminate the PINBALL program. When another key is depressed, the KEYRUPT or UPRUPT routine schedules CHARIN which routes control to the proper routine, the processing is performed and the PINBALL job is terminated. This sequence continues until the required information has been loaded. When the final ENTER key is depressed, this sequence occurs again except that the ENTER routine routes control to another routine which performs the specified action. When the processing required is completed, control is routed to the Executive End of Job routine.

The internal programs of the computer supply the appropriate information to the ENTER routine and can thereby utilize any of the routines which are used as a result of keyboard or uplink inputs. Prior to transferring control to the ENTER routine to utilize one or more of the routines accessible through the ENTER routine, the internal program must make sure that there is not another internal program or that the astronaut is not using the PINBALL program. If the PINBALL program is already in use, control is returned to the internal program. What is accomplished then is up to the internal program but, in most cases, the internal program is put to sleep until the PINBALL routine is available for its use.

3.3.1 DSKY AND UPLINK INTERRUPT OPERATION. The keyboard and uplink telemetry interrupt processing program includes the KEYRUPT and the UPRUPT routines. These routines process the keycodes which will be used by CHARIN. The KEYRUPT and UPRUPT routines accept the input keycodes from the keyboard and uplink telemetry system. PINBALL assembles the accepted inputs into meaningful information and controls the execution of that which **is** indicated **by** the assembled information. ***The*** KEYRUPT and UPRUPT routines operate under control of the computer and processing specified by them is initiated whenever the appropriate input is present.

The KEYRUPT and UPRUPT routines accept input keycodes from the DSKY keyboard or from a ground based keyboard, respectively. The KEYRUPT routine obtains the five bit keycode from input channel **15** register bits **1** through **5** (MAIN DSKY) or input channel **16** bits **1** through **5** (NAV DSKY). The UPRUPT routine obtains the keycode information from the INLINK counter of the computer memory. The information obtained from the INLINK counter consists of **16** bits of data, shown in figure **3-10,** which has been received serially and manipulated into parallel form. Of the **16** bits of information, **15** bits are used for three copies of the keycode being transmitted. Bits **15** through **11** (high **5)** and **5** through **1** (low *5)* contain the keycode, while bits 10 through *6* (middle **5)** contain the complement of the keycode. The 16th bit is always a binary **1** and is used to indicate, through the computer's program interrupt circuitry, that the complete **16** bits of information have been received.

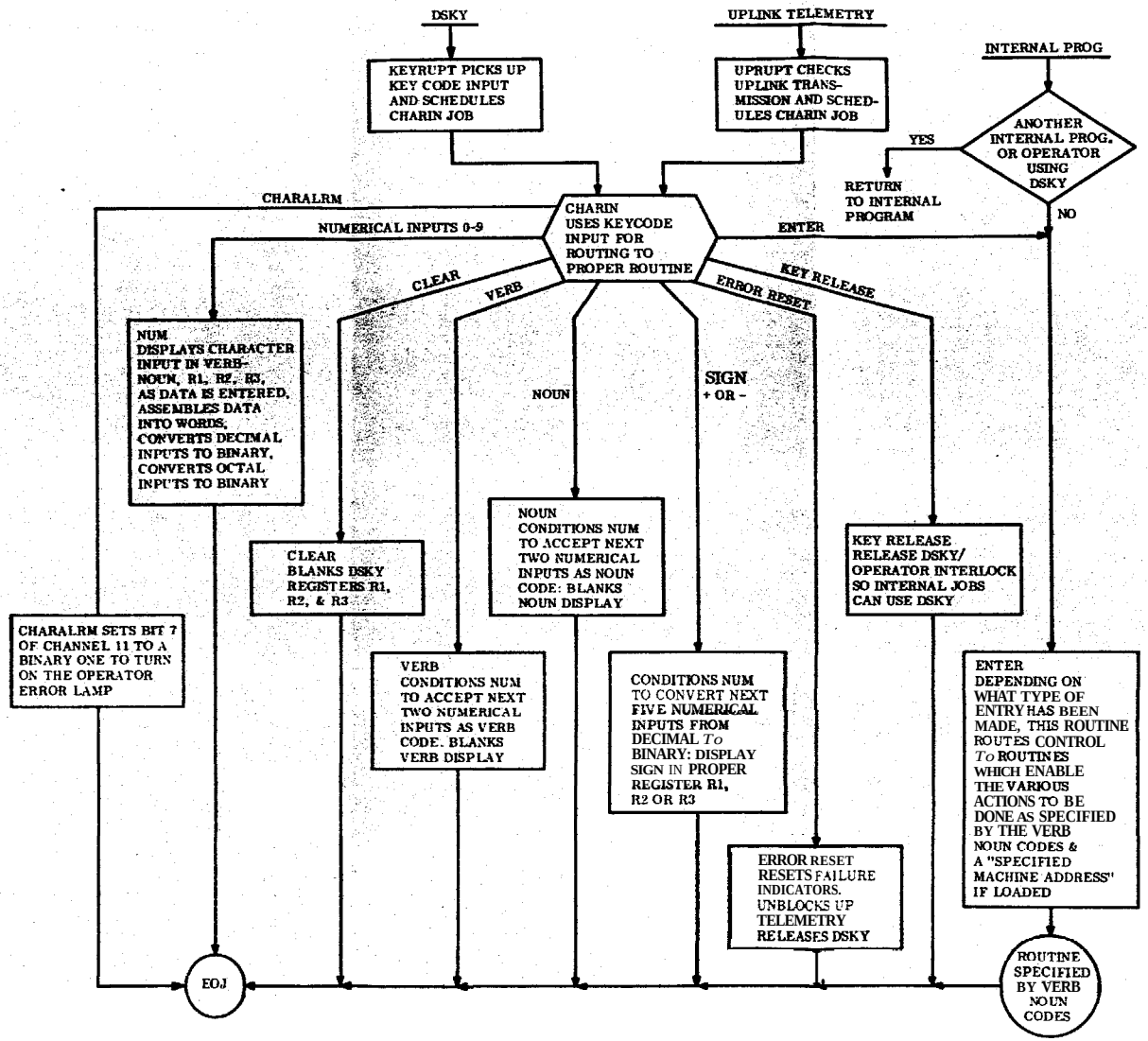
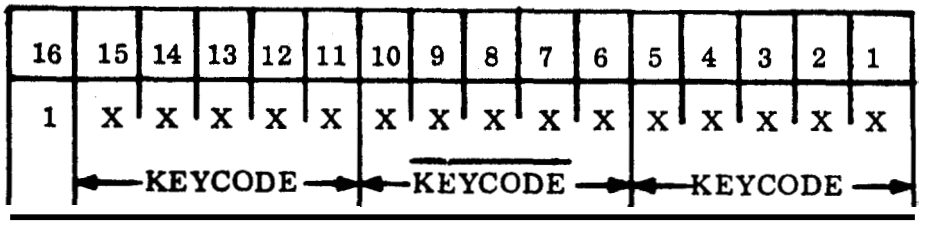


Figure 3-9. General Flow Diagram for Pinball



X CAN EQUAL A BINARY 1 OR *0*
BIT 16 ALWAYS IS A BINARY 1

Figure 3-10. INLINK Word Format

The flow chart for the KEYRUPT and UPRUPT routines are shown in figure 3-11. The processing performed by the KEYRUPT 1 routine (the KEYRUPT 2 routine is associated with mark commands) is presented here. Upon initiation of the KEYRUPT 1 routine, via keyboard entry, real time is recorded. This is accomplished by reading and storing the following computer memory registers:

a. TIME 1 COUNTER

b. TIME 2 COUNTER

These quantities are recorded in case the KEYRUPT 1 was initiated by Noun 65 (Sampled Time) and are recorded immediately to obtain the magnitudes as close to the time of the entry as possible.

The Executive's NOVAC subroutine is used to schedule the Job CHARIN. After CHARIN is scheduled, the keycode is stored in MPAC of the core set area reservied for CHARIN by NOVAC. This concludes the KEYRUPT routine and control is returned to the job which was interrupted by the KEYRUPT routine.

Whenever the UPLINK routine is initiated, real time is recorded. This is accomplished by reading and storing the contents of the TIME 2 and TIME 1 counters, These quantities are recorded in case the UPRUPT was initiated by Noun 65 for the same reason it was during KEYRUPT.

A 16 bit uplink word, which has been assembled in the computer's INLINK counter, is read from the counter and the counter is set to +0 in preparation for the receipt of the next uplink transmission. Bit 3 of channel 11 is set to a logic 1 to turn on the uplink activity lamp. The routine now checks the accuracy of the uplink transmission. This check is performed by comparing the three copies of the keycode contained in the uplink word. If all three copies do not compare, further uplink activity is locked out by placing a 1 in bit 1 of UPLOCK and the interrupted job is resumed. The uplink lock can be removed by the ground station sending an error light reset keycode or by performing a FRESH START.

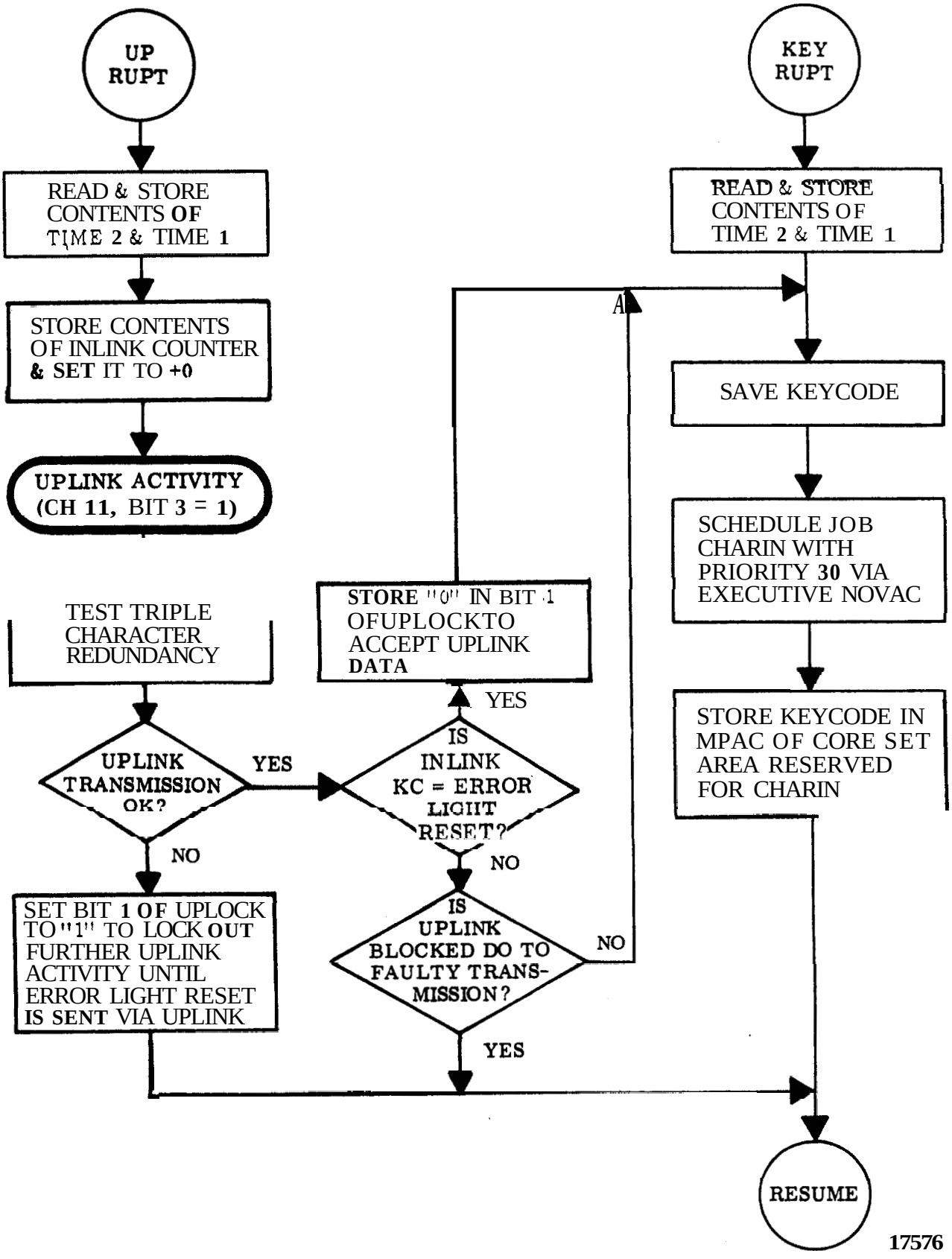


Figure 3-11. KEYRUPT **and** UPRUPT

If the uplink transmission is good, a check is made to determine if an error light reset keycode was transmitted. The 5 bit error light reset keycode is 10010. This keycode is the same as the operator error keycode of the DSKY and will perform the same functions. If the uplink word is error light reset, a 0 is placed in bit 1 of uplock to remove the uplink lock if the lock was in use.

The Executive's NOVAC subroutine is now used to schedule the Job CHARIN. After CHARIN is scheduled, the keycode is stored in MPAC of the core set area reserved for CHARIN by NOVAC and the interrupted job is resumed.

If the uplink transmission is good and the keycode is not error light reset, a check is made to determine if the uplink activity is locked out. If the uplink activity is locked out, the interrupted job is resumed. If the uplink data is not locked out, the CHARIN subroutine is scheduled as before by NOVAC and the keycode is inserted into the reserved core set area as previously noted. The interrupted job is resumed.

3.3.2 THE PINBALL PROGRAM. The PINBALL program as previously stated performs the functions of assembling information as it is entered through the keyboard of uplink telemetry system. It also initiates the proper function as indicated by the keyboard or uplink inputs. This program is divided into various subroutines which are presented in the following paragraphs.

3.3.2. 1 CHARIN. CHARIN performs a routing function to other subroutines of the PINBALL program. The routing performed by CHARIN is based on the keycode input received by the computer from either the DSKY keyboard or the uplink telemetry system. CHARIN is scheduled to be performed by either the KEYRUPT or UPRUPT routines when a keycode is accepted by these routines, and is performed under control of the Executive routine as shown in figure **3-12.**

When the Executive routine determines that the CHARIN is the highest priority job scheduled, control is routed to the CHARIN job. The processing performed by the job is shown by the flow diagram in figure **3-12.** Upon initiation of the processing for CHARIN, an interlock is set up by the computer on itself so that no other programs can use the **DSKY.** In other words, the use of the DSKY is reserved for the PINBALL program and the astronaut or a ground based keyboard operator.

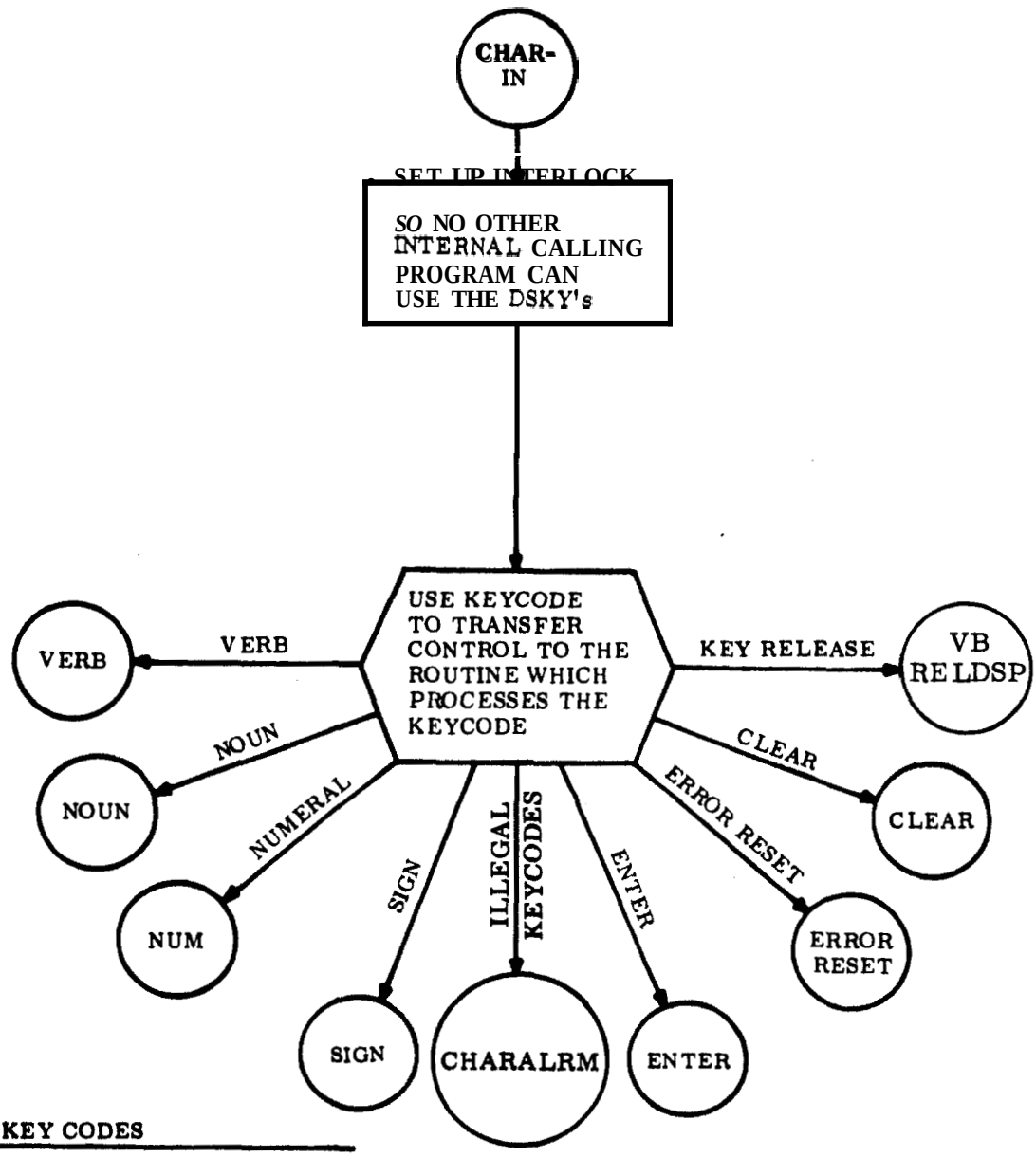
Control is routed to the CHARIN subroutine dependent on the keycode input. For instance, if a numerical keycode has been entered, control is routed to the NUM routine, If the NOUN keycode is entered, control is routed to the NOUN routine, etc. If the keycode input is illegal, control is route? to the CHARALRM subroutine.

3.3.2.2 NOUN Subroutine. Control is routed to the NOUN subroutine of CHARIN whenever the input keycode is that of the NOUN key.

The following is performed by this subroutine:

a. The **NOUN REG,** a memory register used to store the assembled NOUN code, is set to zero in preparation for the receipt of a new NOUN code.

b. The NUM subroutine, which will assemble the new NOUN code as it is entered, is conditioned to accept the next two keycodes as a NOUN code.



LEGAL KEY CODES

10000 = *0*	10001 = VERB
00001 = 1	10010 = ERROR RESET
00010 = 2	11001 = KEY RELEASE
00011 = 3	11010 = +
00100 = 4	11011 = -
00101 = 5	11100 = ENTER
00110 = 6	11110 = CLEAR
00111 = 7	11111 = NOUN
01000 = 8	
01001 = 9	

OTHER 14 CODES ARE ILLEGAL

17514-1

Figure 3-12. CHARIN (Sheet **1 of 11**)

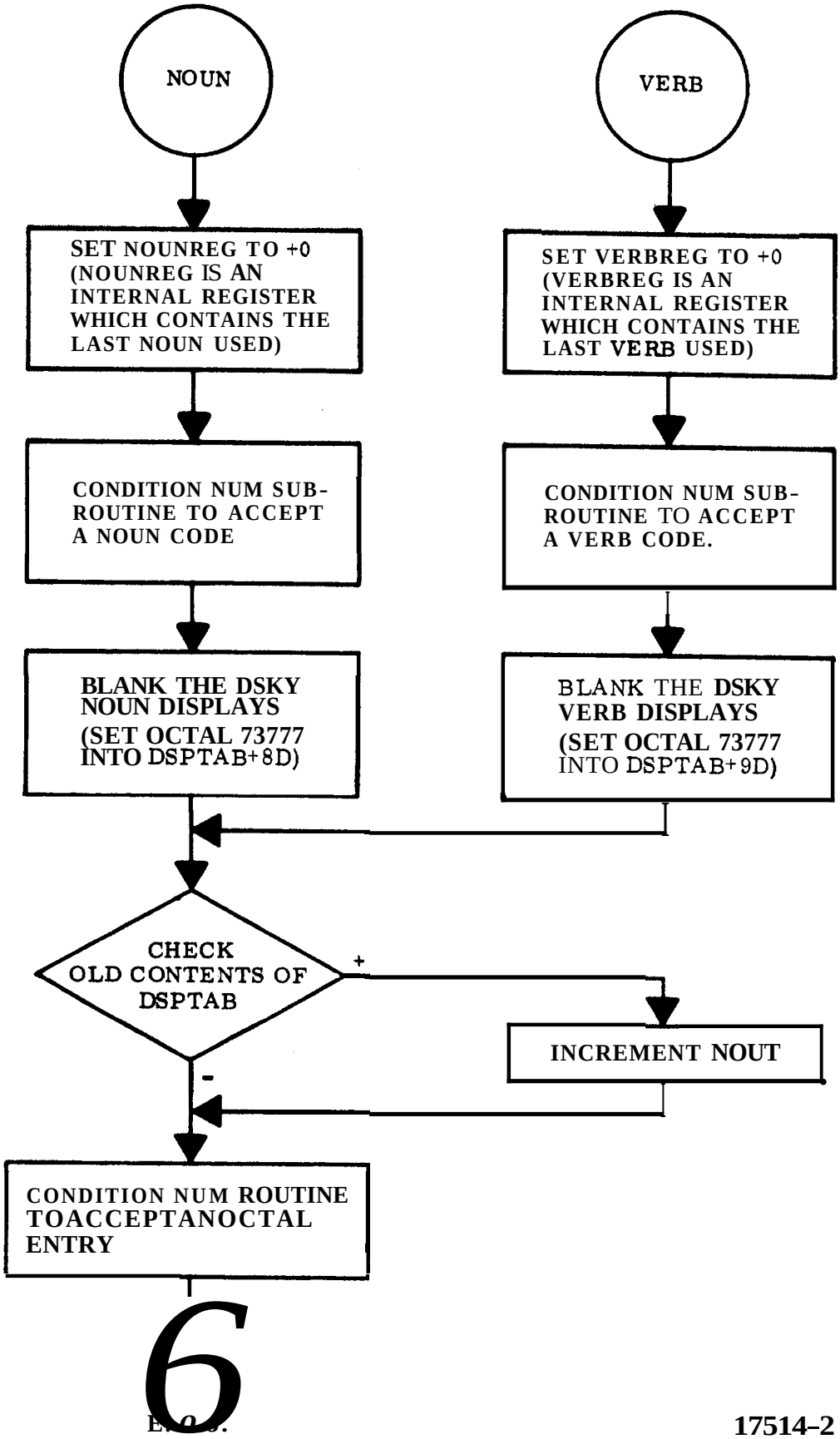


Figure 3-12, CHARIN (Sheet 2 of **11**)

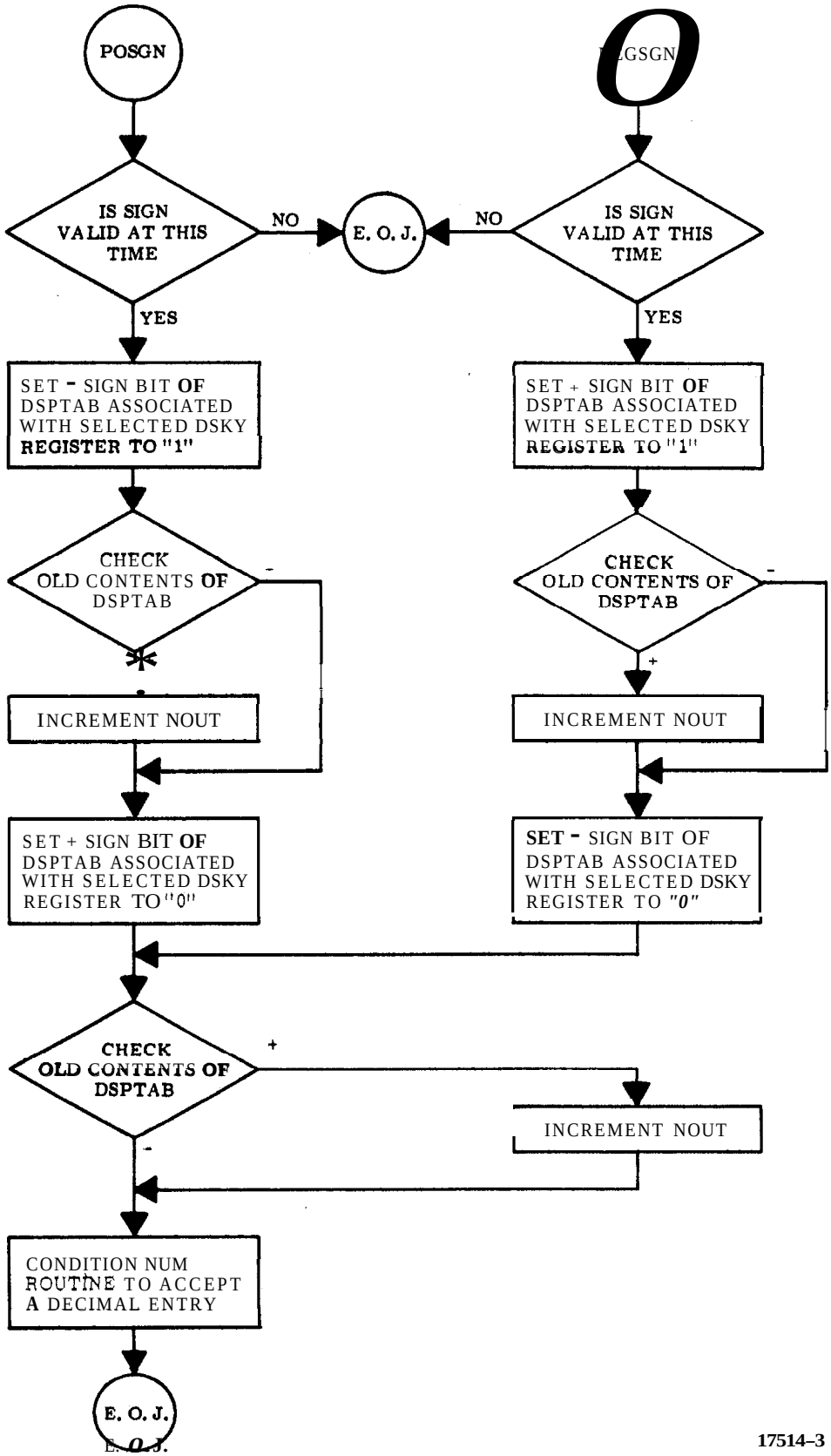


Figure 3-12. CHARIN (Sheet 3 of 11)

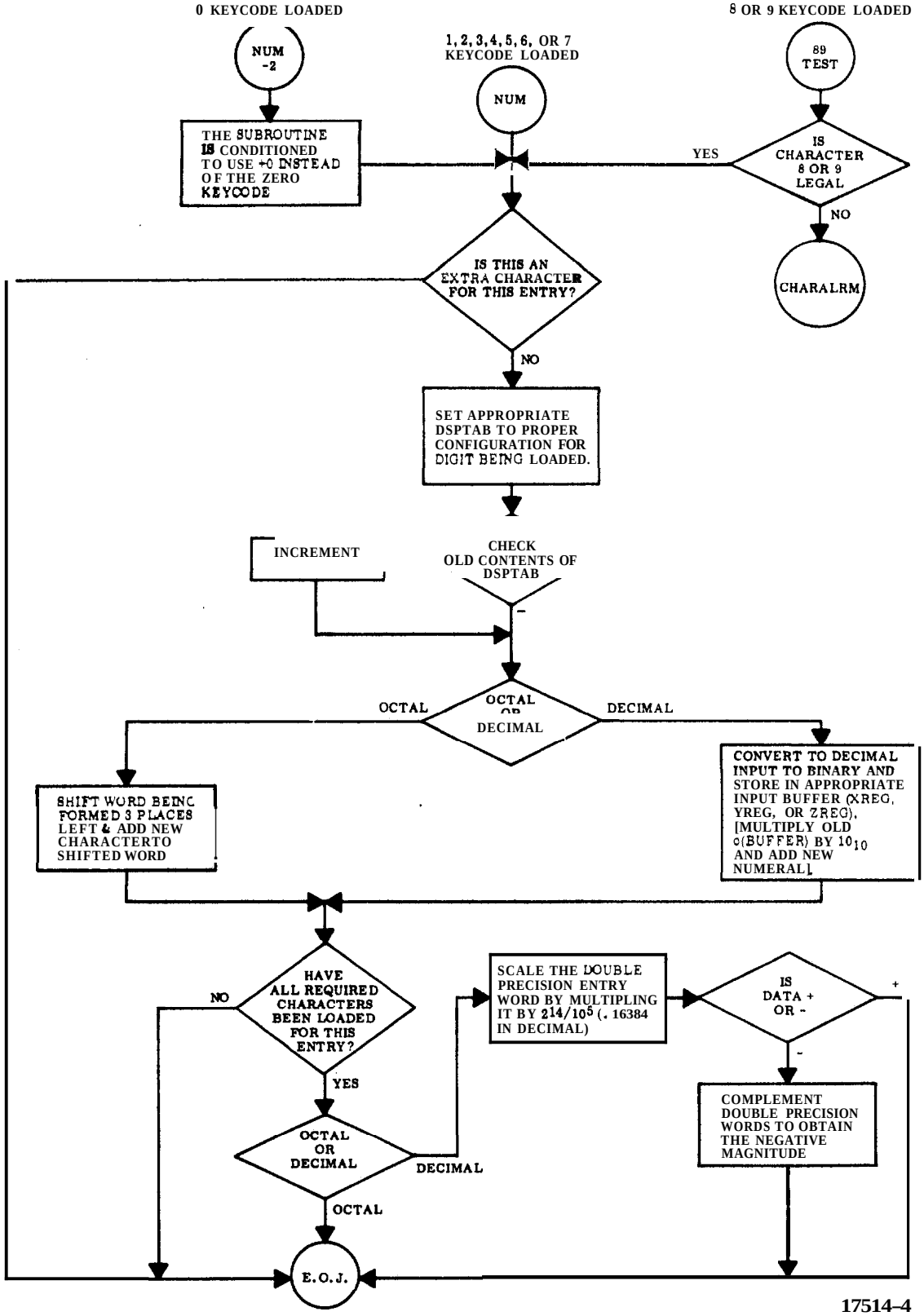


Figure 3-12, CHARIN (Sheet 4 of 11)

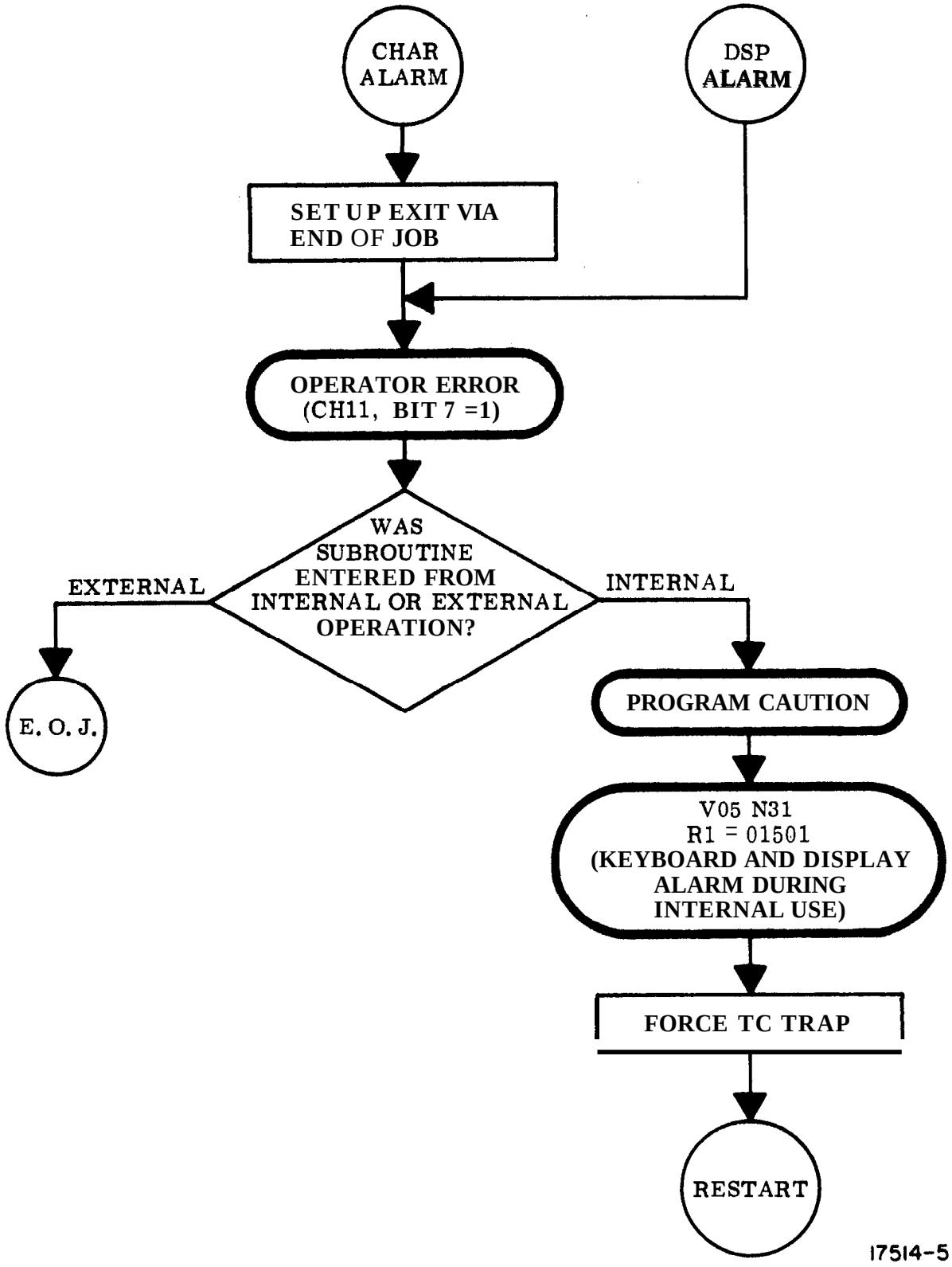


Figure 3-12. CHARIN *(Sheet* 5 of 11)

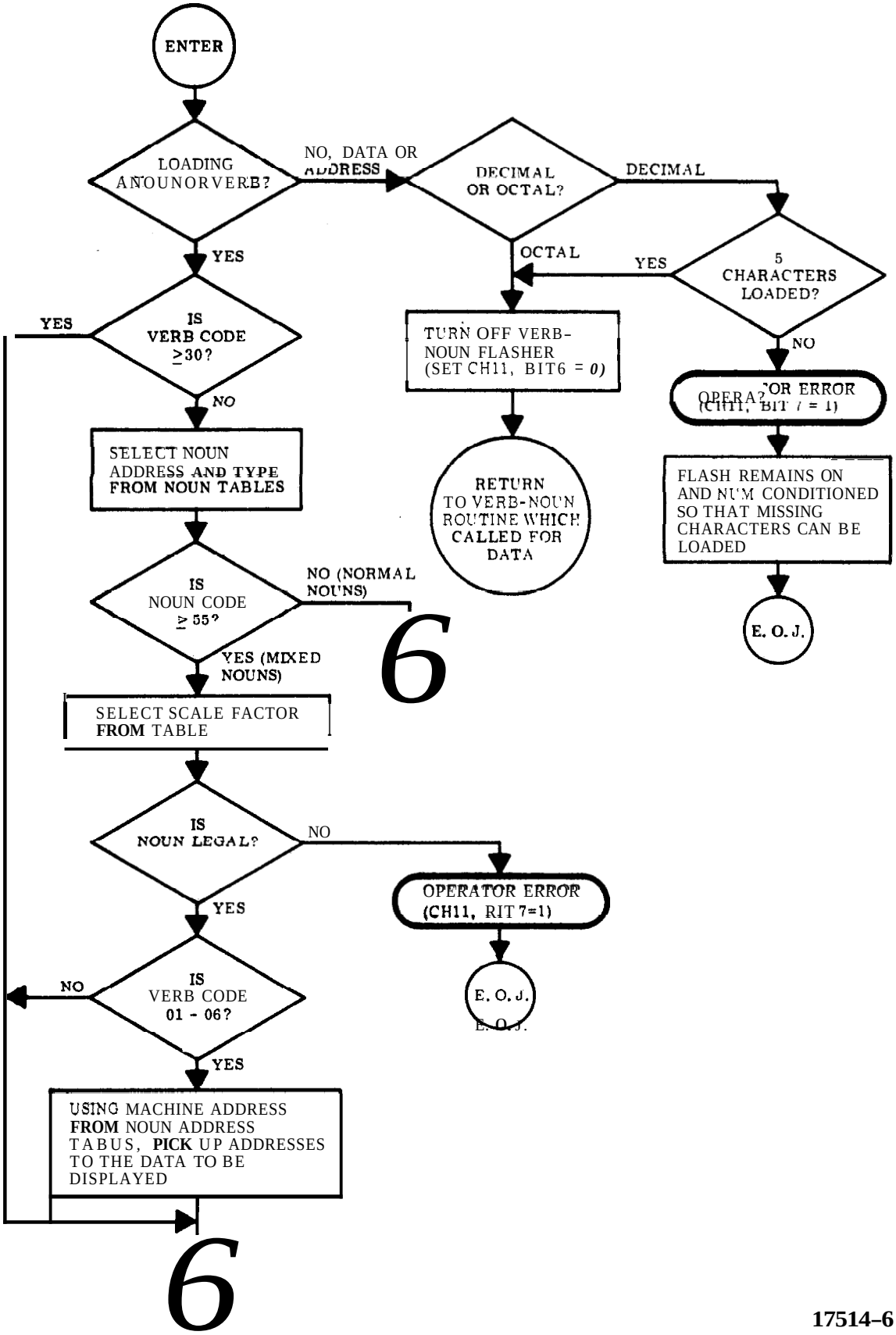
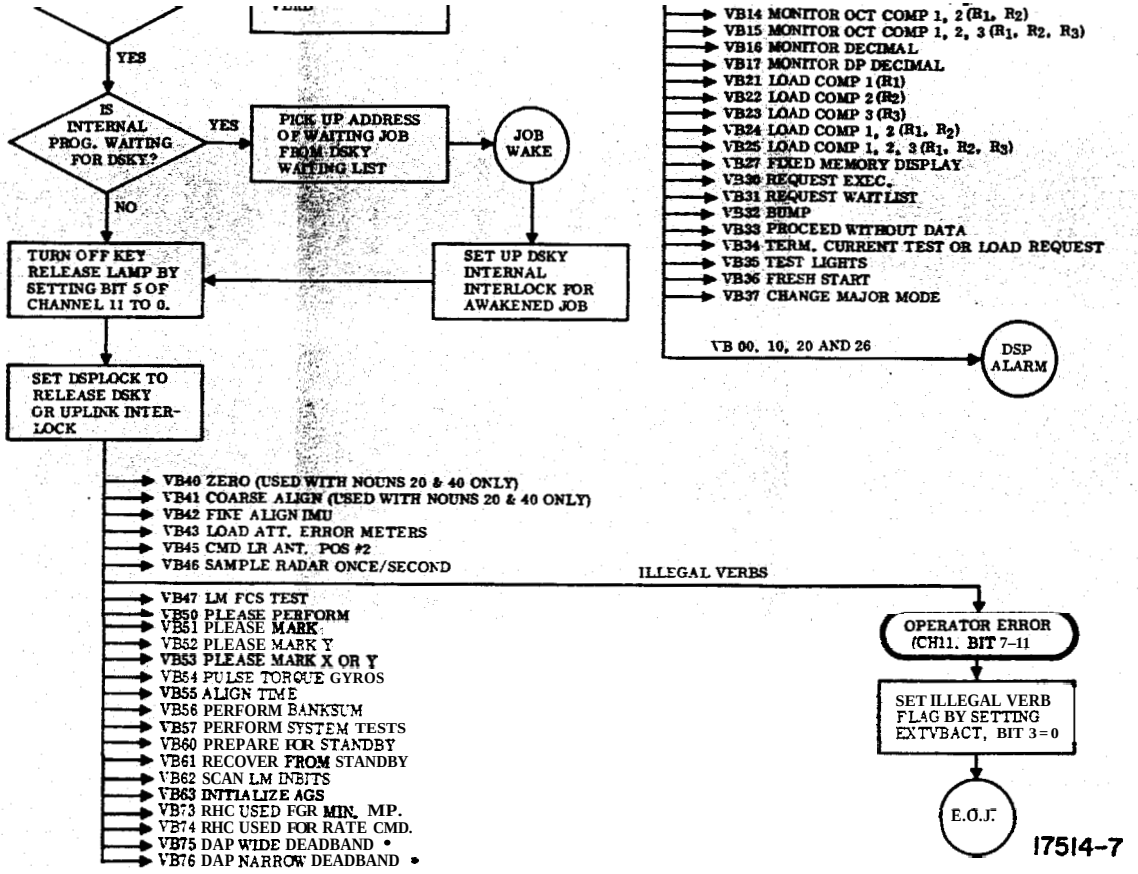


Figure 3-12. CHARIN (Sheet 6 of 11)



* NOT INCLUDED IN SUNBURST REV 14 LISTING

Figure 3-12. CHARIN (Sheet 7 *of* 11)

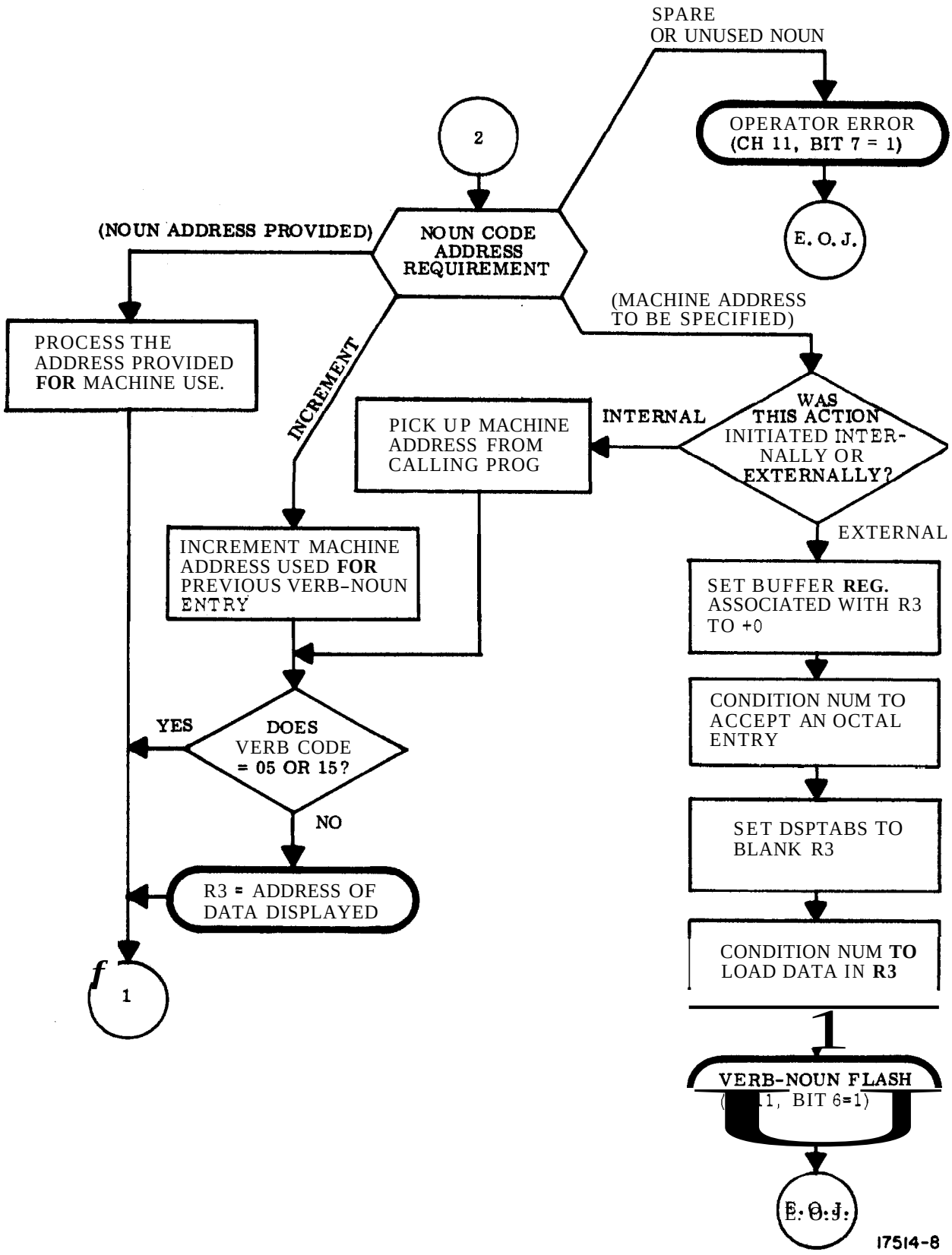


Figure 3-12. CHARIN (Sheet **8 of 11)**

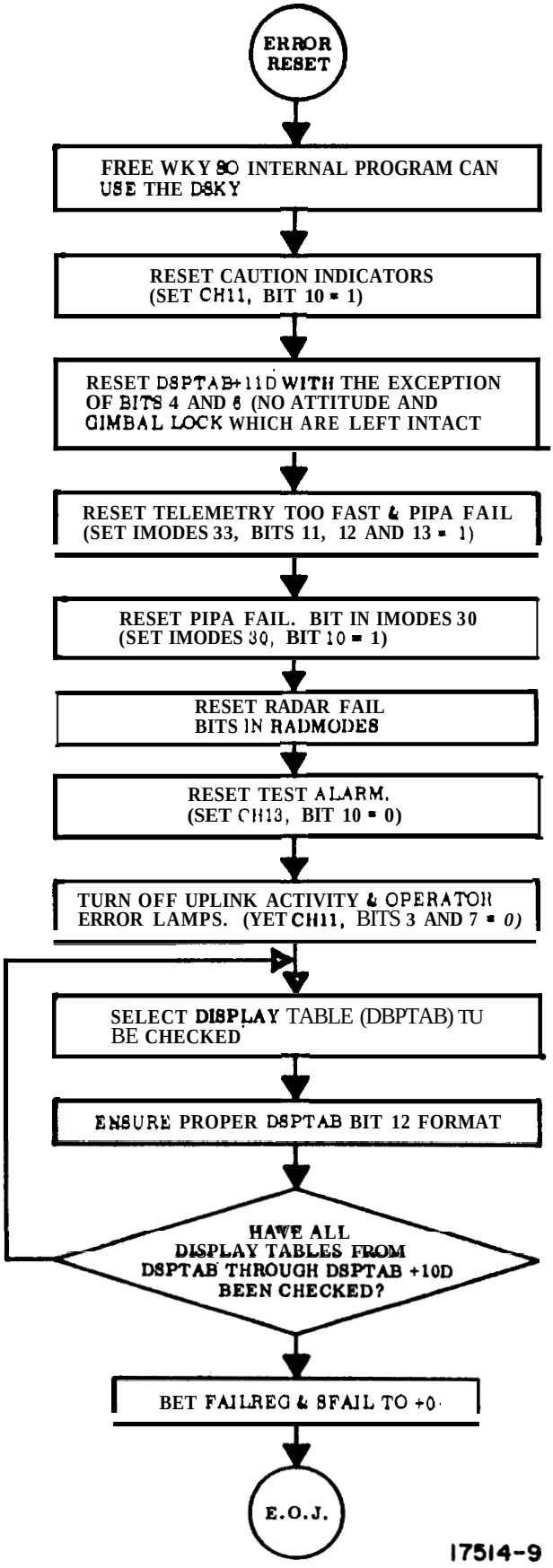


Figure 3-12. CHARIN (Sheet **9 of 11**)

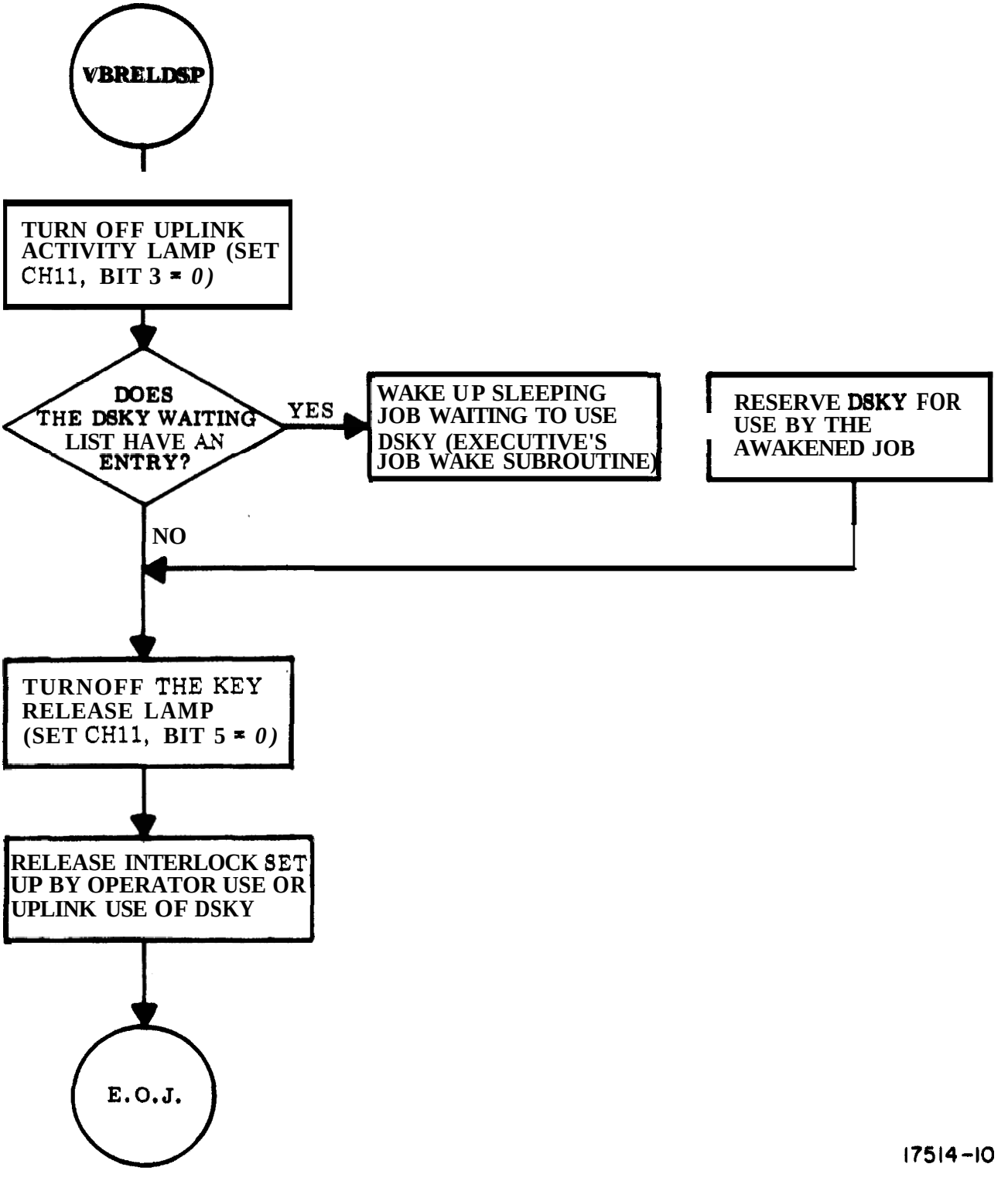


Figure 3-12. CHARIN (Sheet **10 of 11)**

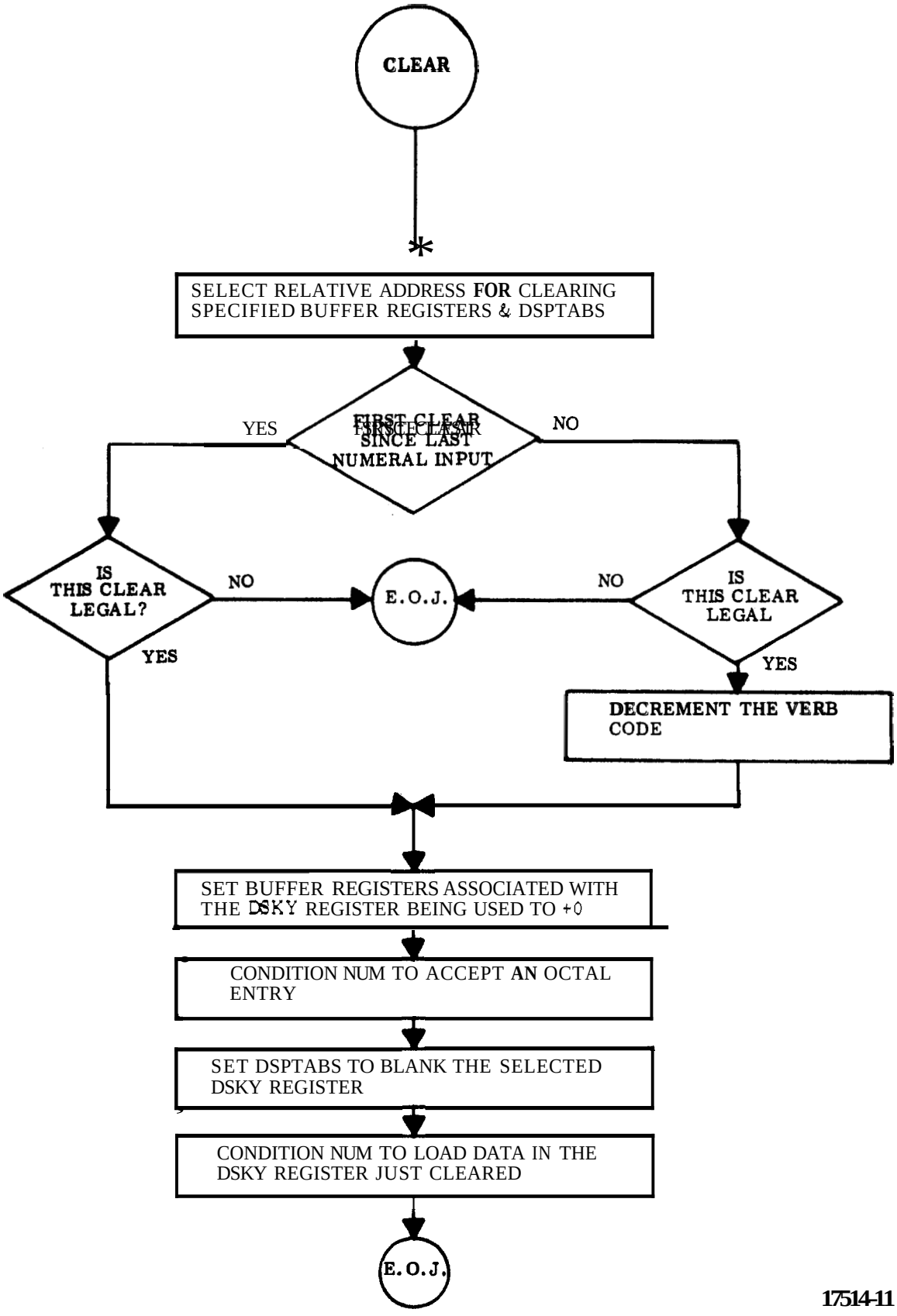


Figure 3-12. CHARIN (Sheet 11 of 11)

c. The display table, DSPTAB+8D, *is* set to octal **73777** to blank the two NOUN display panels.

d. The NUM subroutine ***is*** conditioned to accept octal keycode entries.

After performing these functions, control ***is*** routed to the Executive End Job routine.

3.3.2.3 VERB Subroutine. The VERB Subroutine ***is*** similar to the NOUN Routine, Control **is** routed to this subroutine whenever **a** VERB keycode **is** received.

The following functions are performed by this routine:

a. The VERB REG, a memory register used to store the VERB code, is set to zero in preparation for the receipt of a new VERB code.

b. **The** NUM subroutine ***is*** conditioned to accept the next two numerical inputs as a VERB code.

c. The display table, DSPTAB+9D, **is** set to octal **73777** to blank the VERB display panels.

d. The NUM subroutine is conditioned to accept octal keycode entries.

After performing these functions, control **is** routed to the Executive End Job subroutine.

3.3.2.4 SIGN Subroutine. Control ***is*** routed to the SIGN subroutine of CHARIN **if** the keycode ***is*** for either the + or - key.

The processing performed by the POSGN (positive *sign)* **is** listed below:

a. A check ***is*** made to determine if a SIGN ***is*** valid at this time in the sequence of keycode entries. If it **is** not, control **is** routed to the End **Job** subroutine of the Executive. If the SIGN **is** valid, which it would be prior to entering a piece of decimal data, the processing proceeds.

b. The ***- sign*** bit (bit **11)** of the selected DSPTAB (display table) **is** set to **1.** The DSPTAB **is** using inverted logic ***so*** the above action will reset the - sign bit.

c. The + ***sign*** bit (bit **11) of** the selected DSPTAB is set to ***0.*** Since the DSPTAB uses inverted logic, this action sets the + ***sign*** bit.

NOTE: Different DSPTAB registers contain the + sign bit and the - sign bit for the DSPTAB's used to provide displays in R1, **R2** and **R3** of the **DSKY.** The result **of** the above operation ***is*** that the + **sign** will be displayed **in** the appropriate DSKY register.

d. The NUM routine is conditioned to accept decimal keycode entries.

After performing these functions, control ***is*** routed to the End Job subroutine of **the** Executive.

The processing performed by the **NEGSGN** (negative sign) subroutine is essentially the same as the **POSGN** subroutines. The differences are that the + SIGN bit is set to **1,** the **- SIGN** bit is set to 0 which will result in the negative sign being lighted on the selected DSKY register.

3.3.2.5 NUM Subroutine. The **NUM** Subroutine is used whenever a numerical keycode is received. This routine provides for the assembly of the numerical information as it is entered into the computer. Control over the type of data (either decimal or octal) that is to be entered and the number of characters to accept is established by the **NOUN, VERB** and **SIGN** subroutines. The **NOUN** and **VERB** subroutines condition the **NUM** subroutine to accept two octal characters while the **SIGN** subroutine conditions the **NUM** subroutine to accept a five decimal Characters. If a word is to be entered in octal, the sign keys are not used prior to entering the characters of the word and the NUM subroutine assumes that the data is being entered in octal. In the assembly process, the information being entered is converted from octal or decimal to a natural binary form,

The processing performed by the **NUM** routine is as follows:

If **CHARIN** is provided with a zero (10000) keycode, control is routed to **NUM-2** prior to the keycode being processed by NUM. **NUM-2** converts the zero keycode to +0 for use by **NUM.**

If **CHARIN** is provided with an 8 or 9 keycode, control is routed to 89TEST. The **89TEST** will determine if digit 8 or 9 is valid at this time.

NOTE: If the **NUM** subroutine had been conditioned to accept an octal input and one of these two numerical characters are used, an operator error exists since these digits do not exist in the octal numbering system. Control is then transferred to the **CHARALRM** subroutine. The **OPERATOR ERROR** indicator will be lighted. Then control is routed to the End Job subroutine. If the keycode entered is not an 8 or **9** for octal loads or if decimal data is being loaded, processing proceeds.

A check is made to determine if all of the required characters have been entered for the information being loaded. In the case of entering **VERB** and **NOUN** codes, the required number of characters is two while five are required for either octal or decimal data word entries. If all of the required characters have been processed, control is routed to the End Job subroutine and the character input is not utilized. However, when all of the required characters have not been entered, processing proceeds.

The correct display table **(DSPTAB)** location is selected corresponding to the numerical entry being processed, and this **DSPTAB** is set to the proper configuration to light the digit being loaded on the DSKY.

If an octal load is being made, all previously assembled information for this load is shifted three digit positions toward the most significant digit position. Then the newly received keycode input is added to the shifted word with the keycode in the least significant digits. By shifting the previously assembled portion of a quantity being loaded in octal, the magnitude of the previously assembled information is increased by a factor of eight. As the information is keyed in most significant octal digit first, the information *is* assembled and "converted" to a natural binary form.

If the information is being entered in decimal, a similar procedure is followed, With decimal loads, all previously assembled information for a particular quantity is shifted three digit positions toward the most significant digit position and the same quantity shifted one digit position towards the most significant digit positions. These two quantities are added. This increases the magnitude of the previously assembled information by a factor of 10. After forming the sum of the two shifted quantities, the newly entered keycode is added to this sum.

If a decimal load is being made, all previously assembled information is multiplied by 10 (Decimal) and the newly received keycode is added. The result is the natural binary equivalent of the present decimal entry and is stored in the buffer register corresponding to the DSKY register (R1, R2 or R3) being loaded.

In either case, octal or decimal loading, after the assembly and conversion procedure outlined above is completed, a check is made to determine if the required number of input characters have been provided. If not, control is routed to the End Job subroutine.

If the required number of characters have been entered, a check is made to determine if the load is octal or decimal. If the information loaded was octal, control is routed to the End Job subroutine.

If the load was decimal, the information loaded is scaled by a factor of 0. 16384 and a check is made to determine if the decimal data is negative or positive. If the quantity loaded is negative, the words of the load are complemented putting them in negative form. If the load was positive, the information is used in its uncomplemented or positive form. In either case, control is then routed to the End Job subroutine.

3.3.2.6 CHARALRM Subroutine. The CHARALRM subroutine is used whenever an illegal 5 bit keycode is processed by PINBALL's subroutines. The following is performed:

a. Bit 7 of channel 11 is set to 1 to turn on the operator error lamp on the DSJSY. The operator error lamp flashes.

b, A check is made to determine if the illegal keycode resulted from an external input such as uplink telemetry or the DSKY keyboard. If the keycode originally was an external input, control is transferred to the EXECUTIVE'S End of Job Subroutine.

c. If the keycode originated internal to the computer, control is transferred to the ABORT Routine which will result in the Program Caution lamp being lighted, Verb 05, Noun 31 being displayed and 01501 being in R1.

d. The program abort sets the computer operation into a TC Trap condition which will force a restart.

3.3.2.7 ENTER Subroutine. Control is routed to the ENTER subroutine when the keycode for the ENTER key is received. Internal programs also use the capabilities provided by the routines accessible through the ENTER subroutine.

In the procedure of entering the **VERB** and **NOUN** codes, the **ENTER** key is used. When it is used in this instance, the **ENTER** subroutine performs a routing function to a particular routine specified by the **VERB** code. Also, if it is applicable, the **ENTER** subroutine selects a machine address to be used by the routine specified by the **VERB** code using the **NOUN** code. When the **ENTER** key is used after loading a five character data word, the **ENTER** subroutine returns control to the routine which requested that data be loaded.

The processing performed is as follows:

a. **A** check is made to determine if the **ENTER** is for a **NOUN-VERB.** If it is not, **the ENTER** must have been a result of a data or address load. In this case, the flashing **of** the **VERB-NOUN** display panels is occurring as set by this routine or the routine which requested the load. **A** check is made to determine if the data or address entry was made in octal or decimal. If the entry was in decimal, an additional check is made to insure that five characters were entered. If the entry was octal or five decimal characters, the noun-verb flasher is turned off and control is returned to the caller. If a decimal entry was initiated but five characters were not entered, the **OPERATOR ERROR** lamp on the DSKY is lighted. The noun-verb flasher will remain on and the **NUM** subroutine is conditioned to accept the missing characters. Control is routed to the End of Job subroutine.

b. If the load was that of a **VERB-NOUN** code, a check is made to determine if the **VERB** code is equal to or greater than octal *30.* If it is, control is routed to connecting point **#1.** If the **VERB** code is less than **30,** the processing proceeds by selecting the nouns machine address and type from applicable tables.

c. If the **NOUN** code **is** equal to or greater than **55,** the scale factors associated with that **NOUN** are selected from applicable tables. **A** check is made to determine if the **NOUN** is legal. If the **NOUN** is not legal, the **OPERATOR ERROR** lamp is lighted and control is routed to the End of Job subroutine. Then a check is made to determine if the **VERB** code is **1** through **6.** If it is, the machine address from the Noun Address Table is used to obtain the address of the data to be displayed, Whether the **VERB** code is **1** through *6* or not, control is routed to connecting point **#1.** If the **NOUN** code **is** less than **55,** control is routed to a check which determines the **NOUN** address requirement.

d. The **NOUN** code **is** checked to determine if the **NOUN** machine address is provided, if the previous NOUN's machine address should be incremented, if the **NOUN** is a spare or unused **NOUN** or if the machine address is to be specified.

e. **If a** specified machine address is required, a check is made to determine whether the entry to the **ENTER** subroutine was due to external (keyboard or uplink) action. If the entry was from an internal source, the address is picked **up** from the calling routine. If the entry was from an external source, the buffer registers associated **with R3** of the DSKY are set to **+0** and the **NUM** subroutine **is** conditioned *to* accept **an** octal entry. The display tables **(DSPTAB's)** are set to blank **R3** of the **DSKY.** The **NUM** subroutine is now conditioned to load data into **R3** and the NOUN-**VERB** flasher is turned on.

f. If a spare or unused noun code is used, the **OPERATOR ERROR** lamp is lighted and control is routed to the **End of** Job subroutine.

g. If the noun code address is provided, the address is processed for use in the operation specified by the VERB code.

h. If the noun code requirement calls for the previous address to be incremented, the incrementation is performed on the machine address used for the previous NOUN-VERB entry.

i. If the machine address is picked up from the calling routine or if the machine address of the previous NOUN-VERB entry was incremented, the VERB code is checked to determine if it is **05** or **15.** If these are used, control is routed to connecting point **#1.** If not, the display tables (DSPTAB's) are set so that the T4RUPT routine will display the machine address of the data displayed in R1 in R3. Control is then routed to connecting point **#1.**

j. From connecting point #1, control is routed to a check on the VERB code. If the VERB code **is** less than **40,** the VERB code is used to route control to the appropriate VERB routine. These routines essentially provide for the display, monitor and load of data.

k. If the VERB code *is* greater than or equal to **40,** a check is made to determine if a job *is* waiting to use the DSKY. If it *is,* the waiting job **is** activated using the Job Wake subroutine. Then, an internal/DSKY interlock is set so that no other internal program or job can use the DSKY other than the job just awakened. Then, the KEY RELEASE indicator is turned off if it was illuminated. The operator/DSKY interlock **is** removed to **allow** the internal programs to use the DSKY. The VERB code **is** used to route control to the appropriate VERB route.

3.3.2.8 Error Reset Subroutine. Whenever the ERROR RESET key is depressed on the DSKY or the keycode *is* received via the uplink telemetry system, CHARIN routes control to the ERROR RESET routine, to turn off error lamps and forces bit **12 of** all DSPTAB's to proper configuration,

After the display tables are checked, FAILREG (FAILURE NUMBER REGISTER) and SFAIL **(SELF** CHECK FAILURE REGISTER) are set to zero.

3.3.2.9 Key Release Subroutine, The KEY RELEASE subroutine (VBRELDSP) is processed whenever the key **is** depressed on the DSKY or the keycode *is* received via the uplink telemetry system. This key is used after completing an entry to the computer and releases the operator/DSKY interlock. This enables an internal program to utilize the DSKY.

3.3.2.10 Clear Subroutine. CHARIN transfers control to the CLEAR subroutine whenever it receives a keycode for CLEAR from the **DSKY** or uplink telemetry system. This subroutine **"clears"** the register R1, R2, or R3 of the DSKY which *is* presently being loaded. Successive depressions of the CLEAR key cause the previously loaded registers to be **"cleared".** This enables the correction of erroneously entered data but is effective only if used prior to using the last ENTER in a sequence **of** entering data. Beside clearing the DSKY registers, the memory registers used to assemble and store the information within the computer are also cleared. Neither verbs nor nouns can be cleared by the clear subroutine.

3.4 ISS MODE SWITCHING ROUTINES

The mode switching routines presented in this portion of the study guide provide the computer with the capability of commanding modes for the ISS. These routines can be used whenever the appropriate subsystem is under computer control,

These routines are used by internal programs which use the ISS to perform the mode control function. Also, these modes can be requested to be performed through DSKY keyboard or uplink telemetry entries. When the requests are made through the DSKY or uplink telemetry, control is routed to the routines by the ENTER routine of the Pinball program.

3.4.1 ISS CDU ZERO (Figure 3-13). As a result of the operator entering a V40 N20 E in the DSKY or an internal request for the ISS CDU ZERO mode, control will be routed to the IMU ZERO routine. This routine provides the computer with the capability of synchronizing the computer CDU counters with the actual CDU positions. This subroutine can be reduced to two steps; (1) to send a discrete "ZERO IMU CDU's" to the ISS. This discrete causes the electronic CDU read counter to be set to zero and further inhibit any input(s) to this read counter, and (2) to set the computers CDU counters to zero after a 320 ms time delay.

Upon entry into IMU ZERO, a check will be made of coarse align and gimbal lock, since zeroing of the IMU is not allowed with coarse align and gimbal lock condition. Assuming these conditions do not exist, STATE is set to indicate use of the IMU and a check is made to determine if the CAGE pushbutton is pressed. If CAGING is in progress, the interrupted program is resumed. If not, task IMU ZERO 2 is scheduled to be executed in 320 msec if the IMU OPERATE discrete is still present. If not, a program caution condition exists and a display of a 00210 error code is forced into R_1 identifying that the IMU is not operating and in this case the ISS CDU ZERO cannot be accomplished,

Assuming that the IMU OPERATE discrete is present, the task is scheduled and 320 msec later the TIME 3 counter overflows causing a program interrupt and thus via the T3RUPT program, the IMU ZERO 2 task is entered. Again a check on the caging discrete is made. If caging is not in progress, the computers' CDU input counters are set to zero, thus synchronizing the CDU Counters and the computers' CDU counters, The ISS CDU ZERO bit position in channel 12 is set to zero to terminate this mode and a four second delay is scheduled to allow synchronization of the CDU counters with the gimbal angles.

After the four second delay, another check is made to see if caging is in progress. If not, the failure inhibit bits are removed and the appropriate action is taken with respect to the status of the ISS WARNING light. If the calling program was put to sleep prior to zeroing the CDU's it is awakened via the Job Wake subroutine and this task is terminated,

3.4.2 IMU COARSE ALIGN (Figure 3-14). The IMU coarse align mode is entered as a result of depressing a N20V41E or by an internal entry. It is assumed that the operator or the calling program has provided the desired gimbal angle to this routine *so* that the calculations and eventual positioning of the gimbals can be accomplished.

This routine calculates, scales, and delivers the necessary pulses to cause the positioning of the CDU X Y Z via channel 14 and the OUTPUT COMMAND registers.

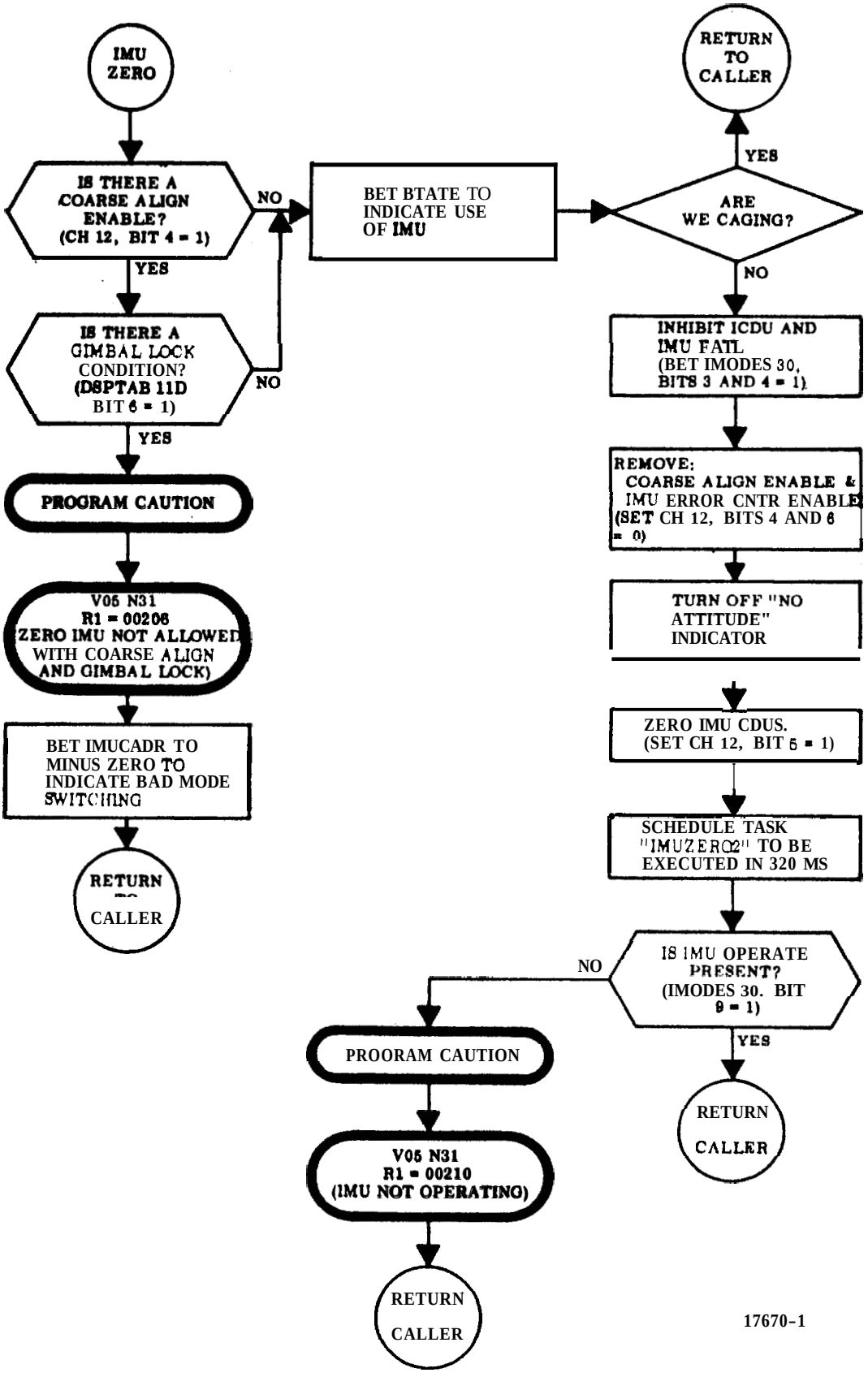


Figure 3-13. ISS CDU-ZERO (Sheet 1 of 2)

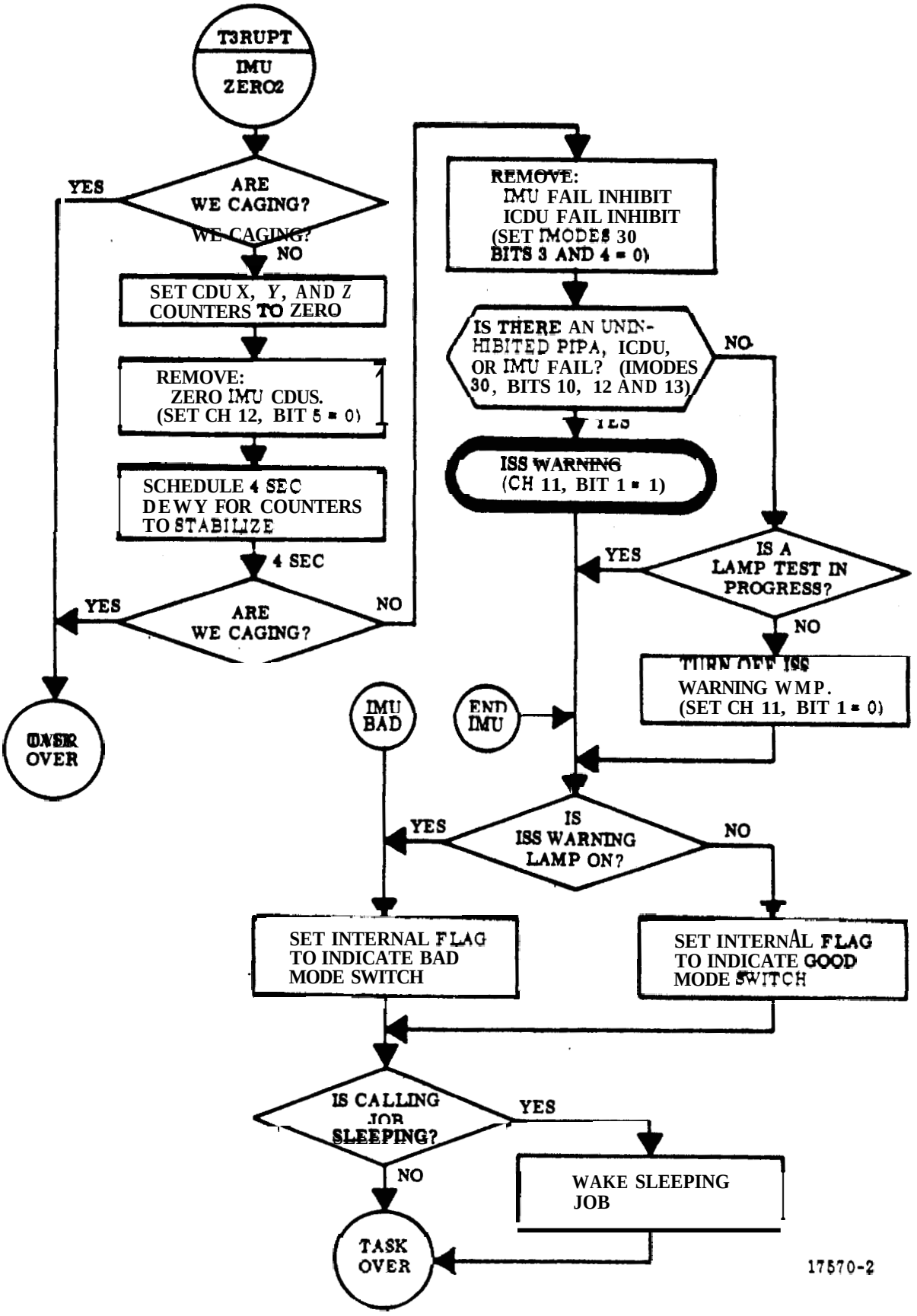


Figure 3-13. ISS CDU-ZERO *(Sheet 2 of 2)*

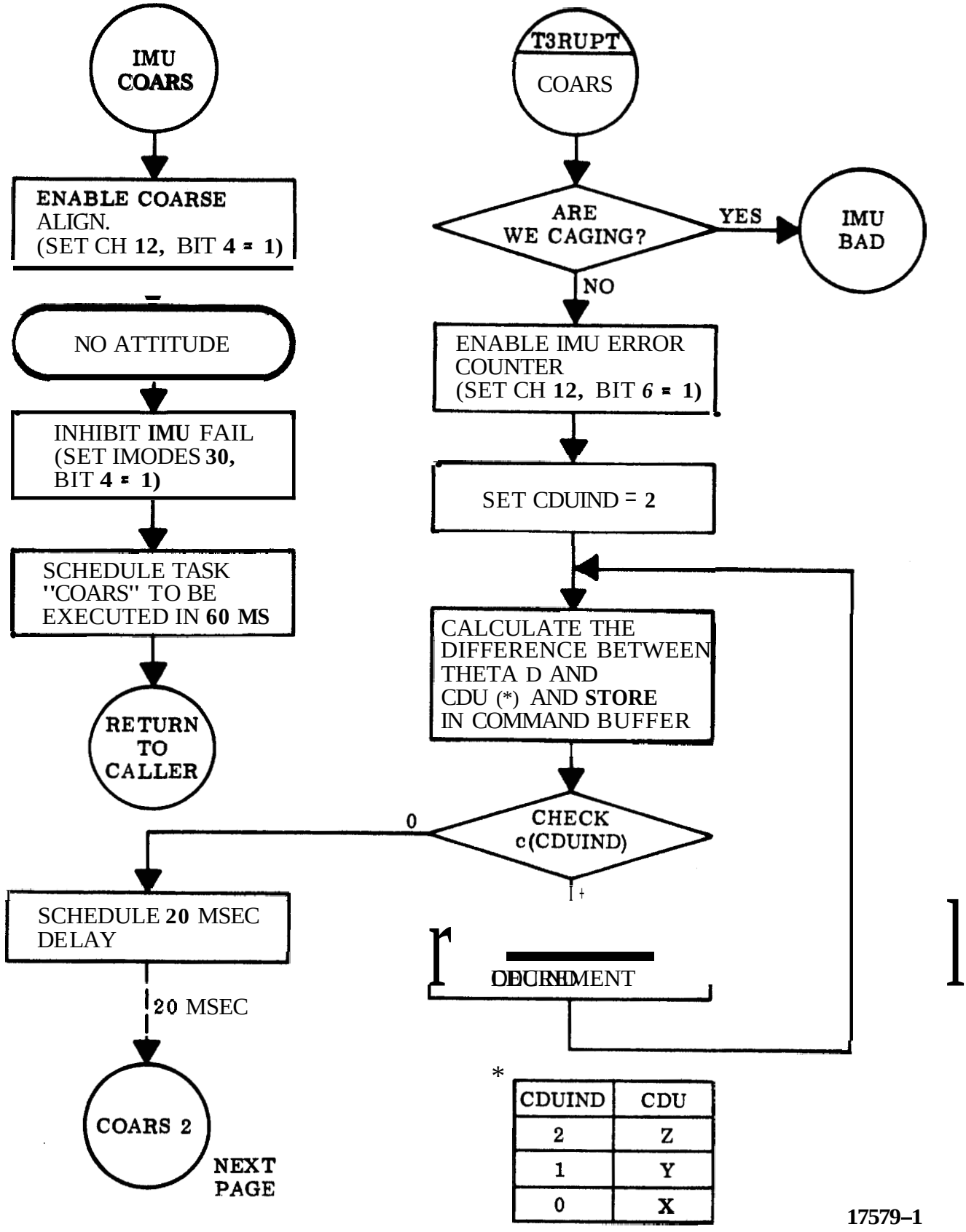


Figure 3–14. IMU COARSE ALIGN **(Sheet 1 of 3)**

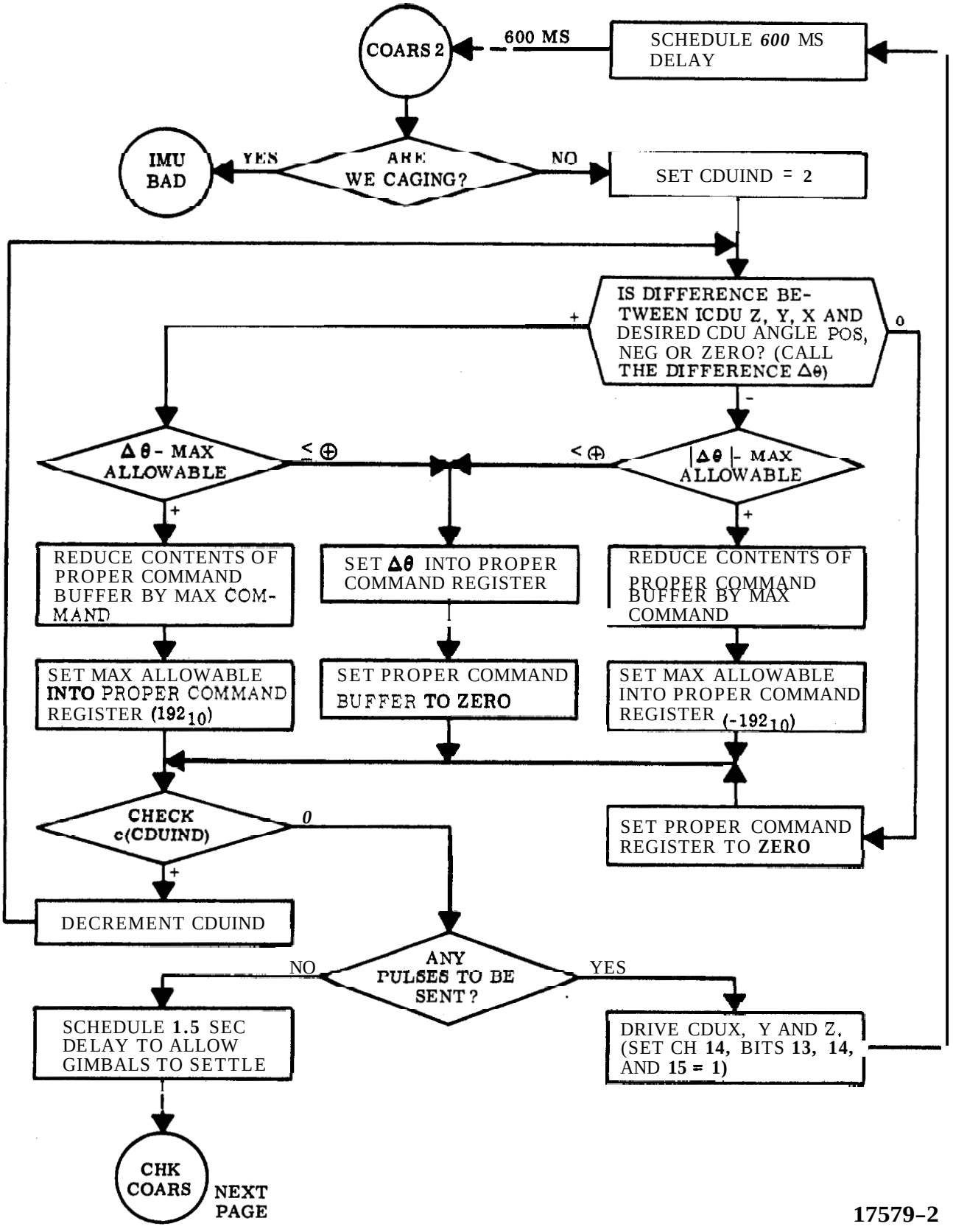


Figure 3-14. M U COARSE ALIGN (Sheet 2 of 3)

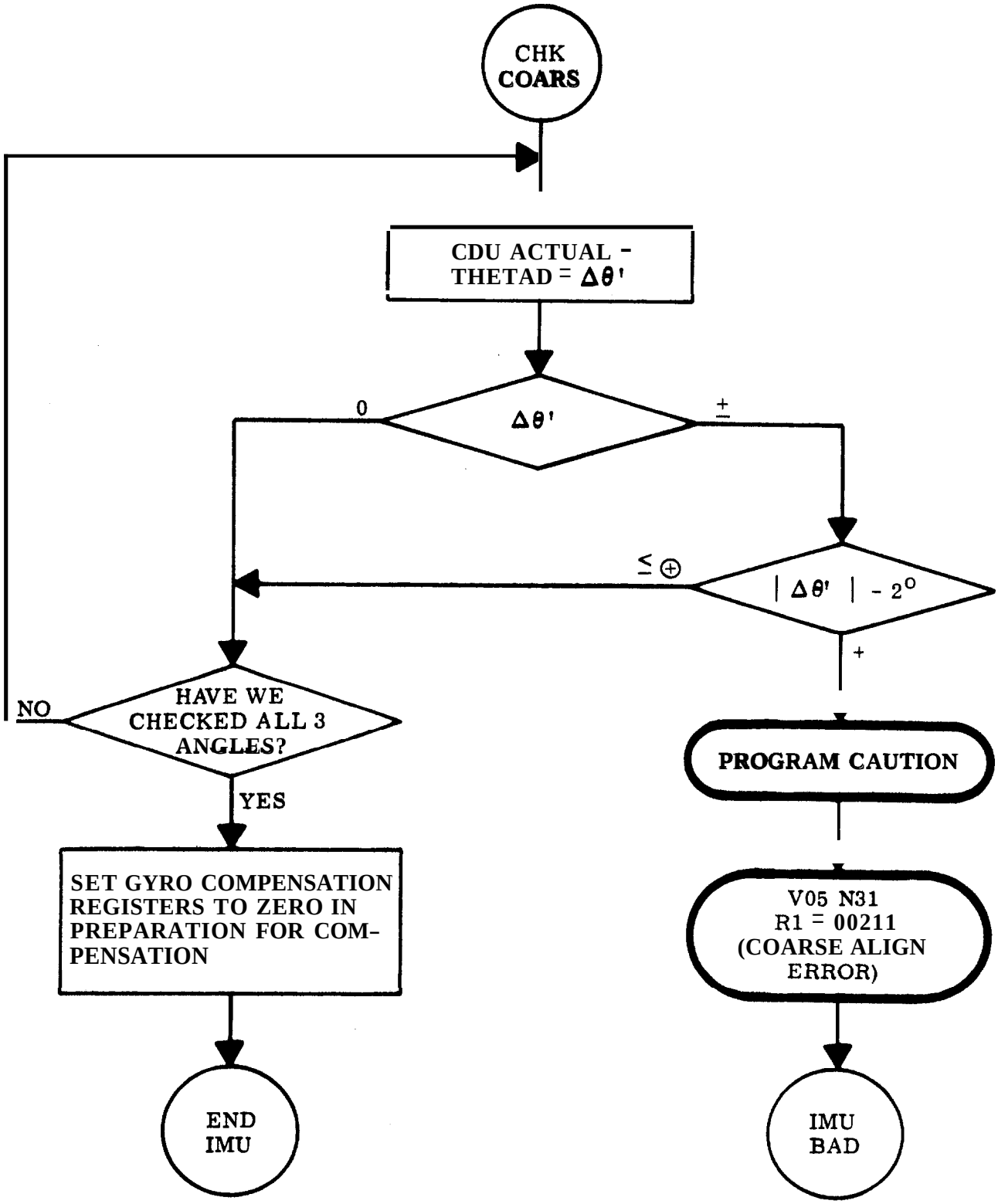


Figure 3-14. IMU COARSE ALIGN (Sheet 3 of 3)

After entering the IMU coarse align routine, the coarse align enable discrete is sent to the ISS. This discrete results in the mode switch to coarse align. The IMU Failure INHIBIT is placed in I MODES 30 because during coarse align an IMU failure could occur during the expected high drive rates. A task "COARS" is scheduled via the waitlist to be executed in 60 msec. This allows time for the mode switching to take place.

In 60 msec the TIME 3 counter overflows and the "COARS" task is started.

A check is made to see if the CAGE switch is energized. If so, the task is terminated, Assuming that caging is not in progress, the IMU error counter is enabled and a cyclic counter called the CDUIND (CDU indicator) is initialized to 2. This counter identifies the specific CDU which is being operated on. If the c(CDUIND) is +2 calculations are on the CDUZ if the c(CDUIND) = +1 the CDUY is operated on and if +0, the CDUX. Notice each time a calculation on a specific CDU angle is performed, the CDUIND is decremented to set the routing control to select the next CDU, After calculating the last $\Delta\theta$ (difference Angle) a 20 msec wait is entered. This insures that the last calculation is complete.

Again a check on CAGING is accomplished. Assume caging is not in progress, the CDUIND is initialized for a limiting check on all three CDU's, The magnitude of the maximum allowable output pulses is ± 192 per *600* msec. This limiting insures that the ECDU will not saturate. If the number of pulses required is less than the maximum allowable, then the actual number of pulses is put into the COMMAND register, If the number of required pulses is greater than the maximum allowable, the maximum allowable is placed into the COMMAND register and any remaining pulses will be evaluated and sent in 600 msec. After all three CDU COMMAND registers have been loaded, a check is made to see if pulses are to be sent.

If there are pulses to be sent, bits 13, 14 and 15 of channel 14 are set = 1 causing driving of all three ECDU channels at once. 600 msec is allowed to send out the pulses prior to going through the loop again.

If there are no pulses to be sent (as will be the case when coarse align is completed) a 1.5 sec settle time is provided.

Since the pulsing requirements for coarse align are completed, a check is now made on the status of the CAGE switch. Assuming no caging, a check is made to see if the coarse alignment of the IMU was accomplished to within 2° about all three axes. If not, a program caution condition exists and the error code 00211 is displayed (coarse align error) and the program caution light is illuminated.

If, however, all axes check to within 20, the gyro compensation registers are initialized in preparation for the gyro compensation routine.

3.4.3 IMU FINE ALIGN (Figure 3-15). The Fine Align mode is entered by either an internal program request or by entering a V42E via the DSKY. (It should be recalled that in the Fine Align mode of operation, the CDU's repeat the gimbal position.)

Assuming that CAGING *is* not in progress, bits 4 and *6* of channel 12 are set to zero and two tasks are scheduled. The first "IFAILOK" is scheduled to be executed in 5 seconds and the second IMUFINED in 90 seconds.

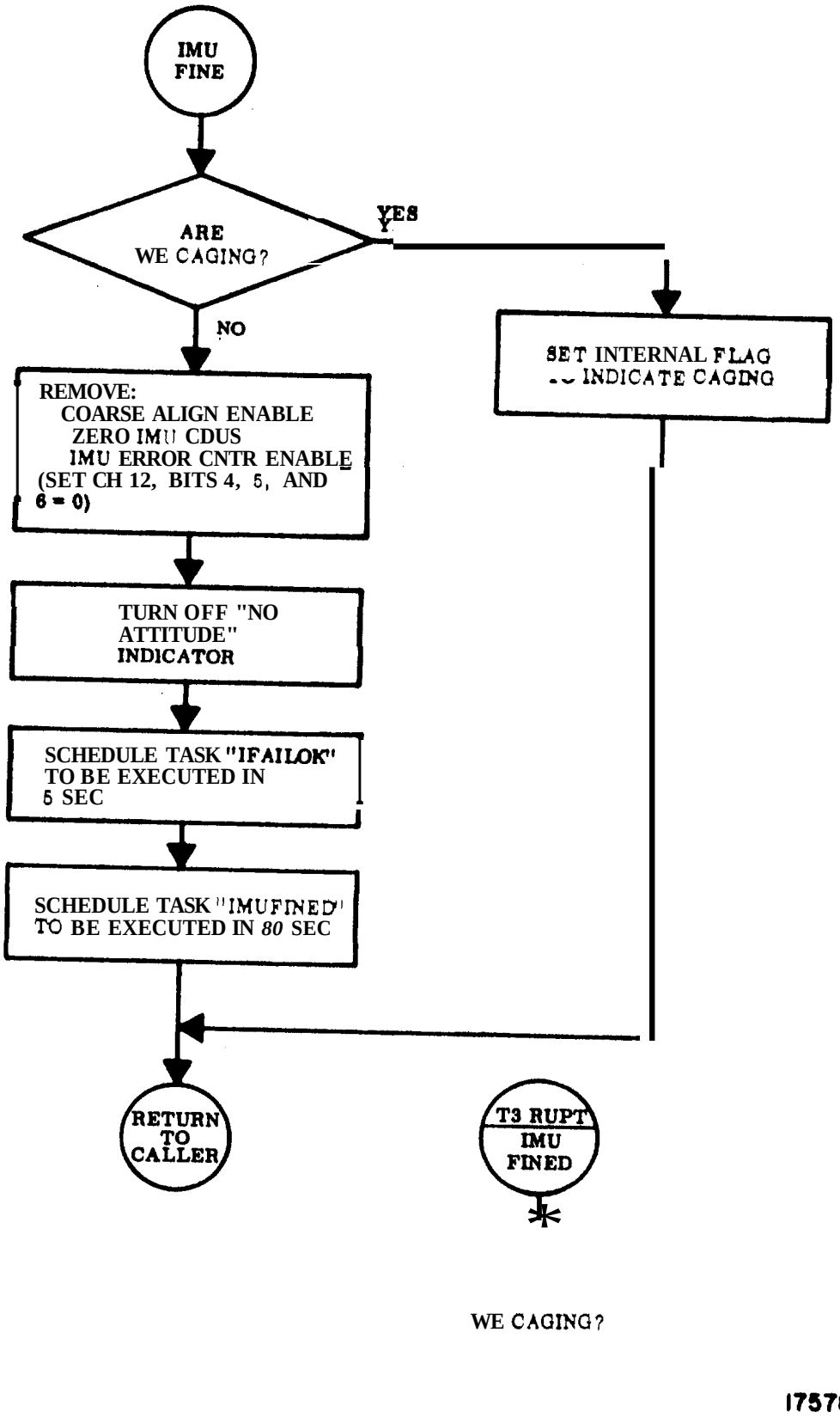


Figure 3-15. IMU FINE ALIGN **(Sheet 1 of 2)**

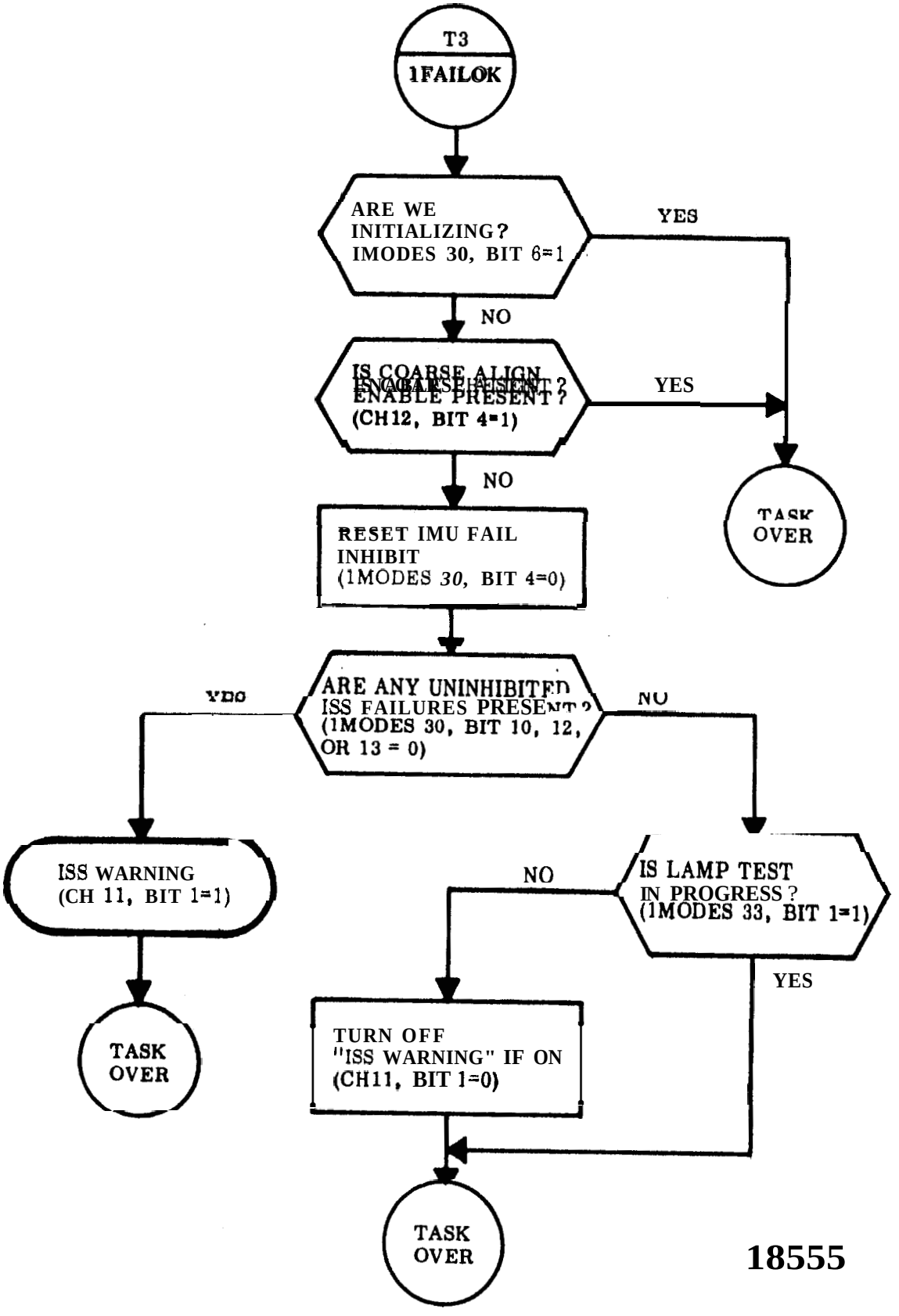


Figure 3-15. IMU FINE ALIGN (Sheet 2 of 2)

Task "IFAILOK" checks to see if CAGING is in progress and to see if a return to coarse align has been requested. If, in both cases, the answer is no, the IMU FAIL INHIBIT bit in I MODES *30* is set = *0* and a check is made for any uninhibited failures.

In 90 seconds, the task "IMUFINED" is executed. The function of this task is to see if the CAGE pushbutton has been activated and to identify a transfer to either GOOD END or BAD END, based on caging information. It is well to note that the overall purpose of waiting 90 seconds prior to ending the FINE ALIGN MODE switching routine is that usually the Fine Align mode switching routine is entered immediately after performing the Coarse Align mode switching routine. During the coarse alignment of the stable member, the floats of the IRIG's are driven to their stops causing the float to be off center. Therefore, during the fine alignment, the 90 second wait is scheduled to allow the IRIG floats to recenter before any drive pulses are generated if required.

3.5 IMU PULSING ROUTINE

Initiation of this routine (Figure 3-16) is accomplished by either an internal entry (from a calling program) or by a keyboard request of V65E.

After initiation of this routine, VERB 25 NOUN 67 (load delta gyro angles), will be requested by the program. This will cause VERB 21 NOUN 67 to be flashed on the DSKY requesting the operator to load the first double precision delta gyro angle. The format of this data is assumed to be ± XX.XXX degrees. Likewise, VERB 22 NOUN 67 and VERB 23 NOUN 67 will be flashed. The double precision data will be loaded into the GYROD registers:

X gyro: GYROD, GYROD+1

Y gyro: GYROD+2, GYROD+3

Z gyro: GYROD+4, GYROD+5

If no data is to be loaded and the present contents of *the* GYROD regioters are to be used as data, a VERB 33 is entered which will allow the program to proceed, If the program is to be terminated, a VERB 34 is entered.

The IMU Pulsing routine examinee the contents of *the* GYROD registers and sets up the high and low order parts of the gyro command. (The gyros are torqued in the sequence Y, Z and X.)

When the mode switching routine is completed, control is returned to the calling program, which is either TORQGYRS or the internal calling program.

After all required data ia loaded, the job IMU PULSE is scheduled via NOVAC. The IMU PULSE routine stores the contents of the 3 DP registers containing the gyro torque commands and checks to see if the IMU is being caged in which case the pulse torque gyro routine would be terminated. Assuming that the IMU is not being caged, a check is made to see if another program is using the gyros. If another program is using the gyros, the IMU pulse routine is put to sleep via the Job Sleep routine to be wakened by the job which is presently using the *gyros.* Once the IMU pulse routine is awakened, it will schedule the task "Strt Gyro" via the Waitlist routine, *to* be executed in 10 msec. (Note that if another program was not using the gyros this task is scheduled to be executed in 40 msec rather than 10 msec.) The high and low order words of the three DP gyro torquing words are forced into sign agreement and control is returned to the caller.

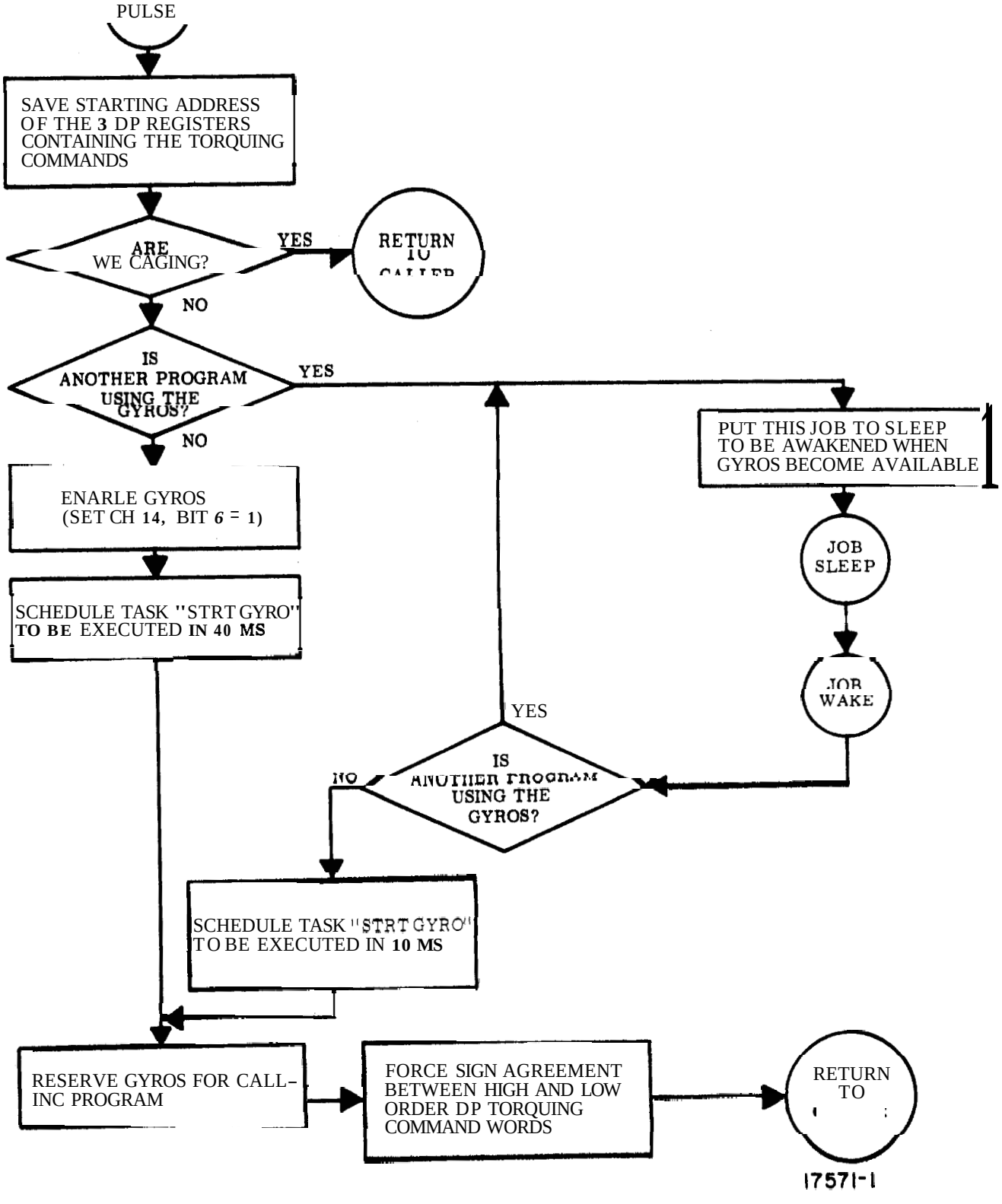


Figure 3-16. IMU PULSING (Sheet 1 of 4)

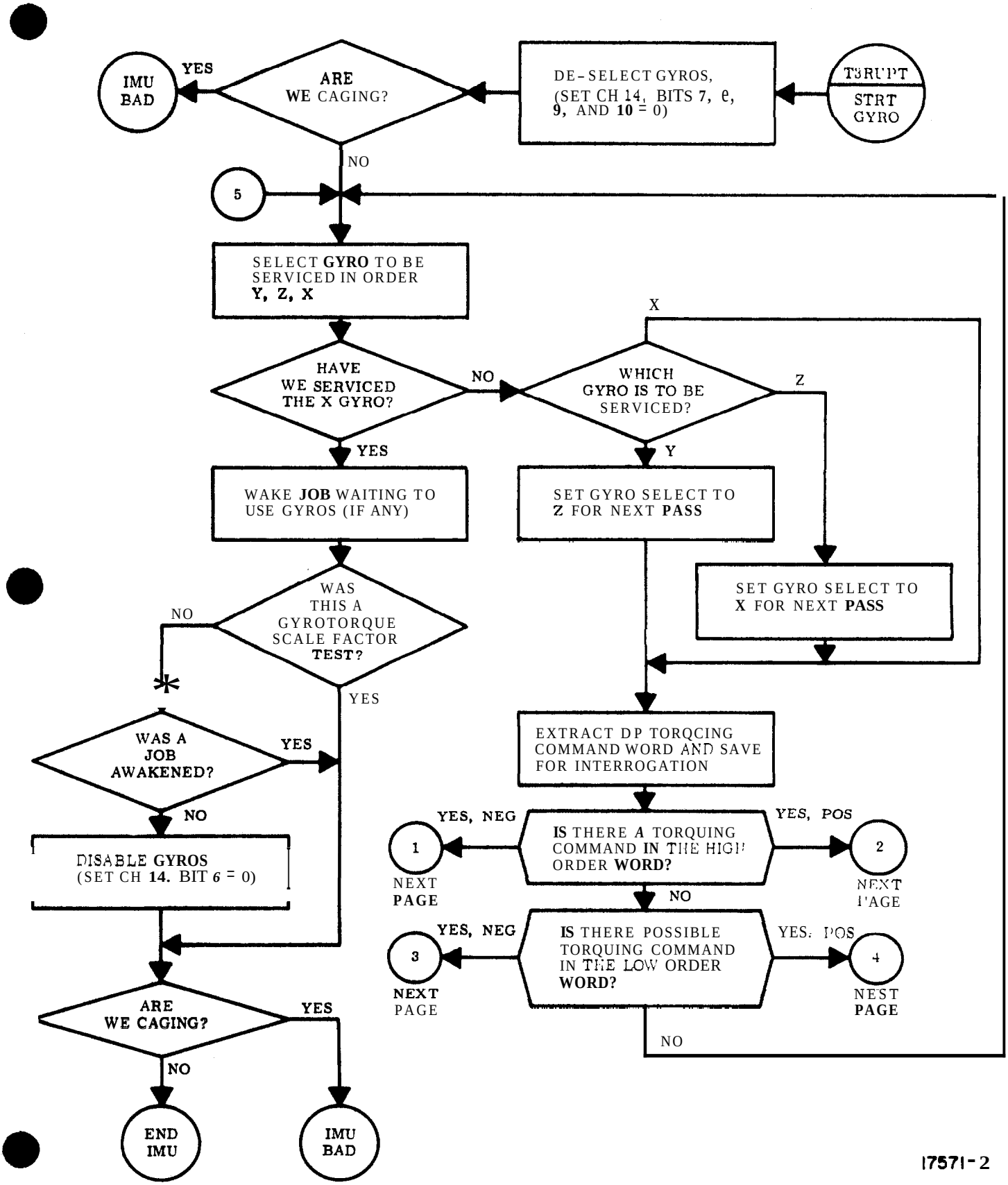


Figure 3-16. IMU PULSING *(Sheet* 2 of 4)

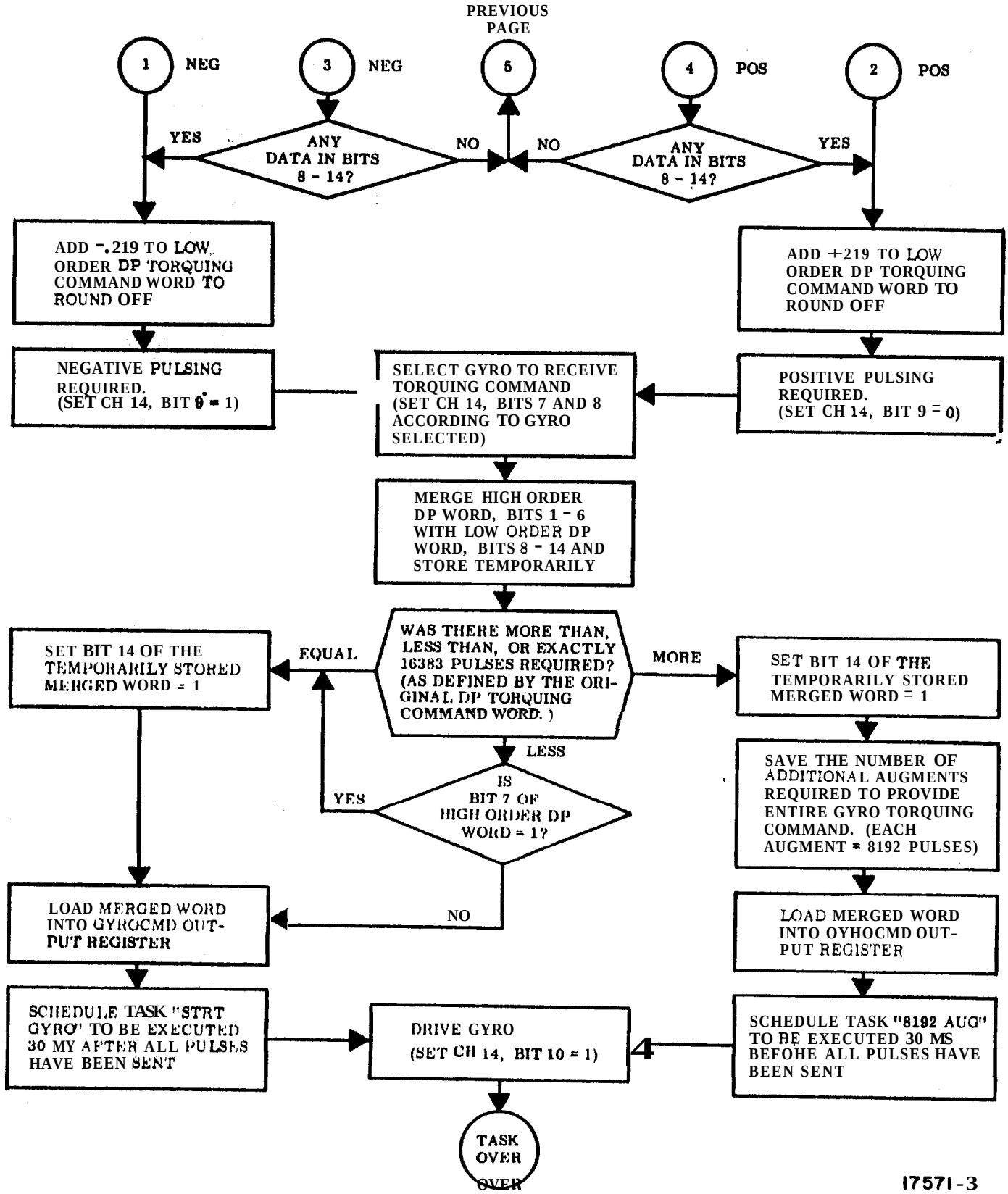


Figure 3-16. IMU PULSING (Sheet 3 of 4)

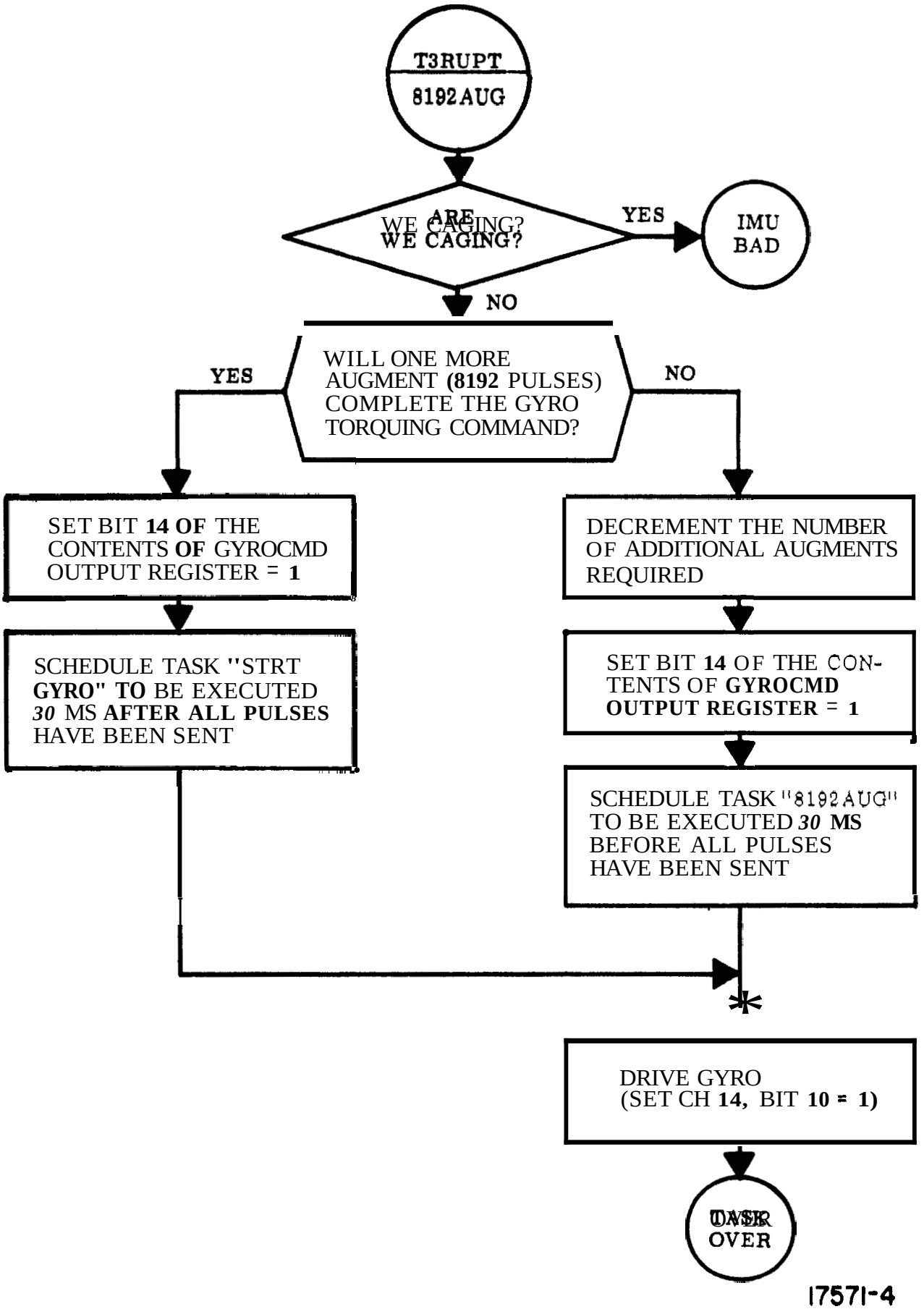


Figure 3-16. IMU PULSING (Sheet **4 of 4)**

When the time 3 counter overflows, the task "STRT GYRO" begins. Immediately upon entering the task bits 7 through 10 of channel 14 are set to zero (gyro select a, b & c and Drive Gyro). *This* action deselects all gyros and disables the driving of any gyro, .Again a check to see if the IMU cage switch is "on". If so end this task and return to caller. Assuming that the IMU is not caged, the specific gyro to be pulsed is selected. (Gyros are pulsed individually in the order of Y Z X. Each gyro will be driven to the desired position before the next gyro is selected.)

The program identifies which gyro is to be selected and provides routing in accordance with this information. Regardless of which gyro is selected, the double precision word associated with that gyro is extracted and stored for interrogation. The interrogation consists of looking at the high order word first to identify whether *or* not the contents are zero. If the high order word is zero, the low order word is checked for a possible command. A logic one in the low order seven bits of the low order word does not necessarily mean that a pulse will be generated since the binary point is located between bits 7 and 8. In order to round out any low order bits a + or -.219 is added to the low order word to force any "borderline case" to result in a full pulse. If the addition of + or -.219 to the low order 7 bits of the low order word does not yield one pulse (a 1 in bit 8), the low order 7 bits are ignored. After the rounding is completed, negative or positive pulsing is selected. In addition to directional information, routing information as to which gyro is to be serviced must be provided.

In order to acquire the resultant 14 bit data word used to drive the gyros, the double precision words are merged by using bits 8 - 14 of the low order word and bits 1 - 6 of the high order word. This merging operation is explained in detail on Figure 3-17. A check is made to see if there is more than, less than, or exactly 16383 pulses to be sent.

If there is exactly 16383 pulses to *go* a logic 1 is set in bit 14 of the merged data word and the entire word is placed in the "GYROCMD" output register and the task "START GYRO" is scheduled to be executed in X sec. (Where X will be calculated based on the number or pulses required. If the number of pulses is less than 16383, the high order word is examined to see if bit 7 = 1. If *so,* it is added to the merged word before it is placed in the GYROCMD output register and the "START GYRO" task is scheduled to be executed in X sec.

If, however, the original DP pulsing words indicated more than 16383 pulses, bit 14 is added to the merged data and is set in the "GYROCMD" register. Now, a different task is scheduled for X sec. The task "8192 AUG" is used for large pulse train outputs and is rescheduled as long as the required output after sequential loops exists. NOTE: Each time thru task "8192 AUG" 8, 192 pulses are added to the GYROCMD register. After sufficient pulse trains have been sent to reduce the required pulses to zero, task "STRT *GYRO"* is scheduled to complete the pulsing requirements of the other gyros. Note that in any case, the final action is to set bit 10 of channel 14 = 1 causing the gyro drive sequence.

The routine for pulsing the *gyros* does the following:

a. Determines direction and amount of pulsing.

b. Services and reschedules itself.

c. Identifies desired gyro to be serviced *(Y Z X).*

d. Terminates itself.

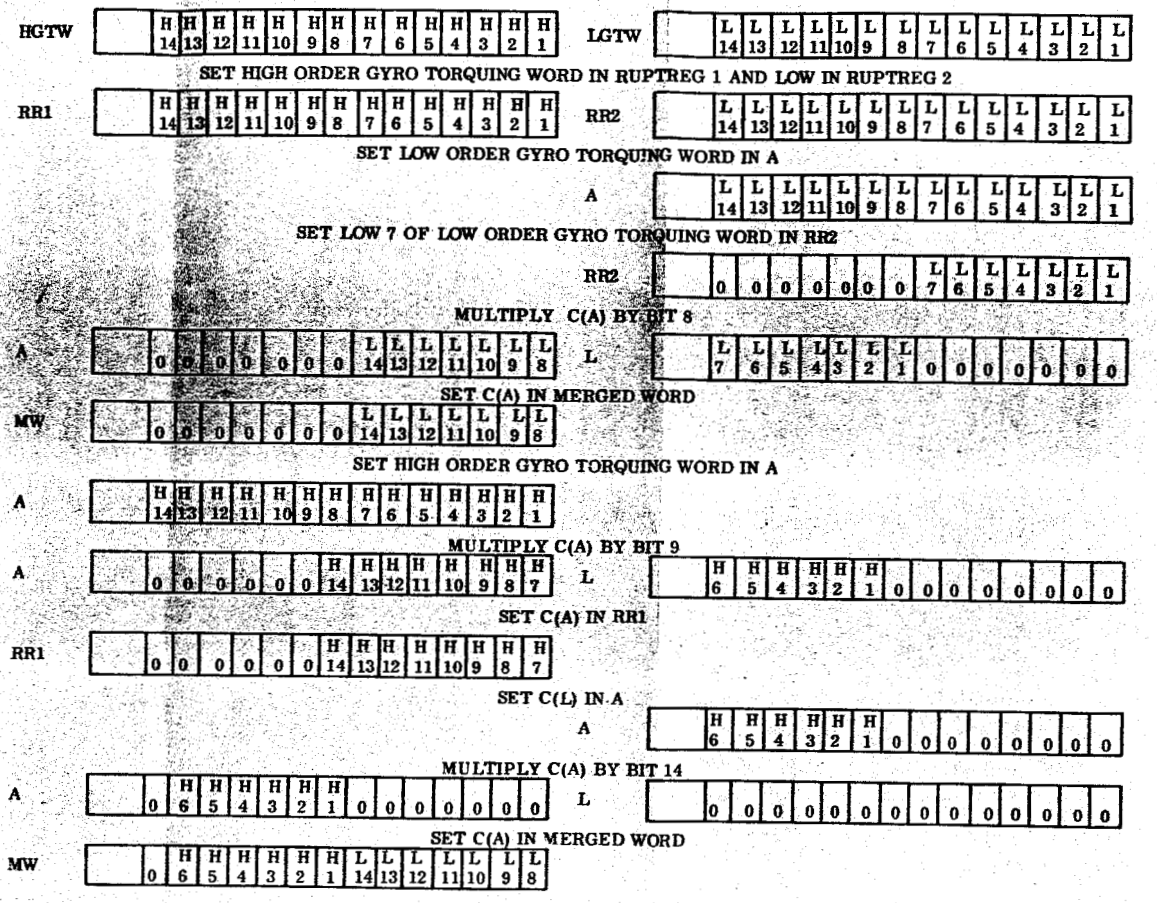


Figure 3-17. GENERATION OF MERGED WORD

3.6 AOTMARK ROUTINE

The LM Optical Sighting MARK Routine, AOTMARK, incorporates two different star sighting procedures to implement alignment of the LM IMU during periods of free fall and prior to launch from the lunar surface. A description of AOTMARK is presented after a brief description of the Alignment Optical Telescope (AOT) and the procedures used to obtain star data during the in-flight and non-flight modes. A more detailed description of the LM optics can be found in MIT R-466. Also included, figure 3-20 is a flow diagram showing the astronaut displays and the general functions of the LGC during the sighting MARK routine.

3.6.1 ALIGNMENT **OPTICAL TELESCOPE (AOT).** The AOT is a unity power periscope with a *60* degree field of view. The shaft axes of the telescope is parallel to the X-axis of the LM.

The center of the field of view forms an angle of 45 degrees with the LM thrust or X-axis. By means of a pinion knob, the astronaut may rotate the telescope head assembly about the shaft axis. This shaft angle rotation, shown in figure 3-18, is detented at three viewing positions: the vehicle XZ plane (zero rotation), 60 degrees to the left, and *60* degrees to the right.

Since the shaft rotation detents and the corresponding centers of fields of view (elevation) will not be exactly ± *60* degrees and 45 degrees respectively, a table of the actual values corresponding to the particular AOT in use will be stored in the LGC memory. The astronaut need only specify one of three code numbers (DETENT) listed below to obtain the correct values of azimuth and elevation.

The AOT reticle pattern is shown in figure 3-19. The pattern consists of two straight lines and a spiral which is so constructed as to depart radially from the center as a linear function of the rotation about the center. The astronaut can rotate the entire reticle pattern about the center of the field of view by turning a knob near the eyepiece. A micrometer readout is provided near the knob to indicate the amount of reticle rotation.

3.6.2 NON-FLIGHT STAR SIGHTING. To perform a star sighting from the lunar surface, the AOT shaft is rotated to one of the three viewing positions (DETENT) such that the desired star falls within the field of view. The astronaut rotates the reticle pattern until the *Y* reticle line intersects the star, The micrometer dial *is* read and the rotation angle *(Y* ROT) is recorded. The astronaut then rotates the pattern until the spiral intersects the star and this rotation *(S* ROT) is recorded. These two angles keyed in by the astronaut along with the DETENT code provide sufficient data to determine a star direction. Only one MARK is required for a star sighting for the non-flight mode. The X MARK button is used to perform this MARK.

3.6.3 IN-FLIGHT STAR **SIGHTING.** During free fall the astronaut uses only the straight lines of the reticle pattern to perform a star sighting. The reticle pattern is set at zero rotation and the spacecraft attitude is changed *so* as to produce crossings of the *Y* and X reticle lines by the navigation star. When the star intersects the Y reticle line, the astronaut presses the *Y* MARK button, The X MARK button is pressed when the star intersects the X reticle line. Since the crossing of a reticle line by a star defines a plane containing the star, the crossing of two different lines by a single star defines the direction of the star. In addition to the two MARKs, the DETENT code must be entered by the astronaut in order to complete the star sighting.

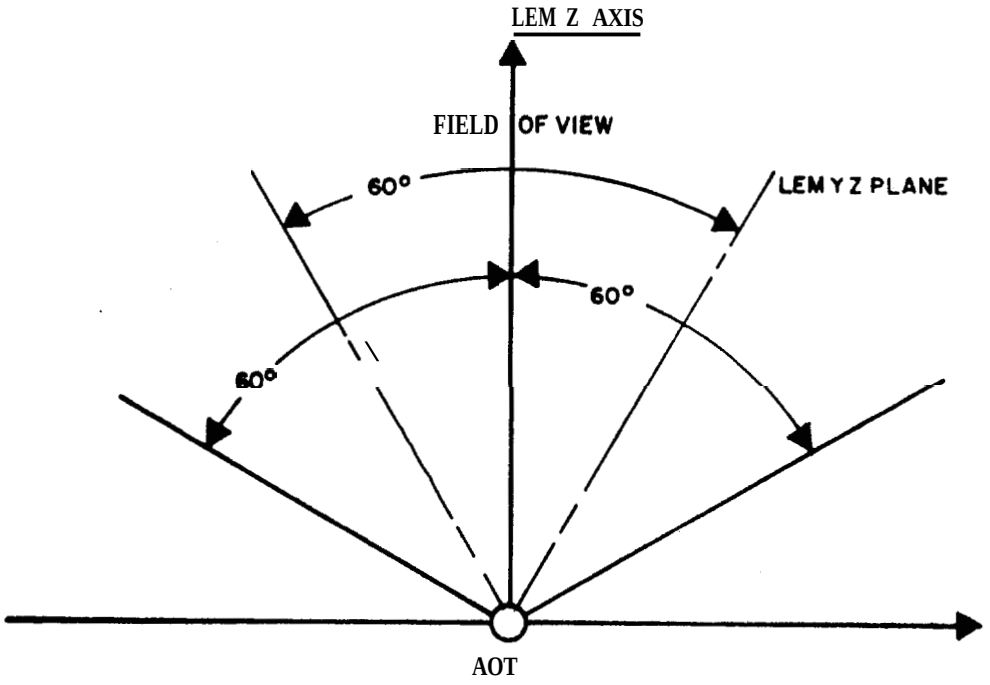


Figure 3-18, LM AOT Azimuth Positions

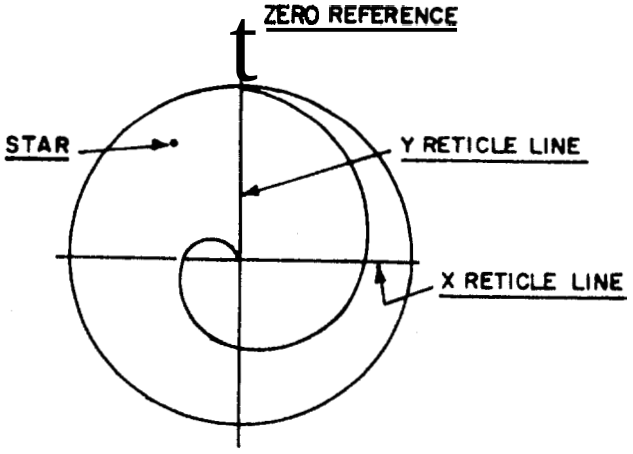


Figure 3-19. **AOT Reticle Pattern**

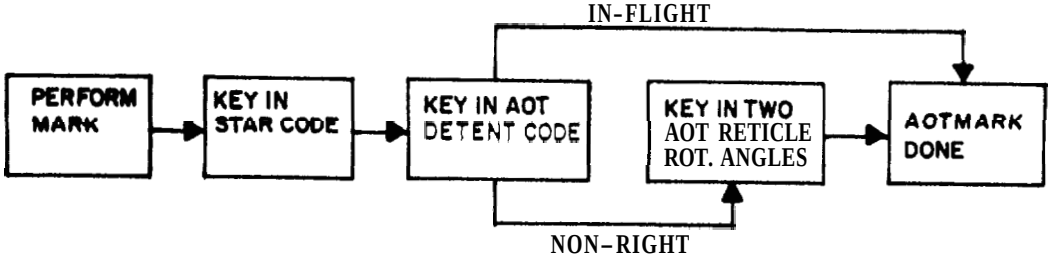


Figure 3-20. Basic Inflight Star Sighting Sequence

Although the optical sighting procedures are different for the two flight modes, the sequence of operations performed by the astronaut are the same as-illustrated in figure 3-20.

During the MARKing phase of the routine, one of three verbs will be displayed to indicate to the astronaut the MARKs that are wanted. The following is a description of these verbs,

Verb 51 — This is a request to the operator to perform a MARK using the X MARK button. During in-flight operations this request would normally appear after a Y MARK has been performed. During non-flight operations this verb is the only MARK request to appear.

Verb 52 — This is a request to the operator to perform a Y MARK, This verb will not appear as a MARK request during non-flight operations,

Verb 53 — This verb is normally used during in-flight operations to indicate to the operator that two MARKs (an X MARK and a *Y* MARK) are wanted. The operator ie free to press either the X or *Y* MARK button at his convenience.

After the required MARKs have been made, Verb 21, Noun *30* will be displayed indicating that the sighted star code is wanted. If the operator is satisfied with the MARKs, he will key in the star code and press the ENTER button showing MARKs accepted (MK ACCEPT).

3.6.4 AOTMARK ROUTINE. The AOTMARK routine is called up for each star sighting which prepares the LGC to accept MARKs and sighting data for one of the two flight modes. For in-flight alignment of the LM IMU, both an X MARK and a *Y* MARK are required and for non-flight alignment only an X MARK is required.

The program sets up a VAC area for storage of the MARK and sighting data and stores the VAC area address and desired flight mode in MARKSTAT as follows:

BITS

1-9 VAC Area Address

10 = *0* Initially to indicate an X MARK ie wanted
= 1 after an X MARK is performed

11 =0 Initially to indioate a *Y* MARK is wanted (in-flight only)
=1 after a *Y* MARK is performed

12 =0 while MARKs are being performed
=1 after a MK ACCEPT (ENTER)

13 =0 after each MARK
=1 after each MK REJECT

14 *-0* for non-flight MARKs
=1 for in-flight MARKs

15 =0

Upon entering AOTMARK, c(MARKSTAT) is tested and if found to be > + 0, the MARK buttons are busy and Alarm (105) is fired and the job terminated. If + 0, the buttons are available and an idle VAC area is found. If none is available, Alarm (1207) is fired and the job aborts, Refer to figure 3-21.

After a VAC area is found and reserved, the GETMKS job is scheduled via the executive NOVAC routine. **Job GETMKS** initiates to appropriate verb code to be flashed (V53 for inflight or V51 for non-flight). The job is then put to sleep until the star code is entered via the DSKY.

Upon receiving a mark, the MARK RUPT routine is entered. The **ISS** CDU angle and time are recorded and stored for later use and control is transferred to the appropriate routine to process the input (mark **X,** mark ***Y,*** mark reject or descent bit) which caused the MARK RUPT.

If the MARK RUPT was caused by a mark **X** or **Y,** a check is made to insure that the particular mark was not made previously c(MARKSTAT). If the particular mark was marked a second time, the PROGRAM CAUTION lamp is lit along with the failure code 00114. If not marked twice, the CDU **X,** Y, Z and time information is transferred into the VAC area reserved for the AOTMARK and the REMARK routine is performed.

REMARK schedules the CHANGE **VB** job via the executive and sets up the information used in changing the verb display. CHANGE VB causes the appropriate verb code to be displayed, V52 **is** displayed if a MARK ***Y*** **is** needed, V51 is displayed if a MARK **X** is needed, V53 is displayed if a MARK REJECT occurred (INFLIGHT) **or** V21N30 is displayed if both marks have been entered.

Upon entering the star code, MK CHEK **is** performed. MK CHEK checks for **both** marks, accepts marks and requests the AOT detent code V21 N43. After entering the AOT detent code, the AOT elevation and azimuth detent calibration is stored in the VAC area for later use and the tilt compensation is calculated and stored in the VAC area. If this is a non-flight AOT mark, the AOT reticle angles must **also** be loaded. **The** VAC area reserved for the AOTMARK routine will contain the following information for the two flight modes.

	INFLIGHT	NON-FLIGHT
VAC	TIME 2	TIME 2
VAC+1	TIME 1	TIME 1
VAC+2	CDU ***Y*** (XMARK)	CDU **Y**
VAC+3	CDU Y (YMARK)	***Y*** ROT
VAC+4	CDU **Z** (XMARK)	CDUZ
VAC+5	CDU *Z* (YMARK)	S ROT
VAC+6	CDU **X** (XMARK)	CDU **X**
VAC+7	CDU X (VMARK)	BLANK
VAC+8	AZIMUTH	AZIMUTH
VAC+9	ELEVATION	ELEVATION

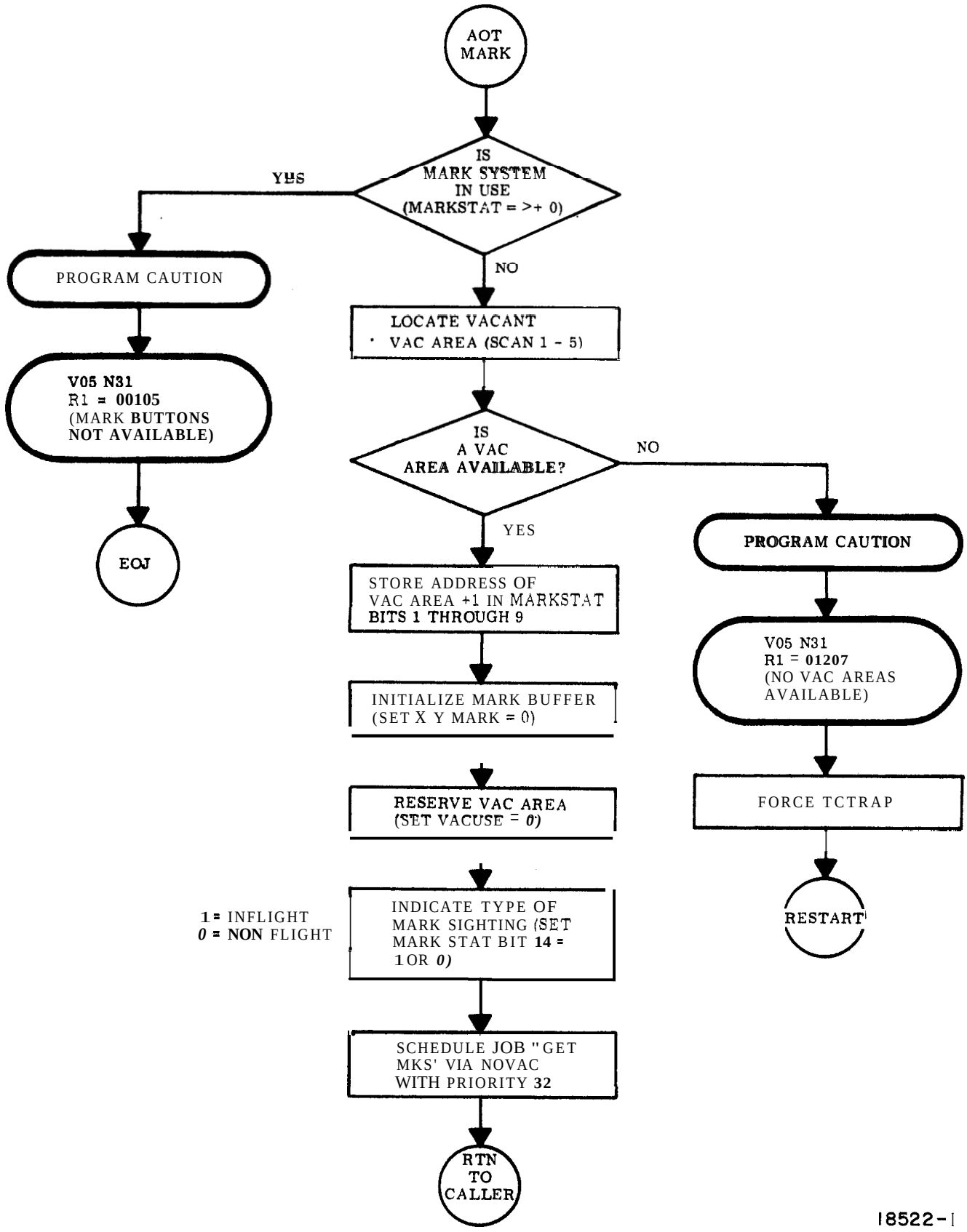


Figure 3-21. AOTMARK Routine *(Sheet* **1** of 9)

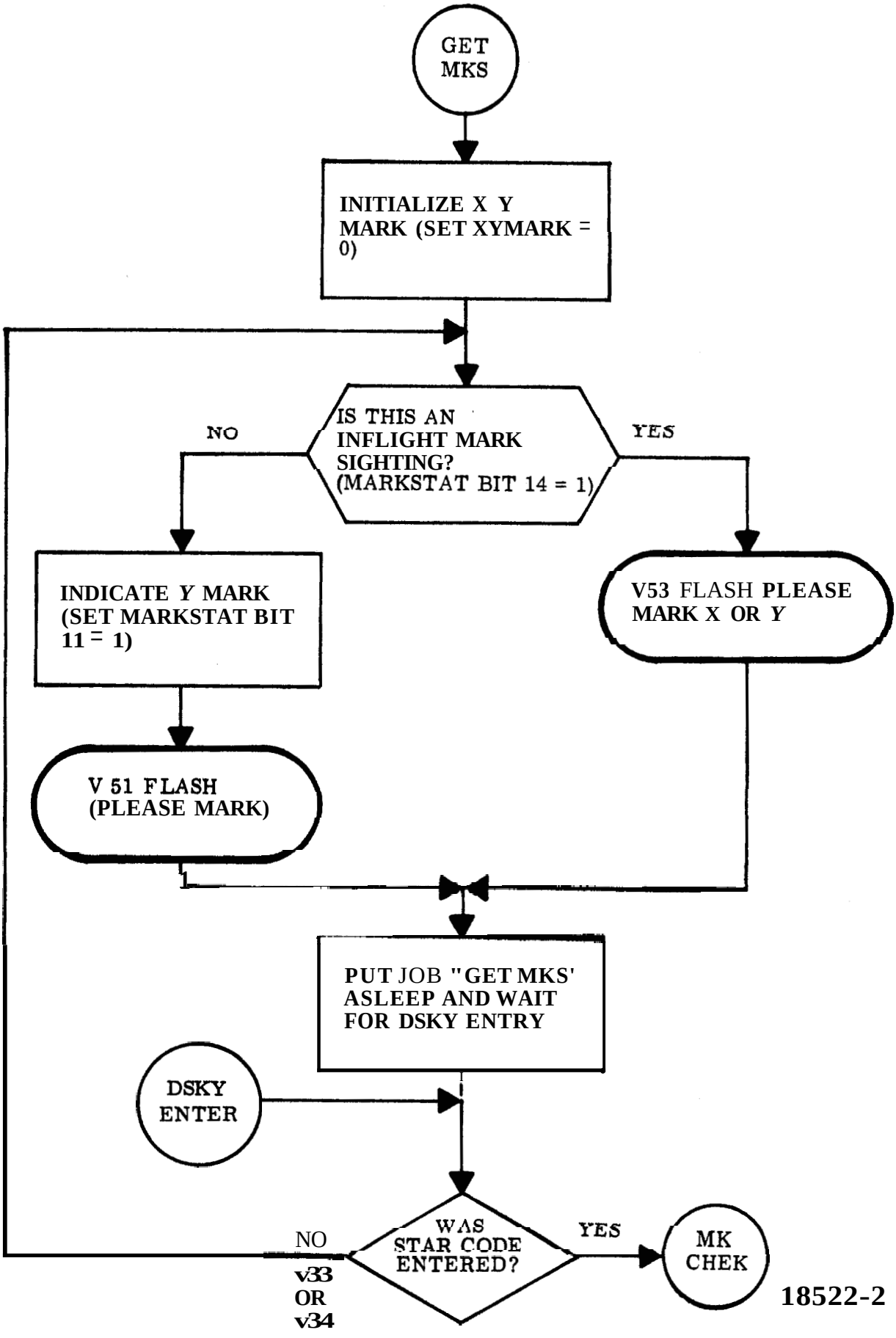


Figure 3-21. AOTMARK Routine (Sheet 2 of 9)

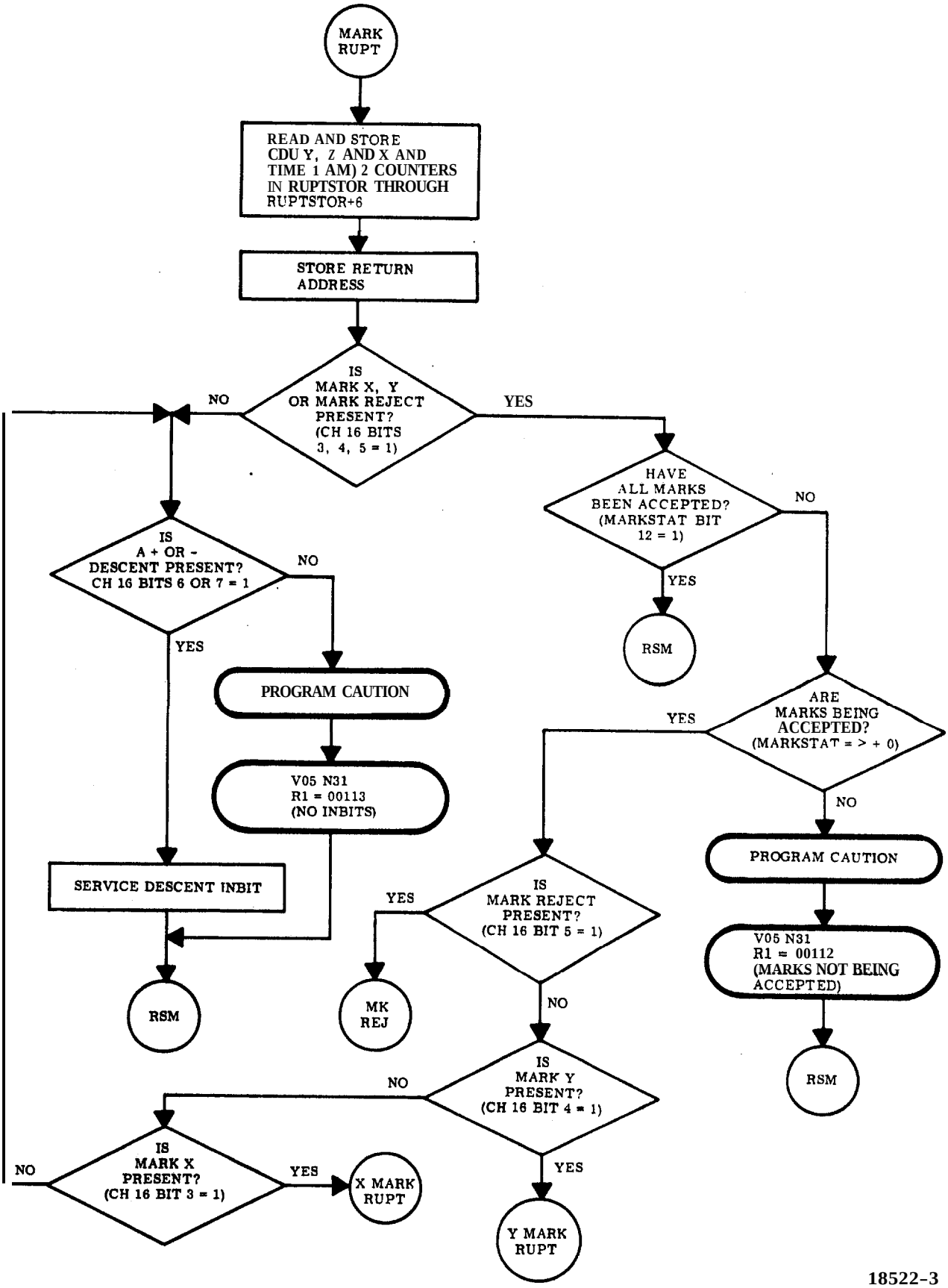
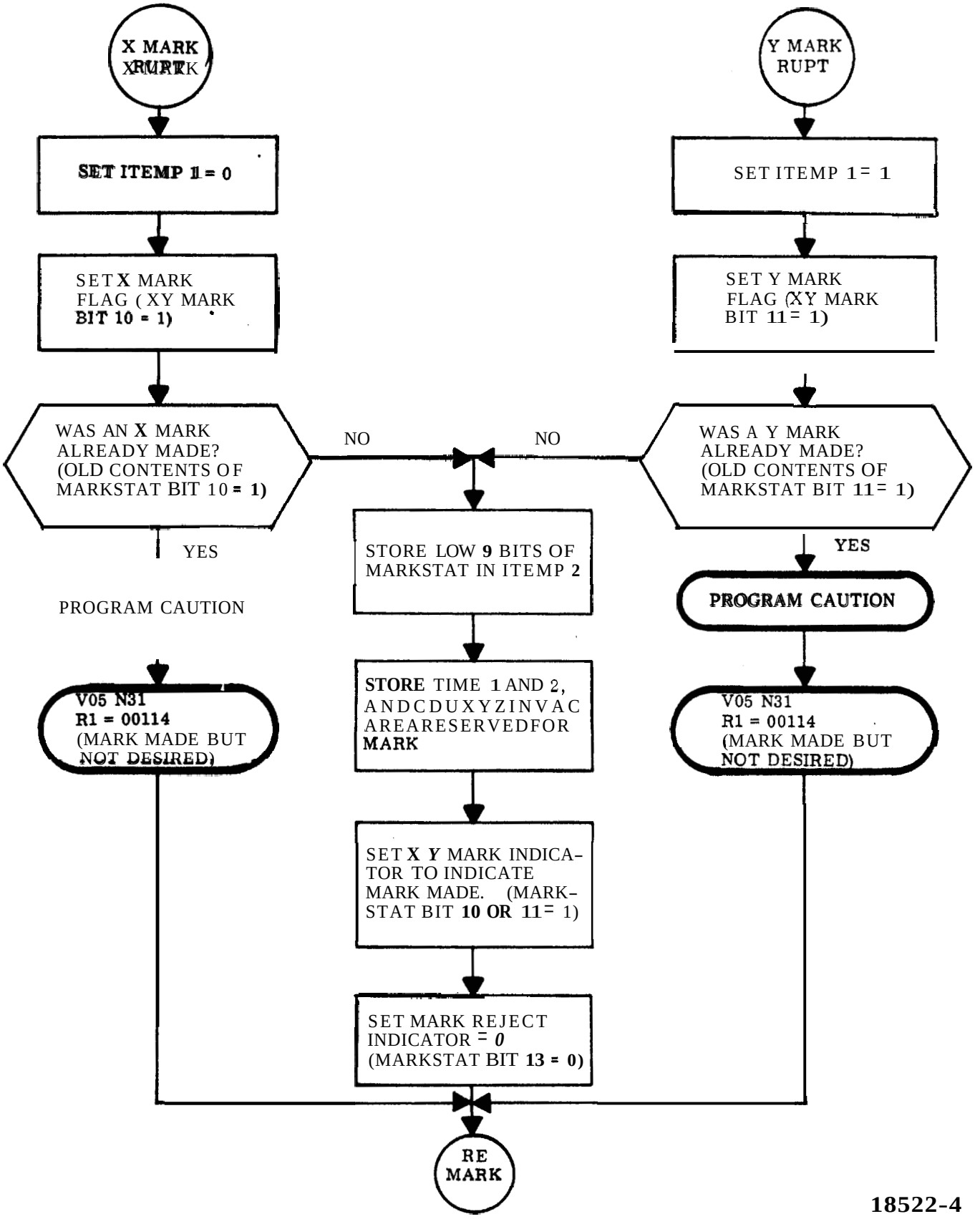


Figure 3-21. AOTMARK Routine (Sheet 3 of 9)



18522-4

Figure 3-21. AOTMARK Routine (Sheet 4 of 9)

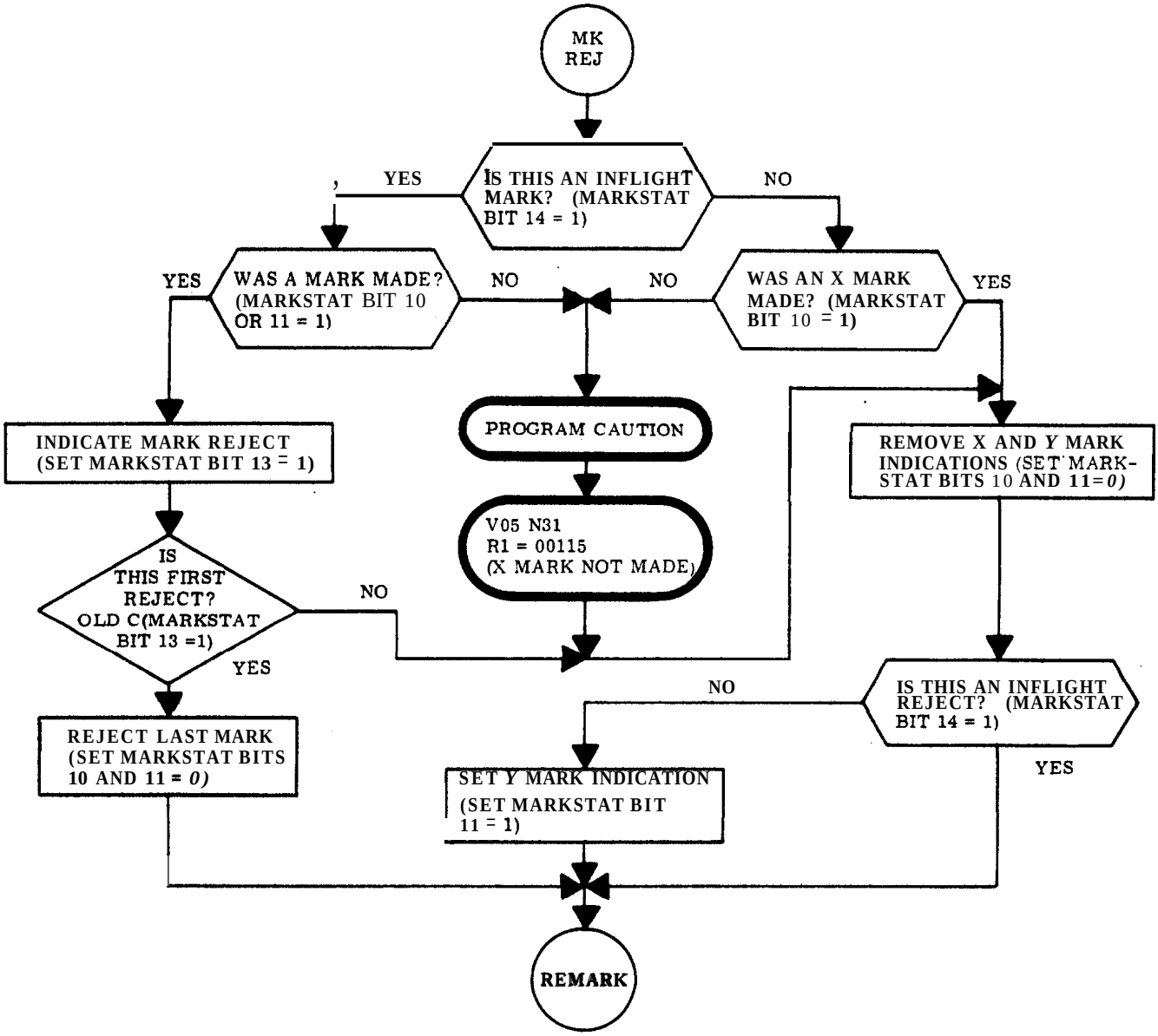


Figure 3-21. AOTMARK Routine (Sheet 5 of 9)

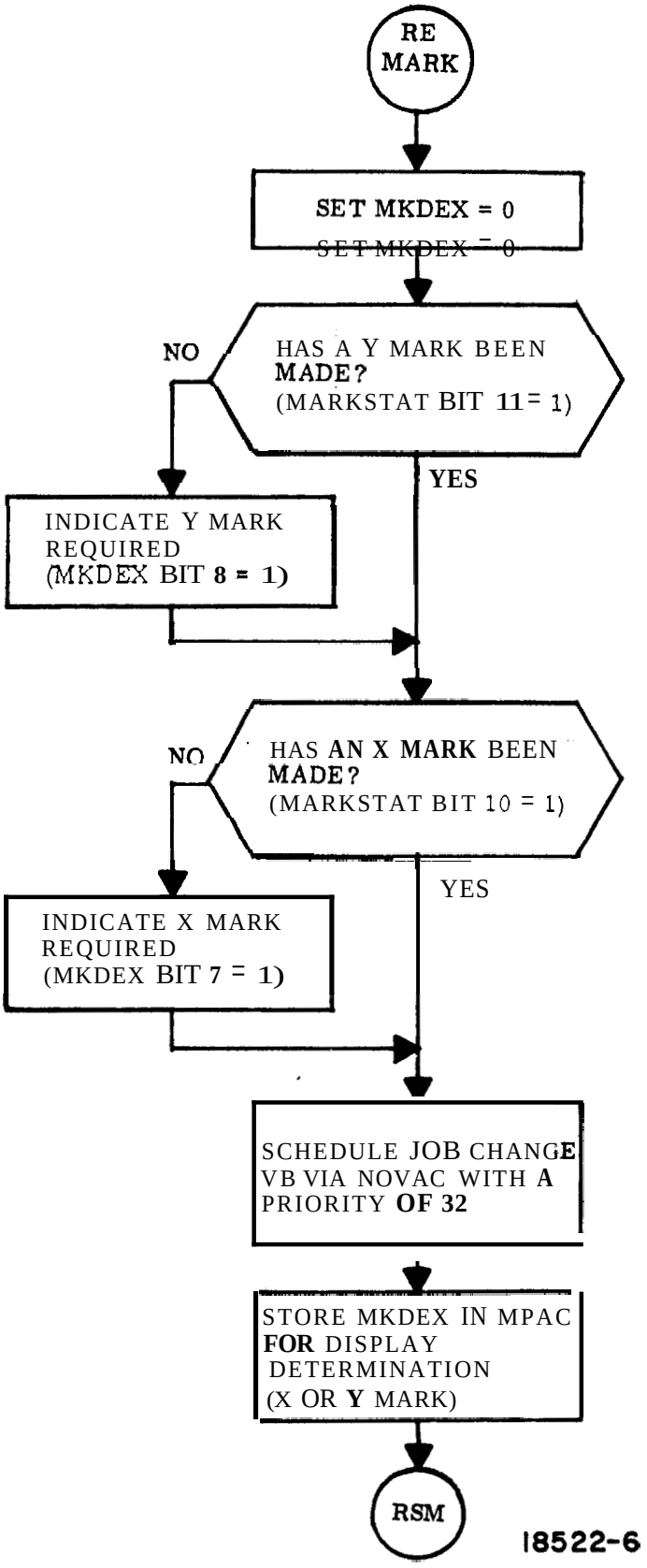


Figure 3-21. AOTMARK Routine (Sheet *6* **of 9)**

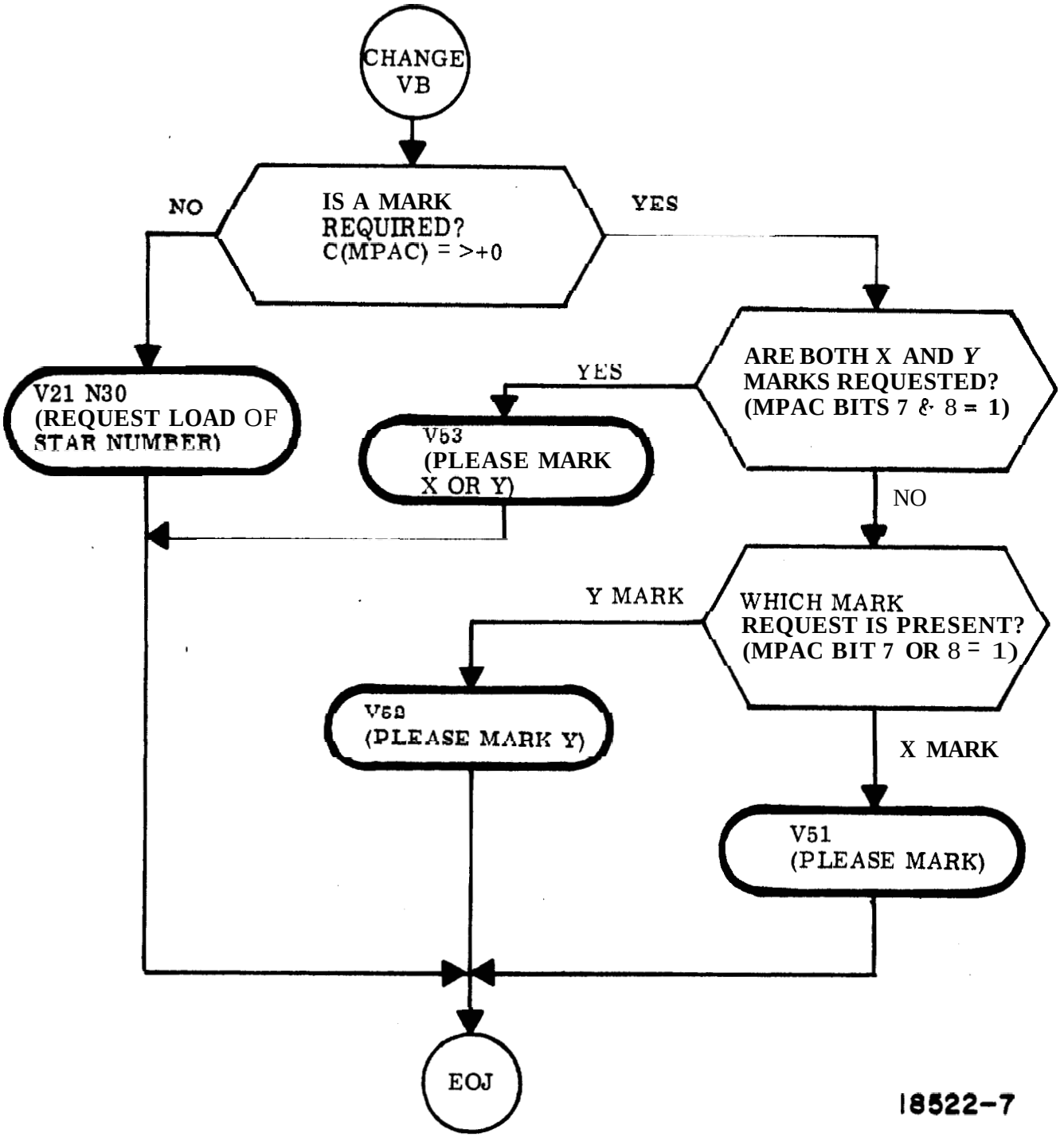


Figure 3-21. AOTMARK Routine (Sheet 7 of 9)

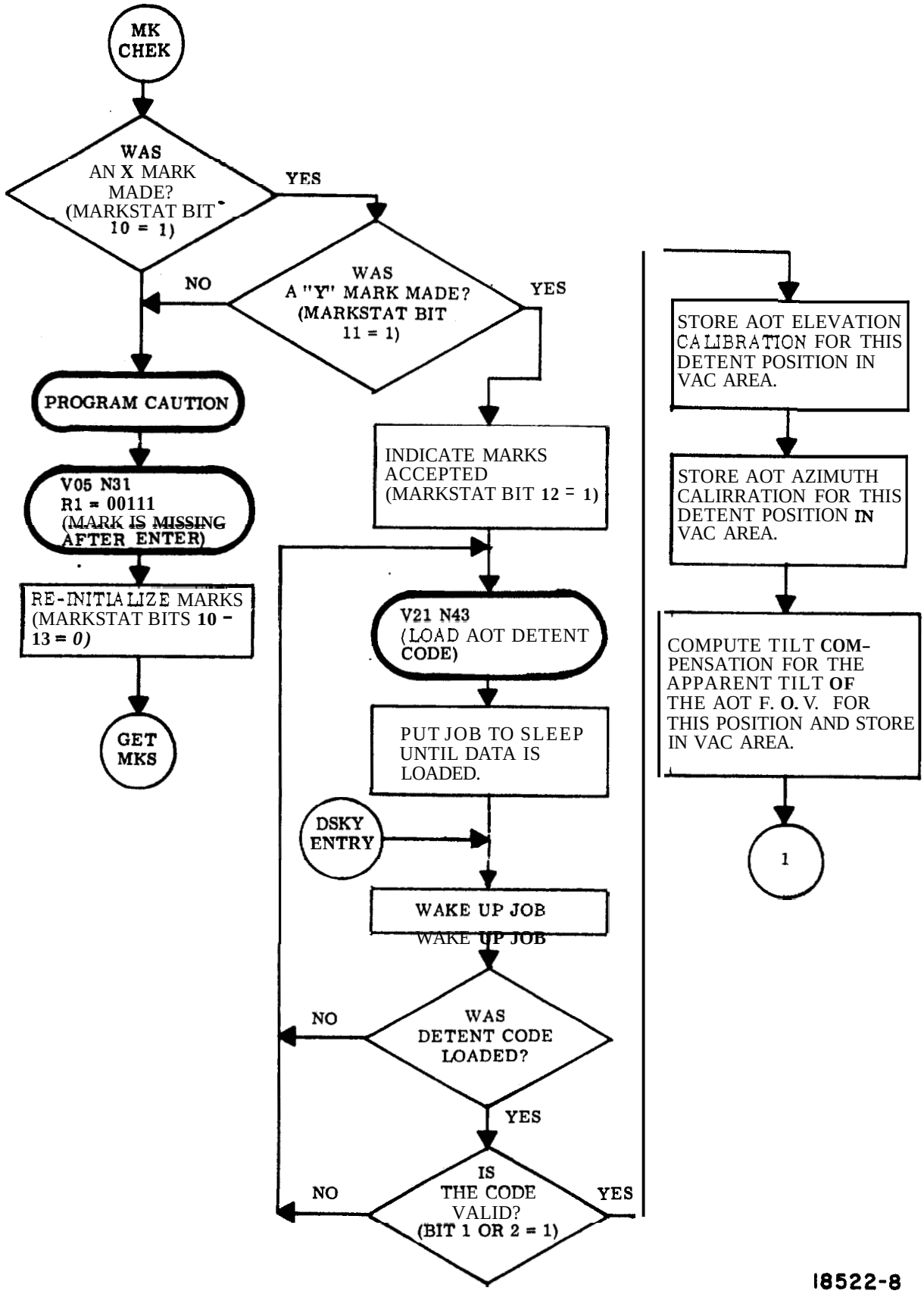


Figure 3-21. AOTMARK Routine (Sheet 8 of **9)**

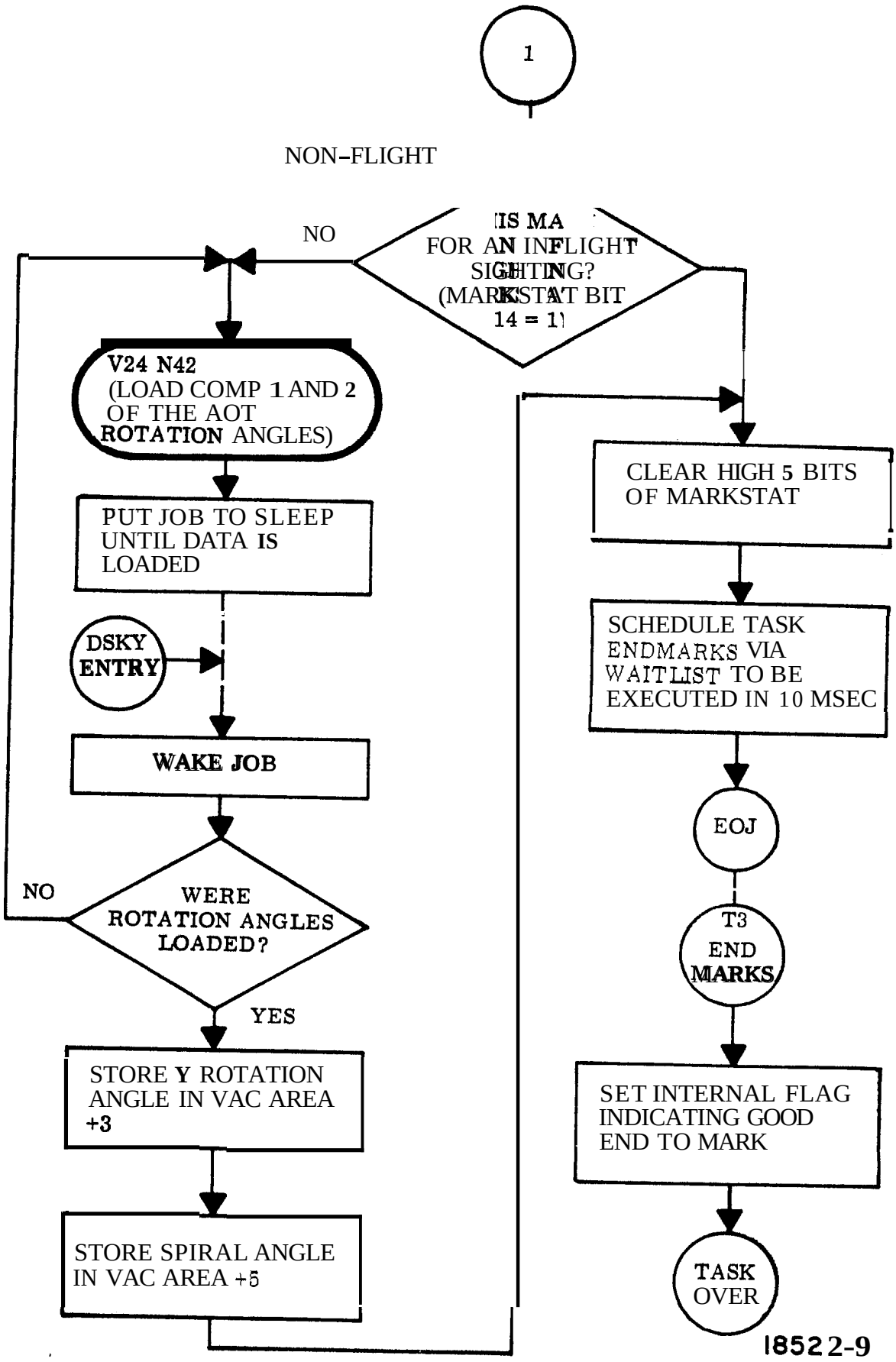


Figure 3-21. AOTMARK Routine (Sheet 9 of 9)

SECTION IV

MISCELLANEOUS ROUTINES

INTRODUCTION

This section of the study guide presents routines that perform various functions which are not categorized in the other sections of the study guide. The routines presented in this section are as follows:

a. PROGRAM ALARM

b. PROGRAM ABORT

c. FRESH START & RESTART

d, SELF-CHECK

4.1 PROGRAM ALARM ROUTINE

The Program Alarm routine is used by all programs which require the display of a program alarm condition. The routine illuminates the Program Caution indicator and causes Verb 05, Noun 31 to be displayed'and a failure number to be displayed in R1, Verb 05, Noun 31 indicates; display octal component 1, 2, 3 - FAILREG, SFAIL, ERCOUNT. The failure number displayed in R1 is supplied by the processing function which is using the Program Alarm routine. This number indicates what failure condition was detected. Table 4-1 lists the failure numbers for the program detected failures processed by this routine.

The program alarms processed by this routine are of a nature which does not require the restarting of the computer operations, Other program caution conditions which require a restart, are processed by the Program Abort routine.

The flow chart for the Program Alarm routine is shown in figure 4-1. Control is routed to this routine by an internal calling program whenever a program alarm condition is detected. The calling program also provides the failure number. Upon entry to the routine, the return address of the calling program is stored. Then, a check is made to determine if this is the first alarm condition since the ERROR RESET key of the DSKY was used. If it is the second failure, bit 15 of FAILREG is set to a binary 1 to indicate a multiple failure and control is returned directly to the calling program using the stored return address. It should be noted that if the T4RUPT routine has already provided the error number for the first failure to the DSKY display (R1) the multiple failure will not be indicated to the operator. If, however, the second error is detected and bit 15 of FAILREG is set before the display R1 will indicate 4XXXX. If it is the third or more failure, control is returned directly to the calling program.

If this is the first alarm condition since an Error Reset, bit 9 of output channel 10 is set to a binary 1 to illuminate the PROGRAM CAUTION lamp, Then the job DOALARM is scheduled to be executed using the Executive's NOVAC subroutine. After performing the scheduling, the FAILREG (FAILURE NUMBER REGISTER) is set to the failure number supplied by the calling program and control is returned to the calling program using the return address.

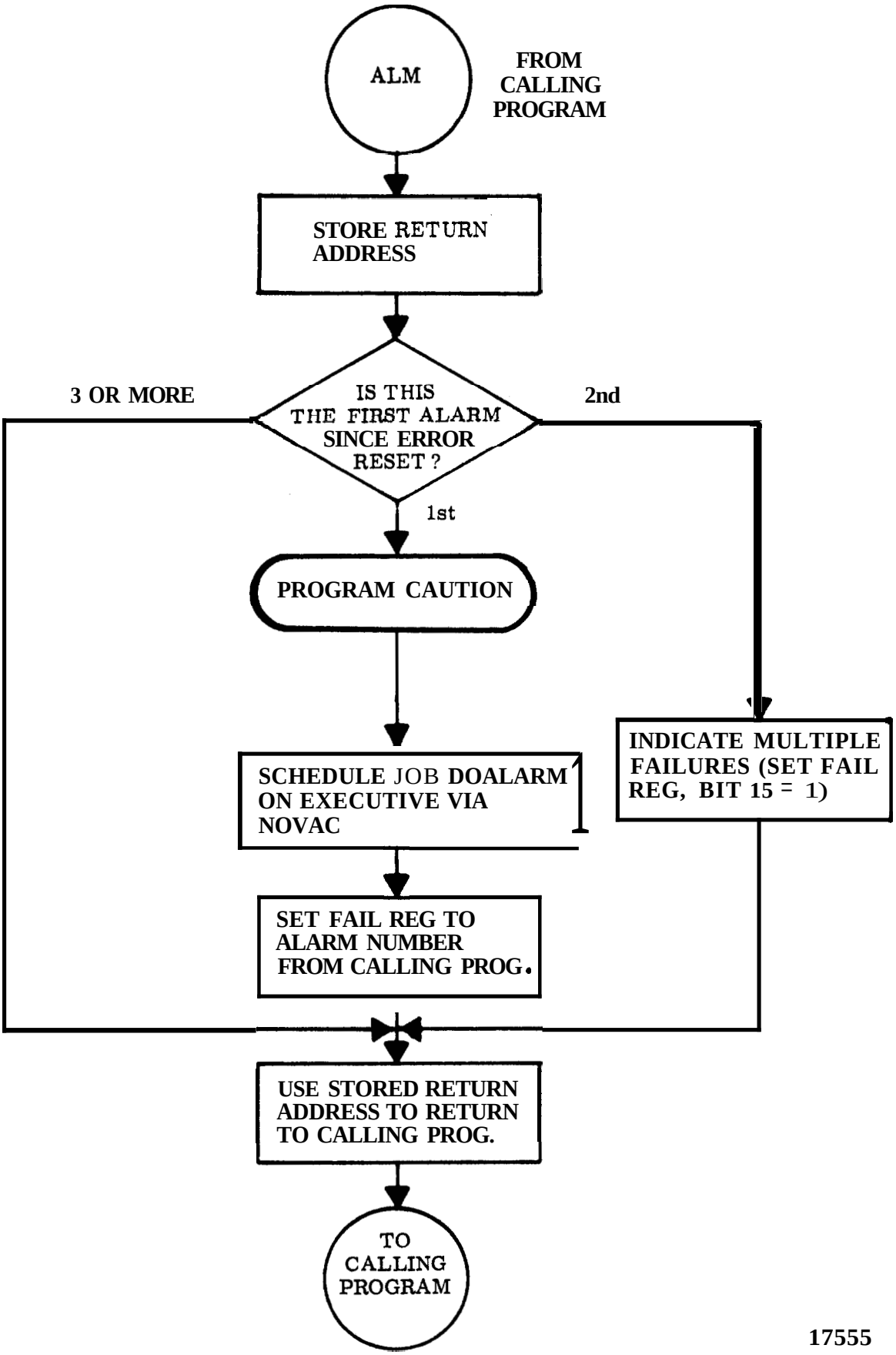


Figure 4-1. Program **Alarm** (Sheet **1** of **2)**

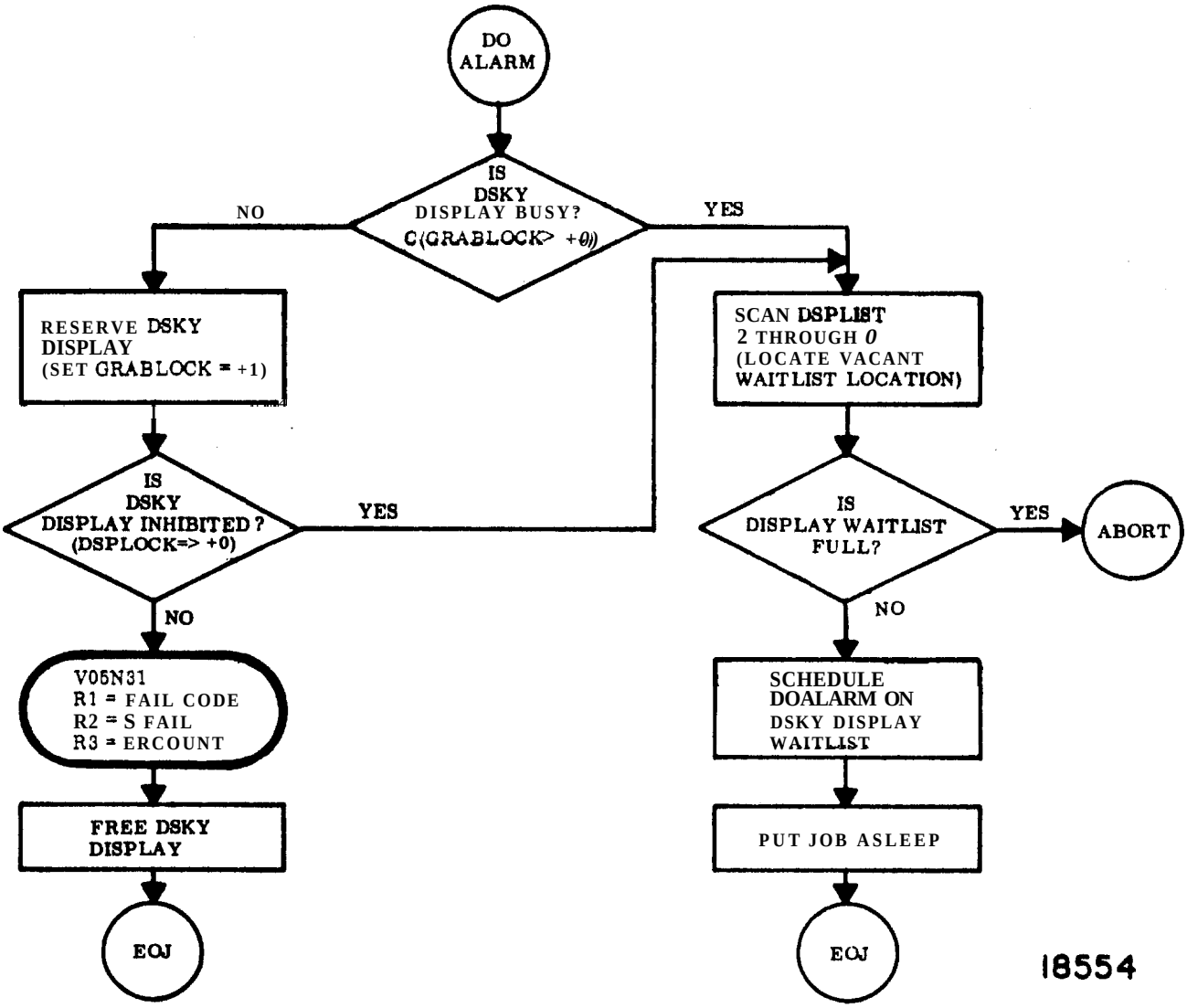


Figure 4-1. Program Alarm (Sheet 2 of 2)

When the processing of the DOALARM job is initiated, it obtains the use of the DSKY's through the **normal** procedure. Having obtained the use of the **DSKY** the display table is set to display Verb **05,** Noun **31** and the contents of **FAILREG** (the failure number) in **R1.** The **DSKY** interlock is then released and the DOALARM job is terminated by routing control to the End Of Job routine.

Table **4-1.** Failure Numbers For Program Alarms

Prog. Alarm No.	**Prog.** Alarm Condition
OPTICS SUB-SYSTEM	
00105	Mark Buttons Not Available
00111	Mark Is Missing After Enter
00112	Mark Not Being Accepted
00113	No Inbits
00114	Mark Made But Not Desired
00115	**X** Mark Not Made
INERTIAL **SUB-SYSTEM**	
00206	Zero CDU Not Allowed With Coarse Align or Gimbal **Lock**
00207	ISS Turn-on Request Not Present For **90** Sec.
00210	IMU Not Operating
00211	Coarse Align Error
00212	PIPA Fail But PIPA Is Not Being Used
00213	IMU Not Operating With Turn-on Request
00214	Program Using IMU When Turned OFF

Table 4-1. Failure Numbers for Program Alarms (Cont)

Prog. Alarm **No.**	Prog. Alarm Condition
PROCEDURAL DIFFICULTY	
00401	Desired Gimbal Angles Yield Gimbal **Lock**
00402	Star Out Of Field Of View
00403	Star Out Of Field Of View
RADAR ERRORS	
00501	Radar Antenna Out Of Limits
00502	Bad Radar Gimbal Angle Inputs
00503	Radar Antenna Designate Fail
00510	Radar Auto Discrete Not Present
00514	Radar *Goes* Out Of Auto Mode While Being Used
00520	No Radar Rupt Expected
00521	Radar Data Could Not Be Read
00522	Wrong LR Position
00523	LR Antenna Did Not Make It
00524	Bad Radar Target
COMPUTER HARDWARE MALFUNCTIONS	
01102	AGC Self Test Error
01105	Downlink Too Fast
01106	Uplink Too Fast
DISPLAY ALARMS	
01400	Pitch And/Or Roll Trim Fail **Is** On **(Shown** With **V50N25).**
01410	Temporary Jet Fail
01411	CDU Does Not Agree With Command To 1 Degree
SYSTEM **TEST** ALARMS	
01600	Drift Test Overflow
01601	**Bad IMU** Torque in Drift, in Compass

4.2 PROGRAM ABORT ROUTINE

The Program Abort routine is used by internal programs which have detected a program abort condition. A program abort condition requires that the computer operations be restarted. Otherwise, the computations or functions being performed would be erroneous or could not be completed.

The end result of a program abort, besides the restarting of the computer's processing, is the illumination of the PROGRAM CAUTION indicator and the display of Verb 05, Noun 31 and a failure number in R1. Table 4-2 tabulates the failure numbers associated with program abort conditions. The processing performed by the program abort routine is shown in figure 4-2.

Upon entry to the Program Abort routine the failure number is available from the internal program which detected the failure condition. The first item accomplished is a check to determine if this is the first failure since the last time the ERROR RESET key was used. If it is the first failure, bit 9 of output channel 10 is set to a binary 1 to illuminate the PROGRAM CAUTION lamp. Then the FAILREG is set to the failure number which was supplied by the internal calling program.

Then, whether this is the first failure or not, a TC TRAP condition is generated. This is accomplished by having a TC (TRANSFER CONTROL) instruction transfer control it itself, After about 10 ms, the computer's TC TRAP detecting circuitry will detect this condition. When it does, a restart is forced which is similar in nature to the program interrupts. Control is forced to the Restart routine. The Restart routine among its other functions causes the display of Verb 05, Noun 31 and the failure number in R1.

4.3 FRESH START AND RESTART ROUTINE

The Fresh Start and Restart routines are closely related and will be presented together. The Fresh Start routine provides the computer with the capability of initialization when the computer is first turned on or if a major malfunction occurs which requires almost total initialization. The Restart routine also prwides an initialization function but not as complete as is performed by the Fresh Start routine. The Restart routine restarts programs at some logical point in their execution. However, when a restart is being performed and the two phase tables maintained by the Phase Table Maintenance routine do not agree, a Fresh Start is essentially performed.

Thr Fresh Start routine is entered as a result of entering Verb 36 via the DSKY or uplink.

The Restart routine is entered whenever one of the following malfunctions is detected.

a. PARITY FAIL

b. POWER FAIL

c. RUPT LOCK

d. TC TRAP

The TC TRAP condition is purposely generated by the Program Abort routine if any program abort condition exist. The processing performed by the Fresh Start and Restart routines is shown by the flow chart in figure 4-3.

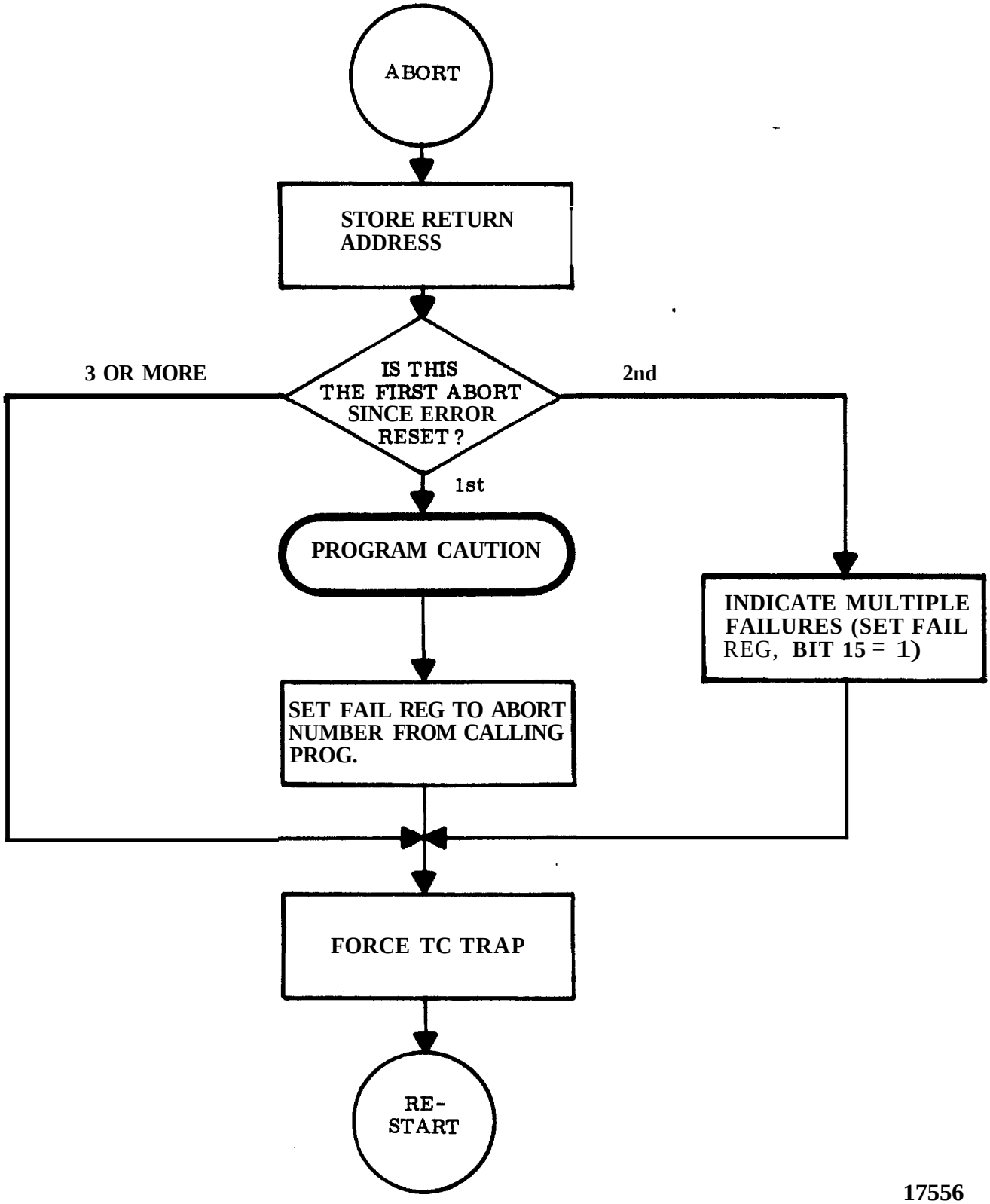
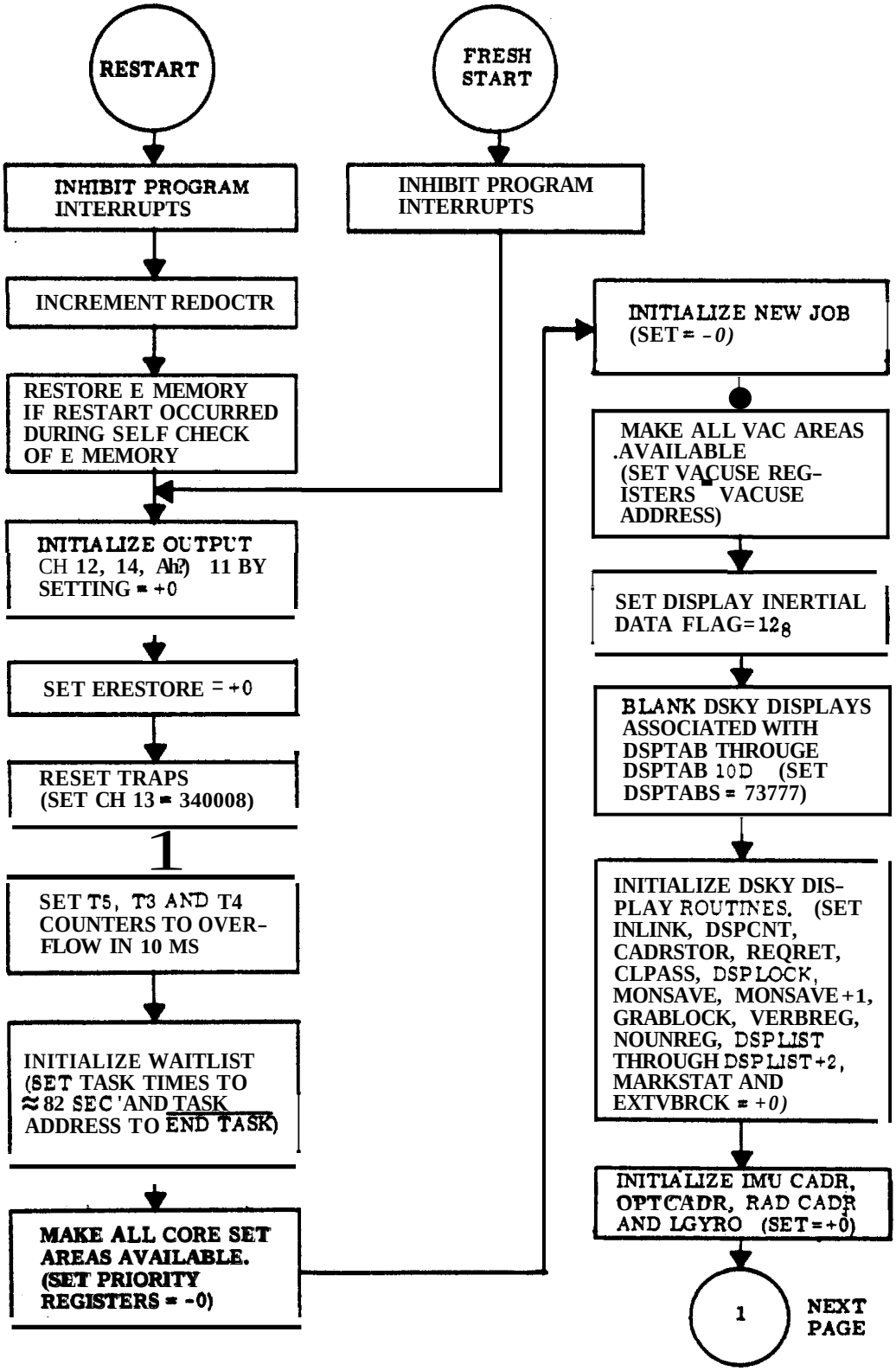


Figure 4-2. Program Abort



'Figure 4-3. Fresh Start and Restart **(Sheet 1 of** *5)*

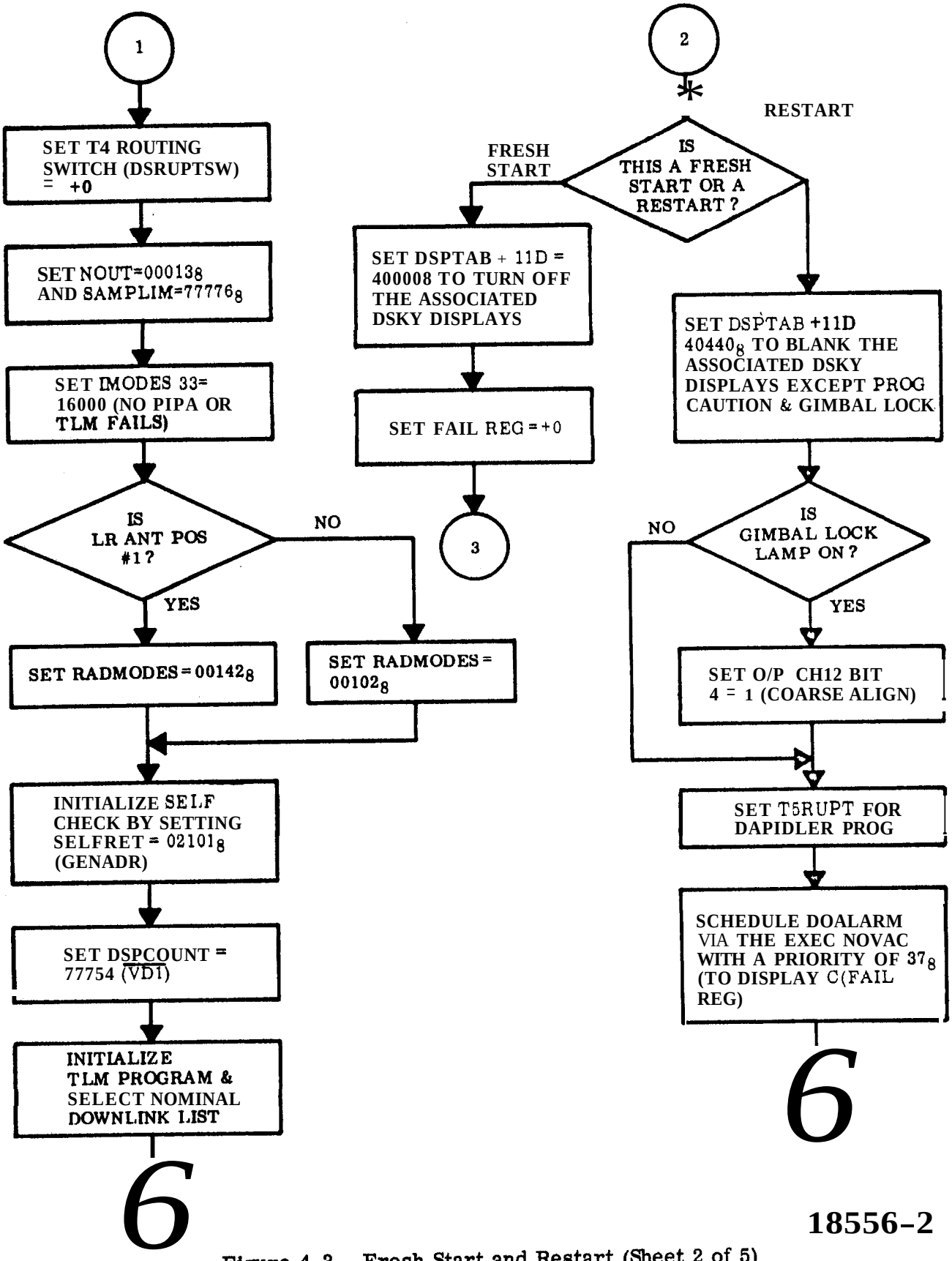


Figure 4-3. Fresh Start and Restart (Sheet 2 of 5)

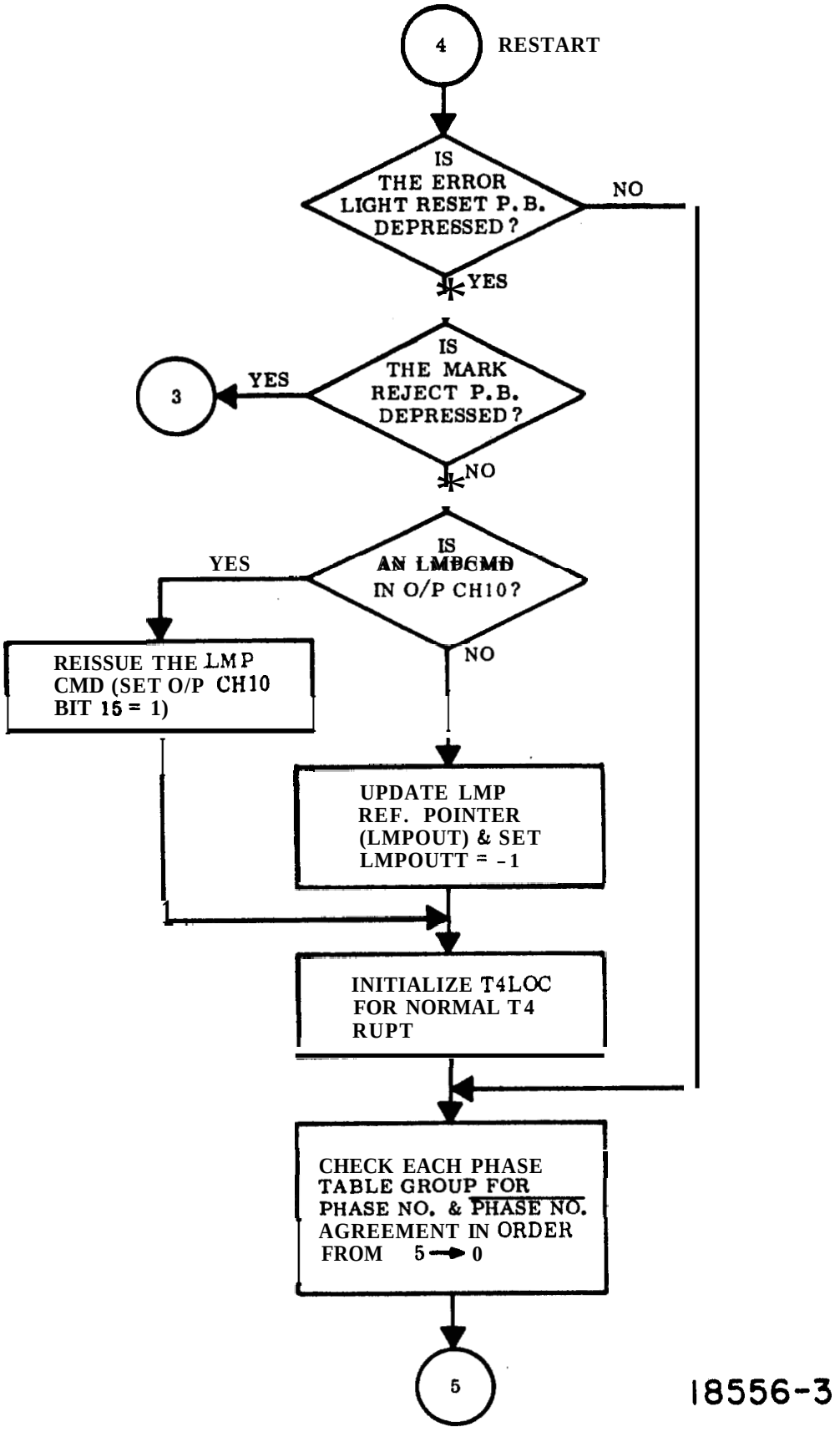


Figure 4-3. Fresh Start and Restart (Sheet 3 of 5)

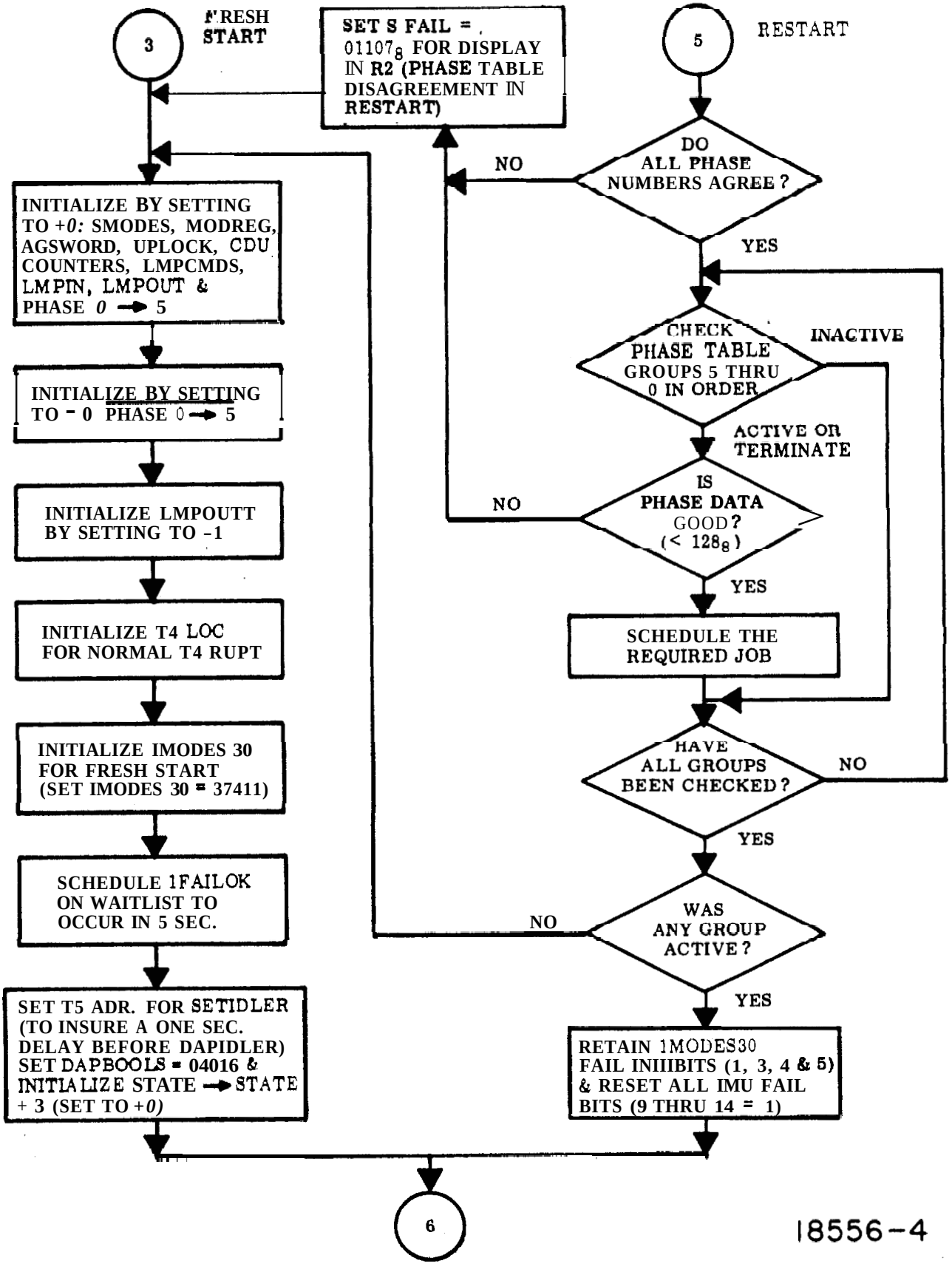


Figure 4-3. Fresh Start and Restart (Sheet 4 of 5)

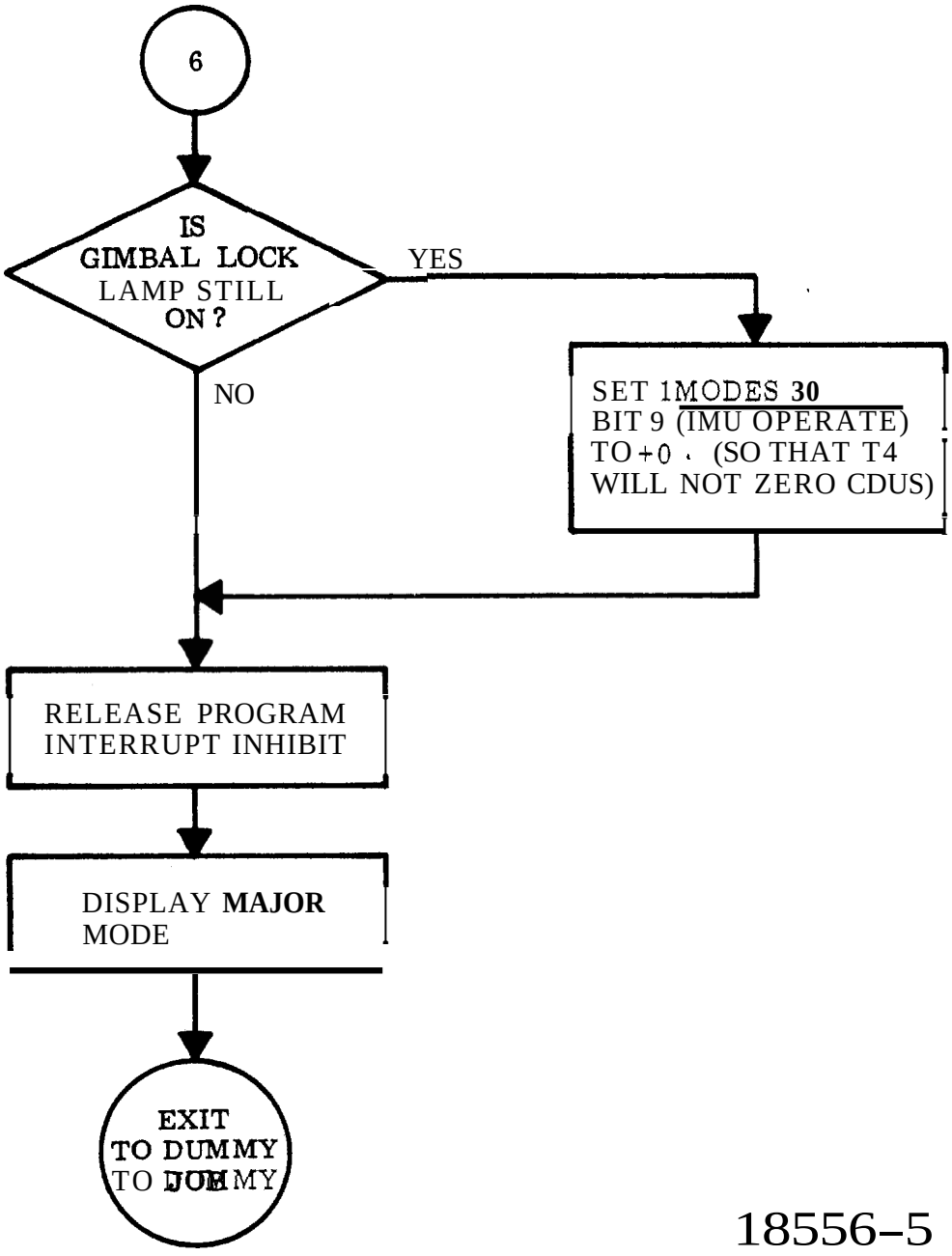


Figure 4-3. Fresh Start and Restart (Sheet 5 of 5)

Table **4-2.** Failure Numbers For Program Aborts

Prog. Abort No.	Prog. Abort Condition
COMPUTER HARDWARE MALFUNCTION	
01103	Unused CCS Branch Executed
LIST OVERFLOWS	
01201	Executive Overflow-No VAC Areas
01202	Executive Overflow-No Core Sets
01203	Waitlist Overflow-Too many Tasks
01206	Keyboard And Display Waiting Line Overflow
01207	No VAC Area For Marks
01210	Two Programs Using Device at Same Time
INTERPRETER ERRORS	
01301	ARCSIN-ARCCOS Input Angle Too Large
*0***1302**	SQRT Called With Negative Argument.
KEYBOARD AND DISPLAY PROGRAM	
01501	Keyboard And Display Alarm During Internal **Use (NVSUB)**

An entry through FRESH START or RESTART will set the computer *so* that all program interrupts will be inhibited during the routine.

If this is a RESTART the REDOCTR will be incremented to maintain the total number of times the RESTART has been performed.

If the RESTART occurs when the self check has two words removed from erasable memory these words are restored before continuing.

Output channels **12, 14,** and **11** are initialized by setting them to +0 and ERESTORE **is** likewise set to +0.

All traps are reset by setting channel **13** bits **12, 13** and **14** to logic **1.**

The TIME **5,** TIME **3** and TIME **4** counters are set to overflow in 10 MS *so* their respective routines will be executed the next time the counters are incremented.

The WAITLIST is initialized by setting all task times to approximately 82 seconds and all task addresses to ENDTASK.

All core set areas are made available for executive use by setting the priority register of each to *-0.* The register NEWJOB will likewise be set to -0.

All VAC areas are made available by setting the contents of each VACUSE register to the corresponding VACUSE address and the display inertial data flag is set to 12_8.

The blanking constant, 73777 is set into each DSPTAB through DSPTAB10D *so* that T4 will blank the DSKY displays associated with these DSPTABs when program interrupts are again allowed.

The following locations, which are concerned with DSKY display routines are initialized by setting them to *+0:* INLINK, DSPLNT, CADRSTOR, REQRET, CLPASS, DSPLOCK, MONSAVE, MONSAVE+ 1, GRABLOCK, VERBREG, NOUNREG, DSPLIST, through DSPLIST+2, MARKSTAT, and EXTVBACT.

IMUCADR, OPTCADR, RADCADR, and LGYRO are all initialized by setting to +0.

The T4 routing switch (DSRUPTSW) is initialized by setting it to +0.

NOUT is set to octal 13 to serve as a flag for the T4RUPT routine to indicate that DSPTAB *0* through DSPTAB 10D all require interrogation and consequently a change in the DSKY displays associated with these DSPTABs.

IMODES 33 FAIL bits are reset, and RADMODES is initialized.

The SELFRET register is set *so* that when the computer first enters self check after a FRESH START or RESTART it will enter at the beginning.

DSPCOUNT is initialized to octal 23, the telemetry program is initialized and the nominal downlink list is selected for transmission of downlink data.

DSPTAB 11D is set to blank all associated lamps on the DSKY if this is a FRESH START, or to blank all except PROGRAM CAUTION and GIMBAL LOCK if this is a RESTART.

If this is a RESTART the status of the coarse align enable (channel 12, bit 4) will depend on the condition of the GIMBAL LOCK lamp,

The job DOALARM is scheduled with a priority of 37 to set up the DSPTABs associated with R1 *so* that the contents of FAILREG will be displayed.

Further action in the RESTART routine is dependent upon the condition of the MARK REJECT and the DSKY RSET pushbuttons. If the MARK REJECT and the DSKY RESET pushbuttons are pressed, the computer will essentially perform a FRESH START. Assuming this is not the case, the LM mission programmer command pointer is updated and if a LMPCMD was in progress, the command is reissued, (T4LOC) is set for a normal T4RUPT, and the phase tables are checked for agreement. The two registers in each phase table group should contain the complement of each other for agreement. If any group disagrees or if the phase data is bad (contains a phase larger than 127 decimal) octal code 01107 will be processed for display in R2. (Phase Table Disagreement DOFSTART) and the computer will proceed as if in FRESH START,

After all phase tables have been determined satisfactory, and all jobs associated with the active groups have been scheduled, all failure codes in IMODES 30 will be reset. Although the failure codes in IMODES **30** are reset, the IMU Fail inhibits will be left intact.

Had a FRESH START been requested initially or entered through RESTART, the following registers would be set to *+0;* SMODES, MODREG, AGSWORD, UPLOCK, CDU Counters, LMPCMDS, LMPIN, LMPOUT, and the PHASE 0 through **5.** The PHASE 0 through 5 would be set to *-0* and LMPOUT T set to *-0.* T4LOC would be set for a normal T4RUPT and IMODES **30** initialized. **Task** IFAILOK would be scheduled on the waitlist to occur in 5 seconds and the T5 address initialized.

From this point on the FRESH START and RESTART routines are identical. The GIMBAL LOCK lamp is interrogated and if on IMODES **30,** bit **9** is set to 0 to prevent zeroing of the CDU's.

The program interrupt inhibit is then removed and the major mode is displayed prior to entry into DUMMY JOB.

4.4 SELF-CHECK ROUTINE

There are **19** possible options in the SELF-CHECK routine. The first 18 options are used to check the internal operation of the computer (±0 to ±10) while the 19th option (±11) checks the electroilluminescent displays and associated controlling hardware on the **DSKY.** Options associated with ±10 or - zero will probably be used the most 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 are useful for diagnostic testing of the computer. The normal use of SELF-CHECK is as a routine to check the computer continuously when the computer is not busy with other routines. The ±10 or -zero options are used for this purpose.

4.4.1 SELF-CHECK OPTIONS. The SELF-CHECK option depends on what is written into the SMODE register to tell the computer what option of SELF-CHECK is desired. 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. The **SMODE** register is set to +0 during FRESHSTART, however, the content of SMODE can be controlled by the operator through use of the DSKY.

Placing a ±NON-ZERO number below octal 12 or *-0* in the SMODE register starts one of the active options of SELF-CHECK. Below is a description of what section or sections of the computer the options check. A block diagram in figure **4-4** shows the options available and indicates the number to put in the SMODE register for the desired option.

±1 octal: SOPTION **1.** Checks all pulses possible by internal control of the computer except those used exculsively by the IN-OUT instructions, In addition, SOPTION **2** will always be performed before re-entering SOPTION **1.**

±2 octal: SOPTION **2.** Checks all the IN-OUT instruction pulses.

±3 octal: SOPTION **3.** Checks central processor registers and all bit **com**binations.

±4 octal: SOPTION 4. Checks erasable memory.

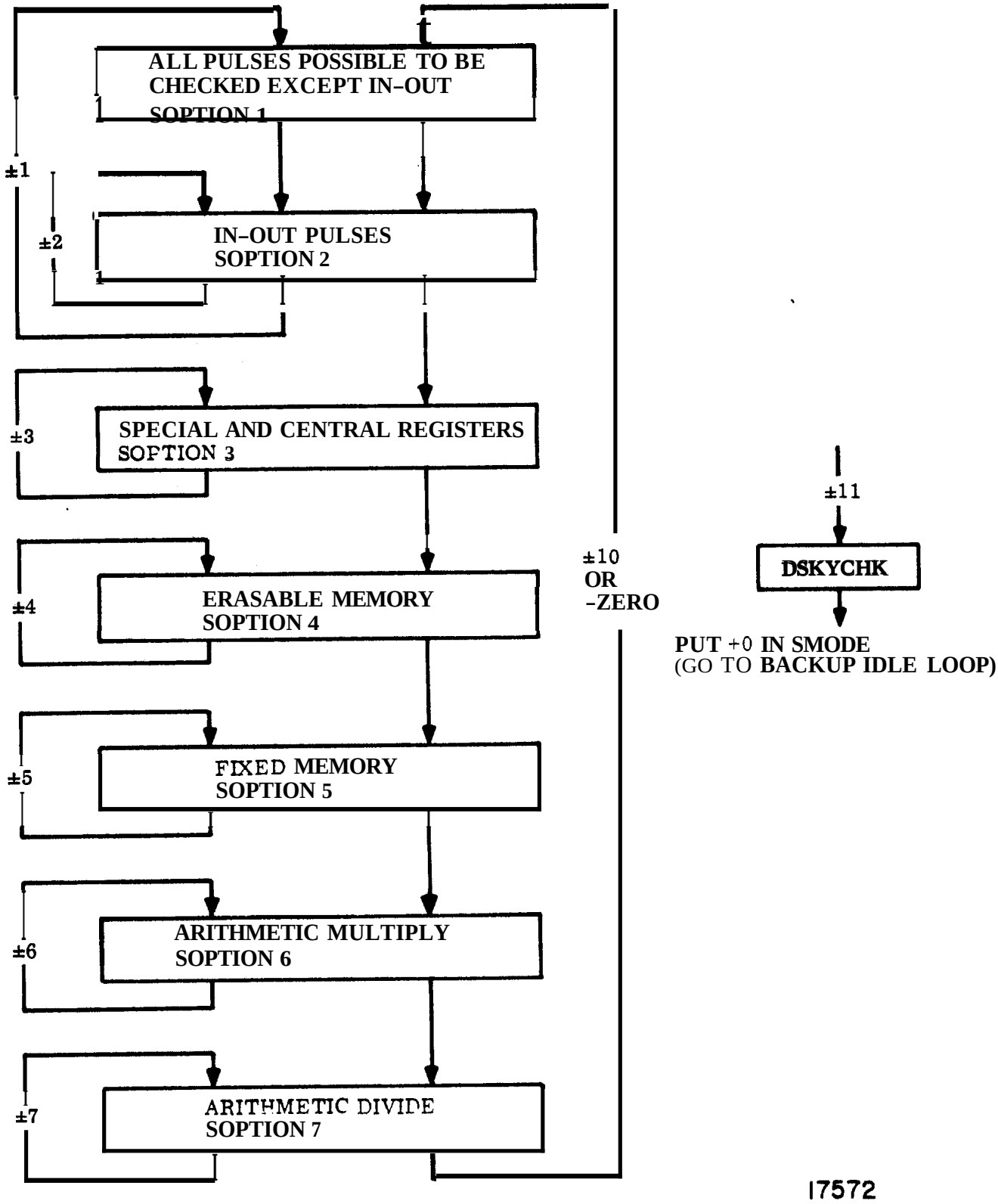


Figure 4-4. Self Check Options

±5 octal: SOPTION 5. Checks fixed memory.

±6 octal: SOPTION 6. An extensive multiply arithmetic check.

±7 octal: SOPTION 7. An extensive divide arithmetic check.

±10 octal or -0: Next SOPTION. Checks everything in the previous seven options (complete self-check of the computer).

±11 octal: DSKY CHK. Checks the electroilluminescent displays in the DSKY.

+ zero: Does not purposely check any part of the computer but forces the computer to stay in the backup idle loop, a tight loop which looks for a new job from the EXECUTIVE.

SELF-CHECK has its own verb-noun combination that is 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.

4.4.2 ERROR DETECTION, The block diagram in figure 4-5 shows the Count Registers and Self-Check Error Detection with ±10 or -0 in SMODE. If SELF-CHECK should detect an error, 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 of where the error occurred, +1)

Step 2: The ERCOUNT register is incremented by one.

Step 3: The Program Caution lamp on the DSKY is turned on.

Step 4: Octal 01102, AGC SELF CHECK ERROR, is inserted into FAILREG, C(FAILREG), C(SFAIL) and C(ERCOUNT) are displayed in R1, R2, and R3 of the DSKY, respectively.

Step 5: (a) Enter Backup Idle if C(SMODE) = +10.

(b) Start at beginning again if C(SMODE) is -10.

(c) Continue on with SELF-CHECK at the next address after the error if C(SMODE) is -ZERO.

If a second malfunction is located octal 41102 is put in the FAILREG register but steps 3 and 4 are omltted since Program Caution lamp is already on and 01102 is already in R1. Steps 3 and 4 are omitted from all successive malfunctions until the FAILREG register is made +zero (normally by performing a "FRESH START"). A FRESH START will also set SMODE (if a Verb 36 had been entered), SFAIL, and ERCOUNT to +zero while a RESTART will set SFAIL to +zero.

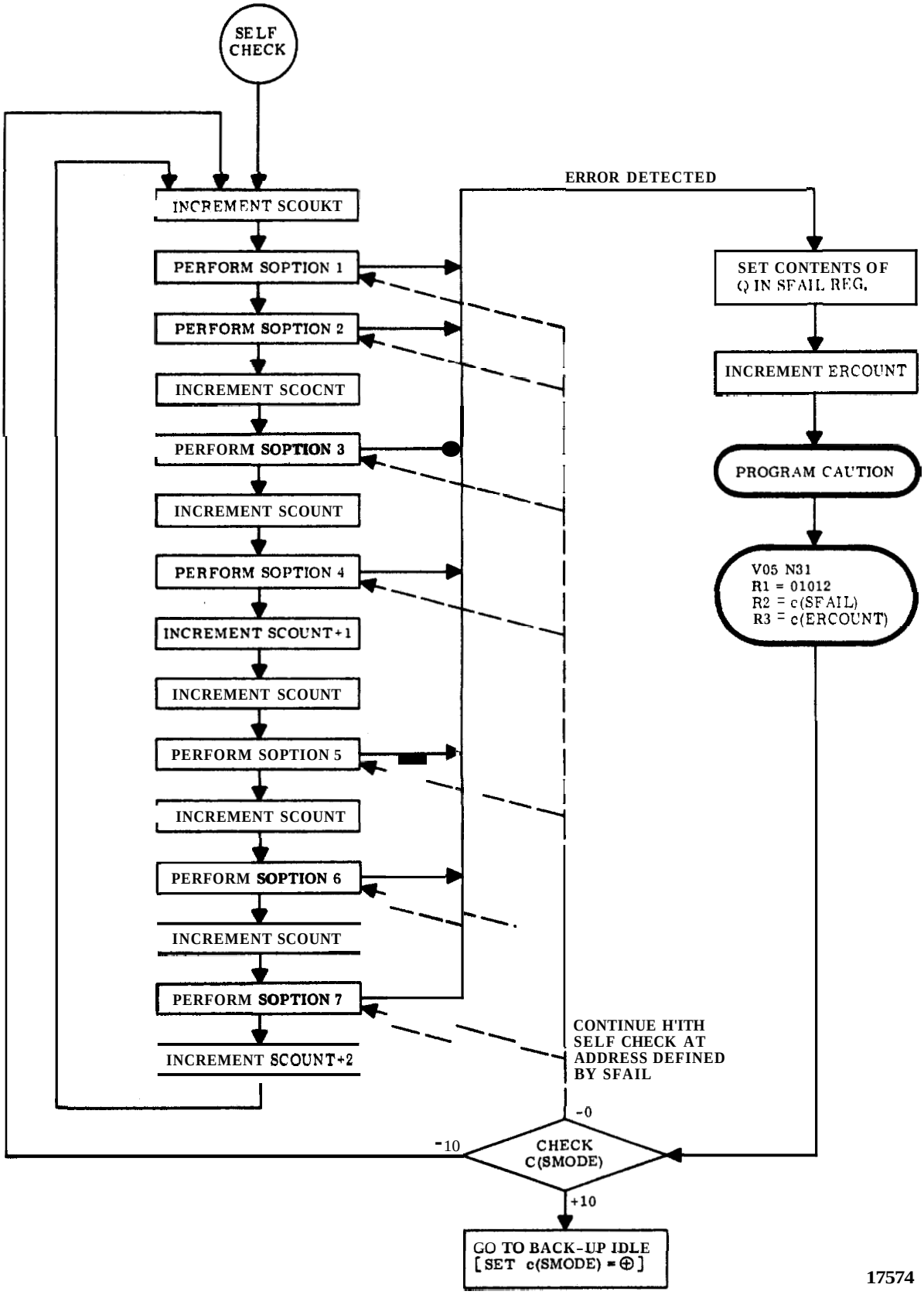


Figure 4-5. Count Registers and Self Check Error Detection with ±10 or -0 in SMODE

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.

Figure 4-6 is the flow diagram for Self Check with ±1 - ±7 in **SMODE.** The corresponding **SOPTION is** performed continuously until an error is detected at which time the error is displayed in the same manner as previously discussed.

4.4.3 DSKY CHECK. Putting a ±11 in the **SMODE** register (see figure 4-7) illuminates all possible electroilluminescent displays on the **DSKY.** The subroutine then 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.

4.4.4 HOW TO USE THE *DSKY TO* MONITOR SELF-CHECK. The block diagram in figure 4-5 shows how the three **SCOUNT** registers may be utilized to monitor the operation of **SELF-CHECK. SCOUNT (1366) is** incremented at the start **of** each of the seven minor loops that make up the internal computer self-check even if they are run through consecutively as they are when ±10 or -zero is in **SMODE.**

> **EXCEPTION:** When ±10 or -zero is in SMODE **SCOUNT** is not incremented at the beginning of **the SOPTION** concerned with **IN-OUT** pulses.

SCOUNT +1(1367) is incremented upon the completion of the erasable memory **SOPTION** when **i4,** ±10, or -0 is in **SMODE. SCOUNT** +2(1370) is incremented upon the completion of the arithmetic divide **SOPTION** when ±7, ±10, or -0 is in **SMODE.** The incrementing of the **SCOUNT** +2 indicates the successful completion of the complete 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, respectively.

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 operation:

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)

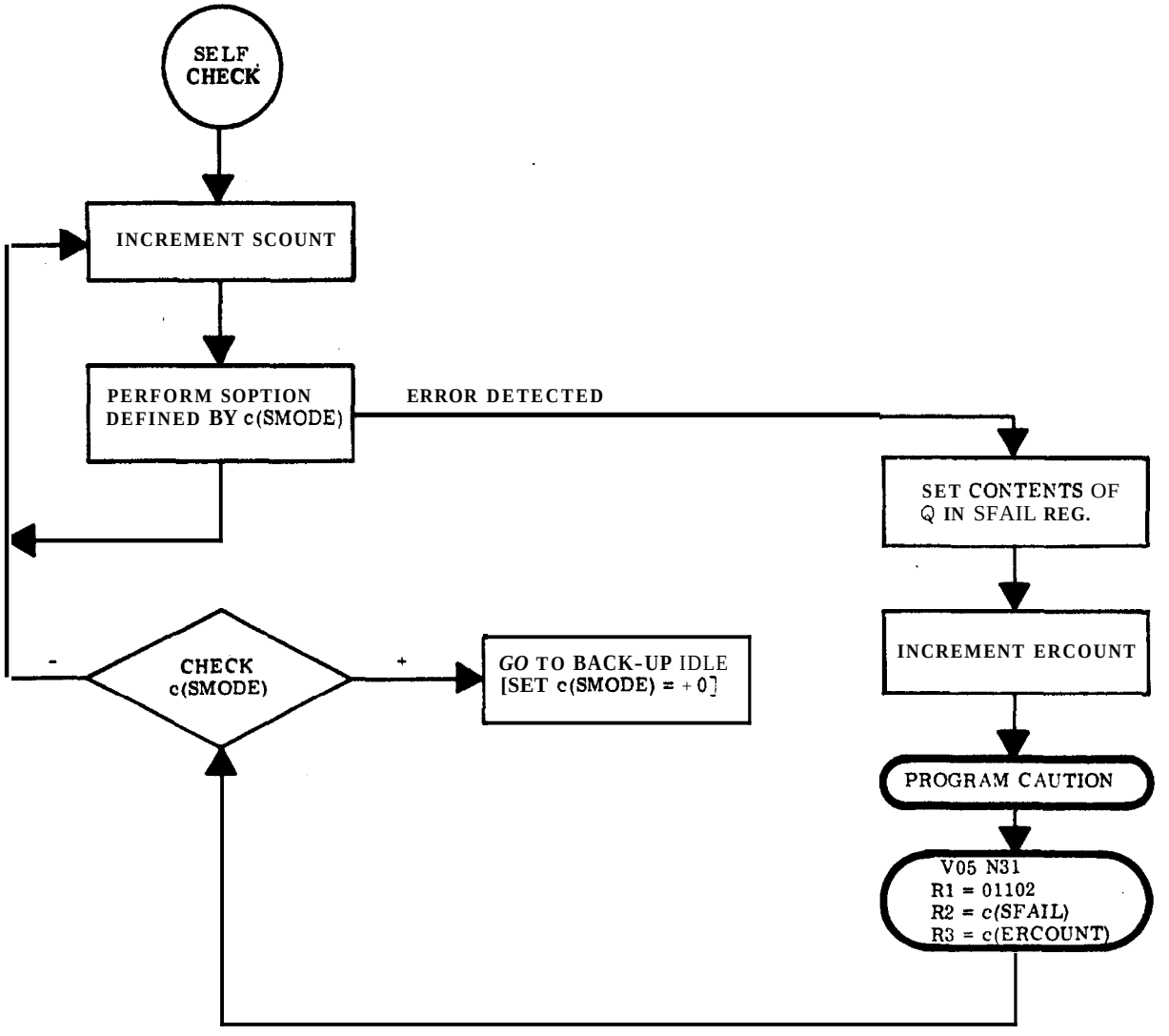
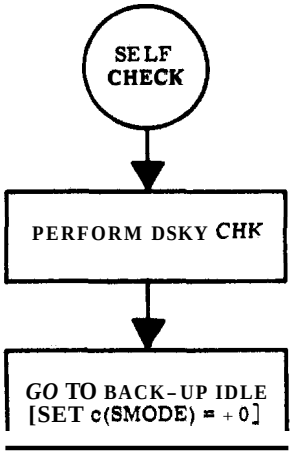


Figure 4-6. Self Check Error **Detection with ±1 - ±7 in** SMODE



17573

Figure 4-7. Self Check with i11 in SMODE

4.4.5 SELF CHECK FLOW. Self check (figure **4-8)** is entered from DUMMY JOB whenever a **job** of higher priority is not scheduled. The self check routtne is composed of eight major subroutines; SOPTION **1** through **7** and DSKY CHECK. "he operator can route the computer to the desired subroutine by loading the SMODE register with the appropriate value, through the use of the DSKY. In order to cause the computer to continuously perform **any** option the SMODE register must be loaded with the option number desired. Either a plus or minus may be used as both will cause entry into the desired option **and** as long as an error is not detected the computer will react identically to both. However, if an error is detected, the computer reacts differently to a plus than to a minus value. If the SMODE register is loaded with ±10 or -0 all seven options will be performed consecutively beginning with SOPTION **1.** Again, the computer reacts identically to all three of these values until an error is detected. If the SMODE register is loaded with ±11, the computer will perform the DSKY CHECK. The operator has control over the contents of SMODE, however, the computer will set SMODE to +0 under certain conditions.

Upon entry into self check, the starting address of SOPTION **1** is saved to be used in case the contents of SMODE = ±10 or -0.

After checking again to make sure that another **job** with a higher priority is not waiting, the SMODE register **is** checked to determine which subroutine to enter. **If** the SMODE register contains *+0,* a tight loop exists in which the computer does not perform a meaningful function, but merely continuously checks for a higher priority job or a change in SMODE. **This** loop is designated the "back-up idle loop".

If SMODE contains *-0* or ±1 through ±10 the SCOUNT register is incremented. This register is incremented each time to provide a total of the number of times the computer passes through this point. **The** SCOUNT register can be loaded through use of the DSKY.

If SMODE contains a quantity greater than +11 or less than -11, SMODE is set to +0 which forces the computer into the back-up idle loop. A number larger than ±11 in SMODE therefore **is** meaningless to the computer,

If SMODE contains ±11, SMODE **is** set to +0 and DSKY CHECK is entered. The next time SMODE is checked (upon completion of DSKY CHECK) the back-up idle condition will exist.

The seven options perform the following:

SOPTION **1**	Checks all instruction pulses generated by the computer except those used exclusively by the IN-OUT instructions.
SOPTION **2**	Checks the **IN-OUT** instruction pulses.
SOPTION **3**	Checks the counting ability and the ability to handle overflow and underflow situations.
SOPTION **4**	Checks erasable memory.
SOPTION **5**	Checks fixed memory.
SOPTION **6**	Checks multiply arithmetic ability.
SOPTION **7**	Checks divide arithmetic ability.

The option specified by **SMODE** will be entered and upon completion (providing an error is not detected) will return to check for a scheduled job of higher priority. If none is found, **SMODE** will again be checked and assuming it has not been changed, the option will be repeated, The specified option will, therefore, be performed continuously whenever the computer is in **SELF CHECK.** If **SMODE** contains ±1 **SOPTION 1** and **SOPTION 2** will be performed. If **SMODE** contains ±10 or −0, SOPTION 1 is entered and all options are performed consecutively. Again upon successful completion of the entire loop, it is repeated as long as the computer is in **SELF CHECK.**

Whenever an error is found, the subroutine **ERRORS** is entered. The address of the instruction immediately following the point where the error was detected is saved and stored in the SFAIL register and the **ERCOUNT** register is incremented.

The **PROGRAM CAUTION** lamp is lit if not already on and **V05 N31** is displayed. R1 displays the contents of **FAILREG,** which is **01102 (AGC** self check error); **R2** displays the contents of **SFAIL;** and **R3** displays the contents of ERCOUNT. If an error had been detected previously and the alarm displayed, the second error will set bit **15** of **FAILREG** to **1** to indicate the existance of multiple errors.

The contents of **SMODE** is examined to route the computer to the desired point. If the contents of **SMODE** is less than −0, the option specified is entered again, at the beginning, If the contents of SMODE is **−0,** self check is continued at the next instruction after the point where the error was detected. If the contents of SMODE is +0 ***or*** greater, SMODE is set to +0 and the computer is forced into the back-up idle loop. Each of the eight major subroutines is discussed below.

SOPTION 1 - This option checks the pulses which are generated to implement all instructions, except those used exclusively with **IN-OUT** instructions. The method used to check the control pulses is to perform the instruction, check the result, and transfer control to the **ERRORS** subroutine whenever the result is not as expected, The branching type instructions are provided with direct access to the **ERRORS** subroutine on a wrong branch, whereas the arithmetic functions generally make extensive use of *−0* **CHECK,** +0 **CHECK, −1 CHECK** and **+1 CHECK** subroutines. Values are chosen such that when manipulated by the instruction(s) being checked a result equal to one of the above values is obtained, A failure will route control to the **ERRORS** subroutine.

Upon completion of **SOPTION 1, SOPTION** 2 is always performed before returning to check SMODE.

SOPTION 2 - This option checks all control pulses used in the **IN-OUT** instructions. The method used is identical to that used in SOPTION **1.**

SOPTION 3 - The **first** part of this option is used to check the ability of the computer to count up a **14** bit number. This is accomplished by initializing **SKEEP *6*** to **37777** and **SKEEP 7** to **40000** and decrementing **SKEEP *6*** and incrementing **SKEEP 7** until **SKEEP *6*** contains +0 and **SKEEP 7** contains ***−0.***

Immediately after initializing SKEEP 6 and SKEEP 7 the contents of SKEEP 6 is checked for +0. Assuming SKEEP 6 is not +0 its contents minus 1 is added to the complement of its contents. The result, provided all control pulses are being generated properly, is -1. An incorrect value will cause a transfer to the ERRORS subroutine while a correct value will cause a continuation of this option. A check is made at this point to see if a new job of higher priority has been scheduled since the last time we looked. After checking for a new job, the complement of SKEEP 6 is examined. If this value is less than -0, the contents of SKEEP 6 is decremented; if the value is -0, SKEEP 6 is not decremented. In either case, the contents of SKEEP 6 and SKEEP 7 are added which should result in -1. A check is made for this quantity as before, after which SKEEP 7 is incremented and the loop is repeated, This action continues until SKEEP 6 contains +0. At this point SKEEP 7 should contain -0. Addition of these two quantities then will produce a -0 and the count check portion of SOPTION 3 is complete.

The second portion of SOPTION 3 checks the ability of the computer to handle overflow and underflow situations when they occur.

Initially, SKEEP 5 is set to 40000 and a maximum overflow quantity is set up. This quantity is checked, determined to be greater than +0, decremented, and stored in SKEEP 4. Next, a check is made to see if the quantity is still overflow. The low 14 bits of the overflow quantity is then added to the contents of SKEEP 5 which should result in a -1. If this is the first time through this point, a maximum underflow quantity is set up and the second half of the OVFLOW loop is entered. This time the quantity is determined to be less than -0 after which it *is* complemented and decremented. The quantity should still be overflow, in which case the low 14 bits is again added to the contents of SKEEP 5 to produce a -1. The computer now looks for a job of higher priority and upon finding none the contents of SKEEP 5 is diminished, the last overflow quantity checked is decremented, and the loop is repeated. This action continues until an overflow quantity after being decremented results in a quantity which is no longer overflow, At this time the contents of SKEEP 5 should be -0 and the contents of SKEEP 4 should be 37777. These two values are checked and control is either transferred to ERRORS or to the lead in section of SELF CHECK to check for a new job and to interrogate SMODE. This option is now complete and the next self check subroutine depends on the contents of SMODE.

SOPTION 4 - This option provides an extensive check of the erasable memory. Most locations are checked to make sure that the computer can write into the location and read out what was written. SOPTION 4 is begin by selecting erasable bank 00 and starting address 01462 as the first location to be checked, The address to be checked is stored in SKEEP 7. The check of the addrdss is started by saving the present contents of the address contained in SKEEP 7 as well as the next successive address. The contents of these two addresses are then set to their own address. (e. g. c(01462) - 01462 and c(01463) - 01463.) The contents of the first address is then added to the complement of the contents of the second which should provide a -1 result. Next, the content of the two addresses is set to the complement of their own address. (e. g. c(01462) = 76315 and c(01463) = 7.6314.) The contents of the second address is now added to the complement of the first which again will result in a -1. If either of the -1 checks is unsatisfactory, control will be transferred to the ERRORS subroutine, After successful completion of these checks, the original contents of the two addresses is restored, a check for new job is made and assuming none is found, the next address is determined. Assuming we have the EBANK set to select bank *00,* a few decisions must now be made. If address 01777 has not been checked, SKEEP 7 is incremented and the ERAS LOOP is performed again.

NOTE: The first time through the ERAS LOOP, addresses **01462** and **01463** were acted upon. This time, since the contents of SKEEP **7** is **01463,** addresses **01463** and **01464** will be acted upon. (The second address of the first pair is the first address of the second pair.)

This loop is repeated until address **01777** has been checked at which time, since address **00062** has not been checked, SKEEP **7** is set to **00062** and the loop continues to be repeated until address **01374** has been checked.

NOTE: Even though EBANK is set to bank *00,* addresses **00062** through **00377** will be picked from bank *00;* addresses **00400** through **00777** will be picked from bank **01;** and addresses **01000** through **01364** will be picked from bank **02.** Addresses **00062** through **01374** will always be picked from these three banks regardless **of** the content of EBANK.

After address **01374** has been checked, checking will continue at the desired address of the next bank. The method of checking the erasable memory with other values in EBANK **is** identical except that the block of addresses checked may be different. Table **4-3** shows the addresses checked and their bank designations with different contents in EBANK.

After checking erasable memory with EBANK = **7,** control is transferred to the CNTR CHECK portion **of** SOPTION **4.** The CNTR CHECK portion of SOPTION **4** performs a CS instruction **on** all addresses from **00061** through **00010,** and manipulates data in all shift and cycle registers. The data is chosen such that when shifted or cycled and added to a constant, the result is **-1** on two occasions. The normal check for **-1** is performed and the normal transfer of control to the ERRORS subroutine will follow if it fails. A satisfactory check will terminate this option, however, before returning to the lead in section of SELF CHECK to check for a new job and look at SMODE the SCOUNT register will be incremented. This register will contain the number of times we have passed through SOPTION **4.**

SOPTION 5 - This option performs a check of fixed memory by adding every word in a bank together. The last word in each bank is always chosen such that the sum of all words in the bank will equal either the bank number or the complement of the bank number.

The option begins by selecting fixed bank *0,* obtaining the first word in the bank and starting it in **SKEEP 1.** Since this **is** the first word in the bank, the next decision will be based on whether we are in ROPECHK (SOPTION **5)** or SHOWSUM. (SHOWSUM is a separate job which is not a part of the SELF CHECK routine. SHOWSUM, however, does use the the same block of instructions and therefore will be discussed here.) Assuming we are in ROPECHK a check is made to make sure a higher priority job is not waiting before the next word in the bank is obtained. This word is then added to the contents of SKEEP **1** and the sum is stored in SKEEP **1.** This loop continues to be repeated until the last word in the bank has been obtained. The sum of the bank is then present in SKEEP **1.** Again assuming we are in **ROPECHK,** control is transferred to the subroutine BNKCHK.

Table **4-3.** Erasable Addresses Checked in SOPTION 4

C (EBANK)	Address Checked	Bank Designation
00	01462 - 01777	00
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	02
01	01400 - 01777	01
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	02
02	01400 - 01777	02
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	02
03	01400 - 01774	03
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	02
04	01400 - 01777	04
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	02
05	01400 - 01777	**05**
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	02
06	01400 - 01777	06
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	02
07	01400 - 01777	07
	00062 - 00377	00
	00400 - 00777	01
	01000 - 01374	**02**

BNCHK sets the contents of SKEEP 1 equal to the absolute value of the old contents of SKEEP 1 decremented by 1, and puts the bank number in the 5 low order bits. The bank number is then complemented and added to the contents of SKEEP 1. The result should be -1. The bank number is then incremented and providing the new bank number is less than 22, SKEEP 1 is initialized to zero and the same process is repeated for the next bank. Each bank is checked until bank number 22 is specified, at which time (again assume ROPECHK) control is transferred back to the SELF CHECK lead in to look at SMODE.

The job SHOWSUM is scheduled via the executive NOVAC with a priority of 02 whenever the operator keys in V56 ENTR. (Perform BANKSUM.) This also sets the content of SMODE to +0 which puts the computer in the back-up idle loop whenever SELF CHECK is entered. The job SHOWSUM begins by reserving the DSKY for use in 'the job. Bank 0 is then selected and all words in the bank are added together and the sum stored in SKEEP 1 as in ROPECHK. The difference occurs after the last word has been obtained at which time the subroutine SDISPLAY is performed. SDISPLAY puts the bank number in the 5 low order bits and sets the DSPTABS to flash V05 N01, (display octal comp 1, 2, 3, - specify address). The DSPTABS will then be set to display the bank sum, the bank number, and the bugger word in R1, R2, and R3 respectively. (The bugger word, also known as CKSM, is the last word in each bank. The value of this word is chosen such that when added to all other words in the bank the resulting sum will be equal to, or the complement of, the bank number.) SHOWSUM then goes to sleep to await operator action. If the operator desires to have the next bank displayed he will key in V33 ENTER (proceed without data). This will wake the sleeping job, increment the bank number and, providing the new bank number is less than 22, will obtain and display the information about the next bank. If the bank number is 22, it will be set to zero and the information for bank 0 will be displayed, The cycle will continue as long as the operator continues to key in V33 ENTER. Job SHOWSUM is terminated by keying in V34 ENTER (terminate). Upon termination of SHOWSUM the computer will not return to the self check option which it was performing when interrupted by V56 ENTER, but will continue in the back-up idle loop. In order to once again enter a meaningful SELF CHECK routine, SMODE must be loaded to the desired option,

SOPTION 6 - This option performs an extensive check of the multiply arithmetic. First, a loop is set up which multiplies 37777 x (37777 through 00001). Each product is checked for accuracy by adding the upper product + lower product + 40000. This will result in a -0 which is then checked. Next, a loop is set up which multiplies -1 x (37777 through 00001). Each product is checked in two ways: 1) the upper product should be equal to -0; 2) the lower product plus the multiplier should also be equal to -0. The next two loops are identical to the first two except the multiplier and multiplicant are interchanged.

In each loop a check is made between every multiply operation to make sure that a job of higher priority is not waiting. At the end of the last loop, control is transferred back to the SELF CHECK lead into check SMODE.

SOPTION 7 - The divide capability is checked during this option. Several divide operations are made and after each operation the quotient or remainder or both are checked. After passing through the divide loop, the contents of SKEEP 4 are incremented and if it is still not equal to 0 the loop is repeated. (SKEEP 4 is initialized to 73777.) When SKEEP 4 becomes 0, register SCOUNT+2 is incremented, after which control is transferred back to SELF CHECK lead in to look at SMODE. SCOUNT+2 will contain the number of times the divide check option has been performed.

DSKYCHK - If the SMODE register contains ±11 the DSKYCHK subroutine will be entered after setting the contents of SMODE to +0. DSKYCHK merely sets the contents of SKEEP 3 to 00012 and schedules the task NXTNMBR on waitlist to be executed in 10 msec. After the delay, task NXTNMBR checks the contents of SKEEP 3 (which was initialized to 00012) and finding a value greater that +0 decrements SKEEP 3. DSPTRBS 0 through 10D are then set to display the contents of SKEEP 3 (00011) in decimal (9) in each DSKY digit location. This task will then reschedule itself (NXTNMBR) in the waitlist to be executed in 5.12 seconds. 5.12 seconds later NXTNMBR again checks SKEEP 3, decrements it, sets DSPTABS 0 through 10D to display a decimal eight in each DSKY digit location, and reschedules NXTNMBR on waitlist. The process will continue until the contents of SKEEP 3 = +0. SKEEP 3 will then be set to 01, the VN flash and COMPUTER ACTIVITY lamps will be turned on, and the DSPTABS will be set to display a minus sign in R1, R2, and R3. All digits will remain zeros from the previous DSPTAB contents. The next pass will set SKEEP 3 = -0, SKEEP 2 = +1, and display a plus sign in R1, R2, and R3. All digits will remain zeros, The next pass will blank each DSKY digit location and the final pass through will turn off the VN flash and COMPUTER ACTIVITY lamps. After DSKY CHK, the computer will remain in the back-up idle loop until SMODE is loaded with a value other than +0.

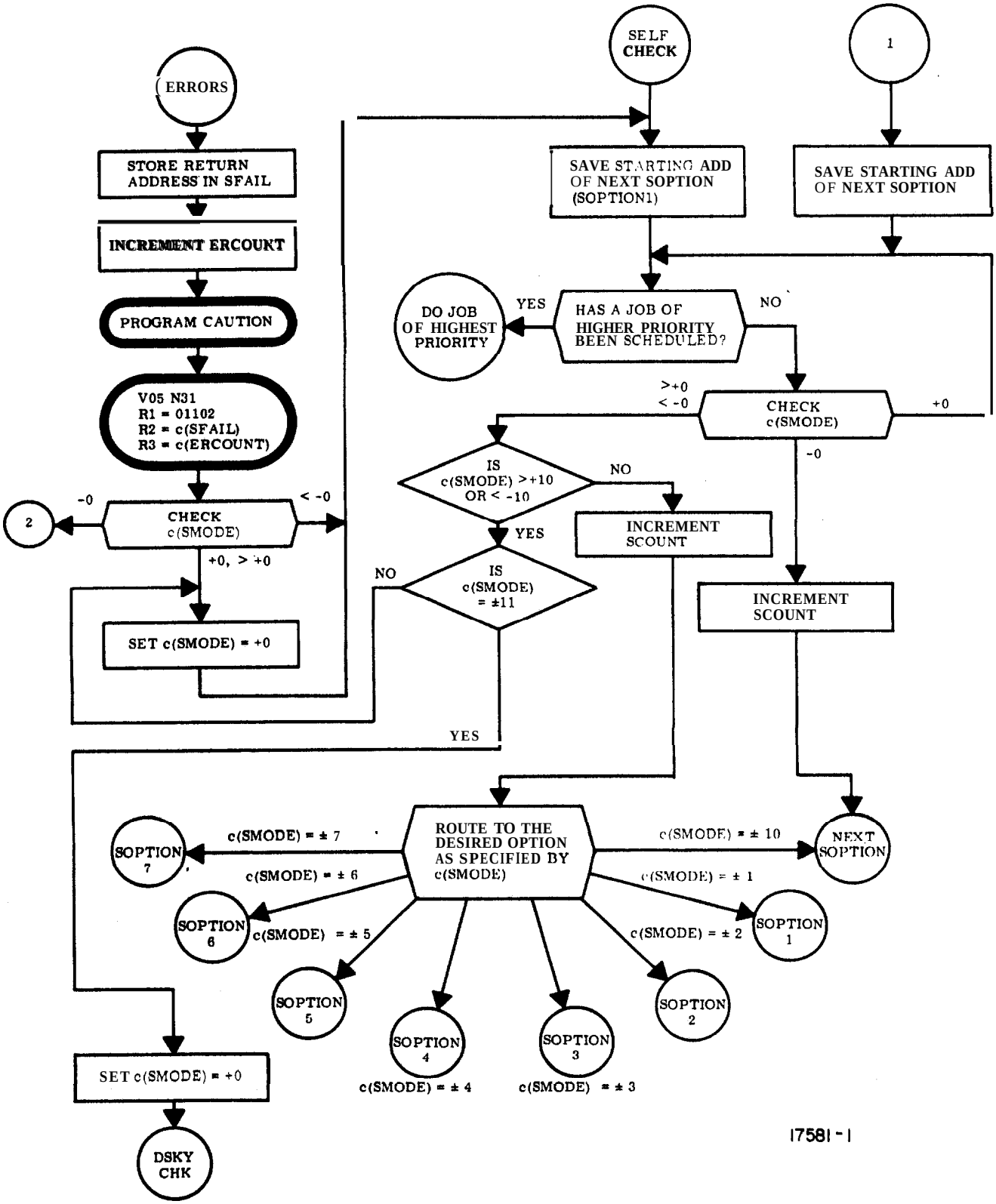


Figure 4-8. Self Check (Sheet **1** of **19)**

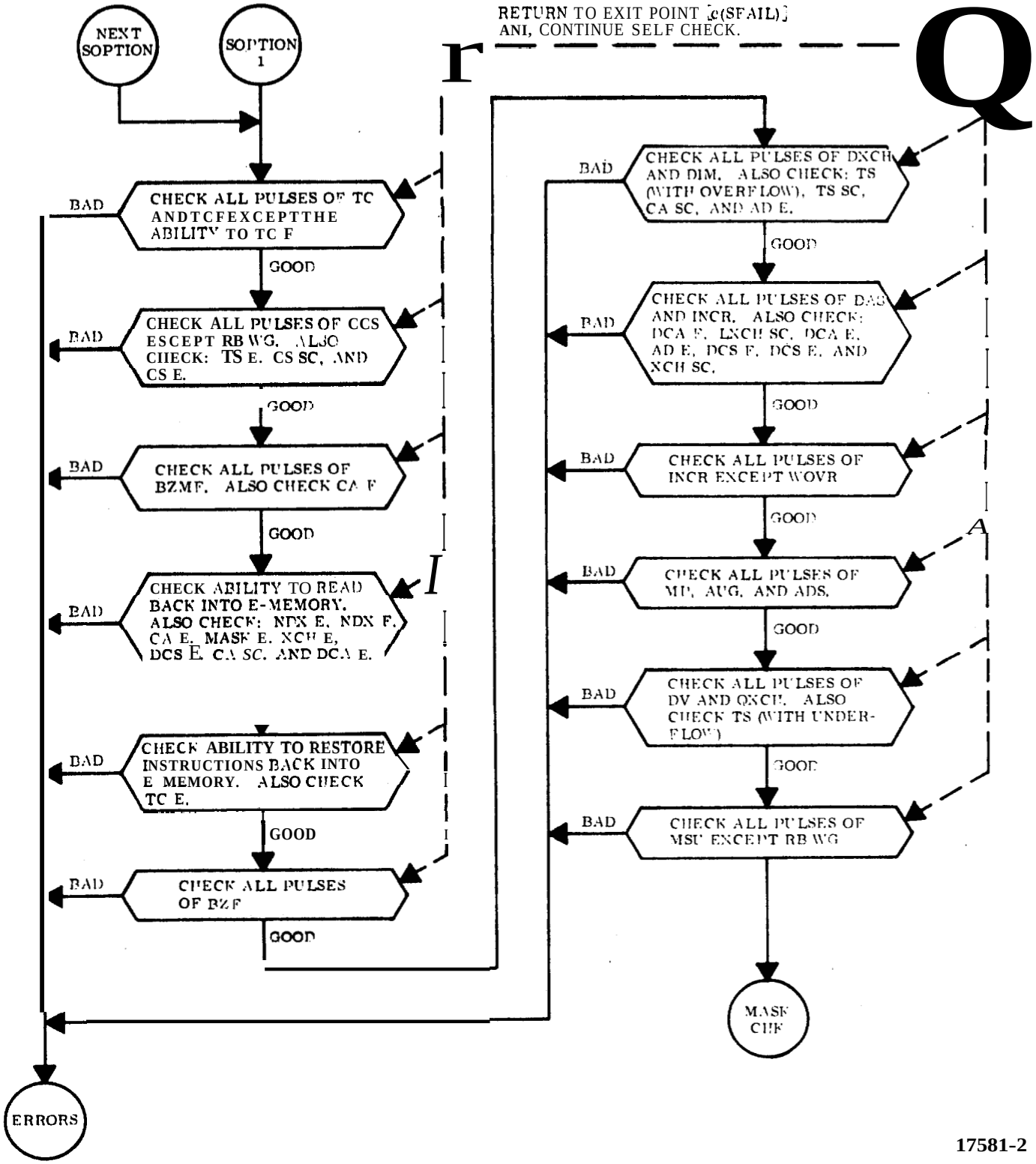


Figure 4–8. Self Check (Sheet 2 of 19)

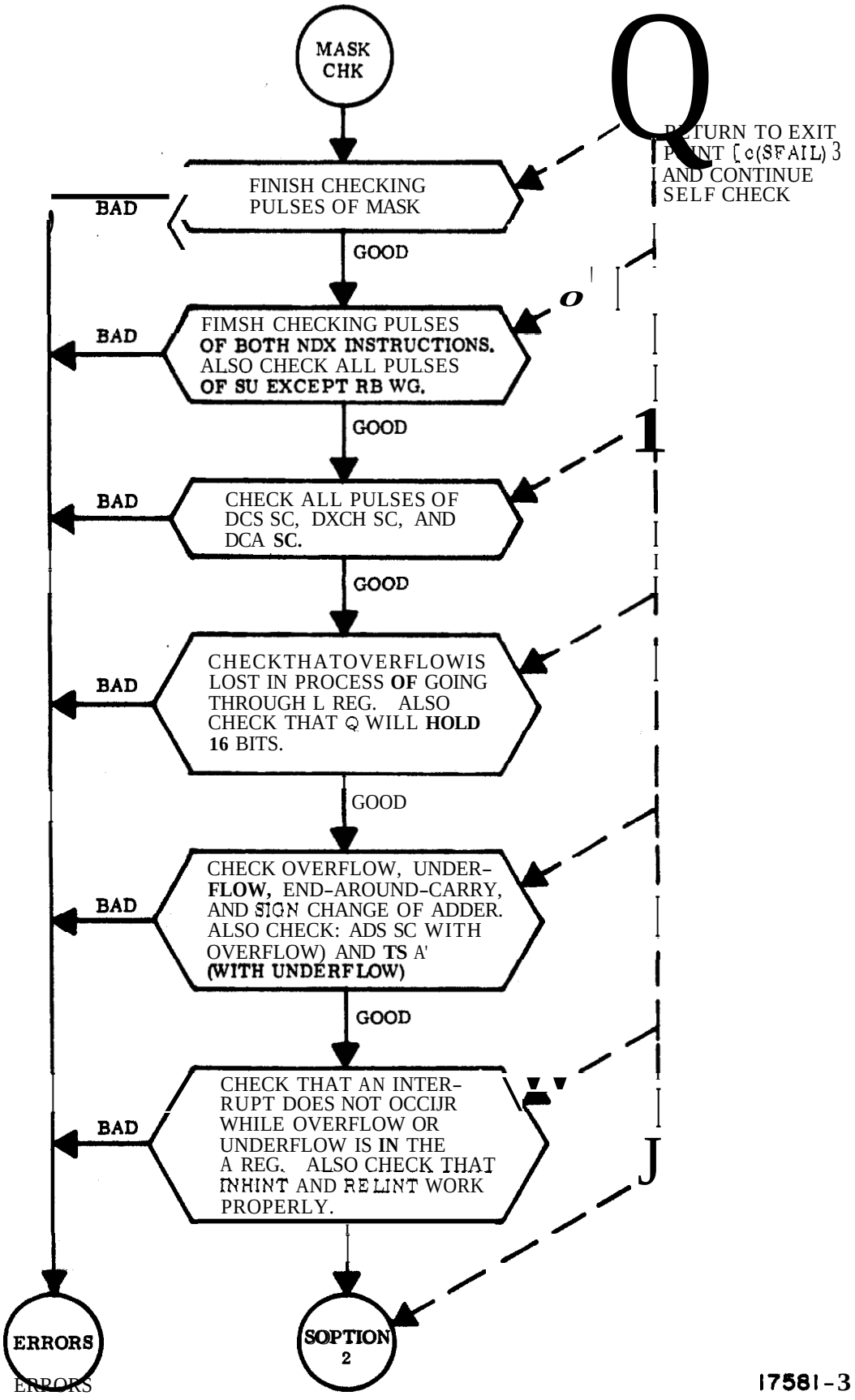
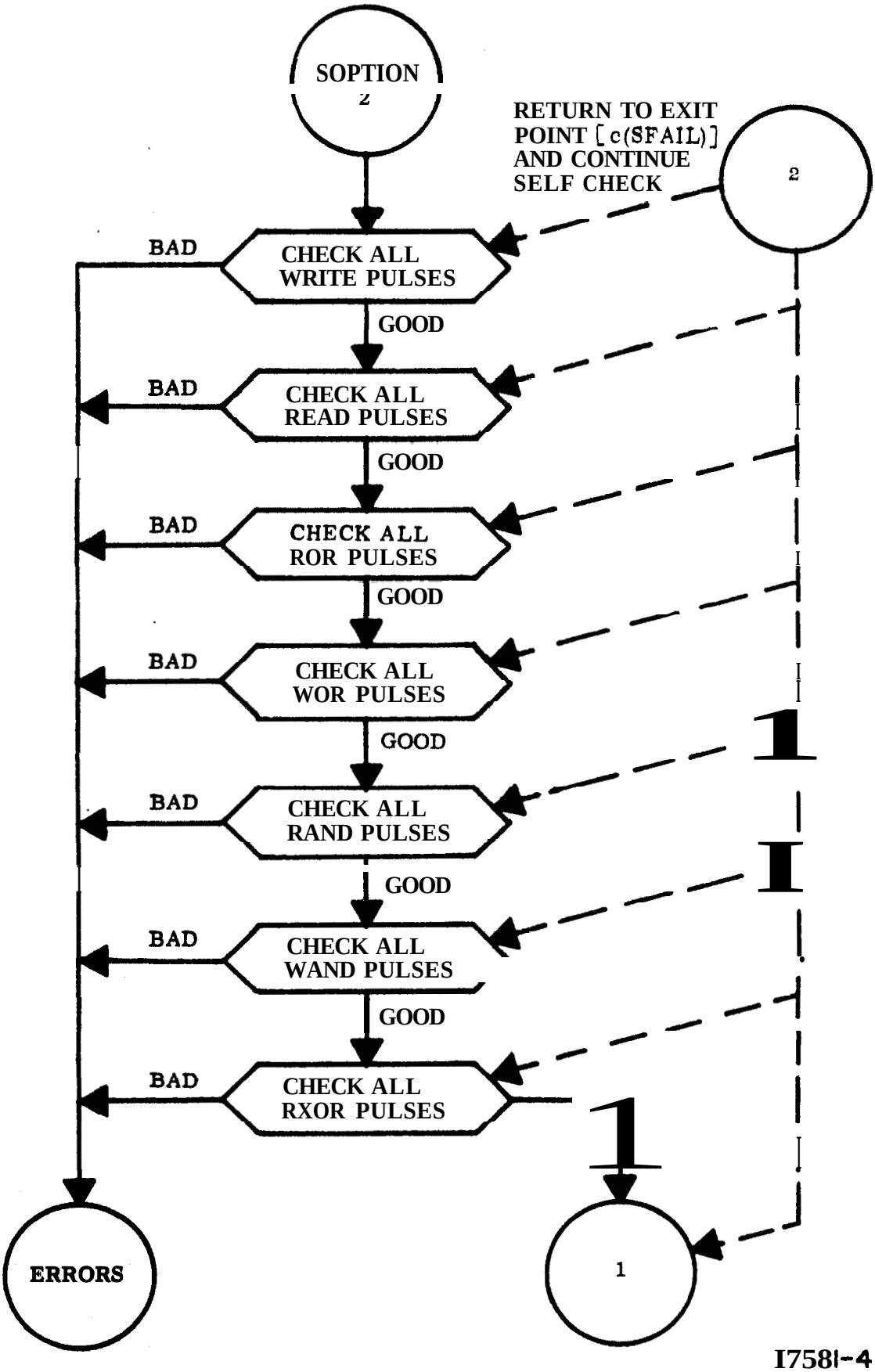


Figure 4-8. Self Check (Sheet 3 of 19)



'Figure 4-8. Self Check (Sheet 4 of 19)

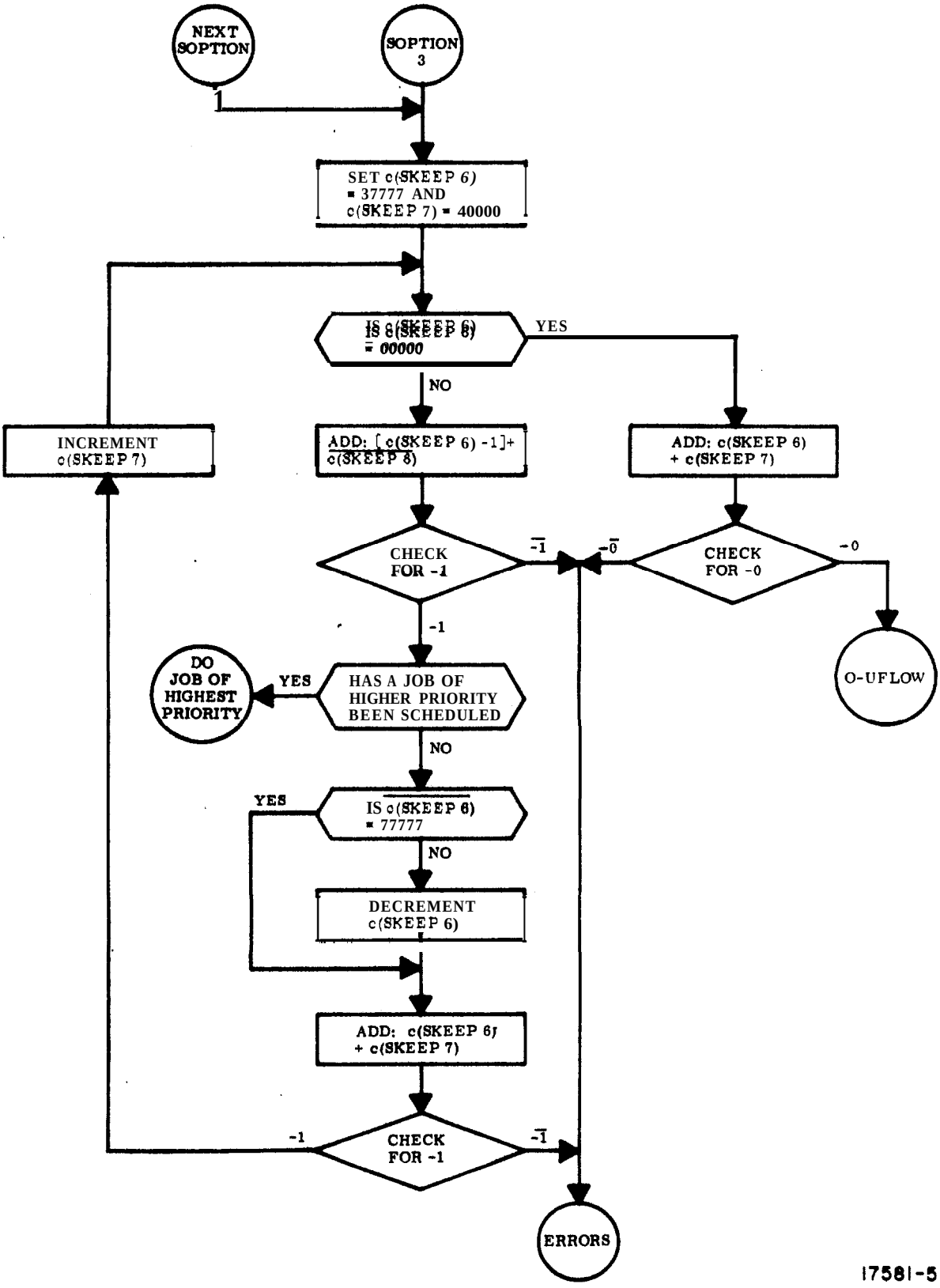


Figure 4–8. Self Check (Sheet 5 of 19)

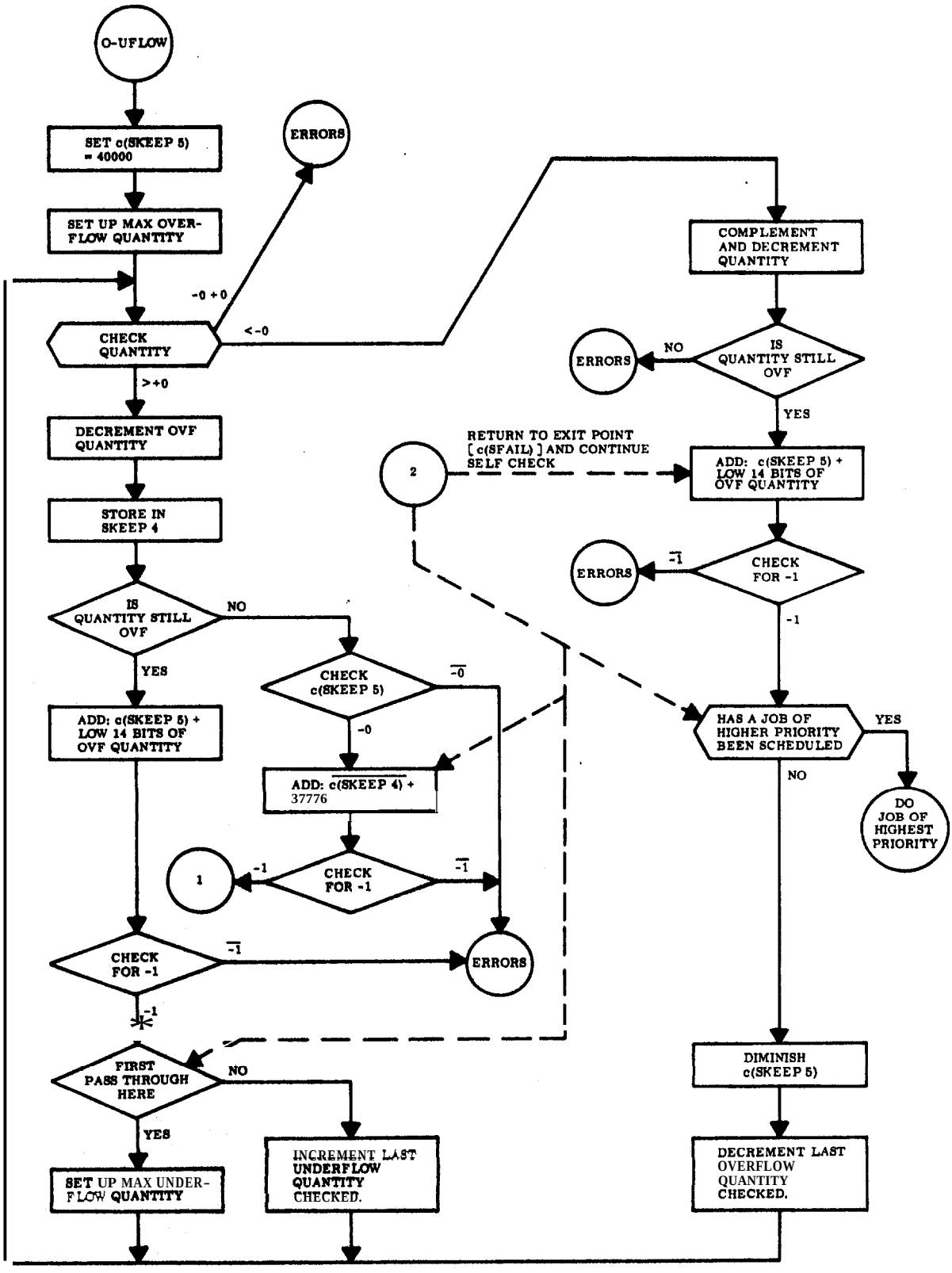


Figure 4-8. Self Check (Sheet 6 of 19)

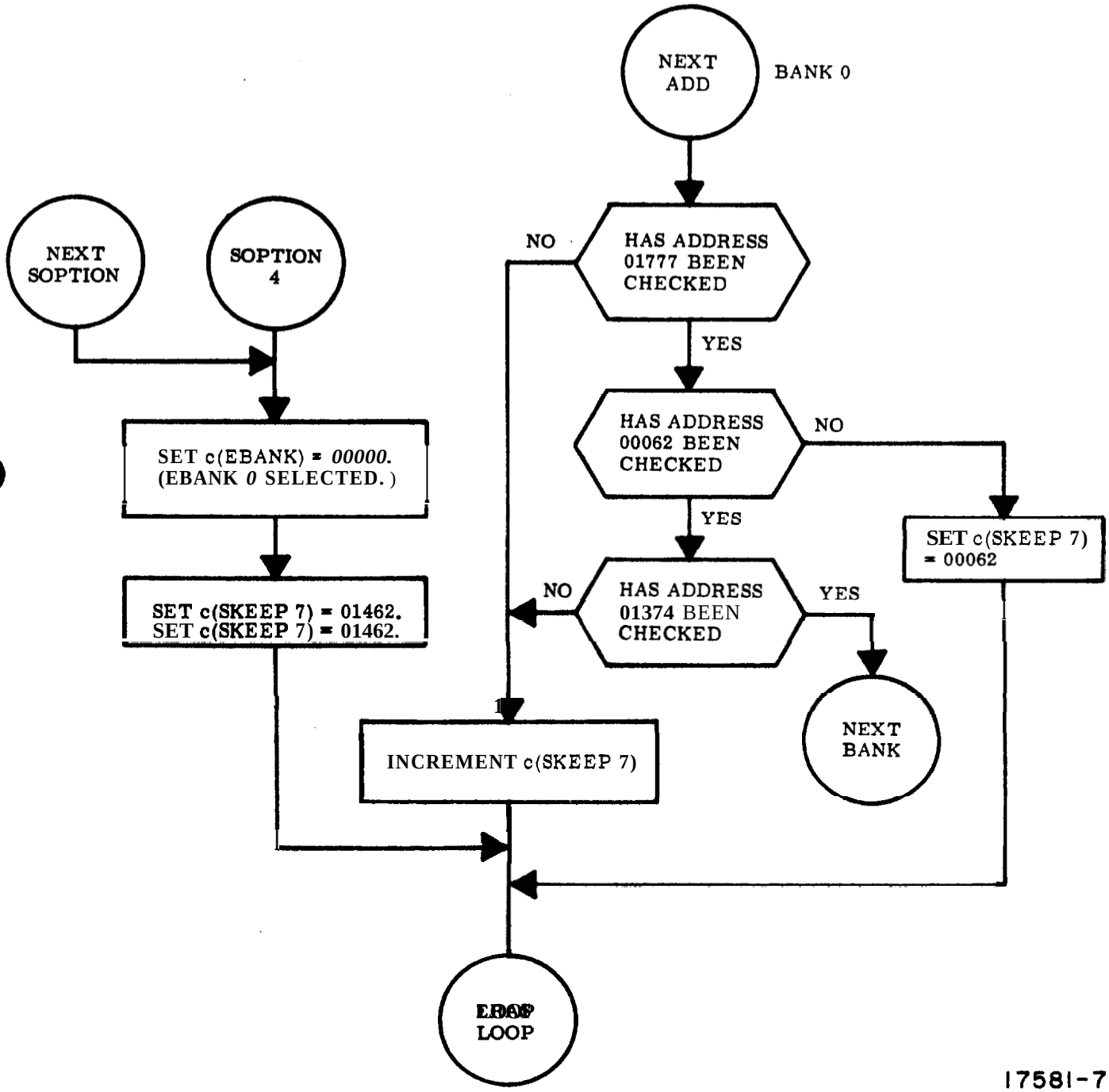
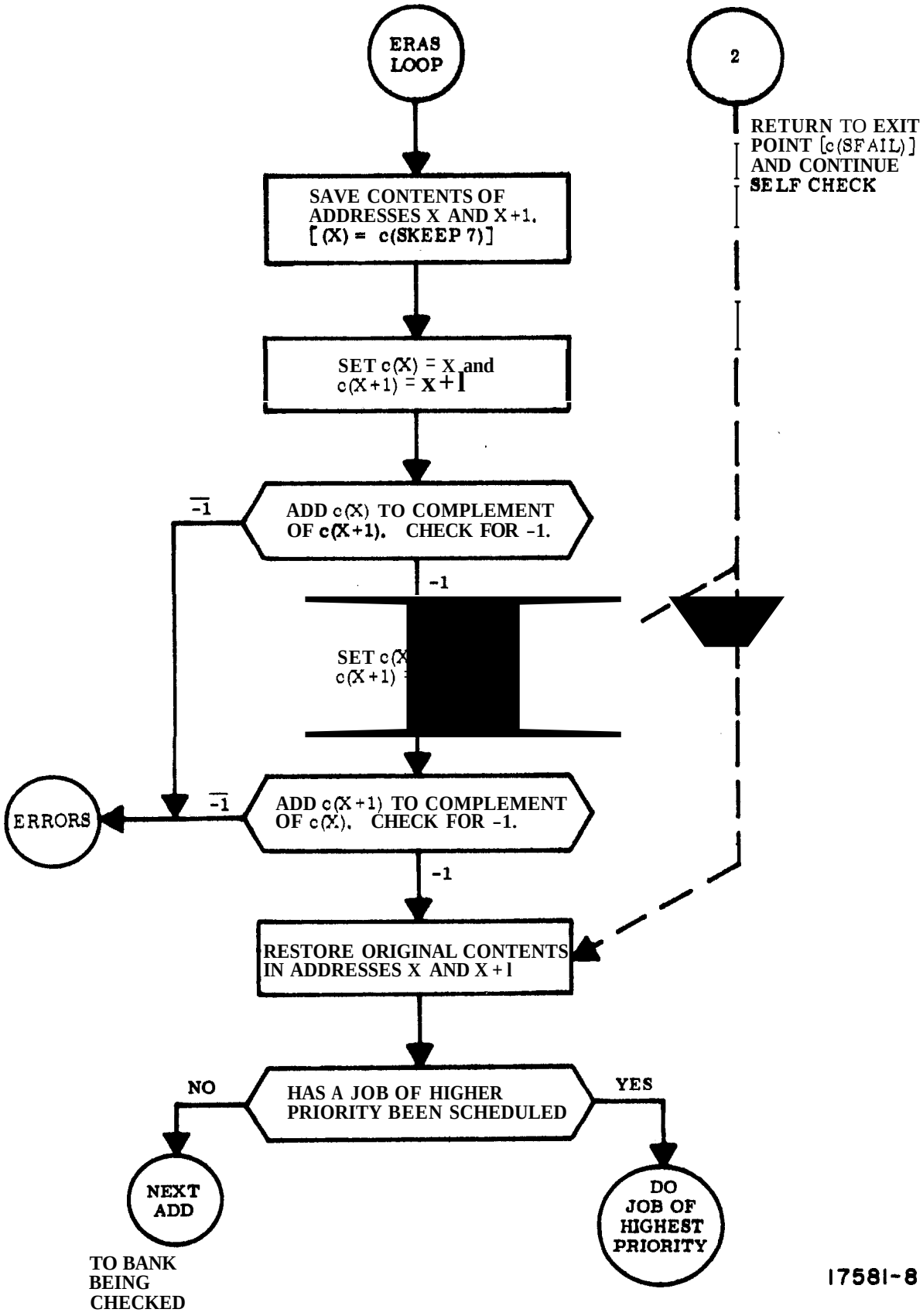


Figure 4-8. Self Check (Sheet 7 of 19)



,Figure 4-8. Self Check (Sheet 8 of 19)

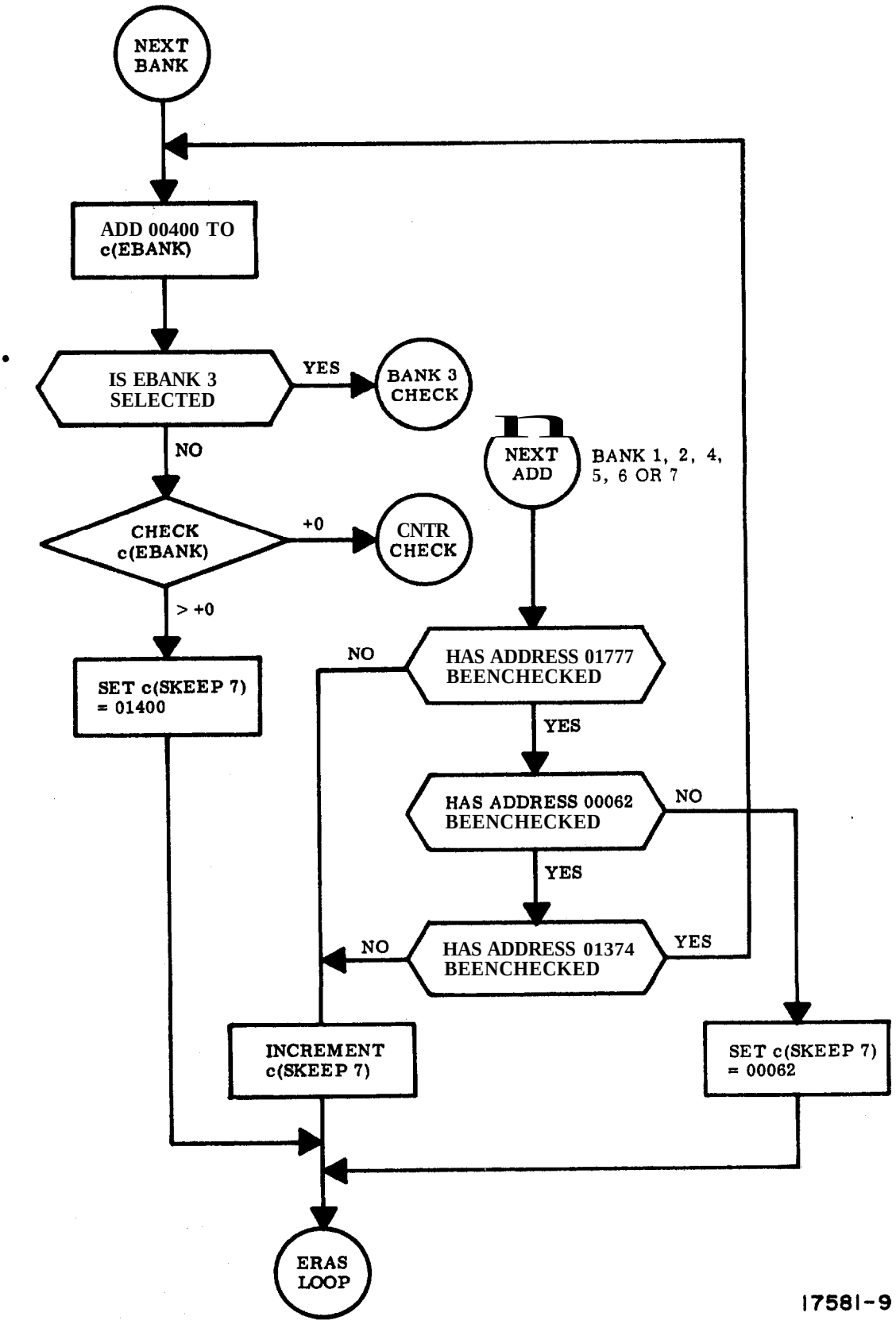


Figure 4-8. Self Check (Sheet **9 of 19**)

MISSING PAGE

4-37

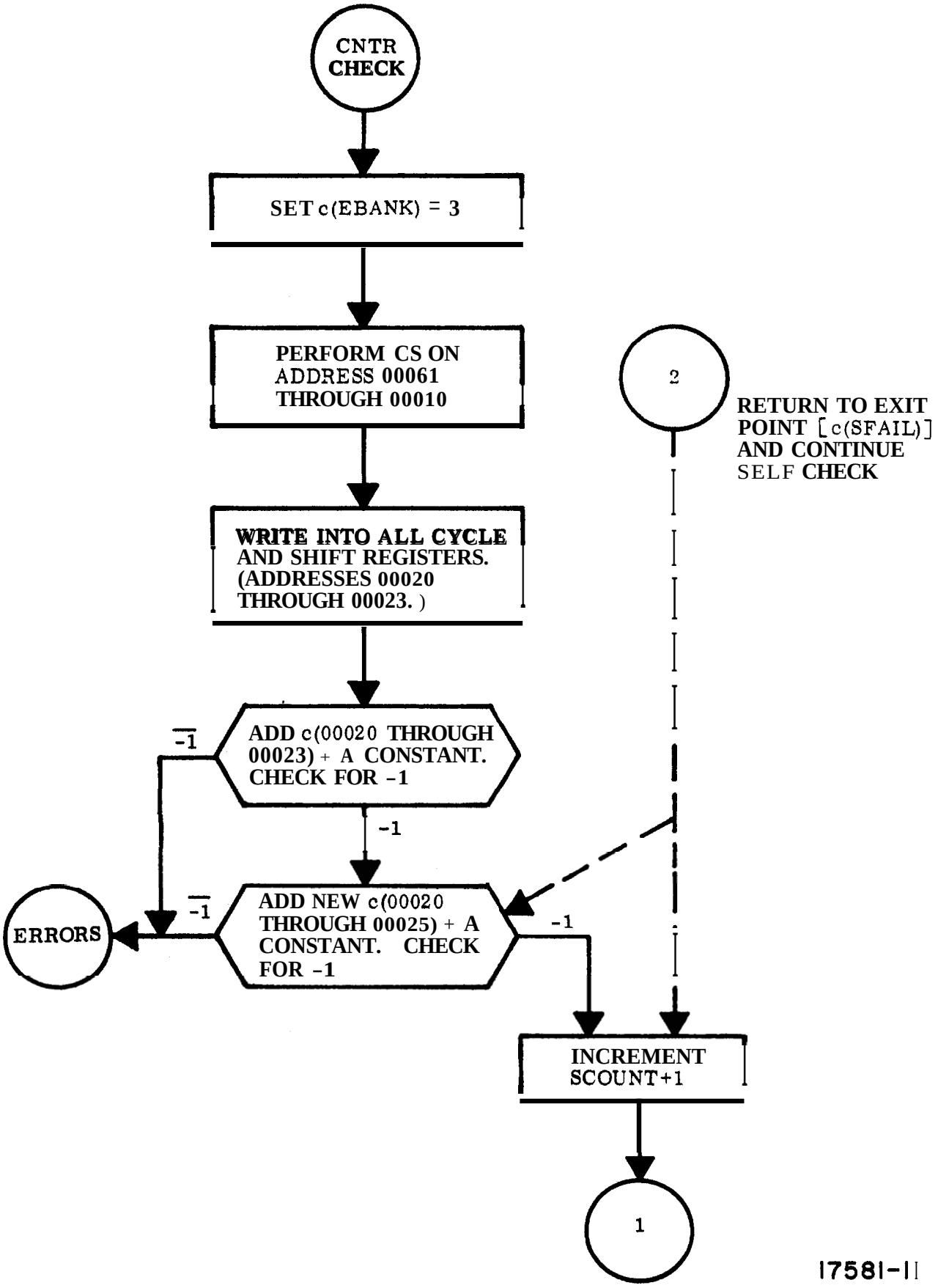


Figure 4-8. Self Check (Sheet 11 of 19)

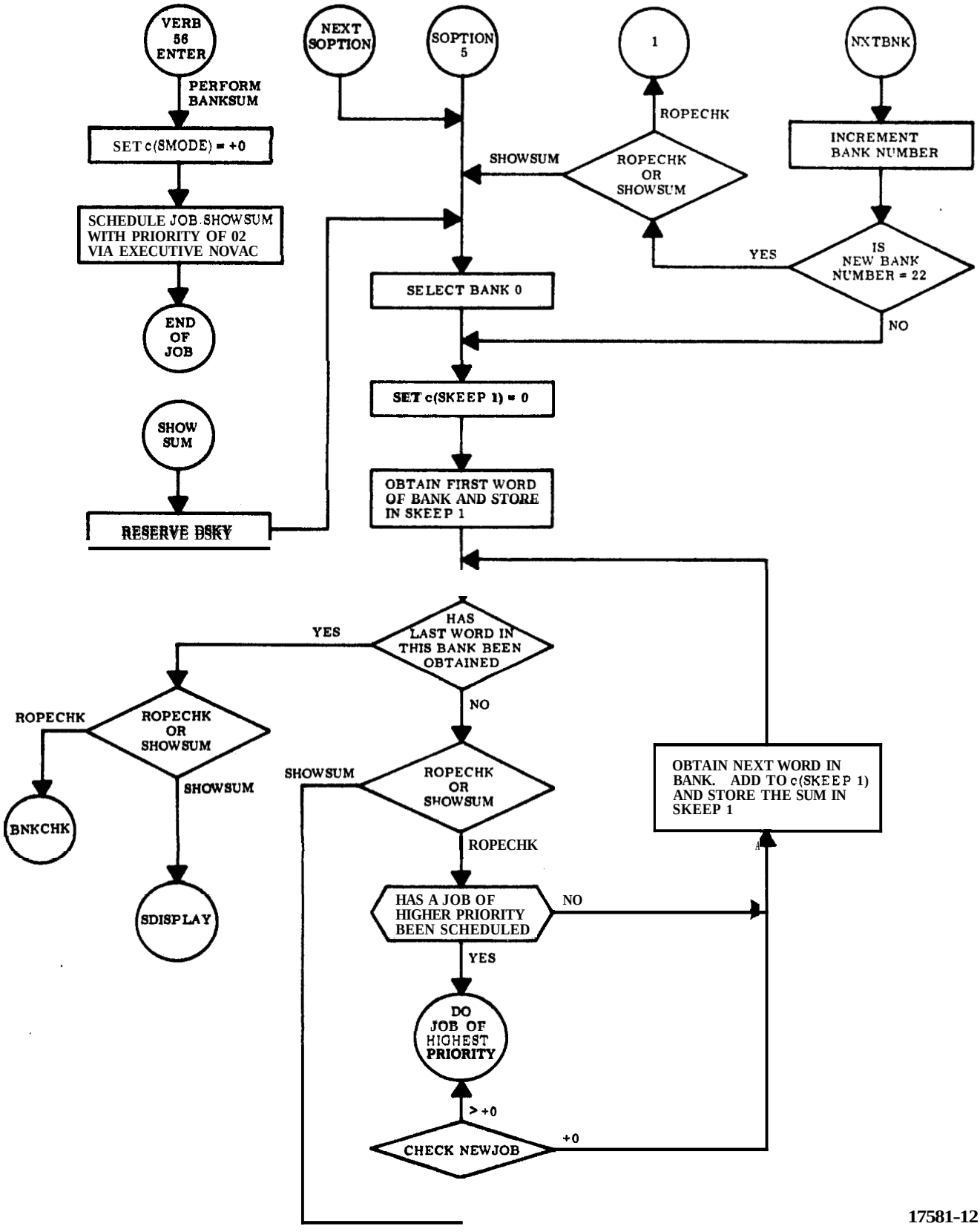


Figure 4-8. Self Check (Sheet 12 of 19)

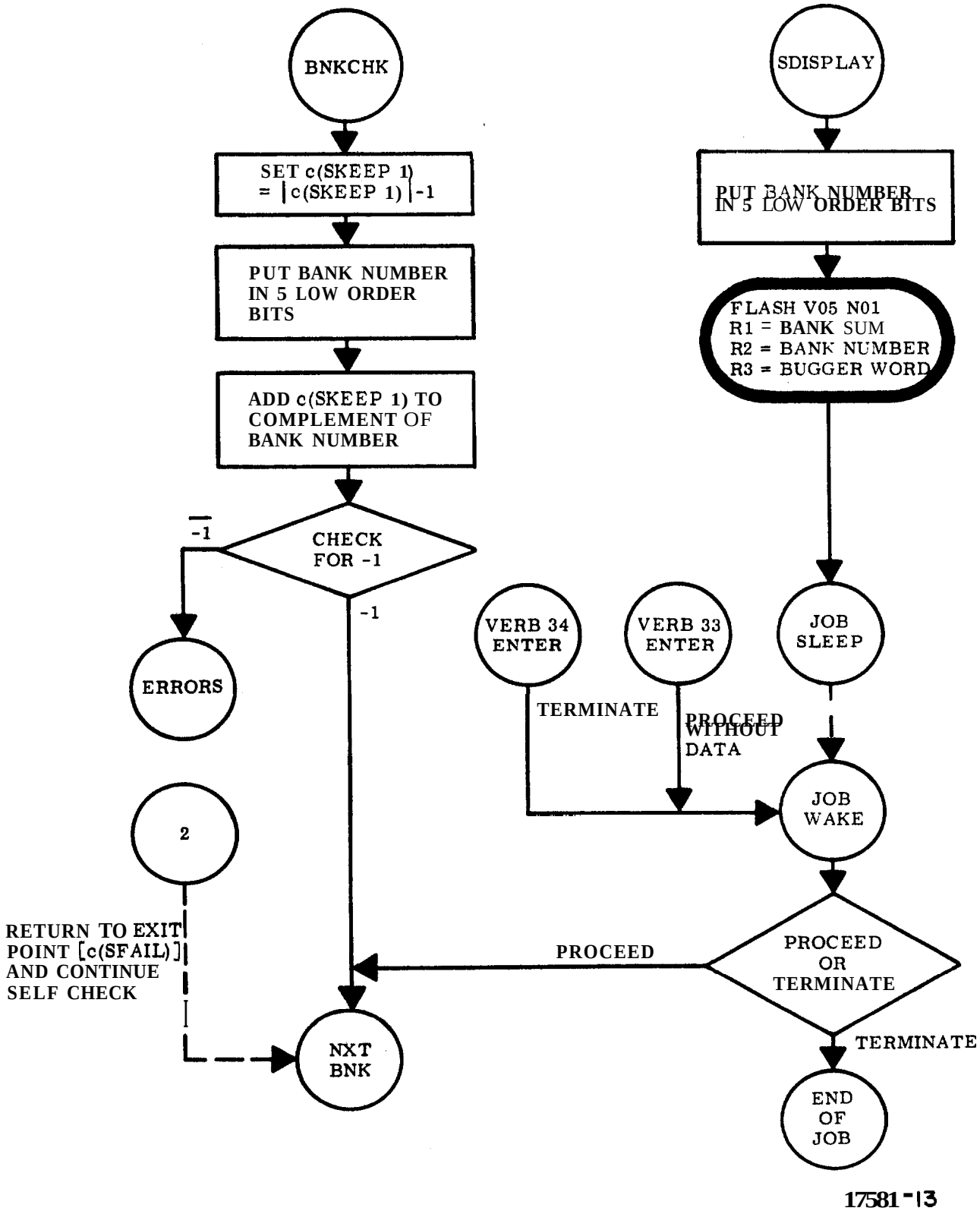


Figure 4-8. Self Check (Sheet 13 of 19)

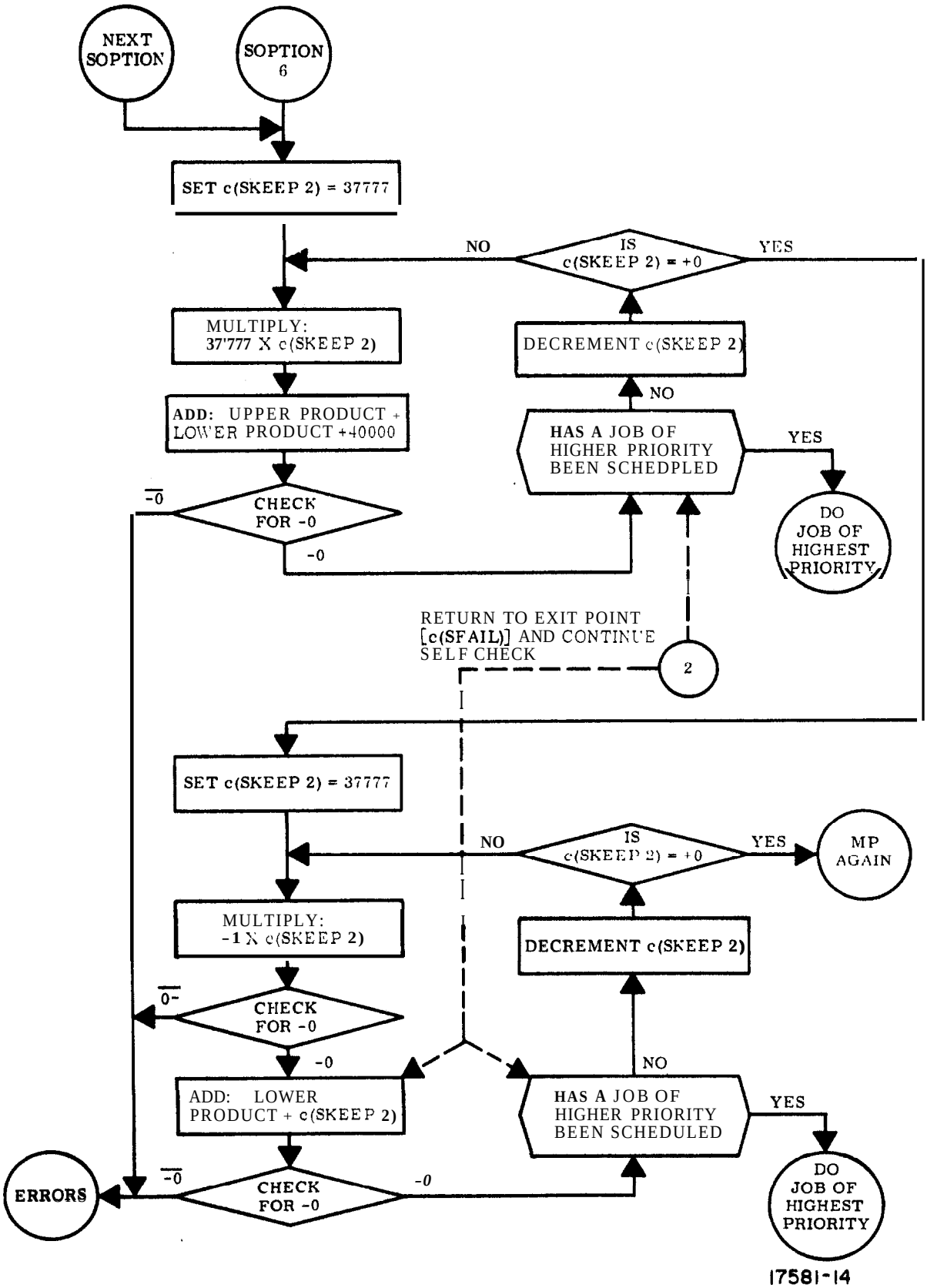


Figure 4–8. **Self Check** (Sheet **14 of 19**)

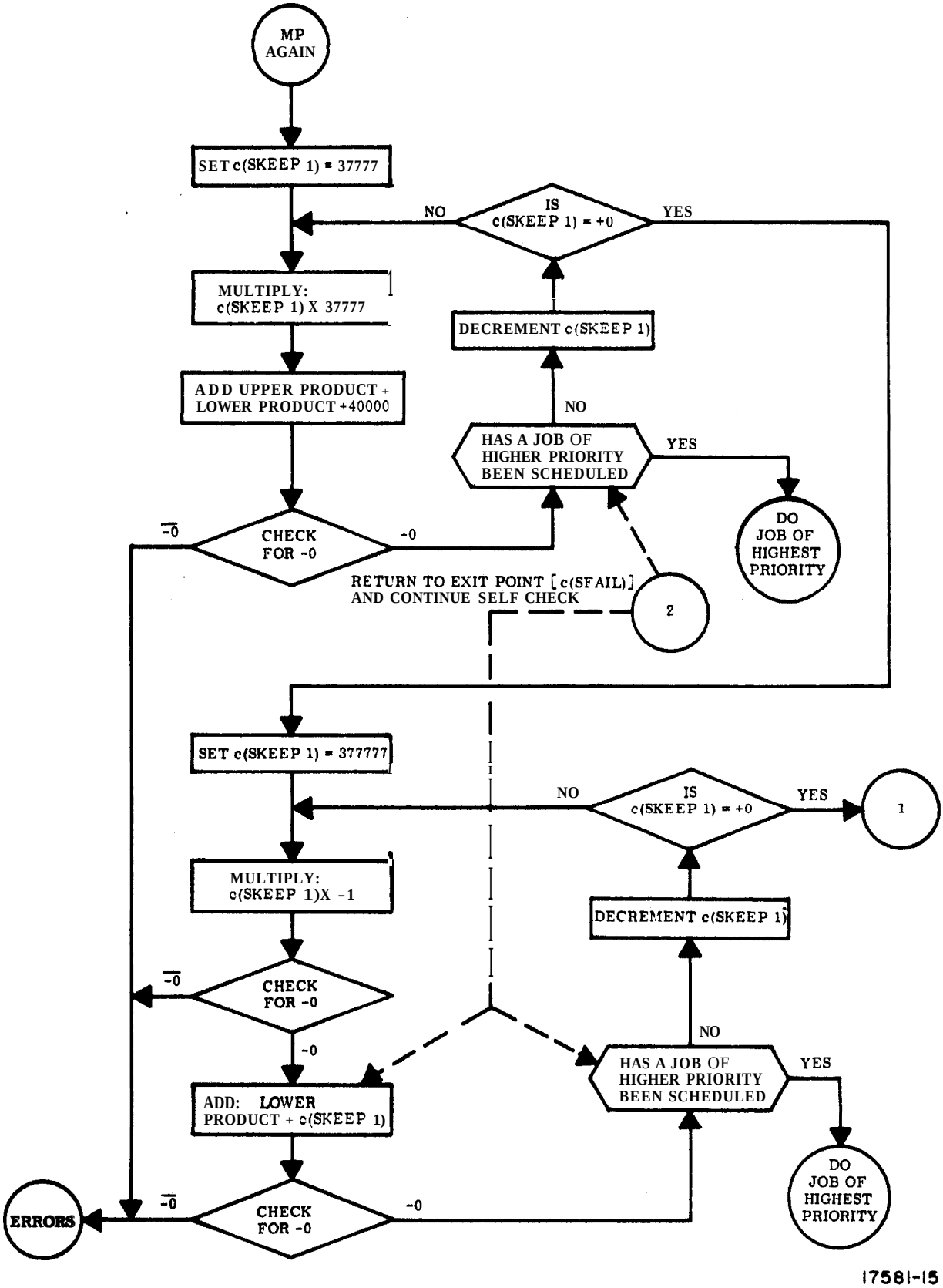


Figure 4-8. Self Check (Sheet 15 of 19)

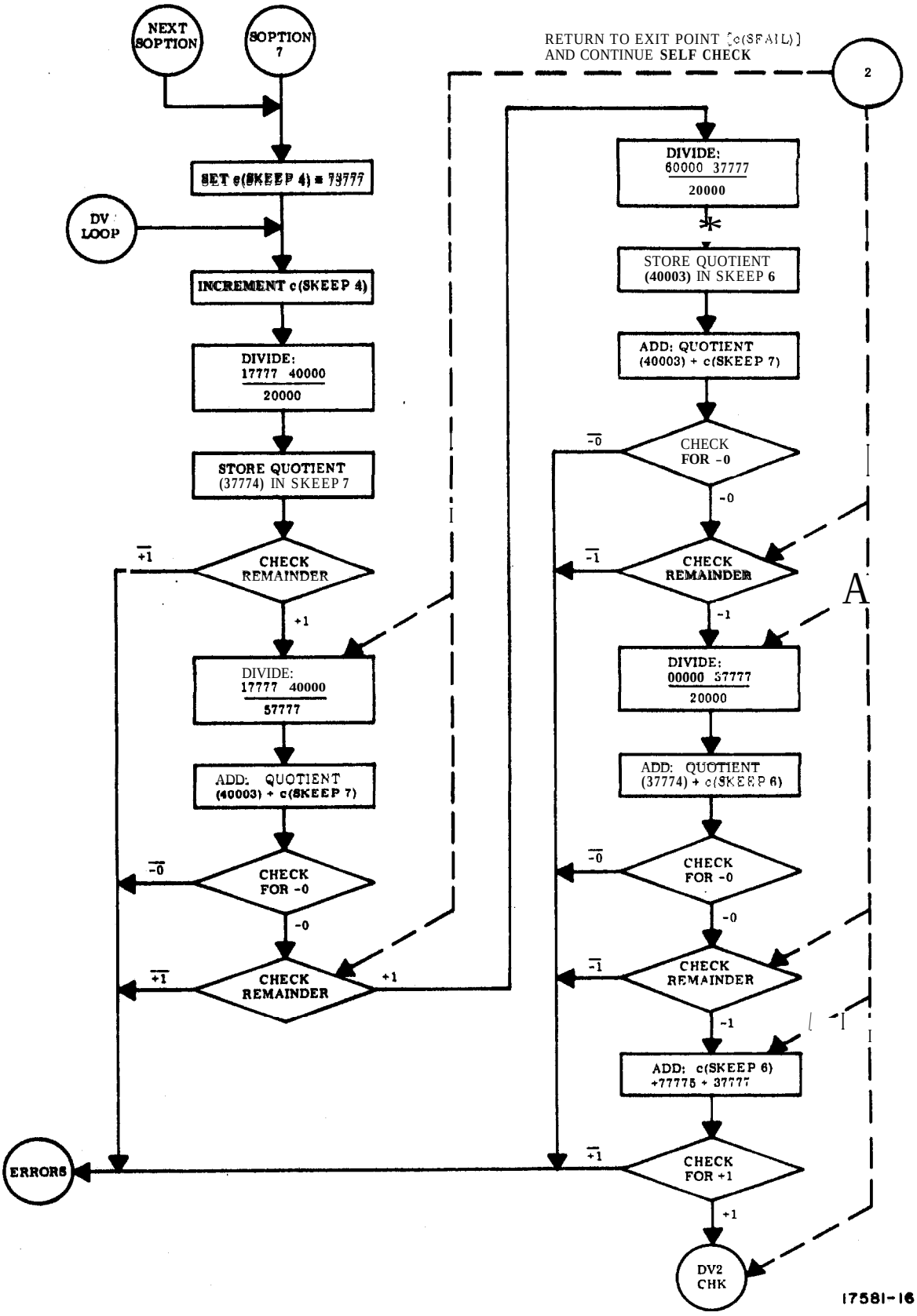


Figure 4-8. Self Check (Sheet **16 of 19**)

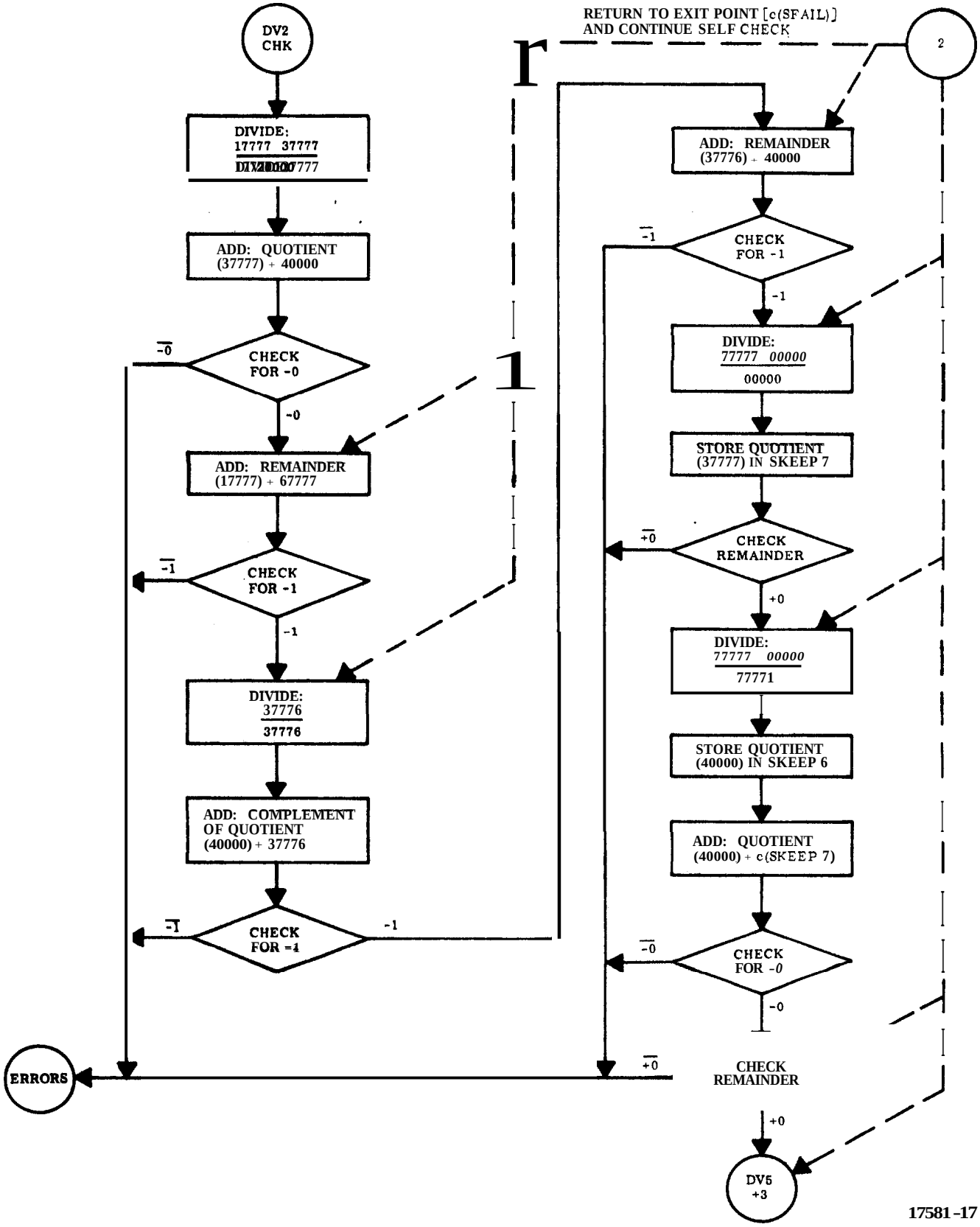


Figure 4-8. Self Check *(Sheet* 17 of 19)

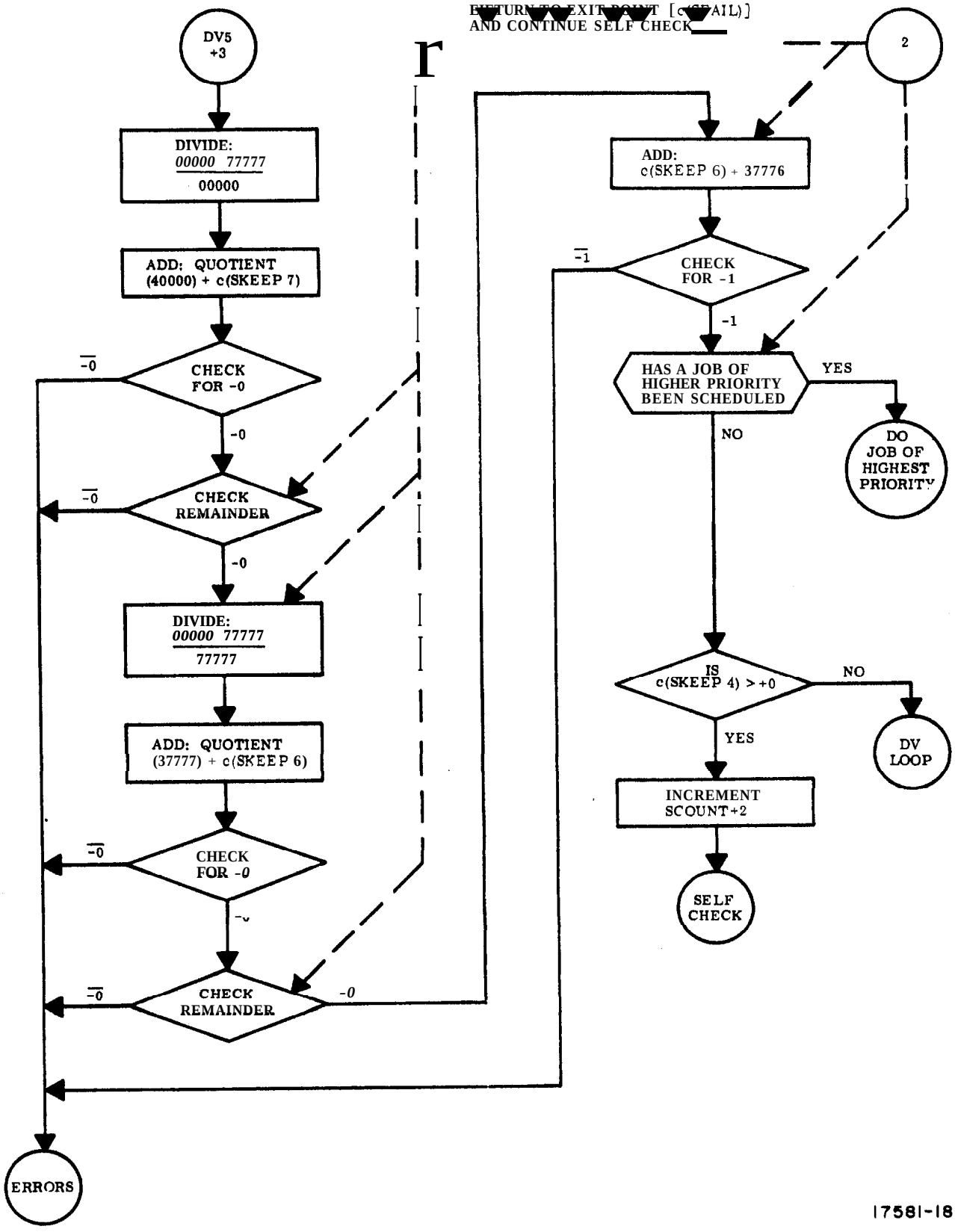
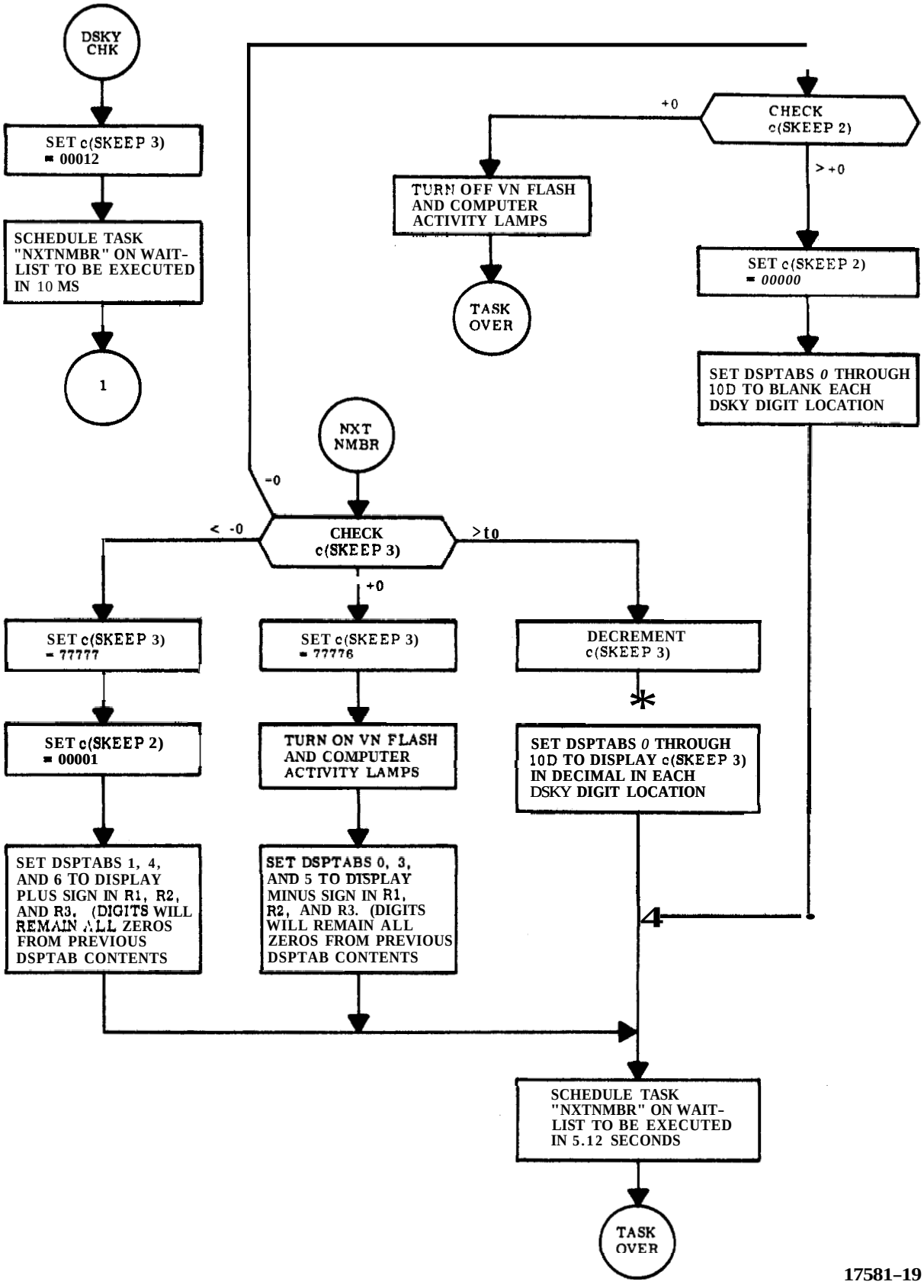


Figure 4-8. Self Check (Sheet 18 of 19)



\Figure 4-8. Self Check (Sheet 19 of 19)

APPENDIX A

COMPUTER PROGRAMS

A. 1 DESCRIPTION OF COMPUTER INSTRUCTIONS

Instructions, which are directions given to perform specific operations, are the same for CMC and LGC. Together with data addresses, they constitute the building blocks of a program. Programs are sequential lists of instruction words. There are two general categories of inetructions, machine and interpretive. Several types of instructions used in the LGC may be categorized as follows:

MACHINE (56)

REGULAR (42)

BASIC (15)

EXTRACODE (12)

CHANNEL (7)

SPECIAL (8)

INVOLUNTARY (9)

INTERRUPT (2)

COUNTER (7)

PERIPHERAL (5)

INTERPRETIVE

The machine instructions can be interpreted and executed directly by using the sequence generator to control the LGC operation. The interpretive instructions are a programmer's convenience and must be interpreted under program control, converted to machine instructions and then executed as machine instructions. Table A-1 lists the machine instructions alphabetically and gives a brief description of each. The reader will find it to his advantage to refer back to this table once he has gained a greater familiarity with the LGC. The following symbols are used in table A-1.

K represents any address in the central processor, erasable memory or fixed memory.

F represents an address in fixed memory only.

E represents an address in the central processor or erasable memory.

H represents any channel address.

C represents any counter address.

A represents the **A** register on the central processor.

L represents the L register in the central processor.

c(K) represents the contents of **K,** i.e., the data located in address K.

1, I + **1,** I + **2** represents the addresses of successive instruction words stored in memory.

c (I), C (I + 1), **C (I + 2)** represents the contents of successive instruction words stored in memory.

A. 1.1 MACHINE INSTRUCTIONS. The LGC has three classes of machine instructions: regular, involuntary, and peripheral. Regular instructions are programmed and are executed in whatever sequence they have been stored in memory. Involuntary instructions (with one exception) are not programmable and have priority over regular instructions: no regular instruction can be executed when the **LGC** forces the execution of an involuntary instruction. The peripheral instructions are used during ground testing when the LGC is connected to the CTS or **PAC:** the LGC cannot perform any program operation during a peripheral instruction.

A. 1.2 REGULAR INSTRUCTIONS. There are four types of regular instructions: basic, channel, extracode, and special. The difference between the regular instructions is directly related to the way in which the LGC interprets an instruction word. Instruction words stored in memory are called "basic instructions words" and consist of a three bit order code and a twelve bit address code. The order code defines an operation and the address code defines a location.

The contents of the **SQ** register will determine what instruction the LGC will perform. The *SQ* register reflects that data transferred into it from memory. The *SQ* register consists of six bits and an EXTend bit (figure A-1). A binary point is assumed to be located between bits thirteen and twelve. When an instruction word is transferred from memory to the *SQ* register, bfts **15** through **10** of the word in memory are transferred to bits **16** and **14 through 10** of the *SQ* register (figure A-2). In the following paragraph, however, only the transfer of bits **15, 14** and **13** from memory to bits **16, 14** and **13** of the *SQ* register will be considered.

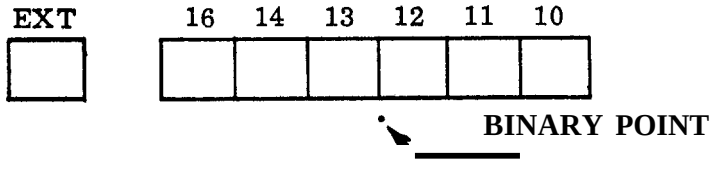


Figure **A-1.** *SQ* Register

Table A-1. Machine Instructions, Alphabetical Listing (Sheet 1 of 8)

Symbolic Instruction Word	Order Code	Description	Execution Time in MCT's
AD K	06.	Basic Instruction: add c(K) to c(A); stores result in A; takes next instruction from I + 1 where I is location of AD K.	2
ADS E	02.6	Basic Instruction: adds c(A) to c(E) and stores result in both A and E; takes next instruction from I + 1 where I is location of ADS E.	2
AUG E	12.4	Extra Code Instruction: adds +1 to c(E), if c(E) is positive and -1 if c(E) is negative; stores result in E; takes next Instructfon from I + 1 where I is location of AUG E. NOTE: AUG, DIM and INCR are slightly modified counter increment sequences. Accordingly, if one of this group overflows when addressing a counter for which overflow (during involuntary incrementing) is supposed to cause an interrupt, the interrupt will occur. It should be noted that all three of these instructions unlike the increment sequences, always operate in one's complement, even when addressing CDU counters,	2
BZF F	11.2 11.4 11.6	Extra Code Instruction: takes next instruction from F if c(A) is ±0; otherwise takes next instruction from I + 1 where I is location of BZF F.	1 if c(A) is ±0; otherwise 2
BZMF F	16.2 16.4 16.6	Extra Code Instruction: takes next instruction from F if c(A) is +0 or negative; otherwise takes next instruction from I + 1 where I is location of BZMF F.	1 if c(A) is +0 or negative, otherwise 2
CA K	03.	Basic Instruction: clears c(A) and copies c(K) into A; takes next instruction from I + 1 where I is location of CA K.	2

Table A-1. Machine Instructions, Alphabetical Listing *(sheet 2 of 8)*

Symbolic Instruction word	Order Code	Description	Execution Time in MCT's
CCS E	01.0	Basic Instruction: if c(E) is nonzero and positive, takes next instruction from I + 1 where I is location of CCS E, adds -1 to c(E) and stores result in A. If c(E) is +0, takes next instruction from I + 2 and sets c(A) to +0. If c(E) is nonzero and negative, takes next instruction from I + 3, adds -1 to the abeolute value of the c(E) and stores result in A. If c(E) is -0, takes next instruction from I + 4 and *seta* c(A) to +0.	2
CS K	04.	Basic Instruction: clears c(A) and copies $\overline{c(K)}$ into A; takes next instruction from I + 1 where I is location of CS K.	2
CYL	,0022	Special Instruction: cycles quantity, which is entered into location 0022, one place to left.	
CYR	,0020	Special Instruction: cycles quantity, which is entered into location 0020, one place to right,	
DAS E	02.0	Basic Instruction: adds c(A, L) to c(E, E + 1); stores result in E and E + 1; sets c(L) to +0 and sets c(A) to net overflow if address E is not 0000_8. Net overflow is +1 for positive overflow -1 for negative overflow, otherwise c(A) is set to +0. Takes next instruction from I + 1 where I is location of DAS E. Note: DAS A doubles the contents of the double precision accumulator - implied address code DDOUBL assembles as DAS A. Since the hardware must operate on the low order operands first, consider DAS as the operation code 20001 to which the address E is added to for the instruction.	3
DCA K	13.	Extra Code Instruction: copies c(K, K + 1) into A and L; takes next instruction from I + 1 where I is location of DCA K.	3

Table A-1. Machine Instructions, Alphabetical Listing (Sheet 3 of 8)

Symbolic Instruction word	Order Code	Description	Execution Time in MCT's
DCS K	14.	Extra Code Instruction: copies c(K, K + 1) into A and L; takes next instruction from I + 1 where I is location of DCS K.	3
DIM E	12.6	Extra Code Instruction: adds -1 if c(E) is nonzero and positive and +1 if c(E) is nonzero and negative; stores result in E; if c(E) is ±0, c(E) is not changed; takes next instruction from I + 1 where I is location of DIM E. See NOTE under AUG.	2
DINC C	None	Counter Instruction: adds +1 to c(C) if c(C) is negative: adds -1 to c(C) if c(C) is positive; provides no change if c(C) is ±0; stores result in C, delays program execution for 1 MCT.	1
DV E	11.0	Extra Code Instruction: divides c(A, L) by c(E); stores quotient in A; stores remainder in L; takes next instruction from I + 1 where I is location of DV E. NOTE: The signs of the double length dividend in A & L need not agree. The net sign of the dividend is the sign of c(A) unless c(A) is ±0, in which case it is the sign of c(L). The remainder bears the net dividend sign, and the quotient sign is determined strictly by the divisor and net dividend signs.	6
DXCH E	05.2	Basic Instruction: exchanges c(E, E + 1) with c(A, L); takes next instruction from I + 1 wher I is location of DXCH E.	3
EXTEND	00.0006	Special Instruction: Take the next instruction from I + 1, where I is the EXTEND instruction and execute it as an extracode instruction. If I + 1 is INDEX (full operation code 15), the following instruction will also be executed as an extracode.	1
FETCH K	None	Peripheral Instruction: reads and displays c(K) as binary numbers on CTS or PAC where K is address supplied by CTS or PAC.	2

Table A-1. Machine Instructions, Alphabetical Listing (Sheet 4 of 8)

Symbolic Instruction Word	Order Code	Description	Execution Time in MCT's
EDOP	.0023	Special Instruction: shifts quantity, which is entered into location 0023, seven places to left.	
COJ	00.	Interrupting Instruction: transfers control to instruction stored in location 40008 and proceeds from there.	2
INCR E	02.4	Basic Instruction: adds + 1 to c(E); stores result in E; takes next instruction from I + 1 where I is location of INCR E. See NOTE under AUG.	2
INHINT	00.0004	Special Instruction: Inhibit program interrupts until a subsequent RELINT.. Take the next instruction from I + 1where I was INHINT. NOTE: The inhibition set by INHINT and removed by RELINT in entirely independent of the one set by an interrupt and removed by a RESUME.	1
INOT LD H	None	Peripheral Instruction: loads data supplied by CTS or PAC into location H where H is channel address also supplied by CTS or PAC.	1
INOTRD H	None	Peripheral Instruction: reads and displays c(H) as binary number on CTS or PAC where H is channel address supplied by CTS or PAC.	1
LXCH E	02.2	Basic Instruction: exchanges c(E) with c(L); takes next instruction from I + 1where I is location of LXCH E.	2
MCDU C	None	Counter Instruction: adds -1 (two's complement) to c(C). NOTE: Incrementing in two's complement modulator notation transfers octal 40000 to 37777 and 00000 to 77777 and is otherwise like one's complement. PCDU and MCDU replace PINC and MINC for counters 0032 through 0036.	1

Table A-1. Machine Instructions, Alphabetical Listing (Sheet 5 of 8)

Symbolic Instruction Word	Order Code	Description	Execution Time in MCT's
MINC C	None	Counter Instruction: adds -1 to c(C); delays program execution for 1 MCT. If negative overflow occurs, c(C) is set to -0.	1
MP K	17.	Extra Code Instruction: multiplies c(A) by c(K); stores result in A and L; c(A, L) agree in sign; takes next instruction from I + 1 where I is location of MP K. A zero result is positive unless c(A) = ±0 and c(K) is non-zero with the opposite sign.	3
MSK K	07.	Basic Instruction: AND's c(A) with c(K); stores result in A; takes next instruction from I + 1 where I is location of MSK K.	2
MSU E	12.0	Extra Code Instruction: forms signed one's complement difference between c(A) and c(E) where c(A) and c(E) are unsigned (modular or periodic) two's complement numbers; stores result in A; the method is to form the two's complement difference, to decrement it if it is negative, and to take the overflow-uncorrected sum as the result; takes next instruction from I + 1 where I is location of MSU E.	2
NDX K	05.0	Basic Instruction: adds c(K) to c(I + 1) where I is location of NDX E; takes sum of c(K) + c(I + 1) as next instruction. INDEX 0017 is an implied instruction to resume an interrupted program.	2
NDX K	15.	Extra Code Instruction: adds c(K) to c(I + 1) where I is location of NDX K; sets extra code switch; sum of c(K) + c(I + 1) becomes an Extra Code Instruction which is taken as next instruction. This INDEX will not act as a RESUME.	2
PCDU C	None	Counter Instruction: adds +1 (two's complement) to c(C); delays program execution for 1 MCT. See NOTE under MCDU.	1

Table A-1. Machine Instructions, Alphabetical Listing (Sheet 6 of 8)

Symbolic Instruction Word	Order Code	Description	Execution Time in MCT's
PINC C	None	Counter Instruction: adds +1 to c(C); delays program execution for 1 MCT. If positive overflow occurs, the counter is set to +0 and an interrupt is set up if the counter is T3, T4, T5 or set up a PINC for T2 if the counter was T1.	1
QXCH E	12.2	Extra Code Instruction: exchanges c(E) with c(Q); takes next instruction from I + 1 where I is location of QXCH E.	2
RAND H	10.2	Channel Instruction: AND's c(H) with c(A); stores result in A; takes next instruction from I + 1 where I is location of RAND H.	2
READ H	10.0	Channel Instruction: copies c(H) into A; takes next instruction from I + 1 where I is location of READ H.	2
ROR H	10.4	Channel Instruction: Inclusive OR's c(H) with c(A); stores result in A; takes next instruction from I + 1 where I is location of ROR H.	2
RELINT	00.0003	Special Instructions: Removes program interrupt inhibits. Allows interrupts after this instruction subject to the restriction that an interrupt cannot occur while there is plus or minus overflow in A.	1
RESUME	05.00 17	Special Instruction: takes next instruction from return address (location of which address is stored in location 0017). This allows the resumption of the interrupted program.	1
RUPT	10.7	Interrupting Instruction: takes next instruction from address supplied by Interrupt Priority Control; stores c(B) (instruction that was to be executed) in location 0017_8; stores c(Z) = I in location 00158 where I is assigned location of instruction stored in 0017. This instruction is for machine checkout only.	3

Table A-1. Machine Instructions, Alphabetical Listing (Sheet 7 of 8)

Symbolic Instruction Word	Order Code	Description	Execution Time in MCT's
RXOR H	10.6	Channel Instruction: forms the exclusive OR of c(H) and c(A); stores result in A; takes next instruction from I + 1 where I is location of RXOR H.	2
SHANC C	None	Counter Instruction: doubles c(C) and adds +1; stores result in C; delays program execution for 1 MCT. This action amounts to shifting c(C) one digit to the right and adding +1. NOTE: SHANC and SHINC are used to convert incoming serial bit streams into words for parallel access.	1
SHINC C	None	Counter Instruction: doubles c(C); stores result in C; delays program execution for 1 MCT. This action amounts to shifting c(C) one digit to the right. See NOTE under SHANC.	1
SR	.0021	Special Instruction: shifts quantity, which is entered into location 0021, one place to right.	
STORE E	None	Peripheral Instruction: data supplied by CTS or PAC is stored in location E where E is address supplied by CTS or PAC; delays program execution for 2 MCT's.	2
SU E	16.0	Extra Code Instruction: subtracts c(E) from c(A); stores result in A; takes next instruction from I + 1 where I is location of SU K.	2
TC K	00.	Basic Instruction: takes next instruction from K; stores I + 1 in Q where I is location of TC K; if K is 0006 (EXTEND), sets extra code switch and takes next instruction from I + 1; if K is 0004_8 (INHINT), sets inhibit interrupt switch and takes next instruction from I + 1; if K is 0003_8 (RELINT), resets inhibit interrupt switch and takes next instruction from I + 1.	1

Table A-1. Machine Instructions, Alphabetical Listing (Sheet 8 of 8)

Symbolic Instruction word	Order Code	Description	Execution Time in MCT's
TCF F	01.2 01.4 01.6	Basic Instruction: takes next instruction from F. Does not change the contents of *Q*.	1
TCSAJ K		Peripheral Instruction: takes next instruction from K where K is address supplied by CTS or PAC.	2
TS E	05.4	Basic Instruction: if c(A) is not an overflow quantity, copies c(A) into E and takes next instruction from I + 1 where I is location of TS E; if c(A) is a positive overflow quantity, copies c(A) into E, sets c(A) to +1, and takes next instruction from I + 2; if c(A) is a negative overflow quantity, copies c(A) into E, sets c(A) to -1, and takes next instruction from I + 2.	2
WAND H	10.3	Channel Instruction: AND's c(H) with c(A); stores result in H and A; takes next instruction from I + 1 where I is location of WAND H.	2
WOR H	10.5	Channel Instruction: Inclusively OR's c(H) with c(A); stores result in H and A; takes next instruction from I + 1 where I is location of WOR H.	2
WRITE H	10.1	Channel Instruction: copies c(A) into H; takes next instruction from I + 1 where I is location of WRITE H.	2
XCH E	05.6	Basic Instruction; exchanges c(A) with c(E); takes next instruction from I + 1 where I is location of XCH E.	2

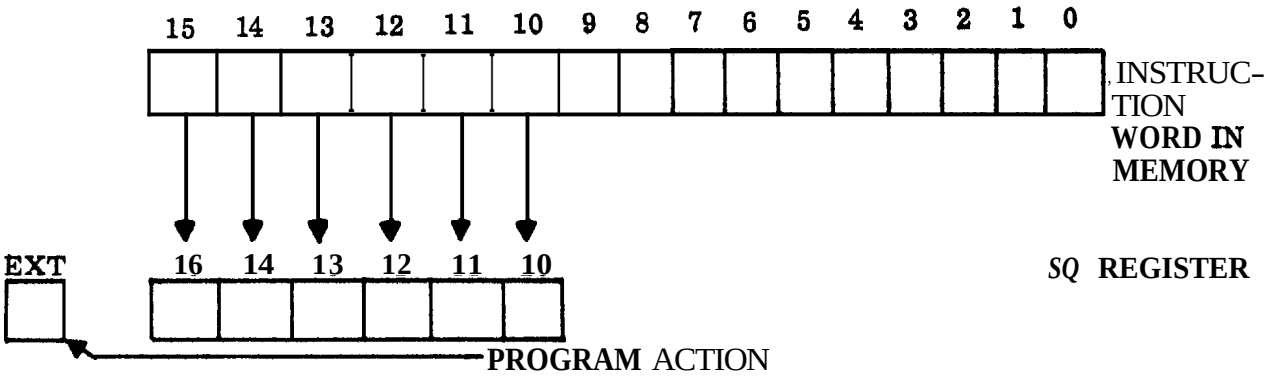


Figure A-2. Memory to *SQ* Register Transfer

The three bit order code in the memory basic instruction words has a capability of uniquely deflning eight operations: To increase the number of operations defined by the *SQ* register, bit EXT (extend) is made a 1 or *0* under program control, therefore, bits EXT, 16, 14 and 13 of the *SQ* register define sixteen operations.

Note the order codes in column 2 of table A-1. These order codes are determined, in most cases, by the contents of the SQ register. Figure A-3 shows how the order codes in table A-1 are related to the actual contents of the *SQ* register. The order code defined by figure A-3 is TS E.

In table A-1, the instructions can be categorized into three distinct groups by their listings in the order code column.

a. Those that list "None."

b. Those that list four digits to the right of the binary point.

c. Those that list two or three digits with the binary point written to the right of the second digit.

Group a contains the counter and peripheral instructions. There are no order codes associated with these instructions.

Group b contains the special instructions that are address dependent basic instructions. Their order codes are, in part, determined by bits 1 through 12. Those special instructions with no digits to the left of the decimal point can be combined with any basic instruction order code. Those with digits to the left of the decimal point are combined with that basic instruction whose order code appears to the right of the decimal point.

Group c contains the basic, extracode and channel instructions, i.e., all the regular instructions with the exception of the special instructions. Also included in this group are the two interrupt instructions; these are not regular instructions.

Note that the instructions in this group may or may not have a digit to *the* right of *the* decimal point. When there is a digit to the right of the decimal point, it is determined by bits 11 and 12 or bits 10, 11 and 12 of the *SQ* register. When bits 11 and 12 are necessary to extend the order code field, their configuration is called a "quarter code." When bits 10, 11 and 12 are necessary to extend the order code field, their configuration is called an "eighth code." Table A-2 shows the configuration of the various quarter and eighth codes associated with this group. Note that there are two ways of defining a zero or an even digit to the right of the decimal point. Observe instructions CCS E and TCF F in table A-1. These instructions are identical if only the digits to the left of the decimal point are considered. There, two instructions can be distinguished, however, if bits 11 and 12 of the SQ register are observed. Note that the content of bit 10 in register *SQ* is irrelevant because only four cases have to be distinguished and, consequently, a quarter code is sufficient to define the necessary operation. Now observe the instruction in table A-2 which have digits 1 and *0* to the left of the decimal point in the order code column. There are eight of these instructions and to differentiate between them, bits 10, 11 and 12 of the *SQ* register are necessary because eight cases must be differentiated. If just bits 11 and 12 were used, only four cases could be distinguished.

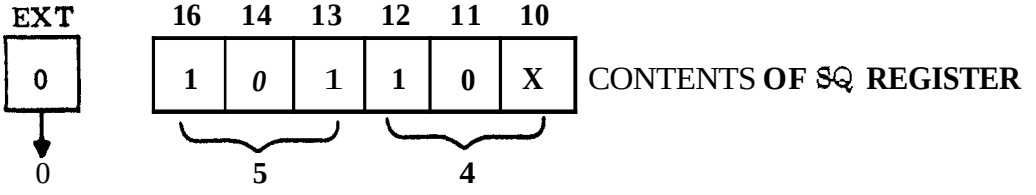


Figure A-3. Order Code Determination

Basic instructions can be differentiated from extracode and channel instructions by the left hand digit of the order code, If bit EXT in the *SQ* register is a *0*, then the left hand digit is a zero and the instruction is a basic instruction. If bit EXT is a 1, then the left hand digit is a one and the instruction is an extracode or channel instruction.

A. 1.3 **INVOLUNTARY INSTRUCTIONS.** The involuntary instruction class contains two types of instructions - interrupt and counter. The interrupt instructions use the basic instruction word format just as the regular instructions do. However, the interrupt instructions ace not entirely programmable, The contents of the order code field and the address field are supplied by computer logic rather than the program. The counter instructions have no instruction word format. Signals which function as a decoded order code specify the counter instruction to be executed and the computer logic supplies the address. The address for these instructions is limited to one of 29 counter locations in memory.

There are two interrupt instructions. One instruction initializes the computer when power is first applied and when certain program traps occur. The other interrupt instruction is executed at regular intervals to indicate time, receipt of new telemetry or keyboard data, or transmission of data by the computer. This interrupt instruction may be programmed to test the computer.

EIGHTH OR QUARTER CODES		SQ REGISTER BITS		
		12	11	10
EIGHTH	.0	0	0	0
QUARTER	.0	0	0	X
EIGHTH	.1	0	0	1
EIGHTH	.2	0	1	0
QUARTER	.2	0	1	X
EIGHTH	.3	0	1	1
EIGHTH	.4	1	0	0
QUARTER	.4	1	0	X
EIGHTH	.5	1	0	1
EIGHTH	.6	1	1	0
QUARTER	.6	1	1	X
EIGHTH	.7	1	1	1

X stands for a 1 or 0

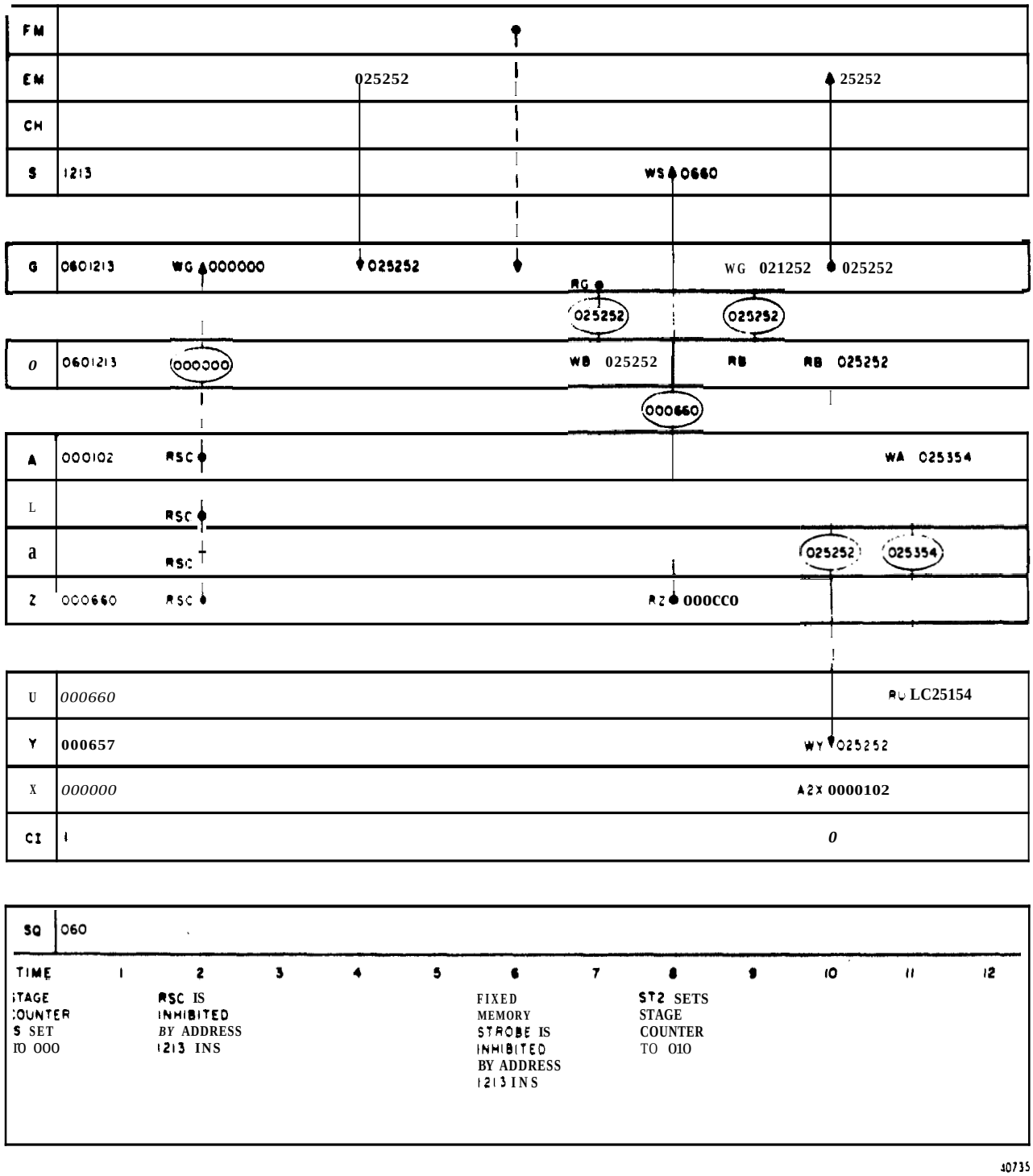
Table A-2. Quarter and Eighth Codes

There are several counter instructions. Two instructions will either increment or decrement by one the content of a counter location using the one's complement number system, Two other instructions perform the same function using the two's complement number system. Certain counter instructions control output rate signals and convert serial telemetry data to parallel computer data.

A.1.4 PERIPHERAL INSTRUCTIONS. There are two types of peripheral instructions. One type deals with memory locations and the other type deals with channel locations. The peripheral instructions are not used when the computer is in the spacecraft. They are used when the computer is connected to peripheral equipment during subsystem and preinstallation system testing. The peripheral instructions are not programmable and are executed when all computer program operations have been forcibly stopped. These instructions are used to read and load any memory or channel location and to start the computer program at any specified address. The peripheral instructions and counter instructions are processed identically.

A. 1.5 INSTRUCTION DATA FLOW. Examples of the instruction data flow are illustrated on *the* subinstruction flow charts ADO, STD2, RSM3, and NDXO, figures A-4, A-5, A-6 and A-'7,

A. 1.6 INTERPRETIVE INSTRUCTIONS. Interpretive instructions, a programmer's convenience and a means of saving memory storage area, must be interpreted under program control, converted to machine instructions and then executed as machine instructions. The coding into interpretive instructions of routines which contain double precision, triple precision, vector, and vector matrix operations results in a considerable saving in program storage area in fixed memory. This saving is achieved at the expense of computer operating speed; however, when operating in basic machine language the computer operates much faster than the equipment with which it interfaces. Since most of the PGNCS problems the computer is required to solve involve complex mathematical equations, the use of interpretive instructions for vector matrix algebra and complex differential calculus is a definite asset. Interpretive instructlone are shown in table A-3.



10735

Figure A-4. **Subinstruction ADO, Data Transfer Diagram**

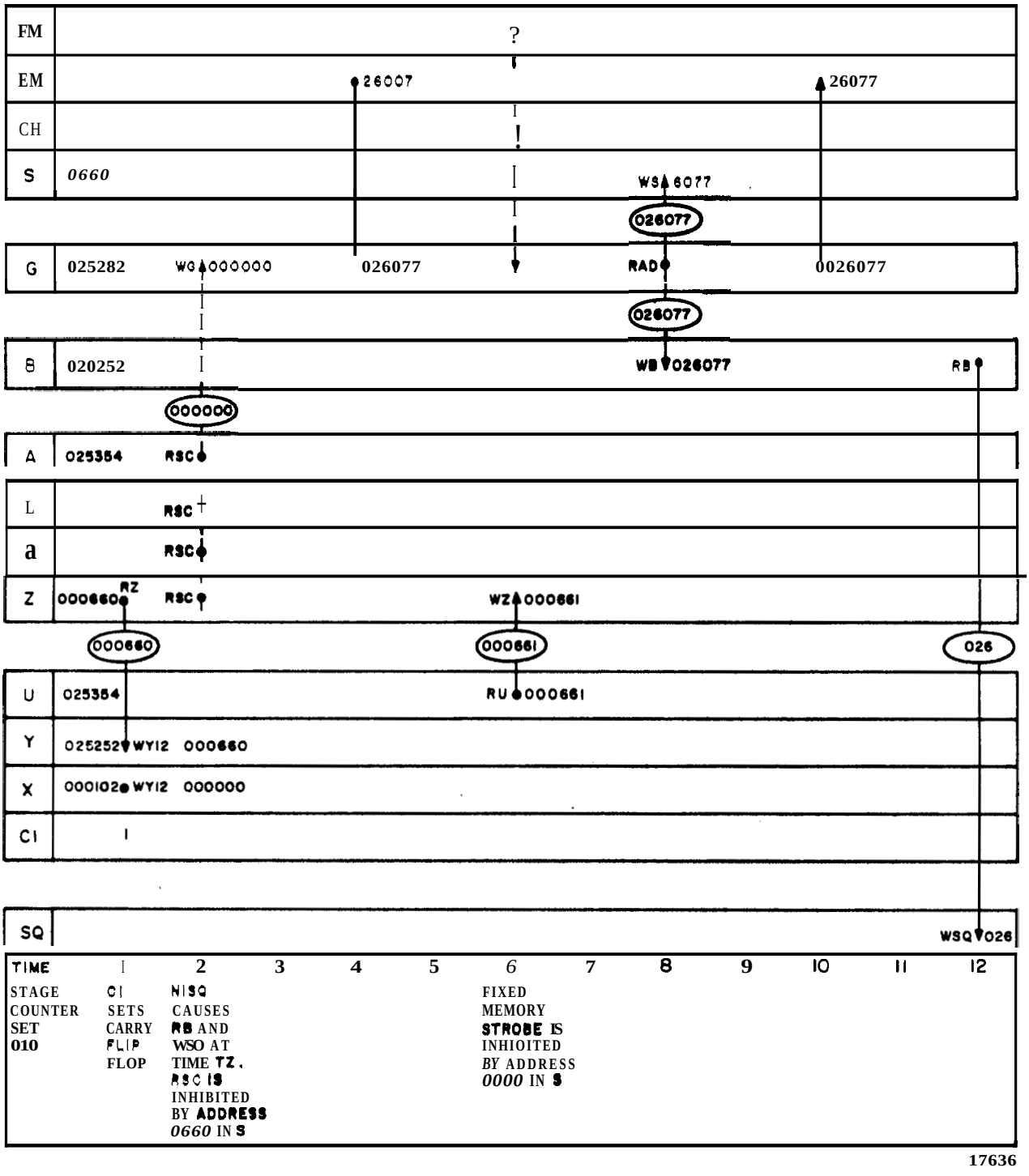
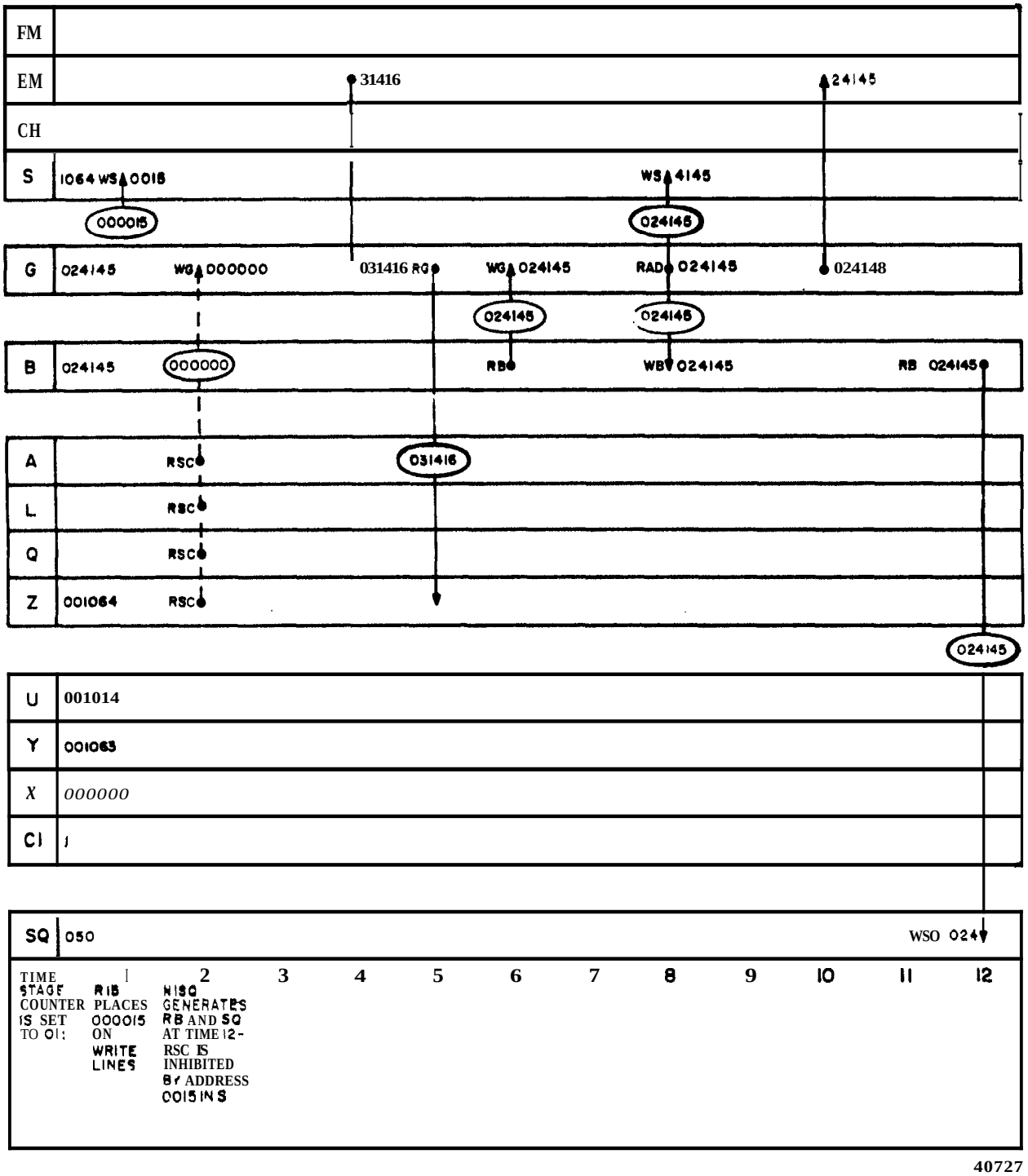


Figure A-5, **Subinstruction STD2, Data Transfer Diagram**



40727

Figure A-6. Subinstruction RSM3, Data Transfer Diagram

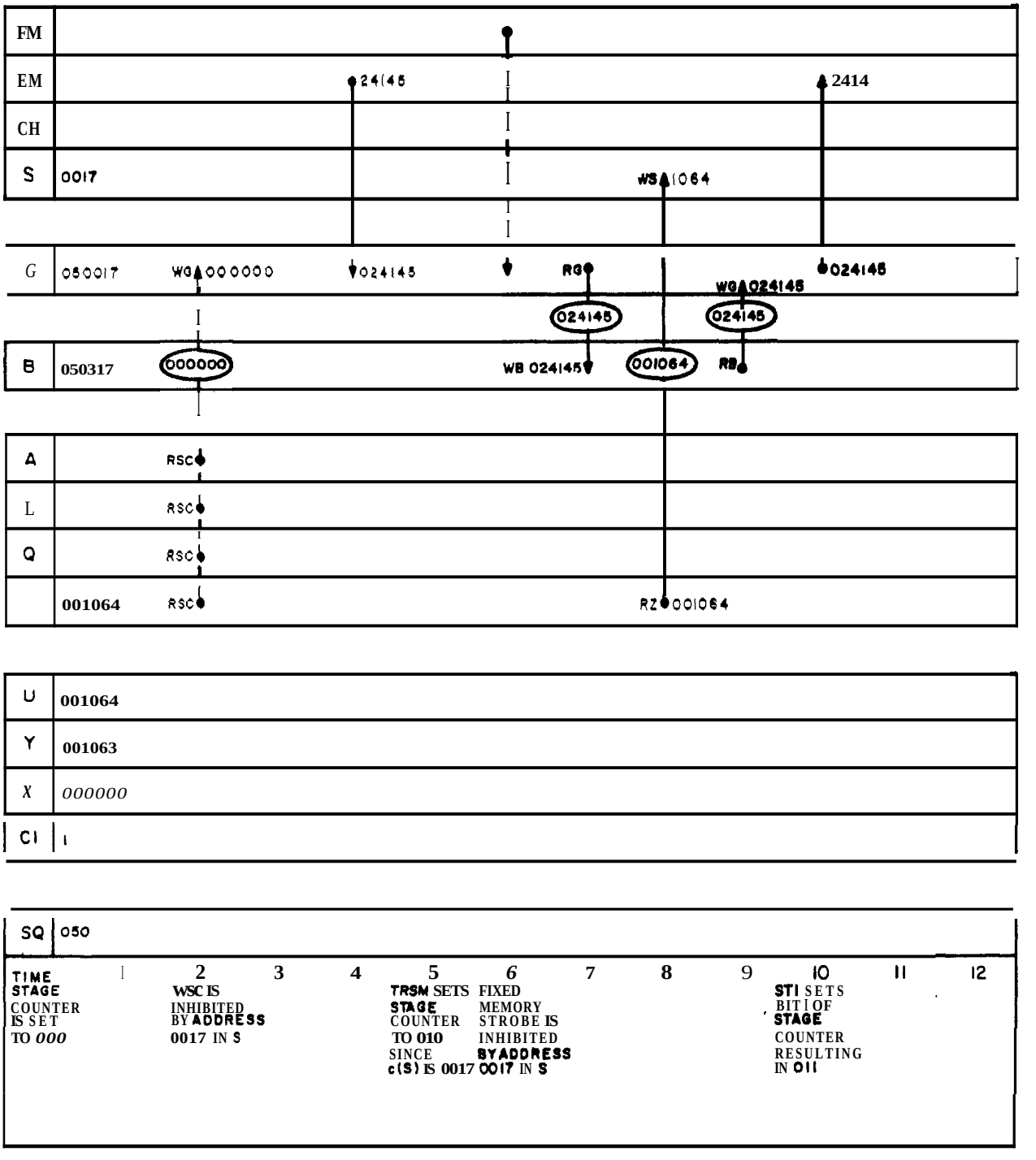


Figure A-7. **Subinstruction NDXO, with Implied Address Code RESUME,** Data **Transfer Diagram**

Table A-3. Interpretive Instructions (Sheet 1 of 14)

Detailed Description of Operation Codes with Probable Average Execution Times.	Addressing Class
A. Store, Load, and Push-Down Instructions.	
STORE X Store MPAC .62 m. s. D(MPAC), T(MPAC) or V(MPAC) replace D(X), T(X), or V(X), respectively. X may be indexed or direct.	01 11
STODL X Store MPAC Y and re-load in DP 1.24 m. s. D(MPAC), T(MPAC) or V(MPAC) replace D(X), T(X) or V(X). (D(Y), 0) become T(MPAC) setting the store mode to DP. X may be indexed, or direct and Y indexed, direct or vacuous (push-up).	01 11
STOVL X Store MPAC Y and re-load as Vector 1.43 m. s. Same as STODL except V(X) become V(MPAC) and store mode is set to vector.	01 11
STCALL X Store MPAC Y and CALL a Routine 1.40 m. s. D(MPAC), T(MPAC), or V(MPAC) replace D(X), T(X) or V(X), leaving the store mode unaltered. Call the routine at Y, leaving a return address (of the location after the second address) in QPRET, Both addresses must be direct.	01 11
DLOAD X Load MPAC in DP .64 m. s. (D(X), 0) become T(MPAC), setting the store mode to DP. Address may be direct, indexed or vacuous.	01 11
TLOAD X Load MPAC in TP .77 m. s. Same as DLOAD except T(X) become T(MPAC) and store mode is set to TP.	01 11
VLOAD X Load MPAC with a Vector .91 m. s. Same as DLOAD except V(X) become V(MPAC) and store mode is set to vector.	01 11

Table A-3. Interpretive Instructions (Sheet 2 of 14)

			Addressing Class
SLOAD	X	Load MPAC in Single Precision .74 m.s.	'01 11
Same as DLOAD except (S(X), 0, 0) become T(MPAC). X may not be vacuous.			
PDDL	X	Push Down and load MPAC in DP .81 m. s.	01 11
D(MPAC), T(MPAC) or V(MPAC) are pushed down; (D(X), 0) become (T(MPAC) with the store mode set to DP. X may be direct, indexed, or vacuous.			
PDVL	X	Push Down and load MPAC with a vector 1.14 m. s.	01 11
Same as PDDL except V(X) become V(MPAC) and the store mode is set to vector.			
PUSH		Push Down .55 m. s.	01 11
D(MPAC), T(MPAC) or V(MPAC) are pushed down.			
SETPD	X	Set Push-down Pointer .58 m. s.	
Set the Push-down Pointer PUSHLOC to X, where X is in local erasable memory. X must be direct.			

B. Scalar Arithmetic Operations - All addresses may be direct, indexed, or vacuous.

DAD	X	DP Add .66 m. s.	01 11
D(MPAC) + D(X) replace D(MPAC). Set OVFIND on Overflow, and leave the overflow-corrected result in MPAC.			
DSU	X	DP Subtract .66 m. s.	01 11
D(MPAC) -D(X) replace D(MPAC). Set OVFIND on overflow, and overflow-correct the result.			
BDSU	X	DP Subtract From .74 m. s.	01 11
D(X) - D(MPAC) replace D(MPAC). Set OVFIND on overflow, and overflow-correct the result.			

				Addressing Class
DMP	X	DP Multiply	1.13 m. s.	01 11
D(X) times D(MPAC) replace T(MPAC).				
DMPR	X	DP Multiply and Round	1.29 m. s.	01 11
D(MPAC) D(X) = P is formed and rounded to DP *so* that (P,*0)* replace T(MPAC).				
DDV	X	DP Divide By	2.48 m. s.	01 11
If | D(MPAC) | < | D(X) | , the DP quotient Q=D(MPAC) / D(X) is formed and (Q, 0) replace T(MPAC). Overflow indication is set if required. ±.999999 replace D(MPAC) in this case.				
BDDV	X	DP Divide Into	2.50 m. s.	01 11
Same as DDV except Q = D(X) / D(MPAC) if | D(X)| < | D(MPAC) | .				
SIGN	X	DP Sign Test	.70 m. s.	01 11
X must be inerasable memory. If D(X) ≥ *0*, no operation occurs. Otherwise if store mode is DP or TP, -T(MPAC) replace T(MPAC); if store mode is vector, -V(MPAC) replace V(MPAC).				
TAD	X	TP Add	.75 m. s.	
T(MPAC) + T(X) replace T(MPAC). OVFIND is set on overflow, with the overflow-corrected result left in MPAC.				

C. Vector Arithmetic Operations.

All addresses may be direct, indexed, and any but MXV and VXM may have vacuous addresses.

VAD	X	Vector Add	.92 m. s.	01 11
V(MPAC) +V(X) replace V(MPAC). Set OVFIND on overflow in any component, leaving the overflow-corrected result.				

Table A-3. Interpretive Instructions (Sheet 4 of 14)

	Addressing Class
VSU X Vector Subtract .92 m. s. V(MPAC) - V(X) replace V(MPAC). Set OVFIND on overflow in any component, leaving an overflow-corrected result.	01 11
BVSU X Vector Subtract From 1.17 m. s. V(X) - V(MPAC) replace V(MPAC). Set OVFIND on overflow of any component, leaving an overflow-corrected result.	01 11
DOT X Vector Dot Product 3.08 m. s. V(MPAC)·V(X) replace T(MPAC), setting the store mode to DP. Set OVFIND if overflow occurs, leaving an overflow-corrected result.	01 11
VXSC X Vector Times Scalar 3.27 m. s. If the initial store mode is Vector, each component of V(MPAC) is multiplied by D(X), the rounded products replacing their respective X components of V(MPAC). If the initial store mode is DP or TP, change it to Vector, and each component of V(X) is multiplied by D(MPAC) to form V(MPAC) as above.	01 11
V/SC X Vector Divided by Scalar 5.39 m. s. If the initial store mode is Vector, each component of V(MPAC) is divided by D(X), the DP quotients replacing their respective components of V(MPAC). If the initial store mode is DP or TP, it is changed to Vector, and each component of V(X) is divided by D(MPAC) to form V(MPAC). If overflow occurs in any component, the operation is terminated with OVFIND set and unspecified results in MPAC.	01 11
VXV X Vector Cross Product 4.98 m. s. V(MPAC) * V(X) replace V(MPAC). Set OVFIND if overflow occurs, leaving an overflow-corrected result.	01 11
VPROJ X Vector Projection 5.75 m. s. [V(MPAC)·V(X)] V(X) replace V(MPAC). Set OVFIND on overflow, and leave the result obtained with overflow-corrected V(MPAC)· V(X)] .	01 11

Table A-3. Interpretive Instructions (Sheet 5 of 14)

	Addressing Class
VXM X Matrix Pre-Multiplication by Vector 8.98 m.s. $(V(MPAC)^T M(X))^T$ replace V(MPAC). Set OVFIND on overflow, leaving an overflow-corrected result.	01 11
MXV X Matrix Post-Multiplication by Vector 8.97 m.s. M(X) V(MPAC) replace V(MPAC). Set OVFIND on overflow, leaving an overflow-corrected result.	01 11

D. Scalar Functions.

SQRT DP Square Root 1.94 m.s. SQRT (D(MPAC)) replace T(MPAC); i.e. the initial contents of MPAC are normalized, the DP square root of the normalized number computed, and that result unnormalized so that MPAC +2 has marginal significance. Receipt of an argument less than -10^{-4} causes an abort.	00
SIN (SINE) DP Sine 5.63 m.s. .5 (Sin (2π D(MPAC)) replace T(MPAC).	00
COS(COSINE) DP Cosine 5.80 m.s. .5 (Cos (2π D(MPAC)) replace T(MPAC).	00
ARCSIN (ASIN) DP Arc-sine 9.26 m.s. $(1/2\pi)$ Arc-sine (2D(MPAC)) replace T(MPAC). This is the inverse of the SIN function. Receipt of an argument greater than .5001 in magnitude causes an abort.	00
ARCCOS (ACOS) DP Arc-Cosine 9.12 m.s. $(1/2\pi)$ Arc-Cosine (2D(MPAC) replace T(MPAC). This is the inverse of COS, Receipt of an argument whose magnitude is greater than .5001 causes an abort.	00
DSQ DP Square .76 m.s. D(MPAC) times (D(MPAC) replace T(MPAC).	00

Table A-3. Interpretive Instructions (Sheet 6 of 14)

			Addressing Class
ROUND	Round to DP	.56 m. s.	00
T(MPAC) are rounded to DP so that (ROUND (T(MPAC)), 0) replace T(MPAC). Set OVFIND if overflow occurs, leaving an overflow-corrected result, +0.			
DCOMP	TP Complement	.52 m. s.	00
-T(MPAC) replace T(MPAC).			
ABS	TP Absolute Value	.48 m. s.	00
|T(MPAC)| replace T(MPAC).			

E. Vector Functions.

UNIT	Unit Vector Function	6.46 m. s.	00
V(MPAC)/2 |V(MPAC)| replace V(MPAC). $\|V(MPAC)\|^2$ replace D(34D) and |V(MPAC)| replace D(36D). Set OVFIND if |V(MPAC)| < 2-21 or |V(MPAC)| ≥ 1 in which case the result is incorrect.			
ABVAL	Vector Length	3.86 m. s.	00
|V(MPAC)| become T(MPAC), changing the store mode to DP. In addition, $\|V(MPAC)\|^2$ replace D(34D). The result is zero if |V(MPAC)| < 2^{-21}. If |V(MPAC)| ≥ 1 set OVFIND to indicate unspecified result.			
VSQ	Square of Vector Length	2.21 m. s.	00
$\|V(MPAC)\|^2$ become T(MPAC), changing the store mode to DP. If |V(MPAC)| ≥ 1, set OVFIND and leave an overflow-corrected result.			
VCOMP	Vector Complement	.63 m. s.	00
-V(MPAC) replace V(MPAC).			
VDEF	Vector Define	.67 m. s.	00
Push up for V_Y and again for V_Z so that (D(MPAC), V_Y, V_Z) become V(MPAC), setting the store mode to vector.			

F. Shift Instructions.

			Addressing Class
1, Short Shifts			00
SR1	Scalar Shift Right	.85 m. s.	00
SR2		.85 m. s.	
SR3		.85 m. s.	
SR4		.85 m. s.	
T(MPAC) 2^{-j} replace T(MPAC) (j= 1, 2, 3, 4).			
SL1	Scalar Shift Left	.72 m. s.	00
SL2		.95 m. s.	
SL3		1.17 m. s.	
SL4		1.39 m. s.	
T(MPAC)x 2^{+j} replace T(MPAC) (j = 1, 2, 3, 4). If significant bits are lost, set OVFIND but leave the overflow-corrected result as T'(MPAC).			
SR1R	Scalar Shift Right and Round	.99 m. s.	*00*
SR2R		.99 m. s.	
SR3R		.99 m. s.	
SR4R		.99 m. s.	
T(MPAC)x 2^{-j} is rounded to a DP number R and (R, 0) replace T(MPAC) (j = 1, 2, 3, 4).			
SL1R	Scalar Shift Left and Round	.88 m. s.	*00*
SL2R		1.10 m. s.	
SL3R		1.32 m. s.	
SL4R		1.54 m. s.	
T(MPAC)x 2^{+j} is rounded to a DP number R and (R, *0)* replace T(MPAC) (j = 1, 2, 3, 4). If overflow occurs, set OVFIND and leave the overflow-corrected result as T(MPAC).			
VSR1	Vector Shift Right and Round	2.01 m. s.	00
VSR2		2.01 m. s.	
VSR3		2.01 m. s.	
CSR4		2.01 m. s.	
VSR5		2.01 m. s.	
VSR6		2.01 m. s.	
VSR7		2.01 m. s.	
VSR8		2.01 m. s.	
Each component of V(MPAC) is replaced by the original value multiplied by 2^{-j} and rounded to DP. (j = 1(1)8).			

			Addressing Class
VSL1	Vector Shift Left	.81 m. s.	00
VSL2		1.18 m. s.	
VSL3		1.55 m. s.	
VSL4		1.93 m. s.	
VSL5		2.30 m. s.	
VSL6		2.68 m. s.	
VSL7		3.05 m. s.	
VSL8		3.43 m. s.	
Each component of V(MPAC) is replaced by the original value multiplied by 2^{+j} (j = 1(1)8). If overflow occurs in any component, leave the overflow-corrected result and set OVFTND.			

2. General Shifts. Addresses may be direct or indexed.

SR	X	General Scalar Shift	1.38 m. s.	01 11
		Right	+.23 INTEGER (X/14) m. s.	
T(MPAC)x 2^{-X} replace T(MPAC) where $-42 < X < 42$ (X can be negative only if the address was indexed. Address limits are $0 < X < 42$ if direct and $-128 < X_S < 128$ if indexed. X_S is the stored address before index modification: X is the net address in any case. On overflow leave the overflow-corrected result and set OVFIND.				
SL	X	General Scalar Shift Left	1.03 m. s. +.22 X m. s.	01 11
Same as SR except that T(MPAC)2^{X} replace T(MPAC).				
SRR	X	General Scalar Shift Right and Round	1.52 m. s. + .23 INTEGER (X/14) m. s.	01 11
Same as SR except that T(MPAC)x 2^{-X} is rounded to a DP number R and (R,0) replace T(MPAC). Address limits are $0 < X < 29$ if direct.				
SLR	X	General Scalar Shift Left and Round	1.18 m. s. + .22 X m. s.	01 11
Same as SL except that T(MPAC)x 2^{X} is rounded to a DP number R and (R,0) replace T(MPAC). Direct address limits are $0 < X < 14$.				

				Addressing Class
VSR	**X**	General Vector Shift Right	**2.61** m. *s.* +. **82** INTEGER **(X/14)m.s.**	**01** **11**
Each component of V(MPAC) is replaced by the original value multiplied by 2^{-X} and rounded to **DP.** If **X** is an indexed address and the result address negative, do a **VSL -X** instead. Address limits are $0 < X < 29$ if direct and $-128 < X_S < 128$ if indexed.				
VSL	**X**	General Vector Shift Left	**.89** m. **a.** +. **37 X m. s.**	**01** **11**
Each component **of** V(MPAC) is replaced by the original **component** multiplied by 2^X. On overflow of any component, leave the overflow-corrected result and set OVFIND. If the address was indexed and the resulting address negative, VSR(-X) instead. Address limits' are $0 < X < 28$ if direct.				

3. Normalization. Address may be direct or indexed.

NORM(SLC)	X	Scalar Normalize	**.88** m. *s.* **+. 21 N m. s.**	**01** **11**
An N is found such that $\lvert T(MPAC) \rvert 2^N \geq .5$ provided T(MPAC) = 0. -N replaces S(X) and T(MPAC)x 2^N replace T(MPAC). If T(MPAC) = 0, -0 replaces S(X) and T(MPAC) are unchanged.				

G. Branching, Sequence Changing, and Subroutine Linkage Instructions.

All have a direct address except EXIT and **RVQ.** Any such address except those associated with transition to basic language **(RTB** and BOVB) is interpreted as indirect if it refers to erasable memory. Any level of indirect addressing is allowed.				
GOTO	**X**	Go To	**.77** m. *s.*	**10**
Begin executing interpretive instructions at **X. QPRET is** undisturbed. GOTO is a right-hand operation code.				
CALL	X	Call a Subroutine	**.89** m. *s.*	10
Begin executing interpretive instructions at **X. A** return address is left in QPRET. CALL is a right-hand operation code.				
CGOTO	**X** Y	Computed GoTo	.90 m. s.	**10**
The contents of X(X in erasable) are added to address ***Y(Y*** in fixed) and the address at ***Y*** + S(X) is selected. Begin executing interpretive instructions there unless the address is in erasable, in which case it **is** interpreted *as* indirect. CGOTO is a right-hand op code.				

				Addressing Class
CCALL	X Y	Computed Call	1.07 m. s.	10
Same as CGOTO except that a return address is left in **QPRET** in addition. CCALL is a right-hand op code.				
RVQ(ITCQ)		Return Via QPRET	.69 m. s.	10
Begin executing interpretive instructions at the location whose address is in QPRET. This may be used to return from a subroutine which contains no CALL or CCALL instructions. If QPRET contains the address of an erasable register, the address is interpreted as an indirect address. RVQ is a "right-hand op code".				
STQ(ITA)	X	Store QPRET	.69 m. s.	10
S(QPRET) replaces S(X) (X in erasable). This may be used to save the return address in subroutines which contain CALL and CCALL instructions. The STQ X in this case is eventually followed by GOTO X to return.				
BPL	X	Branch Plugs	.65 m. s. + .19 m. s. GO	10
If T(MPAC) > 0, do a GOTO X. Otherwise, no operation occurs.				
BZE	X	Branch Zero	.65 m. s. +. 19 m. s. GO	10
If T(MPAC) = 0, do a GOTO X. Otherwise, no operation occurs.				
BMN	X	Branch Minus	.67 m. s. +. 19 m. s. GO	10
If T(MPAC) < 0, do a GOTO X. Otherwise, no operation occurs.				
BHIZ	X	Branch High Order Zero	.6 m. s. +. 19 m. s. GO	10
If S(MPAC) = 0, do a GOTO X. Otherwise, no operation occurs.				
BOV	X	Branch On Overflow	.58 m. s. +. 23 m. s. GO	10
If OVFIND is set, reset it to zero and do a GOTO X. Otherwise, no operation occurs.				
BOVB	X	Branch On Overflow to Basic	.58 m. s. +. 16 m. s. GO	10
If OVFIND is set, reset it to zero and begin executing basic instructions at X. Otherwise, no operation occurs, X must be in fixed memory.				

Table A-3. Interpretive Instructions (Sheet 11 **of 14)**

				Addressing Class
RTB	X	Return to Basic	.71 m. s.	10
Begin executing basic instructions at **X. X** must **be in** fixed memory.				
EXIT		Exit from Interpreter	.26 m. s.	10
Begin executing basic instructions after the last op code or address word referenced by the interpreter as follows: **1)** If EXIT **is a** left-hand **op** code, *go* to the word **after** the EXIT instructions; **2)** If EXIT **is a** right-hand **op** code, go to the word following the last address used by the left-hand **op code.** EXIT is a right-hand op code.				

H. Switch Instructions,

SET	**X**	Set Switch	**1.27** m. **s.**	10
Set switch X to **1.**				
CLEAR	**X**	Clear Switch	**1.25** m. *s.*	**10**
Clear switch **X** to *0.*				
INVERT	**X**	Invert Switch	**1.27** m. *s.*	**10**
Invert switch X; i. e. , if *0,* set to **1;** if 1, clear to *0.*				
SETGO	X Y	Set Switch and Go To	1.54 m. *s.*	**10**
Set switch **X** to 1 and do a GOTO Y. SETGO is a right-hand **op** code,				
CLRGO	**X** Y	Clear Switch and Go To	**1.52 m. s.**	**10**
Clear switch **X** to ***0*** and do a GOTO ***Y.*** CLRGO ***is*** **a** right-hand op code.				
INVGO	**X** **Y**	Invert Switch and Go To	1.54 m. *s.*	**10**
Invert switch **X** and do a GOTO Y. INVGO is ***a*** right-hand **op** code.				

Table A-3. Interpretive Instructions (Sheet 12 of 14)

1. Switch Test Instructions.

				Addressing Class
BON	X Y	Branch if Switch On	1.26 m. s. +.23 m. s.	10
If switch X is set to 1, do a GOTO *Y*. Otherwise, no operation occurs.				
BOFF	X Y	Branch if Switch Off	1.27 m. s. +.23 m. s. GO	10
If switch X is cleared to 0, do a GOTO *Y*. Otherwise, no operation occurs.				
BONSET	X *Y*	Branch if Switch On, Setting Switch	1.37 m. s. +.23 m. s. GO	10
Set switch X to 1. If initially set to 1, do a GOTO *Y*. Otherwise, no further operation occurs.				
BOFSET	X Y	Branch if Switch Off, Setting Switch	1.39 m. s. +.23 m. s. GO	10
Set switch X to 1. If initially cleared to *0*, do a GOTO *Y*. Otherwise, no further operation occurs.				
BONCLR	X Y	Branch if Switch On, Clearing Switch	1.35 m. s. +.23 m. s. GO	10
Clear switch X *to 0*. If initially set to 1, do a GOTO *Y*. Otherwise, no further operation occurs.				
BOFCLR	X *Y*	Branch if Switch Off, Clearing Switch	1.36 m. s. +.23 m. s. GO	10
Clear switch X to *0*. If initially cleared to 0, do a GOTO Y. Otherwise, no further operation occurs.				
BONINV	X Y	Branch if Switch On, Inverting Switch	1.37 m. s. +. 23 m. s. GO	10
Invert switch X. If originally set to 1, do a GOTO Y. Otherwise, no further operation occurs.				
BOFINV	X Y	Branch if Switch Off, Inverting Switch	1.39 m. s. +.23 m. s. GO	10
Invert switch X. If originally cleared to *0*, do a GOTO Y. Otherwise, no operation occurs.				

Table A-3. Interpretive Instructions (Sheet 13 of 14)

J. Index Register Instructions.			Addressing Class
AXT, 1 X AXT, 2 X	Address to Index True	.75 m. s.	10
X replaces S(XT) (T = 1, 2).			
AXC, 1 X AXC, 2 X	Address to Index Complemented	.76 m. s.	10
-X replaces S(XT).			
LXA, 1 X LXA, 2 X	Load Index from Erasable	.78 m. s.	10
S(X) replaces S(XT).			
LXC, 1 X LXC, 2 X	Load Index from Erasable Complemented	.78 m. s.	10
-S(X) replaces S(XT).			
SXA, 1 X SXA, 2 X	Store Index in Erasable	.78 m. s.	10
S(XT) replaces S(X).			
XCHX, 1 X XCHX, 2 X	Exchange Index with Erasable	.83 m. s.	10
S(XT) replaces S(X) which then replaces S(XT).			
INCR, 1 X INCR. 2 X	Increment Index	.76 m. s.	10
The overflow-corrected sum of S(XT) and X replaces S(XT).			
XAD, 1 X XAD, 2 X	Index Register Add	.77 m. s.	10
The overflow-corrected sum of S(XT) and S(X) replace S(XT).			
XSU, 1 X XSU, 2 X	Index Register Subtract	.78 m. s.	10
The overflow-corrected difference S(XT) - S(X) replaces S(XT).			

				Addressing Class
TIX, 1 TM, 2	X X	Transfer on Index	.78 m. s. +.26 m. s. GO	10
If S(XT) ≤ S(ST) (T=1,2), no operation occurs. Otherwise, S(XT) - S(ST) replaces S(XT) and a GOTO X is executed.				

K. Miscellaneous Instructions.

SSP	X Y	Set Single Precision	.67 m. s.	10
Y replaces S(X). Y may be any constant: arithmetic, logical, address, etc.				
STADR		Push Up On Store Code	.26 m. s.	00
During assembly, the appearance of STADR causes the next store code to be stored complemented. During execution, STADR complements the next word to be referenced by the interpreter and enters the store code processor. STADR is a right-hand op code.				

APPENDIX B.

EXPLANATION OF SAMPLE PROGRAM LISTING

INTRODUCTION

A page from a program listing is shown on Figure B-1. Call numbers 1 through 20 have been added for ease of explanation. The numbers below correspond with the call numbers on the figure.

1. Assembler language.
2. Program title,
3. Program listing page number.
4. Routine title.
5. Routine page number.
6. Basic data.
7. Calling sequence of the Alarm routine by another routine. Assume TC ALARM = L and OCT AAANN = L + 1. The alarm number is NN and the general area of the alarm is AAA. A listing of the complete alarm numbers (AAANN) and their definitions is provided in the section of this study guide which contains the explanation of the Alarm routine.
8. This is a card used to indicate to the assembler to start the assembly of the routine at at octal address 5644.
9. This notifies the programmer that the EBANK being used is the same bank as is associated with FAILREG. In this case, E2 is used.
10. Column depicting the programmer's punch card number.

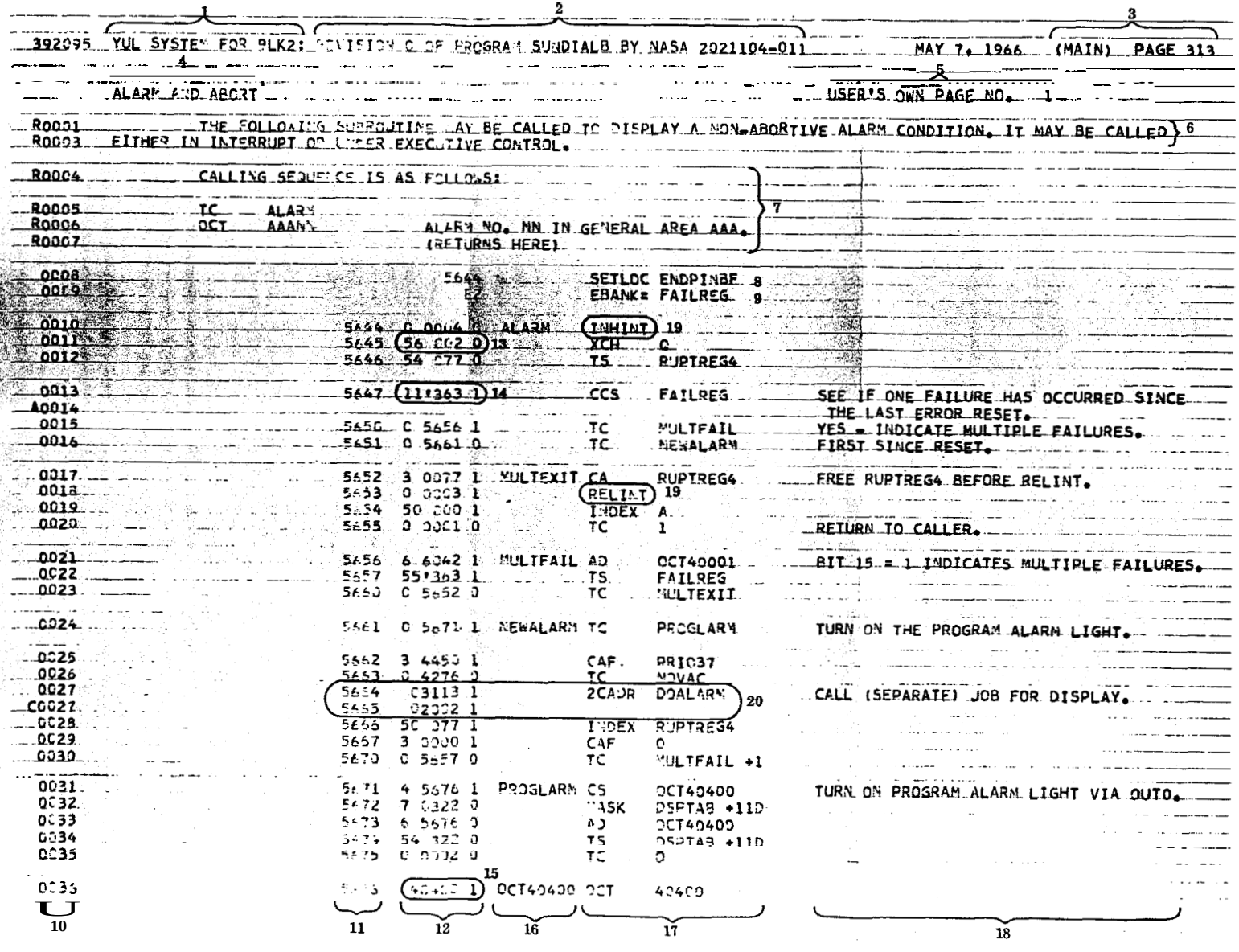
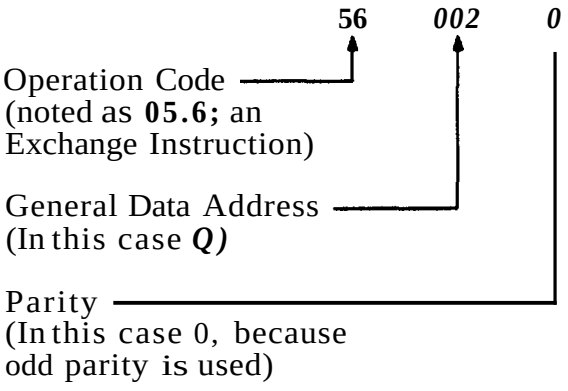


Figure B-1. Sample Program Listing

11. Column depicting the fixed memory address. **FBANK 2 is** implied when the general address is **40008** or *50008*. **FBANK 3 is** implied when the general address **is** *60008* **or** *70008*. Any other **FBANK** is designated by, for example, **01, 2160** where **01 is** FBANK 1 and **2160** is the general address.

12. Column depicting the contents of the adjacent fixed memory address. The contents can be instruction or data words. Two instruction words are explained in **13** and **14.** A data word is explained in **15.**

13. A typical instruction word is **shown** and is broken down into its parts below:



XCH Q (mneumonics)

(TAKE CONTENTS **OF** *Q* **AND** PUT IT **INTO A, AND** TAKE THE **CONTENTS OF A AND PUT IT INTO** *Q)*

14. Shown below is a contraction of an instruction word as noted by the apostrophe mark.

11 ' 363 1 CCS **FAILREG**

To determine the operation code, general address, EBANK and Parity, the word will be converted from octal to binary:

BITS	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
BINARY NOTATION	0	*0*	1	0	0	1	0	1	1	1	1	0	0	1	1
OCTAL NOTATION		1			1			3			*6*			3	1

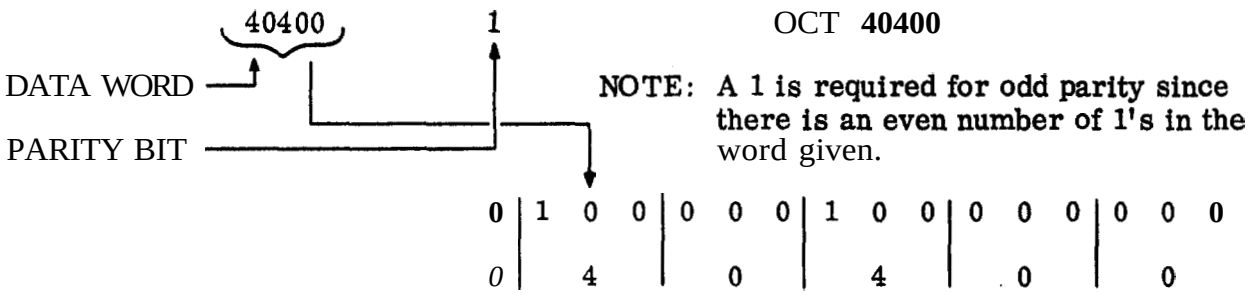
PARITY BIT

Bits **1 - 10** define the general address, 1363_8.

Bits **9 -** 10 define the **EBANK,** 2_8.

Bits **11 - 15** define the operation code 01.0_8.

15. Shown below is a typical data word:



16. Subroutine or content referral.

17. Mneumonics associated with the contents of the adjacent memory location (for ease of reading only).

18. Programmer's remarks (explain unique situations or functions).

19. Note that INHINT, inhibit program interrupt, and RELINT, **remove** program interrupt inhibit, are implied instructions using the operation code **of** "transfer control" to an address which is specifically designated to functionally establish the inhibits or release inhibits of program interrupts.

20. The 2CADR address constant is described below. This constant code is intended to be used as the operand of a DTCB (DXCH **Z**) instruction.

Two constant words are generated by this code, The first word, in this case, Octal **03113,** is called GENADR (General Address) while the second word, Octal 02002, is called BBCON (Both Bank Constant), The explanation of 2CADR DOALARM is as **follows:**

DOALARM is located in FBANK **01.**

c(BBANK) =Octal **02002** = Binary 0 0 0 0 1, 0 0 0 0 0 0 0 0 1 0

FBANK = 01_8 EBANK = 2_8

Since FBANK uses'the top five bits of BBANK (Both Banks), the FBANK associated with the ZCADR is 01. The low three bits of BBANK are associated with **EBANK.** Therefore, the EBANK used will also be **2.**

APPENDIX C

INTERPRETNE PROGRAMMING

INTRODUCTION

The list-processing, algebraic interpreter is a Program Section in the CMC's and LGC's Fixed Memory which is used to decode and execute mission function routines that are stored in the computer's memory in interpretive language. Interpretive language is defined as a pseudo-algebraic language which is oriented toward the specific types of problems that must be solved on the Apollo Mission. Mission function routines can be prepared and then assembled in interpretive language with a considerable saving of required program-storage memory area, however the savings in program-storage is offset by the additional time required to perform a specific arithmetic operation,

The problem orientated programs are assembled by another computer program referred to as a YUL assembler. The assembled program is then wired into the fixed memory of the Apollo computer, The Interpreter Program Section converts the problem orientated program listing into the basic machine instructions, or language, at the time on interpretive program is to be executed.

The organization and operation of the Interpreter Program Section and an analysis of interpretive language programming will be presented after a general discussion of multi-precision operation of the computers under program control.

C. 1 MULTI-PRECISION OPERATION

C. 1.1 MULTI-PRECISION NUMBERS. Most of the variables used in solving the camplex, mathematical equations which make up the Apollo Guidance & Navigation Program require more accuracy in expression than can be obtained from 14 significant bits of binary data. Therefore, the interpretive language system for the Apollo computers is centered around multiple-precision computation. The Interpreter Program Section provides for three, multi-precision modes of operation:

a. Double Precision Mode: where the double-precision quantity x is stored at locations x and x + 1, and where the value of x is equated to $\left[c(X) + c(X+1) \quad x\, 2^{-14}\right]$.

b. Triple Precision Mode: where the triple-precision quantity u is stored at locations U, U+1, and U+2, and where the value of u is equated to $\left[c(U) + c(U+1)\, x\, 2^{-14} + c(U+2)\, x\, 2^{-28}\right]$.

c. Double Precision Vector Mode: where the three, double-precision, orthogonal components r, s, and t of vector v are stored at locations V through V + 5, and where the value of the components are equated to $r = \left[c(V) + c(V+1) x\, 2^{-14}\right]$, $s = \left[c(V+2) + c(V+3)\, x\, 2^{-14}\right]$, and $t = \left[c(V+4) + c(V+5)\, x\, 2^{-14}\right]$.

As with single-precision variables in the basic machine language, the fixed, binary point for a multi-precision variable is considered to be located between bit positions **15** and **14** of the most significant **14** binary bits; and therefore, the variable is expressed entirely fractional.

C. **1.2** INTERPRETIVE ACCUMULATOR - MPAC. In order to carry out multi-precision calculations, the Interpreter Program Section is provided with a Multi-Purpose Accumulator (MPAC) which is used in a manner similar to the accumulator in the computer's Central Processor. The use of the MPAC is completely under program control. The MPAC consists of a set of seven E-Memory locations which are assigned by the Executive and which are, by definition, located in the Core Set Area for the active job that is currently being executed. Actually, the first seven locations in the Core Set Area for each job listed on the Job list can be considered to be the MPAC for that job, (See figure **2-1** Executive Core Set List), However, these locations are really used for temporary storage locations only if the job has been placed on the Job list, but not yet initiated; or, they are used to store the contents of the MPAC when the job is "put to sleep" or when the job is displaced for a higher-priority job **by** the Executive. The array of the seven locations of the MPAC is similar to the array for any multi-precision quantity (i.e. x, u, or v), except that in the Double-Precision Vector Mode, the three double-precision components are stored at locations $\left[\text{MPAC} + (\text{MPAC} + \mathbf{1}) \times 2^{-14}\right]$, $\left[(\text{MPAC} + \mathbf{3}) + (\text{MPAC} + \mathbf{4}) \times 2^{-14}\right]$ and $\left[(\text{MPAC} + \mathbf{5}) + (\text{MPAC} + \mathbf{6}) \times 2^{-14}\right]$, and, location MPAC + **2** is not used,

C. 1.3 MEMORY ORGANIZATION, When a multi-precision quantity is stored in memory, the **14** significant bits stored at location K will always be the most significant **14** bits of the quantity. The Interpreter Program Section automatically uses the locations which immediately follow location K (i.e., K + **1,** ---K + **5)** as required to store the lesser significant, **14** bit segments, **A** vector quantity will always have its orthogonal components stored in the order: i, j, k (i.e., i = K **&** K + **1,** j = K + **2** & K + **3,** k = K + **4** & K + **5).** For this reason, whenever an E-Memory location is assigned to a particular, variable, multi-precision quantity, the next-higher-numbered location(s) must be reserved for storage of the quantity's lesser-significant, **14** bit segment(s).

Most of the banks of the computer's Fixed-Switchable Memory and most of the E-Memory, including all of the E-Banks, may be used for interpretive language program sections, interpretive constants and variables. However, some limitations on usage and **bank** switching do exist, For interpretive considerations, Fixed-Switchable Memory can be divided into three parts, The lower part consists of Banks **04** through **17.** Fixed-Fixed Memory (Banks **02** and *03)* and Banks *00* and **01** Fixed-Switchable Memory contain the Interpreter Program Section itself, and are not available for interpretive language program sections or interpretive constants. The two upper (or higher) parts consist of the higher-order banks of Fixed-Switchable Memory (Banks **21** through **27)** and the five, Fixed-Extension Channels (Banks *30* through **77)** respectively. Bank **20 is** not available for interpretive language program sections or interpretive constants.

The Interpreter Program Section cannot switch Fixed-Extension Channels; it is assumed that any interpretive language program section can be contained in the first and second parts or Fixed-Switchable Memory plus one Fixed-Extension Channel (eight banks). Interpretive language program sections may be stored anywhere in the above three parts of Fixed-Switchable Memory that is available to interpretive language program sections, Any interpretive language program section that is located in the first or second part or in any single channel of the third part of Fixed-Switchable Memory may branch to any other interpretive language program section that is not in another Fixed-Extension Channel. However, constants used by a program must be taken from the section containing the program.

Variables may be stored anywhere in erasable memory except the input/output channels and registers $(00)_8$ through $(57)_8$. The Interpreter Program Section cannot switch E-Banks; however, general-erasable memory from location $(60)_8$ through location $(1377)_8$ plus the current E-Bank (referred to *as* "local erasable") are together available for variable storage and temporary storage for calculated results.

C. 1.4 THE PUSHLIST. The Interpreter Program Section provides a set of 38 locations in the Work Area assigned by the Executive that is known as the Pushlist. The name is derived from the term push-down list or a set of storage locations which exhibit a last-in, first-out behaviour (i.e., the last quantity entered into the list is normally the first quantity to be recalled.) Sufficient work-area locations are provided in the Pushlist to allow more than one double-precision, triple-precision, or vector quantity to be stored. If three such quantities are stored before any quantity is recalled, the last quantity entered would be recalled first, the second quantity entered would be recalled next, and the first quantity entered would be recalled last.

The push-down list that is provided by the AGC Interpreter Program Section is peculiar in its design in that any quantity stored anywhere in the list (double-precision, triple-precision, .or vector) may be read out by an indexed address operation without erasing the quantity stored or changing its location with respect to any other quantity stored in the list. This peculiar capability is in addition to, and does not affect the normal capability which effectively "erases" the content of the last location used after recall to make it available for future temporary storage use.

The advantage of the Interpreter Program Section% peculiar type of push-down list is that a quantity may be formed -- for example, a double-precision multiply operation followed by a double-precision add operation -- and then stored in the Pushlist in the normal manner. It may then be recalled as many times as is needed, without erasing it or changing its value or location in the Pushlist as long as it is always recalled by the indexed-address method instead of the normal method. The indexed-address recall provision does not affect or nullify the normal push-down capability (i.e., the last-in, first-out capability) of placing another quantity "on top of" the first quantity, and then recalling the last quantity first.

The manner in which quantities are entered into and recalled from the Pushlist will be discussed in the following section.

C.2 INTERPRETIVE INSTRUCTIONS

C.2.1 ORGANIZATION OF **INTERPRETIVE LANGUAGE PROGRAMMING.** Interpretive language programming differs considerably in format from basic language programming. An interpretive instruction requires a 7-bit order code for expression; whereas, the basic language instruction order code requires **only** three to five bits. It is desirable to employ address codes for constants that **allow** these constants to appear in a different bank from the instruction program routine. It is **also** desirable to **allow** instruction program routines to branch to other routines anywhere **in** Fixed-Switchable Memory. Therefore, the interpretive address code must be a complete address code (a CADR); whereas, the basic language instruction address code **gives** only the sub-address within the current **F-Bank.**

Due to the limited CGC/LGC word length, it is not practical to pack the interpretive instruction order code and its relative address code into one **word** of computer memory. Interpretive instructions are packed in Interpretive Instruction Words (IIW's) with two order codes in bit positions **1** through **7** and **8** through **14,** respectively. Addresses for constants and branch addresses are given a full computer memory word; whereas, the address of a variable in local erasable memory only requires **10** bits. Instructions which store the contents of **an** accumulator are generally the **only** ones whose addressing capability may be restricted to local erasable memory; therefore, these instructions contain **an** E-Memory address in bit positions **1** through **10,** and the instruction order code in bit positions **11** through **14.**

An IIW which contains two interpretive instruction order codes will normally be followed in an interpretive program listing with two Interpretive Address Words (IAW's) which contain the relative addresses for the two interpretive instructions. If the second interpretive instruction order code (bit positions 8 through **14)** is a branch instruction, the second **IAW** below will be a branch address. Some interpretive instructions such as function codes (for sine, cosine, square root, etc. functions) and some shift instructions (shift right one, shift left two, etc.) are unary in nature and require no IAW. (All function and shift instructions operate exclusively on an accumulator.)

When more than one mathematical execution is involved in solving an equation, the IIW's which program the executions, together with their relative IAW's, are formed into an Interpretive Program String. The first interpretive instruction order code in a program string will always be a LOAD Instruction to "load" the interpretive accumulator (MPAC). This instruction also sets the mode of operation **(D.P., T.P.,** or Vector Mode) of the Interpreter for the program string. When program control is transferred to the Interpreter, program operation extracts one IIW at a time from the Interpretive Program String, decodes the order code(s), then mates each interpretive instruction with its relative CADR. The CADR(s) is/are obtained from the IAW(s) immediately below the IIW in the program string. Each interpretive instruction is then executed by the Interpreter before another IIW is extracted from the program string. Interpretive Program Strings are normally terminated by a **STORE** Instruction which stores the resultant contents of the accumulator **(MPAC)** at some designated E-Memory storage location, The number of successive locations used to store the result is determined by the mode of operation for the program string.

C.2.2 INTERPRETIVE INSTRUCTION **ADDRESSING** CLASSIFICATIONS. **Two** index registers, X1, and **X2,** are provided in the Work Area that is assigned **by** the Executive for addressing modification of IAW's under Interpreter program control. When the interpretive instruction order code specifies that a given IAW is to be indexed, the IAW will be stored complemented for Index Register **2,** and stored normally if Index Register **1** is to be used. The indexed operand address is obtained by subtracting the content **of** the specified index register from the address specified in the IAW. In addition to the index registers, two step registers , S1 and S2, are provided for reducing the contents of the two index registers by specified "step-function" amounts. **A** set of interpretive instruction order codes is provided exclusively for manipulating the contents of the index registers and for counting and branching on the contents of the index registers **by** using the step registers.

The 7-bit order code which characterizes the interpretive instruction allows the general code set to be divided into four addressing classes. The **2** least-significant bits of the 7-bit code are used to specify the addressing class for the instruction:

Class Code	Addressing Class
00	Unary instructions which require no address (i.e., act **only** on an accumulator). Included in this class are the function order codes and some shift instructions.
01	Arithmetic instructions with non-indexed addresses. Address may be in any location of "local erasable" or any location of the part of Fixed-Switchable Memory from which the instruction was taken.
11	Arithmetic instructions with indexed addresses. Content of an index register is subtracted from the given address to form the net operand address. Address location limitations of Class **01** apply.
10	Branch instructions and index register instructions. **This** set of instructions is used to perform sequence changes and to modify the contents **of** an index register. Address may be any location of "local erasable" or any location in any part of locally available, Fixed-switchable Memory.

C.2.3 CLASS 00 INSTRUCTIONS FUNCTION **ORDER CODES** AND UNARY **SHIFT** INSTRUCTIONS, The function order codes that are recognized by the interpreter program section include nine scales and five vector functions. A complete listing **of** these functions and other interpretive instructions appear in the appendix of this study guide. Each of these instructions operates on the contents of MPAC and therefore require no Interpretive Address Word in the interpretive string.

A total of thirty-two unary shift instructions are provided for the interpretive scaling adjustments that are frequently required for fixed-point computation. These shift instructions require no address because the individual order codes specify fixed, short-length shifts. The first 16, "short-shift" order codes provide: Scaler Shift Right, one to four places; Scaler Shift Left, one to four places; Scaler Shift Right And Round (to DP), one to four places; and, Scaler Shift Left And Round (to **DP),** one to four places. The remaining sixteen "short-shift" order codes provide Vector Shift Right And Round (to DP), one to eight places; and, Vector Shift Left (without round), one to eight places.

C. 2. 4 CLASSES 01 AND 11 INSTRUCTIONS, LOAD AND **GENERAL** SHIFT INSTRUCTIONS. These instructions form a special group of interpretive instructions which operate only on the contents of MPAC. The instructions **do,** however, require either a direct address (class 01) or an indexed address **(class 11)** to load the MPAC for an arithmetic or shifting operation. Twelve store or load instructions, seven general shift instructions, nine scaler arithmetic operations and ten vector arithmetic operations are included in these classes of instructions.

C. **2.5** CLASS **10** INSTRUCTIONS, **PROGRAM** MECHANIZATION INSTRUCTIONS. The program mechanization instructions are used for branching, sequence changing, **sub-**routine linkage and for loading, storing and modifying the contents of the two index registers (X1, X2). These instructions must be followed **by** an address word to indicate where to branch to etc. They do not affect the contents of MPAC. Forty instructions fall into this instruction class including fourteen used for sequence changing, branching and subrouting linkage under general interpretive language program control.

A second sub-group of fourteen instructions are used for setting, resetting, inverting, branching and testing a set of sixty-twovalued indicators called Interpretive switches. These switches are located in four consecutive E-Memory locations called STATE, STATE +1, STATE +2, and STATE +3 with fifteen switches per location. Each instruction requires an address word defining which switch is to be operated on.

A third sub-group of class 10 instructions include 10_{10} instructions to manipulate the contents of the two index registers X1, and **X2.**

There are two additional program mechanization instructions with order codes that allow them to fall into the class 00. These instructions are **STADR** (recognize store code) and **SSP** (Set Single Precision).

A complete description of the interpretive instructions are included in the appendix.

C. **3** INTERPRETER PROGRAM OPERATION

C. 3.1 NODAL ANALYSIS OF GENERALIZED PARENTHETICAL EXPRESSIONS. The format for the programmed solution of an equation which requires several Interpretive Program Strings and which uses the Pushlist can best be illustrated in the form of an

example. The equation to be programmed is the solution for the sum and difference roots of the familiar, second-order, quadratic equation:

$$ax^2 + bx + c = 0$$

The equation for the roots is:

$$x \text{ (root 1)},\ x \text{ (root 2)} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

$$= \frac{-\frac{b}{2} \pm \sqrt{(b/2)^2 - ac}}{a}$$

An example of the interpretive programming techniques can beet be shown by a simplified flow-gram of the computer operations necessary to performed. Figure C-1 illustrates the symbols used in an Interpretive flowgram. Use of a temporary storage location, pushlist, is implied in this operation.

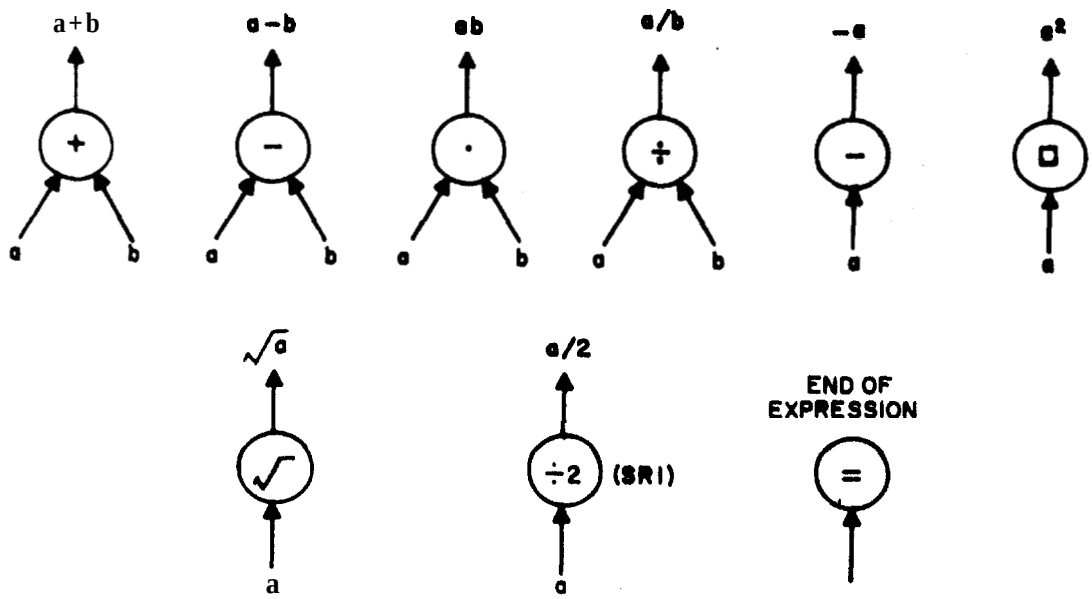


Figure C-1. Network Mapping Symbols

Each node is limited to one or two input links (dependent upon the type of operation), but, each node may have as many output links as required. The topographical network for the interpretive routine for the roots equation is shown in Figure C-2.

A complete expression is directly computable if, and only if, for every two-input node, at least one of the input links comes from a stored operand or a temporarily-stored, partial result. Therefore, for every two-input node in which both inputs are the result of mathematical operations represented by other nodes, one of the partial results must have been temporarily stored before the order code represented by the two-input node is executed.

If any operation node has more than one output link, the result of the mathematical operation represented by that node must be temporarily stored before it is used as an input *to* the first node which uses it. In the symbols used as two-input nodes, it is assumed that the input on the right side is always added to, subtracted from, divided into, etc. , the input on the left side.

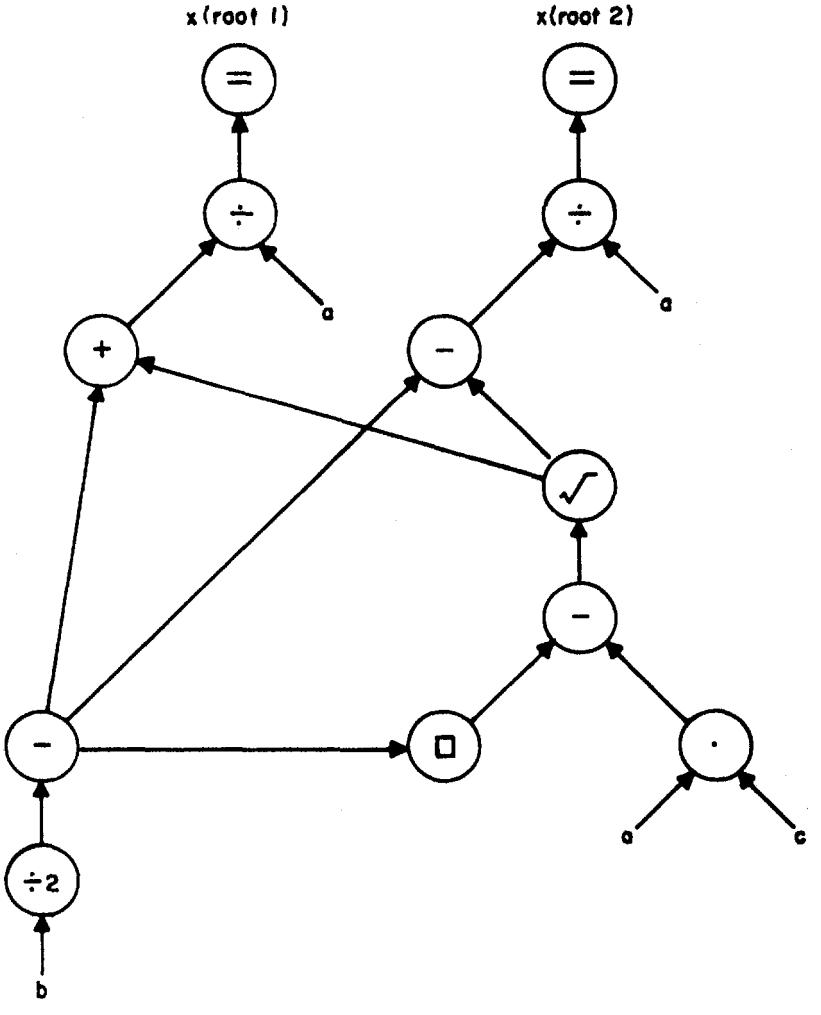


Figure C-2. Interpretive Routing Flow-gram

An interpretive language program routine may be written by starting at the lower-most node on the left side of a network and by proceeding up the linkages of the network toward the end-of-expression symbol. When a two-input node is reached whose other input comes from another node, the present partial result must be temporarily stored, and the other input treated as a sub-network and synthesized in the same manner. When the original two-input node is reached, the partial result is recalled and the synthesis continues up the linkages toward the end-of-expression symbol. The interpretive language program routine is as follows:

```
ROOTS   TC      INTPRET     (basic language)
        DLOAD   SR1         (divide by 2)
                BOPADR
        DCOMP   PUSH        (positions 0, 1)
        DSQ     PDDL        (positions 2, 3)
                AOPADR
        DMP     BDSU        (recalls b 2/4 from 2, 3)
                COPADR
        SQRT    PUSH        (positions 2, 3)
        DAD     DDV
                00000       (index on Pushlist)
                AOPADR
        STODL   ROOT1ADR
        BDSU
        DDV
                AOPADR
        STORE   ROOT2ADR
        EXIT                (return to basic
                             language)
```

In the five Interpretive Program Strings which program the equation for the roots in double precision, the operand address for the operand (a) is assumed to be **AOPADR;** the location ROOTlADR is assumed to be the E-Memory storage location for the first root, x (root 1), and the location ROOT2ADR is used to store the second root, **x** (root **2).** A special feature of the PUSH Instruction is that the pushed-down quantity is retained in the accumulator (MPAC); therefore, it can be temporarily stored for future use without disrupting the present computational flow. The STORE Instruction also retains the stored quantity in MPAC. The address word ***00000*** implicitly refers to the work-area assigned by the Executive; and, it is, by definition, Position Number **1** (DP) in the Pushlist. This is the indexed-address method of referring to the Pushlist that was described in paragraph **2.1.4.** One additional feature **of** the **PDDL** and **PDVL** Instructions is that they can be used to effectively exchange the contents of the accumulator (MPAC) with the last quantity previously entered into the Pushlist. If no operand address is provided for the load portion of these instructions, the Interpreter will extract the last quantity previously entered into the Pushlist, and then will temporarily store the present contents of MPAC in the location vacated by the quantity that was extracted for loading into MPAC. The STODL and STOVL Instructions will **also** extract the last quantity previously entered into the Pushlist if no operand address is provided for the load portion of the instructions.

C. **3.2** THE DISPATCHER ROUTINE - PROGRAM OPERATION. For the purpose of illustration, the Interpreter can be considered to be equivalent to a program-level, sequence generator. The Sequence Generator in the AGC's Control Section consists of an SQ Decoder (command generator) and a Control Pulse Generator. The AGC's Sequence Generator depends on the Central Processor to load an instruction order code from a programmed list of basic language instructions into the *SQ* Register (buffer), The Sequence Generator then decodes this order code and generates a sequence of control pulses which execute the instruction designated by the order code. The method by which the Interpreter decodes and then executes the interpretive language instructions differs from the Sequence Generator in only two points:

a. The Interpreter is a program section in the AGC's Fixed-Memory, and therefore, all interpretive decoding and execution is under basic language program control.

b, The Interpreter uses basic language program control to retrieve interpretive language order codes from memory, and therefore, it does not depend on any external source (such as the Central Processor in the analogy) to obtain each interpretive order code.

The Interpreter Program Section is made up of two major routines called the Dispatcher and the Executer. In the above analogy, the command generator *(SQ* Decoder) can be compared to the Dispatcher portion of the Interpreter. The Dispatcher Routine is made up of several minor routines and a number of subroutines. These routines decode the interpretive language order code(s) in each Interpretive Instruction **Word** (IIW), and also, decode the relative interpretive address(es) and/or constant(s) in the relative Tnterpretive Address Word(s) (IAW's). After taking care of the necessary internal "housekeeping" such as setting up the interpretive mode of operation, etc., the Dispatcher Routine stores the relative address in its decoded form in a location that **is** known to the Executer Routine. It then transfers Interpreter program control to the subroutine in the Executer that is commanded by the decoded, interpretive language order code.

C.3.2.1 THE INTERPRETIVE PROGRAM. The interpretive program section always uses the Interpretive Dispatcher to determine, is the word an instruction or a store code word? what class of instruction code is it? and which of the many subroutines shall be selected? The Executer commands the specific subroutine called to be performed and returns operation to the Dispatcher. The Dispatcher operation requires entry by an instruction TC INTPRET each time an interpretive instruction string is begun.

C.3.2.2 DISPATCHER FLOW. (See figure C-3.)

(1) Whenever the interpretive language on mode of the computer is selected the contents of register (Q) is stored in a temporary storage location LOC. *Q* contains the address of the instruction following TC INTPRET. The contents **of FBANK** is stored in BANKSET register enabling the program to return to the **bank** which requested INTPRET. Bit 15 of FBANK *is* stored to indicate which half of fixed memory is accessable to the interpreter.

(2) The contents of the address defined in LOC is checked to determine **if** the contents are an instruction code pair or an interpretive store code, For the later the operation is transferred to a store operation.

(3) If an instruction word is detected **by** the above test an edit operation is performed. The high seven bits of the instruction code is shifted into storage location EDOP. The low seven bits of the word is loaded into a location designated CYR. **This** transfer operation shifts bits **7** through **2** of the initial code to bits **6** through **1** of **CYR** and the original bit one of the OP code is shifted into bit **15** of CYR.

(4) Bit **15** of CYR is checked to determine if the content is a one or zero. This is one step in determining the class of the interpretive instruction. **A** one in bit **15** defines a class 01 or **11** instruction.

(5) Check the contents of bit **1** of CYR (this was originally bit **2** of the instruction OP code). If bit **1 = 1** a class **11** instruction an index operation **is** permitted in **which** the contents of the INDEX register is subtracted from the given address to form the net operand address. A TC to INDEX is commanded.

If bit 1 of CYR is '0' a class 01 address is defined. The address defines locations in any location in "local erasable" or any location in the half of Fixed-Switchable memory selected. A subroutine DIRADRES is selected eliminating the indexing operation.

(6) & **(7)** If bit **15** of CYR is a '0' when checked in **(4)** above, the contents of CYR are checked again to see if the code is all zeros or a positive quantity. If all zeros an automatic exit is commanded. If a positive quantity is detected **in** CYR **a TC** to OPJUMP3 is commanded. Class 00 and 10 addresses are defined when bit **15** is '0' and the contents of CYR is positive.

(8) We have identified an arithmetic operation with a non-indexed address location - class 01 address - to this point. Now clear and subtract the contents of the address location LOC + **1** and *see* if bit 15 is a '1' or '0'. If the complemented contents

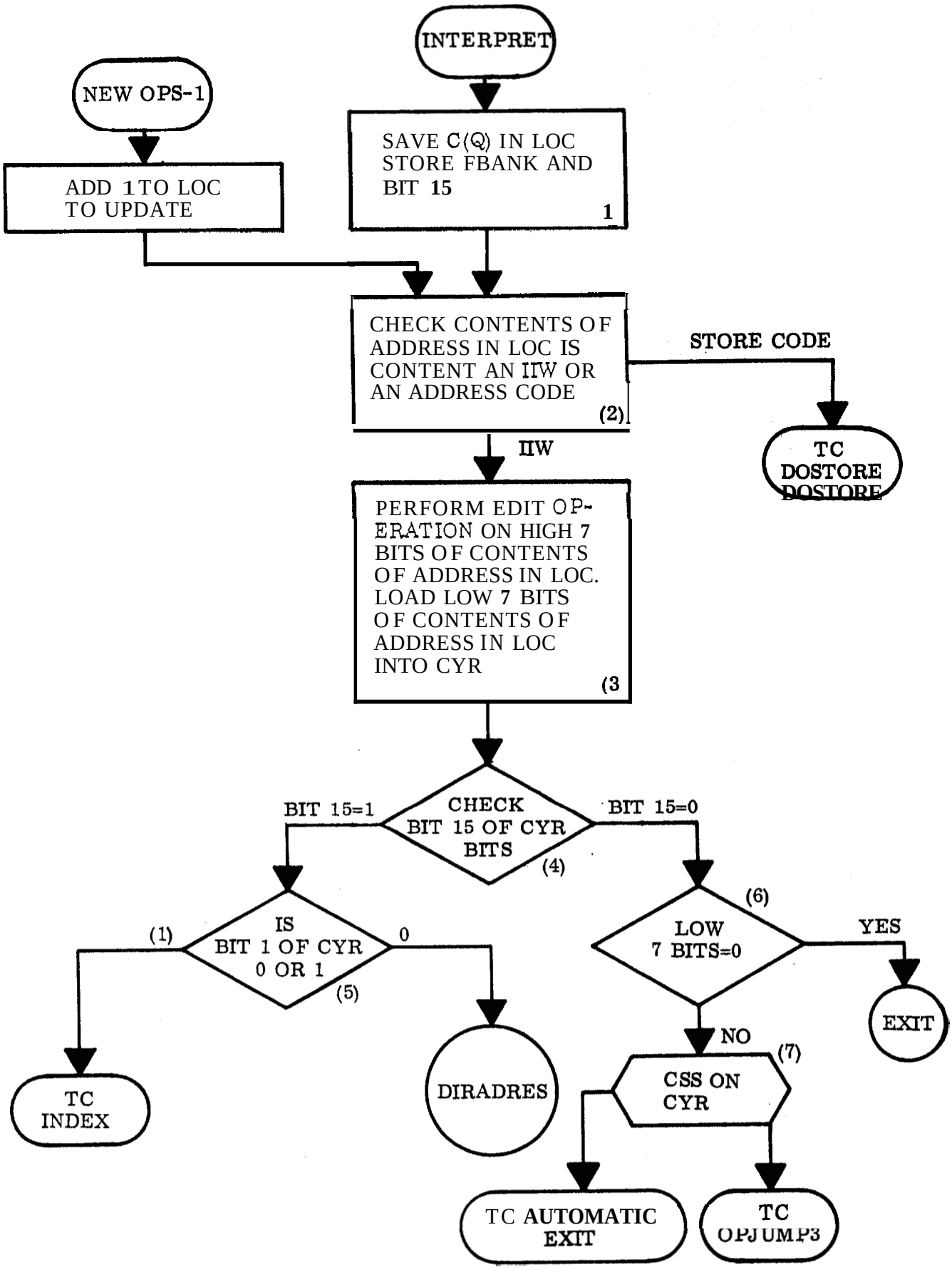


Figure C-3. Interpretive Program Flow (Sheet 1 of 14)

indicate PNZ TC to PUSHUP where the mode of operation, double precision, triple precision or Vector, for the interpreter routine **is** defined. A **NNZ** content defined an address in erasable or fixed memory but not in the PUSHLIST.

(9) If an address is defined in **(8)** the content of LOC and the ADDRWD are incremented to update for the next operation.

(10, **11, & 12)** Where is the address location is the next question asked? Is it in the work area $- < 45_{10}$, in general erasable $45_{10} <$ address $< 977_{10}$ or is it in Fixed? If it is in the work area the specific work area is selected by updating the address word with the contents of FMLOC. FIXLOC contains the address **of** the work area selected.

If the address is in fixed memory the FBANK register is loaded with the new fixed bank number. 2000_8 is added to the contents of ADDRWD making the contents **of** ADDRWD lie between 2000_8 and **3777,** the addresses used to define a memory location in a specified FBANK . At this point the ADDRWD and FBANK, if needed, define the complete address of the constant or variable to be operated on by the interpretive instruction defined by the code in CYR.

(13) After the location of the constant to be operated on is defined a TC instruction is generated by indexing the code in CYR with the address of a jump table INDJUMP to select the subroutine to be performed, These subroutines include all the instructions listed as class 01 or 11 in the interpretive instruction list.

(14) & (15) If an index address code (class **11)** is recognized, the **INDEXLOC** register is loaded with the address of the work area (FMLOC) for indexing operations. The content of LOC is incremented to define a new address for the next following operation. The content of this new address is complemented and loaded into the accumulator. A CCS on the content of address defined by LOC is performed. If a PNZ quantity **is** detected the INDEXLOC register is incremented, If a PZ or a NNZ quantity is detected the contents of a (low **14** bits) are transferred to the ADDRWD register. If **a** NZ quantity is detected the operation is continued in block **(16).**

(16) The contents of the accumulator are anded with 7600_8 to mask out the bank number of the address in the accumulator. If a check indicates a non-zero quantity in the bank (upper **4** bits) portions of the address in **A,** the original value of bit **15** at TC INTPRET must be recalled. If a zero is detected a TC to INDEX2 **is** generated.

(17) A non-zero bank number from (16) above causes the program to pick up the most significant bit (Bit **15)** of the original Bank number when a TC INTPRET was generated. This bit is added to the ADDRWD giving a full **15** bit address containing bank number and address. Bit **15** will define which half of the fixed memory **banks** are being addressed.

(18) & (19) By the use of an index operation subtract the contents of INDEXLOC + **X1** (Index 1 register) from the ADDRU'D. A new ADDRWD (Indexed) is obtained. **A MASK** operation is performed on ADDRWD to determine the relative location of the address defined. If a non-zero quantity is detected the address **is** in fixed memory or general erasable memory. If a ± zero is detected the work area erasable memory address is

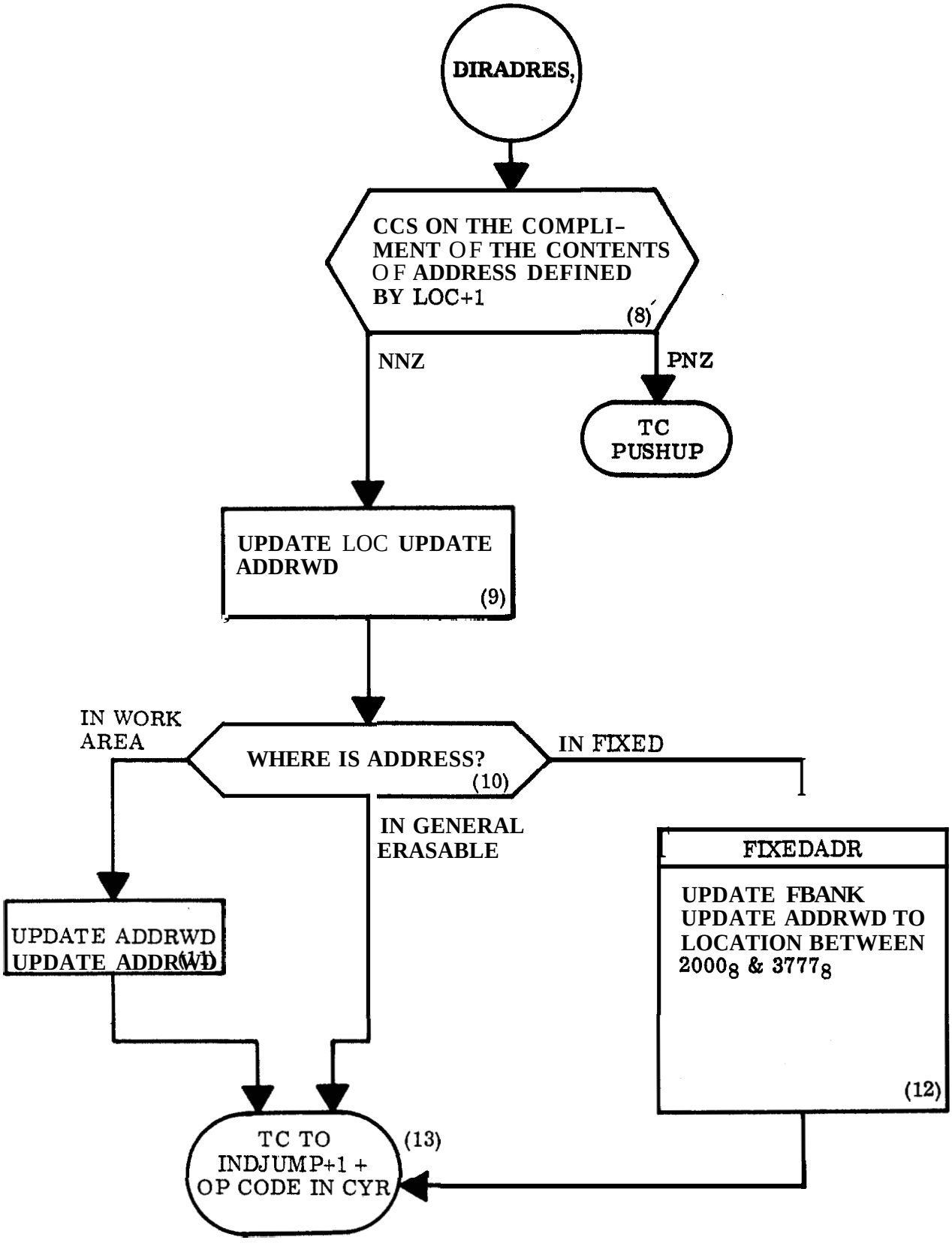


Figure C-3. Interpretive Program Flow (Sheet 2 of 14)

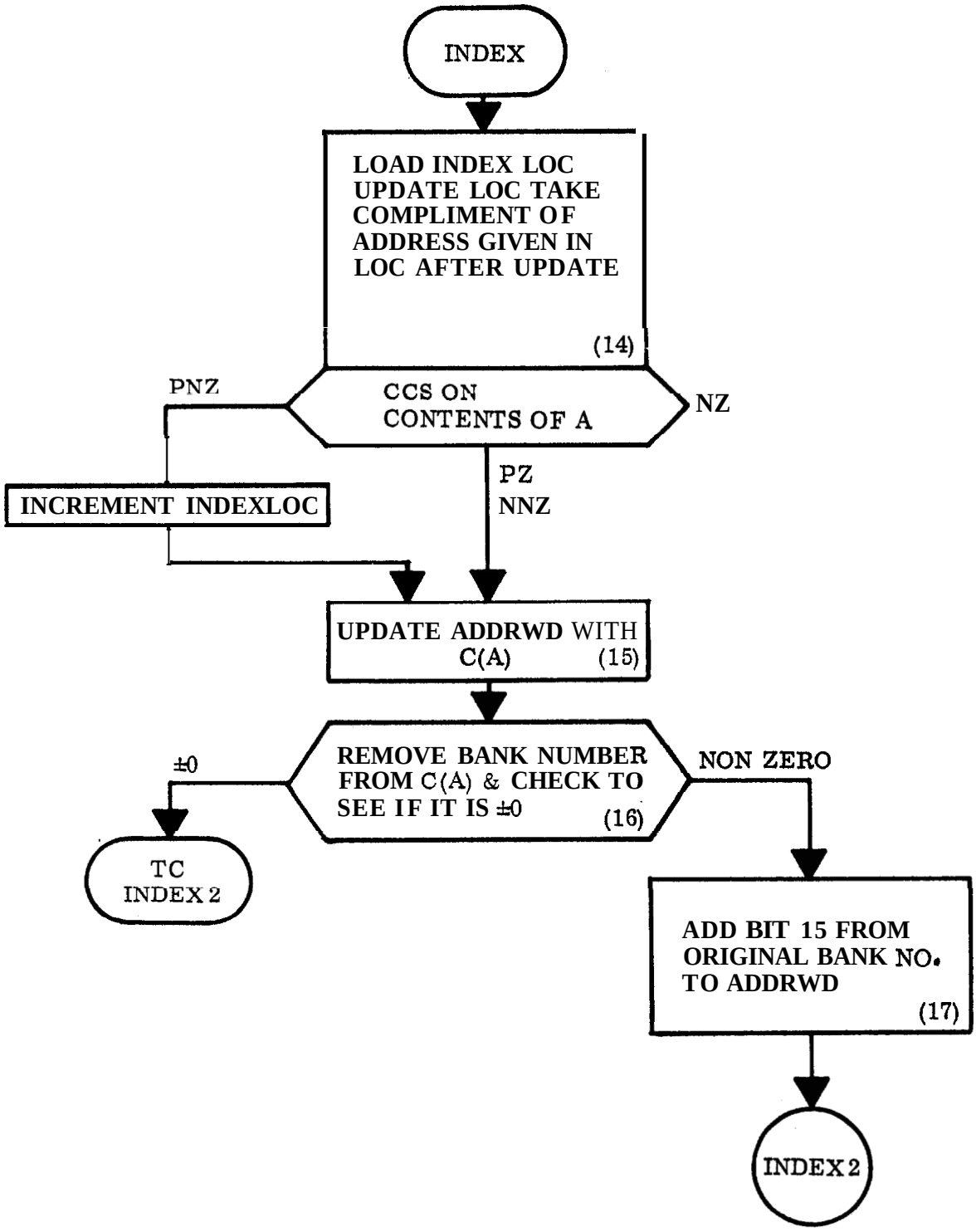


Figure C-3. **Interpretive Program Flow (Sheet 3 of 14)**

modified, as before, by adding the address of FMLOC (work area address) to ADDRWD to obtain the address of the specific work area to be used.

(20) & (22) If the check in block **18** indicated a NON Zero a further check is made on ADDRWD to determine if the address is in general erasable or in fixed memory. If the address indicates a fixed memory location the high order bits of ADDRWD are loaded into FBANK and 2000_8 is added to ADDRWD and 01777_8 to make *the* contents of ADDRWD fall between 2000_8 and 3777_8, the addresses associated with a fixed bank. A complete fixed memory location is now defined.

(21) & (23) If the address is in general erasable from above check, if in the work area **(18, 19)** and after defining a fixed memory location **(20, 22)** an index operation is performed. The contents of CYR - the applicable OP code is added to the address of INDJUMP to identify the subroutine for a specific interpretive instruction to be performed on the content of the address defined in blocks **18, 19, 20,** and **22.** Block **13** is again repeated as a TC operation.

(24) The operation which allowed a TC to **OPJUMP 3** was a **CCS** on **CYR.** This operation would leave the absolute value of the contents decremented by **1** in the accumulator. This value is used to update the Fixed Bank register. The OPJUMP **3** subroutine is used whenever a class OO–Addressing code - (Unary instruction) is found in CYR.

(25, 26, & 28) Following the loading of FBANK register a CCS on CYR is again performed. If the path into OPJUMP **3** was from TC INTPRET **CYR will** contain a PNZ quantity however the possibility exists of entry into this subroutine from another point in the program. If a positive non–zero is found the contents of CYR is added to the address of UNAJUMP to select a subroutine falling into the UNARY instruction group (no address required perform operation on contents of MPAC, **VAC,** etc.). Several instructions included are: SQRT, SINE, UNIT.

If a positive zero (PZ) is found in CYR the previous CCS on CYR should have caused the program to exit. If a negative zero (NZ) is found a **TC to SHORT** T - **short** shift operation subroutine is generated.

(27) When a negative non–zero quantity is found in CYR a check is made on the MODE register. The MODE register defines the interpreter mode, double precision, triple precision or vector. If a double or triple precision made is selected control is transferred to a short shift scalar operation subroutine SHORT T. If the vector mode is used to subroutine SHORT V - Vector Shift is selected.

(29) If the check on the contents of the address contained in **LOC + 1** indicate that the content of the PUSHDOWN LIST (Block 8) is to be used as the operand for the arithmetic or vector operation defined in CYR we end up in this routine (PUSHUP). The first operation in this routine is check the contents of CYR. If CYR contains a code 20_8, 21_8, 22_8, or 23_8 a normal usage of the contents of the PUSHDOWN list is signified. A TC to REGUP is generated. If, however, the contents of **CYR** are 0, **1, 2,** or **3** several possible variations in the use of PUSHLIST contents are possible. A further check on the contents of CYR is performed (Block **32).**

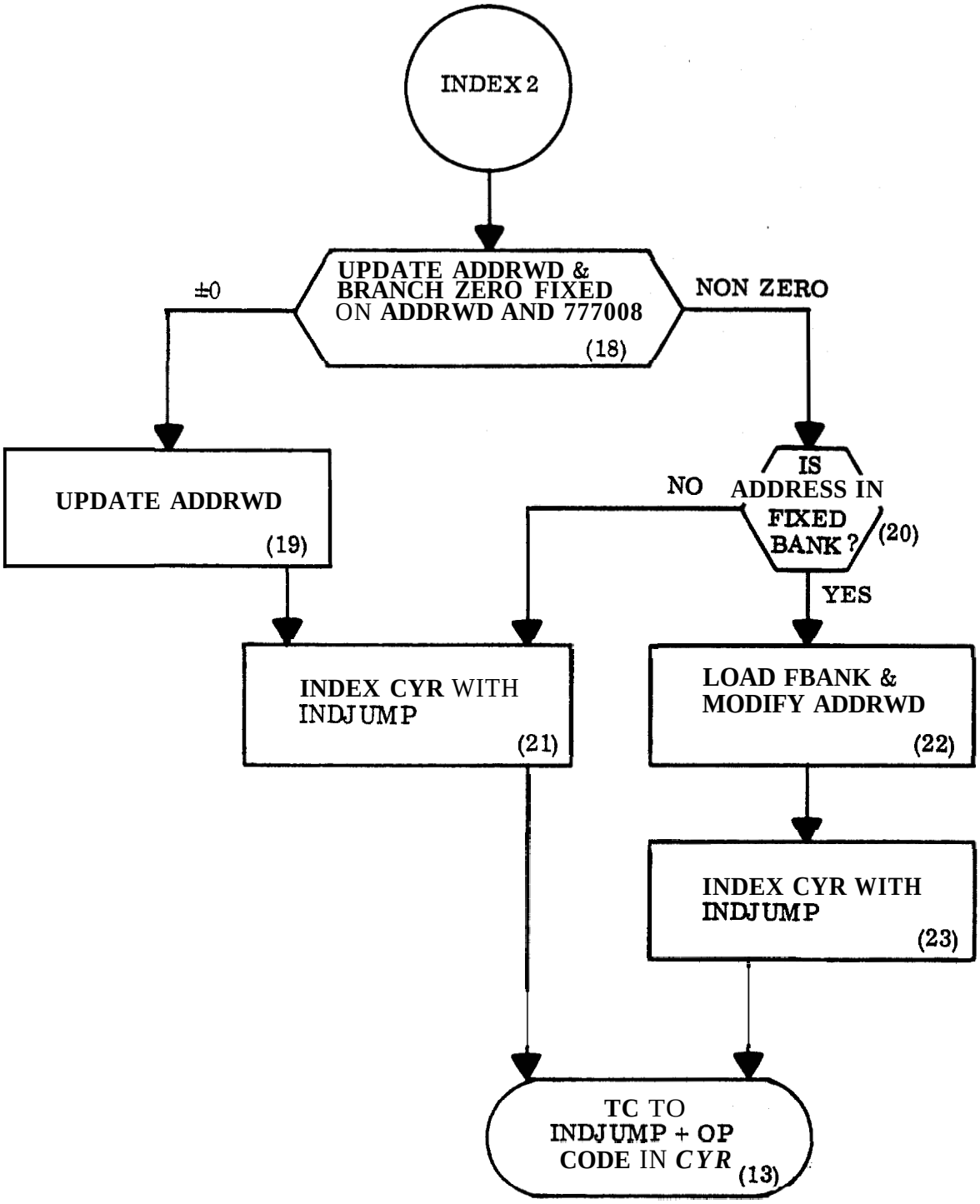


Figure C-3. **Interpretive Program Flow (Sheet 4 of 14)**

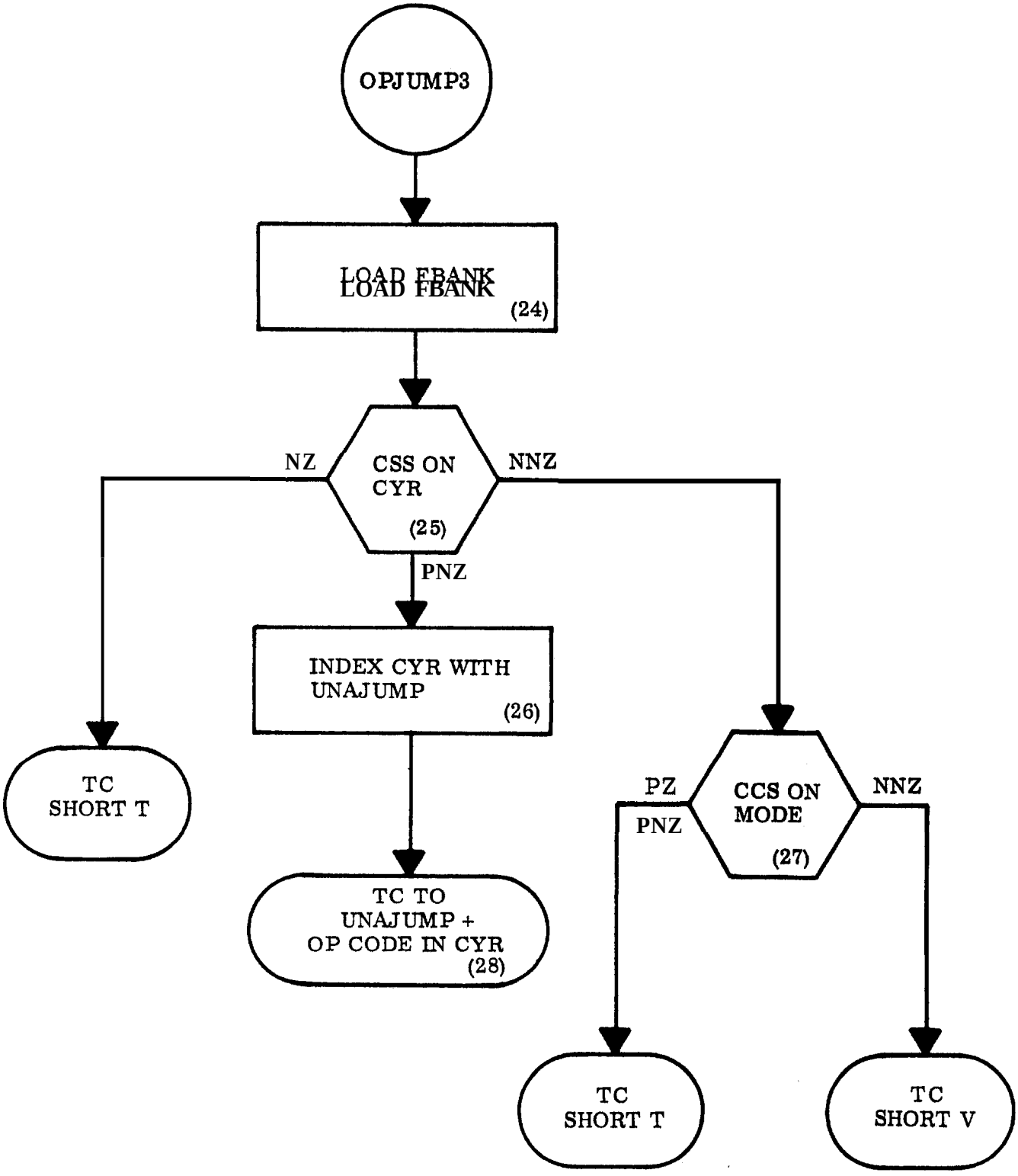


Figure C-3. Interpretive Program Flow **(Sheet 5 of 14)**

(30) Depending on the MODE double precision, triple precision, or vector which was determined elsewhere number **of** words in pushlist **2, 3, or 6,** to be used is entered into the accumulator.

(31) The accumulator contents, **-2, -3,** or **-6** are added to **PUSHLOC** to define a specific address location. This same information is stored in **ADDRWD** to provide an address for the next operation.

(32, 33, 34, & 35) The contents of the accumulator, which will be a **7, 6, 5, or 4** following the previous CCS (Block **29)** are added to a constant negative **4.** A check (CCS) is then made on the contents of the accumulator. The contents **of** the accumulator after the addition will **be** negative zero (4), positive **zero** (5), positive **one** (6), or positive two **(7).** Three possible types of operations are possible at this time:

a. An arithmetic operation which requires a standard type **of** operand regardless of the previous operation (standard operand). If this type **of** operation is defined a clear and subtract **2** (NOWORDS) is commanded for a double precision operation.

b. The second type of operation possible using the **PUSHLIST** is an operation in which the accumulator is loaded with a load code which is independent **of** the previous operation. Examples **of** these codes are **VLOAD, DLOAD, TLOAD,** PDDL, and PDVL. If this type of operation is selected by the **OP** code in **CYR** the accumulator, which contains a plus 1 or plus **2** is indexed with **NOWORDS** to select the double, triple or vector mode of operation.

c. The third type of operation using the **PUSHLIST** as an address location for an operand is a reversing operation. If the last operation yielded a vector result the next operand should be a scaler. VXSC is an example **of** the operation requiring a reversal of this type. **If** this type of operation is commanded the contents of the MODE (0, ± **1)** register are indexed with REVCNT to select the reversed conditions from the previous operation.

In all cases after completion of the selection of double, triple, **or** vector operation for selecting data from the PUSHLIST a TC to REGUP + **2 2 is** commanded returning the operation to block (30) above.

(36) In the event a store code was detected in the operation in Block (2), the type of store operation must be selected. ***The*** contents of the accumulator are transferred to storage as the ADDRESS WORD. The data **in** the accumulator **is** anded with a constant to pull out the local erasable memory locations, bits **1** through 10. The 10 bit address code is stored in the ADDRESS WORD and the complete address word (old one) is put back into the accumulator.

(37) If the store code was recognized the exact operation to be used in the storage routine must be defined. The high order bits, bits **11** through **14** of the contents **of** LOC + 1 are masked out, The resultant is multiplied by 000400_8 ***to*** shift the contents of the accumulator from bits 11→14 to bits 1→5. This enables the programmer to use an index type operation to select a proper storage routine.

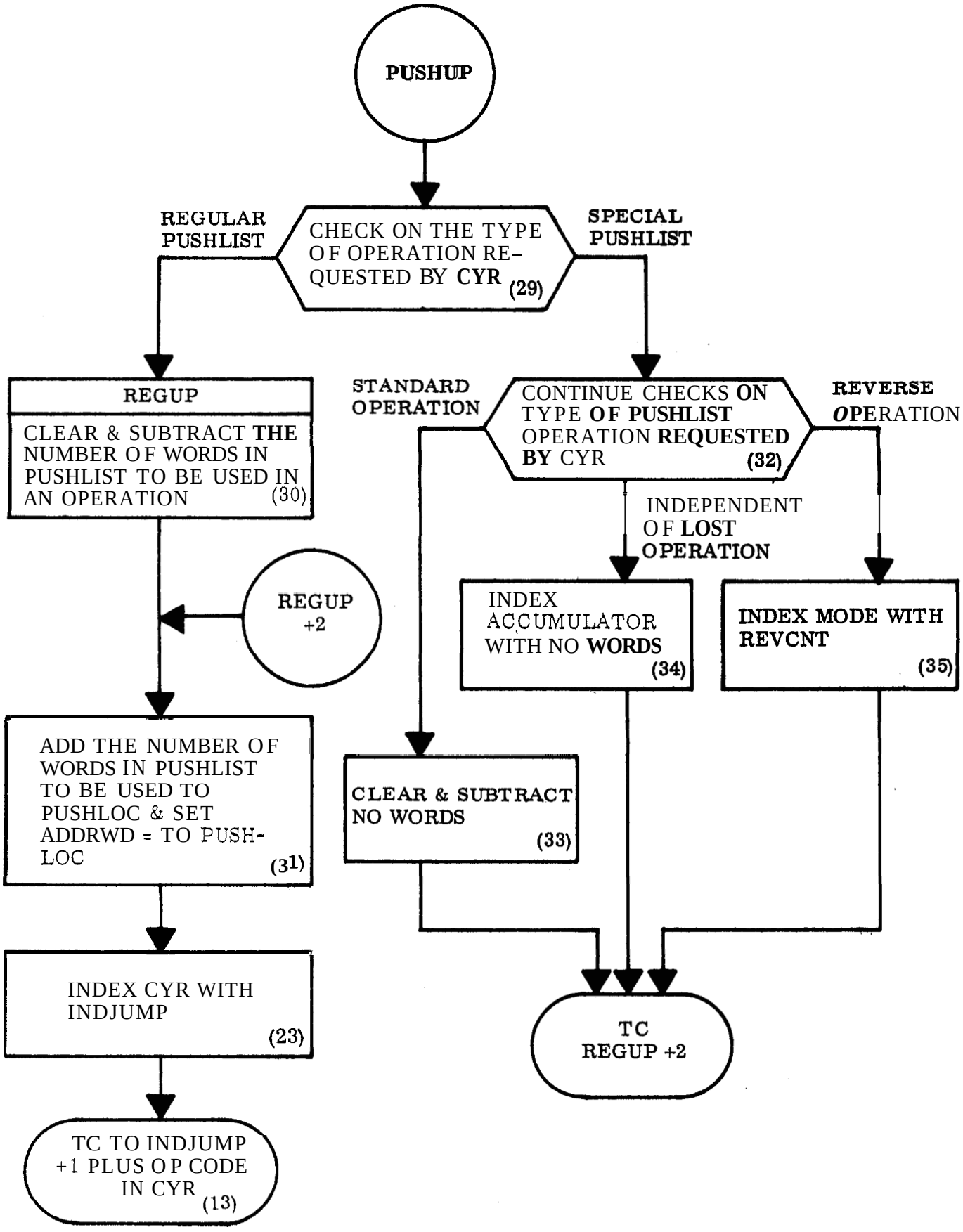


Figure C-3. Interpretive **Program** Flow **(Sheet *6* of 14)**

(38) The contents of the accumulator **00** to 32_{10} are indexed **with** a TC STOREJUMP to generate a store subroutine TC instruction which will in turn select one of the TC STORE commands in the STOREJUMP table.

(39) If the code is found in the high order bits of the old address word it is recognized as a command to update CYR with **a** CALLCODE. The CALLCODE register will contain a new OP code to be used in CYR in following operations.

(40, 41, 42, & 43) If the TC STOREJUMP transfers control to STORE **1** the address contained in ADDRWD is first updated with the contents of FIXLOC which contains the address of the work area selected. Block **41** identifies a work area address or a general erasable address for storage. If a work area is defined **FIXLOC** is again used to update the ADDRWD. If a general erasable address **is** found in ADDRWD a storage operation is performed (Block **43).** The storage operation will transfer the data contained in MPAC and MPAC + **1** into the erasable memory locations defined by ADDRWD and ADDRWD + **1. A** double precision transfer to storage **is** thus performed using ADDRWD to define the location used for storage.

(44) After the initial transfer to storage of the double precision quantities a check is made to see if the quantity to be stored is **a** double or triple precision quantity or **a** vector quantity by looking at a MODE indicator. The MODE indicator will contain a zero for double precision, a plus one for triple precision or a minus one for vector operations.

(45) If a plus one was found in MODE the third component of the triple precision quantity, which is located in MPAC + **2** is transferred to the erasable location defined in ADDRWD + **2.** Operation is picked up in the DODLOAD subroutine. Triple precision data is stored at this time.

(46) If the Vector mode of operation is detected the other two double precision components of the vector, the Y & **Z** components, must be stored. The 2nd and 3rd components of the vector are taken from MPAC + **3, 4,** and MPAC + **5, *6*.** The contents of these four locations are transferred to the storage locations defined by ADDRWD + **2, 3, 4, 5.** MPAC + **2** does not contain a usable quantity as **it is** not used during vector mode operations.

(47) If the MODE indicator contained an indication of double precision or the vector quantity *is* stored, a transfer control to Q is commanded. The contents of ***Q*** are the TC instruction of the next following address which commanded the STORE subroutine to be selected. The operations commanded are shown in the illustration.

(48) & **(49)** If a TC DODLOAD, or DOVLOAD the contents of CYR is updated with **a** new OP code to command a double precision or vector load operation followed to a TC to DIRADRES. This indicates that a class 01 instruction address is to follow.

(50, **51,** & 52) If a TC to DOVLOAD* or DODLOAD* is commanded **(47** above) CYR is again updated with an OP code which will call an address class **11** which indicates that an indexing register operation is permitted for an arithmetic operation.

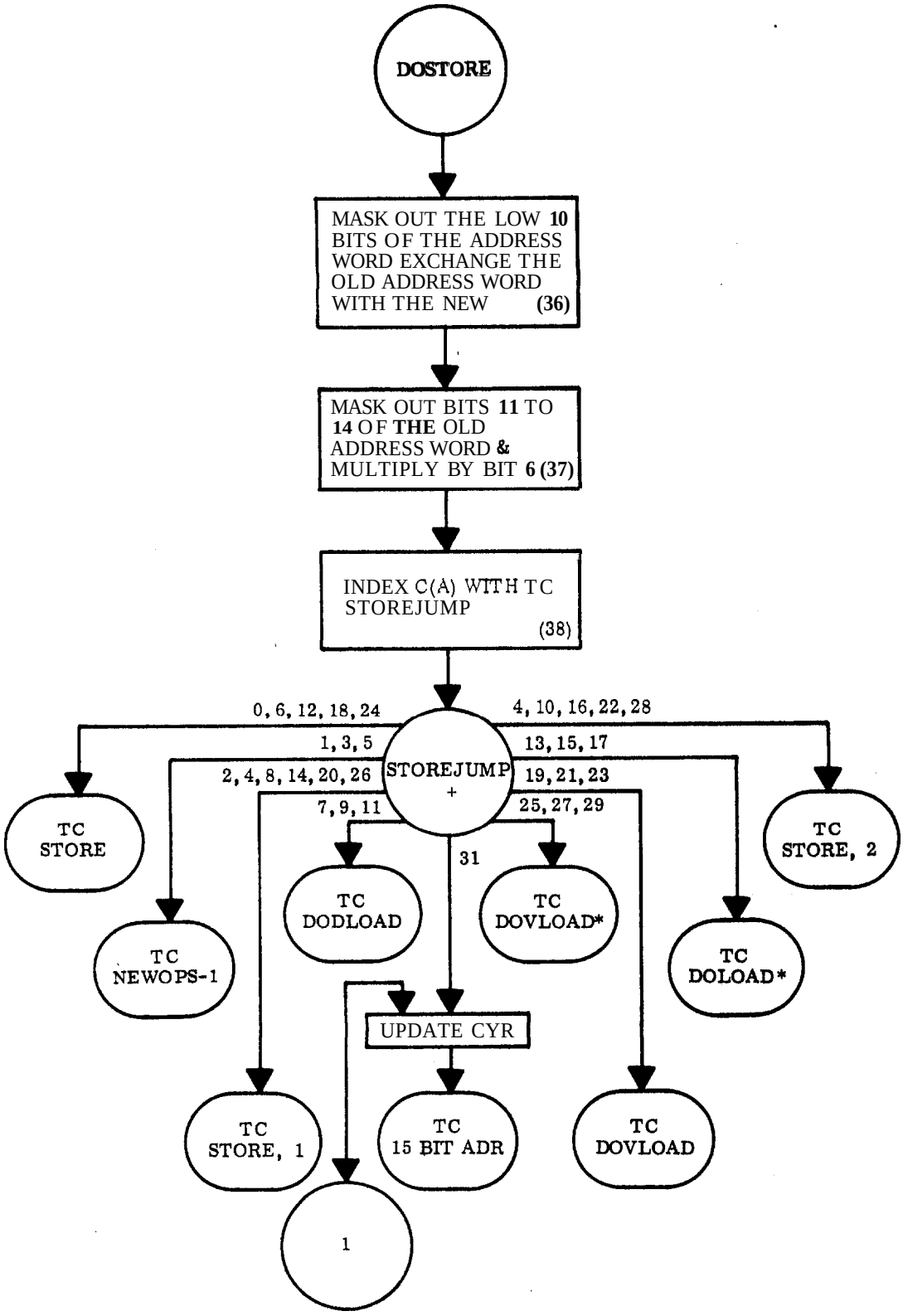


Figure C-3. Interpretive Program Flow (Sheet 7 of 14)

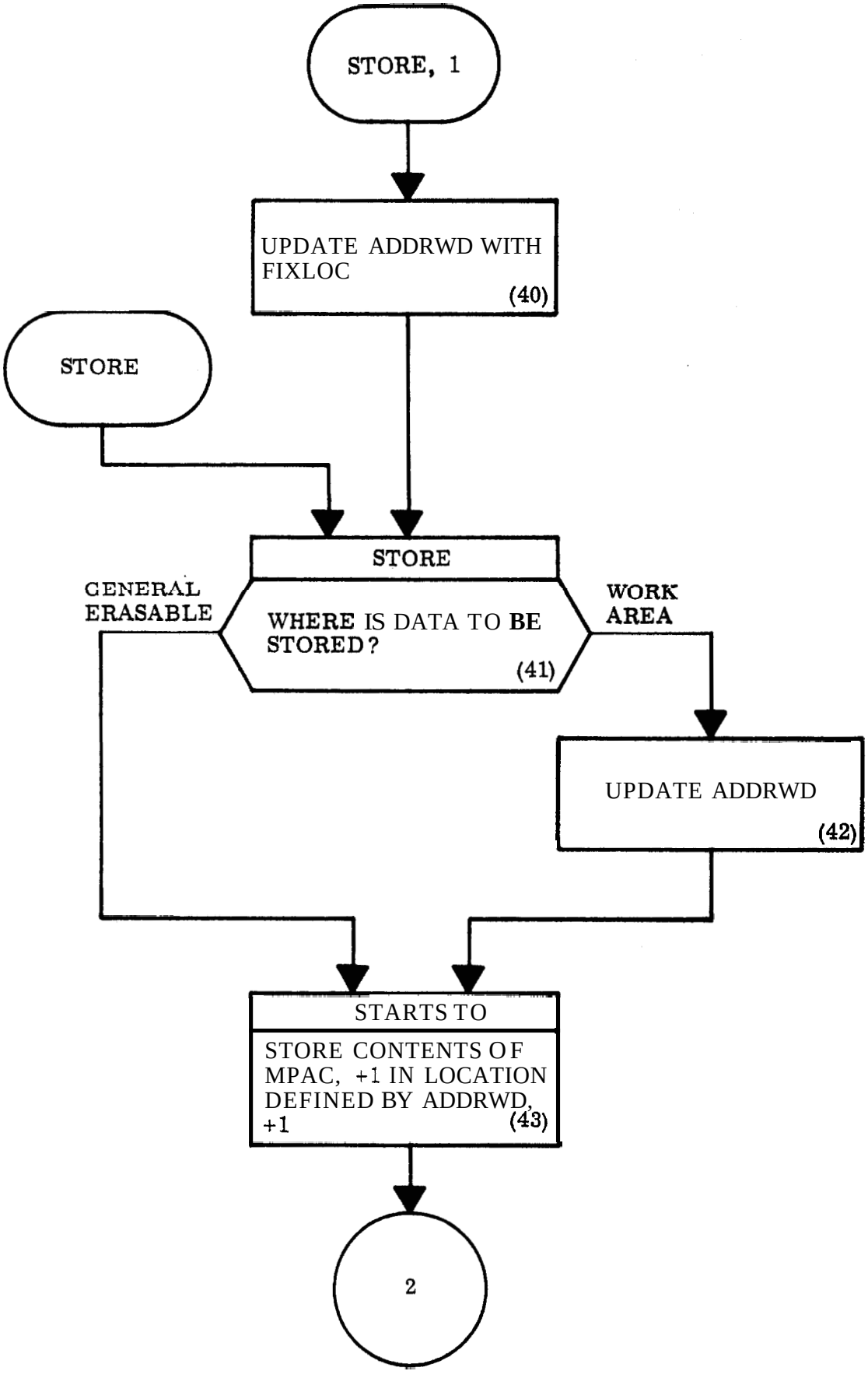


Figure C-3. Interpretive Program Flow (Sheet 8 of 14)

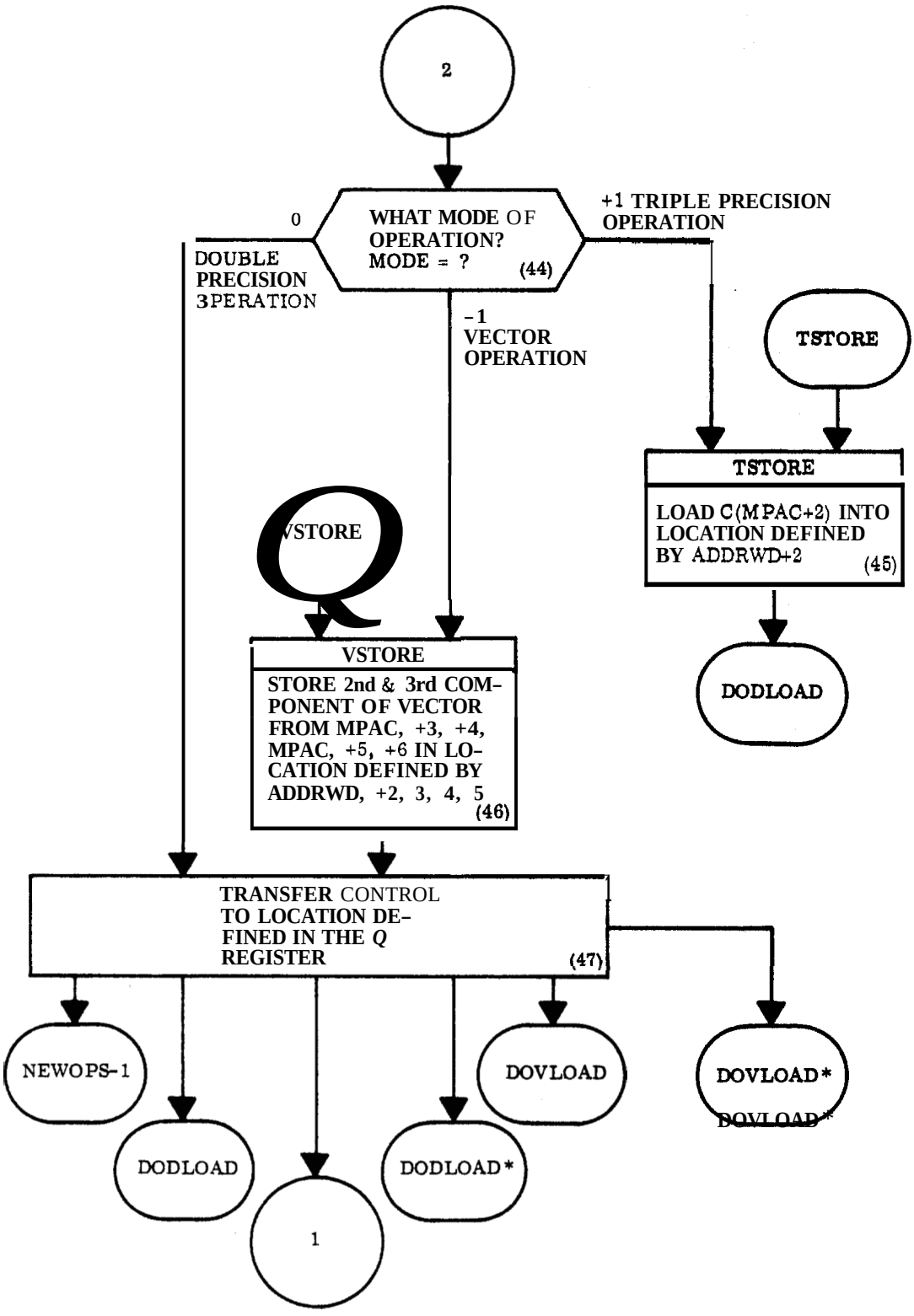


Figure C-3. Interpretive Program Flow **(Sheet 9 of 14)**

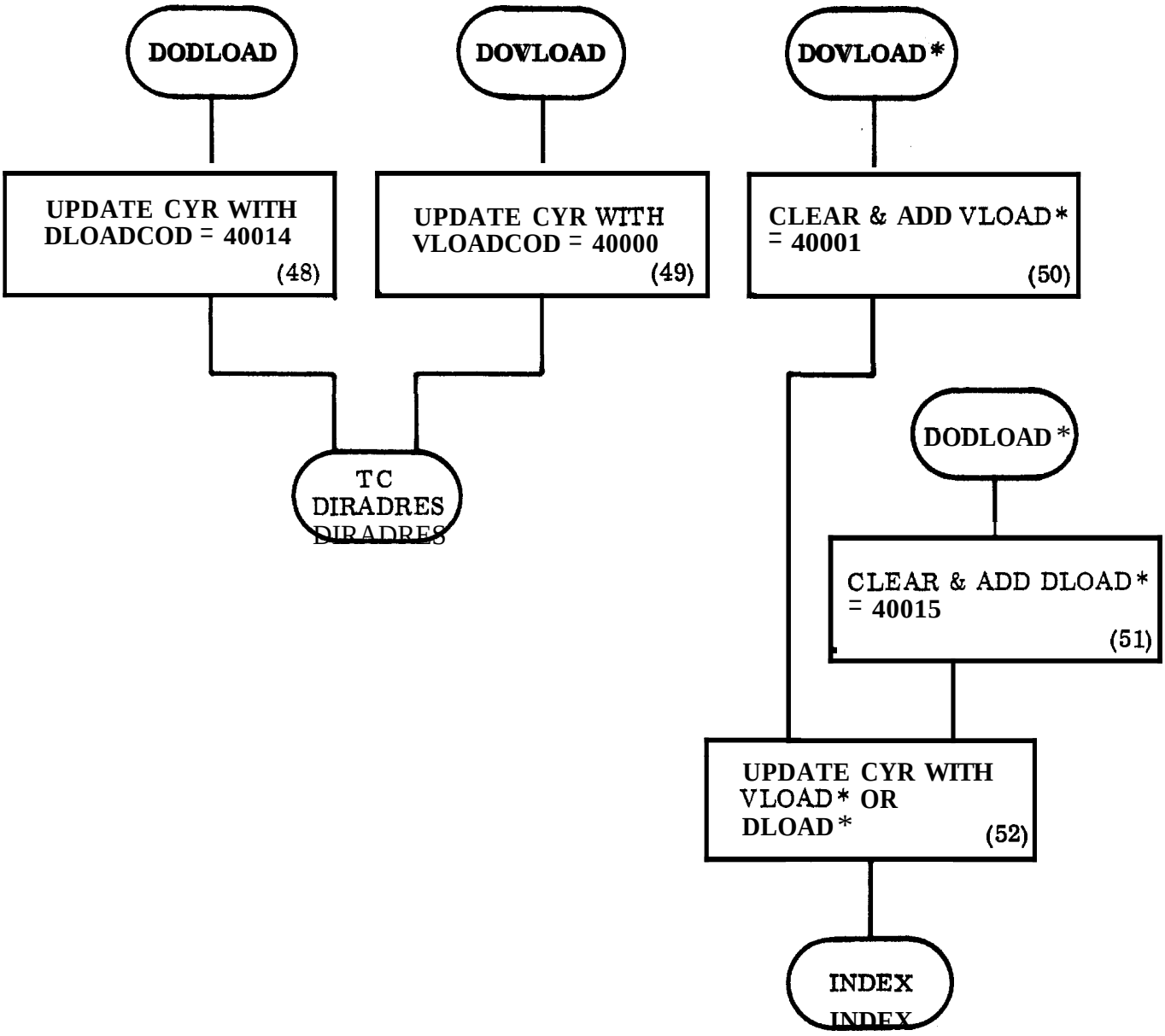


Figure C-3. Interpretive Program Flow **(Sheet 10** of **14)**

(53, 54, 55, & 56) If a load instruction is found, as **will** generally be the case for a first interpretive instruction in **a** string, the single, double, triple precision or vector load operation is defined. The contents of the ADDRWD plus the necessary following words, ADDRRD for single precision ADDRWD, + **1** for double precision; ADDRWD, + **1, 2** for triple precision and ADDRWD, + **1, 2, 3, 4, 5** for vector operations are loaded into their appropriate location in MPAC. MPAC + **2** is not used for loading of vector quantities. In all cases the accumulator is loaded with an appropriate quantity to define interpretive MODE, 0 = double precision, + 1 = triple precision and **–1** = vector. A TC to NEWMODE is commanded.

(57) Here's where you come after you load MPAC for an operation and set up conditions describing the interpretive mode to follow. The MODE indicator is set to a minus **1,** 0 or plus **1** depending on what happened in **Blocks 53, 54, 55,** and **56.**

(58) After the loading of the appropriate data into MPAC by the first instruction in an instruction string we have to look at the second instruction in the same IIW if there is one. The subroutine labeled DANZIG assures that the proper Fixed Banks are selected for future operations and looks at the contents of EDOP which should contain the second half of an IIW if there was one. If an OP code was loaded in EDOP it would have been a Positive Non Zero or a Positive Zero quantity. The CCS on a **PZ** quantity looks at NEWJOB to see if the job presently being done is the highest priority.

If a higher priority is detected a change job subroutine is called. If a higher priority job is not found we close the loop and go back to NEWOPS–1.

If on the check on EDOP an OP code is found we go to OPJUMP which will take the content of EDOP, decrement it by 1 and set this value into **CYR** and we ***go*** around again.

(59, 60, 61, & **62)** For addressing class 10 instructions are located in CYR. The STOREJUMP + CYR transfer control instruction selects a subroutine for selecting a **15** bit address location. The address words are updated by the usual incrementing method and the contents of **CYR** are summed with the address location MISCJUMP. The resulting command is to select a branching or indexing routine ***to*** change the operation sequence of the computer program.

(63) & **(64)** The operations defined by the subroutine PDDL (Pushdown and Load MPAC) in Double Precision is the beginning of the EXECUTER portion of the interpretive program flow. In this operation the contents of the pushlist defined by the location in **PUSHLOC,** is transferred to MPAC, + **1.** A check is made on MODE, is a double or triple precision or the vector mode selected? Depending on the choice **a** subroutine for double, triple or vector is chosen,

(65, **66,** & 67) If a triple precision (65) is detected in the check on MODE the third component of the data is taken from the pushlist and loaded into MPAC + **2.** The MODE Switch is set to double precision. If a double precision **(66)** operation was detected MPAC + **2** is set to zero. If a vector operation **(67)** is found by looking at MODE, MPAC + **2** is set to zero and the remaining components of the vector are taken from the pushlist and loaded into MPAC + **3, 4, 5, 6.** The pushlist indicator, PUSHLOC, is decremented appropriately when data is removed from the list. The data removed from the list is no longer available from the list; it is destroyed. Upon completion of the transfer of data from the pushlist to MPAC for an operation to follow control is transferred back to **DANZIG,** the beginning of the interpretive

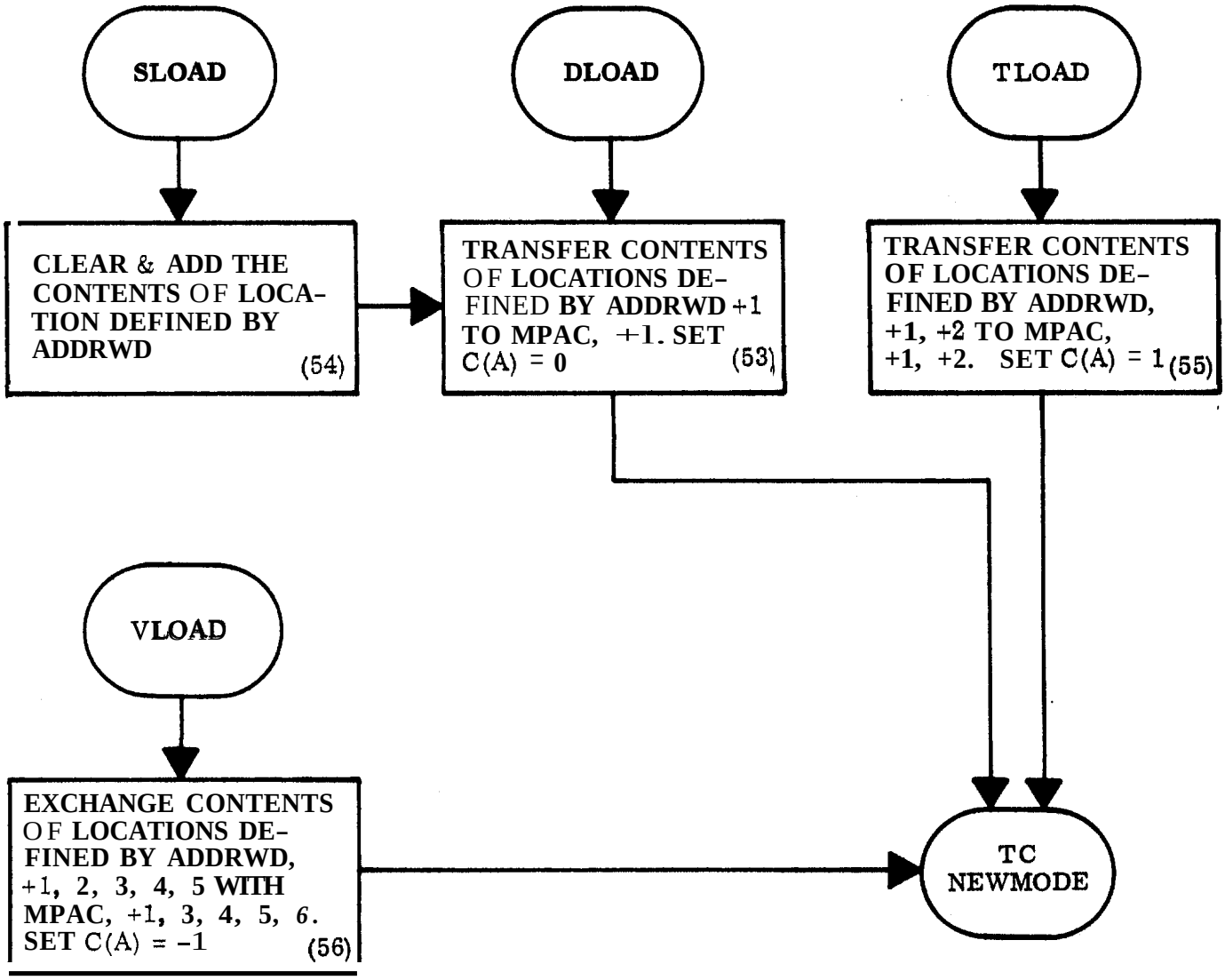


Figure C-3. Interpretive Program Flow (Sheet **11 of 14)**

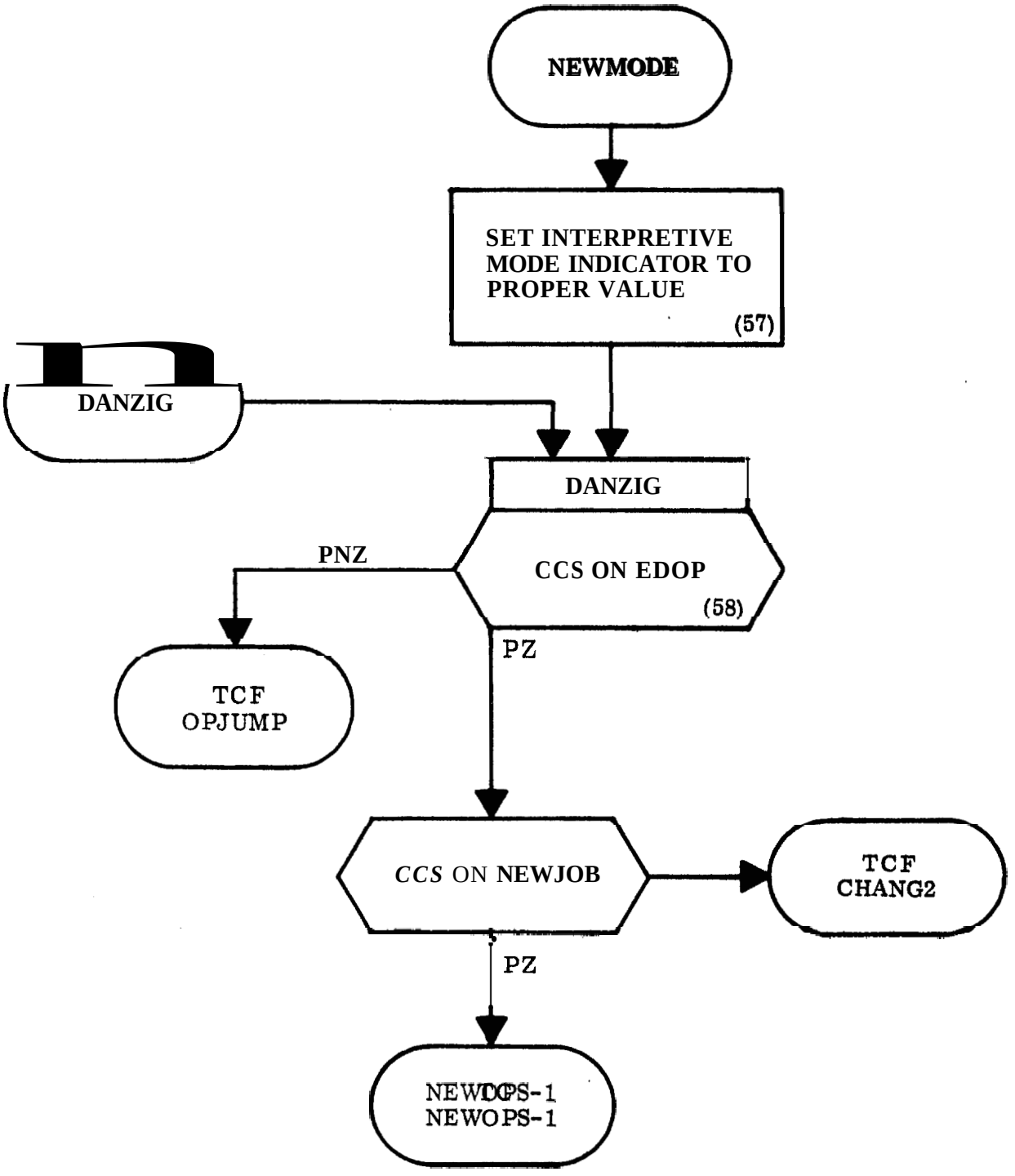
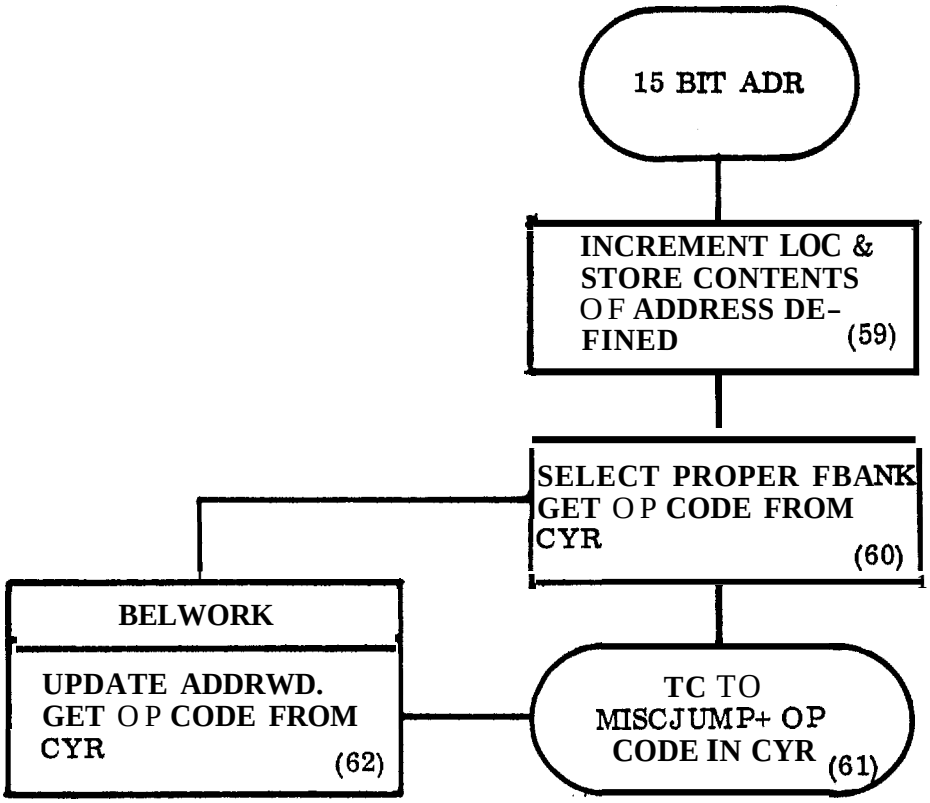


Figure C-3. Interpretive Program Flow (Sheet 12 of 14)



MISCJUMP TABLE INST.	CYR OP CODE
AXT	00
AXC	01
LXA	02
LXC	03
SXA	04
XCHX	05
INCR	06
TIX	07
XAD	10
XSU	11
BZE/GOTO	12
BPL/BMN	13
CALL/ITA	14
RTB/BHIZ	15
SW	16
BOV(B)	17

Figure C-3. Interpretive Program Flow (Sheet 13 of 14)

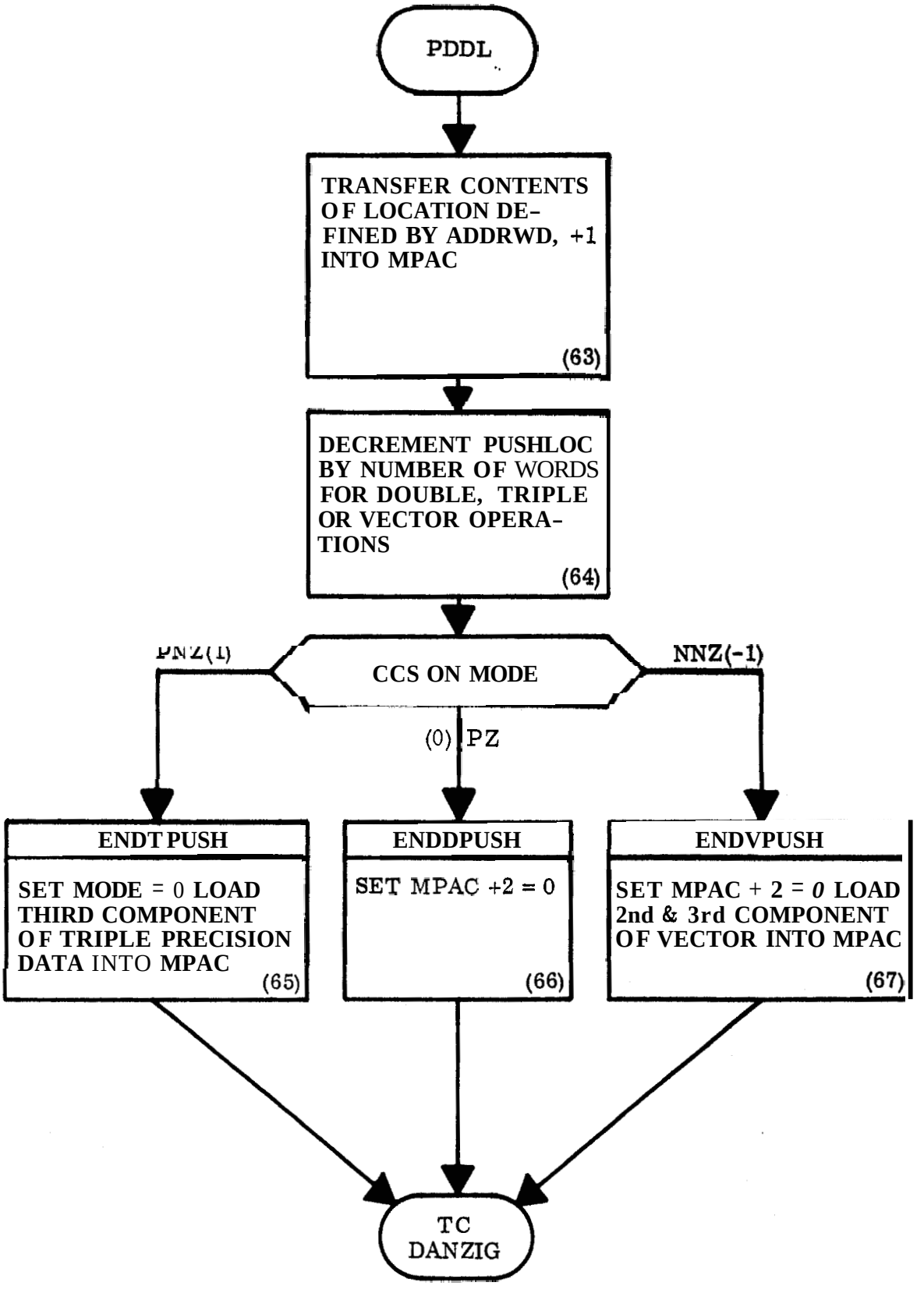


Figure C-3. Interpretive Program Flow (Sheet 14 of 14)

program routine to pick up the next OP code which is stored in EDOP and we start over again.

C.4 SUMMARY

The interpretive section of the computer program is used for programming convenience in the writing of problem orientated computer programs. This program section consists of two basic parts a dispatcher and an executer. The dispatcher determines the type of interpretive program operation commanded by the operation code in an Interpretive Instruction Word and what type of addressing scheme is to follow. The executer portion of the program is the specific subroutines which are selected by the dispatcher, which take an instruction such as SQUARE ROOT and perform a square root operation in the basic machine language. The executer subroutines always return the operation back to the dispatcher to select the next operation or to allow an exit from the interpretive program

The interpretive program by use of seven bit order codes and 15 bit address codes contribute a great deal to the versatility of the computer as a problem solving device.

FINIS