Internal Maintenance Specification

1700 Mass Storage Operating System

Version 3

1700 MSOS 3.0

M136

TABLE OF CONTENTS

System Initialization

I/O Driver Handling Routines

Input/Output Drivers

System & Program Maintenance Routines

Re-entrant Fortran Package

Message Buffering

1.0 LOCORE

Communications region, interrupt traps and preset entry point table.

1.1 Program Function

This program contains three functional parts. These are: {1} communications region-this area contains system constants that are common for all 2.1 MSOS systems; {2} interrupt trap region-this area is set up to provide the interrupt handling desired; {3} preset entry point table-this table allows entry to a protected routine from an unprotected routine.

1.2 Communications Region

Locations $0 thru $FF are directly addressable in the 1700 computer. The communications region is set up to take advantage of this feature. The first part of the table contains constants which provide commonly used masks. The second part of the table contains system parameters such as flags, counters and addresses of system entry points.

1.3 Interrupt Trap Region

The interrupt trap region encompasses the area from $100 to $13F. Each interrupt line has associated with it a four word area within this region: line zero-$100 to $103, line one-$104 to $107, etc. A typical setup for line zero is shown below:

```
LINE0    NUM 0          WORD 1
         RTJ {$F8}      WORD 2
         NUM 15         WORD 3
         ADC IPR0C      WORD 4
```

Word one is reserved for the hardware storage of the program address on interrupt.

Word two is the first instruction executed following an interrupt. A return jump to a system interrupt processor is then made. For line zero the transfer is to the internal interrupt handler {IPR0C}. The other lines transfer to the common interrupt handler {ALUN}. These routines provide for saving of registers and program conditions via an interrupt stack.

Word three is used to hold the priority level associated with the interrupt line. The MASKT table is indexed by this ordinal.

Word four contains the address of the processor for this line. Most lines use the routine EPR0C. Use of EPR0C dictates that the interrupting device must return bit two as an "interrupt status" during a status operation and all hardware devices must be ordered by logical unit number on the interrupt lines. A special interrupt routine may be designated here instead of EPR0C.

1.4 Table of Presets

The table of presets provides entry to a protected routine from an unprotected routine. Entries are made following the interrupt trap region in the following manner:

```
ALF     3,NAME
ADC     NAME
EXT     NAME
```

Name is an entry point to a core resident system program. See section for further information.

2.0 SYSBUF - System tables and parameters.

2.1 Program Function

The following system buffers system tables and miscellaneous subroutines are contained in this program.

1. Logical Unit Tables

   A. LOG1A
   B. LOG1
   C. LOG2

2. Interrupt Mask Table

   A. MASKT

3. Storage Stacks

   A. INTSTK - Interrupt stack
   B. VOLBLK - Volatile storage
   C. SCHSTK - Scheduler stack

4. Core Allocation Data

   A. CALTHD
   B. LVLSTR

5. Diagnostic Tables

   A. ALTERR
   B. DGNTAB

6. Mass Memory Diagnostics

   A. MMDIAG

7. Miscellaneous Programs

   A. IDLE
   B. OVRLAY

8. TIMER, RTMS, TRACE and ODEBUG information.

A. TIMCPS, TIMEC, TIMACK, TODLVL, NSCHED
B. R17LOC
C. STOPIT
D. CHRSFG

9. Dummy Device Driver

A. DUMDRV

10. Physical Device Tables

2.2 Logical Unit Tables

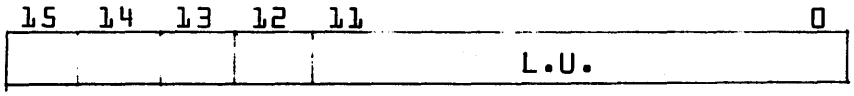
2.2.1 LOG1A - Table

This program segment contains ADC's for each system logical unit. Each ADC will specify the address of the physical device table for a specific logical unit. This table provides the definition of logical unit per physical device.

Certain restraints are imposed on the ordering of this table: (1) the core allocator is logical unit one, (2) use of EPROC requires that each logical unit is ordered by interrupt line number. Standard logical units (i.e. input, list, etc.) are assigned by equates referencing this table.

2.2.2 LOG1 Table

An entry is placed in this table for each logical unit. The format is as shown below:



BIT 14 = 1 implies the logical unit shares a device (ex. FORTRAN and NON-FORTRAN Printer Driver would have two logical units assigned to one physical printer)

BIT 13 = 1 implies the logical unit is marked down

BIT 0 thru 11
(L.U.) is the alternate logical unit (zero implies no alternate)

2.2.3 LOG2 Table

This table is preset to $FFFF for each logical unit. This table is used to hold the top of the thread address for requests for each logical unit.

2.3 Interrupt Mask Table

This table is indexed by priority level to set the M-register. The M-register setting will determine which interrupt lines will be enabled for a given priority level. Two basic rules to be followed are:

1. Unused interrupt lines should have their corresponding bit set to zero throughout the table. (Bits 0 through 15 of the M-register correspond to interrupt lines 0 through 15)

2. A software priority is associated with each interrupt line. More than one line can be associated with the same priority and can have the same mask. A "1" bit is placed in the table for the bit position associated with an interrupt line for all levels below the priority level associated with that line. "0" bits must be placed in the interrupt line position for all the priority levels equal to and above the priority level associated with the line.

2.4 Interrupt Stack

The interrupt stack (INTSTK) is the block of storage set aside for saving the status of interrupted programs. The Common Interrupt Handler stores the Q-, A-I- and P- registers and the overflow indicator and the priority level of the interrupted program in this area. Five words are required for each entry and the stack is of the push-down, pop-up type, i.e., last-in, first out. The number of entries to be allowed for in the table is derived from the number of different priority levels used by interrupts.

2.4.1 Interrupt Stack Entry

Word

0 Q - Register
1 A - Register
2 I - Register
3 Overflow (bit 15), P-Register
4 Priority Level

2.5 Volatile Storage

VOLBLK is the volatile storage area set aside for allocation of data storage for routines that must be re-entrant (i.e., may be operated at more than one level at the same time). Sufficient volatile storage must be reserved for each priority level to accommodate the worst case or maximum amount of volatile storage that can be requested at each level. The system cannot recover from overflow (more requested than is available) of volatile storage.

2.5.1 Core Requirements

Core may be reserved for volatile following these quidelines:

1. For each priority level in which monitor requests are made, eight locations of volatile are required. If the Request Processor itself makes a request, such as a Secondary Scheduler call, an additional eight locations are required. Thus sixteen locations must be reserved per priority level plus additional locations if used by other re-entrant routines.

2. If the re-entrant FORTRAN Object Library and re-entrant Encode/Decode Package are used 49 locations must be reserved for each priority level using the re-entrant FORTRAN library plus 56 locations for each priority level using re-entrant Encode/Decode.

2.6 Scheduler Stack

This stack, SCHSTK, is a series of four-word entries.

Word

0	priority level
1	completion address
2	thread to next entry
3	value of Q-register being passed

A program may request the operation of another program by making a scheduler request. Scheduler requests may also be generated by the Timer Routine after a given interval of time has elapsed. These requests are threaded together on the scheduler stack.

The total number of entries required is equal to the sum of the number of scheduler requests and timer requests that can be on the stack at one time. The size of this stack may be changed by the user.

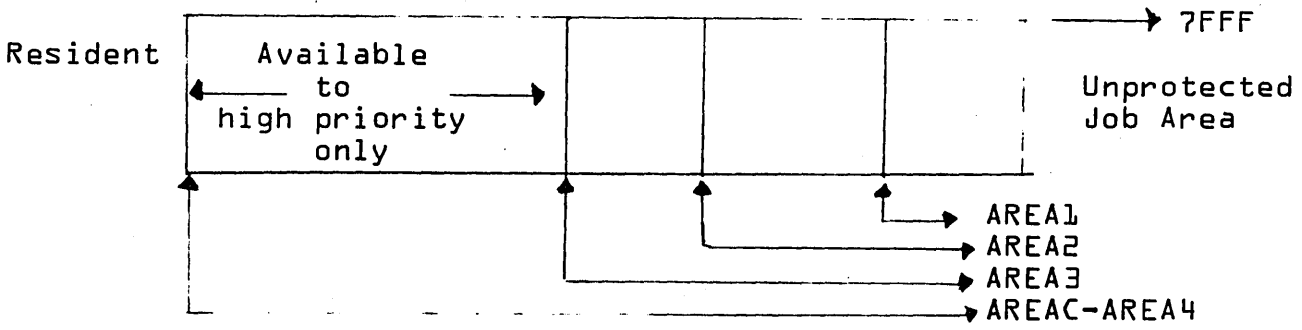
2.7 SECPR0

Normally all entries are left empty, i.e., NUM $7FFF. The address of a special interrupt response routine can be included in the entry for its line, but it is more efficient to put this address in the fourth word of the interrupt trap location instead of using EPR0C and SECPR0.

2.8 Core Allocation Data

LVLSTR is the table of starting addresses for the allocatable core area available to each priority level. The upper bound for protected allocatable area is the same for all levels: the start of unprotected core. To prevent low-priority programs from tying up all of the allocatable area, it is common to restrict the amount available to them while making the entire allocatable area available to the high priority programs.

LVLSTR is set up to reflect the above. AREA1, AREA2, AREA3 and AREA4 are entry point names in the SPACE program used to divide the allocatable area as shown below.



Request priorities 1, 2 and 3 must include sufficient area for the Job Processor. The entire area-AREAC must be available at request priority zero so the system can get started as initialized.

2.9 Diagnostic Tables

2.9.1 Alternate Device Error Table

This table reserves one word for each device which could have a simultaneous failure.

2.9.2 Diagnostic Timer Table

An ADC entry is made pointing to the physical device table for each logical unit to be supervised by the diagnostic timer.

2.10 Mass Memory Diagnostic Routine

MMDIAG is a routine that prints a message of the following form:

MASS MEM ERR n

n is the error code

If the request that resulted in failure was a System Directory request, this routine releases allocated core. Control then returns back to the driver. This routine may be modified to perform additional functions such as making a Timer Request for the scheduled program to be attempted again at a later time.

2.11 Idle Loop

This routine runs at level -1 when no other programs are running. This routine may be modified to provide a counter to monitor the amount of idle time. If IDLE is running a snap dump may be taken by stepping, clearing the A-register and hitting run.

2.12 Overlay Subroutine

The overlay subroutine allows users to call for mass memory to be read over the actual call parameters. This is accomplished in the disk or drum driver by moving the parameter list to the equipment table and using the overlay subroutine to ensure that the return address from the call cannot be written over. Indirect overlay calls are not permitted.

The following example shows a typical overlay disk read.

```
RTJ   OVRLAY
ADC   $200,COMP,0,$8C2,N,BUF,0,ADR
```

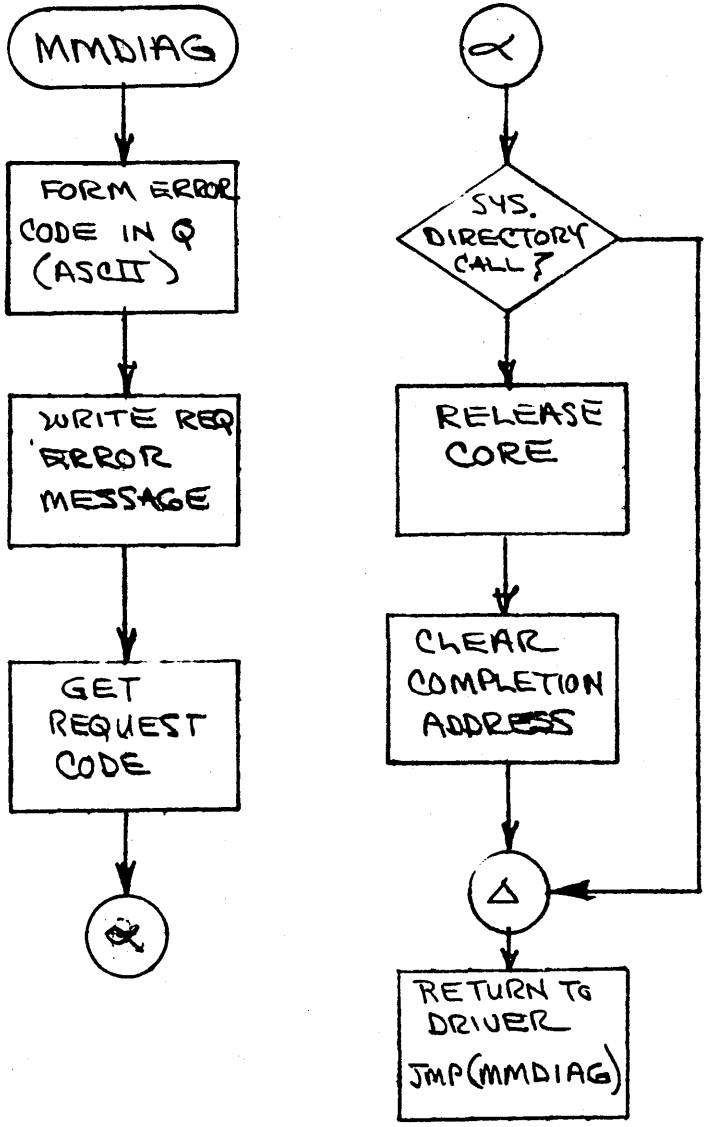
2.13 Dummy Device Driver

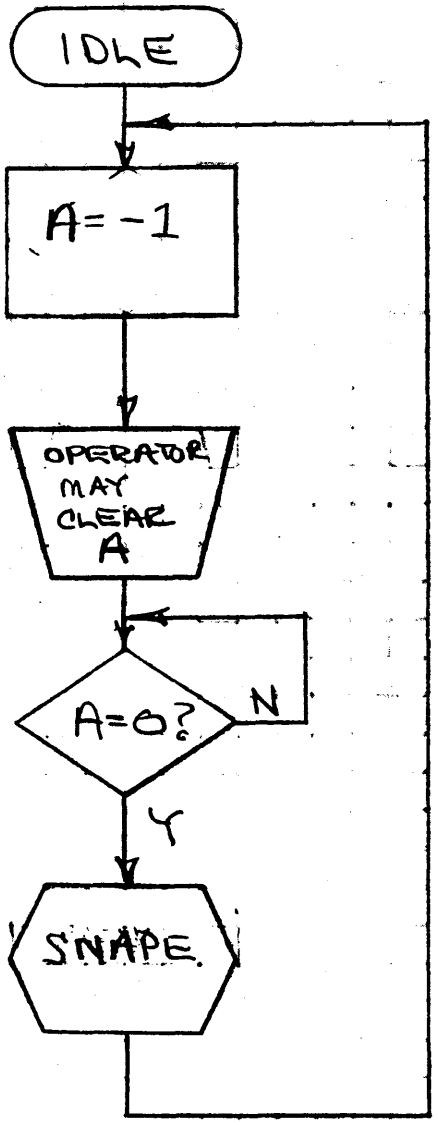
This routine is used with the dummy device table and is assigned a logical unit like a normal device. Read or write requests that address this logical unit cause the dummy driver to be initiated, and the completion address in the request is scheduled with error indication. This allows the Dummy Device to be setup as the alternate for devices where it would not be acceptable to hang up the request waiting for operator action in response to the Alternate Device Handler request for input.

2.14 Physical Device Tables

The physical device table (PDT's) contain all the device data necessary for a device to be operated by its driver. Word 0 through 12 have the same general use for all device drivers. Words from 13 on are used for special purposes appropriate to the driver, if necessary.

Each drivers physical device table setup is outlined in the 1700 MSOS Installation Handbook Pub. No. 60234300B.

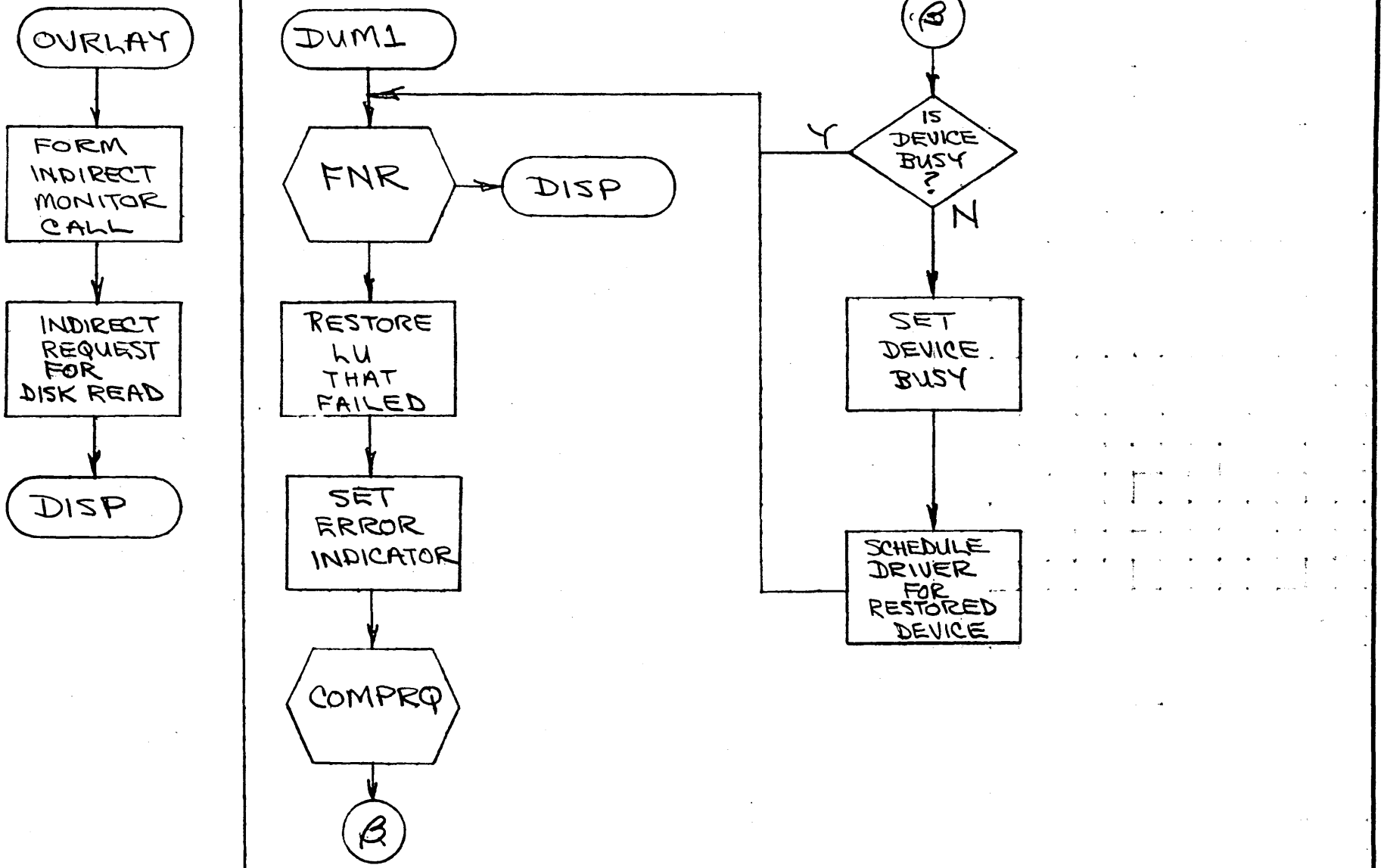




CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MMDIAG	PROJECT MGR.			
	IDLE PAGE 1 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

2.8



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE OVRLAY	PROJECT MGR			
	DUM1 PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

2.9

3.0 SCHEDULER

3.1 EXTERNAL SYMBOLS

TLO — Entry point of SPACE request
SCHTOP — Location in System Tables containing location of top entry in schedule stack

3.2 FUNCTION

In a given system, numerous requests for the execution of programs at specific priority levels may be generated. Specifically, these requests are generated when

a) an I/O request has been completed,
b) a specified time interval has elapsed,
c) core has been allocated,
d) a mass memory request has been executed.

Requests may also be made by any program directly. They are called Scheduler Requests.

It is the function of the Scheduler Request Processor to

a) cause the immediate execution of a requested program if it is of a higher priority level than the requesting (current) program, or
b) thread the request by priority and within a priority by first in first out, if its priority is the current priority.

If the requested program is mass memory resident, the Scheduler Processor will cause allocation of core for this program and transfer of the program from mass memory. After the program has been transferred, a Scheduler Request is made, which results in a) or b) above.

Whenever a program terminates, the Program Dispatcher will select the next program to be run, either from the top of the scheduler thread or the interrupt stack.

3.3 Entry Interfaces

Program is entered from the Request Entry Processor. The calling (requesting) program must have interrupts enabled.

3.4 Exit Interface

The program exits either to the requested program {completion address}, if the level is higher than the current one or to the request exit.

In the first case the priority level, I and the return address leading to the request exit are saved in the proper positions of the interrupt stack and its base adjusted. A, Q and I are saved in volatile, which is not released until the requested program terminates. I contains the base of volatile storage, when control is given to the requested program.

Interrupts are enabled and the requested priority level and mask set.

In the second case the request has been threaded. Control goes to REQXT to restore the registers for the requestor and enable interrupts.

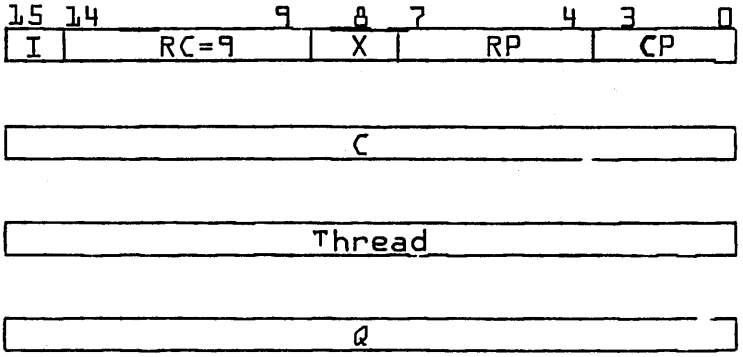
3.5 Internal Description

All Scheduler Requests are identified by the request entry processor, which also allocates a sufficient amount of volatile storage for re-entrancy purposes. Then control is given to the Scheduler Request Processor {Symbol T9}. Interrupts are enabled and I contains the base address of the allocated volatile storage. Volatile is organized in this way.

{I} + 0 contains Q
{I} + 1 contains A
{I} + 2 contains I
{I} + 3 contains Return Address
{I} + 4 contains Priority Level of Request
{I} + 5 contains Pointer to Request Parameter List
{I} + 6 contains First Word of Request {Temp.}
{I} + 7 contains Second Word of Request {Temp.}

First the return address is adjusted by two locations unless the call was indirect, in which case it had already been adjusted by the Request Entry Processor. Then word 1 and 2 of the call are stored in volatile temporarily. If the call is a directory call control is given to DIRCAL. If not a directory call, a test is made to see if the requested program is of higher level than the current one in which case control transfers to HILVL.

Otherwise, a test for a primary call (SCHDLE request) is made and only then, if it is not a directory call, not of a higher level and not a secondary call, is a position in the Scheduler Stack obtained and the request transferred from volatile (I) + 6 and (I) + 7 into the stack. The top of the available (empty) thread is in TOMPT. All "empties" (available entries) are threaded together, the last one containing -0 in the thread. Thus if TOMPT contains -0, it indicates that the stack has overflowed, in which case an error routine is entered. The Scheduler Stack is shared with entries from timer requests. It effectively buffers these requests, so that the requesting programs may leave core. One entry on the Scheduler/Timer Stack consists of 4 words.



The completion address is absolutized before going on the stack as it cannot be absolutized later.

The Scheduler/Timer Stack and TOMPT must be preset by the user.

For directory calls the DIRCAL routine determines after placing the call priority into the directory, if the called program is mass memory resident. In this case, control goes to the Allocator Request Processor (Symbol T10). Otherwise, control goes to SCHED2 or to HILVL depending on the priority of the request.

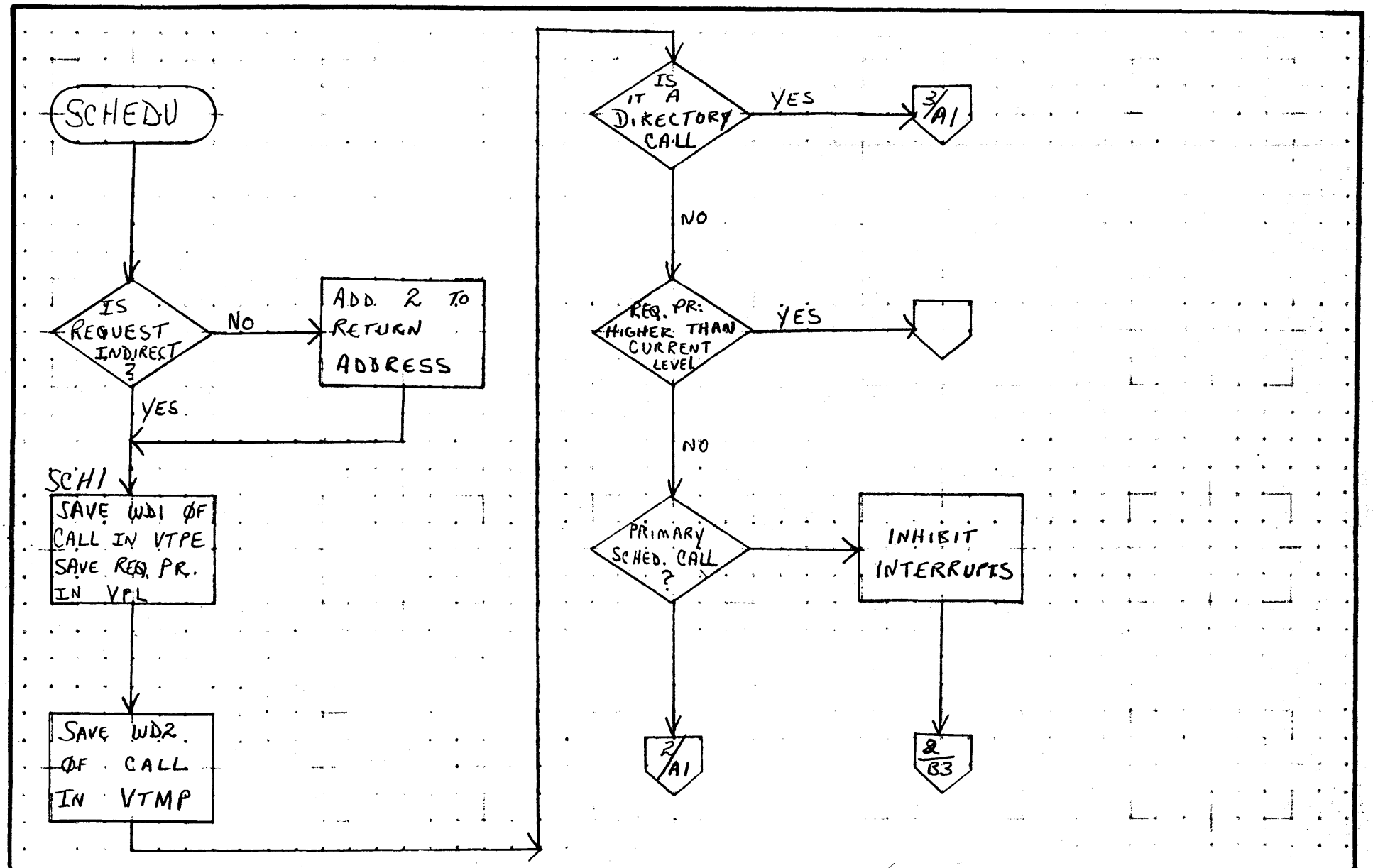
At SCHED2 the original contents of the Q-register are stored in the request from volatile (I) + 0 and the request is then threaded on the Scheduler thread. This thread is not to be confused with the Scheduler Stack since it contains secondary scheduler requests located somewhere in a user program, primary requests located in the stack and directory requested located in the directory.

In the THRED1 routine a position for the new request is determined according to the priority of the new entry. It may be at the end of the beginning of the existing thread in particular if it is the first entry.

The routine THREAD accomplished the actual threading.

HILVL gives control to the requested program immediately after saving all information necessary to continue the requesting program. A request for a higher level program can therefore be considered a pseudo interrupt.

The current priority level and I are saved in the interrupt stack and the interrupt stack base address count is incremented by 5. The request exit is stored as the return address since upon return from the program volatile must be restored as well as A and Q. Then the requested priority level and the associated mask are set and control is given to the new ▽Go To▽ address.



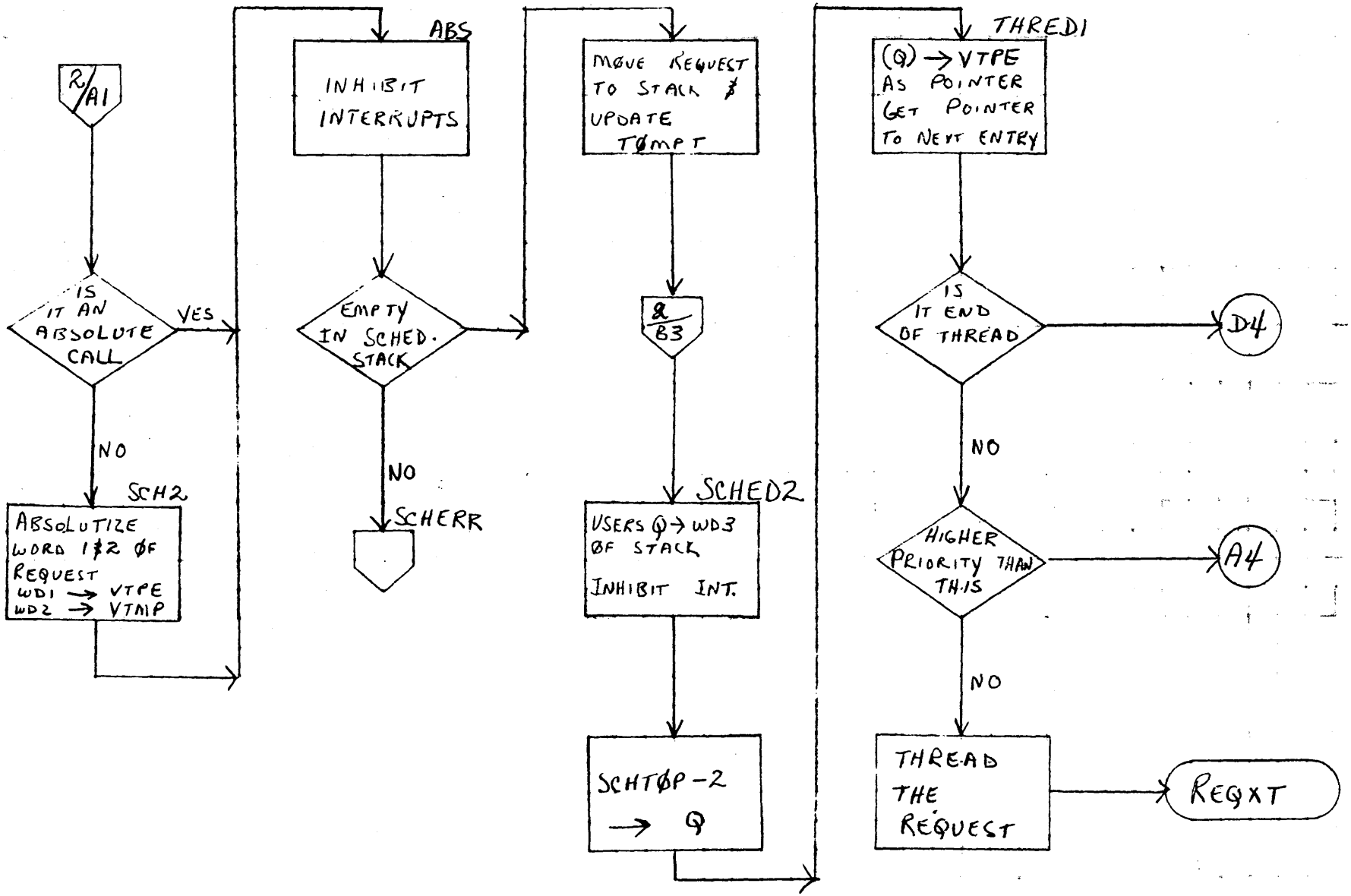
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IAS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SCHEDULER	PROJECT MGR.			
	SCHEDU PAGE 1 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 6/6/70	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

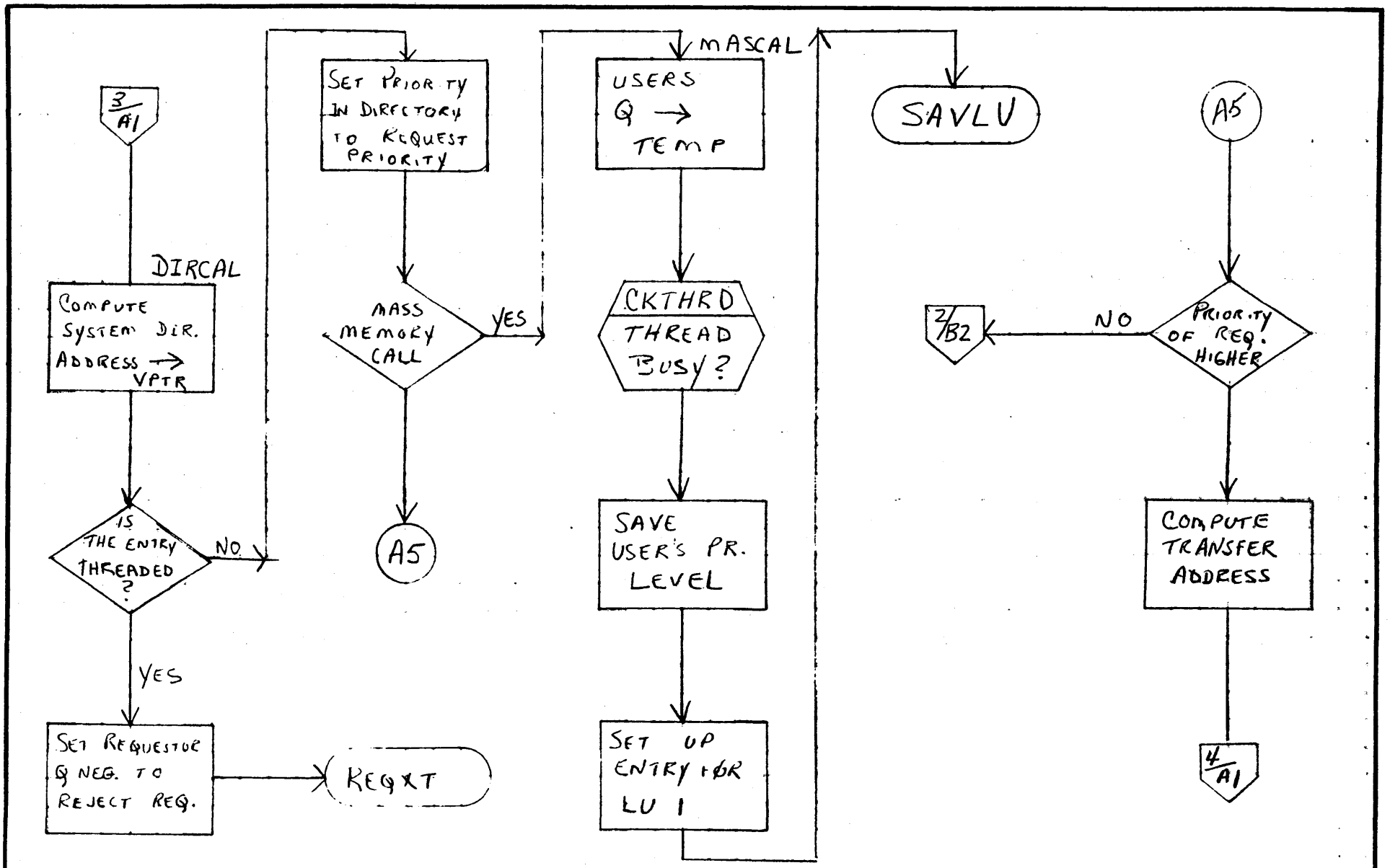
3.5



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE: SCHEDULER		PROJECT MGR.		
FLOWCHART ☐	SCHED0	PAGE 2 OF 4	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE 10/ /70	TASK NAME		

MAR 5 1971

3-6



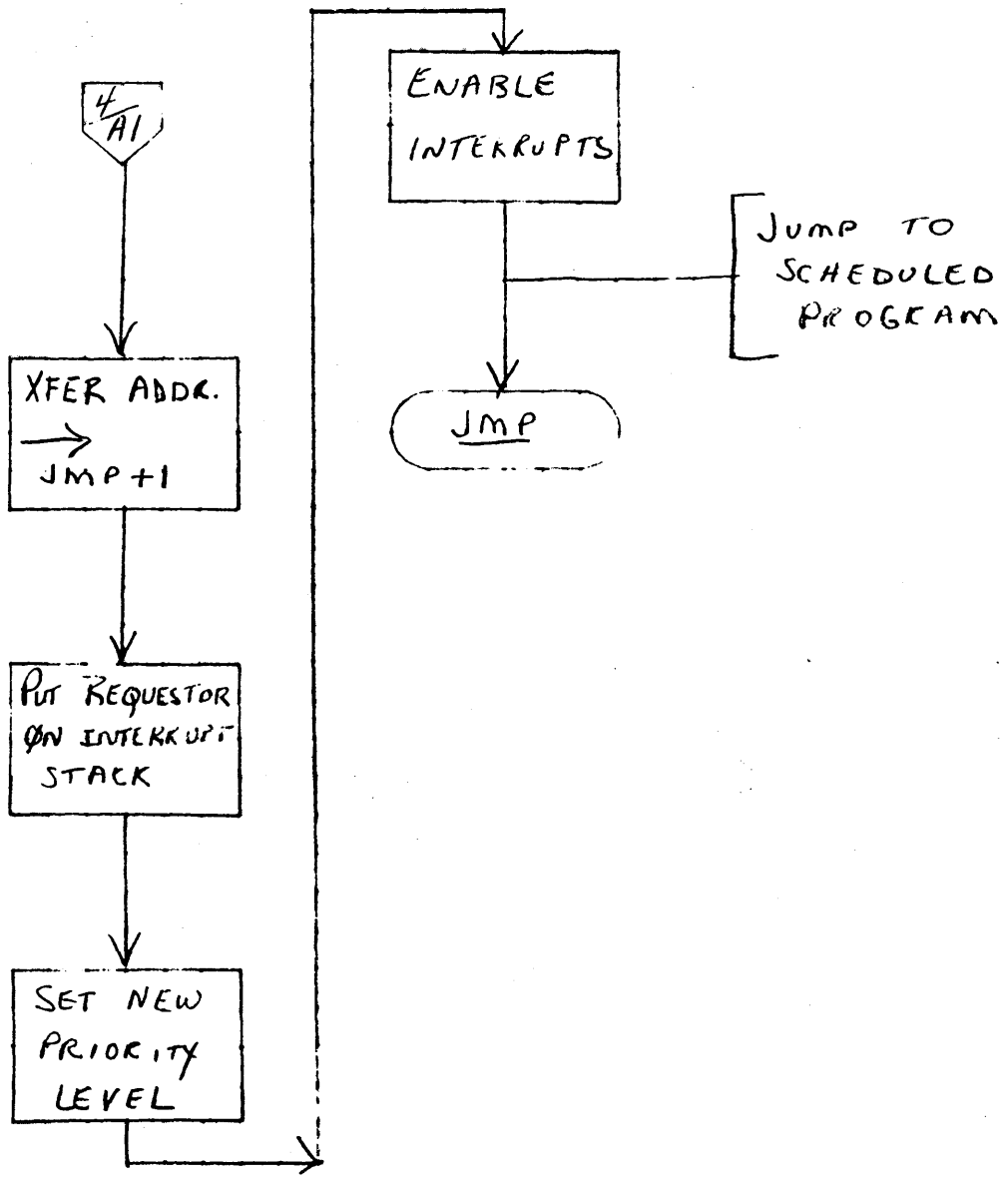
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INIS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SCHEDULER	PROJECT MGR.			
	SCHEDU PAGE 3 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 10/6/70	TASK NAME			

AA1385 (FORM) CA127-1

PRINTED IN U.S.

MAR 5 1971

3.7



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SCHEDULER	PROJECT MGR.			
	PAGE 4 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 10/6/70	TASK NAME			

AA1385 (FORMERLY CA 27)

PRINTED IN U.S.A.

MAR 5 1971 3.8

4.0 Dispatcher

4.1 Internal Symbols

SCHSTC Routine to initiate a program when taken from the scheduler thread.
DISP Start of Program Dispatcher
COMEXT Defined by an EQU and determines the interrupt trap slot location to be used as a common exit.

4.2 Dispatcher Function

Whenever a protected program terminates, it will give control to the Program Dispatcher*. The Program Dispatcher decides which program shall be initiated next. It could be a program previously interrupted and waiting on the interrupt stack, or a program that has been scheduled and is waiting in the scheduler thread. The highest priority program is then initiated by the Program Dispatcher and control given to it.

4.3 Entry Interfaces

Entered via a jump to entry point DISP.

4.4 Exit Interfaces

If control is given to the program that was previously interrupted, the A-, Q-, I-, and M-registers and the overflow are restored to their previous condition, as well as priority level. Interrupts are enabled, and control returns to the location at which the interrupt originally occurred.

If control is given to a program on the scheduler thread, ▽A▽ will contain the address of the scheduler thread entry, Q will contain the fourth word of the entry (the original Q in scheduler calls, or an error indication in I/O calls, or the time of day in timer calls), priority level and M will contain the configuration specified in the first word of the entry, and I and overflow will be an arbitrary configuration. Interrupts are enabled.

4.5 Internal Description

After the program is entered, a test is made to determine whether the priority of the highest interrupted program is ≥ to the priority of highest program waiting in the scheduler thread. If the interrupted program is to be resumed, the return address is stored in the common exit and I and A are restored. Then, the interrupt stack base is adjusted down by 5 and stored in COUNT, and the priority level restored

*Protected programs may also terminate with a RELEAS request which jumps to the Program Dispatcher.

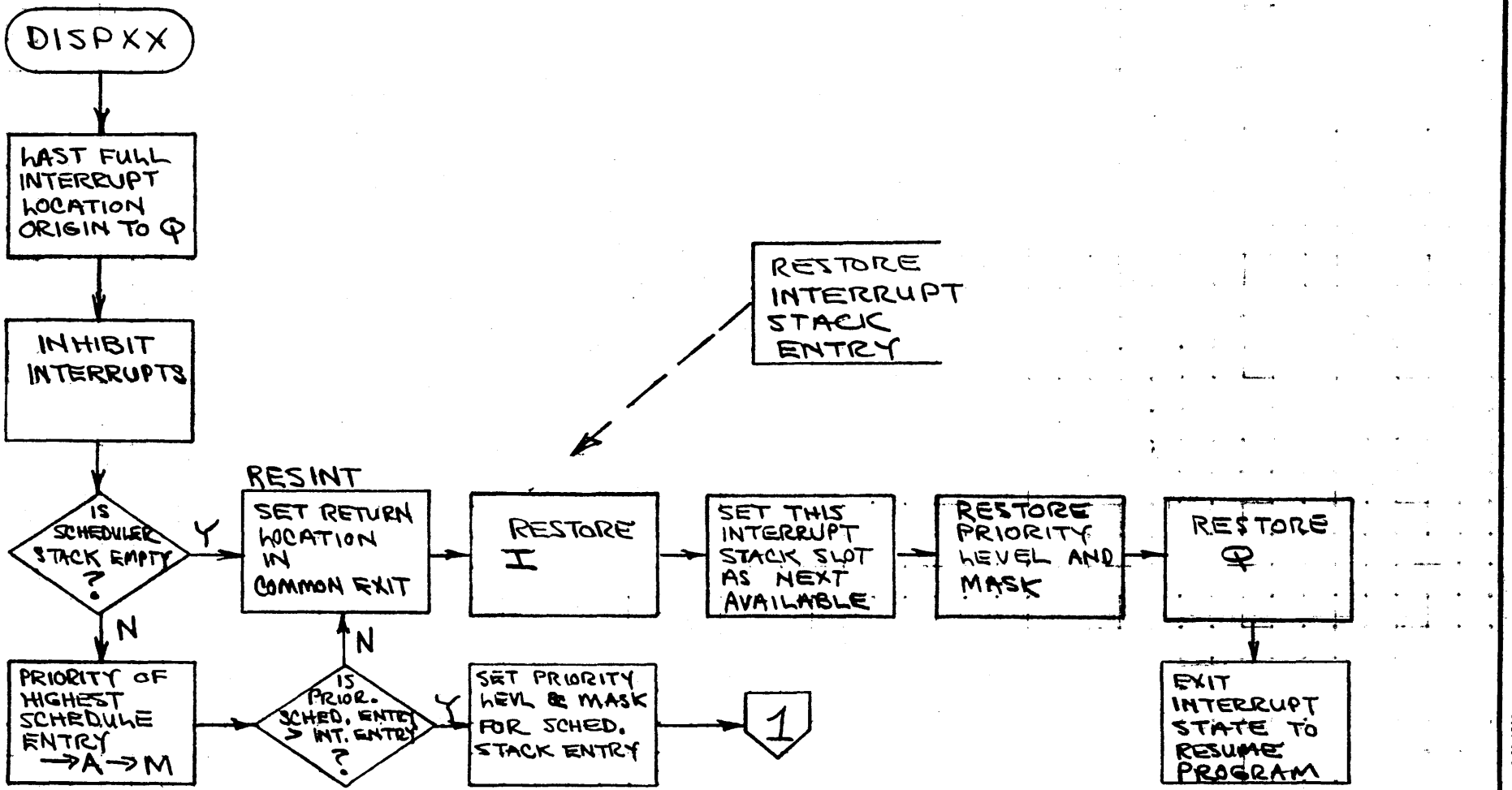
into the cell containing priority level. The mask associated with this level is transferred into M (which restores M), and then Q is also restored. Control is returned to the interrupted program by an EXI instruction which restores overflow and enables interrupts.

If the program of highest level is on the scheduler thread, the priority specified in the highest thread entry (the address of this entry is in SCHTOP) is placed into the cell containing priority level, and the associated mask placed into M. Then SCHTOP is updated pointing now to the next entry in the thread. If there is no other entry, it contains -0.

Next, a test is made whether the scheduler thread entry was a primary entry (i.e., not resulting from a completed I/O call or an expired timer call) and is in the scheduler stack (see specification on scheduler for difference between scheduler stack and thread).

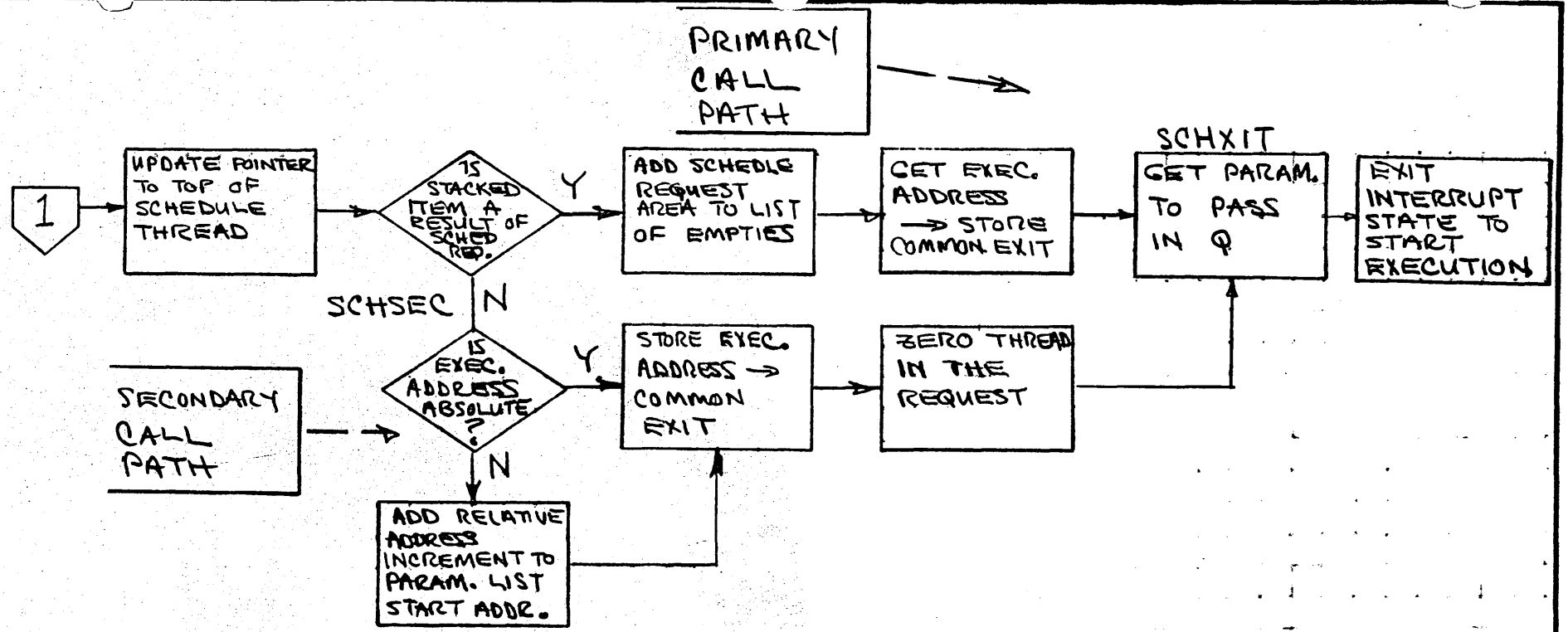
If yes, the scheduler stack position is added to the thread of "empties" and the address to which control is to be given is stored in the common exit. Then the address of the entry is put into A and the fourth word of the call into Q. Control is transferred with an EXI instruction which enables interrupts.

If the scheduler thread entry resulted from an I/O or Timer call, the specified completion location may be relative. If it is, the absolute address is determined and the address stored in the common exit. Then the third word of the entry (containing the thread) is set to 0 as an indication that the call is completed and could be made again. A and Q are loaded and control is transferred as above.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DISPATCHER	PROJECT MGR.			
	NDISP PAGE 1 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	ASK NAME			

MAR 5 1971



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE DISPATCHER		PROJECT MGR		
FLOWCHART ☐	NDISP	PAGE 2 OF 2	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

5.0 Scheduler-Dispatcher for Re-entrant FORTRAN Library

5.1 Program funtions of RDISP

This version of the scheduler/dispatcher is the same as the scheduler (SCHEDU) and dispatcher (NDISP) with the additional capability to save or restore the FORTRAN scratch area and the library temporary locations on changes of priority level which would give control to another FORTRAN user.

5.2 Entry Interfaces (see sections 3.3, 4.3)

5.3 Exit Interfaces (see sections 3.4, 4.4)

5.4 Internal Description

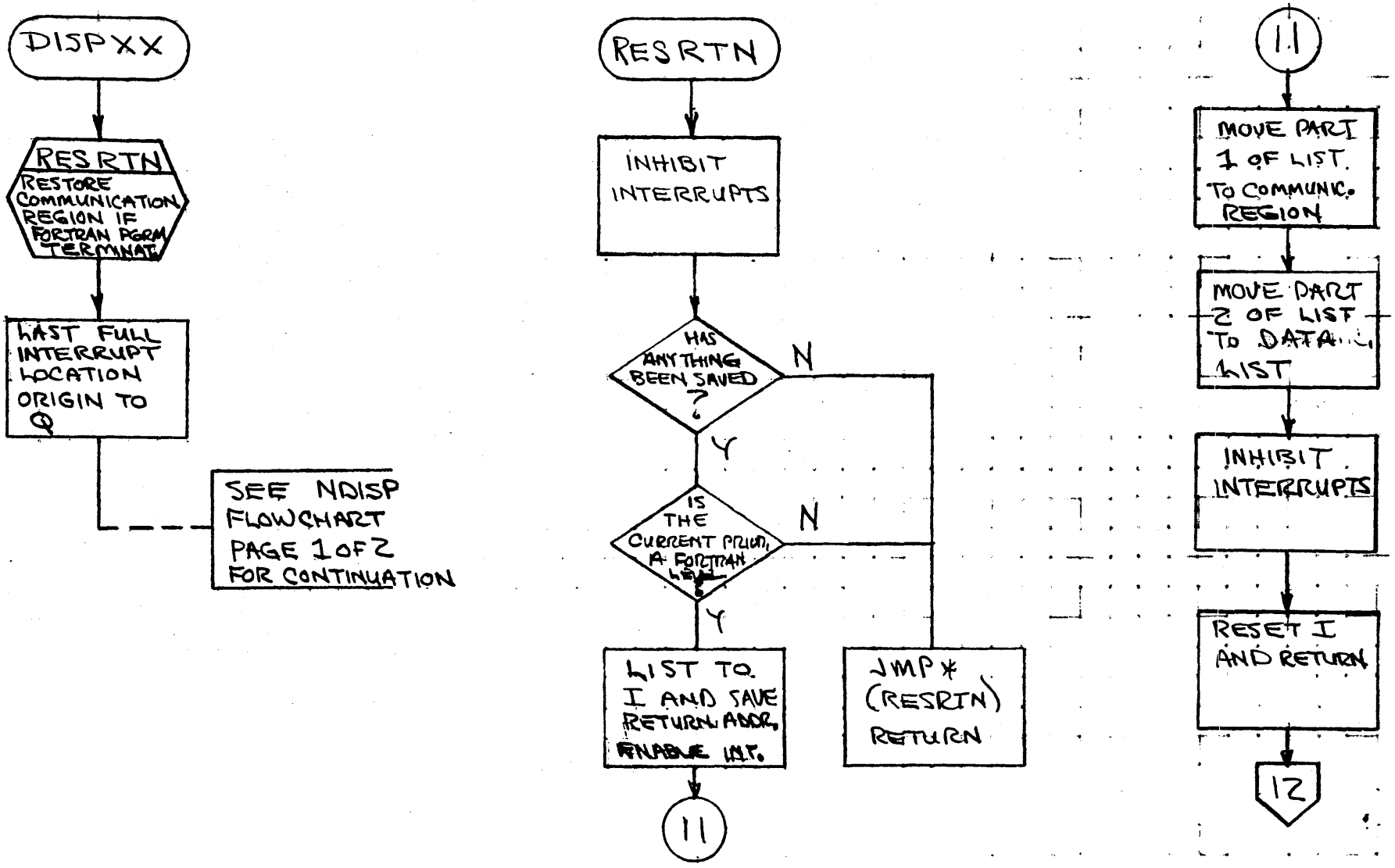
The sections 3.5 and 4.5 of this IMS should be referenced for the internal description of the normal dispatcher and scheduler. The text in this section only describes the differences in this module from the standard.

If entry into the scheduler is by a program requesting a higher level program to be run, other than a mass memory directory call, then the requestor is put on the interrupt stack and control is passed to SAVE at the requested priority.

At SAVE a test is made to see if the new priority level is a FORTRAN level and a new FORTRAN level. If so we must save our data to provide re-entrancy. If not we exit SAVE back to the scheduler. If we must save our data we calculate the data area required to save $C5 thru $E5, our normal A, Q and I plus the data list specified by FLIST. The Q register is set equal to the old FORTRAN level FLEVEL and the A register is set equal to the volatile region associated with the old level (FTOP). Volatile is then requested. The return address for SAVE is stored in the third word of volatile and the new FLEVEL and FTOP are set up. Interrupts are enabled and the FORTRAN scratch area is now saved in volatile. The conditions on entry are reset and exit is made.

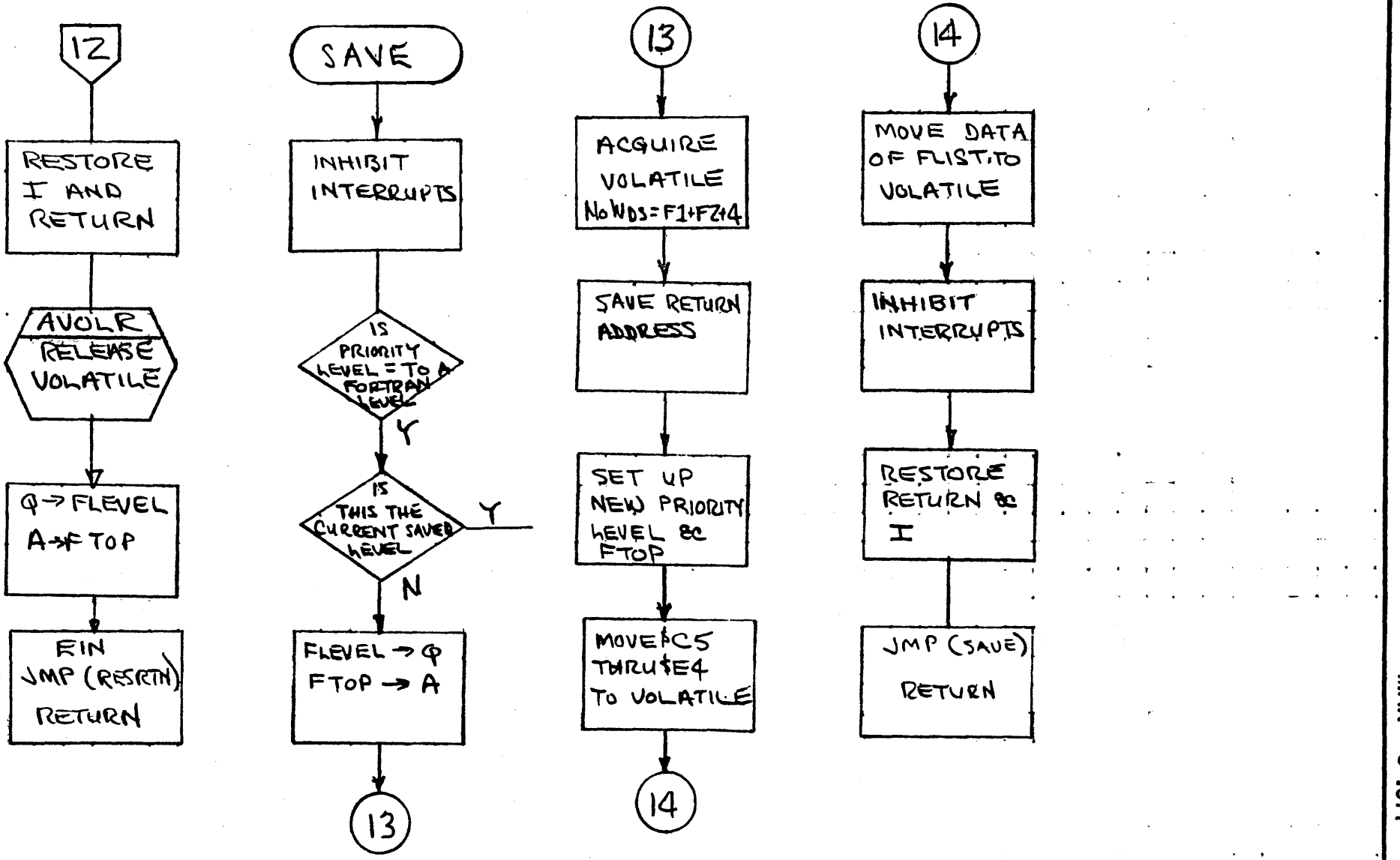
Upon entering the dispatcher control is passed to RESRTN. This routine will restore the communications region or scratch area for FORTRAN if a FORTRAN program has just terminated. If no other FORTRAN programs have their data stored no action is taken and the scratch area remains the same. If the program which just terminated was not a FORTRAN program no action is taken.

If the dispatcher determines the highest priority program is on the scheduler thread control is passed to SAVE. If the program to be executed is a FORTRAN level we must save the FORTRAN scratch as described above, otherwise, control is returned to the scheduler operations.



SEE SECTION 3 AND 4 FOR DETAILED FLOWCHARTS OF THE DISP AND SCHED.

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE REENTRANT FORTRAN		PROJECT MGR.		
	DISP & SCHED. ROUT.	PAGE 1 OF 2	PROJECT NAME MSOS 2.1		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE REENTRANT FORTRAN	PROJECT MGR			
	SYHED & DISP ROUTINES PAGE 2 OF 2	PROJECT NAME MSOS 2.1			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

Page 5.4

AA1385 FORMER CA127

PRINTED IN U.S.A.

6.0 COMPLETE REQUEST FOR DRIVER ROUTINE

6.1 SYMBOLS

COMPRQ Entry Point

6.2 FUNCTION

The functions of this subroutine are to initiate completion requests to the Scheduler for threaded I/O requests and to perform other housekeeping details upon completion of an I/O action by an I/O device driver.

6.3 ENTRY INTERFACES

COMPRQ is entered via a return jump with the physical device table address for the device in I.

6.4 EXIT INTERFACES

The contents of the I register are not disturbed. The contents of the A, Q and Overflow registers are destroyed. Interrupts are enabled.

6.5 INTERNAL DESCRIPTION

The routine is entered from an I/O device driver via a Return Jump to COMPRQ. Interrupts are immediately inhibited.

The Diagnostic Clock cell in the Physical Device Table is set idle.

For logical units which do not share devices, the completion address, if not zero, is scheduled with the error field from Word 9 of the Physical Device Table replacing the V field of the I/O request parameter test. If the call was a system directory request the V field is not stored in the parameter list since the system directory entries do not have a V field. Bits 14 and 15 of Word 8 in the Physical Device Table are set to ▽zero▽ and an indirect secondary scheduler call is made. The request parameter list, which contains a request code designating it an I/O call, is flagged as a secondary schedular call by setting bit 15 of the first word {field I} to ▽one.▽ The scheduler later resets it to ▽zero.▽ The device is not released from its logical unit assignment.

EXAMPLE:

Indirect call in COMPRQ

```
        54F4
        C001
```

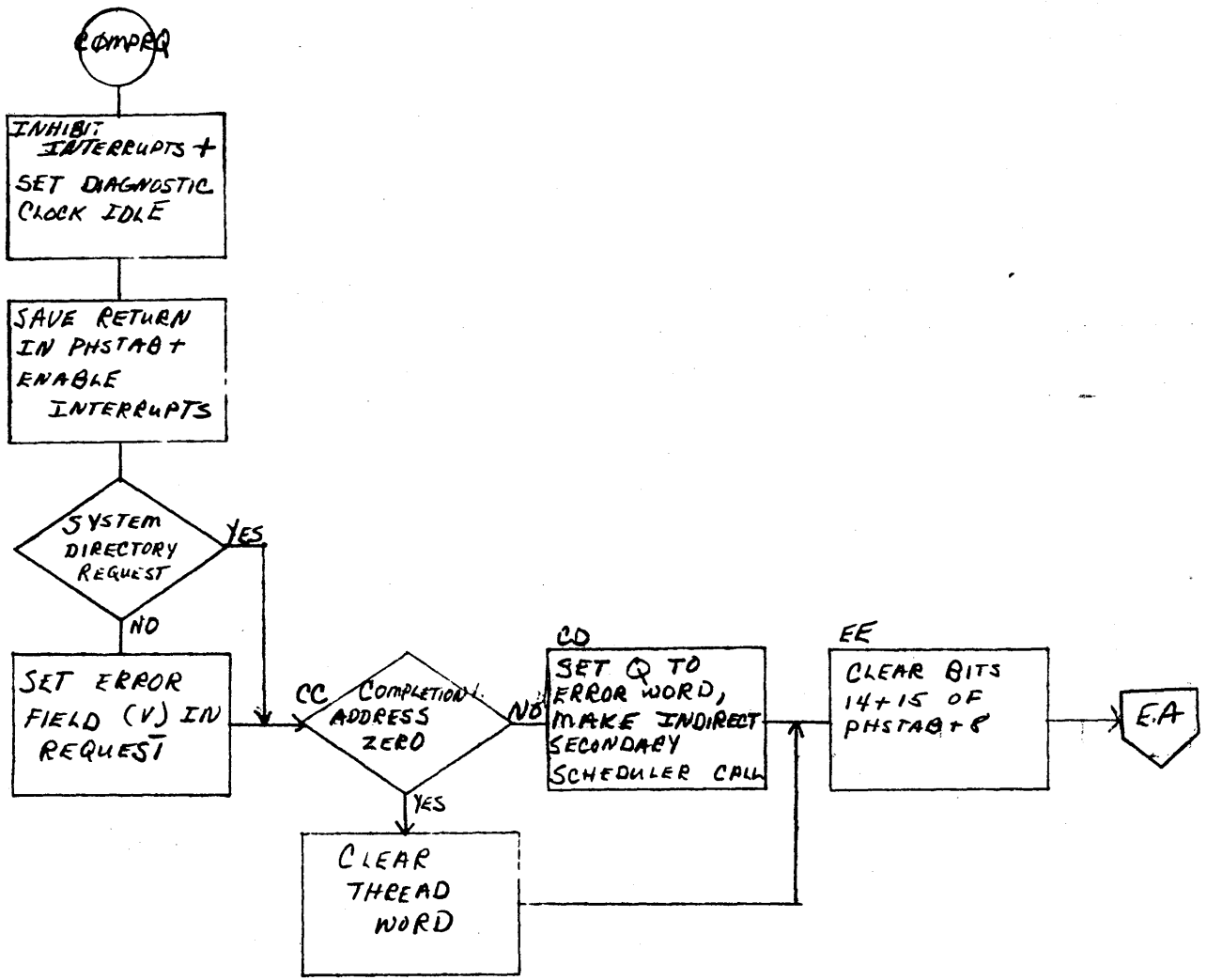
points to I/O call which is now a secondary scheduler request:

```
4000    54F4
        8802  Bit set by COMPRQ, cleared after SCHEDU threads
        5010  This address scheduled.              request.
        2000
        0008
        0016
        1000
```

If the completion address is zero, the thread word of the request is cleared and no address is scheduled.

For logical units which share devices, completed requests are treated like requests to ordinary logical units. The device is then assigned to a pseudo logical unit, $FFFF {see section 7.4}.

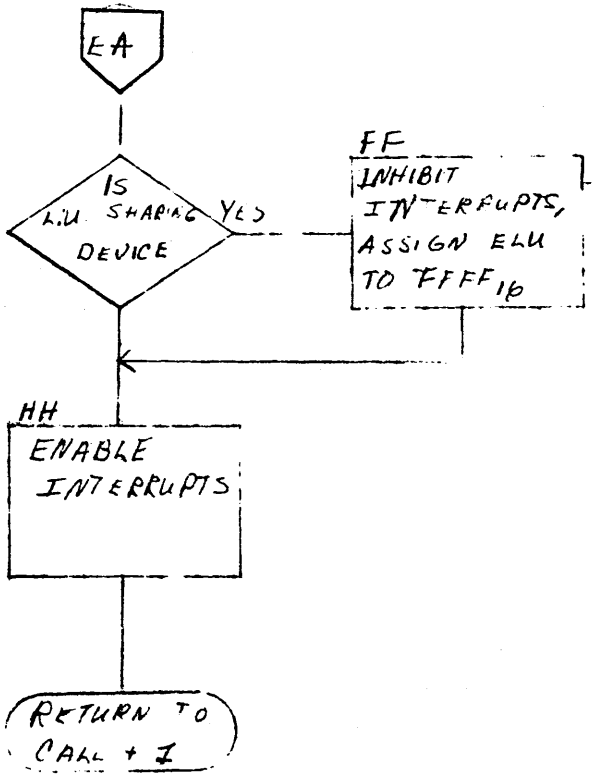
The subroutine exits to the location following the return jump which called it.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE COMPILE REQUEST		PROJECT MGR.		
FLOWCHART ☐	ROUTINE	PAGE 1 OF 2	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

6.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IM~ MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE COMEBACK REQUEST	PROJECT MGR			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

CA1385 (FORMERLY CA127-)

PRINTED IN U.S.A.

7.0 FIND NEXT REQUEST FOR DRIVER (FNR)

7.1 FUNCTION

The function of this subroutine is to find the request which should be processed next by a driver for a device. FNR also stores the information in the physical device table concerning the request that is common to all drivers.

7.2 ENTRY INTERFACES

FNR is entered via a return jump to entry point FNR with the physical device table address in I.

7.3 EXIT INTERFACES

If there are no more requests for action by a device, the subroutine returns to the driver at call +1. The device has been made unassigned by storing a zero in word 5 (ELU) of the physical device table.

If FNR has found a request it returns to the driver at call +2. The I register is undisturbed. The A & Q registers are not restored. The following information has been stored in the physical device device table by FNR:

1. Operation in progress bit is set in word 8.
2. Address of the I/O parameter list is set in word 6.
3. Word 5 (ELU) remains assigned to the same logical unit or to a logical unit which shares the device.
4. Words 10 and 11 contain the first word address and the last word address +1.
5. Word 9 (switch word) is cleared except for the mass memory bit (if set) and bits 0 and 1 which indicate whether the type of operation is ASCII or binary, formatted or unformatted.

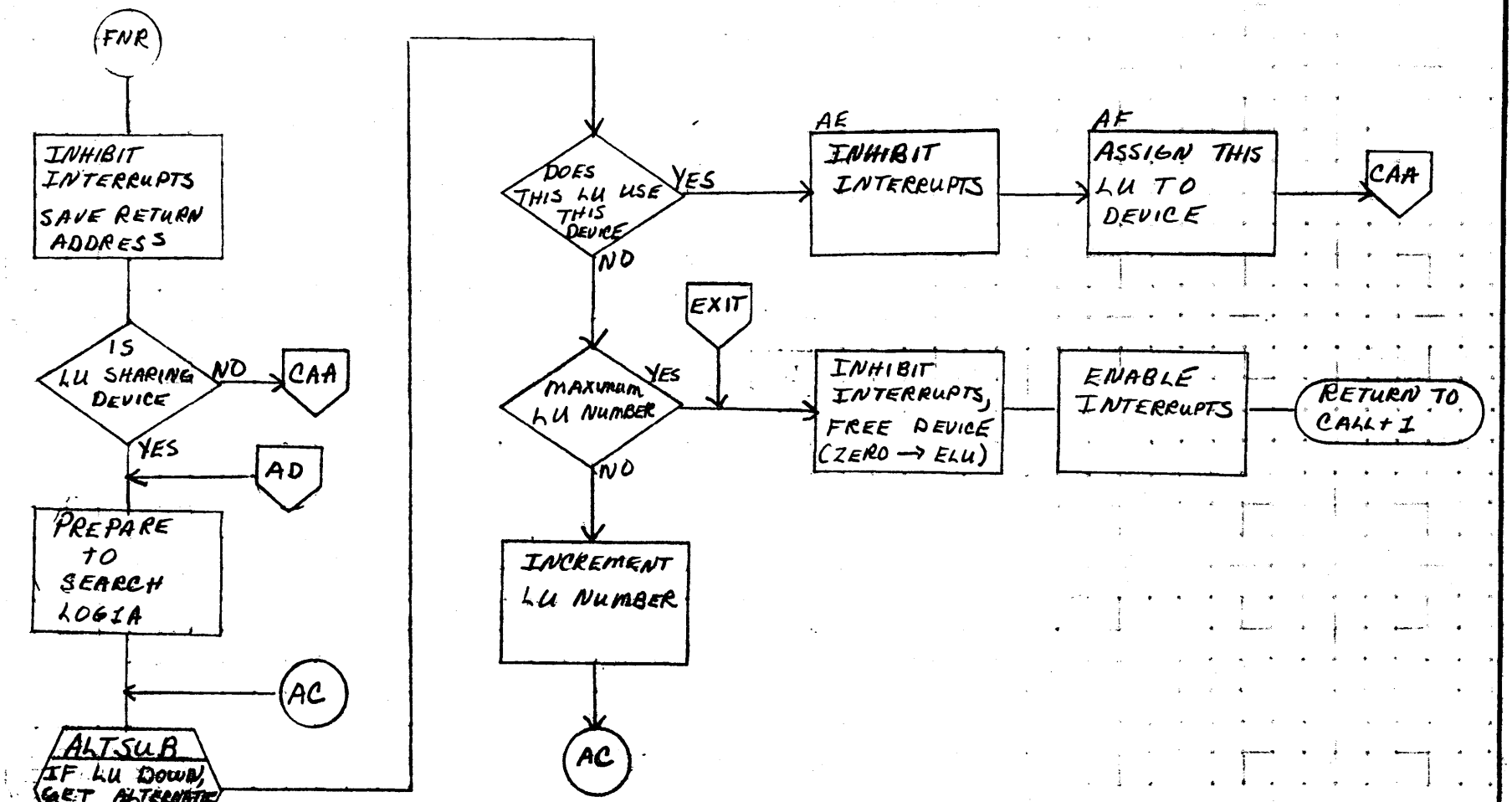
7.4 INTERNAL DESCRIPTION

For logical units that do not share a device, FNR examines the thread in the LOG2 table associated with the logical unit (L.U. assigned by RW before it schedules the driver) to obtain the next request. If there are none, FNR exits to call +1.

The top request on the LOG2 thread is removed and its parameter list address, the first word and last word +1 addresses, the operation-in-progress bit, and the type of operation bits are stored in the physical device table. Exit is to call +2.

If the request was a system directory I/O call {request code of zero}, the first word and last word +1 are computed from the system directory entry. The type of operation bits are not set. Exit is to call +2.

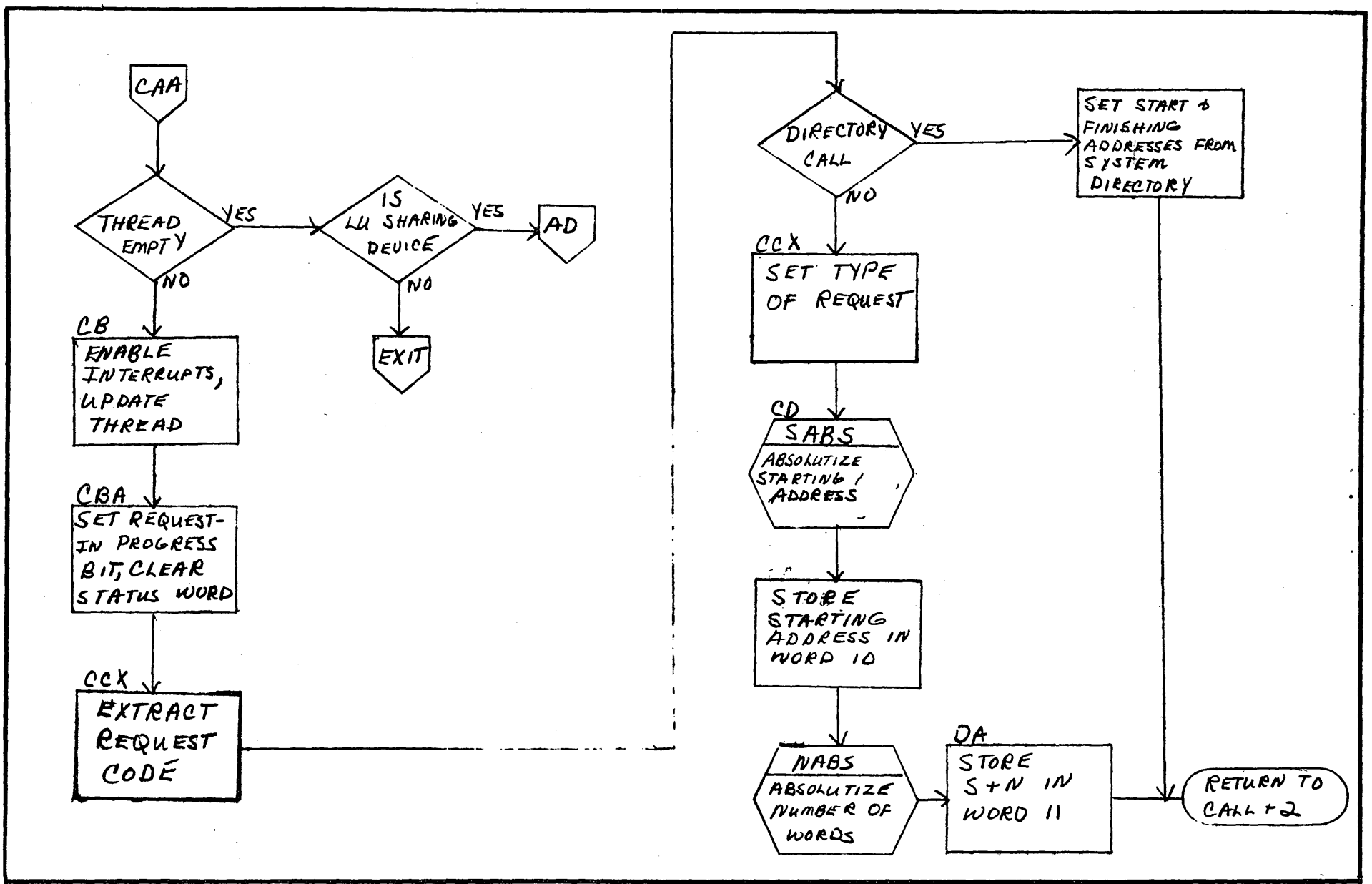
If the device is shared by several logical units, the COMPRQ subroutine has set word 5 {ELU} of the physical device table to $FFFF. Upon finding that a device is assigned to $FFFF, FNR searches the LOGLA table for the highest priority {i.e. the lowest number} which requires the available device. When a logical unit with a waiting request is encountered, FNR places the device into operation in the same manner as for unshared devices. This provides sharing of devices by several user routines. However, once a request to a device is started it will be completed before a request of higher priority to the same device can be initiated.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FIND NEXT REQUEST	PROJECT MGR.			
	PAGE 1 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

7.2



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS Ims MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FIND NEXT REQUEST	PROJECT MGR			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

7.3

CA-185 (FORMERLY CA127)

PRINTED IN U.S.A.

8.0 ALTERNATE DEVICE HANDLER

8.1 Function

The Alternate Device Handler is responsible for processing irrecoverable errors for many of the drivers.

Upon entry, it determines whether the device has an operational alternate. If an alternate exists, the request for the failed device is assigned to the alternate and the operator is notified of the switch-over. This re-assignment is done at a high priority level. However, if no alternate exists, the program reschedules itself at a low priority to request operator intervention. In either case, all message output is executed from a low priority section of the program.

The Alternate Device Handler continues to assign alternate to failed devices without waiting for completion of its message I/O. This requires that the table ALTERR be provided to store the error words in for processing by the low priority section.

8.2 Entry Interfaces

Entry is made via a jump to the entry points DEVERR, ADEV, or ALTDEV. These names are equivalant and can be used interchangeably. On entry Q should contain the following information:

Bits 0-5 Error code
- 0 - timer expired
- 1 - reject
- 2 - alarm
- 3 - parity error
- 4 - checksum error
- 5-X - other error codes as defined for driver

Bits 6-15 Logical Unit Number associated with the the requested device.

NOTE: The logical unit number in the error word is formed by the driver of the device using the device's ELU word from the PHSTAB. Since the RW and FNR routines store the requested logical unit number in ELU, the L.U. in the error word reflects the L.U. whose request is being processed by this device when it fails and not necessarily the L.U. assigned to this device.

e.g., if the device assigned to L.U. 2 has failed previously and L.U. 8 is the alternate of L.U. 2:

1. The error word will contain L.U. 2 if L.U. 8 was processing a request for L.U. 2 when the device failed.

2. The error word will contain L.U. 8 if the device failed while processing a request for L.U. 8.

8.3 Entry Symbols

ALTDEV ADEV DEVERR	Alternate Device Handler Entry from Drivers
ALTSUB	Find Alternate Logical Unit Subroutines
CONVER CONVRT	HEX to ASCII Conversion Subroutine

8.4 External Symbols

JBCNCL	Core resident program which causes the JBKILL module of the job processor to be executed.
JOBIND	A location in the TRVEC module which is non-zero if job processing is in progress.
ALTERR	A table used to save the error word in case several failures occur at one time. This table is included in the System Tables and Parameters program and is of the form:

```
          ENT     ALTERR
ALTERR    ADC     NUMLU     ALTDEV ERROR TABLE SIZE
          BZS     {NUMLU}   SPACE FOR {NUMLU}
                              SIMULTANEOUS FAILURES
```

LOG1A LOG1 LOG2	Logical Unit Tables

DUMALT — Logical Unit Number of the Dummy Device Driver.

Included in the System Tables and parameters program as an EQU entry.

```
e.g., EQU DUMALT {SDA-LOG1A}
      ENT DUMALT
```

If the Dummy Driver is not present, set to zero

```
e.g., EQU DUMALT {0}
      ENT DUMALT
```

and remove the entries from the LOG1A, LOG2, and LOG1 tables.

MAS300 — Entry in the Mass Memory Driver Control program. It will check if any mass memory drivers are waiting to use core.

8.4 Initial Operations

The Alternate Device Handler is entered at the priority of the driver, or at the level specified if scheduled into operation. The latter method is used if the driver must continue after calling the Alternate Device Handler. In either case the Alternate Device Handler reschedules itself at a high priority level determined by the symbol LEVEL.

LEVEL should be equated to a value one greater than the highest priority of any driver using the Alternate Device Handler.

A check is then made for space in the error word table. If the next location in the table is not empty {i.e., zero} the size of the ALTERR table is not large enough. This error is irrecoverable and the Alternate Device Handler will hang in a loop.

8.5 Operations When No Alternate Exists

The low priority section {NOALT} is scheduled at level 4 and the error word is stored in the ALTERR table. Exit is made to the dispatcher.

8.6 Operations when the failed device is an alternate or Has an alternate

If the requested L.U. specified in the error word is already down, a search is made for the alternate that actually failed and that L.U. is stored in the error word. The failed L.U. is then marked down and made a shared device (i.e., bits 13 and 14 of LOG1 are set = 1) and a check for an operational alternate is made.

If no alternate of the failed L.U. exists, the LOG1 entry of the failed L.U. is restored to what it was on entry to ALTDEV. The low priority section is scheduled, the error word is stored in ALTERR, and exit is made to the dispatcher.

If an operative alternate of the failed L.U. is found, the request is rethreaded to the LOG2 thread of the requested L.U. The ELU word of the L.U. that actually failed is cleared and the operational alternate is made a shared device (i.e., bit 14 of LOG1 is set = 1).

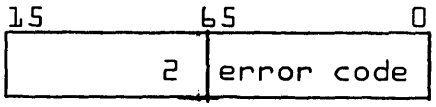
A search of the ALTERR is made and if the error is already in the table the error word is cleared (i.e., set to zero).

If the operative alternate device is not busy (i.e., ELU word = zero), the alternate's driver is scheduled via its PHSTAB and made busy by storing the requested logical unit number in the alternate's ELU word.

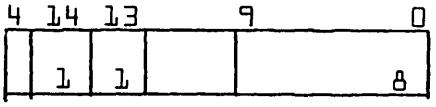
Then the low priority I/O section (NOALT) is scheduled if not busy and the error word, if not zero, is stored in the ALTERR table and exit is made to the dispatcher.

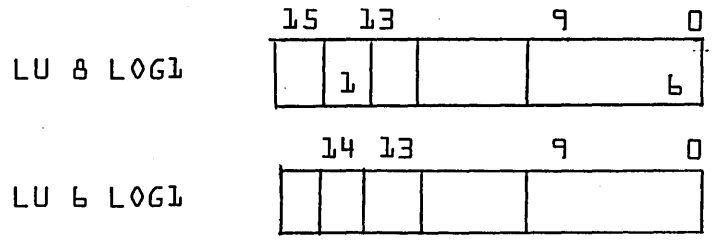
EXAMPLE: On entry to ALTDEV, if L.U. 8 failed while processing a request for L.U. 2 the error word and LOG1 entries would look like this (L.U. 6 is the alternate of L.U. 8):

ERROR WORD

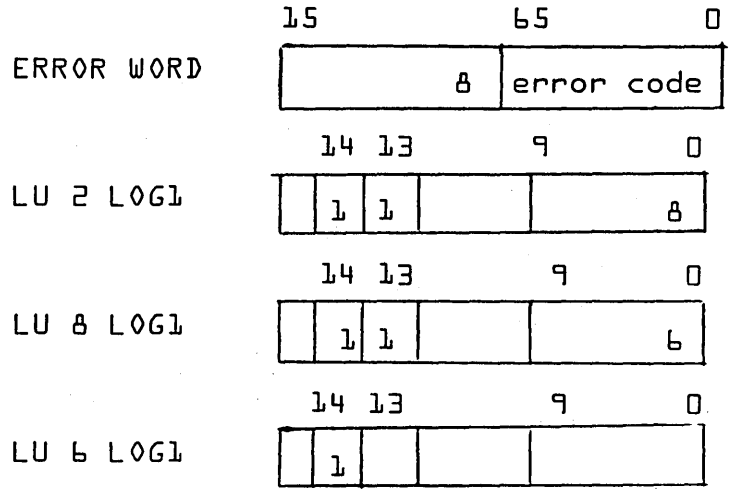


LU 2 LOG1





On exit from the high priority section of ALTDEV, the entries will now look like this:



The request has been rethreaded to L.U. 2 LOG2 thread and the ELU word in the PHSTAB of L.U. 6 will contain L.U. 2 {if the L.U. 6 driver was not busy}.

8.7 Low Priority I/O Section {NOALT}

All I/O requests are executed in this section at priority level 4. This allows several alternates to be re-assigned even before the first message has been output. The NOALT section picks up one entry at a time from the ALTERR table, processes it, and then returns to process further entries until none remain in the ALTERR table. At this time NOALT clears its busy flag and exits to the dispatcher.

Each error word from a driver to ALTDEV is stored in successive positions in the ALTERR table. When the top of the table is reached the indexes are set back to the bottom of the table. If the next location in the table is not empty {i.e., zero} the size of the ALTERR is not large enough.

NOALT informs the operator of errors and interrogates the operator for guidance when there is no alternate available.

If an operative alternate exists, the following message results:

```
L,nn FAILED ee
ALT,mm
```

If no alternate is assigned, the following message results:

```
L,nn FAILED ee
ACTION
```

followed by an input request to allow the operator to specify the action to be taken.

nn = Logical unit that failed {decimal}
ee = Error code passed from driver {decimal}
mm = Logical unit of alternate {decimal}

All Input/Output is via the Comment Device.

If the input request following the ACTION message is completed with error indication {e.g., a timeout occurred} then the RP option is assumed. The ACTION and input request cannot be repeated otherwise the ALTERR table may be filled with error word entries corresponding to the input comment device failure.

If no alternate exists the operator is requested to specify further action. His options are as follows:

RP — Repeat the request - the current request is rethreaded and the initiator portion of the driver scheduled.

CU — Continue - the completion address is scheduled, the error is reported to the caller and the driver initiator rescheduled.

CD — Continue and mark the device as ▽down▽. This option causes all completion addresses for requests in the queue to be scheduled with error indicators in Q. In addition, the device is marked ▽down▽ by setting bit 13 in LOG1 table. This results in scheduling completion addresses with error codes for any subsequent request for this device.

DU — Delete the job - if job processing is not in effect this option will cause repetition of the action request printoutM. If job processing is in effect, the job processor job cancel entry is scheduled at level two. All the actions of the CU option are then executed.

MJob processing is not in effect when LIBEDT or RECOVERY modules are in control.

DD — Delete the job and mark the device as "down". This action will result in a repeat of the action request printout if job processing is not in effect. The job processor job cancel entry is scheduled at level 2. All the actions of the CD option are then executed.

8.9 Subroutines

ALTSUB is the subroutine used to find the operational alternate for a given logical unit. The logical unit is in Q on entry and if this logical unit is operational {not marked down} then Q returns unchanged. Otherwise, Q will be set to the logical unit of the first operational alternate assigned. If no alternate is assigned Q will be zero. If the only alternate is the device that just failed, then Q will be set to DUMALT on return. The value of DUMALT is either the logical unit of the Dummy Driver or zero, and is set at Initialization. {See 8.4}

The ALTSUB subroutine is called from the following programs:

RW Read-Write Request Processor
FNR Find Next Request Subroutine

Note that ALTSUB allows alternates to be assigned in a circular arrangement

e.g., L.U. 6→7→8→9→10→8

CONVER {=CONVRT} is a hexadecimal to ASCII coded Decimal conversion subroutine. The hexadecimal value in A is converted and returned to A. It is restricted to a maximum of two decimal digits {i.e., 99_{10}} and is used to convert the logical unit and error code for printout. The subroutine is re-entrant and may be used by other programs.

8.10 Dummy Driver

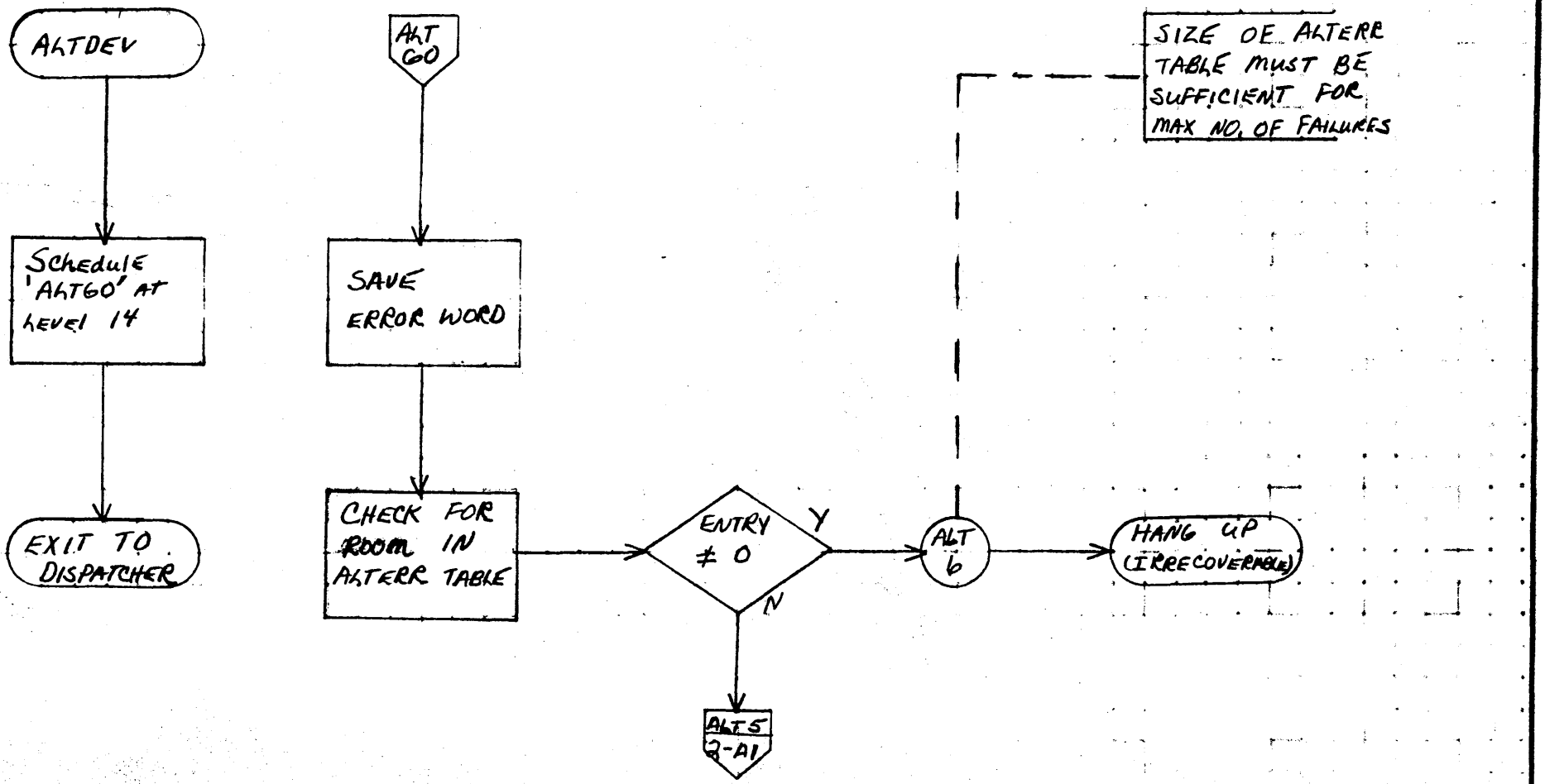
If a device fails and no alternate is available, the dummy device can be assigned as the alternate. This causes the Dummy Driver to be scheduled via the Dummy Equipment Table. The Dummy Driver picks up the request using the FNR subroutine and sets the error field in the request. Then the logical unit that failed is restored by clearing the "device down" bit in the LOGL logical unit table. Finally the COMPRQ subroutine is entered to complete the request and then returns to the Dummy Driver Initiator to re-enter FNR and process any further requests on the thread.

The Dummy Driver allows a user program to make an I/O request and ensure that the completion address will be scheduled without requiring operator intervention. Otherwise, the user program could be suspended indefinitely. The user program must check for I/O errors at the completion entry and take appropriate action if an I/O error occurred (i.e., if $Q_{15}=1$).

The Dummy Driver and Dummy Equipment Table must be included in the System Tables and Parameters Program if this feature is required. A logical unit entry must also be assigned to the Dummy Driver in the logical unit tables, LOGLA, LOGL and LOG2. This logical unit number can then be assigned as the alternate in the LOGL entry for the logical units that have no other alternates. See section 2.13.

DUMALT is the logical unit number assigned to the Dummy. If the Dummy Driver is not included DUMALT should be equivalenced to zero. DUMALT is assigned as the alternate when all alternates are marked down such that the failed device is assigned as its own alternate.

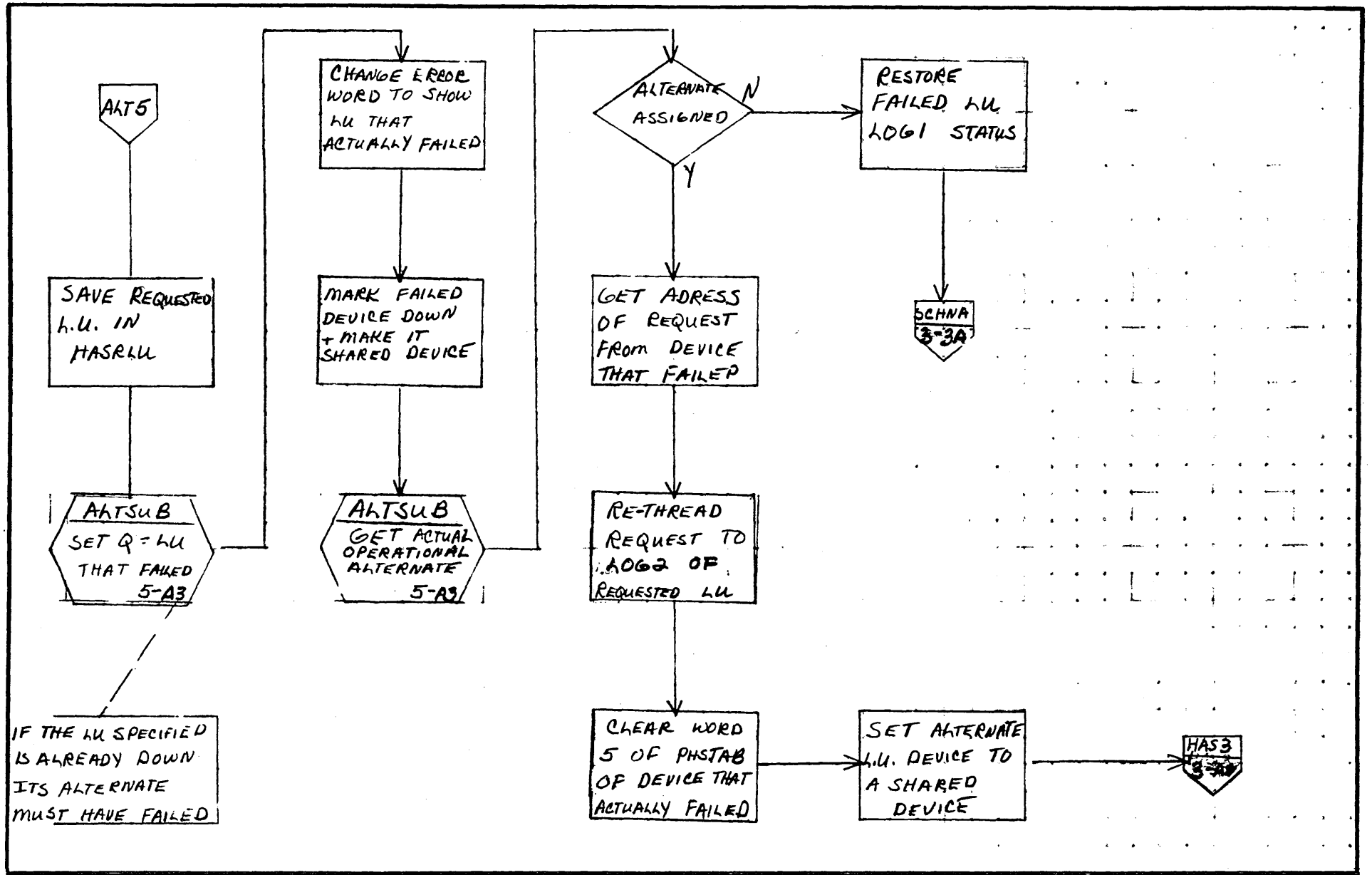
The dummy may also be used as the alternate for the comment device. Comment device failures are not recoverable unless an alternate is assigned for the comment input device.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3.0 IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALTERNATE DEVICE	PROJECT MGR.			
	HANDLER PAGE 1 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

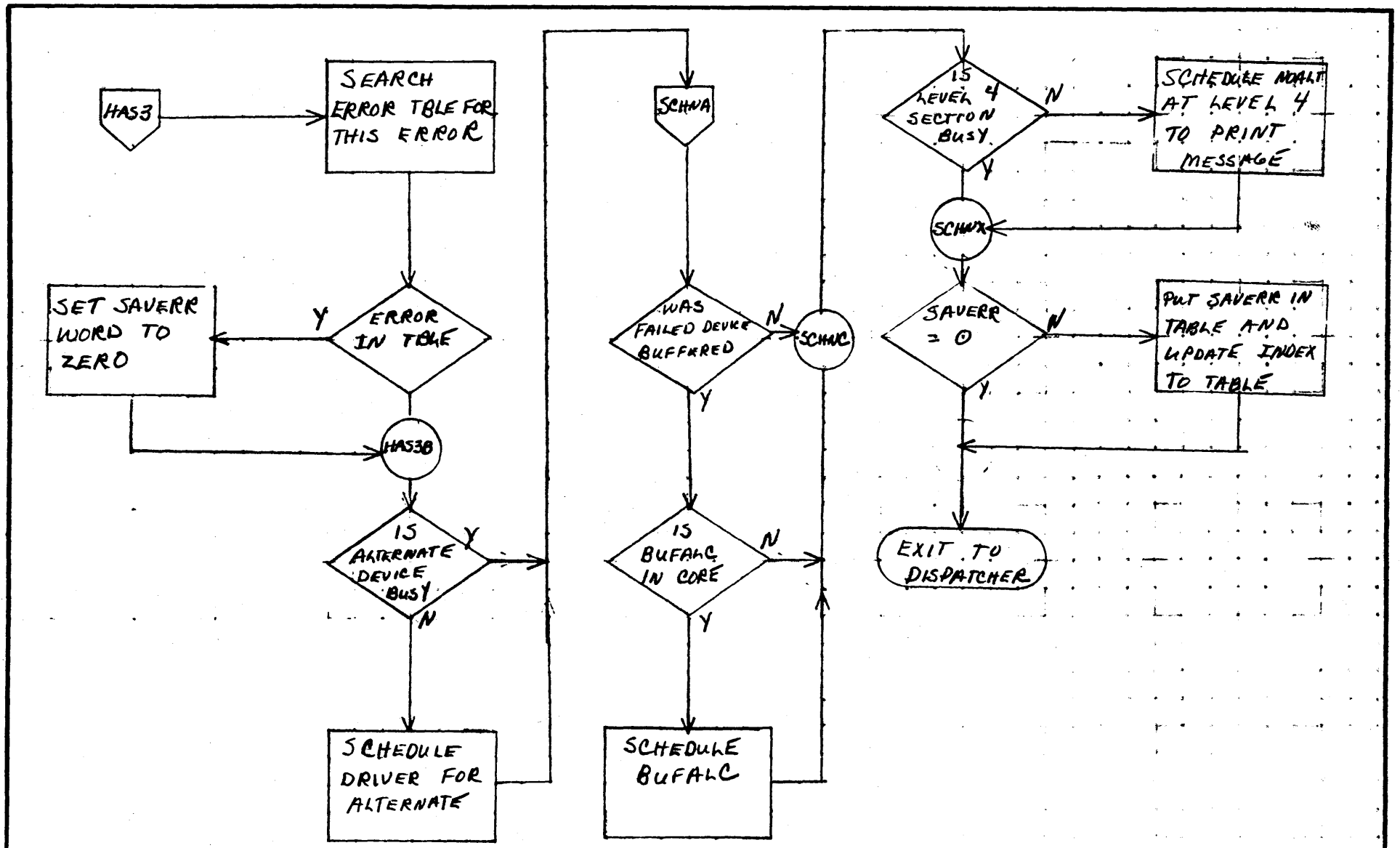
B-5



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3.0 IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ALTERNATE DEVICE HANDLER	PAGE 2 OF 10	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

8.10



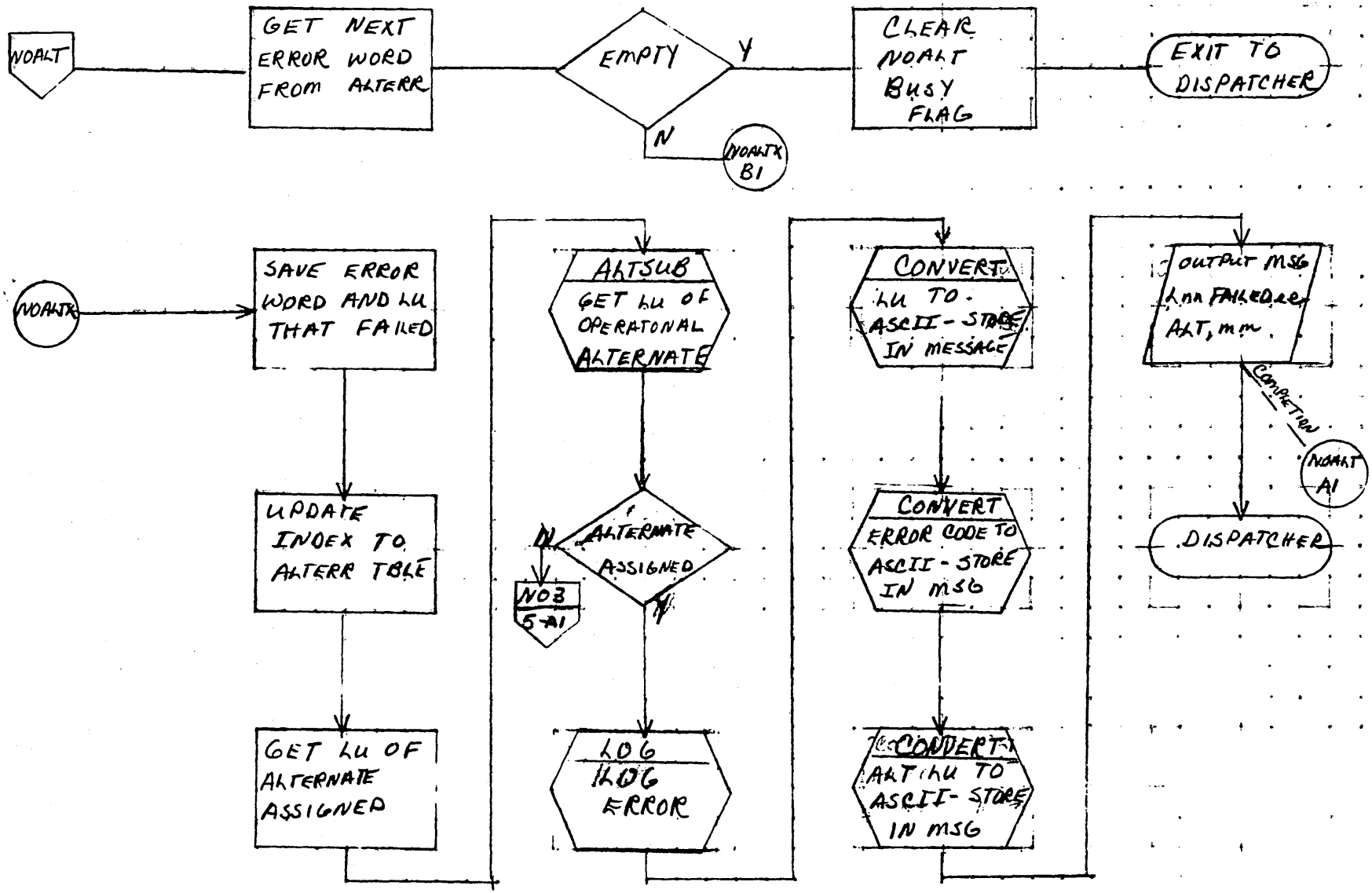
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS 3.0 IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ALTERNATE DEVICE		PROJECT MGR.		
FLOWCHART ☐	HANDLER	PAGE 3 OF 10	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

B.11

AA1385 (FORMER CA127-1)

PRINTED IN U.S.A.



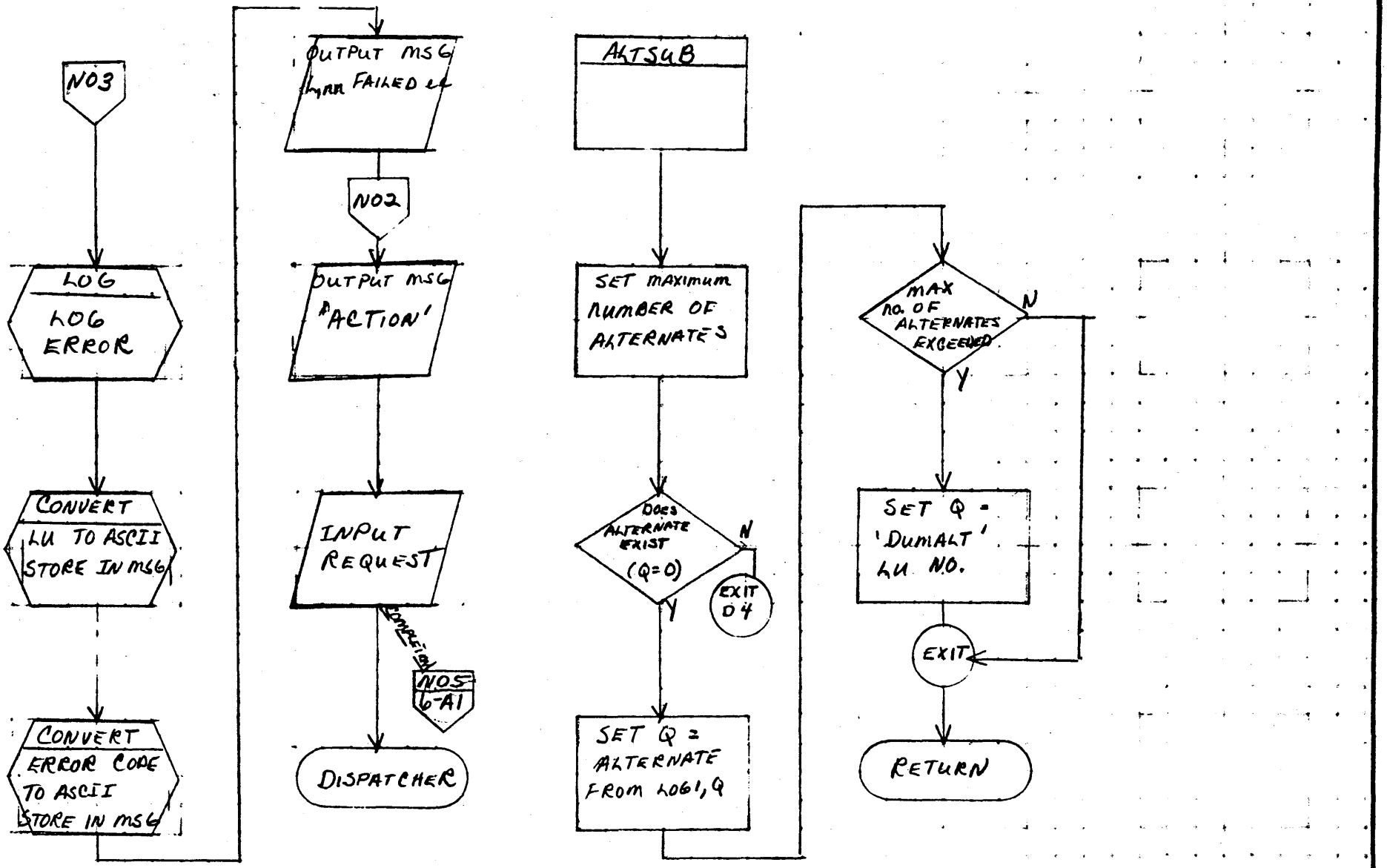
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS 3.0 IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ALTERNATE DEVICE		PROJECT MGR.			
FLOWCHART ☐	HANDLER	PAGE 4 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

8.12

AA1385 (FORMERLY CA127-1

PRINTED IN U.S.A.



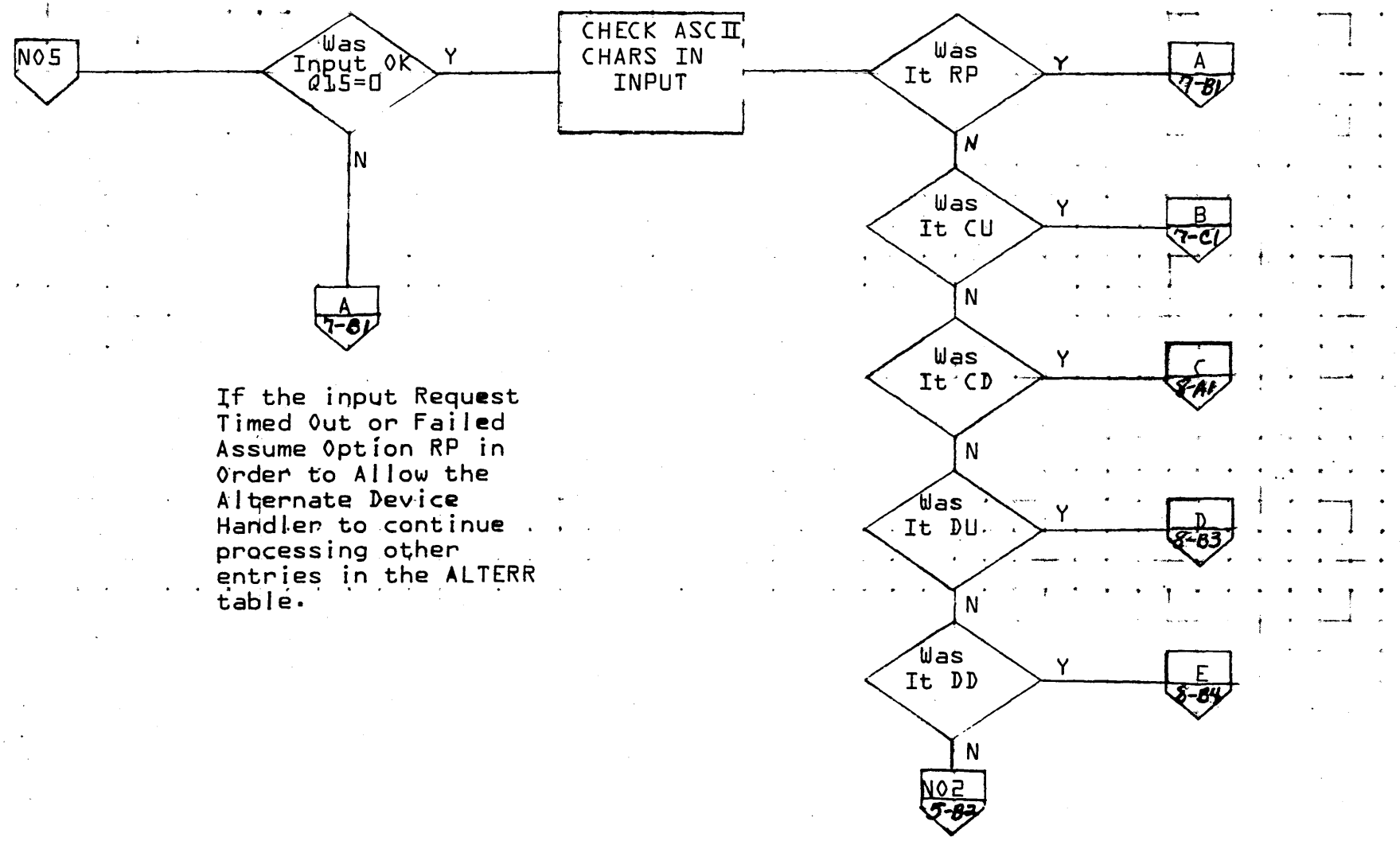
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3.0 IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALTERNATE DEVICE HANDLER PAGE 5 OF 10	PROJECT MGR.			
	NUMBER ISSUE DATE	PROJECT NAME			
	DRAWN BY DATE	TASK NO.			
		TASK NAME			

MAR 5 1971

D.TE

AA1385 (FORMERLY CA127-1)

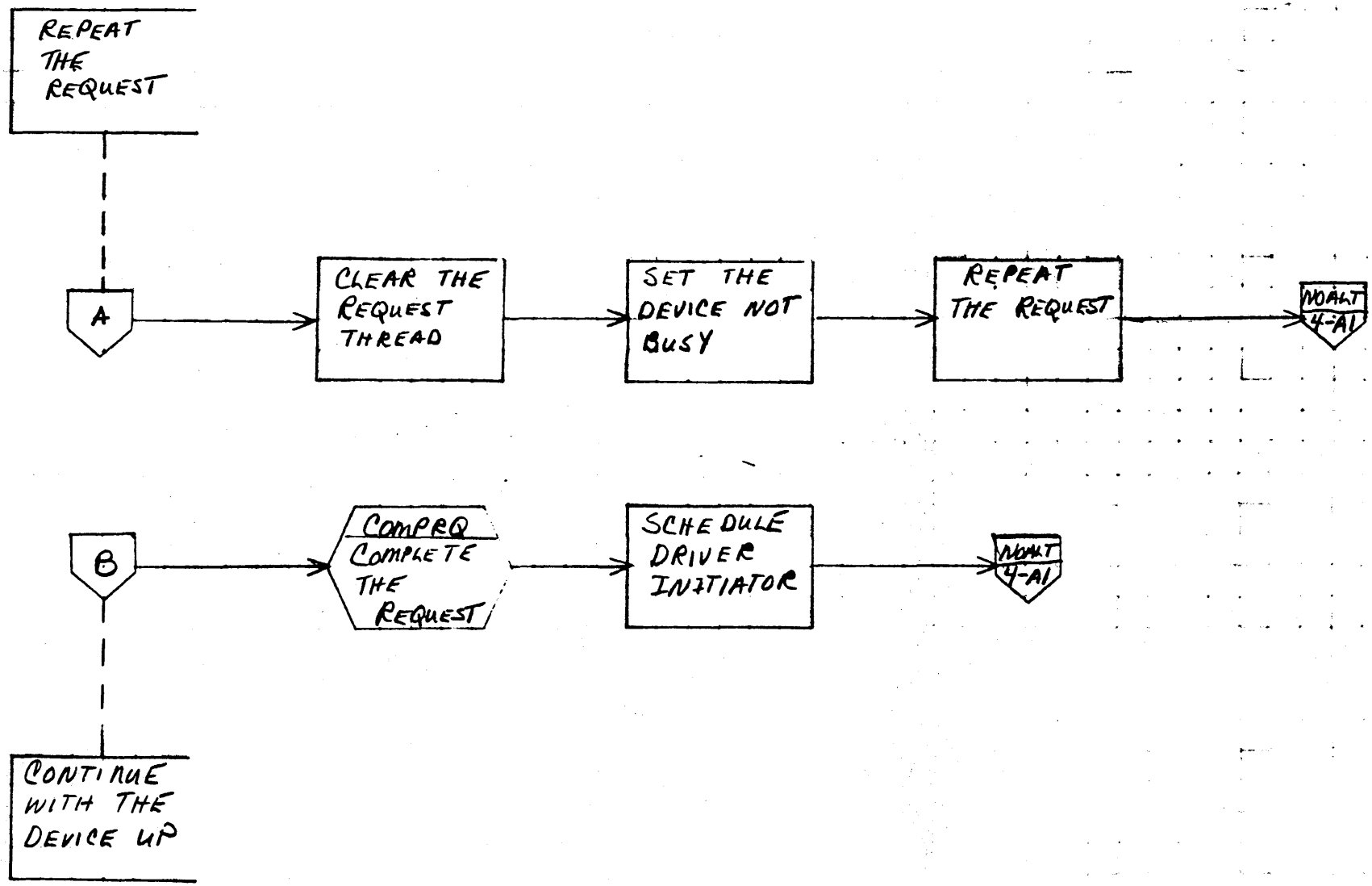
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3.0 IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALTERNATE DEVICE HANDLER PAGE 6 OF 10	PROJECT MGR.			
		PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

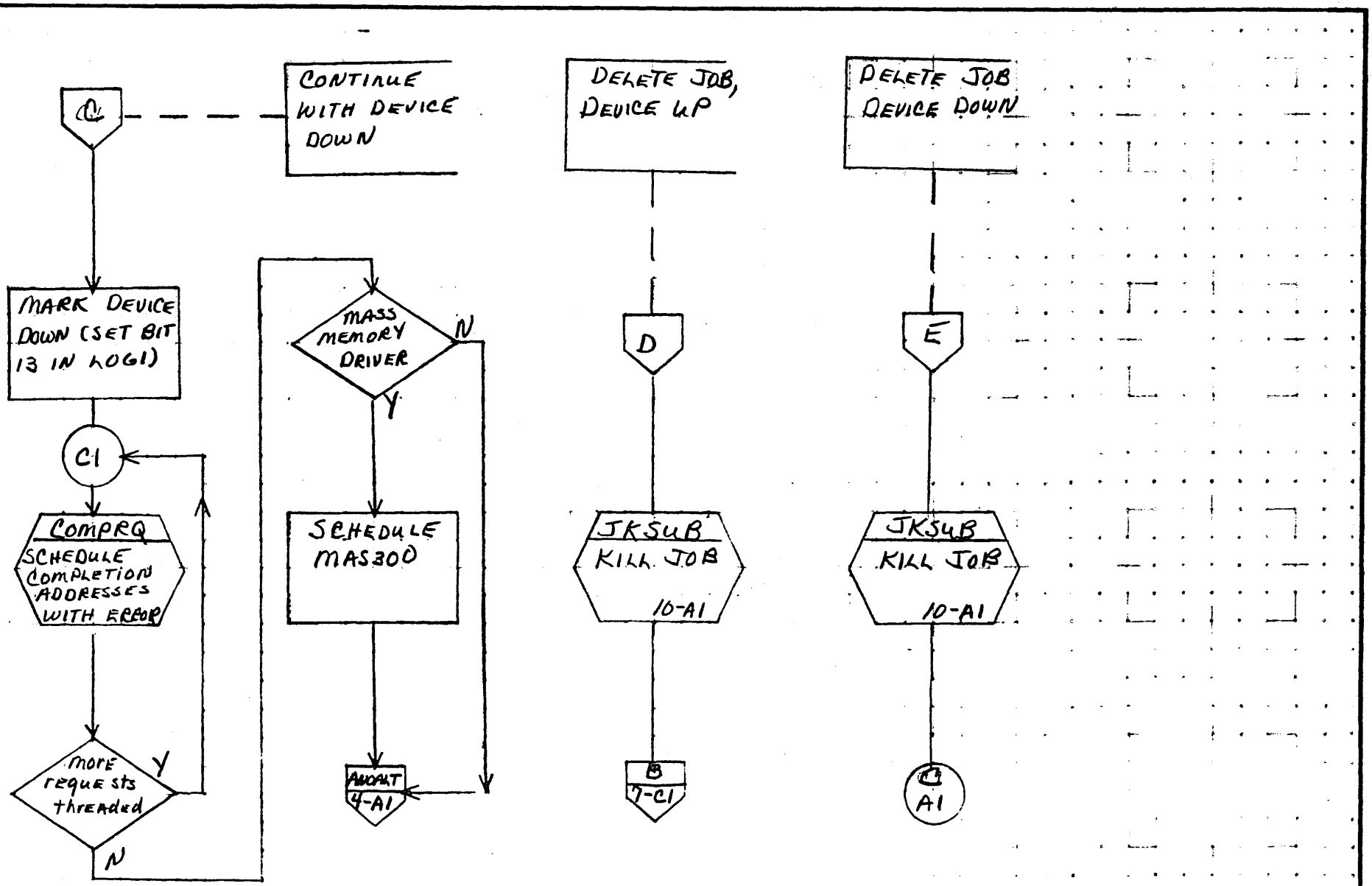


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS 3.0 Ims MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ALTERNATE DEVICE	PROJECT MGR.			
FLOWCHART ☐	HANDLER PAGE 7 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

AA1385 FORMERLY CA127-1

PRINTED IN U.S.A.

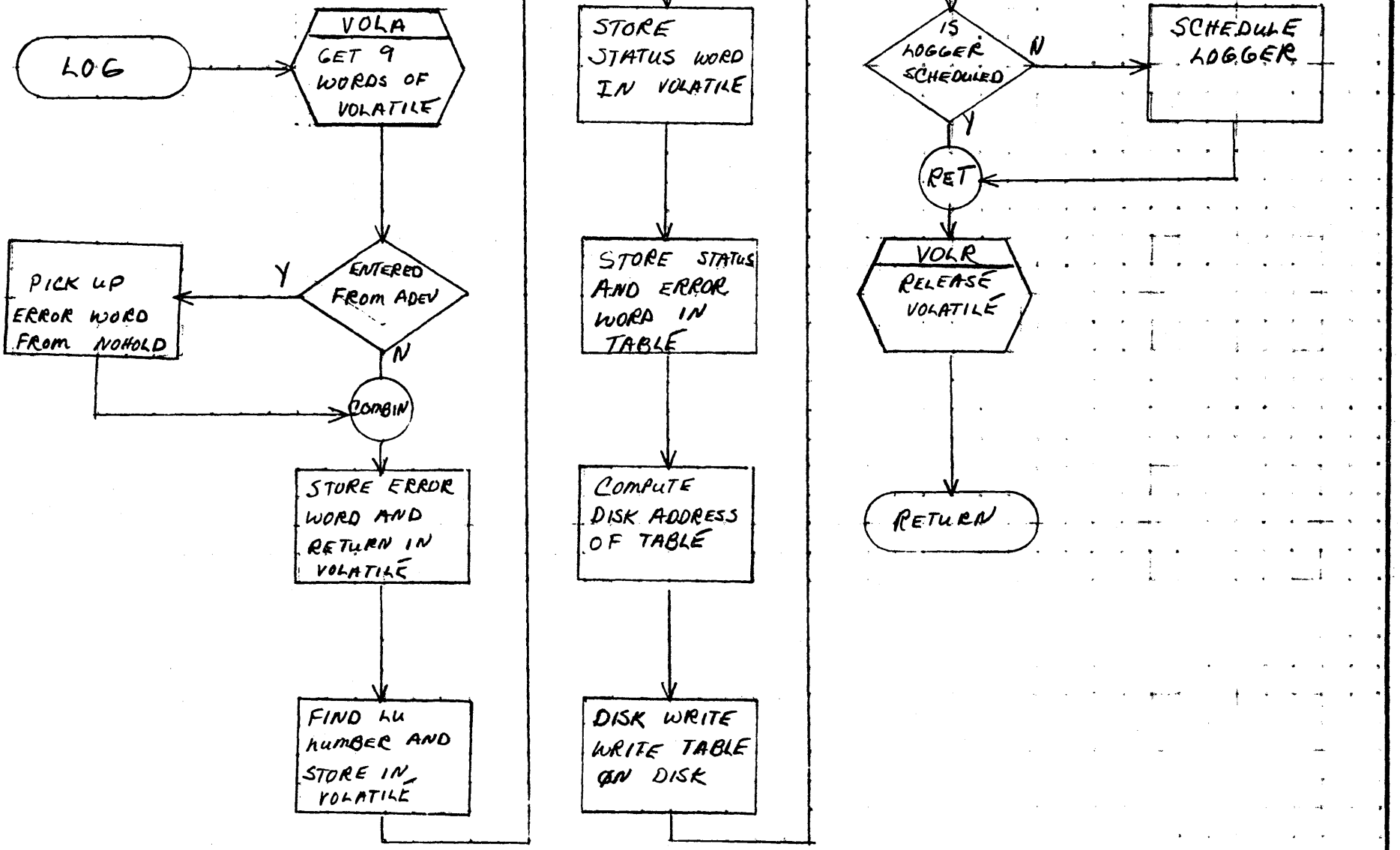
MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3.0 Ims MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALTERNATE DEVICE	PROJECT MGR.			
	HANDLER PAGE 8 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 6 1971

8451



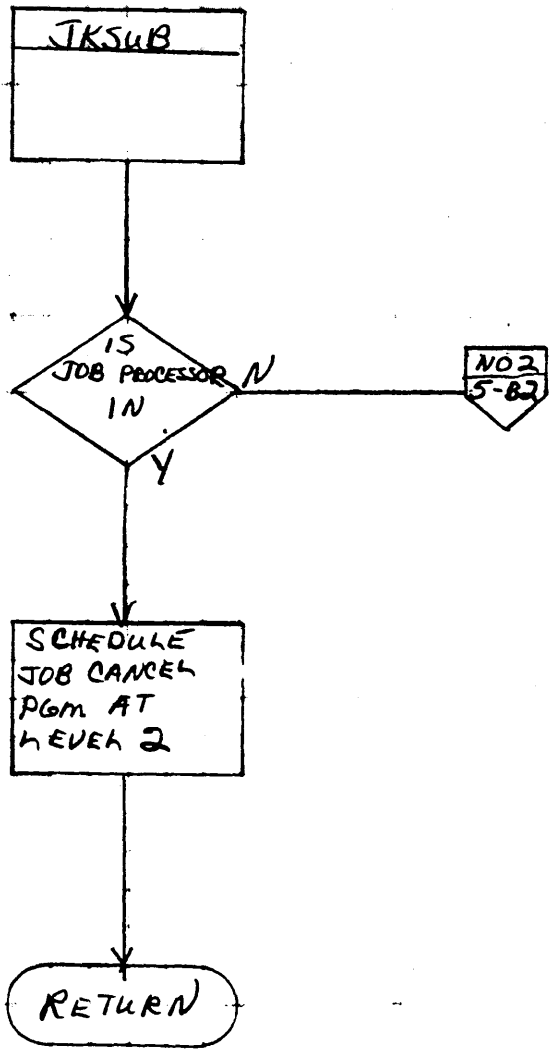
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3.0 IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALTERNATE DEVICE HANDLER PAGE 9 OF 10	PROJECT MGR.			
	NUMBER ISSUE DATE	PROJECT NAME			
	DRAWN BY DATE	TASK NO.			
		NAME			

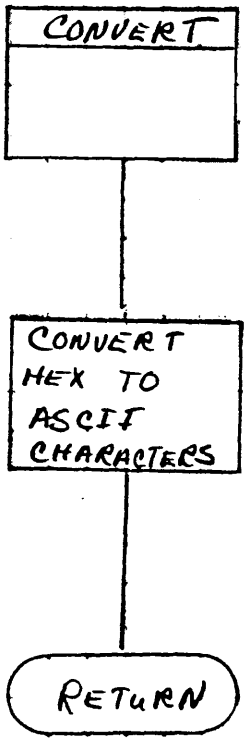
MAR 5 1971

B-16

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.





CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3.0 Ims MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALTERNATE DEVICE	PROJECT MGR.			
	HANDLER PAGE 10 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

B.17

9.0 MAKE Q FOR DRIVER ROUTINE

9.1 SYMBOLS

MAKQ Entry Point

9.2 FUNCTION

The functions of this routine are to determine if a short read and/or a device error occurred during the driver's operations. If either of these conditions has occurred, MAKQ will set the proper bits in Word 9 {switch word} of the physical device table. The COMPRQ subroutine uses these bits to form the V field of an I/O call {see section 6.5}.

9.3 ENTRY INTERFACES

MAKQ is entered with the address of the return to the driver in the A register and the physical device table address of the device in the I register.

9.4 EXIT INTERFACES

The contents of the I register are not disturbed. The contents of the A and Q registers are destroyed.

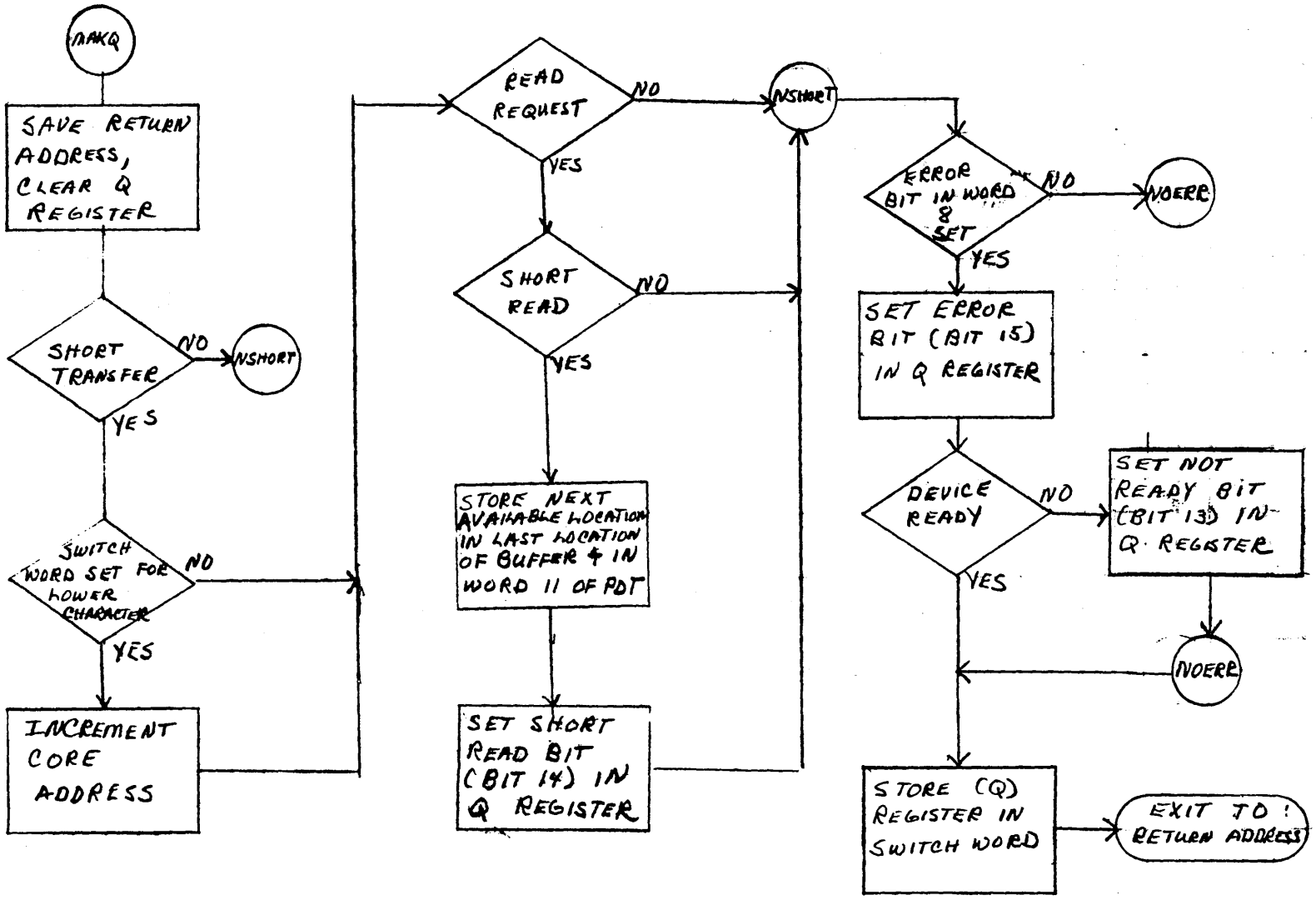
9.5 INTERNAL DESCRIPTION

On entry the return address of the driver is stored in the physical device table and the Q register is cleared.

A test for a short read is made by comparing the last location read {▽core▽ address} to the last word address +1. If, before exiting to MAKQ, the driver was expecting the next character to be a lower character, the ▽core▽ address is incremented by one before the short read test. This is necessary because the driver does not increment the ▽core▽ address until the lower character is received. If a short read condition exists, bit 14 is set in the Q register.

A check is then made for a device error condition and if true, if the the device was not ready. Bits 15 and/or 13 are set reflecting these conditions.

The final content of the Q register is stored in word 9 {switch word} of the physical device table and exit is made to the return address.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE MAKQ		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

4.2

10.0 COMMON

10.1 External Symbols

ALLIN - Entry point of COMMON

10.2 Function

This program saves the Q, A, I registers along with the return address and priority level prior to an interrupt. It adds this information to the interrupt stack and sets the mask register with a new mask and sets the new priority lever in the current priority level cell.

10.3 Entry Interface

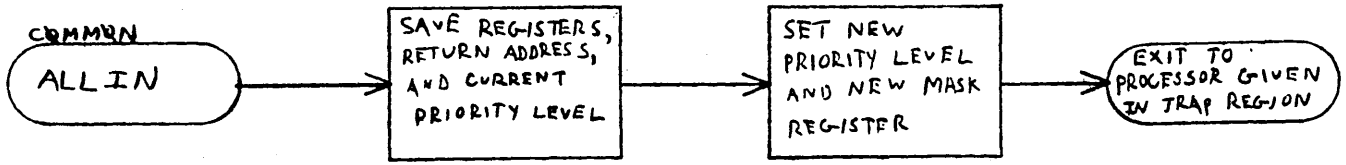
This program is entered by an indirect return jump thru location $FE. This locore location contains the address of the entry point ALLIN.

10.4 Exit Interface

This program exits to the interrupt processor chosen in the 4th word of the trap location for the line that has interrupted.

10.5 Internal Description

Upon entry the top of the interrupt stack is found and Q, A, Priority level, I are stored in the stack the address of the interrupt processor is determined and the return address is picked up and saved in the stack. The new priority level is determined from the trap region and put in location $EF. The new mask is found and put in the mask register. An exit jump is made to the processor chosen by the trap area.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: COMMON		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

10.2

11.0 NIPROC

11.1 External Symbols

IPROC	Entry point to IPROC
IP1	Address constant of IPROC1
TBLE	Used for HEX to ASCII conversion
JOBIND	Used to see if job processor is in core
SWTCH	Used to lock out job processor while LIBEDT or Recovery in core
TIMACK	Used to acknowledge timer interrupts
IPROC1	Processor in protect errors
PARITY	Processor for parity errors

11.2 Function

The function of the internal interrupt processor is to handle internal interrupts on line zero caused by parity errors in memory, memory protect errors and power failure conditions.

11.3 Entry Interface

NIPROC is entered upon an interrupt of line zero. Its address is contained in the fourth word of line zero's trap.

11.4 Exit Interface

Exit from IPROC depends upon the error condition that caused the interrupt. Protect errors exit to IPROC1 which is the protect processor. Parity errors cause a branch to PARITY which is a user supplied module to process parity errors. If this module is not present the computer is hung in a $18FF loop. For power failure errors the computer is hung in a $18FF loop until power returns at that time a EXI instruction is executed which returns control to the P+1 instruction is executed which returns control to the P+1 interrupted address.

11.5 Internal Description

After entry to IPROC a protect fault is tested for if a protect fault is found a jump to IPROC1 is made. If no protect fault is found a parity error test is made if no parity error is found a power failure is assumed. The contents of the registers are saved and a jump to the power failure routine is set up at absolute location 0 and 1. When a power failure occurs the user master clears and hits the run switch, location 0 and 1 are executed giving control to the power failure routine which in turn exits the interrupt state through the return address

12.0 External Interrupt Handler {EPROC}

12.1 Program Function

External Interrupts are handled by this routine. There are three conditions that can exist on an interrupted line. They are as follows:

1. The line is connected to devices controlled by the Operating System; i.e., they appear in the system equipment table.

2. The line is connected to devices not controlled by the Operating System.

3. The line is connected to devices, some of which are controlled by the Operating System and some of which are not.

12.2 Internal Description

Upon entrance to ▽EPROC▽ the ▽I▽ register contains the slot location through which the interrupt came. This location is changed to a line number from which the number of devices on this line can be found. If no devices are present, a check is made for the secondary processor.

If devices are present, the logical unit number of the first device is found. The number of devices is stored in the I register and volatile storage is called. Only three words are saved; they are Q, A and I where:

Q Line number interrupted
A Logical unit number of first device on line
I Negative number of devices on line

Each device is checked in turn for having the interrupt status set. When one is found, the continuator address is set in the Q register of volatile storage. Volatile storage is released and a jump made to location specified in the Q register.

If no devices on the line interrupted, volatile storage is released and a secondary processor is checked for being present. If one is present, control is passed to it; if not, the ghost interrupt message is scheduled, and control is returned to the dispatcher.

The ghost interrupt message prints GI {line no.} where line no. is a hexadecimal number. Upon completion of the printout, control is returned to the dispatcher.

12.3 Entry Interfaces

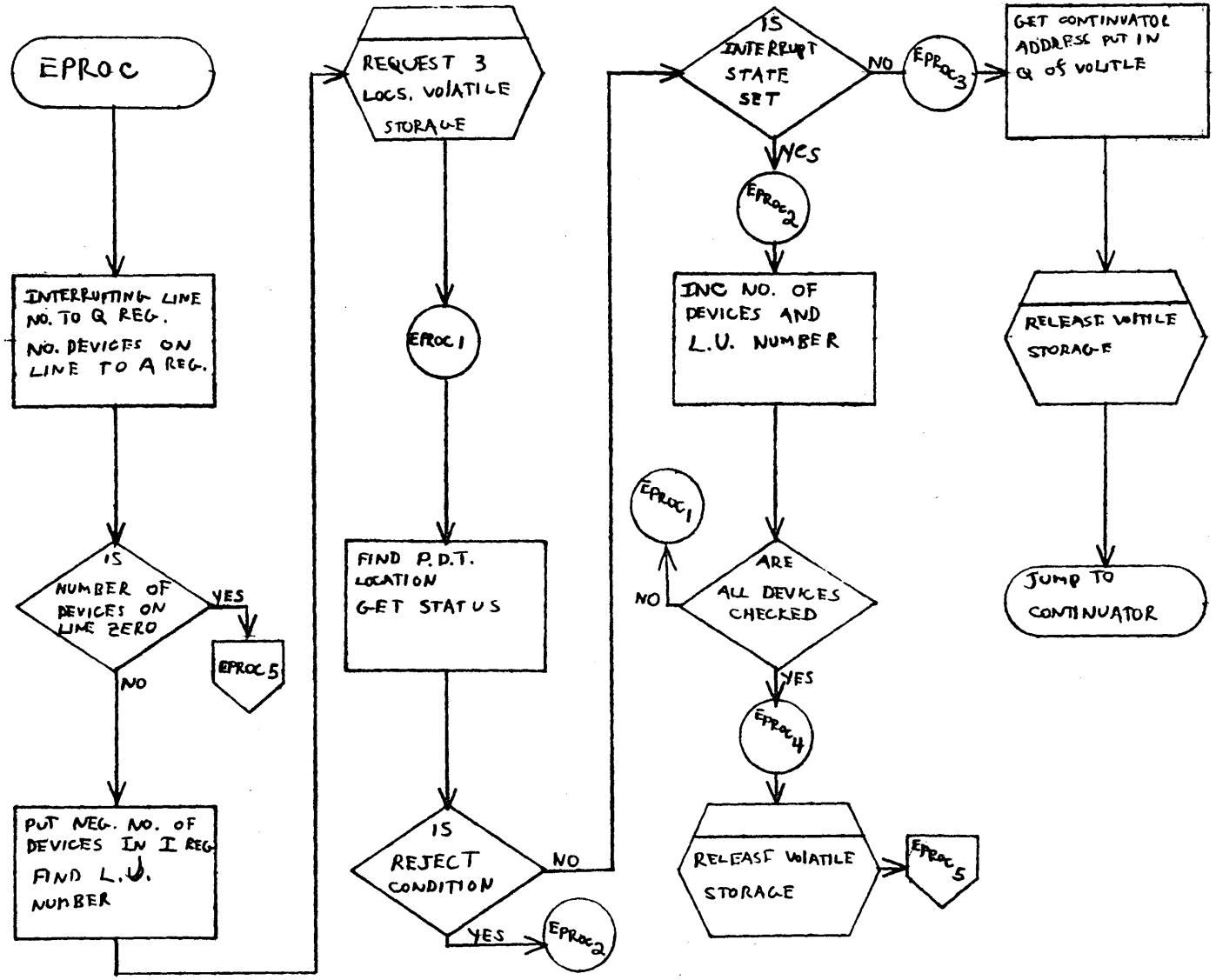
EPROC Entrance to External Interrupt Handler
I set to slot location through which interrupt came.

12.4 Internal Entry

PRINT Prints ghost interrupt error message
GI {line number in hex} exits to dispatcher

12.5 External References

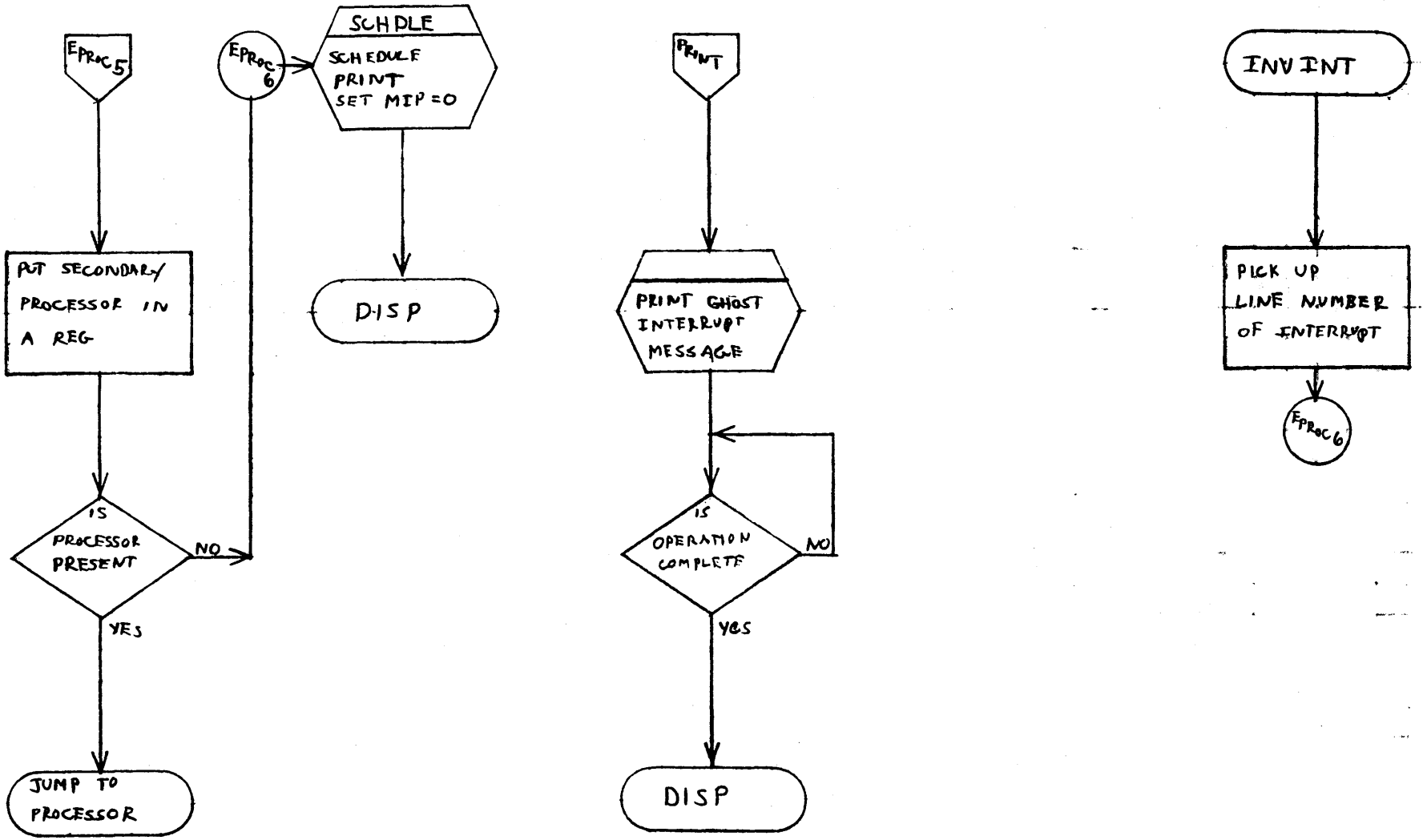
SECPRO Secondary Processor Table
LOGLA Table of logical units determines
L1-L15 number of units on line



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE EPROC		PROJECT MGR		
		PAGE 1 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

12.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE EPROC	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME MSOS 2.1			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

12.4

13.0 Volatile Storage Handler

13.1 Internal Symbols

VOLBLK - Volatile storage block starting address
VOLATL - Current address of entry into VOLBLK. Located in the communication area.
VOLEND - Address of the location following the volatile storage block.
OVFVOL - The entry name of a module entered in case of volatile storage overflow.
ZERO - A communication area cell containing zero.
VQ - Index for saving users Q.
VA - Index for saving users A.
VI - Index for saving users I.

16.1.1 Entry Symbols

VOLA - Allocate volatile.
VOLR - Return volatile previously allocated.

13.2 Function

A block of core, VOLBLK, is reserved in the system tables module for assignment to users as volatile storage. This block is handled by the subroutines VOLA and VOLR.

13.3 Entry Interfaces

The calling sequences to allocate a block of n words is:

```
RTJ VOLA      Execute with interrupts inhibited.
NUM n         Parameter N follows the RTJ.
```

The calling sequence to return a block is:

```
RTJ VOLR      With I containing the address of the block
              returned, and interrupts inhibited.
```

13.3.1 Entry Conditions

The interrupts must be inhibited.

When entering VOLR, ▽I▽ must contain the address of the block to be returned.

13.4 Exit Interfaces

Upon exit from VOLA, the registers A, Q and I will be saved in the volatile block. I will contain the address of the allocated area. VOLATL will be incremented by n, the number of words requested. The format for a block of volatile storage is shown below.

```
          0     Q saved      I upon return points here.
          1     A saved
          2     I saved
n words   3     {Open}
          4     {Open}
                  .
                  .
                  .
        n-1
```

The minimum size of a block of volatile is three {3} words.

Upon exit from VOLR, the registers A, Q and I will be restored from the volatile storage. VOLATL will point to the entry just returned. Once volatile has been returned, the contents of the volatile block are no longer reliable.

13.5 Internal Description

VOLA handles the allocation of the volatile storage. When it is entered, the registers A, Q and I are saved in the allocated area. The top of the volatile stack, VOLATL, is incremented by n, the number of words allocated.

If the stack overflows, control is given to the common overflow handler, OVFVOL, to take the appropriate shutdown or recovery actions. Control is not returned, if overflow occurred.

VOLR handles the return of allocated volatile storage. On entry the I-register must point to the core being returned. The I-register is simply stored into the top of the volatile stack. A, Q and I contain the initial contents {upon entry to VOLA} on the return.

13.6 Limitations

All users of volatile storage must conserve the address of the allocated volatile. Entry must be made under lockout. At least three {3} words must be requested. Once volatile has been returned, the contents of volatile are no longer reliable.

13.7 User Instruction

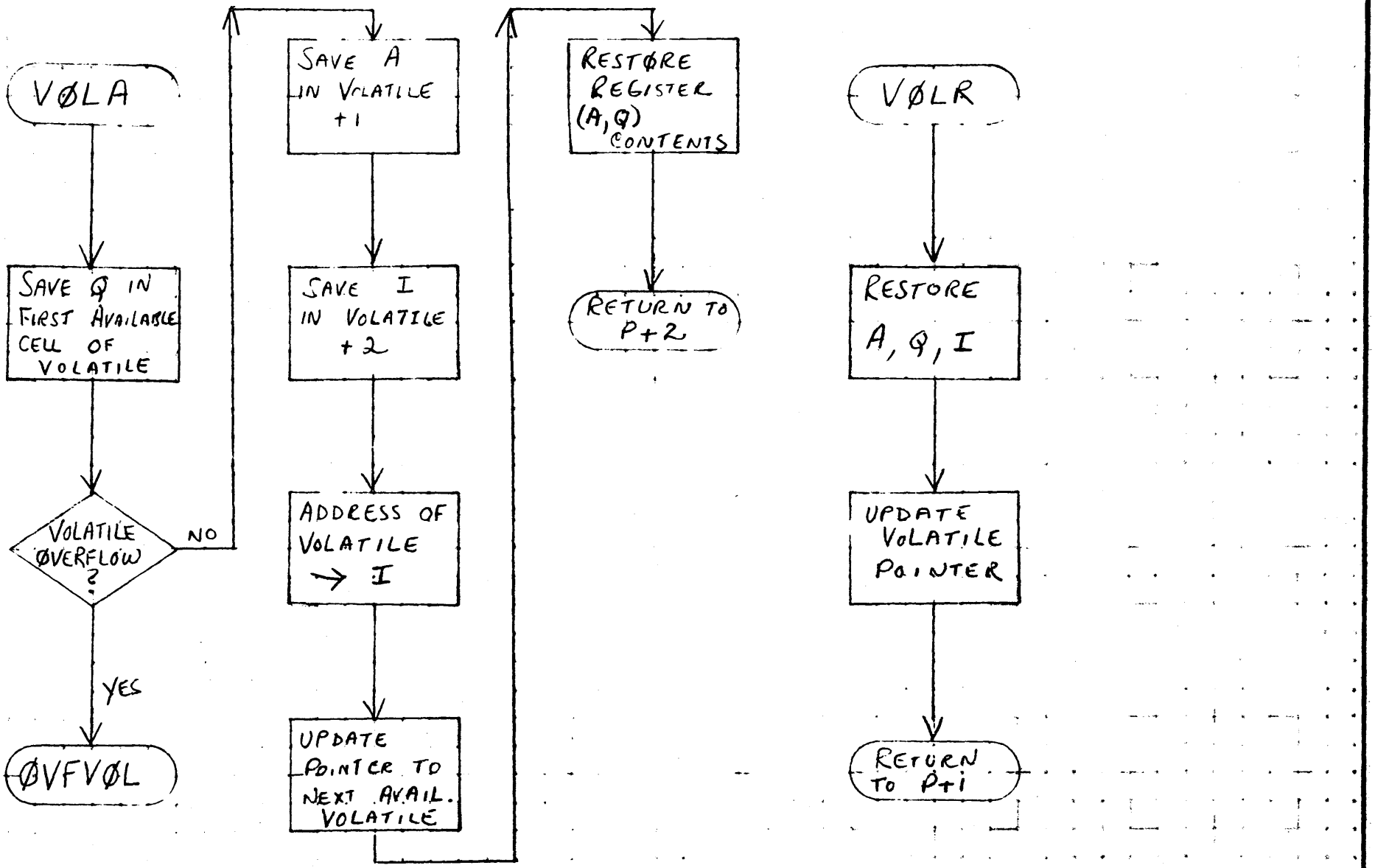
The volatile block is assembled with the system tables module. Enough must be allocated so that the maximum amount required for any level will be available.

13.8 OFVOL

If the volatile overflow module {OFVOL} is present, in the event of an over subscription of volatile storage, it will clear the mask, type "OV" and hang in a one cell loop. If the volatile overflow module is not present, then a jump to 7FFF will occur with unpredictable results.

13.8.1 Entry Point

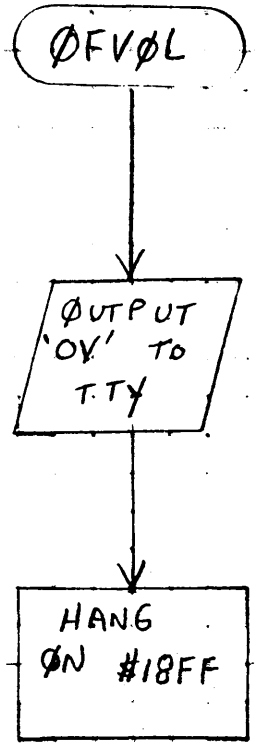
OFVOL



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ALVØL		PROJECT MGR.			
FLOWCHART ☐	MSOS 3.0	PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

13.4



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ØFVØL	PROJECT MGR.			
	NOS 3.0 PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

13.5

14.0 Monitor Entry and Exit for Requests

14.1 Internal Symbol Definitions

VR	Relative location in volatile containing the user program's return address. {Equals 3}
VPTR	Relative location in volatile containing the pointer to the user program's parameter list {5}.
ZERO	A location in the communication region containing a zero {$22}.
ONEBIT	The first location of a table constructed so that entry n contains 2^n. This is normally $23.
RCSCHD	The code for the scheduler request {9}.
LPMSK	The first location of a table constructed such that entry n contains 2^n-1. This is normally location 12.
VTMP	Relative location in volatile containing the request code {7}.
AMONI	A location in the communication region containing the location of this program. This is normally location $F4.
V	The number of words of volatile allocated per request {8}.
AREQXT	A location in the communication region containing REQXT {request exit}. This is normally $B9.
MONI	The subroutine entry point to the Request Entry Processor.
RCTV	T_0 through T_{30}.
REQXT	Common exit for monitor requests.
AVOLA	Equated to $BB {address of volatile}.
AVOLR	Equated to $BA {address of release volatile}.

14.2 Program Function

User programs generate requests for various functions such as I/O, core allocation, and scheduling. All of these requests are processed by the Request Entry Processor {REP}. Its function is to reserve volatile storage, save the registers A, Q, P, and I in volatile storage, and give control to one of the request processor routines $T_0 \cdots T_{30}$, depending upon the request code, RC, in the user's calling sequence.

14.3 Entry Interface

Entered from protected programs as a result of a monitor call. Entered from unprotected programs via IPROC {Chapter 4}.

14.4 Exit Interfaces

The Request Entry Processor gives control to the request processors, T_0 through T_{30}, with specific information in the registers. Each request processor upon entry can assume the following:

REGISTER	CONTENTS
A	A_{14-0} is the location of the parameter list. If $A_{15} = 0$, then the reference to the parameters in the call was direct. Otherwise, $A_{15} = 1$, and the reference was indirect (an INDIR request).
Q	Absolute address of the request processor being executed.
I	I contains the location of an eight (8) word block of volatile storage.

Location	Mnemonic	
(I) + 0	VQ	The user's Q-register is saved here.
(I) + 1	VA	The user's A-register is saved here.
(I) + 2	VPL	Not set by the Request Entry Processor. Intended to hold the request priority level.
(I) + 3	VR	The return address of the user. If this was an indirect call, then the return address has been incremented by one (1) to give the correct return address. Otherwise, this was a direct call and the return address must be adjusted by the request processor.
(I) + 4	VI	The user's I-register is saved here.
(I) + 5	VPTR	The location of the user's parameter list. This is in the accumulator A. See discussion of A above.
(I) + 6	VTDS	Not set by the Request Entry Processor. It is intended to contain the top of the stack for the desired logical unit.
(I) + 7	VTMP	A temporary storage cell containing the request code, RC.

14.4.1 Return to Requestor

Control will be returned to the next instruction with the registers A, Q, and I restored. Overflow will not be saved. Interrupts will be enabled and the priority level will be the same as upon entry.

14.5 Internal Description

The Request Entry Processor handles all monitor requests made by the user program. The user enters the Request Entry Processor via an indirect return jump to MONI. The Request Entry Processor inhibits all interrupts, saves the user's registers Q, A, I, and return address in an area unique to this request, and then enables interrupts. The Request Entry Processor is re-entrant beyond this point, and works only with the data area unique to this request. The I-register is used to hold the address of this unique area which is called volatile storage. The location of the parameter list is then stored in volatile. If this request has an indirect reference to the parameter list, the return address to the program is adjusted to return control to the next sequential instruction. If this indirect call was made as the result of the completion of an I/O operation, the registers are adjusted to make this look like a scheduler call since the request code in the user's request parameter list may not be altered. Control is then given to the request processor specified by the request code.

14.6 Restrictions

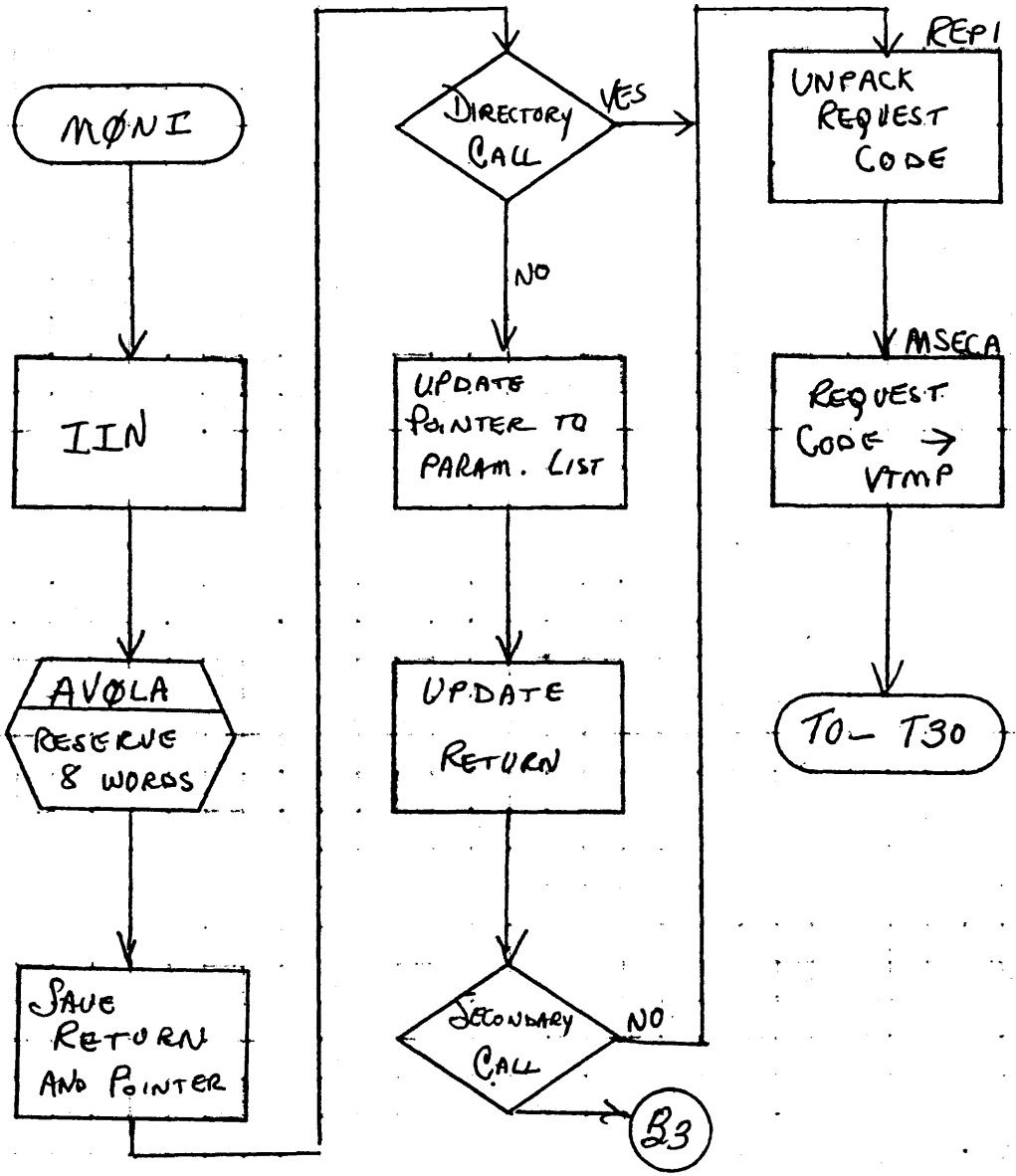
The I-register must be conserved throughout the request processor called since it contains the address of volatile storage. Each request processor must be re-entrant since it runs at the requestor's level. When each request processor finishes it must return the volatile core storage by jumping to REQXT.

Label	Op	Address	
	JMP-	{AREQXT}	Address of request exit. REQXT is contained in AREQXT.

NOTE: The ▽MINI MONITOR REQUEST ENTRY▽ is identical in every way with this module with a single exception: It is equipped to handle only 13 requests.

14.7 Adding Requests

When adding requests with codes larger than 13 consult the chapter dealing with the protect processor for necessary changes.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE MØNI		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

4.5

AA1385 (FORME A127-1)

PRINTED IN U.S.

15.0 Processor for READ, WRITE Format READ, Format WRITE

15.1 Entry Interfaces

The Request Processors (T1, T2, and T6) are entered from the Request Entry Processor with the A, Q and I Volatile set up as shown below.

Register	Contents
A	A14-0 is the location of the parameter list. If A15=0, then the reference to the parameters in the call was direct. Otherwise, A15=1, and the reference was indirect.
Q	Absolute address of the request processor being executed.
I	I contains the location of an 8-word block of volatile.

Volatile Storage	Mnemonic	
(I) + 0	VQ	Q saved by Request Entry Processor.
(I) + 1	VA	A saved by Request Entry Processor.
(I) + 2	VPL	Used to hold request priority level.
(I) + 3	VR	P-register saved by Request Entry Processor. If indirect all, P is already incremented by 1 for proper return address.
(I) + 4	VI	The I-register saved by REP.
(I) + 5	VPTR	Used to hold the user's parameter list location, also in A above.
(I) + 6	VTPE	Used to hold the preceding thread location.
(I) + 7	VTMP	A temporary used to hold logical unit number.

15.2 Exit Interfaces

Exit to the Driver:

The driver will be scheduled, if the device associated with this logical unit is not busy. The Q register upon entry to the driver initiator will contain the location of the physical device table entry for the device.

Exit to the User:

The request processor returns control to the REQXT where the volatile storage is released and control is returned to the caller.

Upon return to the user, the registers A, I and Q_{14-0} will be restored. If $Q_{15=1}$, the thread location in the parameter list is not zero, implying that this request is already on some other thread. In this case, no action will be taken on this call. This action is apparent only to protected callers.

Scheduling of the Completion Address, C

Control will be returned to the Completion Address C at level CP when the I/O requested has finished or if the device is down and no alternate exists. A message will be typed in forming that the unit is down. Q will contain word 3 of the parameter list. The high order bits of Q will contain the error code V.

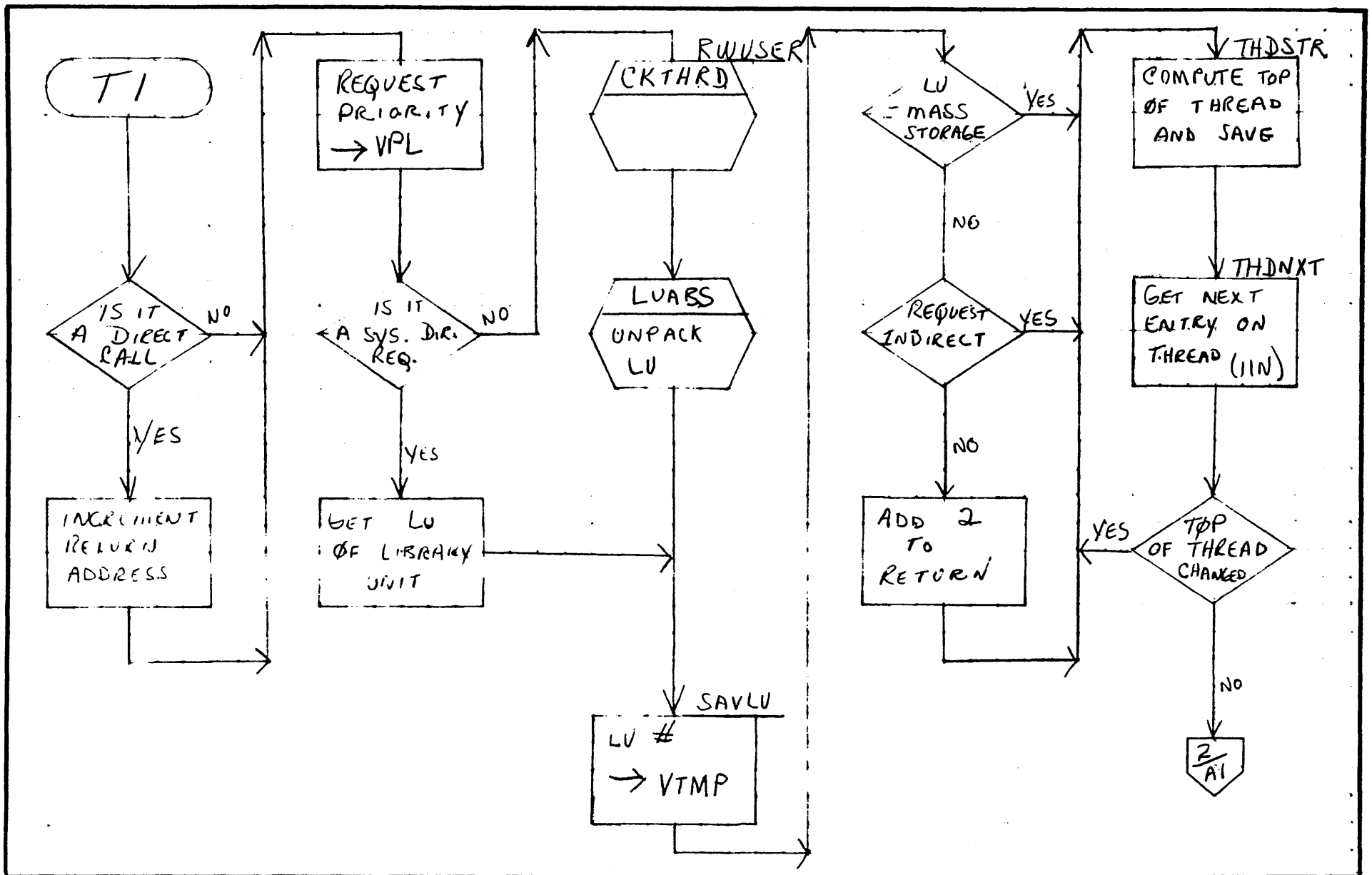
15.3 Internal Description

Requests are threaded onto the logical unit according to Request Priority. If the associated device is not assigned to a logical unit and is operational, the driver for the device is called; or, if the device has failed and has no alternate, the completion address is scheduled with an error code indicating failure returned to the completion address. Subroutine ALTSUB, in the Alternate Device Handler, Section 17.0, is used to obtain the alternate logical unit if required.

NOTE: The ¤MINI¤ RW PROCESSOR¤ module is identical to this module. If the ¤MINI ERROR PROCESSOR¤ module is used, ALTSUB simply returns to the caller.

15.4 Request Code Zero

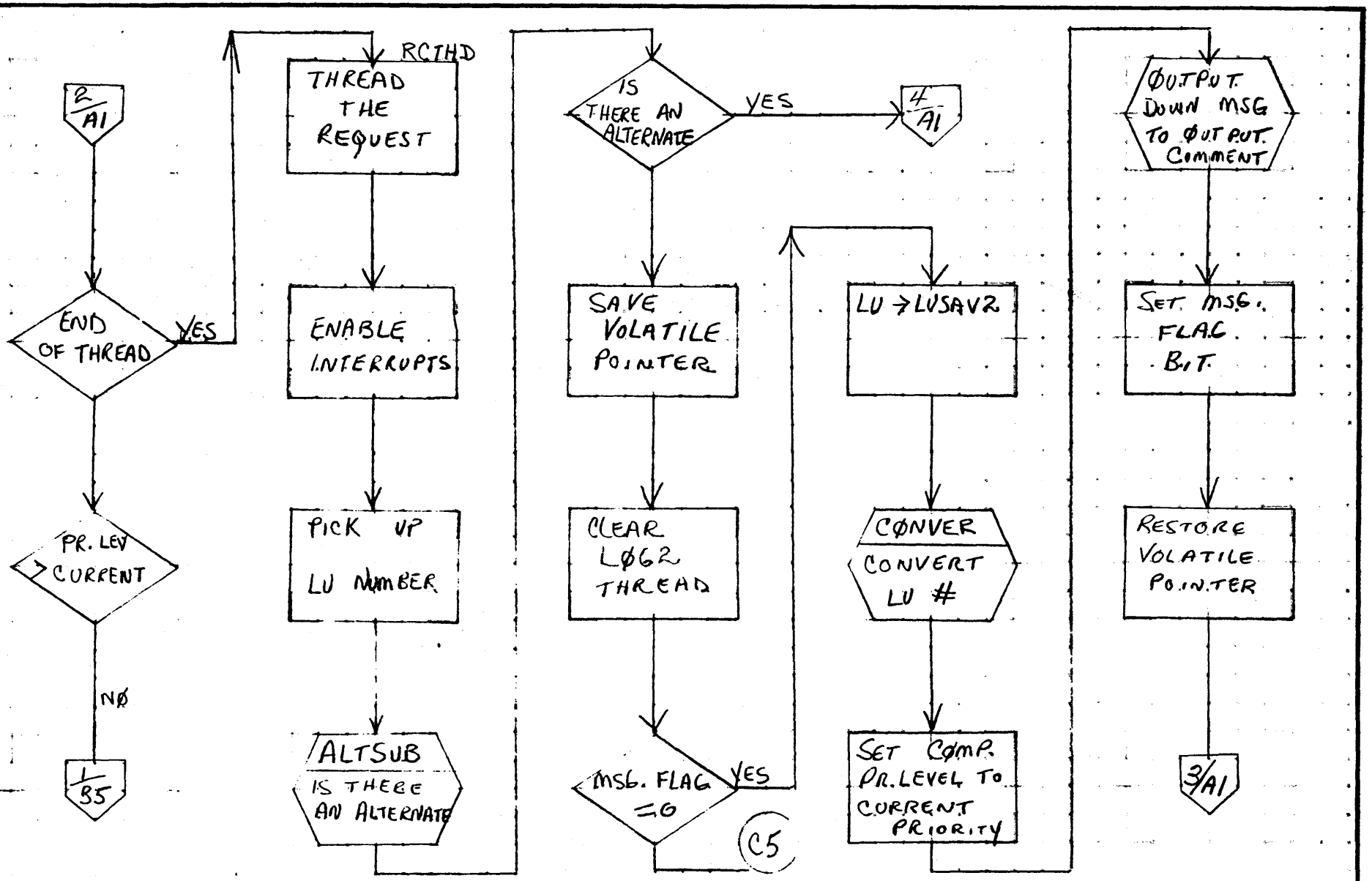
The zero request code is used to cause mass storage reads which result from SCHDLE requests. For example, if a mass storage resident program is scheduled, the SCHDLE request processor passes the system directory entry to the SPACE processor for allocation of space. The SPACE processor then passes the system directory entry to this processor to effect a transfer of the program from mass storage. The apparent request code carried in the system directory entry is zero.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE CW		PROJECT MGR		
	MSOS 3.0	PAGE 1 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

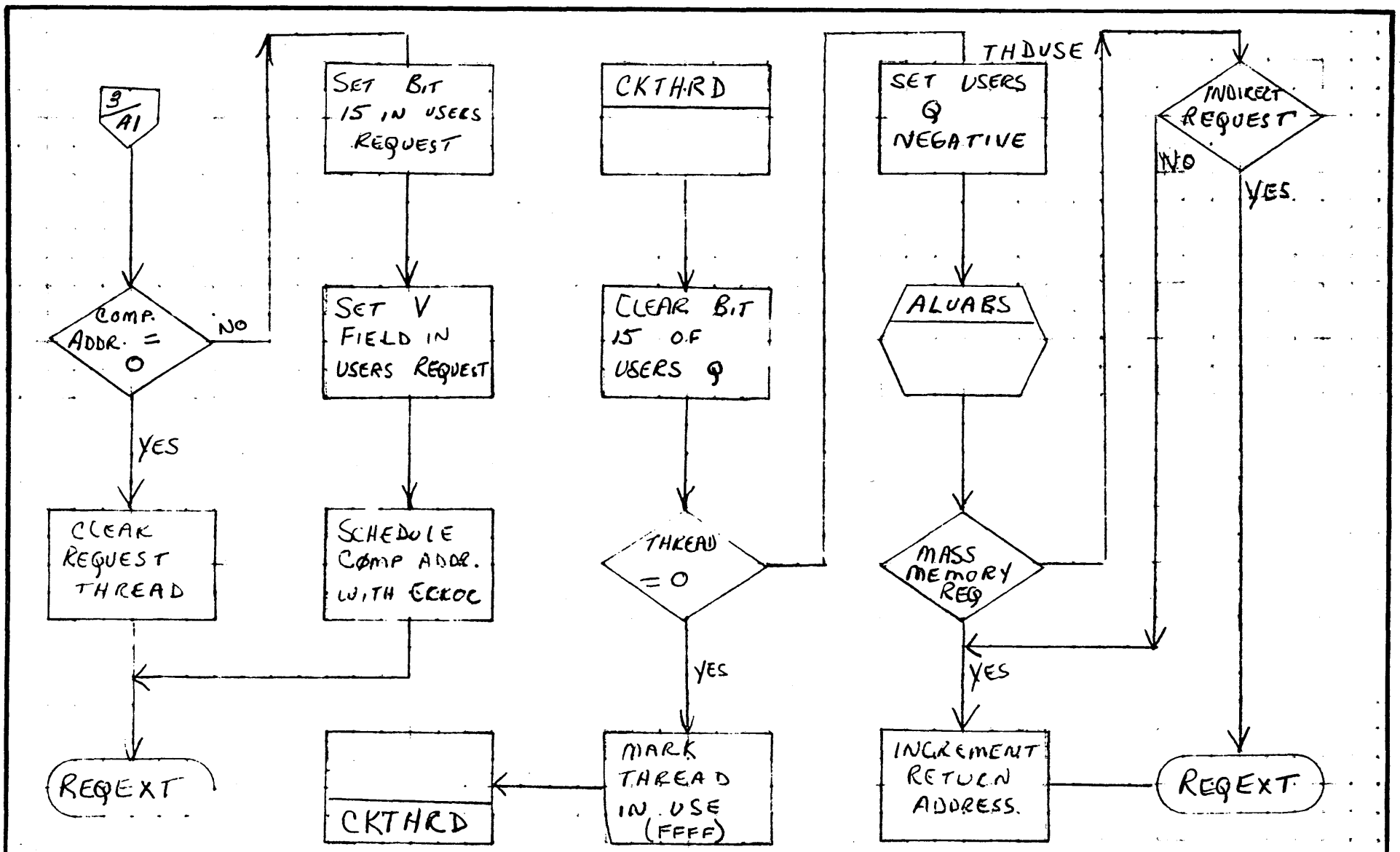
15.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IIIS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RW		PROJECT MGR.			
	NOS 3.0	PAGE 2 OF 4	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

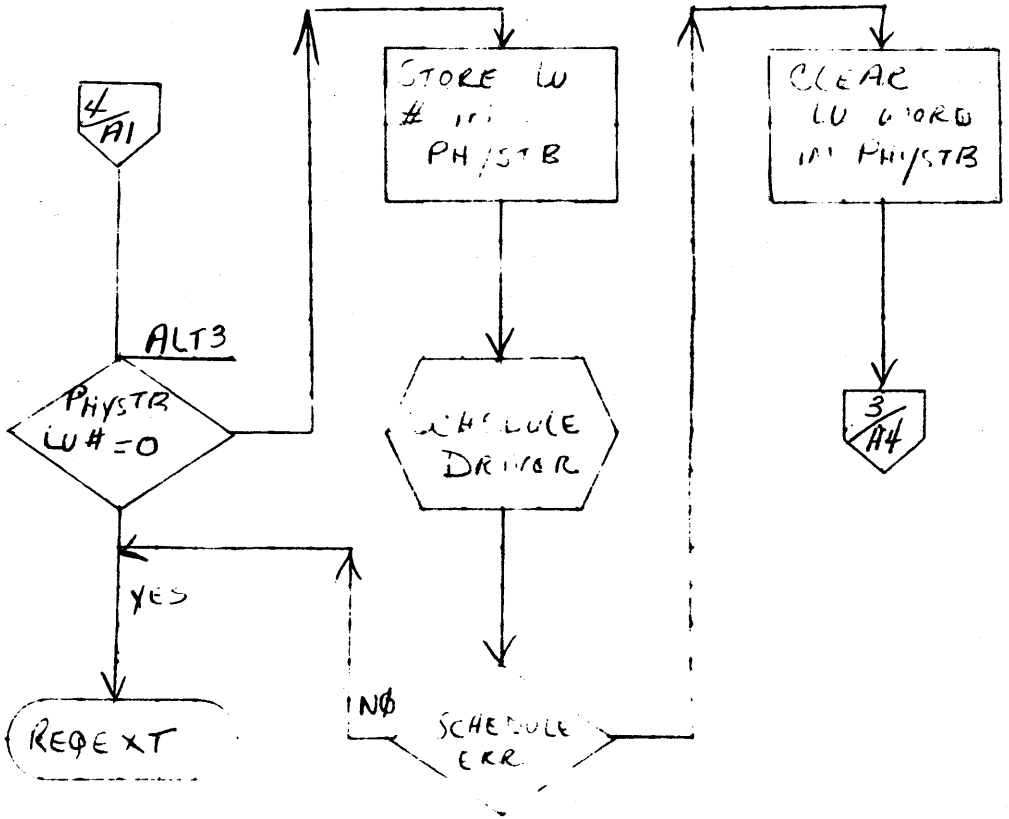
15.4



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE R.U.		PROJECT MGR.		
		PAGE 3 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		

MAR 9 1971

15.5.5



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	3.0 PAGE 4 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	SK NAME			

MAR 5 1971

15.9

16.0 SPACE, RELEAS, SWAPPING AND RESTART

16.1 GENERAL BACKGROUND

Many modules are non-resident, i.e., they are not kept in core. Therefore, when they are operated, it is necessary to read them in from the library. There is an area reserved for this purpose, the size of which varies from system to system. Each non-resident program, prior to operation, must be assigned space in this area and read into it. Similarity, when a non-resident program completes its function, it must cause the area allocated to it, to be restored to the block of empty space available for allocation to other non-resident programs. The SPACE and RELEAS requests deal with these operations.

Scheduling a mass memory resident system directory program causes the following operations to be executed.

1. Space is assigned in the allocatable core area.
2. The program is read into core from mass memory.
3. The starting address of the program, i.e., the start of the assigned core area, is scheduled at the requested priority.

All mass memory resident system directory programs must be written to be "run anywhere" (using relative addressing, etc.) since the program may be assigned different core areas on successive operations.

If it is necessary to allocate space in the non-resident area and insufficient space is available, it may be possible to pre-empt that area of core used for job processing. The procedure involved is called swapping.

Restart is the initial procedure followed from a dead start condition. For purposes of allocating core space in as simple a manner as possible, the area to be allocated is treated as an I/O device. This pseudo device is operated by a pseudo controller (the core allocator) which is operated via a driver (DRCORE). The SPACE and RELEAS requests take the place of READ and WRITE requests in this situation. In order for this operation to work smoothly, the pseudo device is always considered to be logical unit #1. This is true for all systems. The modules to be discussed in this chapter are:

CORE ALLOCATOR
DRCORE
SPACE REQUEST PROCESSOR

16.2 CORE ALLOCATOR

16.2.1 External Symbols

LVLSTR	Level start table
LEND	Level end
CALTHD	Core allocator thread

16.2.2 Internal Symbols

MINSIZ	Minimum allocatable area (assembled as 2)
MAXNO	Largest single precision positive number

16.2.3 Function of the Program

The Core Allocator module allocates core to program which are mass memory resident. It also allocates core to programs which require additional temporary working area at execution time.

The Core Allocator is required in the monitor on all systems which have a mass memory used for program storage.

The Core Allocator accepts returned areas of core and, if possible, combines the returned area with adjacent areas.

Requests for core allocation are stacked by request priority and core is allocated on a priority basis. i.e., the higher priority programs have access to more of the allocatable core.

16.2.4 Comprehensive Program Description

The Core Allocator threads together all the pieces of available core memory. Initially there is one piece of core which is the entire area. As allocations are made, the available area gets broken up into many pieces. As pieces are returned, they are regrouped into as few pieces as possible.

16.2.4.1 Organization of Core

Total core memory is diagramed in Figure 1. It is divided into three parts: Part 1) the core resident programs constants; Part 2) the allocatable area; and Part 3) unprotected core.

Part 2 is allocated by the core allocator according to the request priority in the parameter list. A fixed amount of the available core is available to each priority level. As shown in Figure 1, higher priority levels have access to more of the core than lower priorities. This has the effect of guaranteeing that many low priority programs cannot use an area set aside for a high priority program. An area can always be available to a higher level by restricting the area available to lower levels. The core allocator also selects the core from the smallest available piece. This has the effect of minimizing the number of pieces of core that are too small to be usable. The technique uses the small leftover pieces first while leaving the big pieces for future requests.

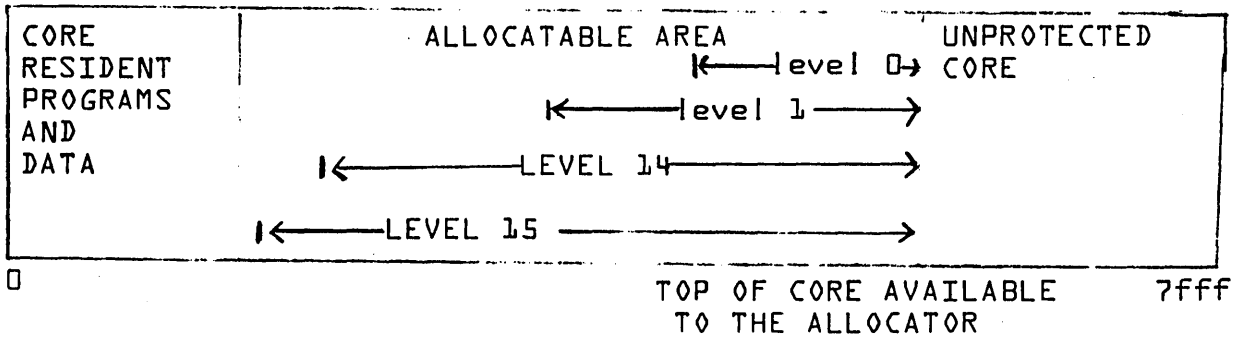


FIGURE 1

Core Memory is initialized as follows:

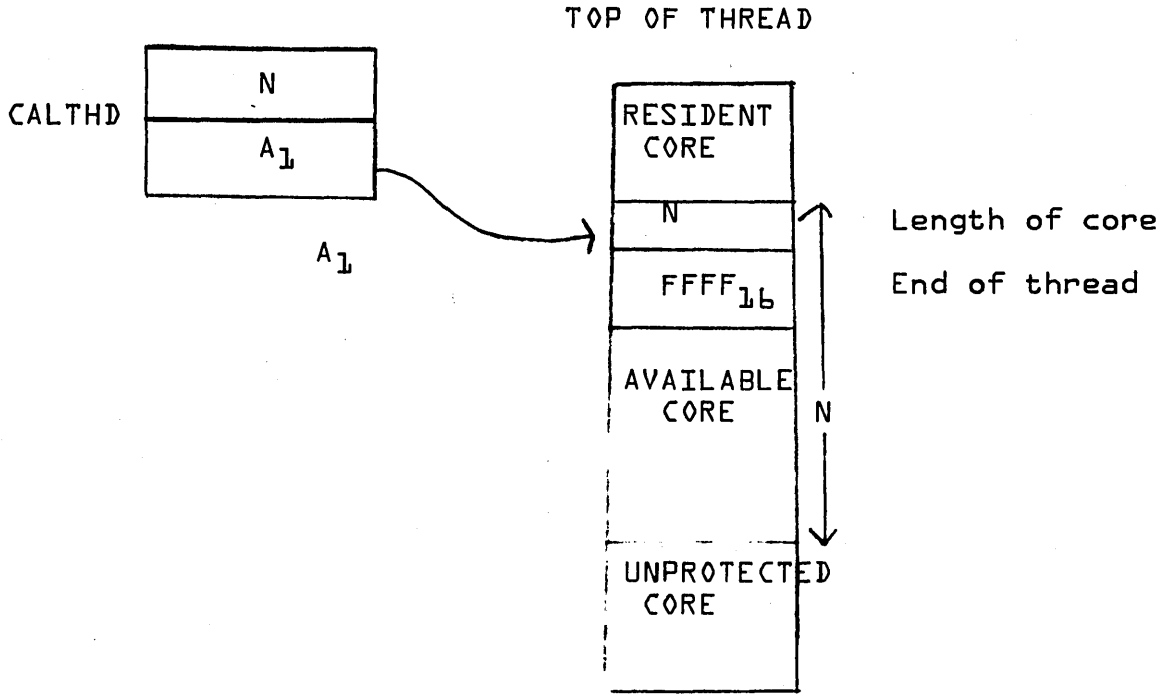


FIGURE 2

Individual pieces of allocated core are oganized as shown in Figure 3.

The core allocator stores two control words into the allocated core area. The first word, located at °A-2° always contains the requested length N, plus 2, and represents the actual length of the allocated area. The second word, located at A-1, always contains the address of the area, A.

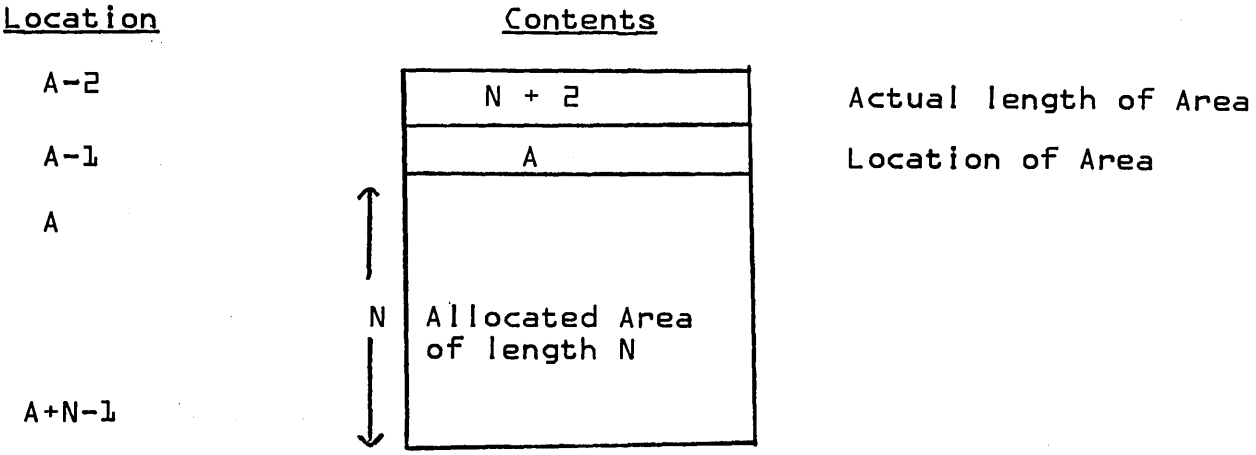


FIGURE 3

After an allocation has been made, core memory appears as shown below:

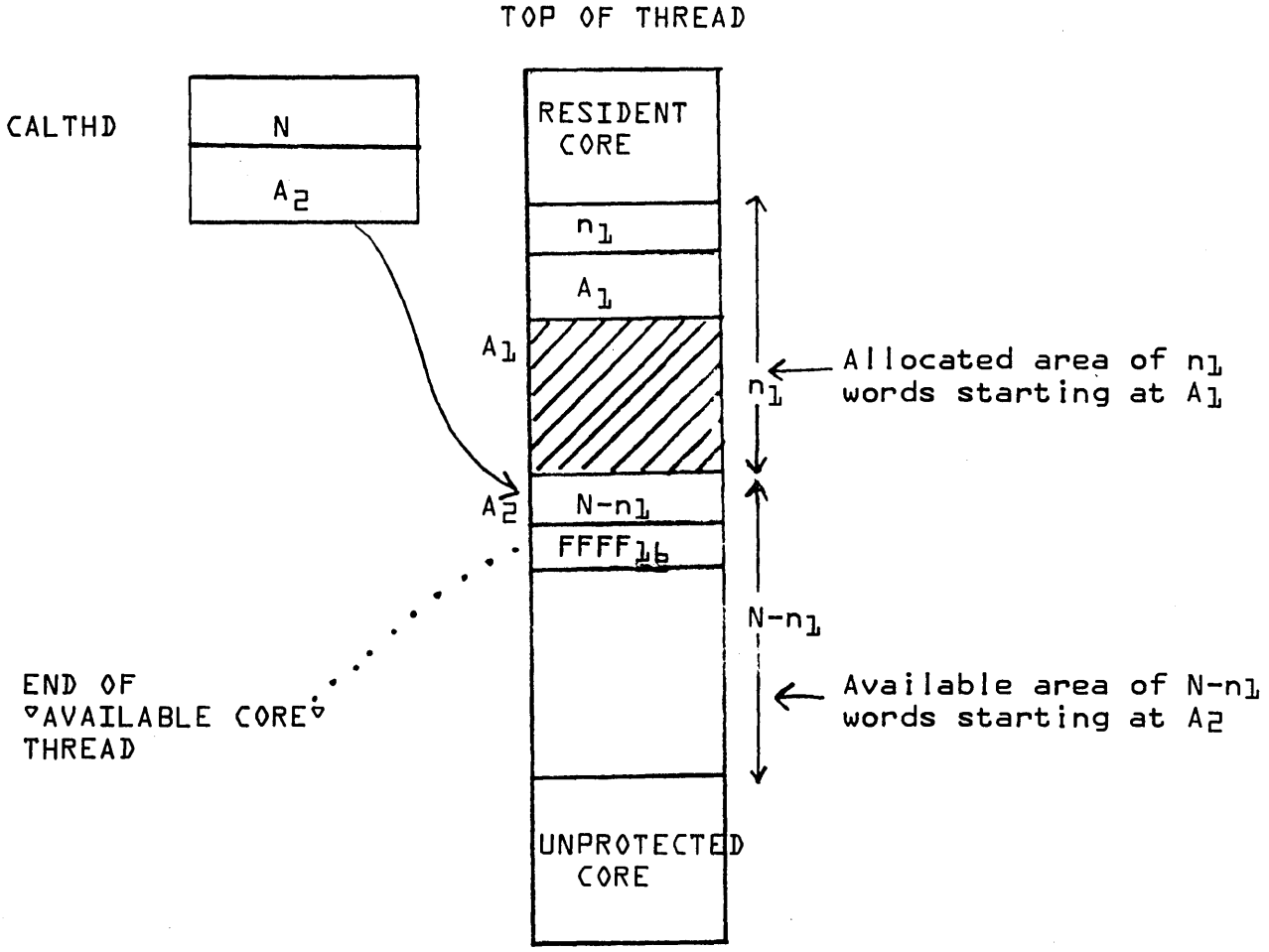


FIGURE 4

16.2.4.2 Core Allocation Logic

The subroutine, REQALC, {request allocation} actually does the analysis to select the available area of memory. The logic is discussed below. REQALC is called by the Core Allocator Driver with the parameters, requested length and level.

If the requested length is larger than the area available to the requested level, then REQALC, immediately returns with a zero parameter to the driver.

Otherwise, a search of all available core to the requested level is made to select that piece which has the following properties.

1. The piece must contain N+2 words available to the requested level.
2. The remaining piece {after N+2 words are allocated} is smaller than the corresponding piece of all other allocatable areas to the requested level.

If no such piece is found, then the parameter, -1 is returned to the Core Allocator Driver. Otherwise, the optimal piece is broken into two or three parts, and the thread of available core is strung through the left-over piece. The left-over pieces are restricted to being larger than MINSIZ so that they can contain the thread information. Figure 5 shows how a piece is broken up into three pieces. Piece #1 lies below the area available to the level and Piece #2 remains after the requested piece has been removed.

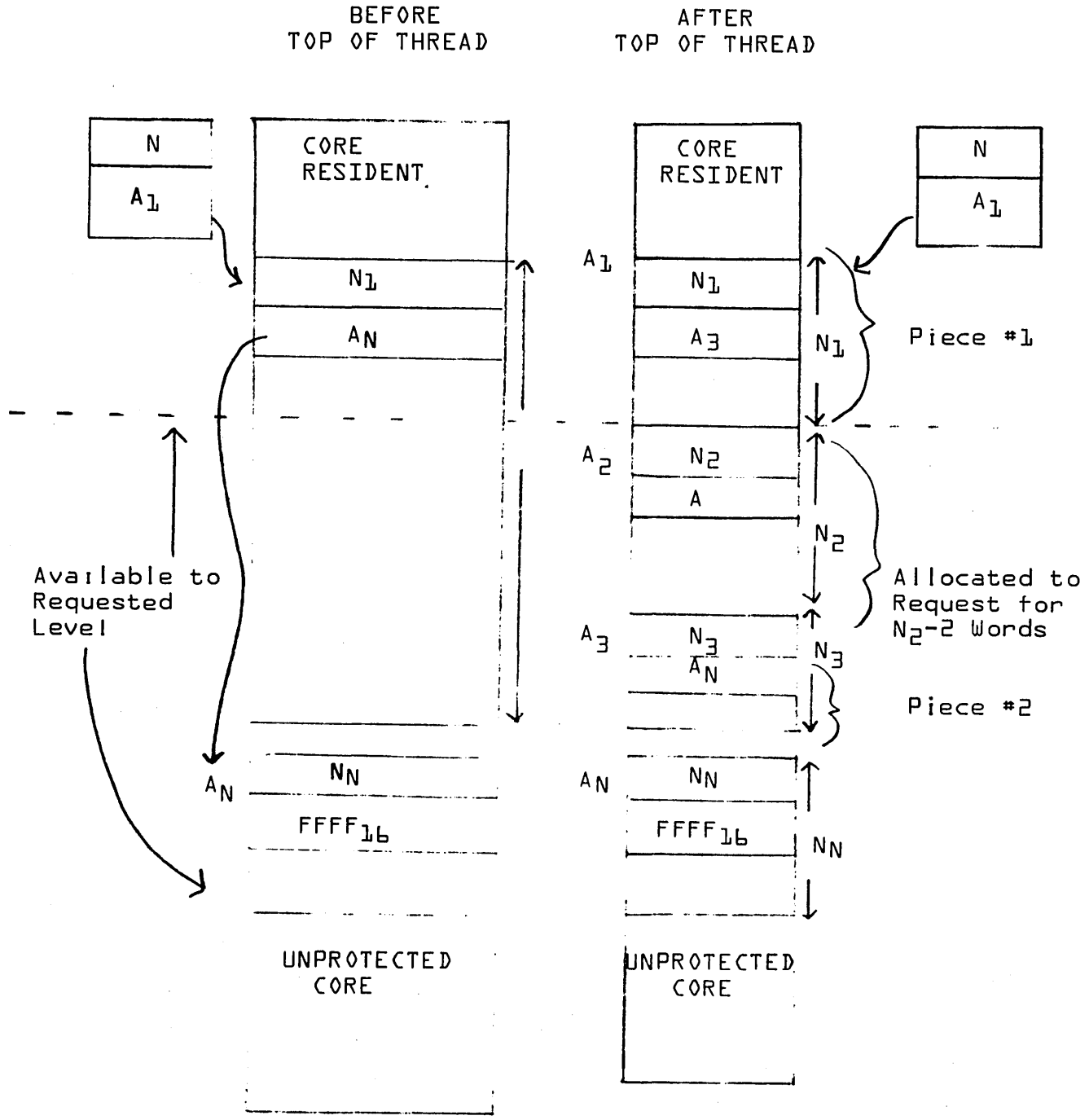


FIGURE 5

16.2.4.3 Core return logic

The subroutine RTNCOR does the analysis to combine the return piece of core with the already available pieces. RTNCOR is entered from the RELEAS request processor {DRCORE}.

A search is made to find the first piece of available core which is below the returned piece. The returned piece is threaded into its proper position {the available core thread is ordered by ascending core location}.

A check is made to see if the returned piece touches its lower and/or upper neighbor. If so, the adjacent pieces are combined into one piece and the thread is updated.

16.2.5 Tables

LVLSTR This table contains 17 cells and is located in the system table module {SYSBUF}. The first 16 cells are indexed by priority level. Each entry contains the core address of the first cell allocatable to programs with request priorities of the level represented by the index. The last cell contains the address of the last cell in the area which is controlled by the core allocator.

16.3 DRCORE

16.3.1 External Symbols

LAND Address of last location in the area controlled by the core allocator.

LOGIA Logical unit table containing PHYSTB addresses for each logical unit.

CALTHD Core allocator thread.

RTNCOR Entry to core allocator for releasing space.

CORE PHYSTB entry for the core allocator.

LVLSTR Level start table.

SWAPAR Mass storage address of area where unprotected core contents are saved during swap. Filled by the initializer.

UNPIO — Count of number of unprotected I/O calls pending.

SPASW — A switch in TRANV used to inform the protect processor that a swap is desired.

LOG2 — Logical unit table containing thread tops for all logical units.

REQALC — Entry to the core allocator for allocation of space.

AREAC — Start address of block controlled by the core allocator.

16.3.2 Function of the Program

DRCORE serves as the driver for the core allocator and as the request processor for RELEAS requests. In this capacity it makes all decisions in the area of swapping and stacking calls for space.

SWAPCK is the entry point to a subroutine used by the job processer and library edit programs to count down the UNPIO unprotected I/O counter and restart the space driver if it is waiting to swap and UNPIO is zero.

16.3.3 Requests for Space

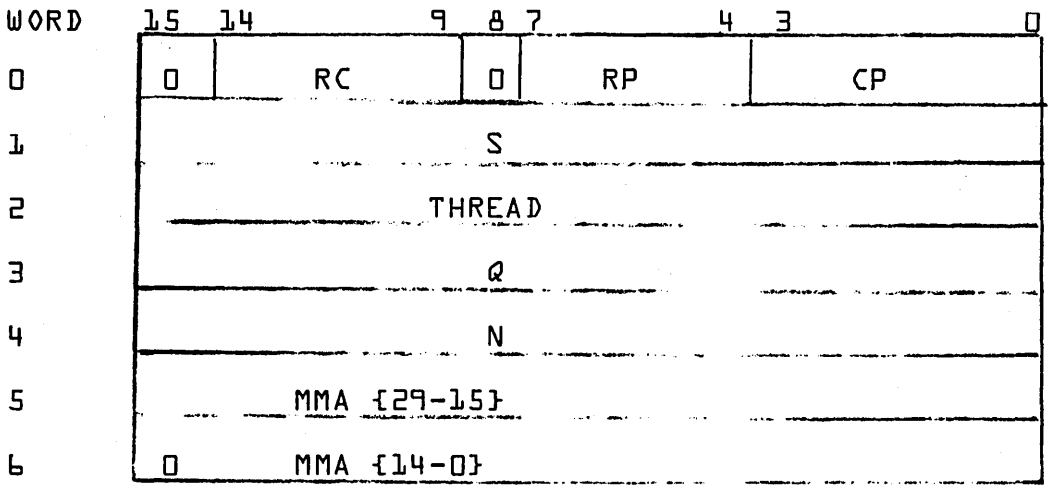
Requests for space come from two sources.

1. Schedule calls for non-resident system directory programs.
2. SPACE requests.

16.3.3.1 System Directory Format

The scheduler gives control to DRCORE when a system directory request for a mass memory resident program is made. DRCORE determines the starting address of the program, based upon the areas of core that are currently available and enters this address in word 1, S, of the System Directory entry.

The format for the system directory is shown below:



7 words per entry in the Directory for Mass Memory Resident Programs

RC is the request code for the System Directory and is zero.

RP is the request priority used in the allocation of core memory. RP is a number from 0 to 15. {set by the LIBEDT ΔS statement}. RP=1 to 3 is reserved for use by the Job Processor.

CP is the completion priority at which the mass memory resident program will be scheduled after the read is complete. CP is set for the Scheduler and is obtained from the requesting program's scheduler call.

S is the starting Core address of the program and also the first location of the allocated core. This is set by the core allocator.

THREAD is the thread location used to point to the next entry on a threaded list. This directory entry will be placed on the following threads.

THREAD NAME	POSITION DETERMINED BY	WHEN
Core Allocator	RP	after scheduling
Mass Memory I/O Driver	RP	after allocation
Scheduler	CP	after Mass Memory Read

The thread location is set non-zero by the Core Allocator Request Processor and is cleared to zero on completion.

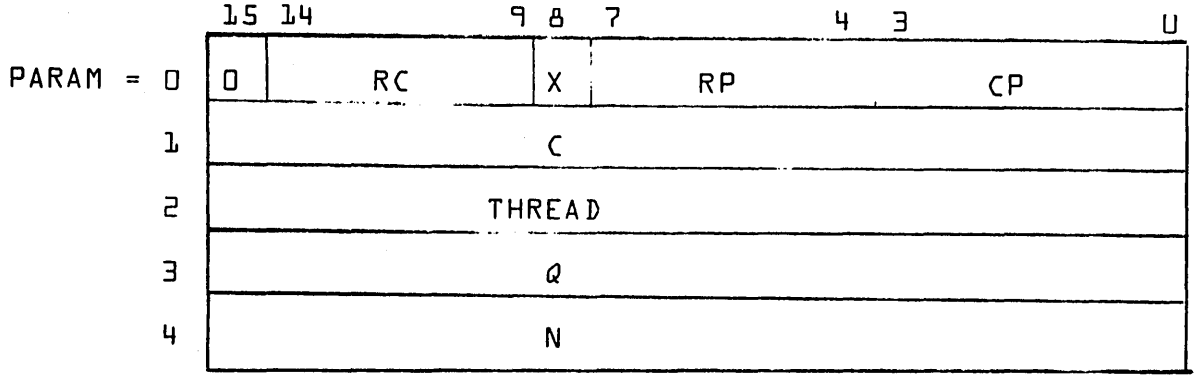
Q is the parameter passed from the requesting program to the requested program.

N is the length in words of this program on mass memory.

MMA is a double length word containing the mass memory address of this program. The first word contains the most significant 15 bits. The second word contains the least significant 15 bits.

16.3.3.2 SPACE Requests

The user program may make a Monitor request for allocating core. The core area will be allocated to the requesting program and must be returned by the requesting program before it will be reassigned to another program. The list of parameters is as follows.



RC is the space request code and is equal to 10.

X is a relative/absolute indicator, modifying C.

RP is the request priority, the relative priority of this request used to determine the position on the core allocator thread and also to determine area of core allowable. RP is a number from 4 to 15.

CP is the completion priority, the level at which control will be returned to C.

THREAD is the thread location used to point to the next entry on a threaded list. This monitor request will be placed on the following threads:

THREAD NAME	POSITION DETERMINED BY	WHEN
Core Allocator	RP	after request
Scheduler	CP	after allocation

The thread must initially be zero, and is reset to zero on completion.

Q contains the address of the area allocated and is in the Q register when control is given to the completion address, C. If allocation is impossible Q will be set negative.

N is the number of words requested.

16.3.3.3 Internal Description of Allocation

The Space Driver DRCORE is operated by a SCHDLE request from the request processor {just like any other driver}. It uses subroutine FNR for new requests and uses the Core Allocator Subroutine CORALC to obtain the space required. If sufficient space is available then COMPRQ is used to complete the request. Q will be set to the address of the allocated area when the completion address for the space request is scheduled via COMPRQ. If it is impossible for sufficient space to be available and swapping is in effect then the completion address will be scheduled with Q set negative denoting an error. Errors of this type due to system directory calls cause the system directory call to be ignored but cannot be detected by the caller as no completion address is available.

If insufficient space is not available then an attempt is made to swap, the request is rethreaded and the driver is set "not busy". If core is released before swapping is effected, then the space driver will be re-entered and the request will be completed if

sufficient space is available. Otherwise the request will be processed after the core swap area is released. For swapping to be executed the following conditions must all be true.

1. The completion priority is greater than 2. This is necessary since programs of level 2 and below are not operated after a swap since they might involve job processing.
2. A swap is not already in effect.
3. A suitable time internal, since the last swap has passed.
4. No unprotected I/O is in progress.

If any of these conditions are not fulfilled, the request is put back on the core request thread just before DRCORE exits to the dispatcher.

Additionally, in the case of condition 4, SPASW is set non-zero so that the protect processor will schedule DRCORE whenever UNPIO=0 and the allocator is not busy.

If the above conditions for swap are fulfilled, then the following operations occur:

1. A write is started which transfers the contents of unprotected core to a designated area on mass storage. This area is set up at system initialization.
2. A loop is scheduled at level 2 to lock out all programs at that level and below.
3. The LVLSTR table and LEND are updated to reflect the additional space available for allocation.
4. SWAPON is set to one, to indicate a swap has occurred.

At the completion of these operations the space driver is marked "not busy" and the request that caused the swap is re-threaded to the top of the LOG2 request thread. When the swap transfer to mass storage is completed, the space driver resumes as follows.

1. The core allocator is entered to release the space just made available.

2. The area is protected.
3. A space request for the swapped area is added to the wait list for threading on the allocator thread at completion of DRCORE processing.
4. A new attempt is made to allocate the space to the call which caused the swap.

When enough space is released so that the area is again available for job processing {the SPACE request made above is completed} the above procedures are reversed and the job is resumed as if no swap occurred.

NOTE: For swapping to combine the allocatable ^unprotected^ areas, the space request processor must be the last resident module.

The priority level of the space driver is determined by the completion priority set in Word 0 of the CORE physical device table. It is usually set to seven {7}. When a swap occurs the space driver must set all the protect bits in the unprotected core area. To do this requires 6.6 micro-seconds per location. Thus, for an ^unprotected^ area of size 10K the driver level will be busy in this loop for approximately 66 milliseconds when a swap is requested or released.

The space driver rethreads a request back on to the allocator thread if it is not possible to allocate enough space for the request at that time. No attempt is made to process lower priority requests even though they may require less space. The exception to this rule is if the request to be re-threaded has a completion priority of less than three {3}. These requests are put on a wait thread temporarily and then an attempt is made to allocate space to the next request on the allocator thread. When any other requests have been processed requests on the wait thread are returned to the allocator thread.

On completion of job processing, routine JOBEND in the Manual Interrupt Processor is entered to cause a core swap. This is done by making a special space request that can only be satisfied at the given request priority by a core swap. The special area so allocated is released when the job processor is requested. This area occupies only four cells for the allocator thread at the end of the ▽unprotected area▽.

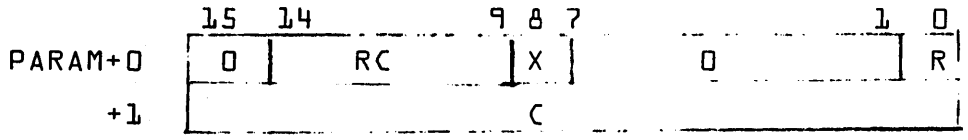
Unnecessary swapping is thus avoided when the job processor is not in use. Excessive swapping on temporary overloads during job processing can be avoided by setting the minimum interval between swaps. Table LVLSTR must be set up very carefully noting that programs that are not independent cannot be assigned to the same request priority, i.e., they must have separate allocatable areas in which to run. It is not sufficient to provide a total allocatable area at one request priority sufficient for two dependent programs since one of the programs could be assigned to the middle of this area leaving insufficient area for the other program.

16.3.5 RELEAS Requests

16.3.5.1 Monitor Request for Returning Core

All programs that have been allocated core memory, must return the allocated core to the Core Allocator, when they are finished. This includes all mass memory resident programs.

The calling sequence is shown below:



RC is the request code twelve (12) for returning core.

X is an absolute/relative indicator.

R is the return control indicator.
If R=0, control is given to the dispatcher after core is returned. This is the value of R to be used when

a program returns the core in which it resides. Since the core will be re-allocated, the program residing in it may be destroyed. Thus control is not returned to the program but to the Dispatcher instead. Otherwise R-1, control is given to the user at the next instruction.

C specities the area being returned.

If C_{15}=0, X is ignored and C_{14}-0 is the absolute core address of the area being returned. {absolute direct}

If C_{15}=1 and X=0, then C_{14} 0 is the location that contains the absolute core address of the area being returned. {absolute indirect}

If C_{15}=1 and X=0 then C_{14}-0 is a 15 bit relative address which when added to the address of the parameter list gives the core address of the area being returned {relative, direct}

Note that relative indirect is <u>not</u> allowed.

Notes on returning core:

User programs must return each piece of core which they have been allocated. Otherwise the piece of core will remain allocated indefinitely. Each piece must be returned once only.

A check is made to determine if the area of core being returned belongs to the allocatable area. If the area of core being returned is outside the allocatable area, then the request is ignored and control does not come back to the user, but instead goes to the Dispatcher. Using this feature all programs, whether mass memory or core resident, can be written identically. At the end of a program, the RELEAS request is made with R, the return indicator, set to zero, and C specifying the start of the program. For core resident programs no core is returned and control goes to the dispatcher. For mass memory resident programs, the core is returned and control is given to the dispatcher. The coding for both core resident and mass memory resident routines is the same.

16.4 SPACE REQUEST PROCESSOR

The SPACE Request Processor is entered in the same manner as the R/W Processor. Its purpose is to set necessary parameters {logical unit number, etc.} so that the R/W Processor can complete processing of the request. In addition, this processor contains the block of core controlled by the Core Allocator and the restart program.

16.4.1 External Symbols

CKTHRD Routine in R/W Processor which checks for non-zero thread.

LEND Address of LOC that contains LAND in DRCORE.

SAVLU Location in R/W Processor to which the SPACE Request Processor exits.

CALTHD Core allocator thread location in SYSBUF.

RPMASK Mask for request priority

IDLE The level -1 idle loop.

DTIMER Entry point to diagnostic timer

16.4.2 Internal Symbols

AREAC Start of allocatable core area.

AVCORE Size of the allocatable core area.

LAND End of the allocatable core area

AREA1, AREA2, AREA3, AREA4 Size of areas 1-4.

16.4.3 Restart Routine

Since this program is operated once immediately after autoload, it is located in the block to be controlled by the core allocator.

It is entered via the following procedure when the system is on mass storage.

1. Master clear the machine.
2. Depress the autoload button on the mass storage device.
3 Depress the run switch. This causes the machine to execute a program which reads the resident portion of the system from mass storage. When this is done, the program jumps to the address specified in location 1, which is the address of the restart program.

The restart program performs the following operation before jumping to the idle loop.

1. Protects all locations which must be protected and unprotects all others

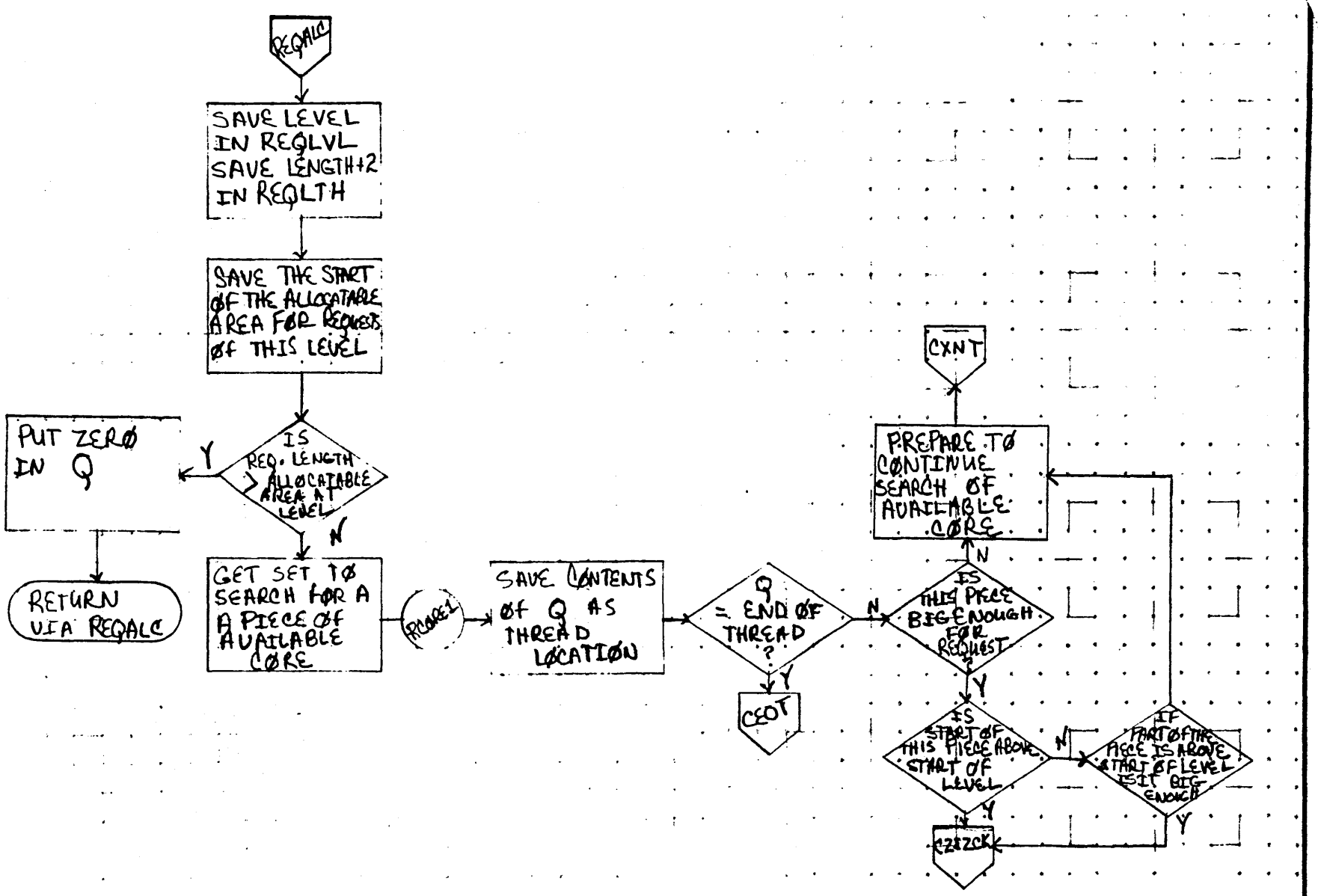
2. Enables the timer interrupt and initiates the diagnostic timer if present.
3. Requests that the protect switch be activated.

The 1573 Line Synch. Timing Generator (timer) is assumed to be interfaced via a 1570 Data and Control Terminal (DCT) that is assigned to Equipment No. 8 It is started by an output with A=A000$_{16}$. If this output results in a reject, the following message will be printed on the output comment device:

TIMER RJ

This message will occur if the Timer is not present or if the 400hZ power supply is switched off or the equipment code assigned to the DCT is not 8.

The message PP is then typed to request that the operator set the protect switch ON and enter an M. On input of an M CR the Restart program exits by a jump to the level 1 idle loop IDLE.

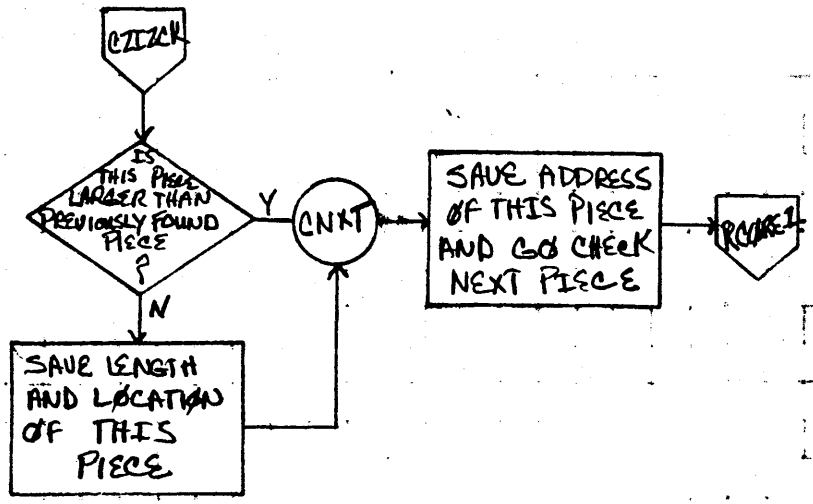


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALCORE		PROJECT MGR.			
		PAGE 1 OF 4	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



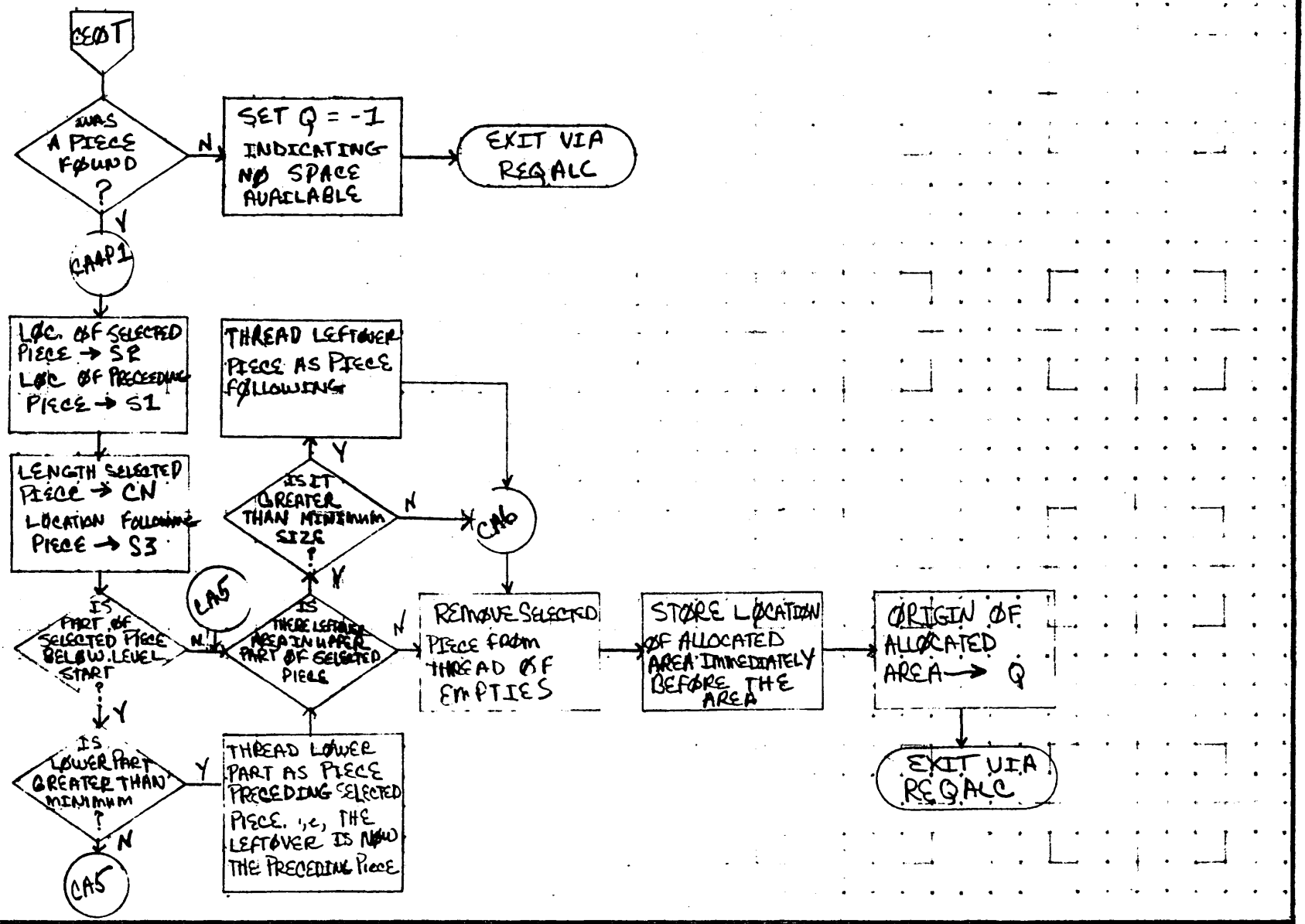
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ALCORE		PROJECT MGR.		
FLOWCHART ☐		PAGE 2 OF 4	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AA1385 (FORM CA127-1)

PRINTED IN U

MAR 5 1971

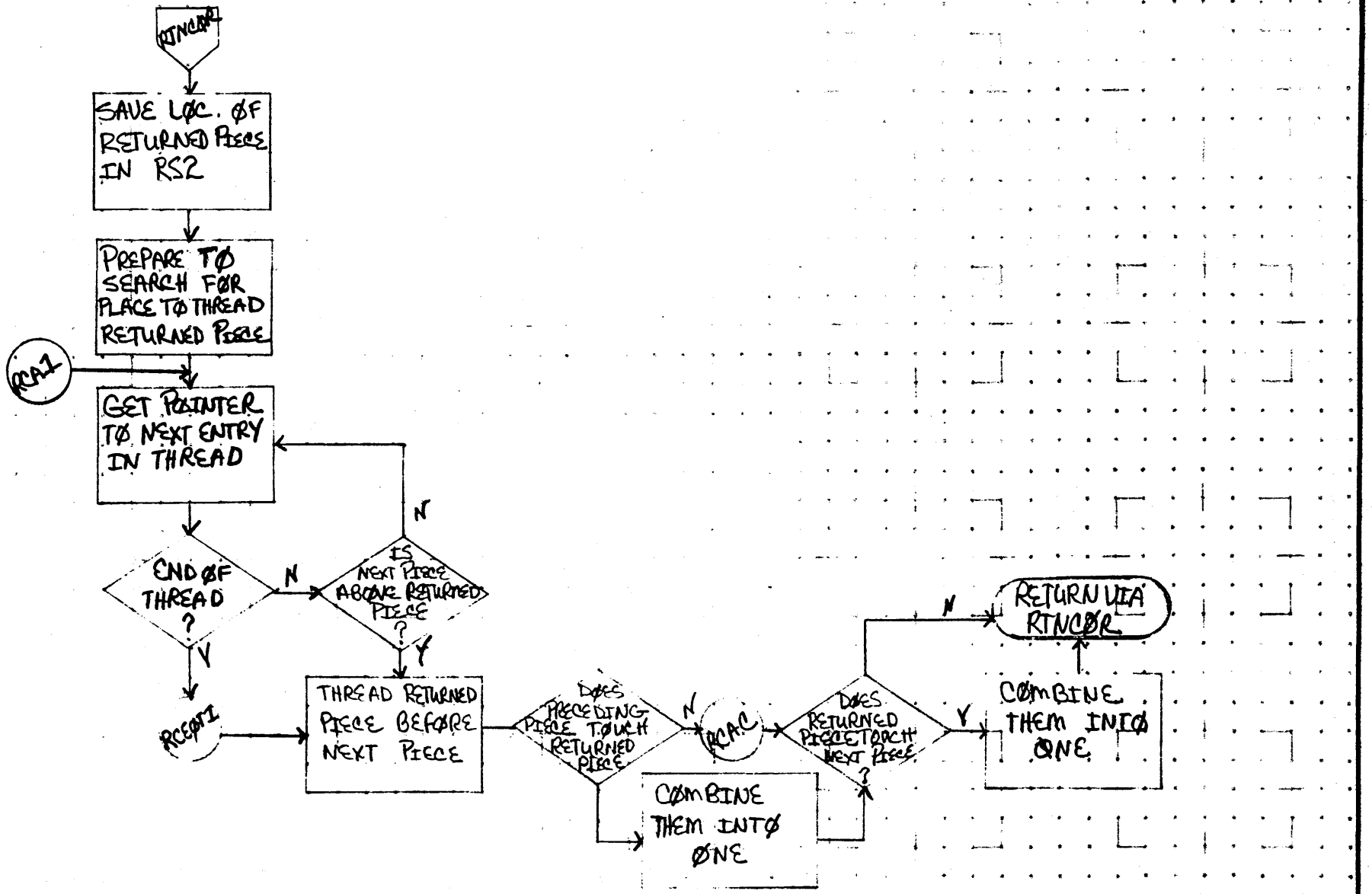
J.6.70



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ALCØRE		PROJECT MGR.		
		PAGE 3 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

16.21



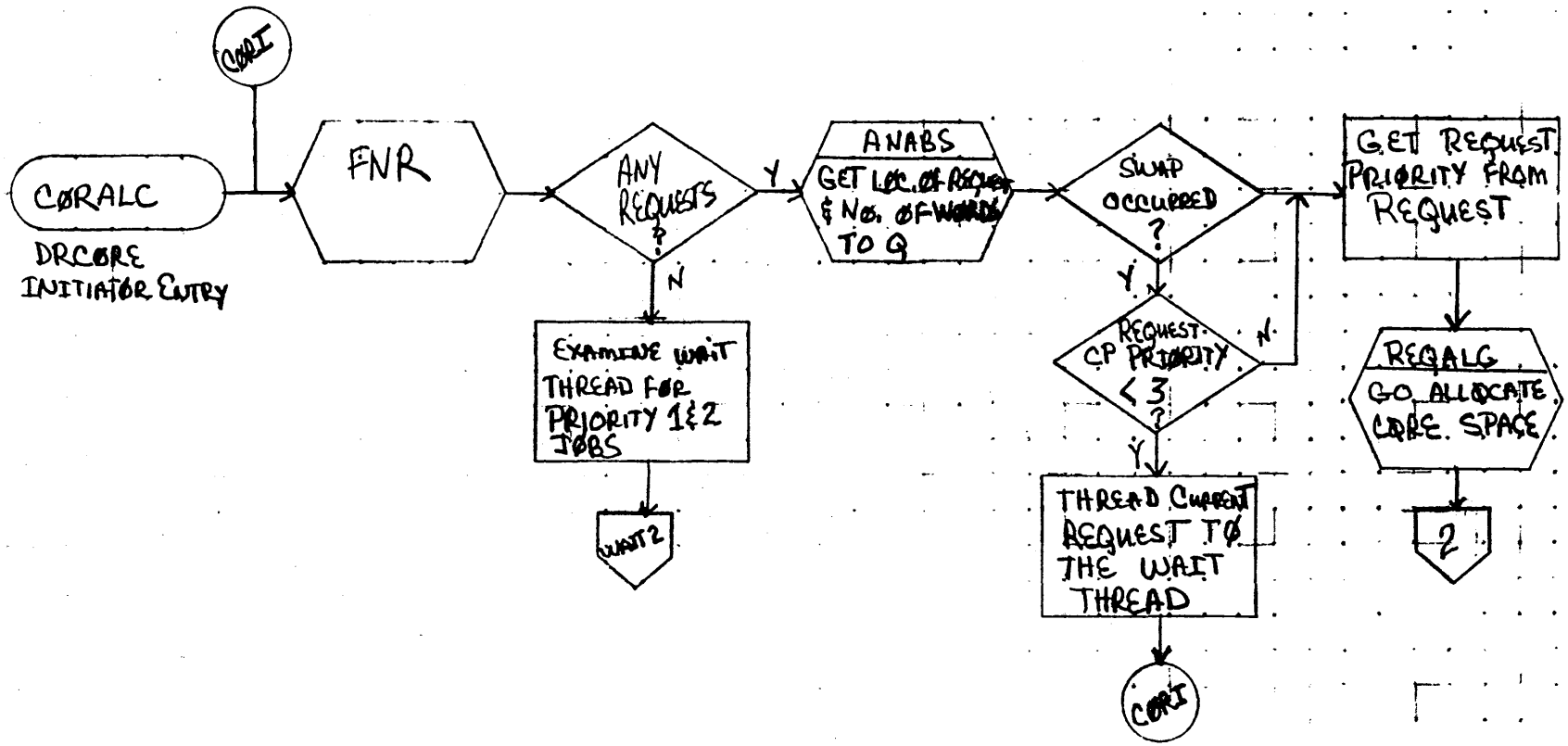
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ALCØRE	PROJECT MGR.			
	PAGE 4 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORM CA127-1)

PRINTED IN U

MAR 5 1971

168222

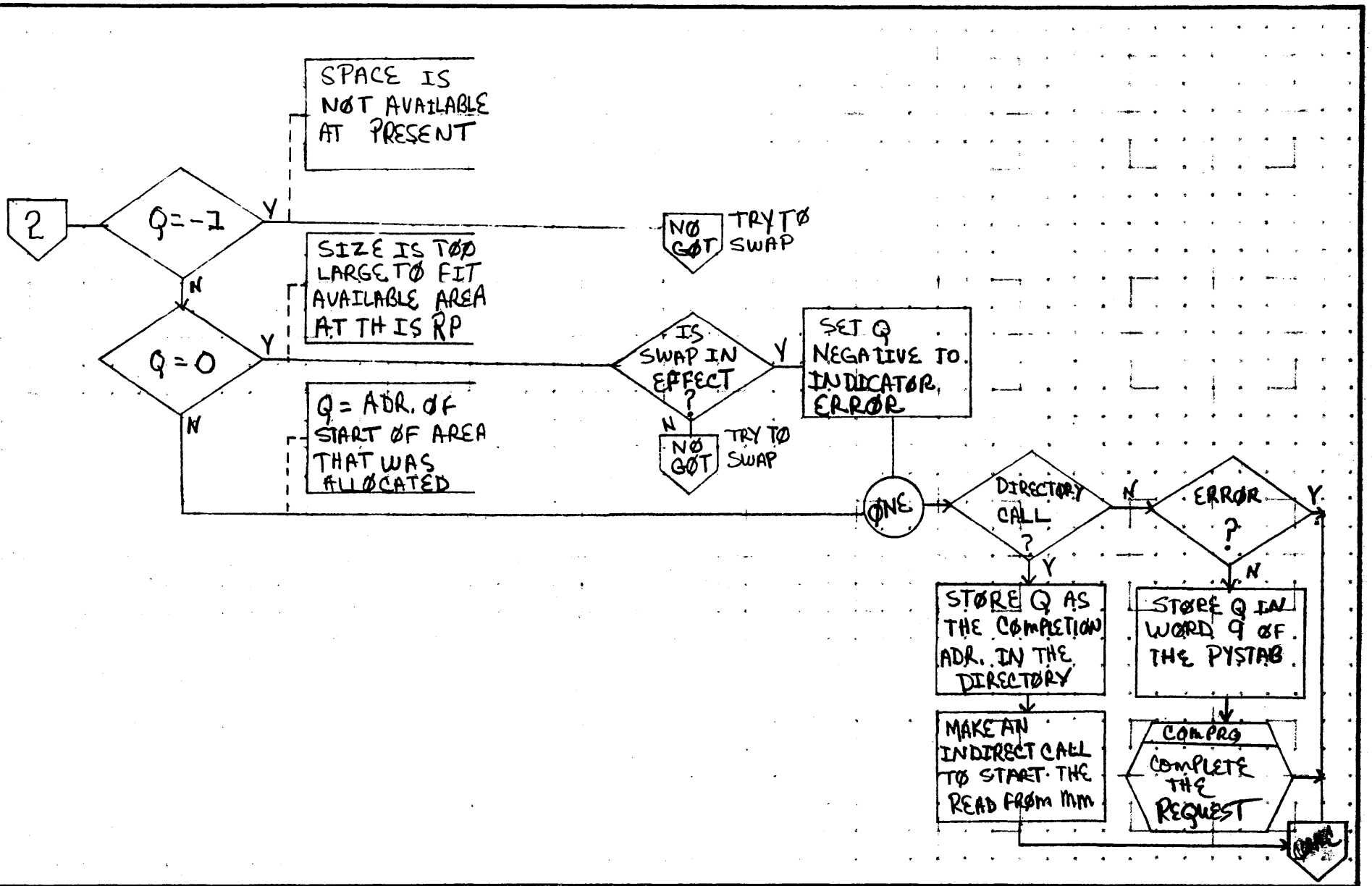


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DRCØRE		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

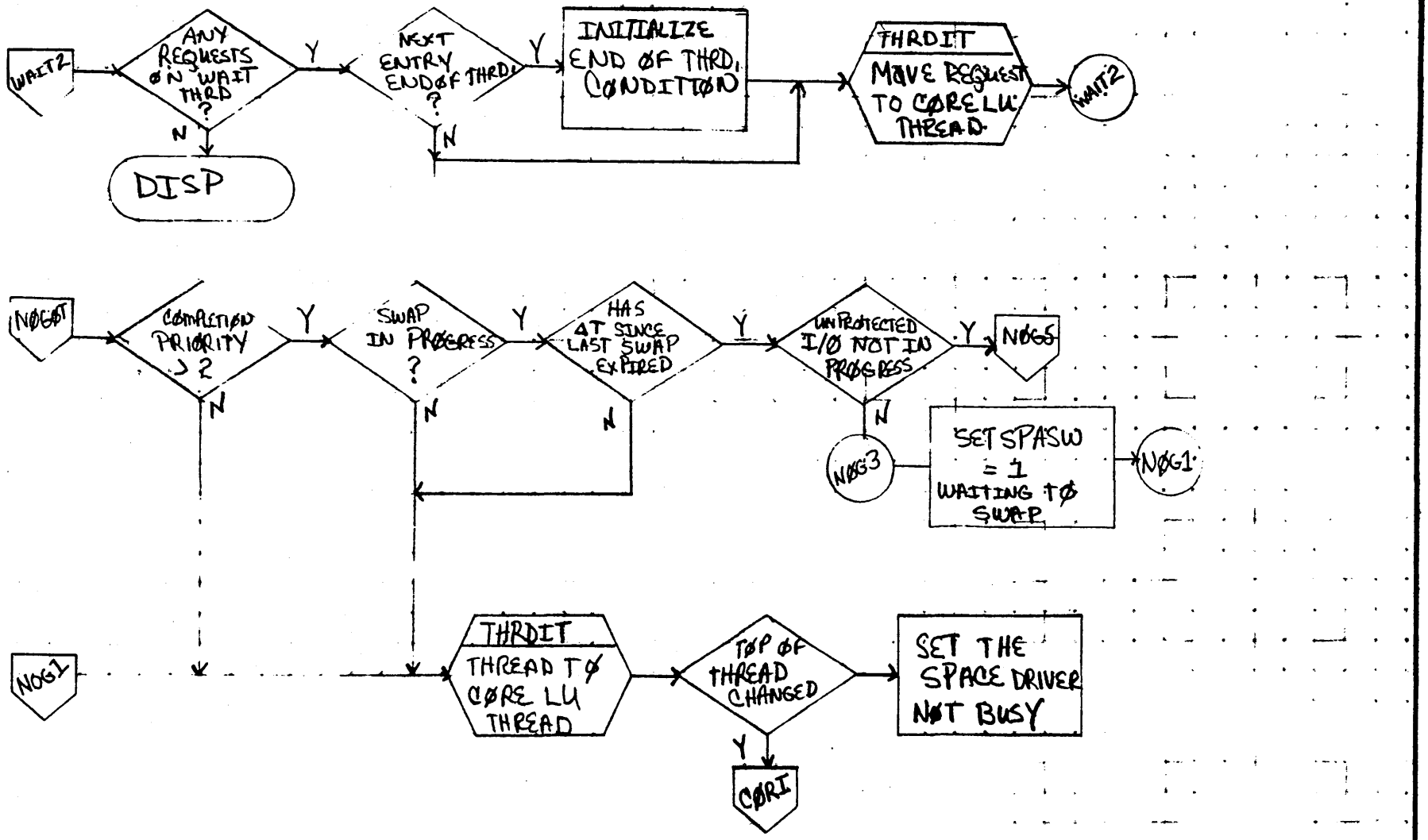
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IM	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DRCØRE		PROJECT MGR.			
		PAGE 2 OF 6	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971



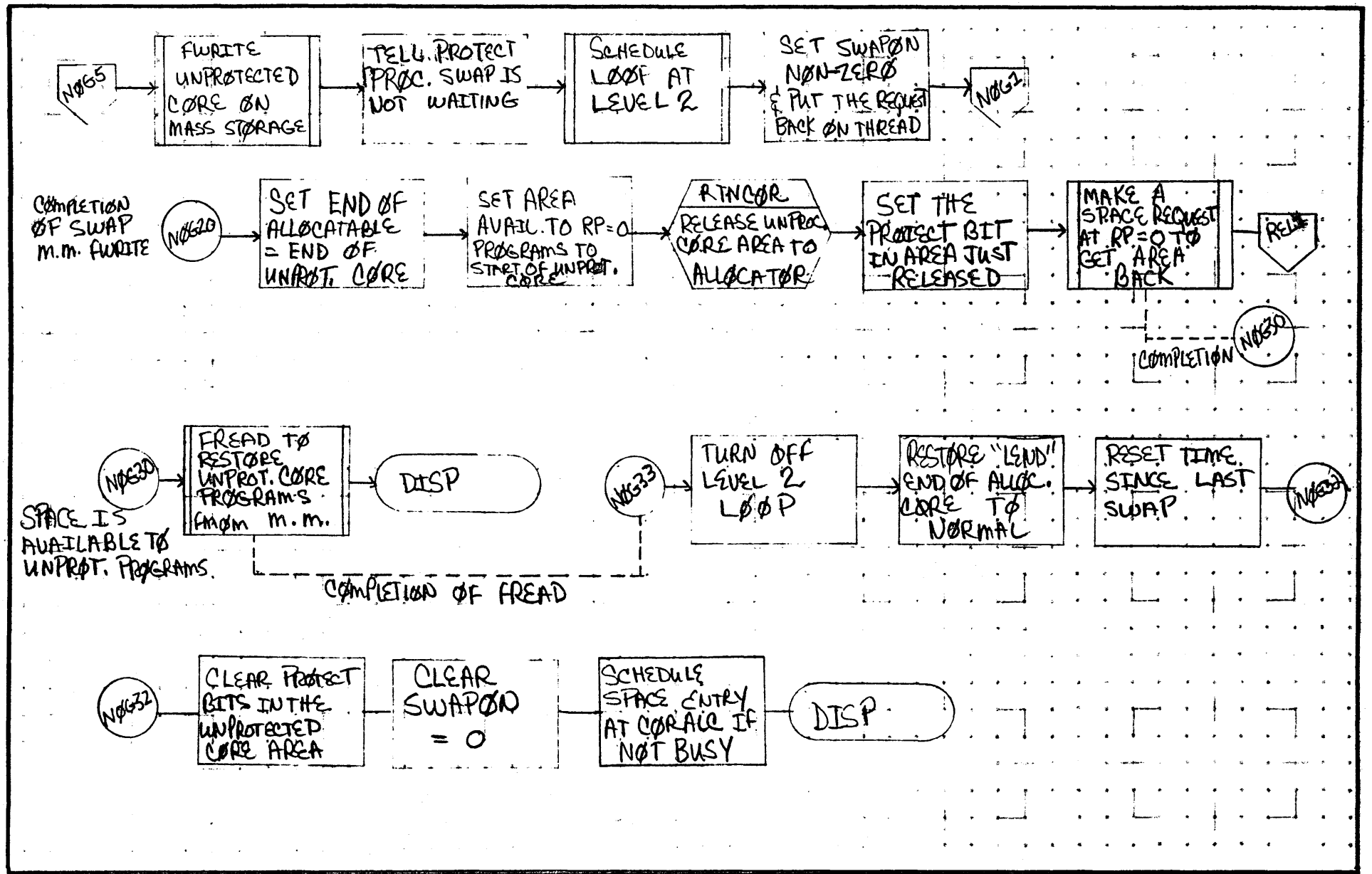
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS ims	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DRCORE		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

16.25

AA1385 (FORMERLY CA127-1

PRINTED IN U.S.A.

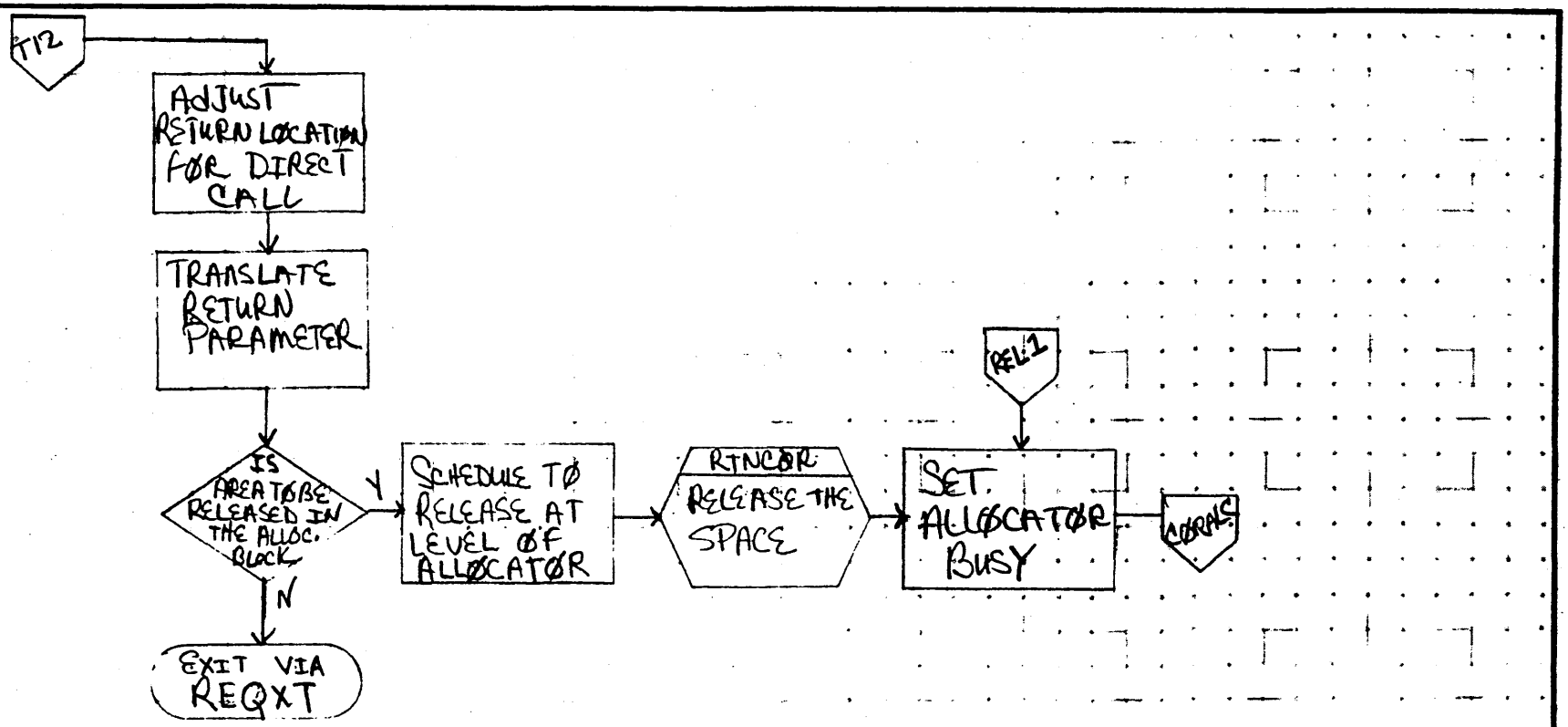


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DRCØRE		PROJECT MGR.			
FLOWCHART ☐		PAGE 4 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

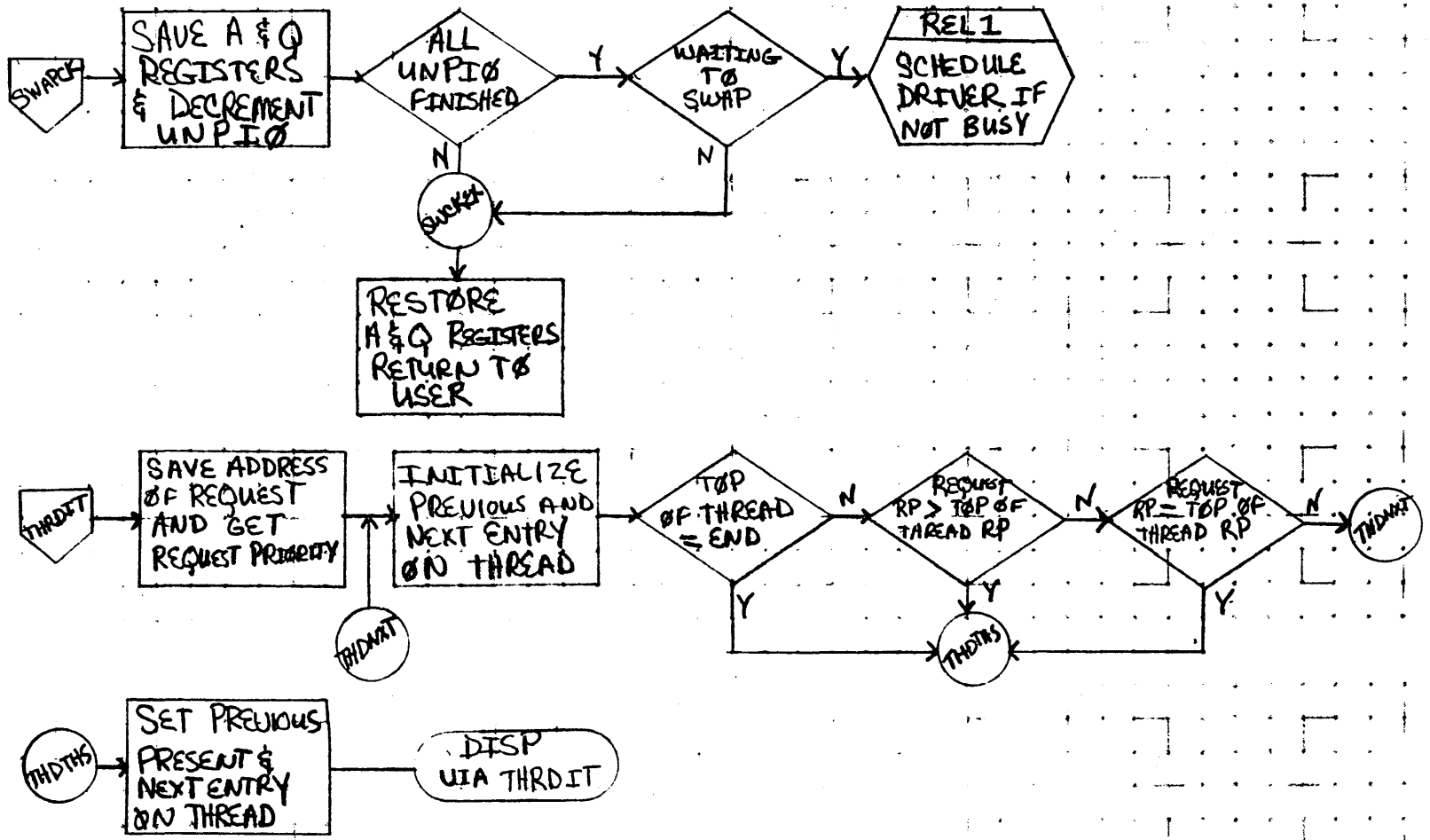
AA1385 (FORM CA127-1)

PRINTED IN U.S.A.



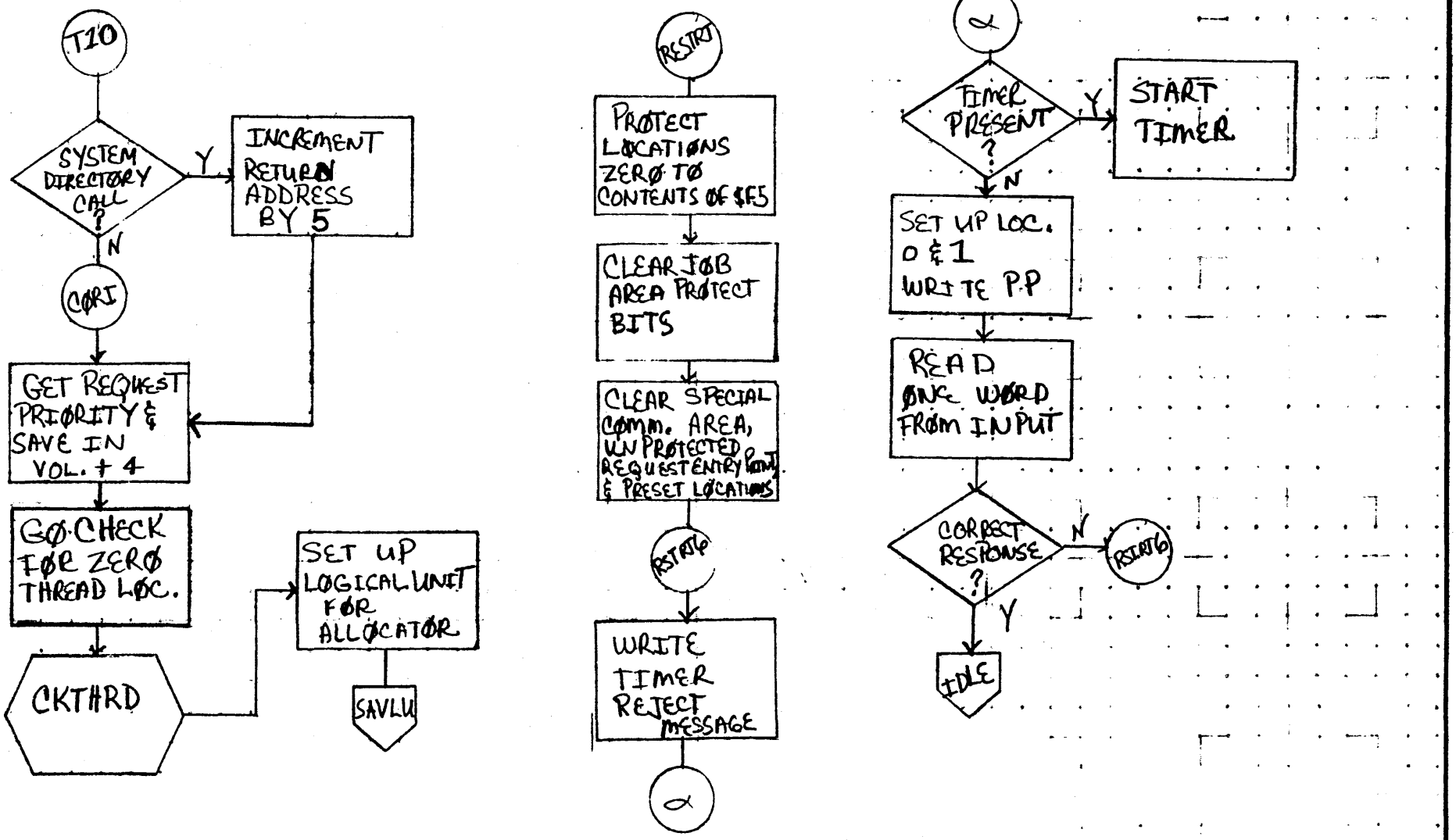
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	DRCØRE	PAGE 5 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 19/1



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DRCØRE	PROJECT MGR.			
	PAGE 6 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971
16:28



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	SPACE	PAGE | OF |	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

16.29

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

17.0 PARAME - Parameter List Conversion Routines

20.1 Program Function

Four routines are provided in this module for the purpose of decoding parameters in monitor requests. The parameters decoded are:

LU - Logical unit
S - Starting address of I/O buffer
N - Maximum number of words to be transferred
C - Completion address

20.2 Entry Interfaces

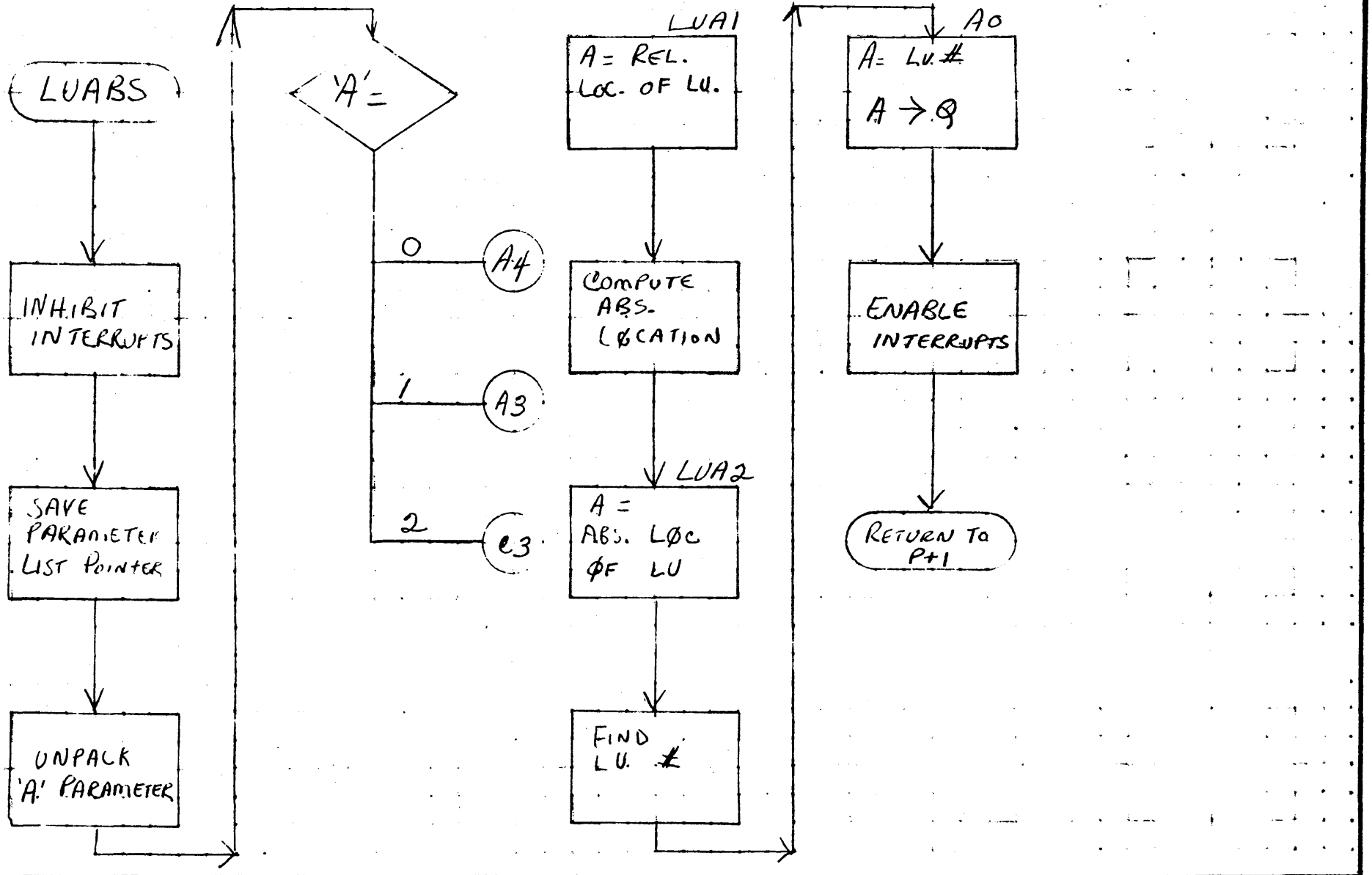
The entry points are as follows:

LUABS
SABS
NABS
CABS

The routines are entered by a RTJ to one of the above entry points or by an indirect RTJ to locations BC-BF in LOCORE. On entry Q contains the location of the parameter list.

20.3 Exit Interfaces

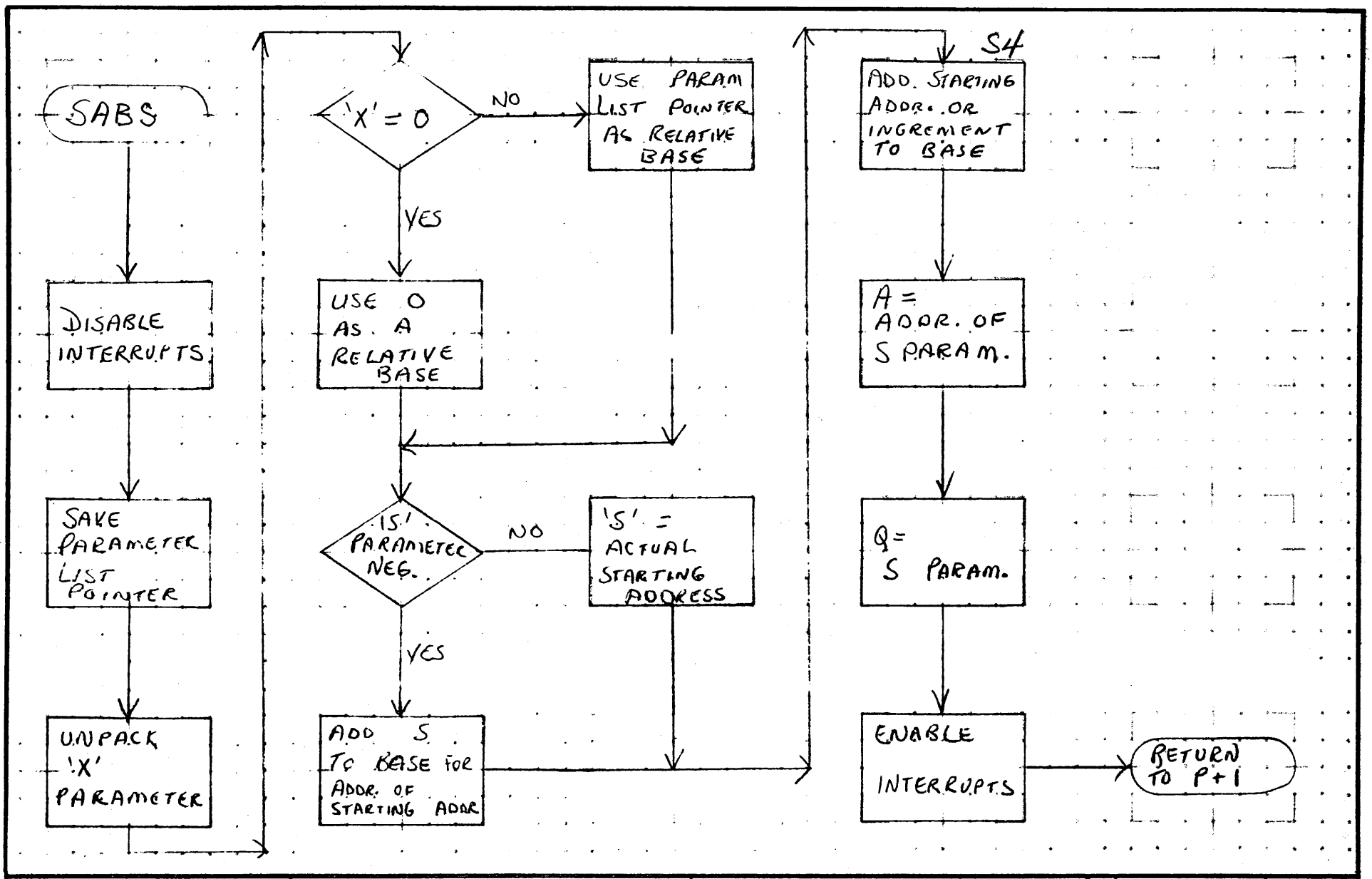
All routines exit with the decoded parameters in Q. In addition, SABS exits with the location of the S parameter in A. A and Q are not conserved, but I remains unchanged during the routines.



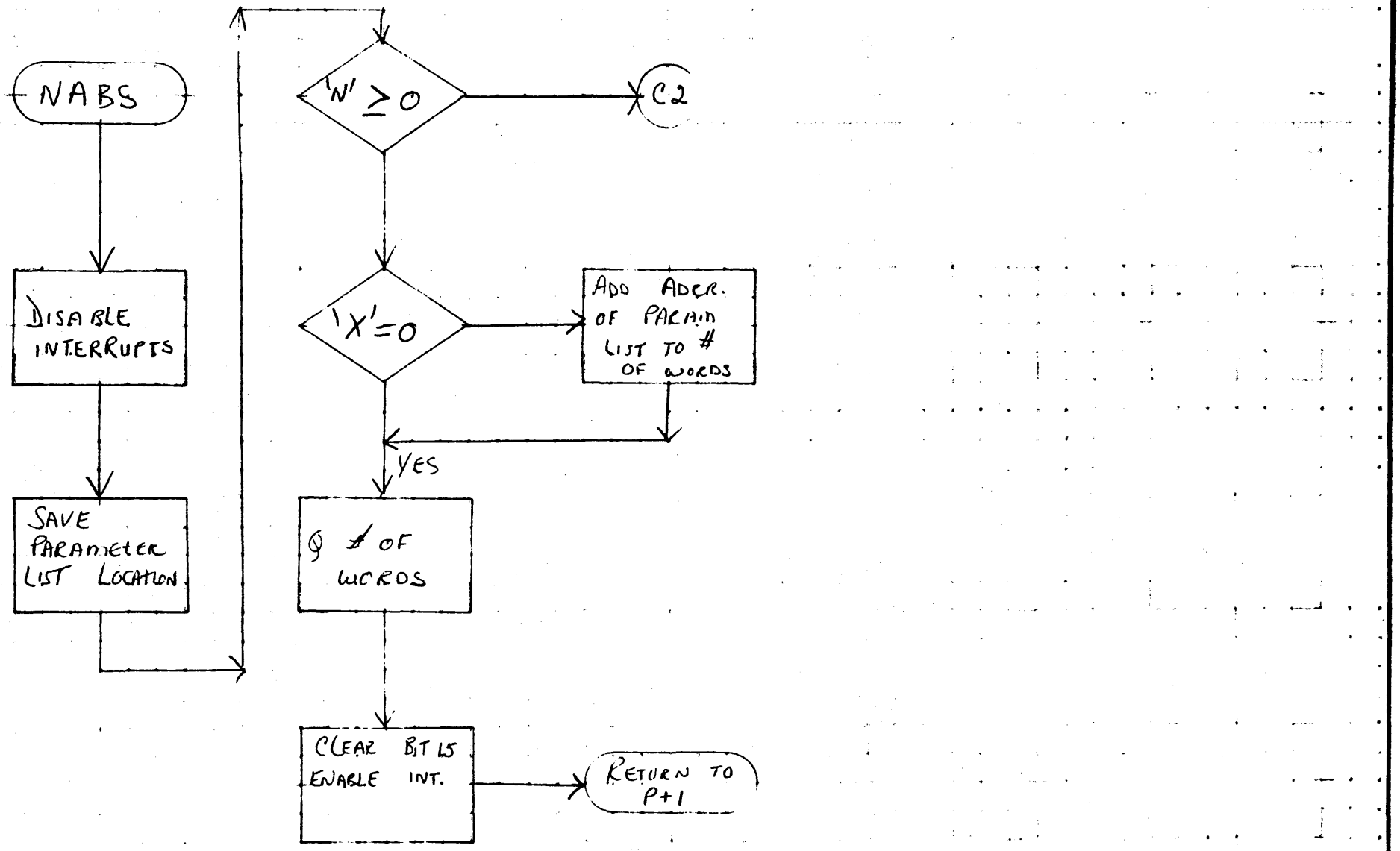
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PARAME	PROJECT MGR.			
	MSOS 3.0 PAGE 1 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

17.2



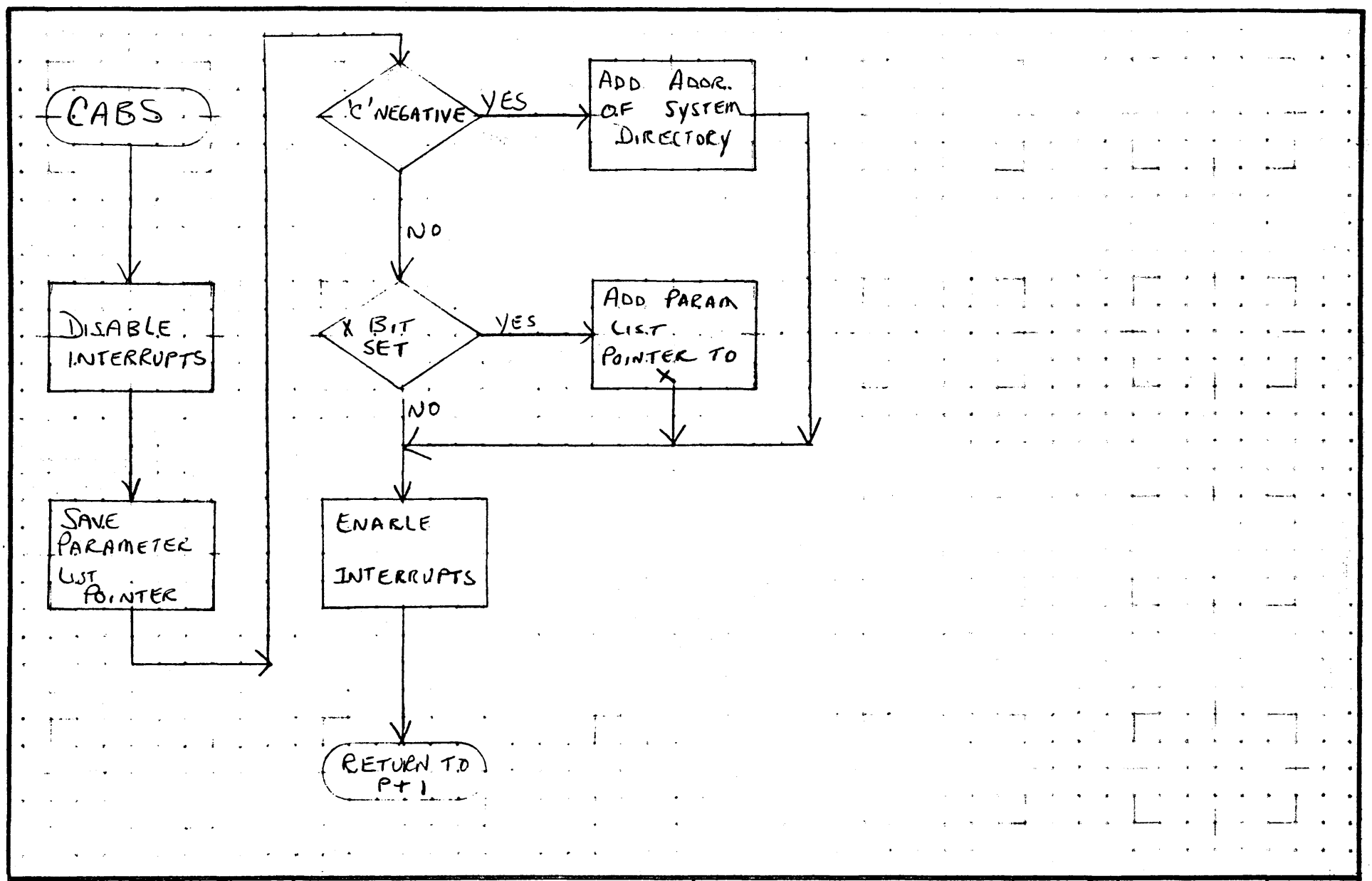
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS 1015	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PARAME		PROJECT MGR.			
FLOWCHART ☐	MSOS 3.0	PAGE 2 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PARAME	PROJECT MGR			
	MSOS 3.0 PAGE 3 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

127-4



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PARAME		PROJECT MGR.			
FLOWCHART ☐	MSOS 3.0	PAGE 4 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

18.0 TRVEC - Transfer Vector Table

18.1 FUNCTION

This module functions as a communications area between the mass memory resident Job Processor and Library Editing modules and core resident modules. TRVEC divides functionally into three parts: (1) the vector table - a table of flags and addresses necessary for communication between core resident and mass resident programs, (2) JBCNCL - routine to schedule either JBKILL or PROTEC and (3) JPRETN - routine to facilitate return to either T7 or JPLOAD from the Loader.

18.2 ENTRY POINT NAMES AND FUNCTIONS

TRVEC	Entry Point to TRVEC module.
TRANV	Contains the absolute address of the Job Processor transfer vector table.
JBPROE	Location in JOBENT to Process Job Processor modules.
ERRMSG	Absolute address of ERRM routine in JOBENT.
MIBUF	Absolute address of MIPBUF buffer in JOBENT.
TRNVEC	Absolute address of TRNTBL buffer in JOBENT.
LIBET	Contains location in LIB in JOBENT module (routine to schedule LIBEDT).
RECOV	Contains absolute location of RECOVR in JOBENT module (routine to schedule the Recovery program)
RELSLA	Location in JOBENT to release a specified file.
JOBIND	A flag which indicates whether the Job Processor is in core.
UNPIO	Contains the number of unprotected I/O calls pending. Used by the Core Allocator to indicate when a swap may be made.
IUP	Pointer to the comment device. Used by the Job Processor (initially set to 18FD).
SPASW	Flag which is set non-zero when the Core Allocator wishes to swap core.
NSTACK	Max. number of stacked requests.

VRESET Flag when set indicates to JBKILL to change IUP back to its initial value {$18FD_{16}$}.

PCOMFL Protected Common Flag {LIBEDT}.

PRORET Contains the absolute address in JPLOAD to return to after PROTEC is scheduled.

JPSWT Temporary location for MIINP buffer address or an index to the tranta table or a negative value set by JOBENT or JBKILL.

FILE1 Contains the address of the area in allocatable core where JOBENT, T11, T7, T5 and T3 are brought into.

FILE2 Address of the area for the Job Processor and LIBEDT modules.

FILE3 Address of the area for the Protect Processor and JBKILL modules.

FILE4 Address of the area for the tape driver buffers.

LOCF Contains the location of the routine in the Protect Processor which puts out J01 and J02 error messages.

LPTRS Location of PTRs in the Protect Processor.

SWTCH Switch to lock out Job Processor when LIBEDT or the recovery program is in operation.

LOADIN Flag for protect processor to allow the loader to read and write below the scratch area on mass storage.

UNPTIM Number of unprotected timer requests outstanding {checked by JBKILL before job is terminated}.

JKIN Contains the address of JBKILL when in core.

JBCNCL Job Processing cancel routine

JBCNFG Set when JBKILL is active.

JPRETN Routine to interface between loader and Job Processor modules.

JPRET1 Contains a return address to T7 or JPLOAD.

18.3 EXTERNALS AND DESCRIPTION

PROTEC - An entry in the system directory.
The mass memory resident Protect Processor.

18.4 ENTRY INTERFACES

None

18.5 EXTERNAL INTERFACES

Within the JBCNCL routine an exit is made to the dispatcher to schedule either PROTEC or the starting address of JBKILL. Within the JPRETN routine an exit is made to either T7 or JPLOAD depending on what address is stored in JPRET1.

18.6 GENERAL PROGRAM INFORMATION

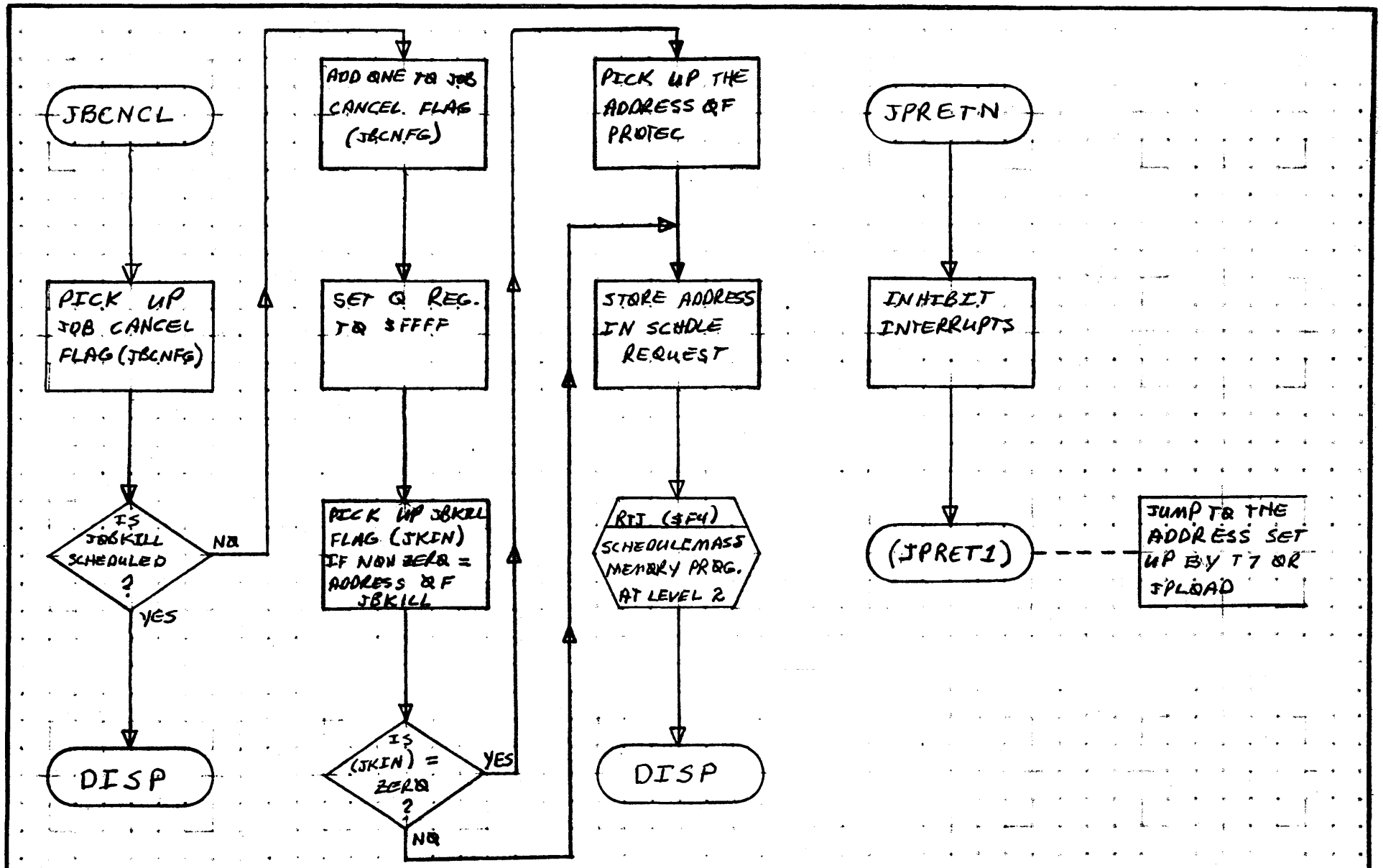
18.6.1 The equate of NSTACK to 5 sets the maximum number of I/O requests that may be stacked at one time.

18.7 GENERAL DESIGN PECULIARITIES

18.7.1 If a new entry is made to the vector table, it must be inserted following the constant RELS1A. The constants from TRANV through RELS1A are part of a table transfer from JOBENT.

18.8 PROGRAM LOGIC

18.8.1 The JBCNCL routine first checks the job cancel flag (JBCNFG) which is set if JBKILL has been scheduled. If scheduled jump to the dispatcher. If not scheduled add one to JBCNFG, set the Q register negative to indicate that JBKILL is requested and check the JBKILL in core flag (JKIN). JKIN, if non zero, will contain the starting address of JBKILL. If JBKILL is in, schedule that address and if it is not in, schedule the Protect Processor and jump to the Dispatcher - both the address and PROTEC are scheduled at level two so that no other Job Processor routines may interrupt them.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TRVEC		PROJECT MGR.		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

8.4

19.0 TMINT

TMINT processes TIMER requests, timer interrupts and delay expiration.

19.1 External Symbols used by Timer Package

SCHERR Used to exit if the schedule is full

TIMACK Acknowledge code for time interrupts

19.2 Time Request Processing

19.2.1 Entry Interface

Entered from the monitor entry for requests via a jump. ▽I▽ contains location of volatile, and ▽A▽ contains location of the request.

19.2.2 Exit Interfaces

Exit is made to SCHERR if no schedule stack space remains open. Exit is made to request exit after the request has been added to an appropriate stack.

19.2.3 Internal Operation

On entry, the request processor translates the completion address and attempts to fill an empty schedule stack entry with a SCHDLE request at the level specified in the TIMER request. If no empty exists, exit is made to SCHERR.

The newly filled schedule stack entry is then threaded to one of 4 lists depending on the ▽U▽ parameter. The callers delay time is added to the stack entry as the ▽Q▽ parameter. Exit is then made to the request exit.

19.3 Time Interrupt and Expiration Processing

After the interrupt is acknowledged, each of the counters for the 4 lists (see 8.4.3) are examined to see if one count for that list has expired. If no, the respective count is decremented and exit is made to the dispatcher. If the count is expired, it is reset and the threaded list corresponding to that counter is examined. The delay in each member of the list is decremented. Those delays which are decremented to zero cause SCHDEL requests which result in operation of the concerned program. When this process is complete, the next counter is decremented etc.

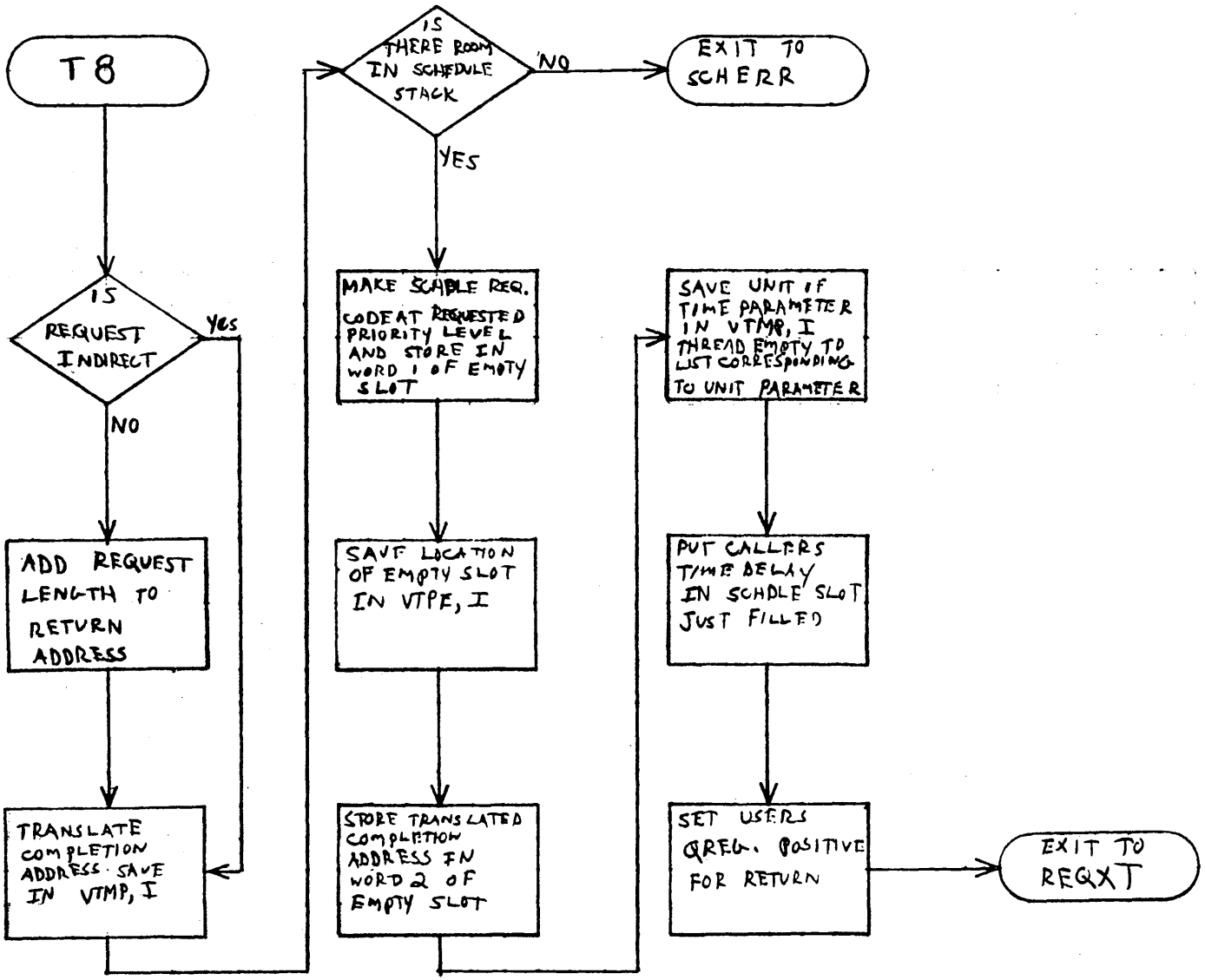
If the acknowledge of a time interrupt is rejected, the program will exit to the dispatcher.

19.4 Installation

The TMINT module may be added in the same way with one additional requirement. The entry point name ▽TIMINT▽ must set as the primary processor for the interrupt line where time interrupts are trapped. This is done by suitable re-assembly of LOCORE module of the system .

19.5 Internal Symbols used by TMINT

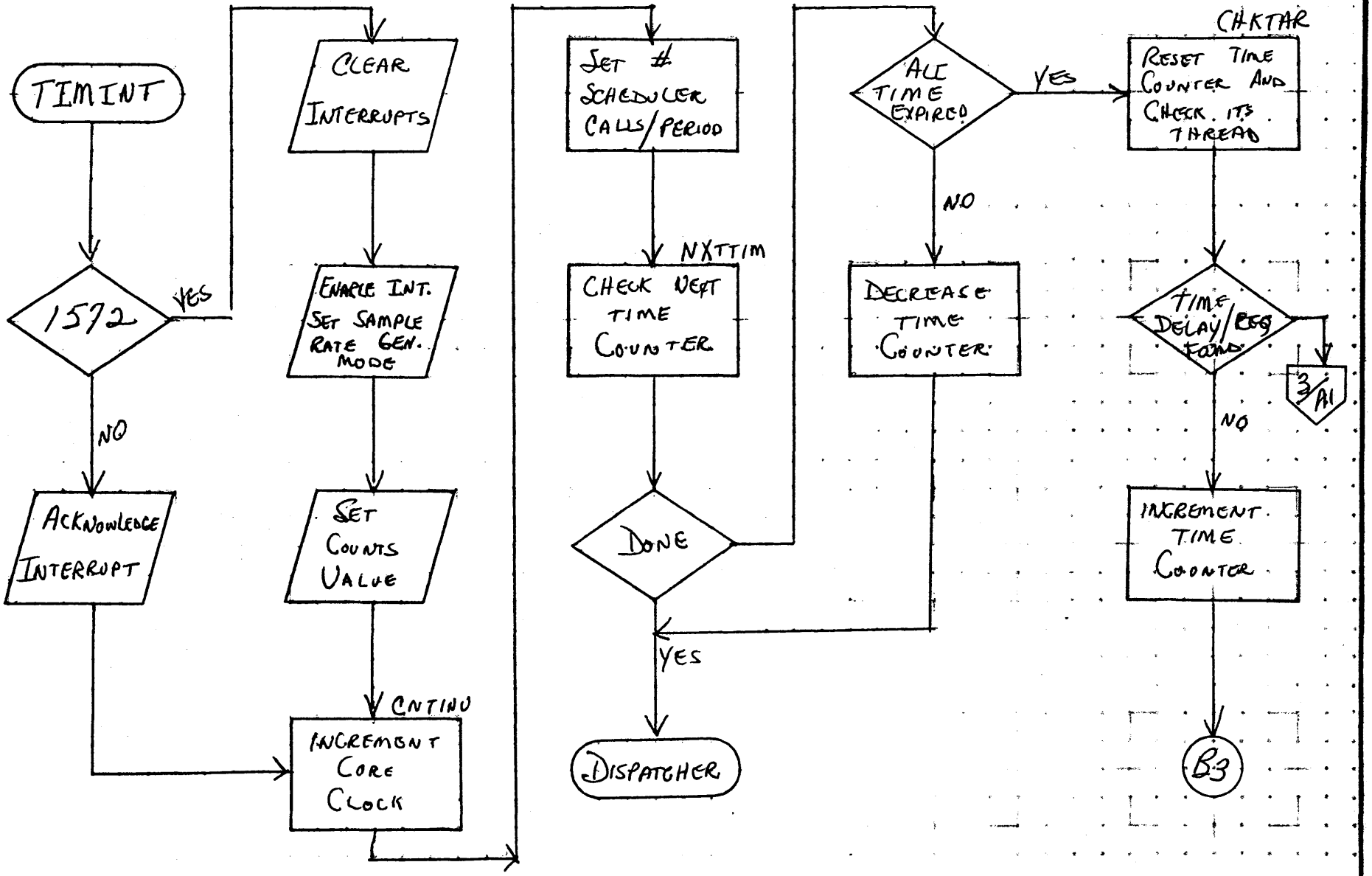
These symbols are defined via EQU pseudo operation and can be easily deduced from the listing.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TMINT		PROJECT MGR.		
		PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971 MAR 5 1971

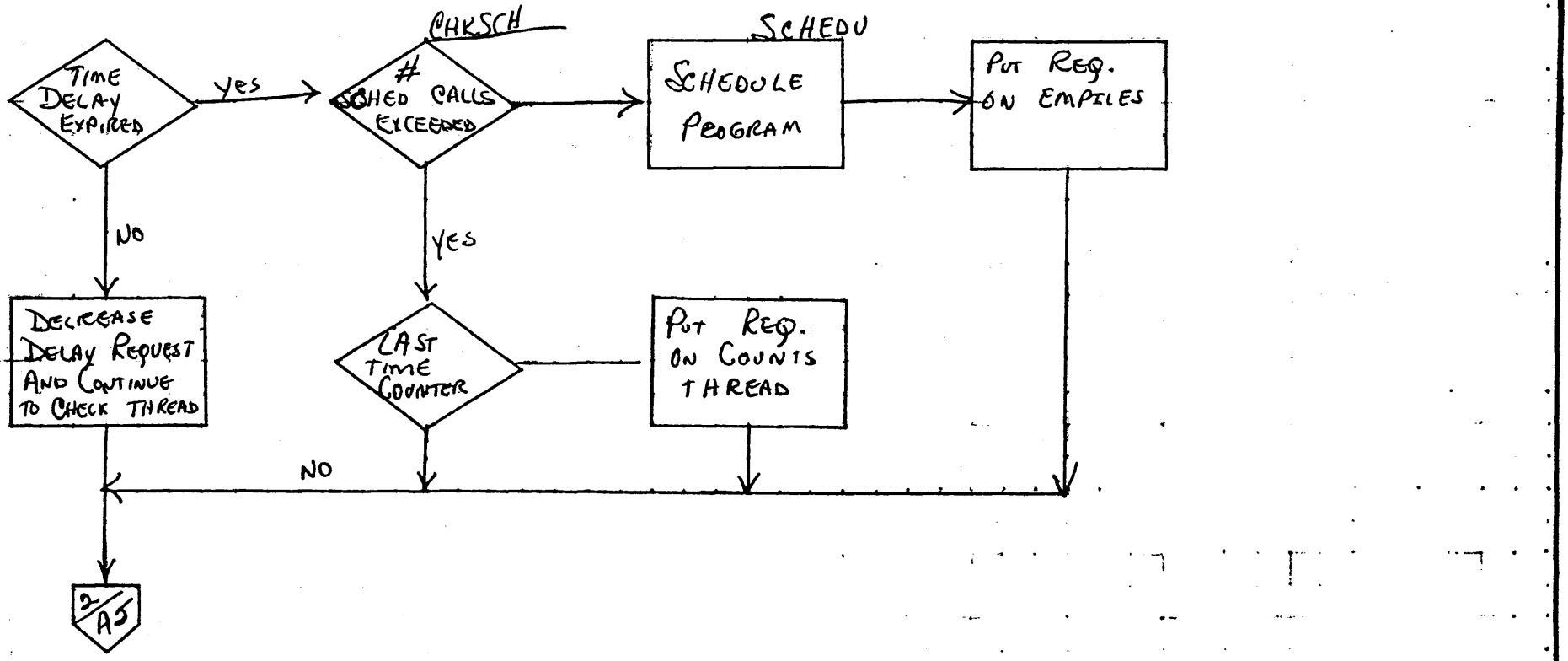
19.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TIMINT		PROJECT MGR.		
		PAGE 2 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORM CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TIMINT		PROJECT MGR.		
		PAGE 3 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

1700

20.0 DTMER

This diagnostic timer module processes I/O hang ups.

20.1 External Symbols used by Diagnostic Timer

The starting address label for each PHYSTB entry, to be interrogated by this module, is declared as an external symbol.

20.2 Diagnostic Timer Operations

This module is operated periodically as the result of a TIMER request generated by itself. The first TIMER request is made in the startup routine at autoload time. On entry, this module decrements the clock cell {in PHYSTB} of each non-idle device. If the clock cell becomes minus, the device is assumed to be hung up and the error entry to the driver is scheduled. When this process is complete for each device, the module makes a TIMER request, to cause its next execution, and exits to the dispatcher.

20.3 Installation

The Diagnostic Timer module need only be added to resident {ML statement} during system initialization.

20.4 Internal Symbols sued by the Diagnostic Timer

EDCLK Index to diagnostic clock in each PHYSTB entry

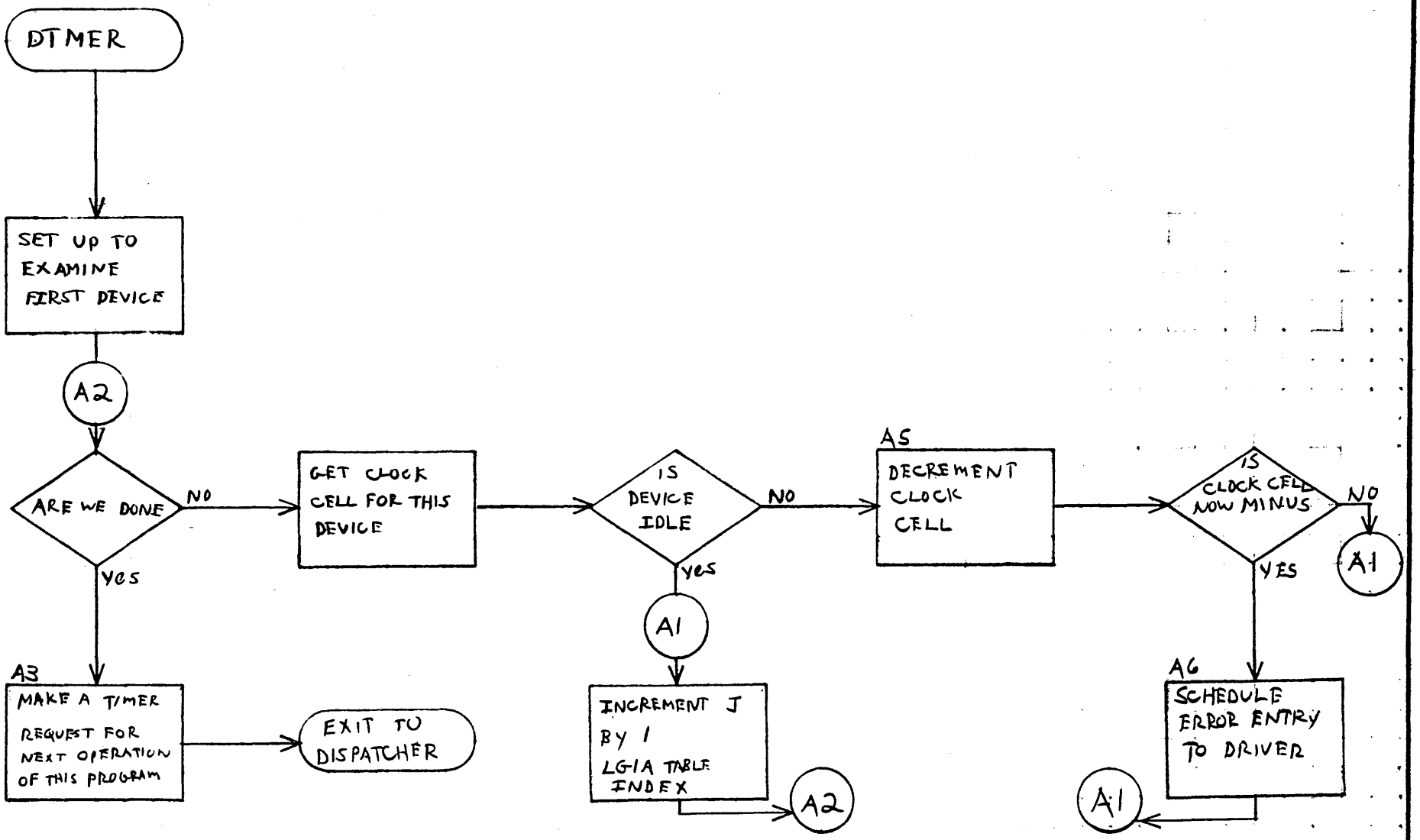
EDPGM Index to location of error routine in each PHYSTB entry

SECOND Number of timer pulses per second

DELAY Number of seconds between successive operation of the diagnostic timer

DTLVL Priority level at which the diagnostic timer operates. {assembly value is 13.}

NUMPU Number of physical devices



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE DTMER		PROJECT MGR.		
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

20.2

21.0 MINT

The manual interrupt program is entered at the level of the teletype driver (or equivalent) from that driver. On entry at MI the flags MIB and MIBX are checked. If either flag is set the entry is ignored and the program exits to the dispatcher. Otherwise the manual interrupt program continues by scheduling itself down to level 3. Note that flags MIB and MIBX are not set or tested by the teletype driver, the driver simply schedules MI at its own level.

No attempt is made to allow the manual interrupt to interrupt the current request in progress on the comment device (if any). Therefore, the current request must be completed by allowing the type out to finish and/or entering a carriage return to complete input before the Manual Interrupt Processor can output MI and request operator input. Requests in progress may also be completed by causing the comment device to fail provided it is assigned the dummy device as an alternate (see 17.0).

After scheduling down to level 3 the manual interrupt processor types "MI" on the comment device and requests an input of up to 24 characters on the comment input device. While waiting for input the program hangs in a loop at level 3 thus suspending job processing. If the input request fails it is repeated. Otherwise, thei input buffer is checked.

The first character of the input buffer (MIINP) determines whether the statement is a job processor statement, i.e., if the first character is an asterisk (*). If not an asterisk, and the program MIPR0 is present, this program is scheduled via the system directory entry for MIPR0 with Q set to the address of the MIINP buffer. MIPR0 may be a core or mass-memory resident system directory entry. If MIPR0 is not present (i.e., and unlimited external) a J05 error is typed and the program clears MIBX and exits.

If a Job Processor statement is entered and the job processor is not in core (J0BIND=0) and is not locked out (SWTCH=0) then the job processor entry module J0BENT is scheduled at level zero with Q = address of MIINP. If the job processor is already in core (J0BEND≠0) only the following job processor statements are accepted.

A) Job Processor not locked out (SWTCH=0)

*Z	Terminate Job	Schedules JBCNCL at level 2
*R	Restore l.u.	Sets MIBX Schedules RESTOR at level 3

M	Continue	No action
		Resumes job processing at level 0

B) Job Processor locked out (SWTCH=FFFF)

Library Edit or Recovery in operation

MZ	Terminate Job	Sets SWTCH=1
MR	Restore l.u.	Sets MIBX
		Schedules RESTOR at level 3
M	Continue	No action
		Resumes job processing at level 0

Any statements other than the above will cause a J05 error if the job processing is in core.

If MIBX flag is set when a high level job processor statement is accepted and is cleared by the appropriate module (RESTOR or JBCHGE) when the statement is processed. These modules run at level three and do not return control to the job processor on completion in this mode of operation. If MIBX is set further manual interrupts are ignored.

MIBX is also set when MIPR0 is scheduled and must be cleared by that program on exit.

The MIB flag is set on entry to the manual interrupt processor and cleared on exit, or by the job processor if at level zero. MIB is also set when transferring control between job processor modules to prevent an MZ from being entered at that time.

RELFLE is entered from the Job Processor to release the allocated core for each of the four possible allocated areas used by the Job Processor. The addresses of these allocated areas are saved at FILE1, FILE2, and FILE4 (see 26.2.2). After releasing the allocated core J0BIND is reset to zero.

Provision is made to force a core swap when the Job Processor is not in use in orcer to take advantage of the extra core available to the protected programs and to eliminate unnecessary swapping. This is done by routine J0BEND which is entered after executing FELFLE. A space request is made that can only be completed when swapped. Two locations will then be allocated at the end of the

swapped area. The space thus allocated is not released until a new Job Processor statement is entered and JOBENT is scheduled.

NOTE that when the Job Processor is not in use, the level two loop set in operation by the core swap will be in effect. All protected programs must be run above level two. Since the swap remains in effect any overloads in the normal allocatable area will be accommodated immediately by assigning part of the swap area.

21.1 Load Requirements

The Manual Interrupt Processor is a core resident program. It must be loaded under an ML System Initializer statement and before MM, mass storage resident Job Processor modules.

21.2 Program Entry Points

MI	Scheduled by the Comment Device Driver at the drive priority level
RELFLE	Releases core defined by FILE1, FILE2, FILE3, FILE4.
JOBEND	Forces a core swap by a special space request when Job Processing is finished.

21.3 Program Switches and Flags

MIB	Manual Interrupt Busy Switch Zero = Not busy Non-zero = Busy Must be set to zero before another manual interrupt can be processed.
MIBX	Manual Interrupt Busy Switch secondary level simulator to MIB
SVLU	Saves the Job Processor comment logical unit in case an MV statement is in effect.
MIINP	24 character manual interrupt input buffer.

21.4 External Switches and Flags

SWITCH Job Processor lock-out switch

0 = not locked out

$FFFF = locked out

1 = locked out, ¤Z has been entered

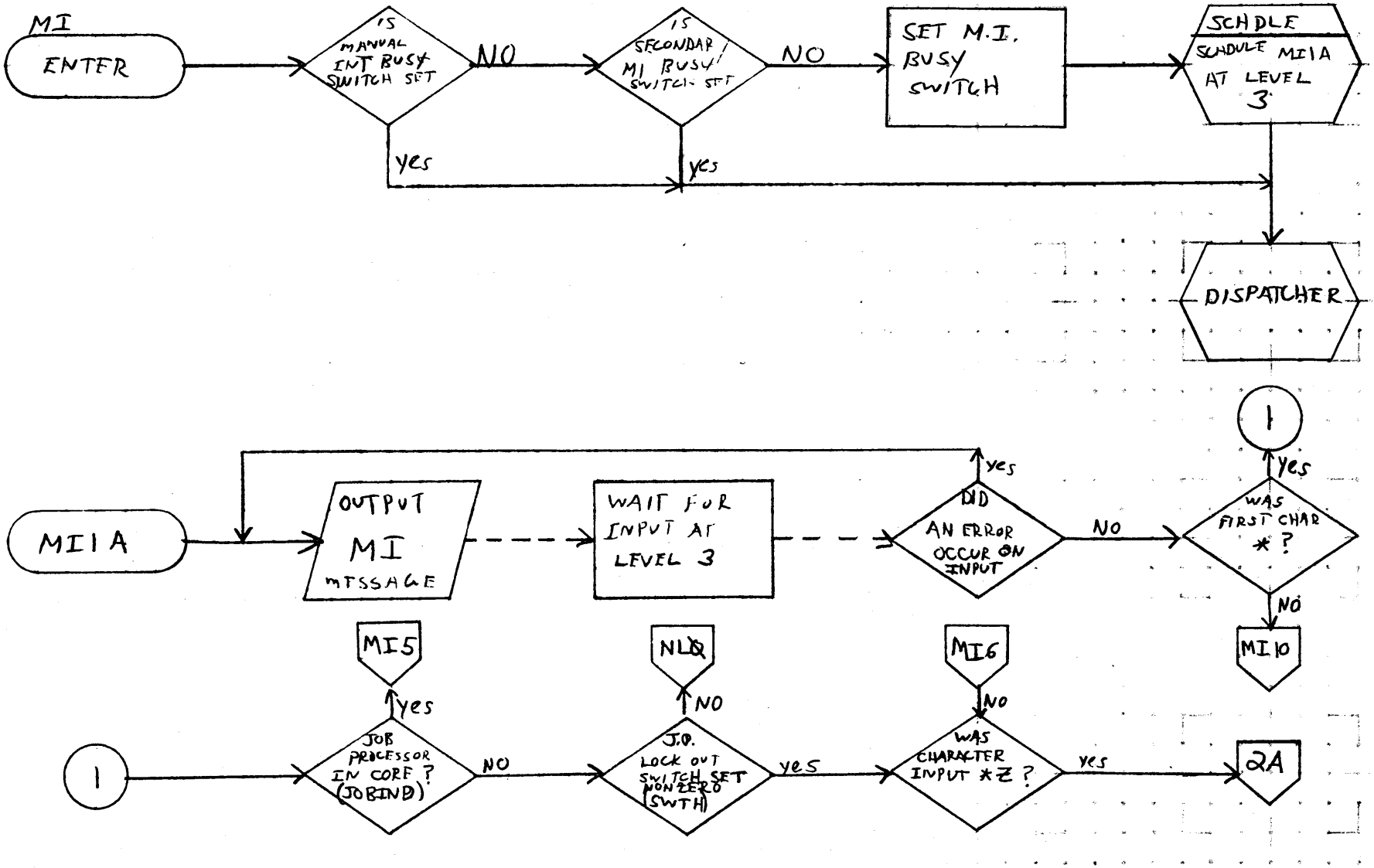
JOBIND Set by Job Processor to indicate if Job Processor in core and its entry point location

IUP Job Processor Logical Unit Number

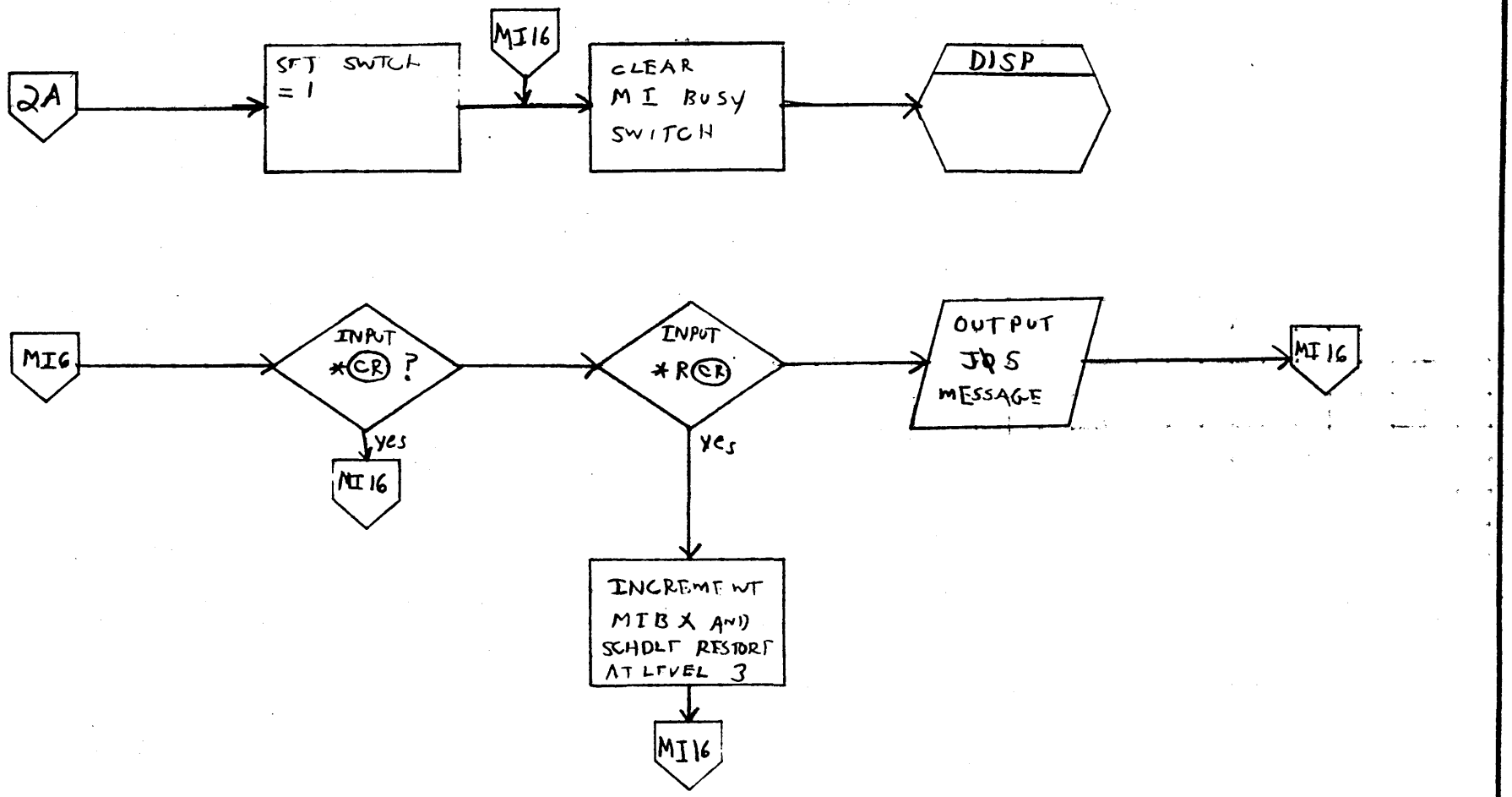
21.5 External Programs

JOBENT 1st Job Processor module to be executed. Upon scheduling, @ contains location of MIINP buffer.

MIPRO System Directory Entry Name for the program to be scheduled when a statement is entered that does not begin with an asterisk. @ contains the location of the MIINP buffer entry. MIPRO must clear MIB on completion.



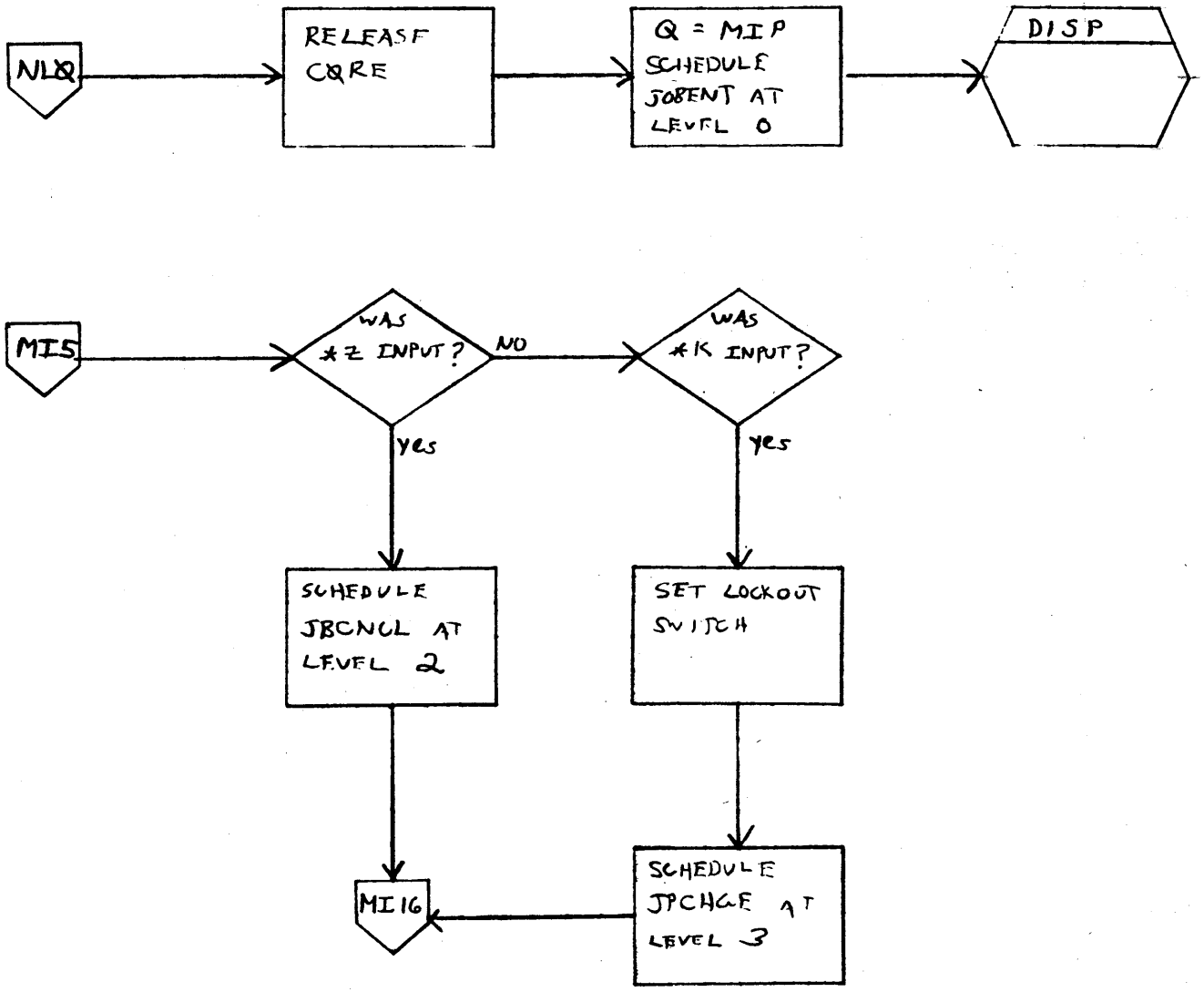
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MINT	PROJECT MGR.			
	PAGE 1 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE MINT		PROJECT MGR.		
		PAGE 2 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

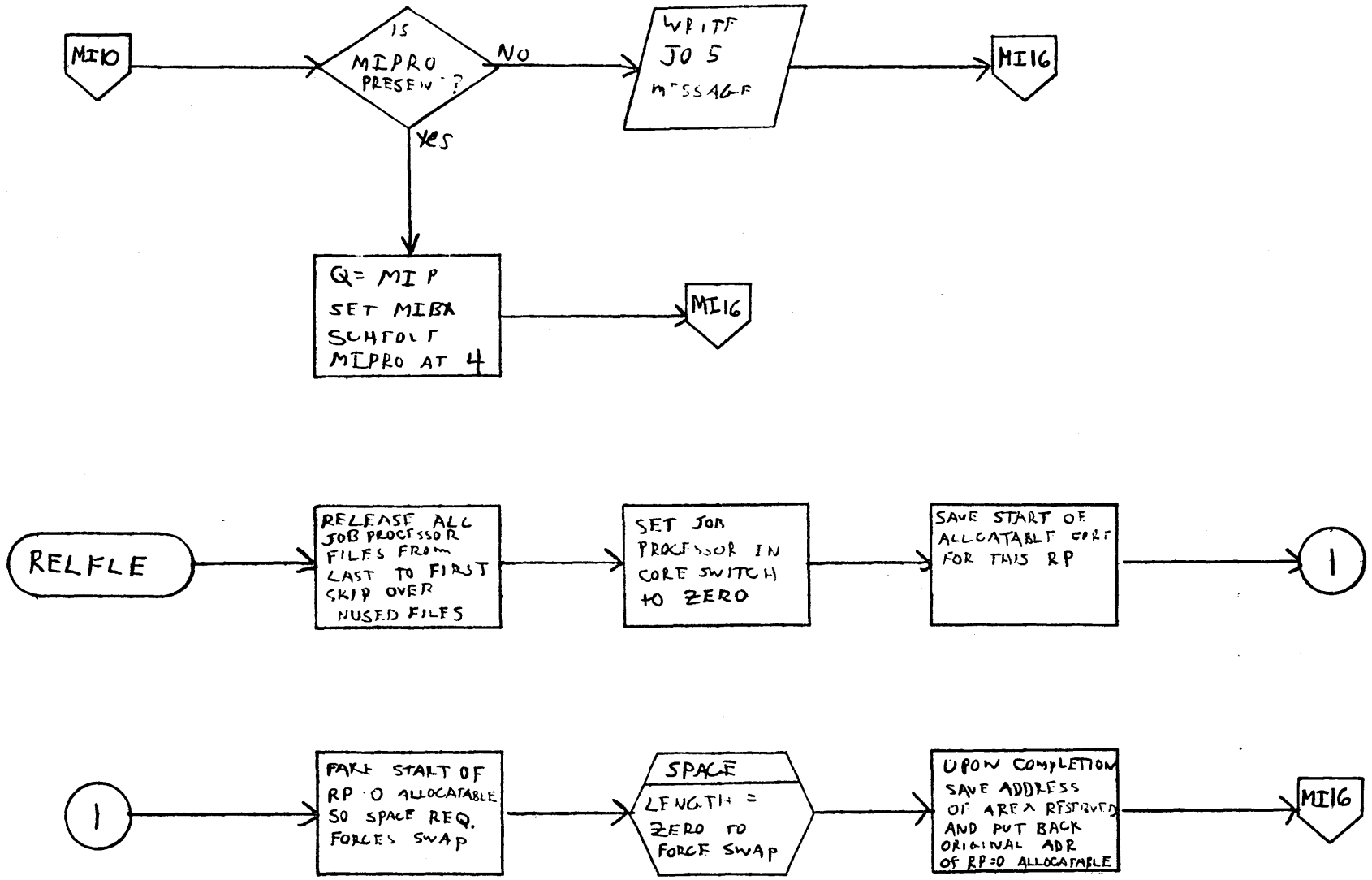
21.6



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: MINT		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

21.7



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE MINT	PROJECT MGR.			
FLOWCHART ☐	PAGE 4 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

21.8

A1385 (FC ... CA127-1

PRINTED IN A.

22.0 Manual Interrupt Processor {MIPR0}

22.1 Function

Mipro processes all non-job processor statements.

22.2 Entry Interfaces

Mipro is scheduled at level 3 by MINT

Q= address of the ASCII input buffer

22.3 Exit Interfaces

Mipro schedules the appropriate processor for the control statement entered.

22.4 Control Statements

$$	Diagnostic timer is scheduled {start timer}
=S	Schedule system directory program
SELF	Dump Engineering File
NOTIME	Disable Timer
SYSCOP	Start System Checkout
DB	Start On-Line Debug Program
DX	Stop On-Line Debug Program

22.5 Tables

22.5.1 Vector Table

Table of one word addresses to statement processors.

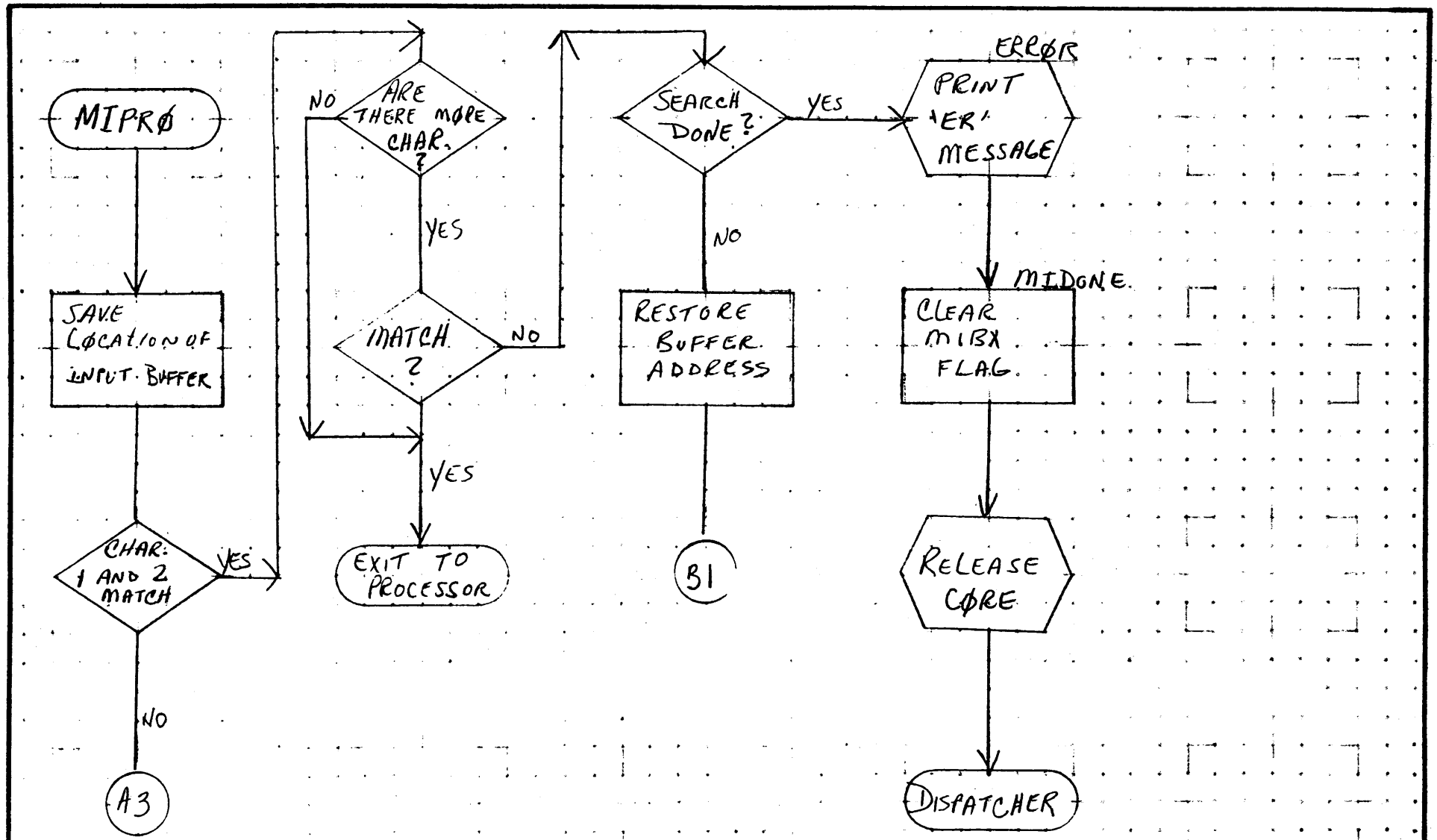
22.5.2 Function Code Tables

COD1 - Control statement, characters 1 and 2

COD2 - Control statement, characters 3 and 4

22.5.3 Index Table

Table of addresses for scheduler calls



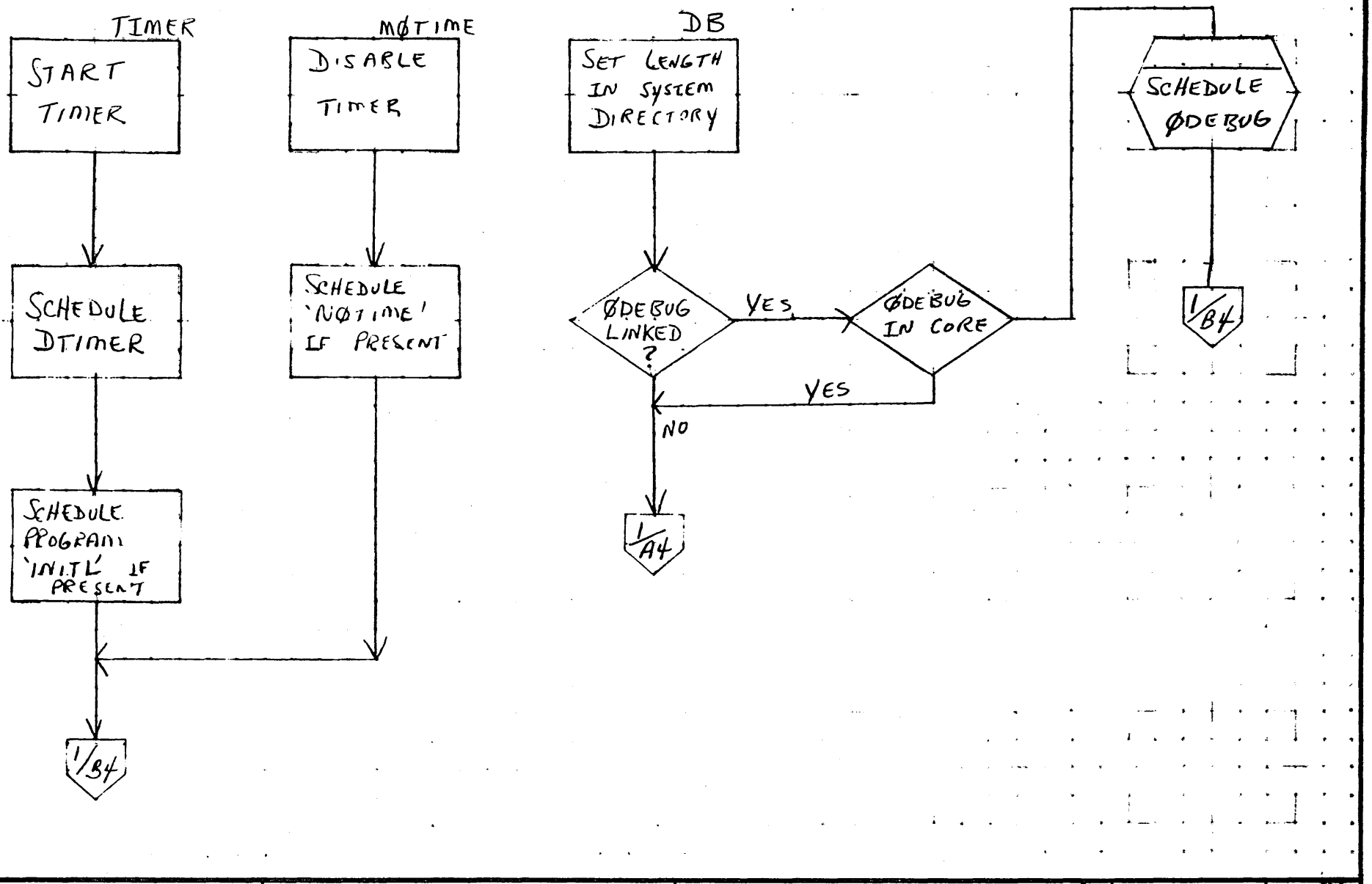
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE MIPRO		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 10/27	TASK NAME		

AA1385 (FORME CA127-1)

PRINTED IN U.S.

MAR 5 1971

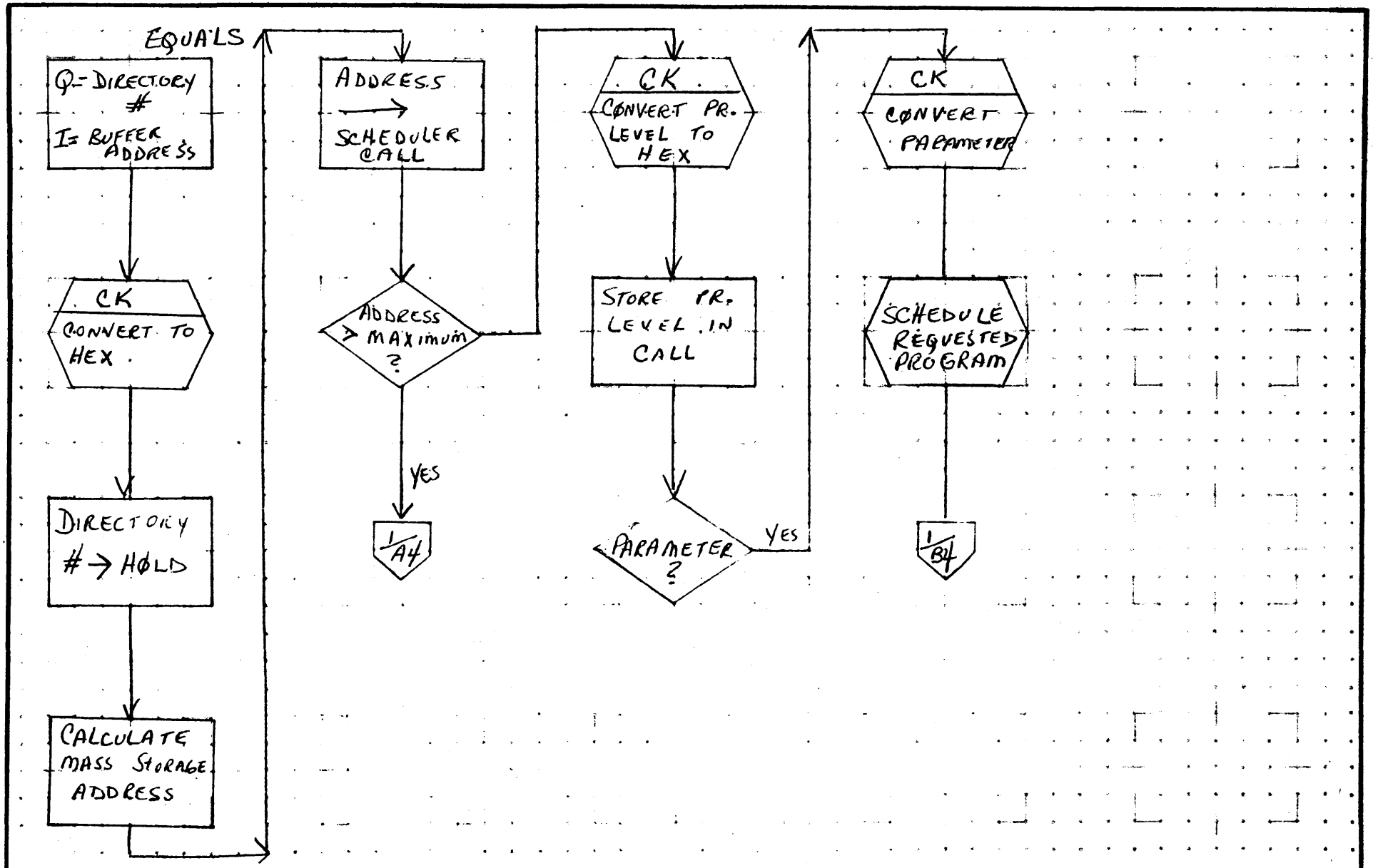
22.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IM	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MIPRO		PROJECT MGR.			
		PAGE 2 OF 9	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 10/27	TASK NAME			

MAR 5 1971

22.4



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE MIPRO		PROJECT MGR.		
	MSOS 3.0	PAGE 3 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

23.0 JOBENT JOB PROCESSOR CONTROL MODULE

23.1 FUNCTION

The JOBENT module has three primary functions.

a. Calculates the absolute addresses of the Loader, Core, Exit and Status request processors and stores them in the RCTV table NMONI.

b. Holds the input and transfer table buffers which are accessed by the other Job Processor routines.

c. Releases and schedules the correct Job Processor, LIBEDT or System Recovery module as determined by the parameters passed to it.

23.2 ENTRY POINT NAMES

JBENT
JBPRO
MIPBUF

23.3 EXTERNALS AND DESCRIPTION

PCOMFL	LIBEDT protected common flag {TRVEC}
JBCNFG	Job cancel flag {TRVEC}
MIB	Manual interrupt lockout flag {MINT}
FILE1	Contains absolute address of JOBENT when in core {TRVEC}
FILE2	Contains absolute address of JP or LIBEDT modules when in core {TRVEC}
JOBIND	Job Processor in core flag {TRVEC}
SWTCH	Switch to lock out the Job Processor when LIBEDT or System Recovery is in operation {TRVEC}
LIBEDT	Library Editing Module
RCOVER	System Recovery Module
UNPTIM	Number of unprotected timer requests outstanding {TRVEC}
RELFLE	Routine to release all files {MINT}

JPSWT	Temp. loc. for MIINP buffer address or an index to the tranta table or a negative value set by J0BENT or JBKILL {TRVEC}
L0CF	Address of error routine in PR0TEC {TRVEC}
J0BPR0	Job Processor Module
JPL0AD	Job Processor Loader Module
JPCHGE	* K Processor Module
JPT13	Getfile Request Processor Module
RCTV	Request Code Transfer Vector Table. {NM0NI}
RESTOR	Restore Logical Unit Module
TRANV	Contains absolute address of J0BPR0 Tranta Table {TRVEC}
JKIN	JBKILL in core flag {TRVEC}
T3	Status Request Processor Entry Point {T3}
T5	Exit Request Processor Entry Point {T5}
T7	Loader Request Processor Entry Point {T7}
T11	Core Request Processor Entry Point {T11}

23.4 ENTRY INTERFACES

The initial entry to J0BENT is made at JBENT when the Job Processor is called with a manual interrupt. After processing has begun and prior to FILE1 being released, entry to J0BENT may be made at any of four routines.

23.4.1 JBPR0 - A routine that schedules the requested Job Processor module as determined by the parameters passed in the A and Q registers.

23.4.2 LIB - This routine is entered by a jump from JPL0AD when it has been determined that a request for °LIBEDT° has been made.

23.4.3 REC0VR - This routine is entered from J0BPR0 if Q register is zero and from JBKILL if negative. The sign of Q upon entry indicates whether the Job Processor is to be scheduled upon return from System Recovery.

23.4.4 RELF - This routine releases an area of allocatable core depending upon the contents of the Q register on entry.

23.5 EXTERNAL INTERFACES

23.5.1 JBENT

Exit is made via a schedule of J0BPR0. If Q register is negative upon entry to J0BPR0 print a "J" and input next control statement.

23.5.2 JPPR0

Exit is made via a schedule of the requested Job Processor module. The Q register will contain an index to a routine within the scheduled module or an execution address.

23.5.3 LIB

LIB schedules LIBEDT with the return address to LB2 routine in J0BENT in Q. Upon return from LIBEDT, J0BPR0 is scheduled with Q set negative to indicate that the next control statement has not been read.

23.5.4 RECOVR

RECOVR schedules the System Recovery module {RCOVER} after having stored its return address in the absolute location $EE. Upon return from RCOVER a check is made of the value of Q when RECOVR was first entered. If Q was initially negative the Job Processor is scheduled with Q negative. If Q was positive a jump is made to RELFLE {routine in MINT to release Job Processor from core}.

23.5.5 RELF

RELF exits to the Dispatcher after releasing the file specified.

23.6 GENERAL PROGRAM INFORMATION

23.6.1 ENTTBL

ENTTBL is a table of relative entry point addresses in J0BENT. The addresses are added to the FWA%J0BENT and transferred to TRVEC.

23.6.2 TBL

TBL is a table of Job Processor module addresses. These addresses are picked up by the JBPR0 routine to schedule the requested Job Processor module.

23.7 GENERAL DESIGN PECULIARITIES

23.7.1 BUFF1

This area of the J0BENT program as indicated by the asterisks will be overlaid by the contents of the MIINP buffer in MINT or the SM1 buffer in J0BPR0. MIPBUF is equated to BUFF1 for addressing purposes.

23.7.2 BUFF2

This area of the J0BENT program as indicated by the asterisks will be overlaid by the contents of the TRANTA table in J0BPR0. The three locations, BPS, RI and LOADEP, within the BUFF2 area, must initially be set to zero, prior to scheduling J0BPR0. TRNTBL is equated to BUFF2 for addressing purposes.

23.7.3 ERRM and SM

The ALF statements following ERRM and SM must immediately precede the BUFF1 area. These two statements in conjunction with the BUFF1 buffer are used to print out J03 and J04 error messages.

23.7.4 ENTTBL

Any entries made to this entry point table must also be made in TRVEC in the same relative position. This restriction is made because the table is transferred to TRVEC with the table transfer rountine {MVTBL}.

23.7.5 TBL

Any entries made to this table must be made following the RESTOR entry due to the indexing scheme in the JBPR0 routine.

23.8 PROGRAM LOGIC

23.8.1 General flow of J0BENT routine

J0BENT is entered initially from MIPR0 with the Q register containing the address of the manual interrupt buffer (MIINP). This address is saved and the L00P routine is entered. This routine transfers the contents of ENTTBL table to TRVEC. The absolute addresses of the Core (T11), Loaded (T7), Exit (T5) and Status (T3) Request Processors are calculated and stored in the RCTV table in NM0NI. The three locations BPS, RI and L0ADEP are then cleared in what will be the transfer table buffer (TRNTBL). The transfer of the MIINP buffer in MINX to MIPBUF (BUFF1) is then accomplished. Upon completion of the transfer, the Load and Go sector number is set to one and J0BPR0 is scheduled with Q set negative which indicates a "J" is to be printed out and the next control statement accepted.

The JBPR0 routine schedules a job processor module as determined by the contents of the A register. The module is then scheduled with the Q register either set negative to print out a "J", set to an index within the scheduled module or set to an execution address.

The LIB routine is entered from JPL0AD to schedule LIBEDT.

The LB2 routine is entered upon return from LIBEDT and exits to schedule J0BPR0.

The REC0VR routine is entered from either J0BPR0 or JBKILL to schedule the System Recovery routine (RC0VER).

The RC2 routine is entered upon return from the System Recovery routine (RC0VER). Exit is to either the RELFLE routine in MINT or to schedule J0BPR0.

REL is a routine entered only from within J0BENT to release FILES 2 and 3 if active. Exit is to the calling routine.

RELF is entered from outside the J0BENT module by routines wishing to release a file they specify by the contents of the Q register.

MRELF is the master release routine that releases the file specified. The routine is entered from the JBPR0 and RELF routines and exits to the sender.

23.9 SUBROUTINE LOGIC

23.9.1 JBPR0

Upon entry to this routine the Q register contains either a negative value, an index to a routine within the module to be scheduled or an execution address. This value is saved in SAVQ1 and the contents of the A register is transferred to the Q register to be used as index to schedule the requested Job Processor module. These index values are as follows:

0 = JPT13, 1 = J0BPR0, 2 = JPL0AD,
3 = JPCHGE and 4 = RESTOR.

The requested address is picked up and stored in the S. edule Request. The location F1 that contains the address of FILE1, which contains the FWA of AREA1 is picked up and a return jump is made to MRELF, the master release routine. Upon return pick up in the Q register, the contents of SAVQ1 and schedule the proper Job Processor module.

23.9.2 LIB

This routine is entered by a jump from JPL0AD when it is determined that LIBEDT is requested. Upon entry a return jump is made to the routine REL to release any outstanding files. The address of LB2 is then calculated and LIBEDT is scheduled with absolute address of LB2 in the Q register. On return from LIBEDT the Load and Go sector is set to one and the MIB flag in MINT is set to lock out other entries. The Job Processor lockout switch (SWTCH) is cleared, the Q register and the Job Processor Switch (JPSWT) are set negative and a jump is made to SJBPR0 to schedule J0BPR0.

23.9.3 RECOVR

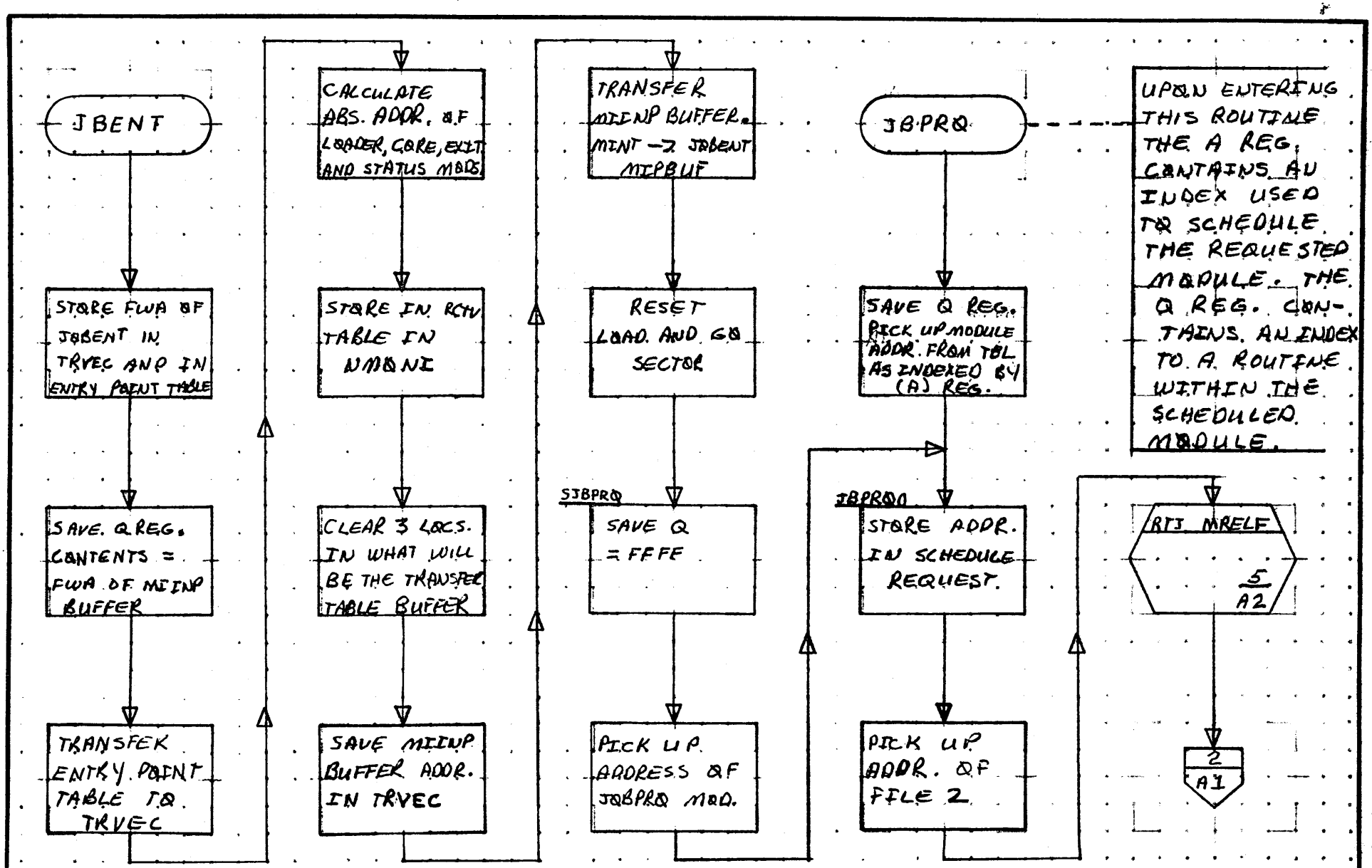
This routine is entered from the SGNOFF {*Z} routine in JOBPRO or from SG7 routine in JBKILL if the Recovery Indicator {RI} switch was set. The Q register contents, 0 if JOBPRO - negative if JBKILL, are saved in SAVQ to be used on the return from RCOVER. The absolute address of the RC2 routine is calculated and saved in location $EA to be used as a return from RCOVER. The System Recovery module is then scheduled. Upon return from System Recovery the contents of SAVQ is checked. If negative {from JBKILL} a jump is made back to LB4 to schedule JOBPRO. If positive {from JOBPRO} inhibit interrupts, clear the protected common flag {PCOMFL}, clear Job Processor lockout switch {SWTCH} and return jump to RELFLE in MINT to release all files and wait for input to MINT.

23.9.4 REL

This routine is entered only from within JOBENT. Its function is to release FILE2 and FILE3 if active and if the job cancel flag {JBCNFG} is not set.

23.9.5 RELF

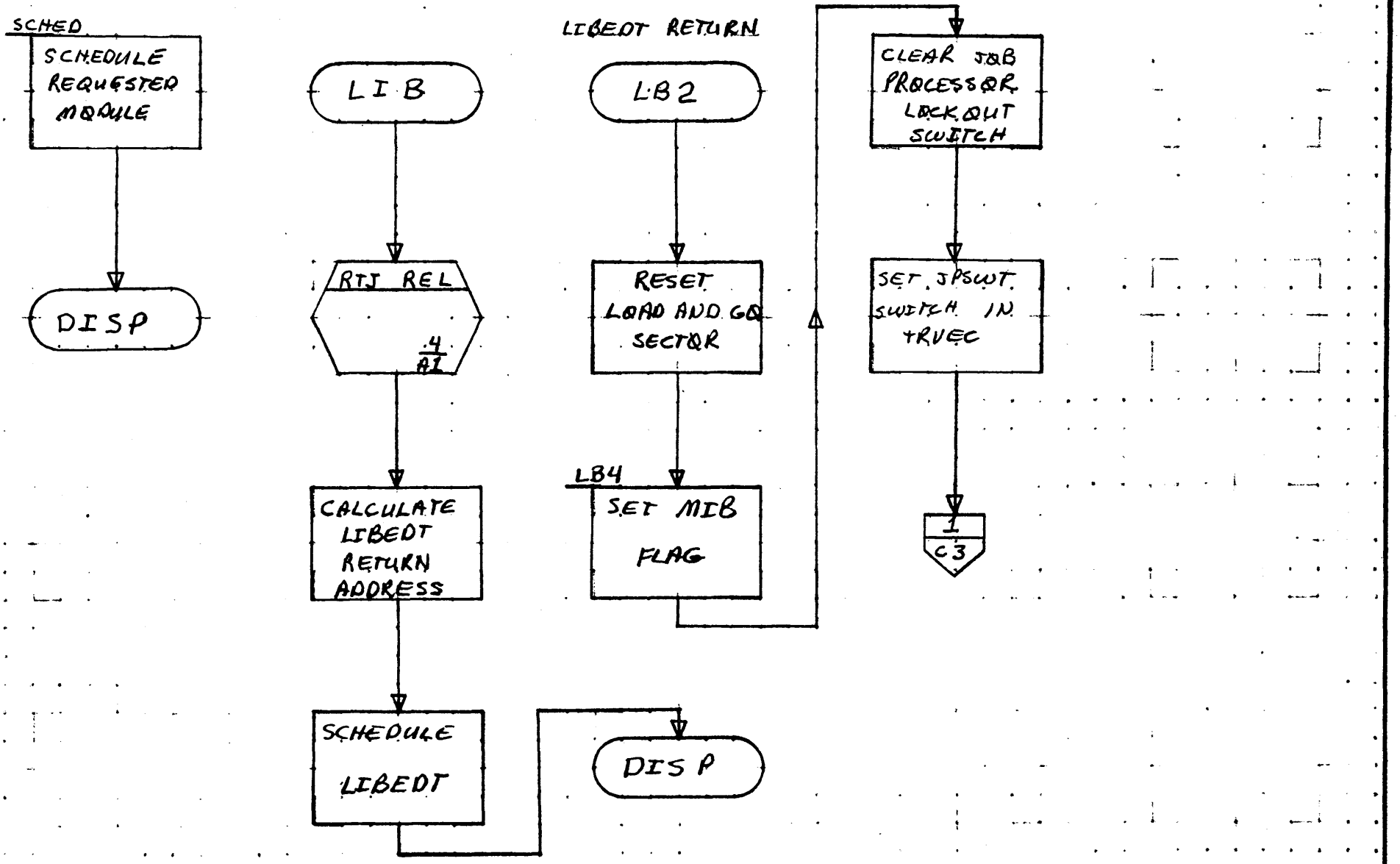
This routine return jumps to MRELF to release the FILE specified in the Q register.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE JOBENT		PROJECT MGR.		
FLOWCHART ☐		PAGE 1 OF 5	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

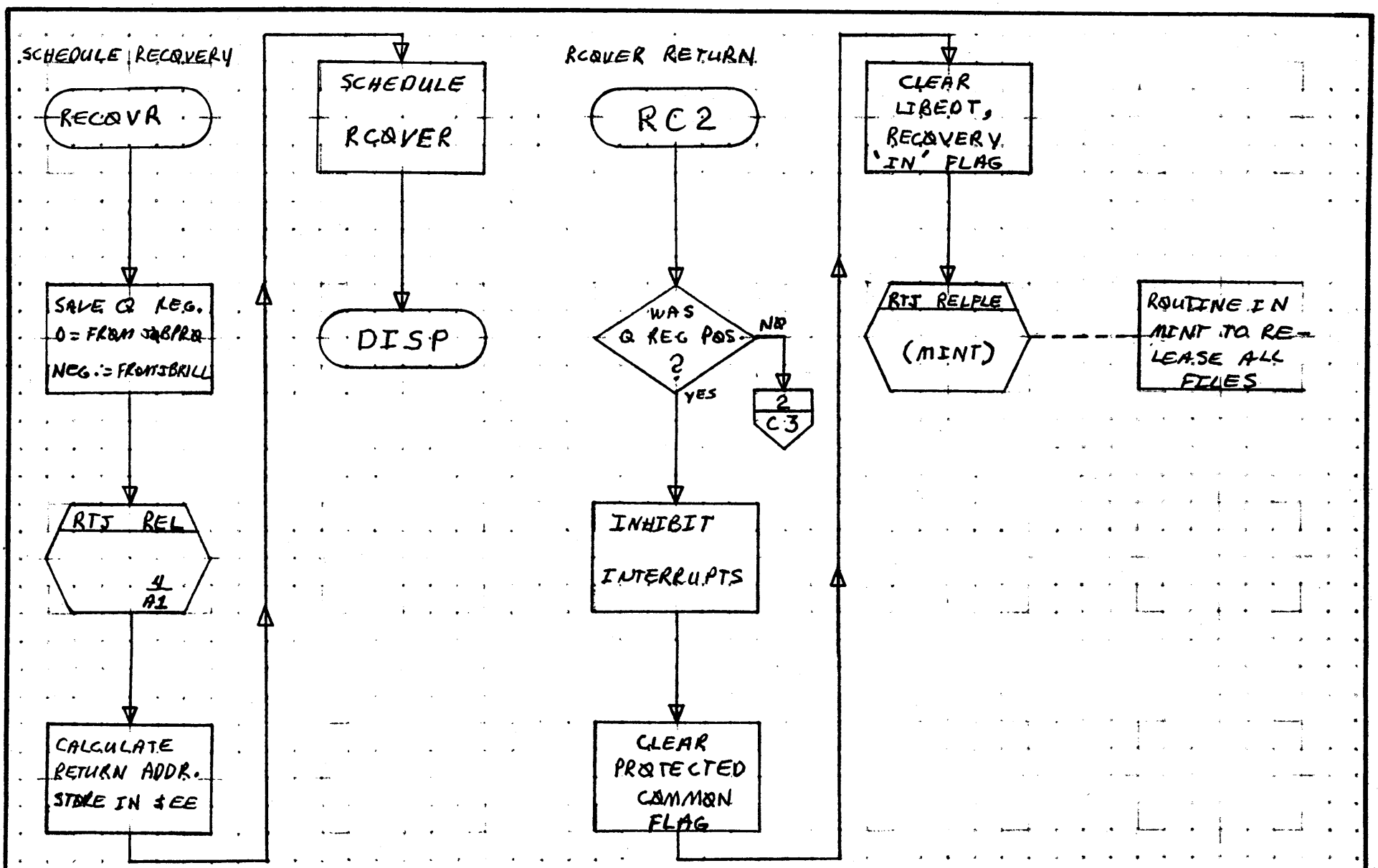
23.8



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JOBENT		PROJECT MGR.		
		PAGE 2 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

23.9

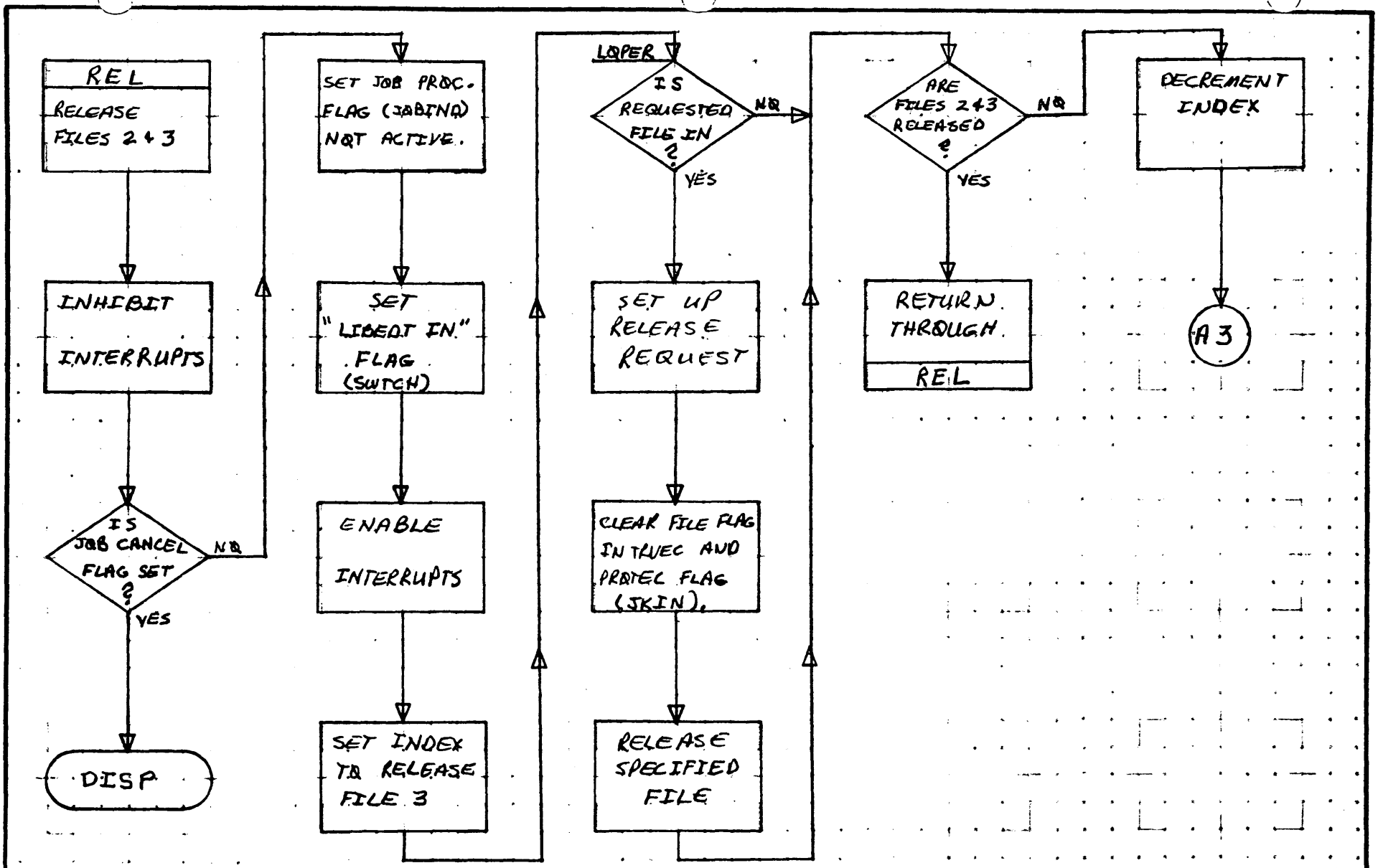


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE JOBENT		PROJECT MGR.			
		PAGE 3 OF 5	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORM CA127-1)

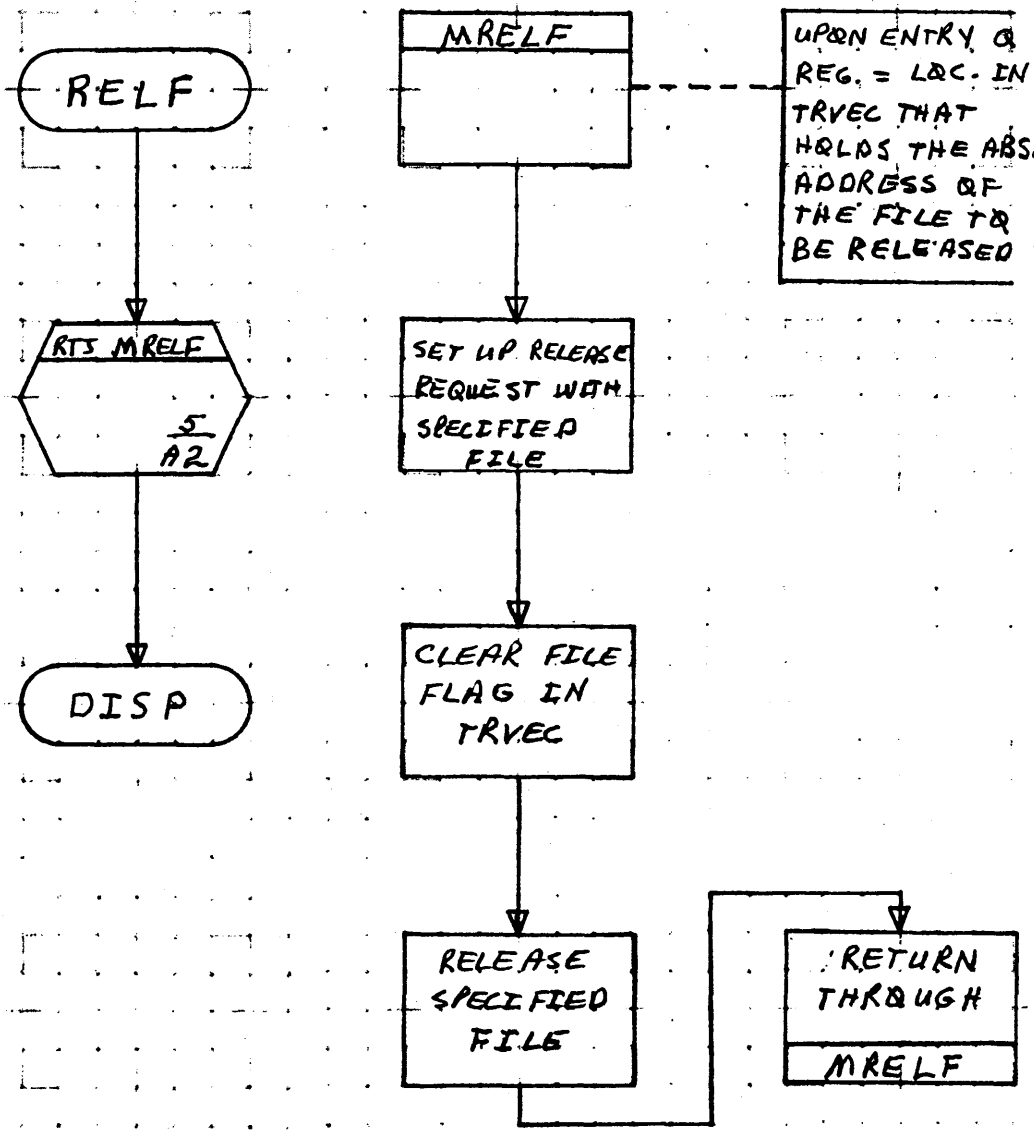
PRINTED IN U.S.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE JOBENT		PROJECT MGR.		
FLOWCHART ☐		PAGE 4 OF 5	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

23.11



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JOBENT		PROJECT MGR.			
FLOWCHART ☐		PAGE 5 OF 5	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

24.0 T11 - Core Request Processor

24.1 FUNCTION

This module is used to set or determine the bounds of available core. The A and Q registers are used to pass parameters when a core request is made. If A and Q are zero, the current bounds are passed to the requester. If A and Q are both addresses in unprotected core the current bounds of available core are set to these values. Available unprotected core is that portion of unprotected core which is available for loading of a subroutine or overlay.

24.2 ENTRY POINT NAMES

T11

24.3 EXTERNALS and DESCRIPTION

JBCNFG Job cancel flag. Set when JBKILL is in progress.

REQERR Error routine in T7

LOADIN Flag in TRVEC to indicate LOADER is in core.

24.4 ENTRY INTERFACES

T11 is entered from NMONI with the first word of the parameter list in A and the address of volatile in I. Volatile contains the requester's Q, A and I, the return location in the request program (word 3), and the pointer to the request parameter list (word %0.

24.5 EXTERNAL INTERFACES

Two possible exits may be made from T11.

1. Normal exit to REQXT in NMONI by an indirect jump through location $B9.
2. Error exit to REQERR (in T7) made if A and Q are not both zero or if either of the addresses in A or Q is not in unprotected core or if A is not greater than Q.

24.6 GENERAL PROGRAM INFORMATION

None

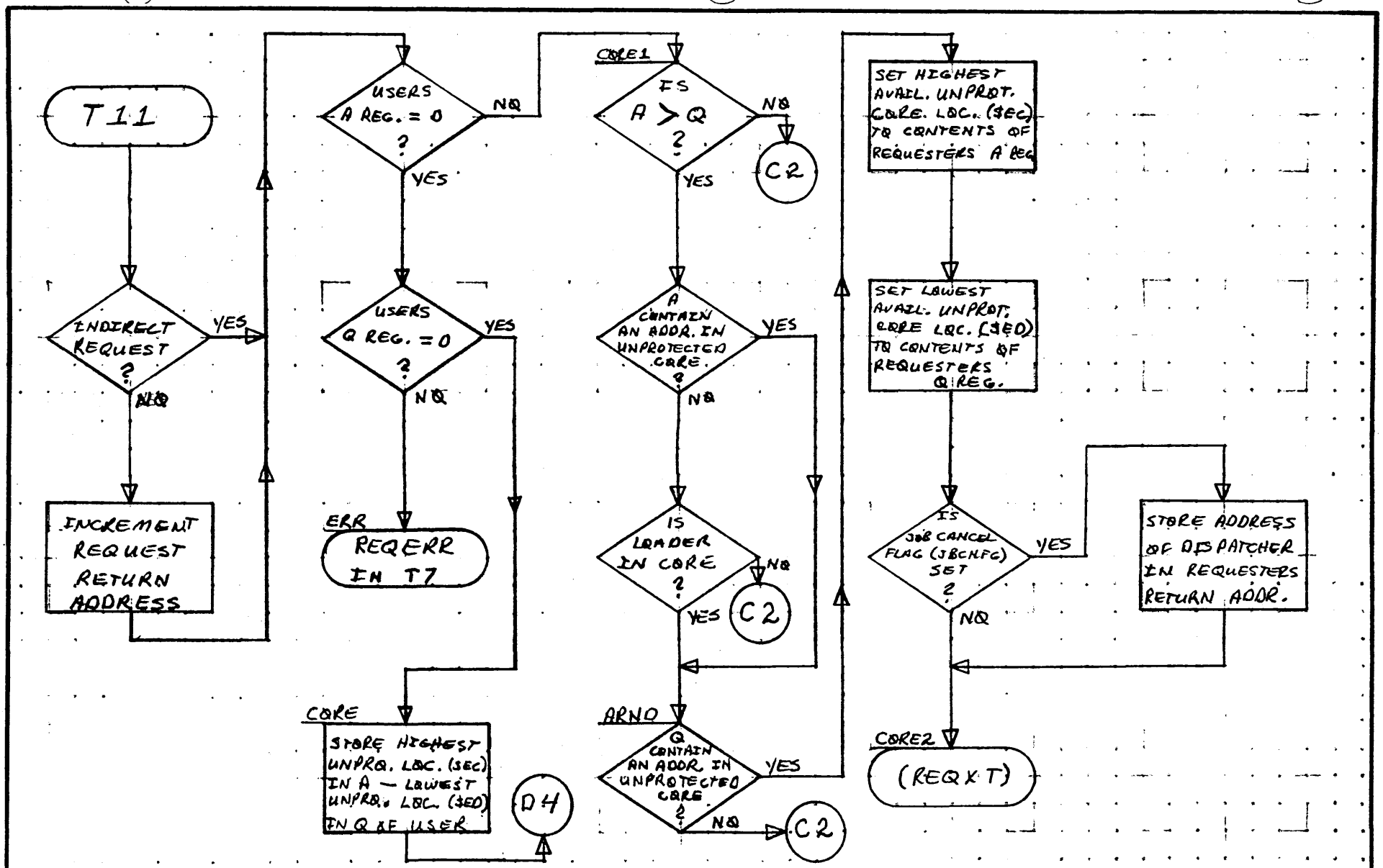
24.7 GENERAL DESIGN PECULIARITIES

None

24.8 PROGRAM LOGIC

On entry A and Q of the requesting program are checked to see if they are zero. If they are zero the temporary highest {contents of location $EC} and temporary lowest {contents of location $ED} bounds of available unprotected core are put in the requestor's A and Q respectively.

If the requestor's A and Q were initially locations in unprotected core {within the bounds contained in locations $F6 and $F7}, then these values are stored in locations $EC and $ED respectively to define new bounds of available unprotected core.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T11 CORE REQUEST		PROJECT MGR.		
	PROCESSOR	PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

2h.3

25.0 T7 LOADER REQUEST PROCESSOR

25.1 FUNCTION

This routine is used to bring the Relocating Loader into the highest available unprotected core. The parameters to be supplied to the loader are in A and Q. These are passed to the loader when it is executed.

25.2 ENTRY POINT NAMES

T7
REQERR

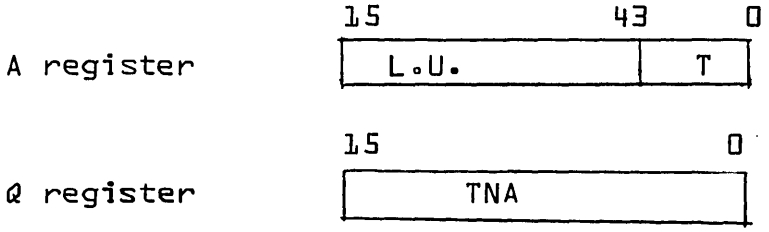
25.3 EXTERNALS and DESCRIPTION

LOADSD	Mass storage System Directory Table.
JPRET1	Location in TRVEC in which the return address to T7 is stored.
JPRETN	Location in TRVEC to which the loader returns upon completion.
JPRET	Locations in preset tables in which the locations of JPRETN is stored.
LOCF	Location in TRVEC which contains the location of F in PROTEC.
LPTRS	Location in TRVEC which contains the location of PTRS in PROTEC.
LOADIN	Location in TRVEC to indicate the loader is in core.
UNPIO	Flag in TRVEC which indicates the number of unprotected I/O requests in process.
SWAPCK	Routine in DRCORE which decrements UNPIO on completion of an unprotected I/O request.
JBCNFG	Job Cancel Flag. Set when JBKILL is in process. {TRVEC}

25.4 ENTRY INTERFACES

T7 is entered from NMONI with the pointer to the first word of the parameter list in A and the address of volatile in I. Volatile contains the requestor's Q, A and I, the return location in the requestor's program {word 3} and the pointer to the request parameter list {word 5}.

The parameters in the requestor's Q and A are as follows:



where:

L.U.	Input unit for loading operation.
TNA	Core location of the entry name of routine to be loaded.
T	One of the following types of loading operation
0	Relocatable binary load
1	Subroutine load
2	Program load
3	Memory map
4	Entry point look up
5	Subroutine load, no memory map
6	Patch to core resident programs {CREP}
7	Set data base

{A more detailed explanation of types of loading operations can be found in the discussion of the loader}.

Entry may also be made from T11 or T3 to REQERR.

25.5 EXTERNAL INTERFACES

Normal exit is made via a jump to the requestor's return address. If the Job Cancel Flag (JBCNFG) is set, exit is to the Dispatcher.

Error exit is made from RE@ERR to entry point F in PROTEC to print out a J02 error. The requestor's parameter list pointer or the return location is stuffed in PTRS in PROTEC to be encoded for the error message.

25.6 GENERAL PROGRAM INFORMATION

LIC Local loader in core flag

ONEBIT An equate to absolute location $23 =0001

25.7 GENERAL DESIGN PECULIARITIES

None

25.8 PROGRAM LOGIC

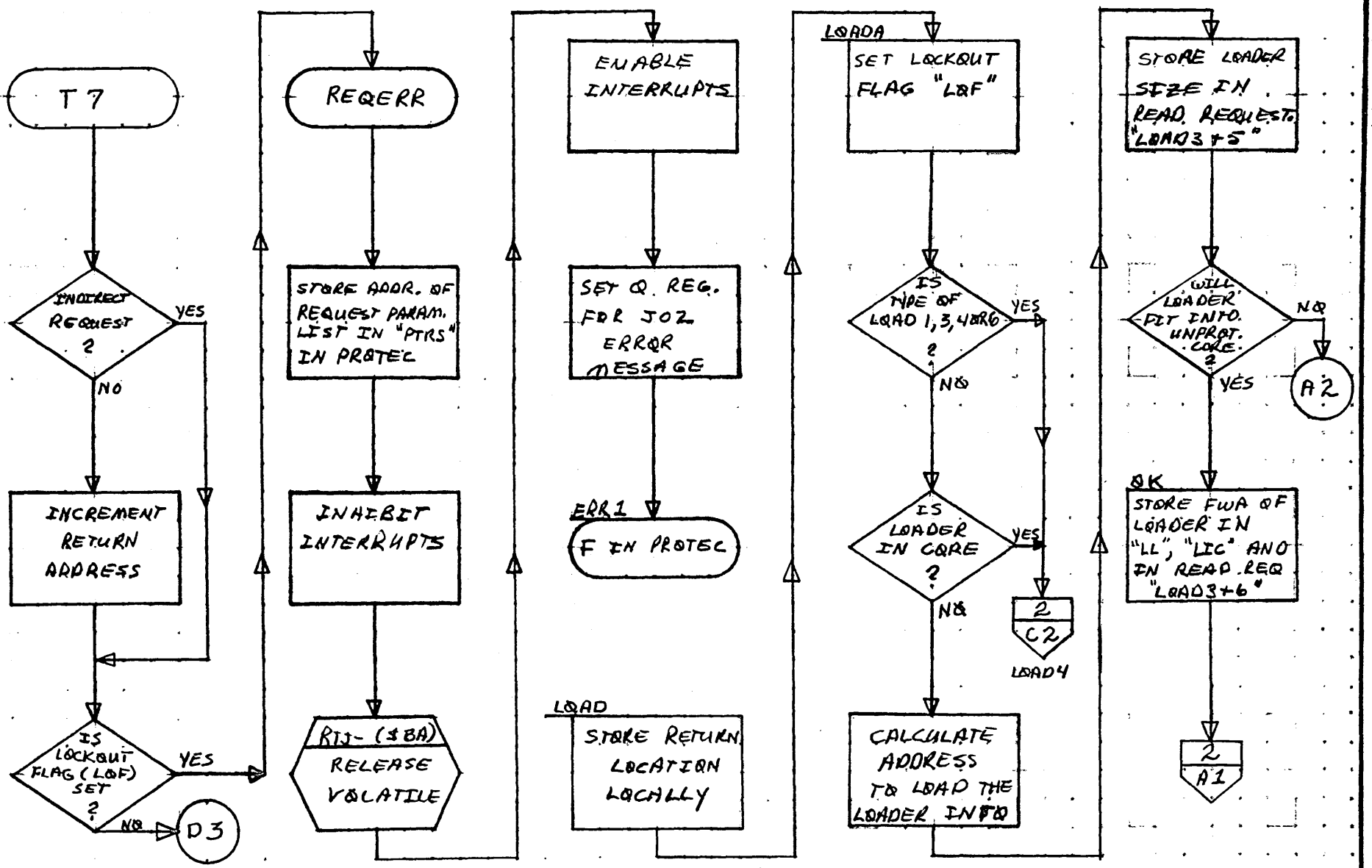
T7 uses a lockout flag LOF to prevent the execution of more than one loader request. This flag is checked on entry and if set a request error (J02) is issued. If no errors the lockout flag (LOF) is set.

A check is made if the loader is in core. If the load type is 1,3,4 or 6, or if the flag LIC is set it is assumed the loader is in core.

If the loader is determined not to be in core, it is read into highest available unprotected core (not to exceed the location contained in $EC or $F6 whichever is smaller). The flag UNPIO in TRVEC is set during this process to prevent a core swap during the read.

A jump is made to the loader with the return address patched to JPRETN in TRVEC, with the flag LOADIN set in TRVEC to allow the loader to write below the scratch area on disk and with volatile released.

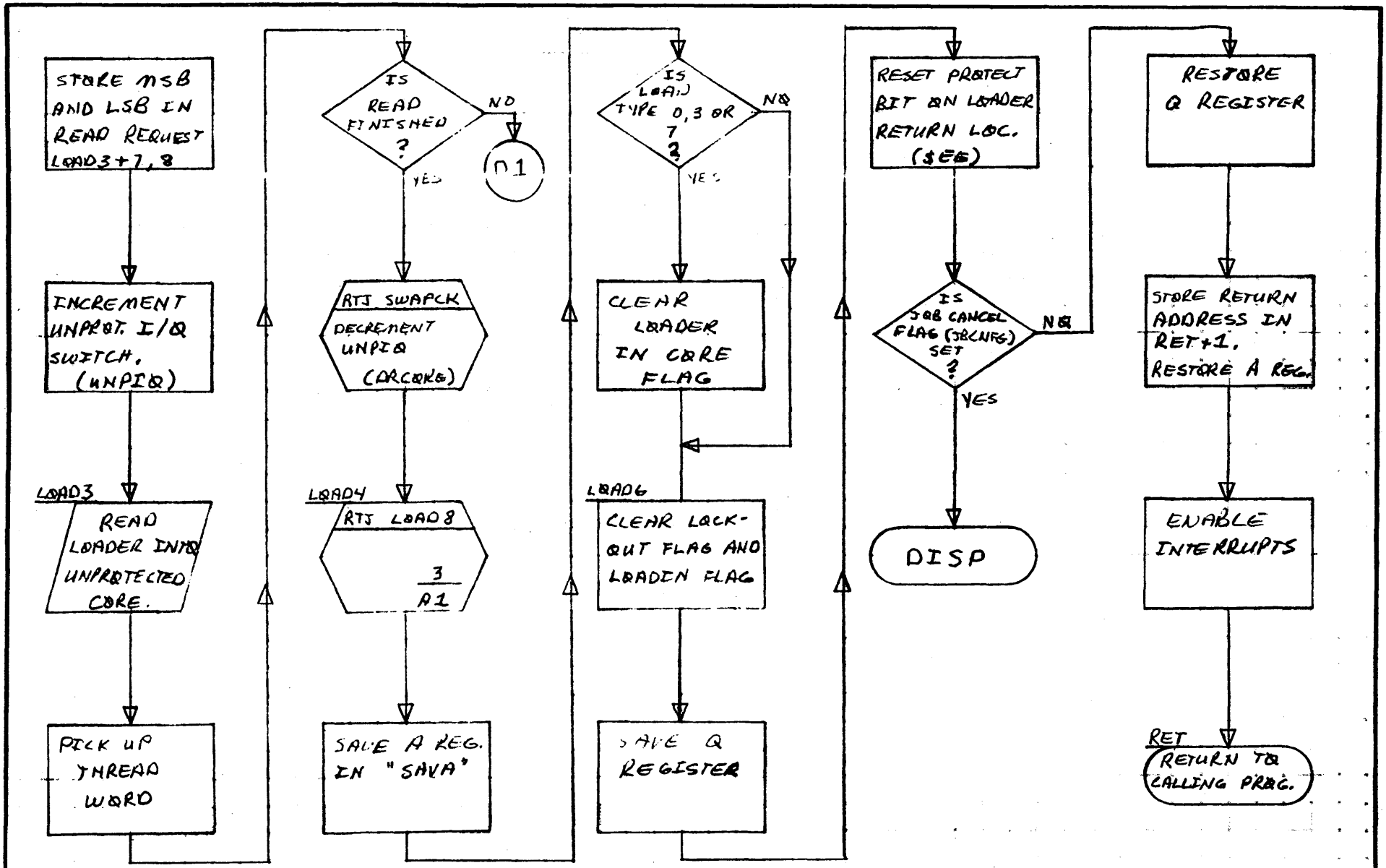
Upon return from the loader, if the load type is 0,3 or 7 the flag LIC is cleared. The lockout flags LOF and LOADIN are cleared.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T7 LOADER REQUEST		PROJECT MGR.		
	PROCESSOR	PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

25.4



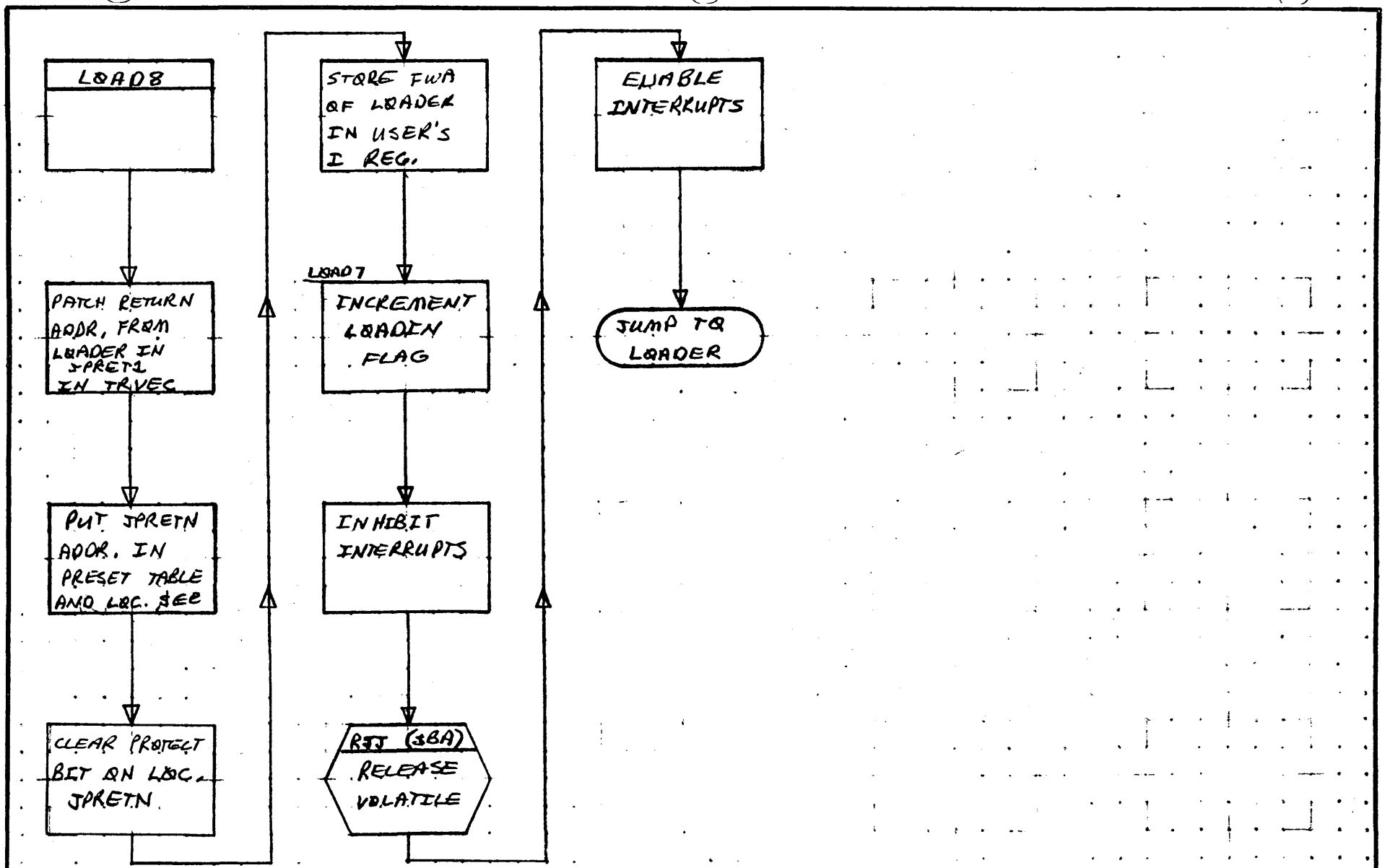
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 7600	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE T7		PROJECT MGR.		
		PAGE 2 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORM CA127-1)

PRINTED IN U

MAR 5 1971

25.5



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T7		PROJECT MGR.		
		PAGE 3 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

25.0

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

26.0 Exit Request Processor (T5)

26.1 Function

This request is used by unprotected programs to signal completion of a job or a user's completion routine. That is, when a computation is completed, this request notifies the Operating System of that fact.

26.2 Entry Point Names

T5 Exit request entry

26.3 Externals and Description

UNPTIM	Number of unprotected timer requests waiting
UNPIO	Number of unprotected requests waiting to be completed
INTSTK	Location of Interrupt stack
JBCNCL	Job cancel routine in TRVEC

26.4 Entry Interfaces

+Q = entry from the Monitor

-Q = entry from the Protect Processor

If the entry was from the Monitor A = location of the request parameter.

26.5 External Interfaces

None

26.6 Tables

None

26.7 General Design Pecularities

None

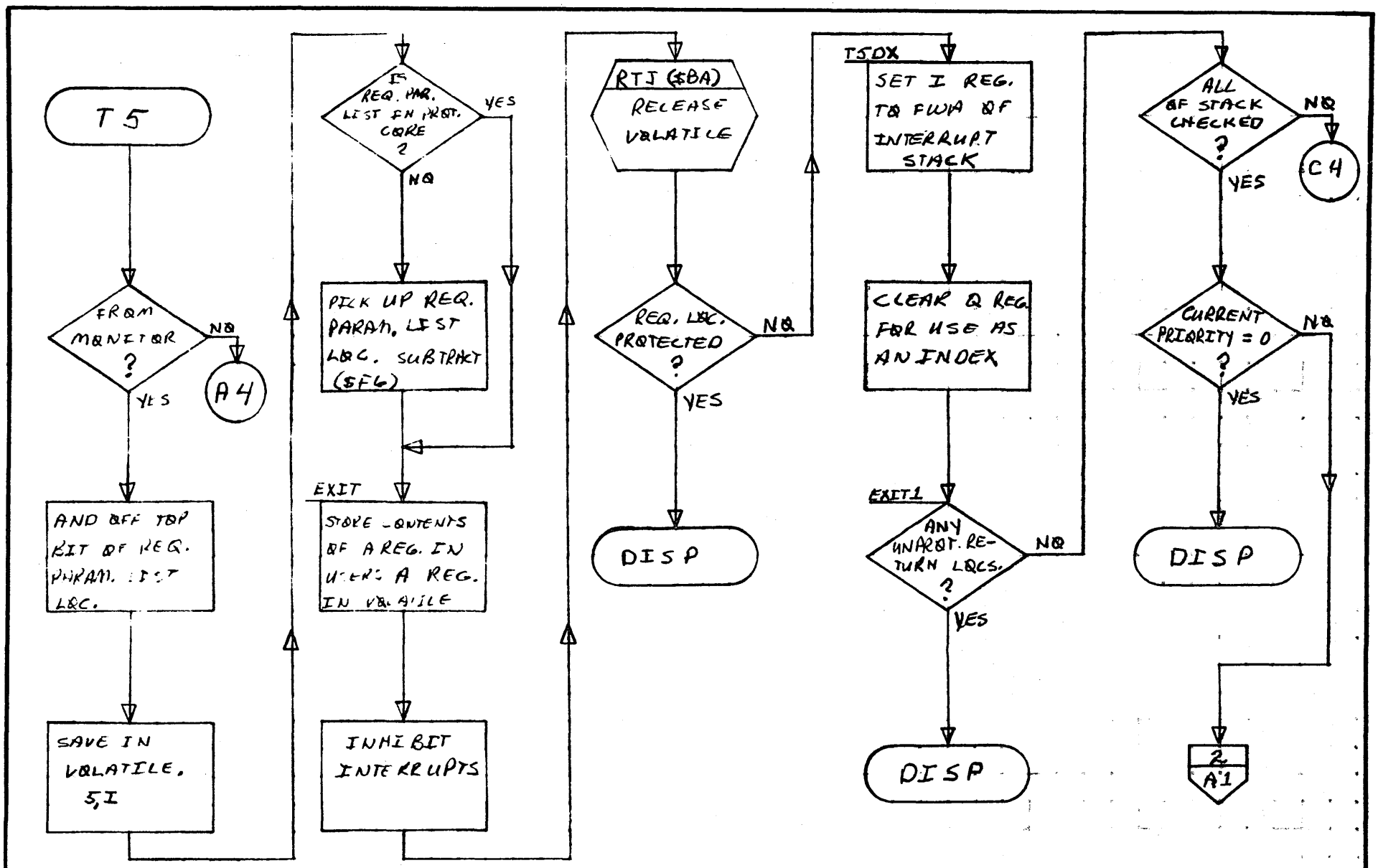
26.8 Program Logic

Upon entrance, the Q register is checked to determine whether entry was from the Protect Processor or the monitor. If entry is from the monitor, volatile storage is released, and the request location is checked. If the request is from protected core, T5 exits to the dispatcher. Otherwise, all stack entries are checked for unprotected core return location. If any exist, control is returned to the dispatcher.

If none exist, °UNTIM° is checked for being non-zero. If so, °TIMER in effect°, the flag is checked until it goes to zero. Then °UNPIO° flag is checked for non-zero. If so, {unprotected request in operation} this waits until flag is zero and then schedules JBCNCL and exits to the dispatcher.

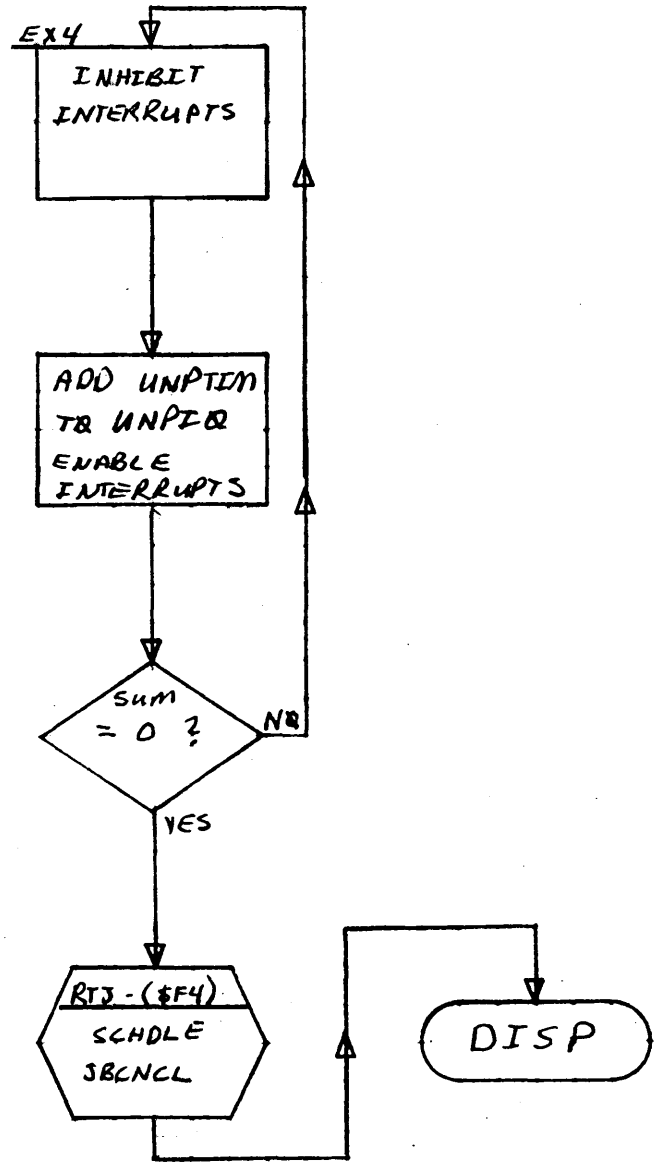
26.9 Subroutine Logic

None



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS I.N.S.	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T5 EXIT REQUEST		PROJECT MGR.		
	PROCESSOR	PAGE 1 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE T5		PROJECT MGR			
		PAGE 2 OF 2	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO			
	DRAWN BY	DATE	TASK NAME			

27.0 T3 Status Request Processor

27.1 FUNCTION

The function this routine performs is to get the request status, equipment status and location of the parameter list of the most recent request processed on a given logical unit and return them in the requestor's Q, A and I respectively.

27.2 ENTRY POINT NAMES

T3 Status Request Entry Point

27.3 EXTERNALS and DESCRIPTION

LOGLA	Table in SYSBUF of PHYSTB addresses arranged by logical unit number.
REQERR	Error routine in T7
JBCNFG	Job Cancel flag in TRVEC. Set if JBKILL is active.

27.4 ENTRY INTERFACES

T3 is entered from NMONI with the first word of the parameter list in A and location of volatile in I.

Volatile contains the requestor's Q, A and I registers as well as the return location {word 3} and the location of the Status request parameter list {word 5}.

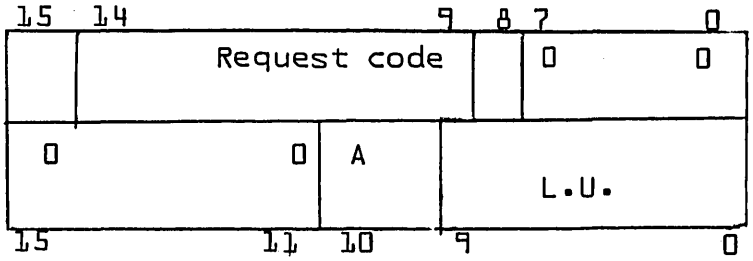
27.5 EXTERNAL INTERFACES

Two possible exits may be made from T3:

1. Normal exit to REQXT in MONI by an indirect jump through location $B9.
2. Error exit to REQERR {in T7} made if the logical unit is not legal for the particular system.

27.6 GENERAL PROGRAM INFORMATION

The request parameter list is:



where:

L.U.	the logical unit
A	relative/indirect indicator for L.U.

27.7 GENERAL DESIGN PECULIARITIES

None

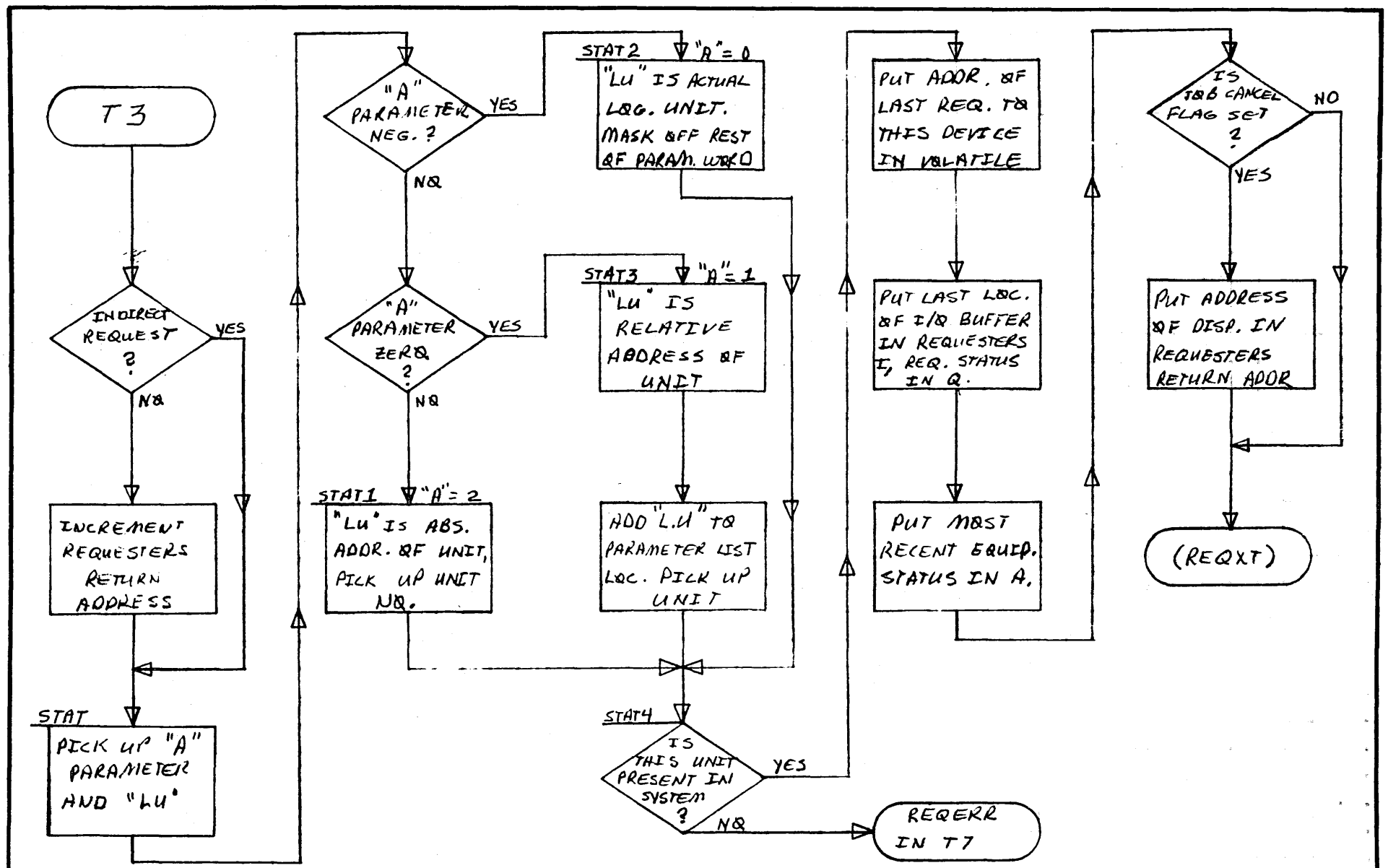
27.8 PROGRAM LOGIC

First the logical unit is found using the A parameter to indicate whether LU is the actual logical unit {A=0}, a signed relative address of the logical unit {A=1} or the actual location of the logical unit {A=2}. Then it is checked for validity.

If the logical unit is valid the table of the physical device is found. From this the location of the current I/O request parameter list is taken {word 6} and stored in word 4 of volatile. The last location of the I/O buffer {word 11} is put in I of the requestor. The request status bits {word 8} are stored in the requestor's Q and the last equipment status read is put in the requestor's A.

27.9 SUBROUTINE LOGIC

None



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE T3 STATUS REQUEST		PROJECT MGR			
FLOWCHART ☐	PROCESSOR	PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

27.3

28.0 JOBPRO

28.1 FUNCTION

The functions of the JOBPRO module are:

1. Read in the Job Processor control statements.
2. Analyze the statements read in.
3. Set up the parameters to schedule the processor needed to handle the input statements.
4. Terminate a Job Processor function.

28.2 ENTRY POINT NAMES

JOBTWO

28.3 EXTERNALS AND DESCRIPTION

TRNVEC - Abs. Addr. of TRNTBL buffer in JOBENT - TRVEC
JBPROE - Abs. Addr. of JBPRO routine in JOBENT - TRVEC
MIBUF - Abs. Addr. of MIPBUF buffer in JOBENT - TRVEC
JPSWT - Job Processor Switch - TRVEC
JBCNFG- Job Cancel Flag - TRVEC
RELFLE- Release Files Routine - MINT
RECOV - Abs. Addr. of RECOVR routine in JOBENT - TRVEC
IP1 - Protect Processor entry point addr. - NIPROC
VRESET- *V Reset Flag - TRVEC
RELS1A- Abs. Addr. of RELF routine in JOBENT - TRVEC
ERRMSG- Abs Addr. of ERRM routine in JOBENT - TRVEC
FILE2 - Holds Job Processor Addr. when in core - TRVEC
NSTACK- Max. No. of stacked requests - TRVEC
TRANV - Abs. Addr. of JOBPRO TRANTA Tab - TRVEC
IUP - Pointer to comment device - TRVEC
MIB - Manual Interrupt lockout flag - MINT
JKIN - JBKILL In Core flag - TRVEC
FILE3 - Holds Protect Proc. Addr. when in core - TRVEC
JOBIND- Job Processor in core flag - TRVEC
RCTV - Request Code Transfer Vector Table - NMONI
LOADIN- Loader flag for Protect Processor - TRVEC
ONE - User subroutine

TWO - User subroutine

THREE - User subroutine

28.4 ENTRY INTERFACES

Entry to J0BPR0 can be made only at the J0BTW0 entry point. Upon entering J0BPR0, the Q register is examined to determine where control will be transferred. If the Q register is set negative control is transferred to the tag "JB". If Q is not negative, control is transferred to one of five pseudo entry points obtained from the TRANTA table.

1. JBPR0 - Return when statement already input.
2. J0BP - Return when no statement.
3. J04 - J04 error entry.
4. J03 - J03 error entry.
5. RF3 - Return to release File3 and output a "J".

28.5 EXTERNAL INTERFACES

The majority of the Exits from J0BPR0 are to J0BENT to schedule the proper Job Processor module as determined by the input statement. Exit is made to RELFLE in MINT when a *Z statement is input and the RI {recovery indicator} flag is not set.

28.6 GENERAL PROGRAM INFORMATION

28.6.1 Transfer Table {TRANTA}

The following table contains relative and absolute addresses and flags which are transferred to J0BENT before exiting from J0BPR0.

TAG	FUNCTION
TRANTA	Abs. loc. of J0BTW0
Rel. to JBPR0	Return when statement already input
INPBUF	Abs. Addr. of input buffer in J0BENT
Rel. to J0BP	Return when no stmt.
Rel. to J04	Error return "J04"
Rel. to J03	Error return "J03"
	NOT USED
BPS	Breakpoint Switch
RI	Recovery On/Off Switch
LOADED	Loader Entry Point
@REG.	Request No.
STCK	Loc. of Protect Processor Request Stack
	NOT USED
NN	No. of entries in Protec. Stack
Rel. to RF3	Return to release FILE3

28.6.2 Table of Job Processor Input Statement Subroutines

The following table is used in indexing to the proper routine as determined by the input statement. A value (@REG) is picked up from the transfer table and is used as an index to the proper routine relative to the index base JPST.

TAB

RELATIVE ADDRESS	STATEMENT
BPLOAD	*B cr statement
TYPEIN	*U cr statement
STDINP	*V cr statement
SGNOFF	*Z cr statement
SETREC	*SR cr statement
RESUME	* cr statement

28.7 GENERAL DESIGN PECULIARITIES

28.7.1 Alteration of the JPTAB in J0BPR0 which is used to set up the @REG index value would cause the wrong routine from the TAB table to be executed.

28.8 PROGRAM LOGIC

This module is entered via a SCHDLE request from J0BENT. Upon entrance the absolute addresses of J0BTW0, JBPR0 and TRANTA are stored in TRVEC. The contents of the BPS, RI and LOADEP locations in the TRNTBL buffer in J0BENT are then transferred to like locations in the TRANTA table.

A check is then made to see if the Q register was negative upon entrance. If so, the next Job Processor statement is requested. If positive, Q contains an index to the relative address in the TRANTA table of the routine to be executed.

Internal to the "JOBPRO" module is a list of acceptable statements. Associated with specified groups in the list is an address of a module to which control is transferred to carry out the operation directed by the statement. This list of standard input statements for the Job Processor includes:

INPUT STATEMENT	NAME OF SUBROUTINE TO PROCESS	MODULE TRANSFERRED TO
*K	CHANGE	JPCHGE MODULE
*B	BPLOAD	
*U	TYPEIN	
*V	STDINP	
*Z	SNGOFF	
*SR	SETREC	
*	RESUME	
*R	RESTOR	RESTOR MODULE
*1	ONE	ONE SUBROUTINE
*2	TWO	TWO SUBROUTINE
*3	THREE	THREE SUBROUTINE

If not one of the above, a loader statement is assumed and control is passed to the "JPLOAD" module, otherwise, the appropriate routine is set up to be scheduled by JOBENT. The following are the input statements that refer to "JPLOAD".

INPUT STATEMENT	NAME OF SUBROUTINE
*ENTRY POINT NAME	TRLOAD
*L	RBLOAD
*X	EXECUT
*P	PREJOB

Statements *1, *2, and *3 are dummy statements which produce a J03, statement error, if requested and subroutine not present. These are provided so that the user may add statements with special functions. These must be loaded with a *YM, - Job Processor Load.

Input statements not recognized by the Job Processor (i.e., does not start with an asterisk), are rejected and a J03, statement error informs the operator. The Job Processor then interrogates the specified device for a new statement. All other statements that are not recognized, but start with an asterisk, are assumed to be "entry point" name statements.

The contents of Q have been saved upon entry and are checked. If Q is negative, entry was made from the Namual Interrupt Processor, the MIB switch is cleared and input is requested. If Q is positive, entry was made from J0BENT and a jump is made to SSI to process the statement.

The 1st character is checked for an asterisk. If not present, a J03 statement message is typed. The 2nd and 3rd characters are checked against a table, and the proper routine is scheduled at the current priority level. Upon returning, J is typed out and another request to read is made. The previous loop is again executed.

All Job Processor functions except the *Z return back to the J0BPR0 module to release FILE3 and wait for input of the next control statement. If a *Z has been input, exit is made to M0NT where all files are released.

28.9 SUBROUTINE L0GIC

28.9.2 MVTBL

This routine moves the TRANTA table to the TRNTBL buffer in J0BENT prior to exiting to J0BENT. Once in J0BENT, the requested module is scheduled.

28.9.2 JBKMIB

This routine checks for the Job Cancel flag (JBCNFG) set and if clear, sets the MIB flag for job lockout.

28.9.3 JBKILL

This routine checks the Job Cancel flag (JBCNFG). If set, exits to J0BENT to release FILE2 and jumps to the DISPATCHER to await entry from J0B KILL. If not set, return is to the caller with interrupts inhibited.

28.9.4 BPLOAD *B (cr)

This routine processes the *B statement. Upon entry the Breakpoint Switch {BPS} is set. Return is through the RF3 pseudo entry point.

28.9.5 TYPEIN *U (cr)

This routine processes the *U statement which causes all control statements to be read from the comment device. Upon entry the constant $18FD is stored in pointer to the comment device {IUPP}. The *V flag {VRESET} is cleared and return is through the RF3 pseudo entry point.

28.9.6 STDINP *V (cr)

This routine processes the *V statement which causes all control statements to be read from the standard input device. Upon entry the constant $18F9 is stored in the pointer to the standard input device {IUPP}. Return is through the RF3 pseudo entry point.

28.9.7 SETREC *SR (cr)

This routine processes the *SR statement which indicates that recovery is desired. Upon entry the Recovery Indicator {RI} flag is set. Return is through the RF3 pseudo entry point.

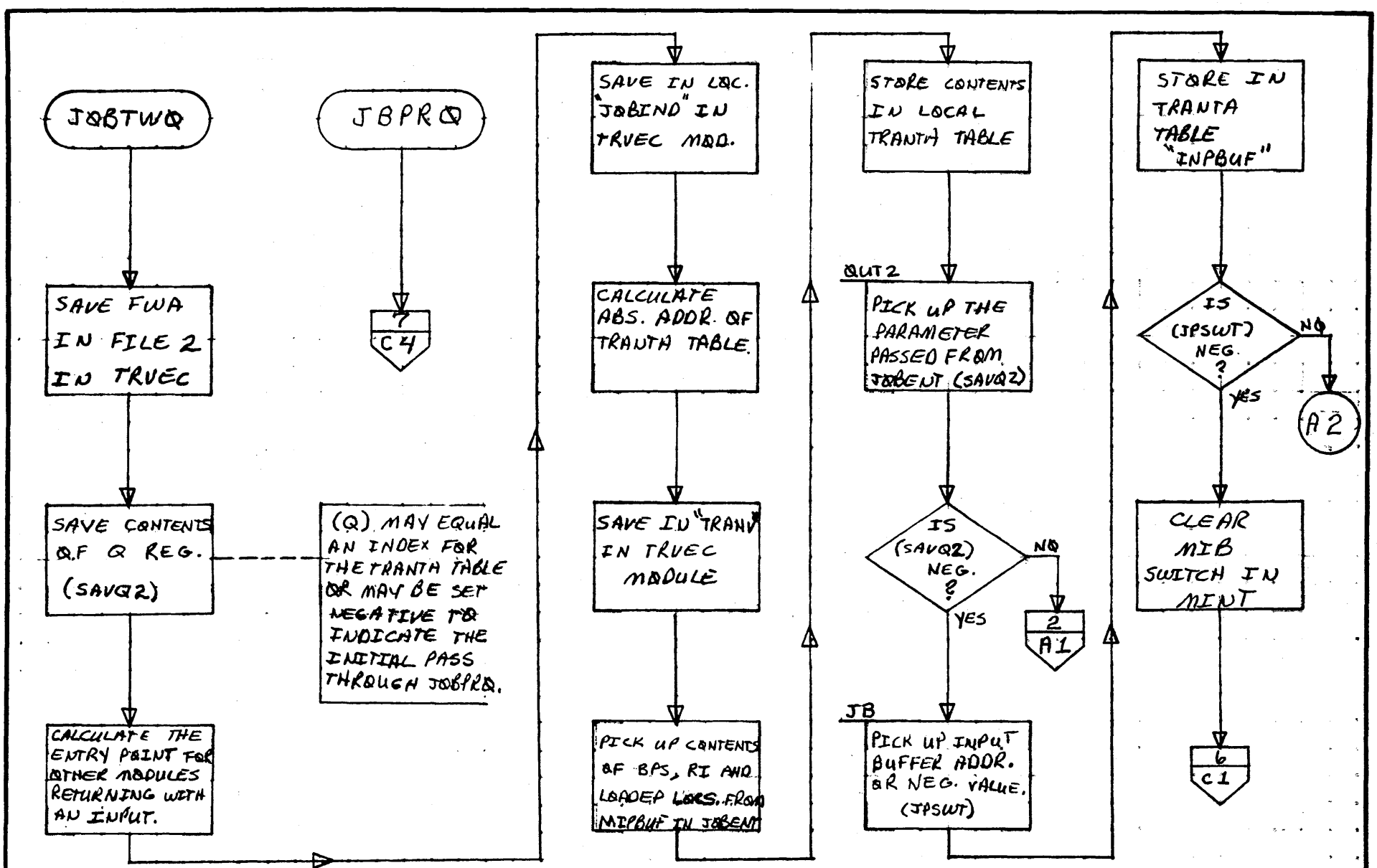
28.9.8 RESUME * (cr)

This routine processes the * (cr) statement which causes the Job Processor to continue at the beginning of scratch. Upon entry the absolute location for load and go, $E4 is set to one. This location is unprotected. Return is through the RF3 pseudo entry point.

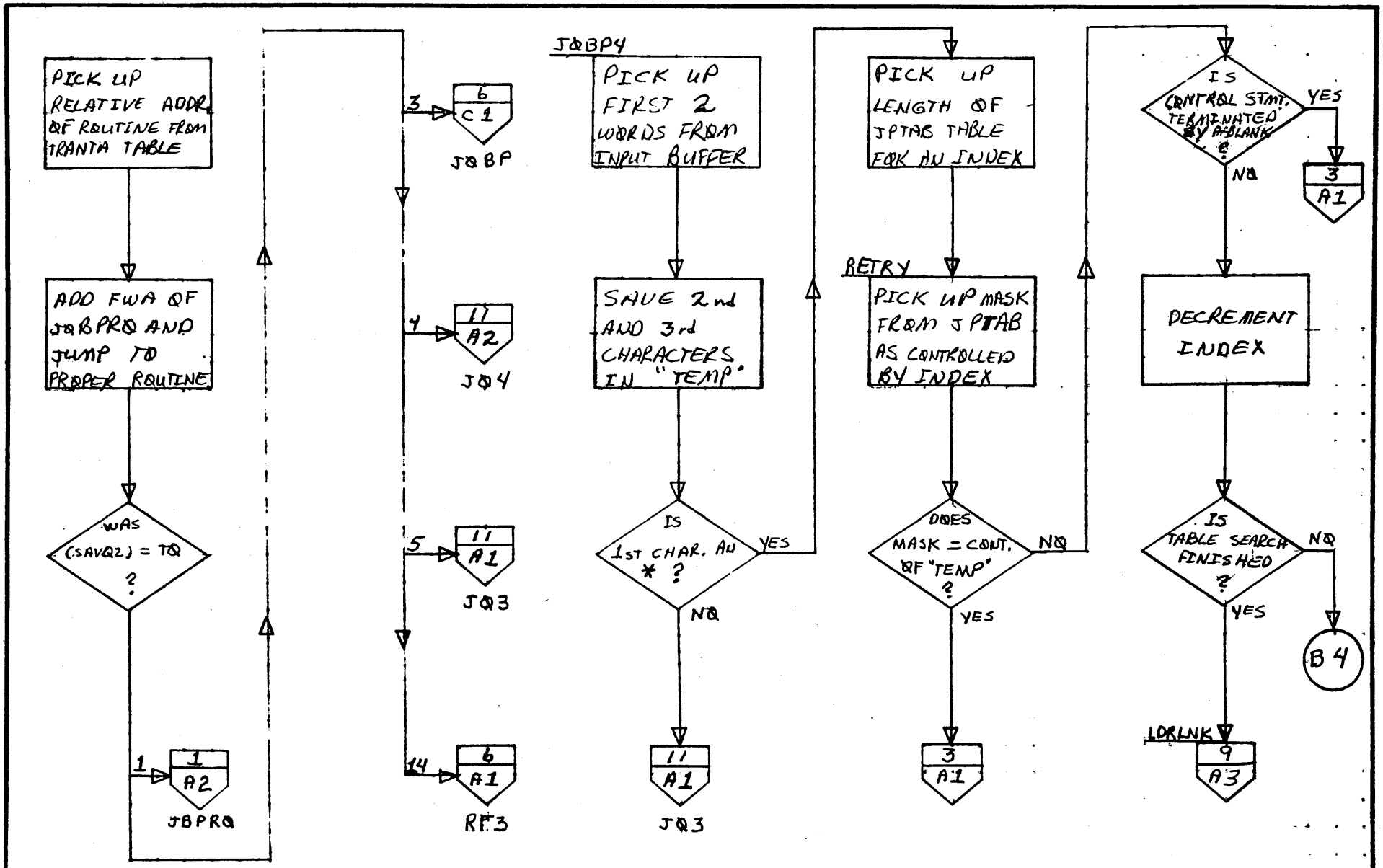
28.9.9 SGNOFF *Z (cr)

This routine processes the *Z statement which marks the end of Job Processing. Upon entry the Job Processor not in core flag {JOBIND} and the protect processor flag {LOADIN} are cleared. The LOADIN flag when set allows the loader to read and write below the scratch area on mass storage. The LGO {$E4} location is then set to one. The address of the Request Exit {$B9} is then stored in the RCTV table in NMONI for the Status, Exit, Loader, Core and GETfile requests.

Set the protect fault return to the Dispatcher switch {IPl} and reset the *V switch. The recovery indicator {RI} flag is then checked. If set, the flag is cleared and a jump is made to J0BENT to schedule RC0VER, otherwise Exit is made to RELFLE in MINT.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JOBPRQ		PROJECT MGR.		
		PAGE 1 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



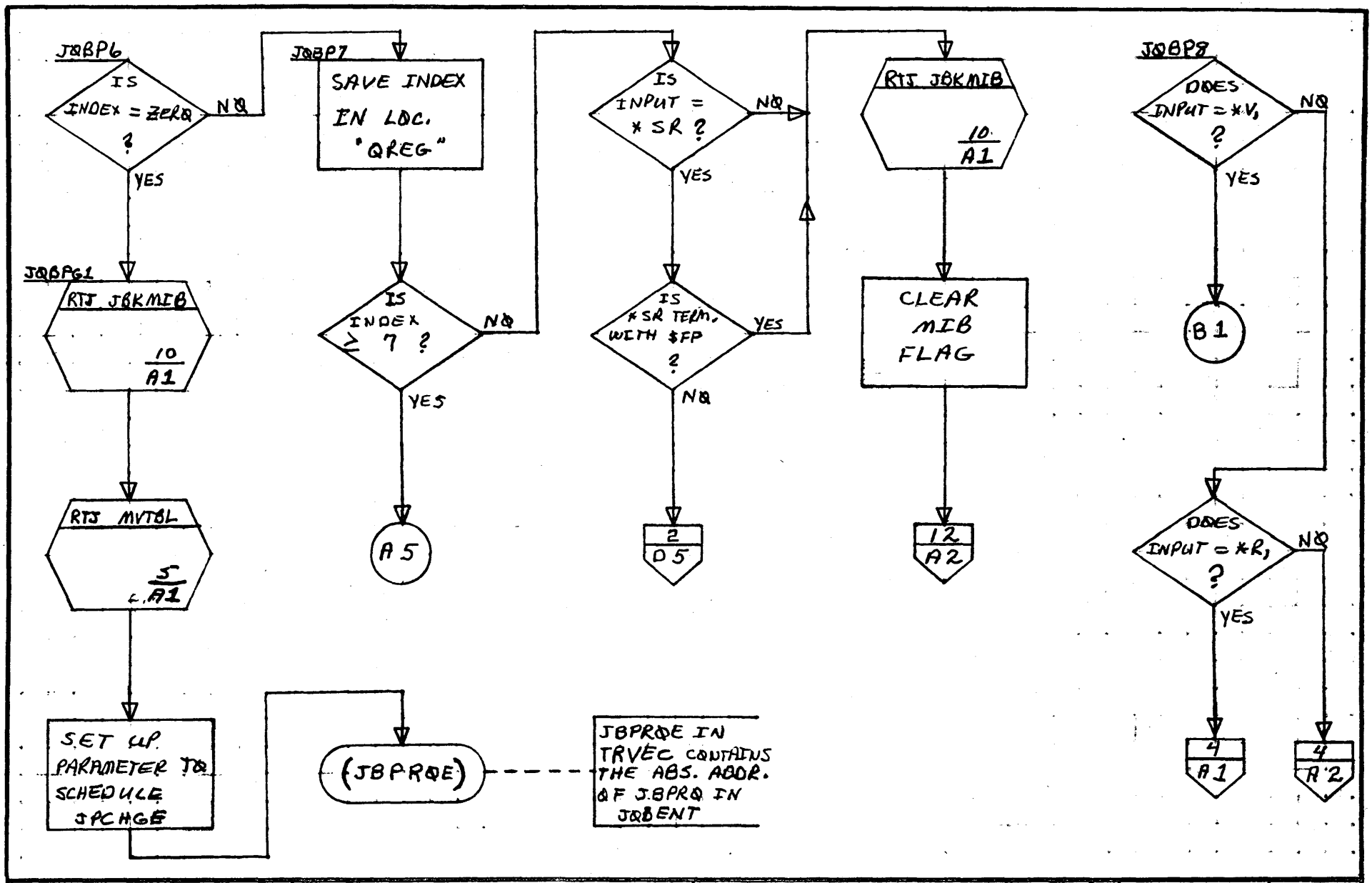
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE JOBPRO	PROJECT MGR.			
	PAGE 2 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

28.9

AA1385 (FORMERLY CA127-1)

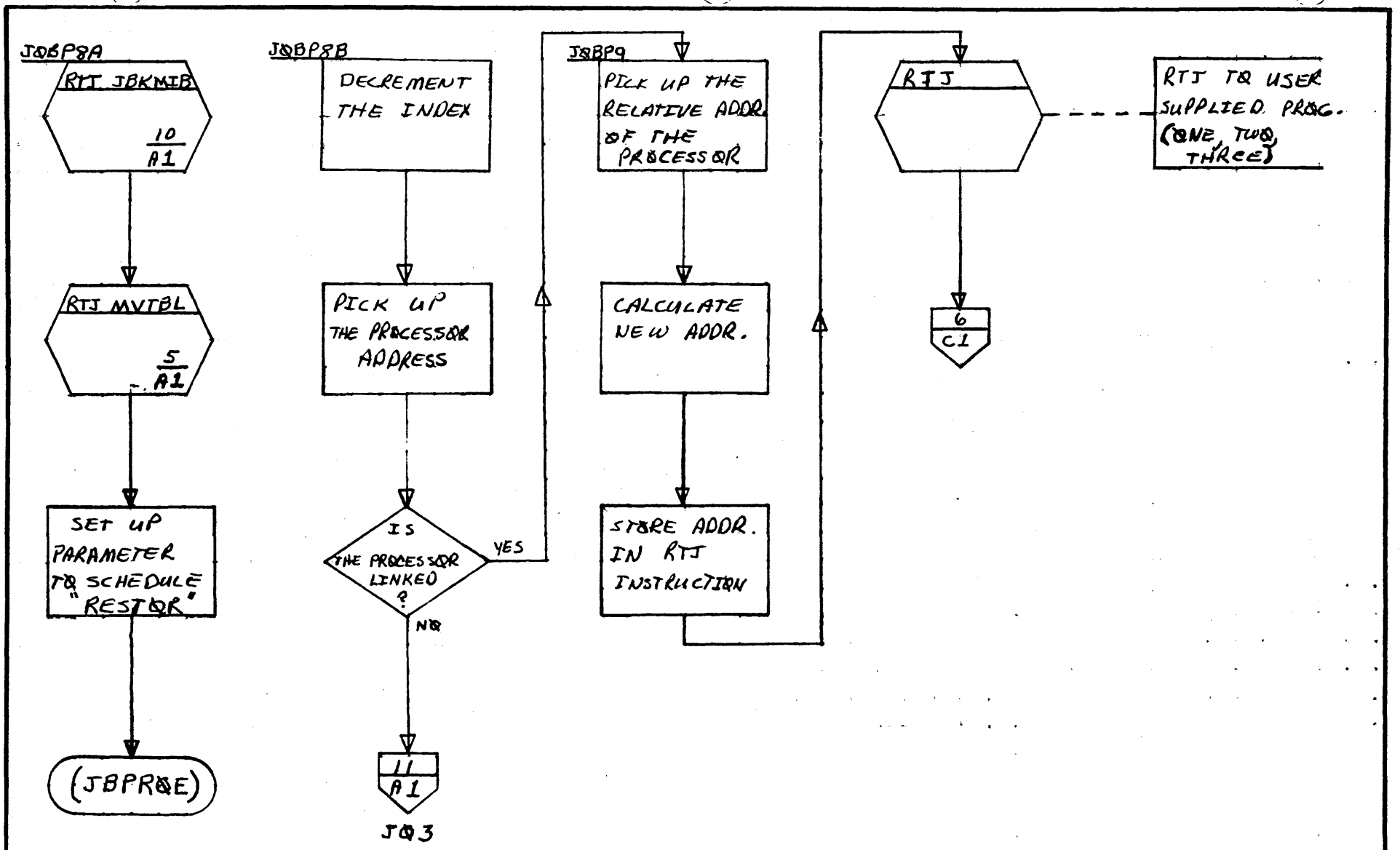
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JOBPRQ		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

28.10



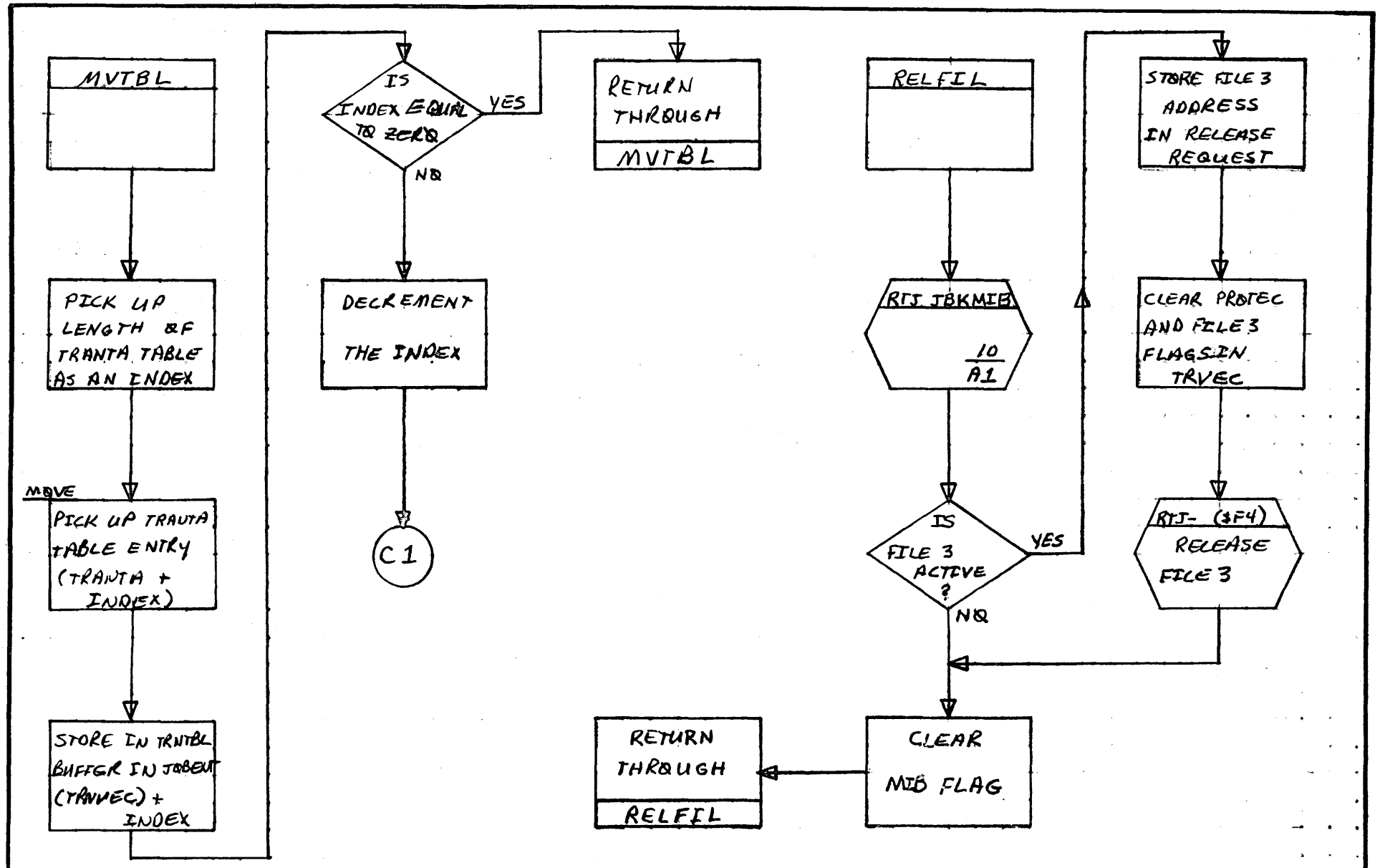
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JOBPRO		PROJECT MGR			
FLOWCHART ☐		PAGE 4 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

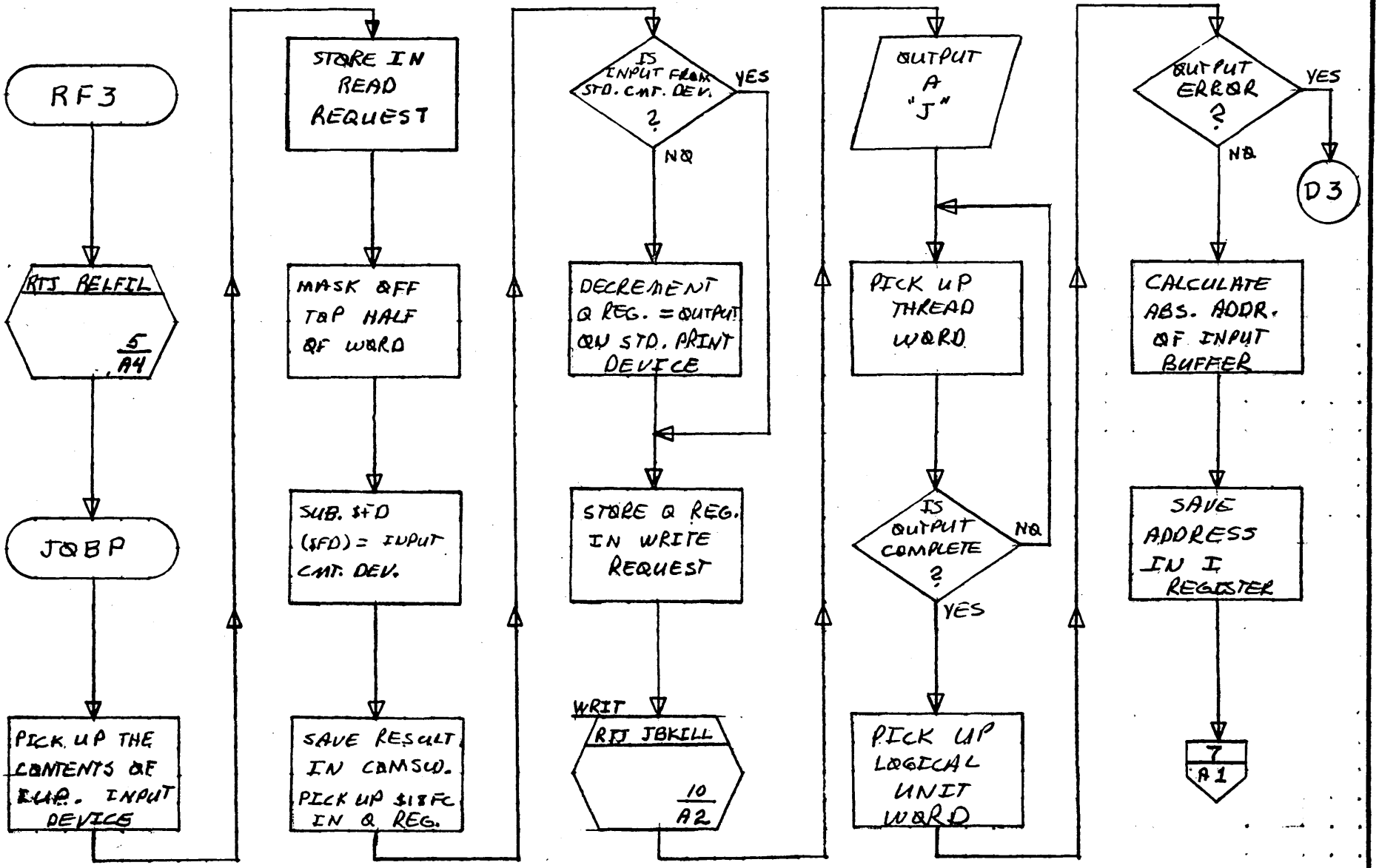
PRINTED IN U.S.A.

MAR 5 1971

28.11



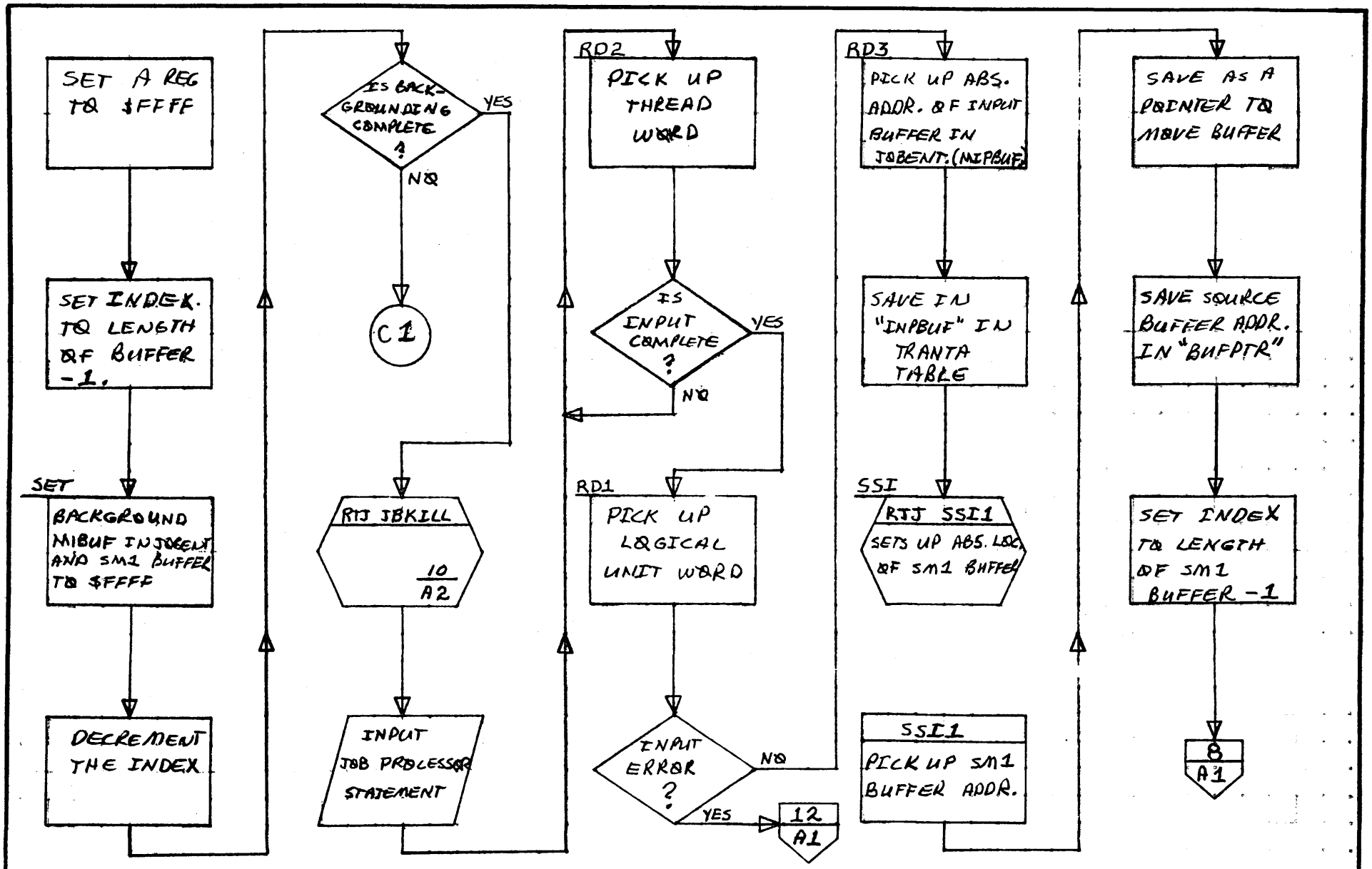
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JOBPRO		PROJECT MGR.		
		PAGE 5 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JOB PRO		PROJECT MGR.			
FLOWCHART ☐		PAGE 6 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

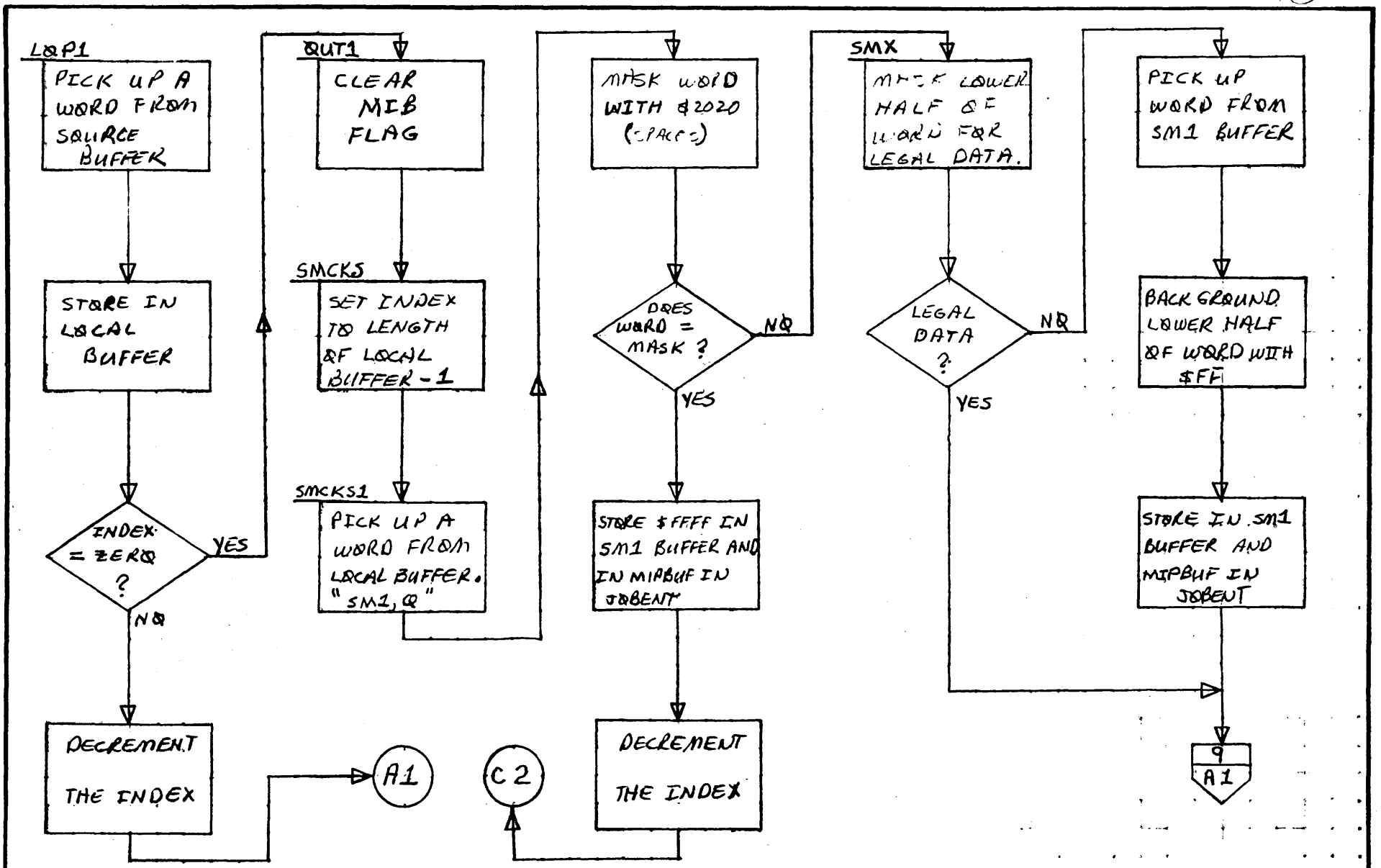
28.13



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE JOBPRO		PROJECT MGR		
		PAGE 7 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

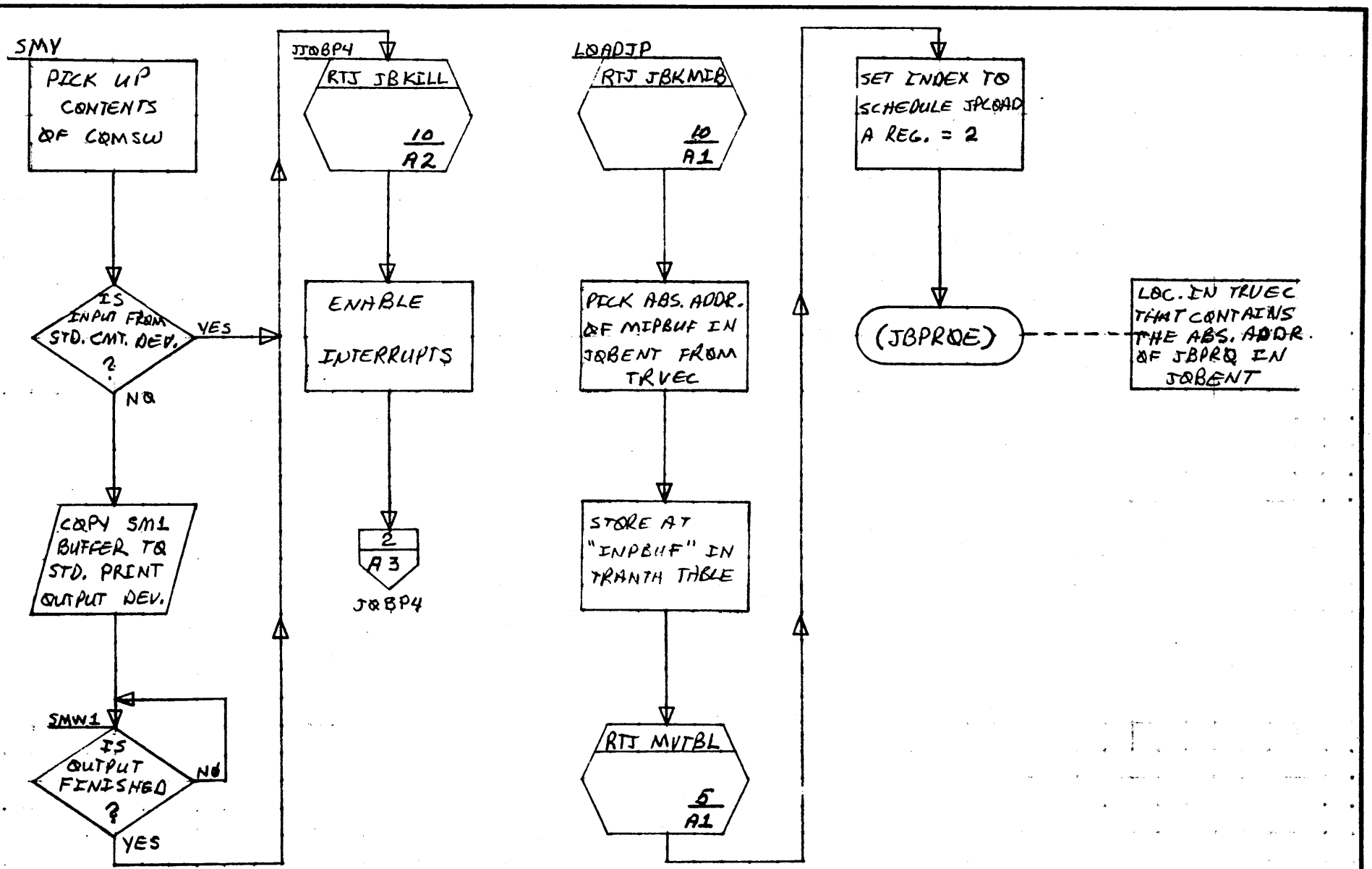
28.14



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE JOBPRQ	PROJECT MGR.			
	PAGE 8 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

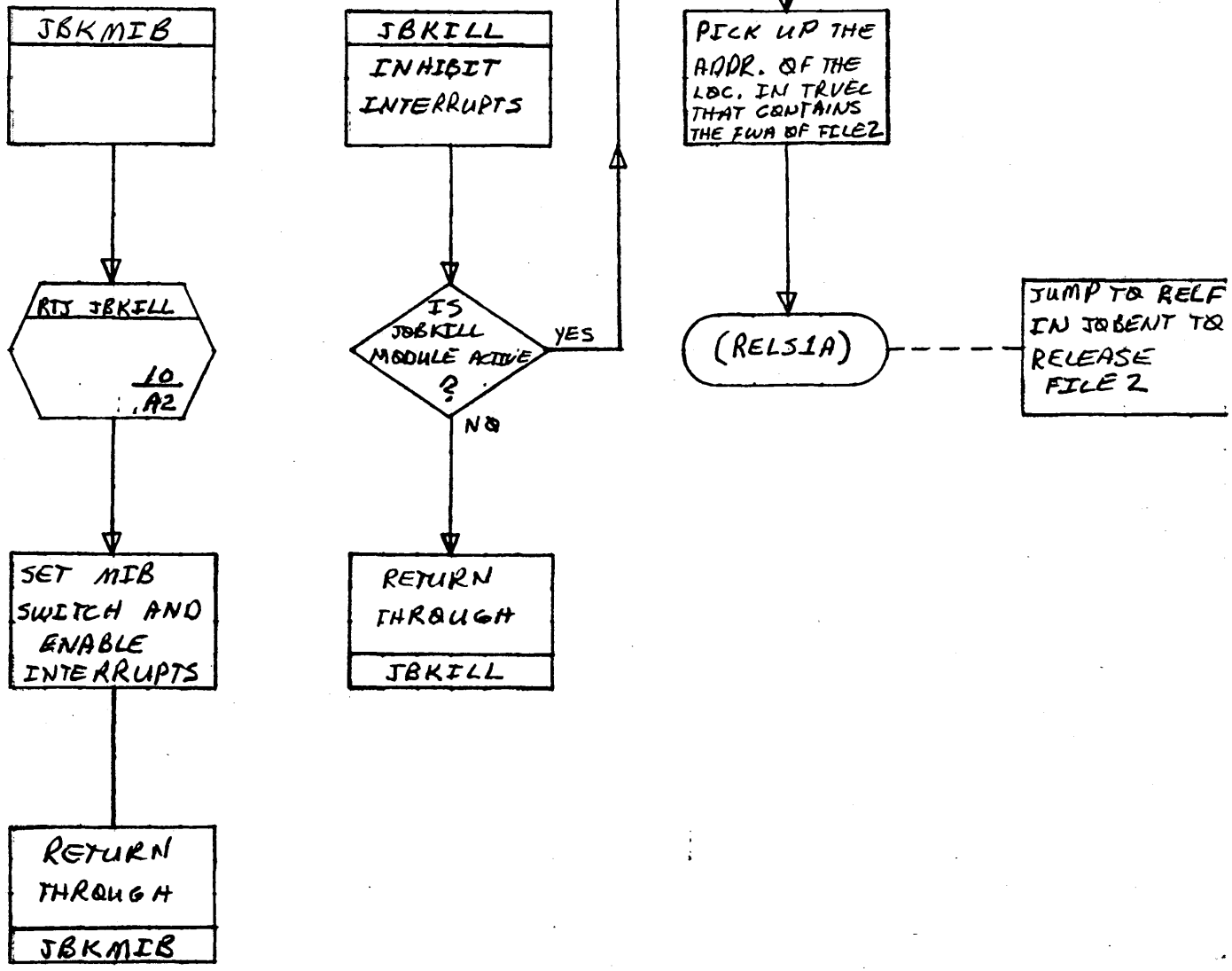
28.15



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JOBPRQ		PROJECT MGR		
		PAGE 9 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 6 1971

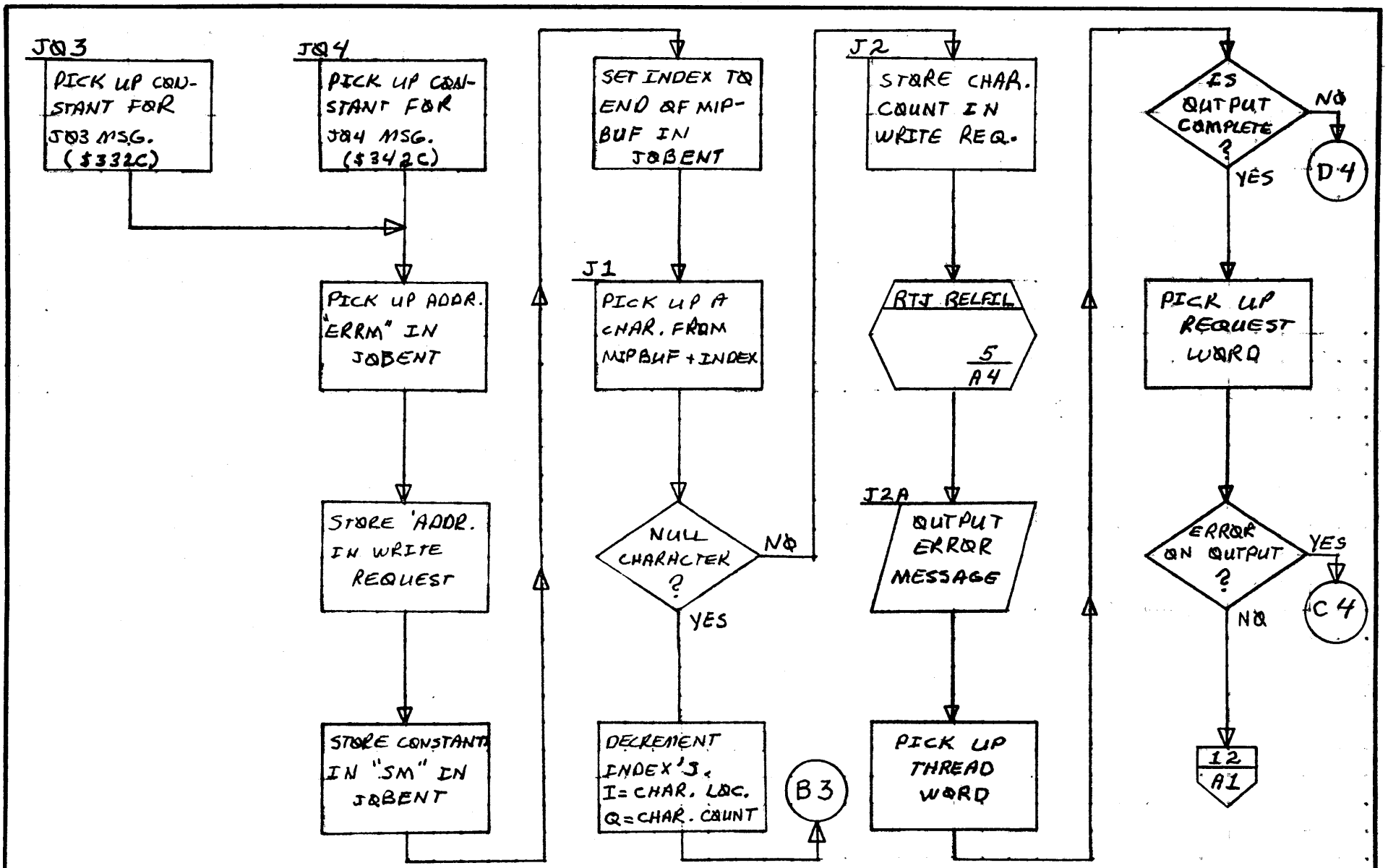
28.16



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE JOBPRO		PROJECT MGR.		
FLOWCHART ☐		PAGE 10 OF 13	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

28.17



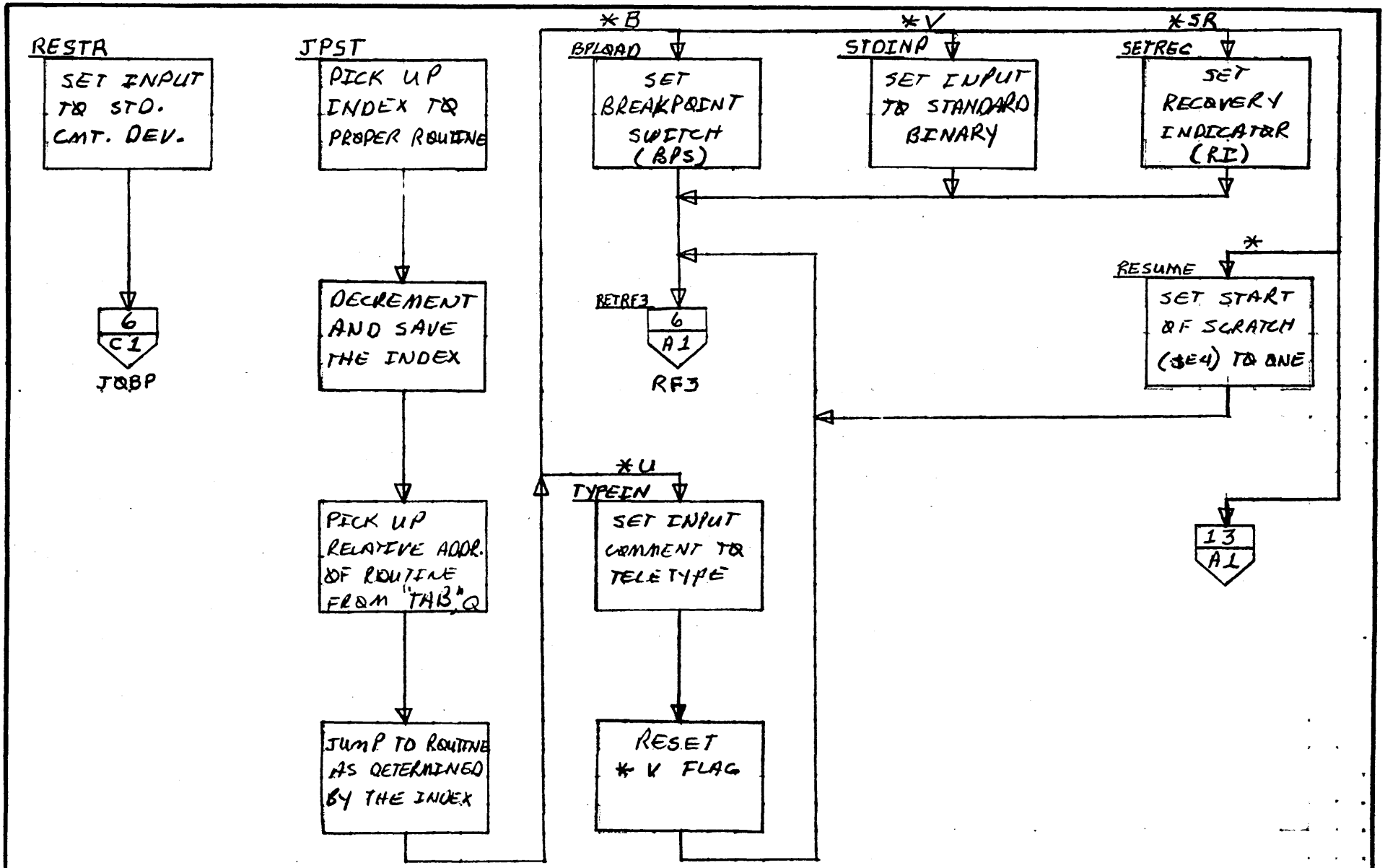
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JØBPRØ		PROJECT MGR.			
FLOWCHART ☐		PAGE 11 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 19/1

28.18

AA1385 (FORM CA127-1)

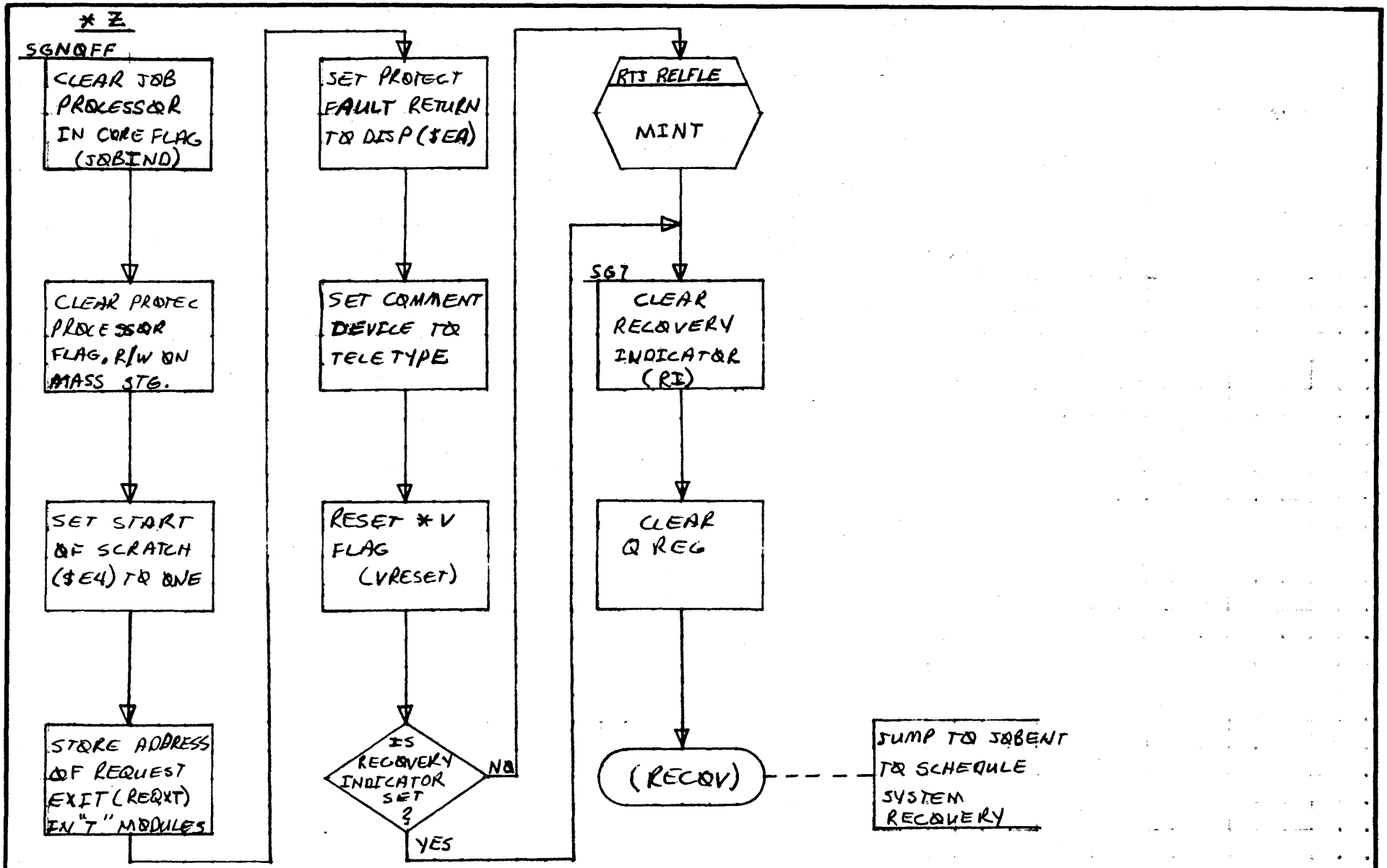
PRINTED IN U



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE JOBPRO	PROJECT MGR.			
	PAGE 12 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

28.19



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JOBPRO		PROJECT MGR.		
		PAGE 13 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

28.20

29.0 PROTEC . Protect Processor

29.1 FUNCTION

1. Analyze monitor call for invalid parameters.
2. Set up entry to a Preset Entry Point.
3. Initialize entry to users completion address after completion of unprotected I/O.
4. Detect illegal or legal violations
5. Read in the JBKILL module when requested.

29.2 ENTRY POINT NAMES

PRT Initial entry when PROTEC is scheduled in by JPLOAD.

IPROC1 Entry point when a Protect Violation occurs.

29.3 EXTERNALS AND DESCRIPTION

RCTV The table {in Monitor Entry for Requests} which is indexed by request code and contains the location of each request processor.

LOG1 Logical unit table {indexed by logical unit} which contains the alternate logical unit number and various indicators.

LOG1A Logical unit table {indexed by logical unit} which contains the core location of the physical device table entry corresponding to each logical unit.

TBLE A table for Hex to ASCII conversion located in the Internal Interrupt Processor.

LOADIN A location in TRVEC which when non-zero, indicates that relocatable loading is in progress.

UNPIO A location in TRVEC which contains a count of the number of unprotected I/O requests in progress.

UNPTIM A location in TRVEC which contains the number of unprotected TIMER requests waiting.

JBCNCL Routine in TRVEC to schedule PROTEC when JBKILL is requested.

SWAPCK Routine in DRCORE which decrements UNPIO on completion of an I/O request.

WDADR A flag that when set indicates that the words addressable disk drives to be used.

JKIN JBKILL in core flag {TRVEC}

JBCNFG Job Cancel wait flag {TRVEC}

TRNVEC Absolute address of the TRNTBL buffer in JOBENT {TRVEC}

PRORET Holds return address to JPLOAD for PROTEC {TRVEC}

TRANV Holds absolure address of TRANTA table in JOBPRO. {TRVEC}

FILE3 Holds FWA of PROTEC when in core {TRVEC}

IPL Entry points address of IPROCL {NIPROC}

LOCF Pointer to error processor routine. {TRVEC}

LPTRS Holds address of PTRS in PROTEC. {TRVEC}

PROTEC Mass Resident Protect Processor.

JKILL Entry point to JBKILL module.

29.4 ENTRY INTERFACES

When a protect violation occurs the Internal Interrupt Processor transfers control to PROTEC. The Protect Processor saves the machine state {Q,A,I, PL} and arbitrarily sets up priority level one prior to enabling interrupts.

29.5 EXTERNAL INTERFACES

Exit after a violation is as follows:

Monitor Call - users completion address

Preset - the preset entry point

Legal Violation - user's program at P+1

Illegal Violation - Cancel the job and back to the Job Processor

29.6 GENERAL PROGRAM INFORMATION

29.6.1 This table is indexed by request code. It contains the length of the parameter list in the upper 4 bits and the distance to the processors in bits 0-10.
Bit 11=1 for requests which are stacked but not threaded.

```
REQTB   ADC   F-REQTB
        ADC   Y-REQTB+$6000          READ       1
        ADC   Y-REQTB+$6000          WRITE      2
        ADC   BB-REQTB+$2000         STATUS     3
        ADC   Y-REQTB+$6000          FREAD      4
        ADC   BB-REQTB+$1000         EXIT       5
        ADC   Y-REQTB+$6000          FWRITE     6
        ADC   BB-REQTB+$1000         LOADER     7
        IFA   TIMERQ,EQ,1
        ADC   X-REQTB+$3800          TIMER      8
        EIF
```

```
IFA     TIMERQ,EQ,0
ADC     F-REQTB                 TIMER       8
EIF
ADC     X-REQTB+$2800           SCHDLE      9
ADC     F-REQTB                 SPACE       10
ADC     BB-REQTB+$1000          CORE        11
ADC     F-REQTB                 RELEAS      12
ADC     {Z-REQTB+$2000}         GTFLE       13
ADC     W-REQGTB+$5000          TAPE        14
```

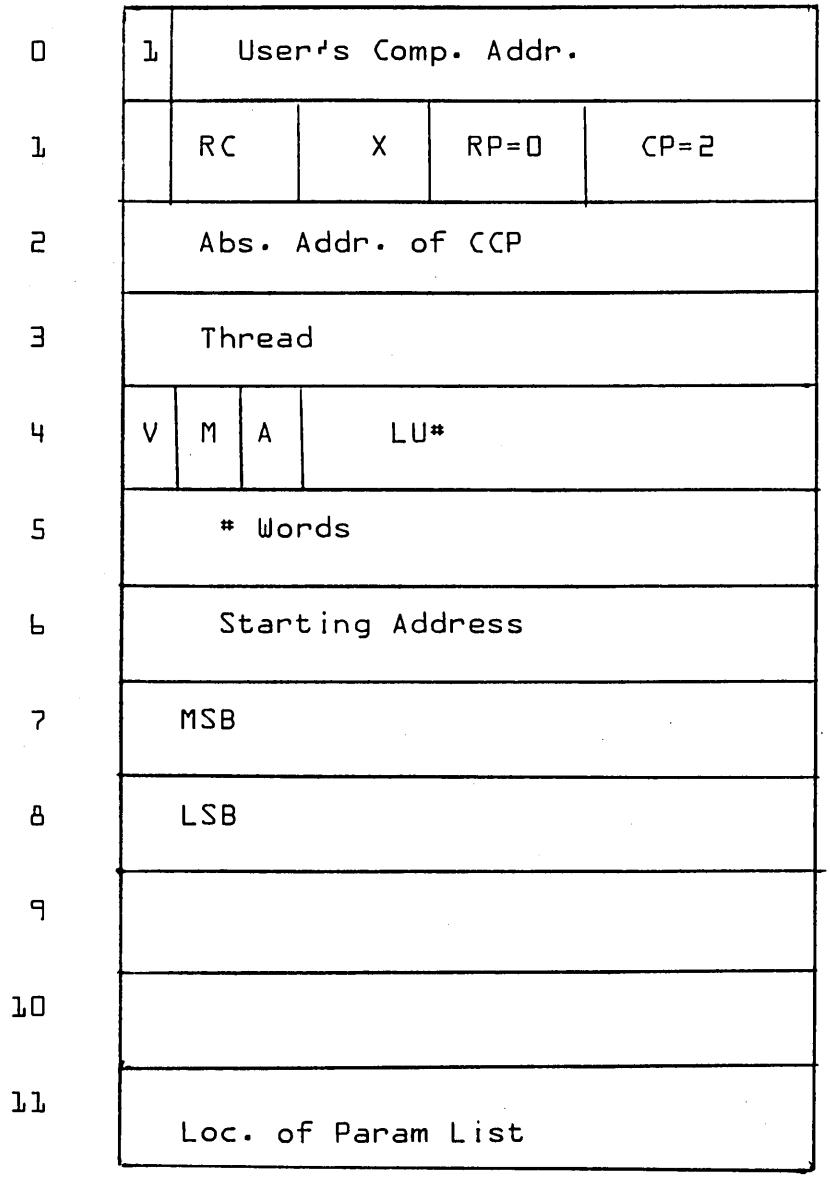
29.6.2 STACK

Length is 12N where N is the number of requests permitted on the stack at one time. Each slot requires 12 words.

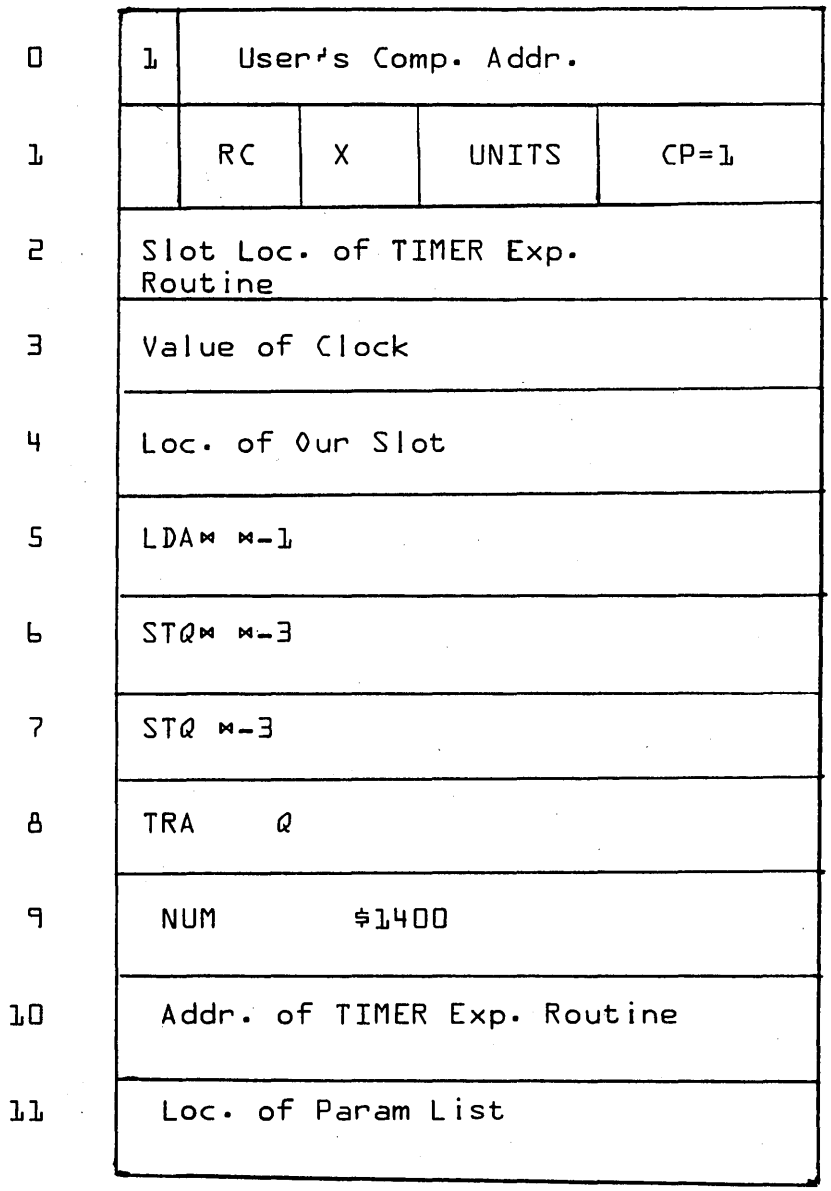
Bit 15 of word 0 designates that the slot is in use if set to 1.

Examples of the Protect Processor stack follows:

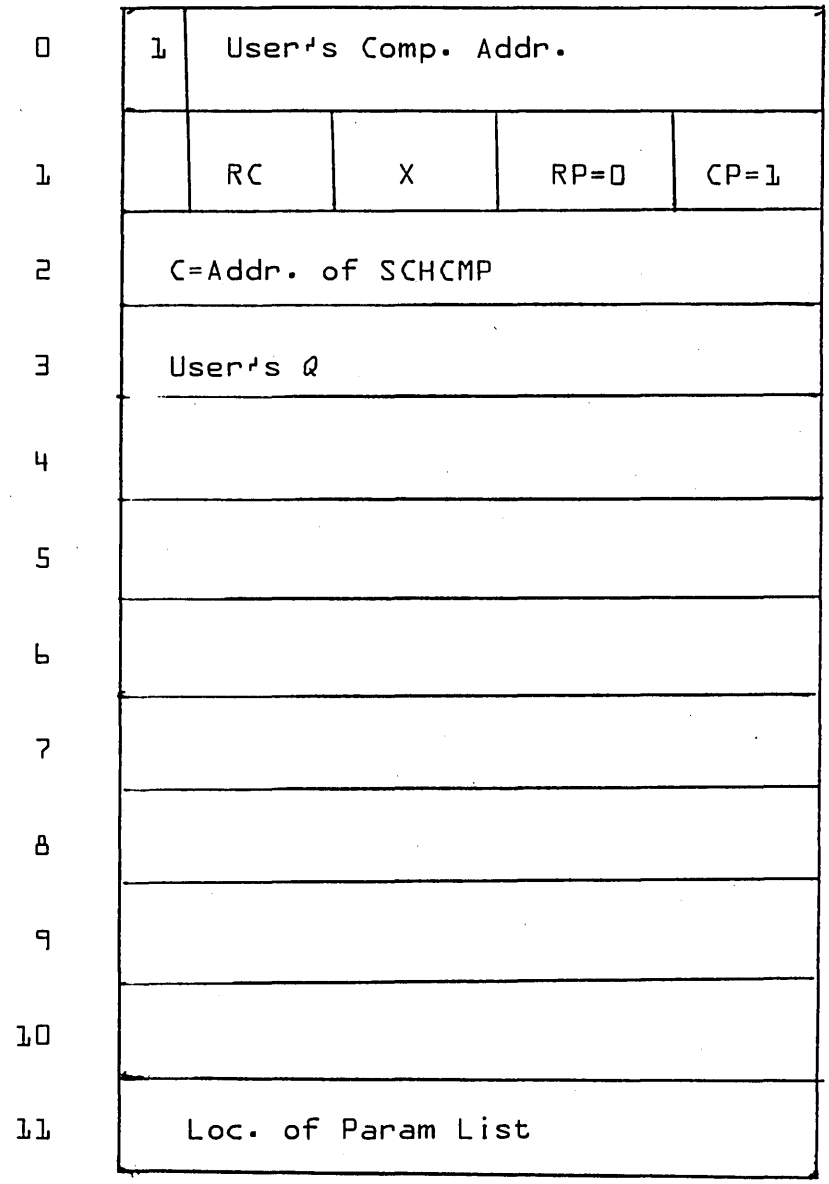
29.6.2.1 Stack for READ, WRITE, FREAD, FWRITE



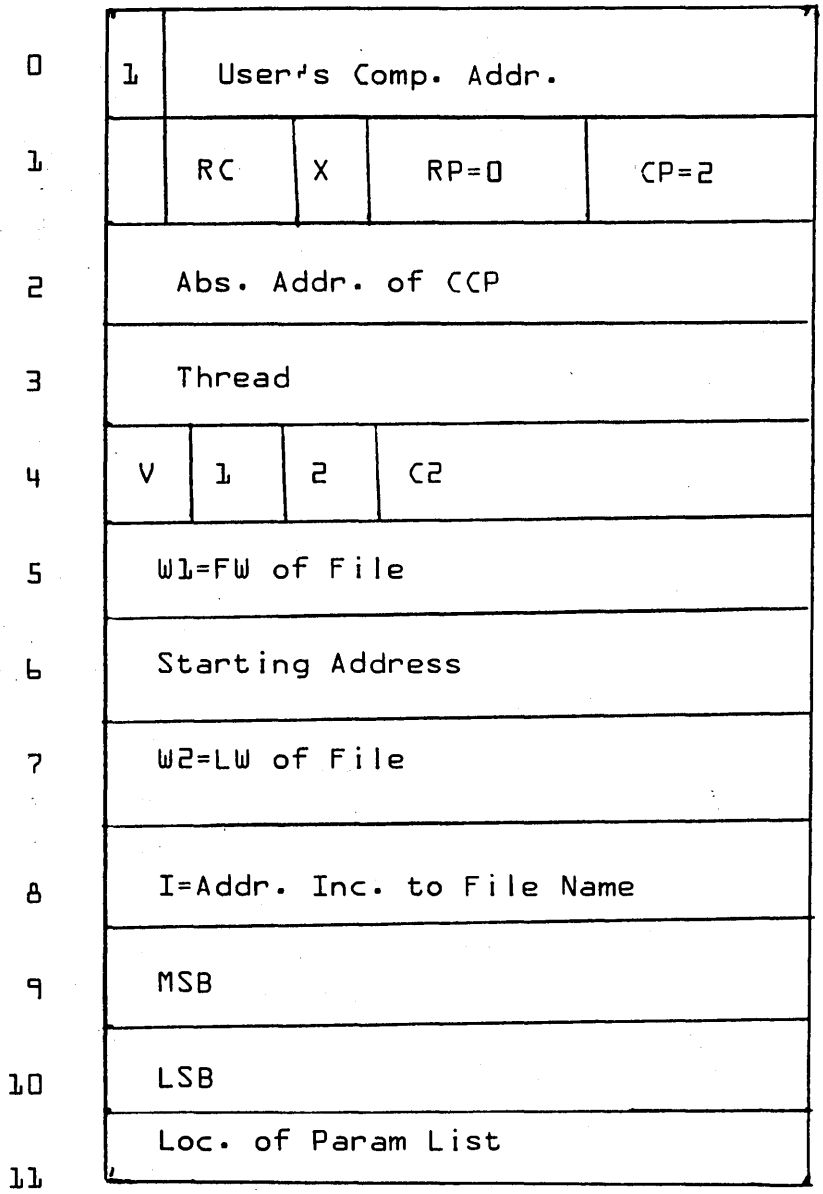
29.6.2.2 Stack for TIMER Requests



29.6.2.3 Stack for SCHDLE Request



29.6.2.4 Stack for GTFILE



29.7 GENERAL DESIGN PECULIARITIES

CODE Largest request code +1

N maximum number of stacked requests

29.7.1 Adding Requests

To add requests it is necessary to add entries for each request code in REQTB. Be sure that CODE is always one greater than the largest request code used. The user may process his request with any of the available subroutines {X,Y,Z,G,J} or specify his own via PROCTB.

29.7.2 Stacking

The maximum number of requests allowed to be stacked at one time is set to five. To change this it is necessary to change the value of N. This will automatically set the size of the STACK table and all indices.

29.7.3 Overlays

29.7.3.1 Protect Stack

Upon initial execution of PROTEC the area ▽PRTA▽ is used to place parameters into other areas which require them. After it is used to fix parameters, the area is cleared and then becomes the Protect Stack {TABLE}. This table is used to stack unprotected parameter lists for execution.

29.7.3.2 AREA. - Equated to the area of routine C.

This area is where the overlay of the error routine and JBKILL begins. Routines which are in this area initially are not required after an error has occured or JBKILL is scheduled.

29.8 PROGRAM LOGIC

This program operates at level one and is not re-entrant as no protect violations occur above level 1. It processes all protect violations including illegal violations, and unprotected protected communication in unprotected monitor calls. It does validity checking for monitor calls which need it.

On initial entry to PROTEC a check is made to see if JBKILL is requested (Q Neg.). If requested a jump is made to the routine GET which reads in the JBKILL module and jumps to it to be executed. If not, a jump is made the routine PRTA which does some initial housekeeping, sets up the address of IPROC1 in IP1 in NIPROC to handle interrupts and returns to the JPLOAD module (PRORET) address.

On entry to IPROC1 after a Protect fault has occurred, the Protect Processor stores Q, A, I, P and priority, then sets up level one. The Protect Processor first attempts to determine if the violation resulted from a monitor call. This is done by examining the contents of the words which must contain °54F4° if the violation is indeed a monitor call. All monitor requests cause operation of subroutine A, which analyzes the request code and exits to the correct processor.

Protect violations which are not monitor calls are analyzed in subroutine C. The violation is examined to see: (1) if an attempt was made to communicate with certain programs whose entry point names are in the list of preset entry points, (2) if it was a jump to the Dispatcher, or (3) a violation from a legal instruction.

Appropriate action is taken as follows:

Preset - exit to the program whose entry is a preset

Dispatcher - exit via a jump to the Exit request processor

Legal Violation - return to the users program at P+1 {subroutine K}

Illegal Violation - print the appropriate error message and schedule JBCNCL. {subroutine E}

29.8.1 COMMUNICATION

If the caller was at level zero, the Protect Processor must go through the dispatcher in order to operate any level one programs {completion routines} that were scheduled as a result of interrupts occurring while the Protect Processor was in operation. This must be done because if we immediately go to level zero, level one programs could be left waiting. This is not tolerable.

In order to go through the dispatcher and also preserve the mark in the called routine, the Protect Processor then can safely cause entry to the called program with interrupts off.

If the caller is at leve one, the entry is achieved without need for rescheduling.

29.9 SUBROUTINE LOGIC

29.9.1 Subroutine A

This subroutine does preliminary legality checking of requests and passes control on to the appropriate processor. The legality checks are as follows:

1. The Monitor Entry for Requests must contain a mark from the RTJ- {$F4}.

2. INDIR requests must point to an unprotected parameter list.

3. Sign bit of word containing request code must always be zero.

4. Request code must be less than CODE.

5. A processor must be resident. This is checked by examining the RCTV table to find out if the request processor is patched in. Unpatched processors leave 7FFF (16) in the RCTV table in NMONI.

Failure of any of the above checks will result in an illegal request diagnostic except for number one which will be flagged as a protect violation.

When the checks are completed, the REQTB table is used to get to the next level processor. The processors available and their functions are:

BB Passes the request to the Monitor Entry for Requests at the level of the caller. No further processing is done. Requests using this routine are: CORE
STATUS
EXIT
LOADER

F Gives an illegal request diagnostic and terminates the job. This is used internally and also for requests which are illegal from unprotected core.
SPACE
RELEAS
{request code zero}

X A special processor for TIMER and SCHDLE requests.

Y A processor for READ, WRITE, FREAD, and FWRITE requests.

Z A processor for GTFILE requests.

W A processor for TAPE motion requests.

29.9.2 Subroutine BB

This subroutine is entered for requests which are not legality checked such as CORE, STATUS, EXIT, LOADER and also to set up exit to a present entry point.

The Protect Processor schedules itself down to level 0 in order to allow processing of any level one programs which may have occurred. Exit is then made to the return {P+ request length} or to the preset.

29.9.3 Subroutine X

Operation begins with use of subroutine G and then is separate for TIMER and SCHDLE requests.

TIMER requests are stacked in the Protect Processor stack and there await expiration of the delay. At expiration, the TIMER request count {UNPTIM} must be decremented, the stack slot marked empty and the user's program must be entered at the desired level. This is accomplished by saving the user's completion address and substituting an address in the table to which a bookkeeping routine is moved.

When the delay time expires, the bookkeeping routine is entered. After completing its operation, the bookkeeping routine exits to the user completion address via the SCHCMP routine.

SCHDLE requests have their completion address and user parameter saved and the location of subroutine SCHCMP substituted for the completion address. In this way when the SCHDLE request is completed, the SCHCMP routine can complete release of the table entry and enter the user's program.

29.9.4 Subroutine Y

This routine calls subroutines G, J and L in order to process READ, WRITE, FREAD and FWRITE requests. It exits to subroutine H after updating the number of unprotected requests, and storing the location of the stacked requests in the user's thread to set it non-zero.

29.9.5 Subroutine Z

This subroutine calls subroutines G and J to complete preliminary processing and then performs the following legality checks peculiar to GTFILE.

1. File buffer must totally reside in unprotected core.
2. W2 must be greater than or equal to W1.
3. Specified sector numbers must not be negative.

The following operations are also performed:

1. A bit is set to tell the GTFILE processor that the request is unprotected.
2. The pointer to the file name is adjusted for the request's position in the Protect Processor Table.
3. The stack entry is set full.

29.9.6 Subroutine G

This routine first looks for an empty slot in the stack. If no empty slot exists and the caller is at level zero, the request is repeated. If the caller is not at level zero the job is deleted because no empties can ever occur while we are at level one, since completion routines are executed at level one and, therefore, cannot be entered until we exit.

The following legality checks are performed:

1. Completion address cannot be a directory call.
2. Completion address cannot be protected.
3. Thread for non-SCHDLE and non-TIMER requests may be zero. If a thread is found non-zero, the request is repeated for level zero callers and causes job termination for level one callers. The reasoning is the same as for the stack full condition.

The following operations are performed:

1. Completion address is absolutized and stored in the first word of an open slot.

2. Ten words of the request are moved to the slot starting at word two of the slot which contains the request code word.

3. The common completion address is stored in the 3rd slot cell so completion bookkeeping can be done.

4. Request priority is set to zero in stack.

5. Completion priority is set to one in stack.

6. X is set to zero in the stack.

7. The location of the user request is saved in the last cell of the slot.

29.9.7 Subroutine J

This subroutine performs the following operations:

1. Absolutize logical unit and store in stack.

2. Absolutize starting address and store in stack.

The following legality checks are performed:

1. Logical unit must be less than or equal to the largest and not zero or one.

2. Starting address must be unprotected for read type requests.

29.9.8 Subroutine L

The following operations are performed:

1. Length of block in transfer is absolutized and stored in stack.

2. If starting address is indirect, move sector number to stack for requests involving mass storage.

3. For requests to the library unit, add scratch sector start to sector number unless loading is in progress. {lets Loader get at library}.

4. Set stack entry full.

The following legality checks are performed:

1. The buffer must be in unprotected core for read type requests.

2. Sector number must be non-zero and positive.

3. A call may not read a write-only device or write a read-only device.

29.9.9 Subroutine H

This subroutine makes one indirect call pointing at the second word of the slot, thus executing the legal request stacked there. It then calculates the proper return to the caller and confirms that the return is unprotected. Violation of a return to be unprotected will cause a protect violation diagnostic. Subroutine continues on to subroutine K.

29.9.10 Subroutine K

This subroutine makes a phony entry in the interrupt stack and then exits to the dispatcher to effect return to a caller after a legal request has been initiated.

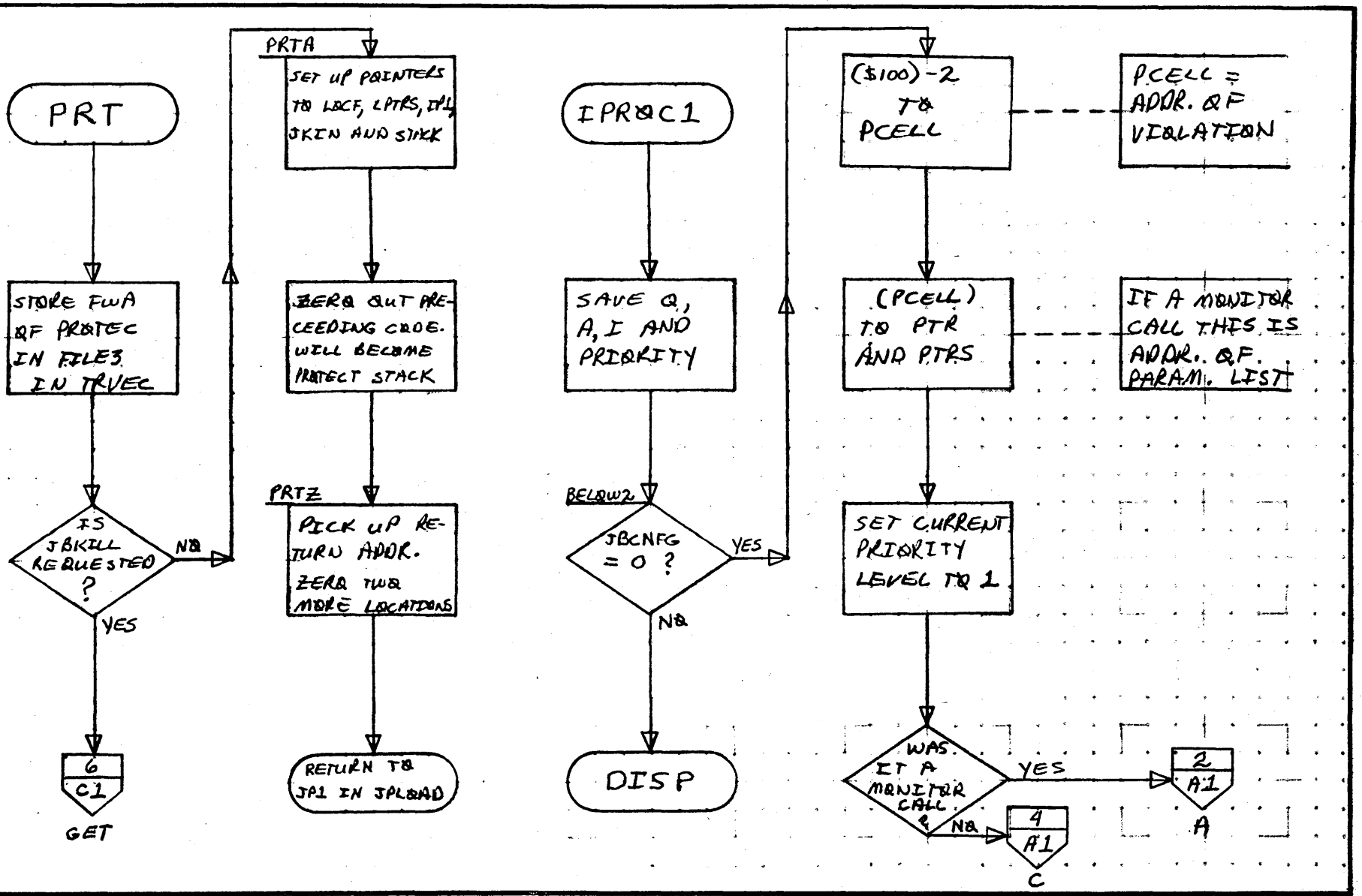
29.9.11 Subroutine CCP

This subroutine is entered upon completion of each I/O call from unprotected core. It does final bookkeeping and enters the users completion address.

29.9.12 Subroutine W

This subroutine analyzes Tape Motion Control requests.

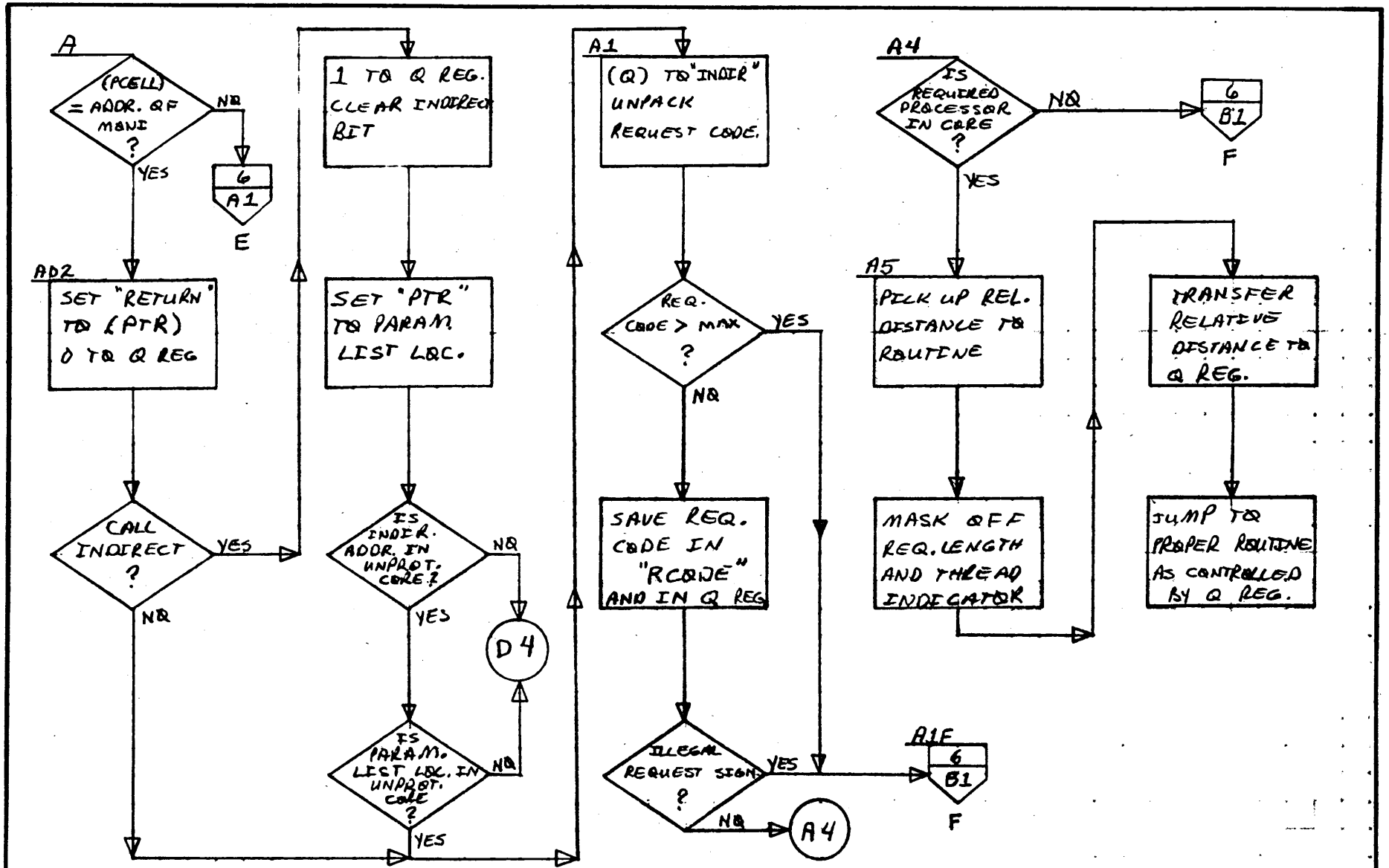
1. Checks legality of the logical unit.
2. Confirms that parameters are within range.
3. Checks to see if device is protected.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.			
FLOWCHART ☐	PROCESSOR	PAGE 1 OF 18	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

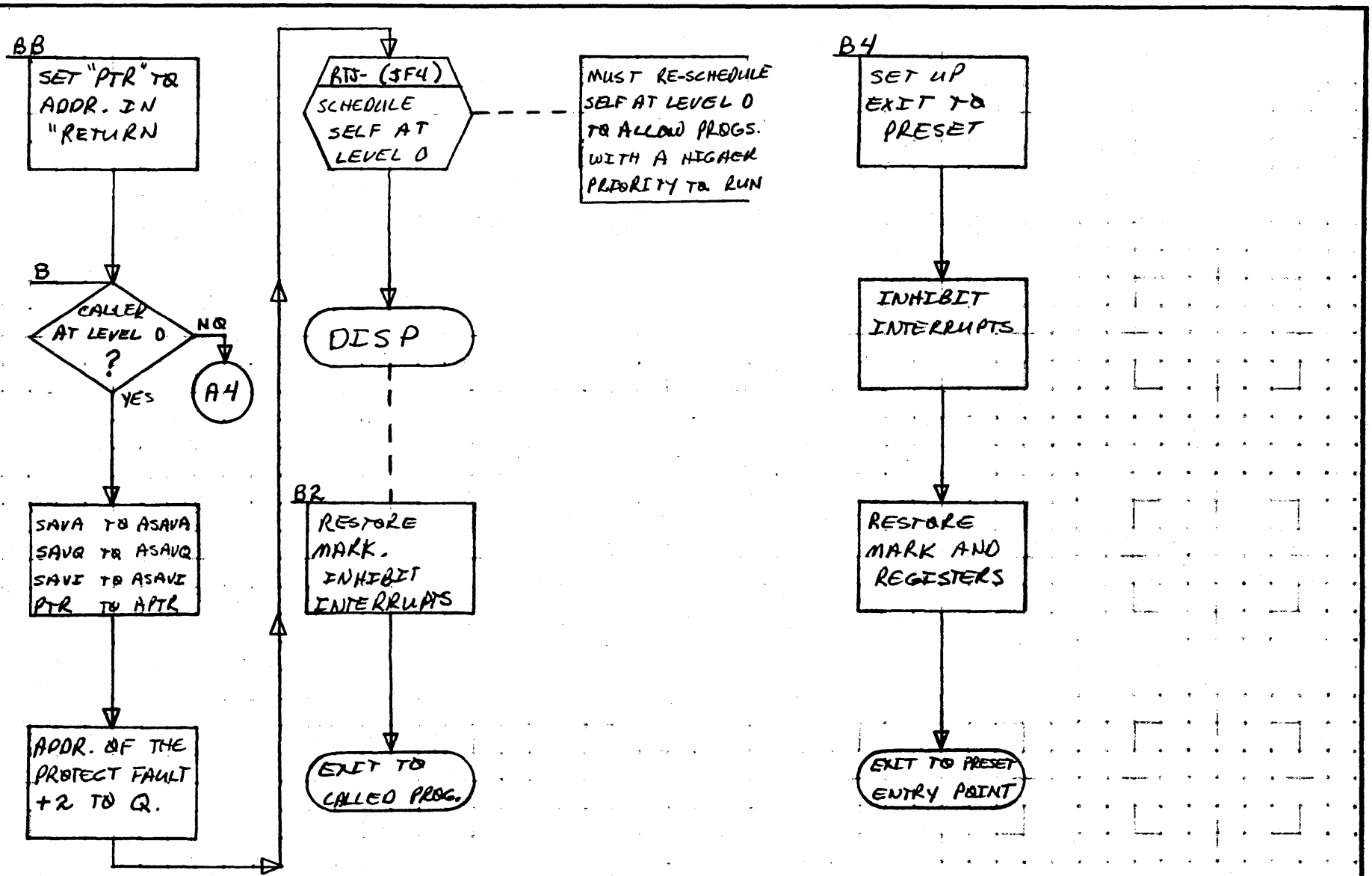
29.18



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.			
FLOWCHART ☐	PROCESSOR	PAGE 2 OF 18	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

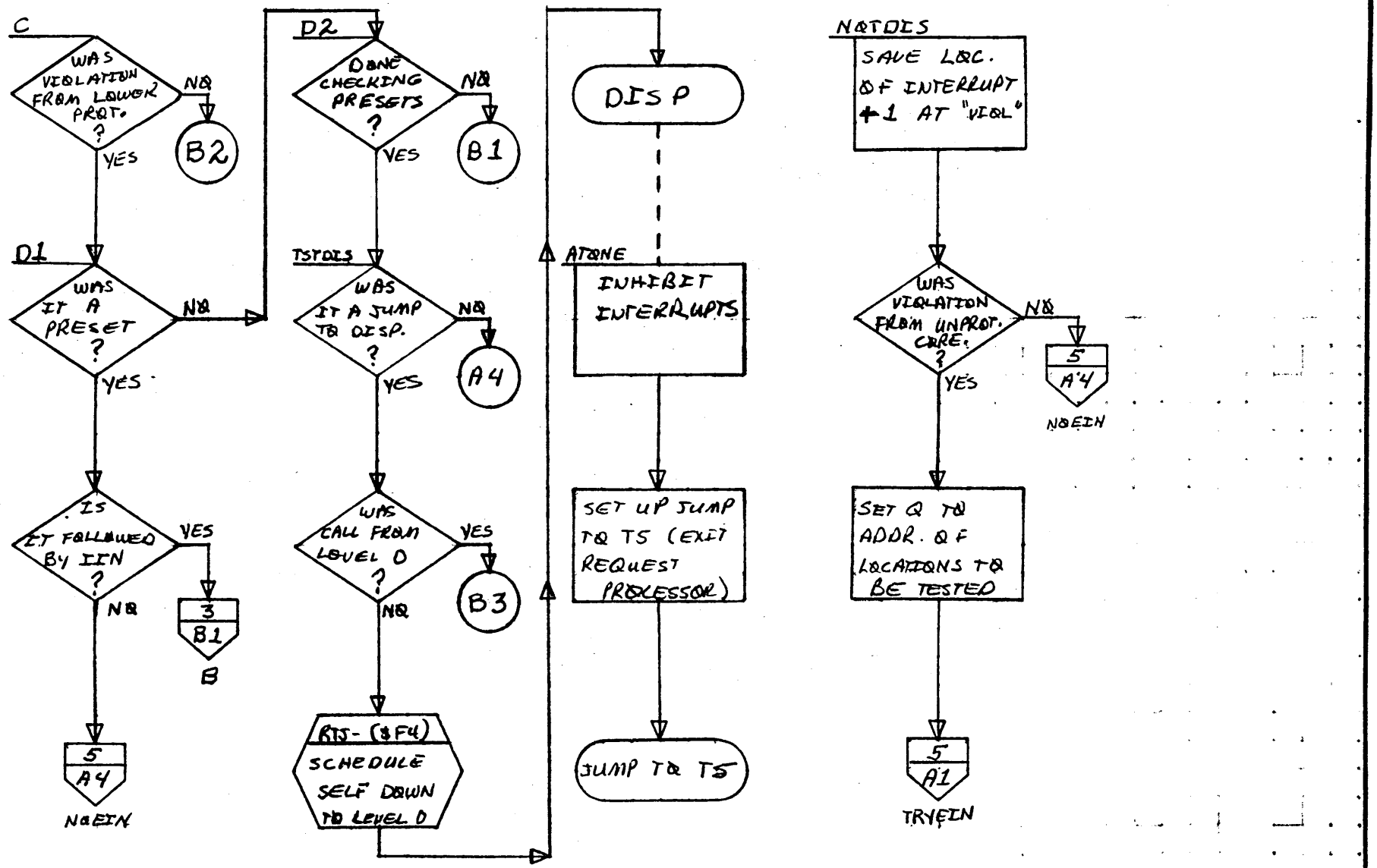
29.19



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.			
FLOWCHART ☐	PROCESSOR	PAGE 3 OF 18	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

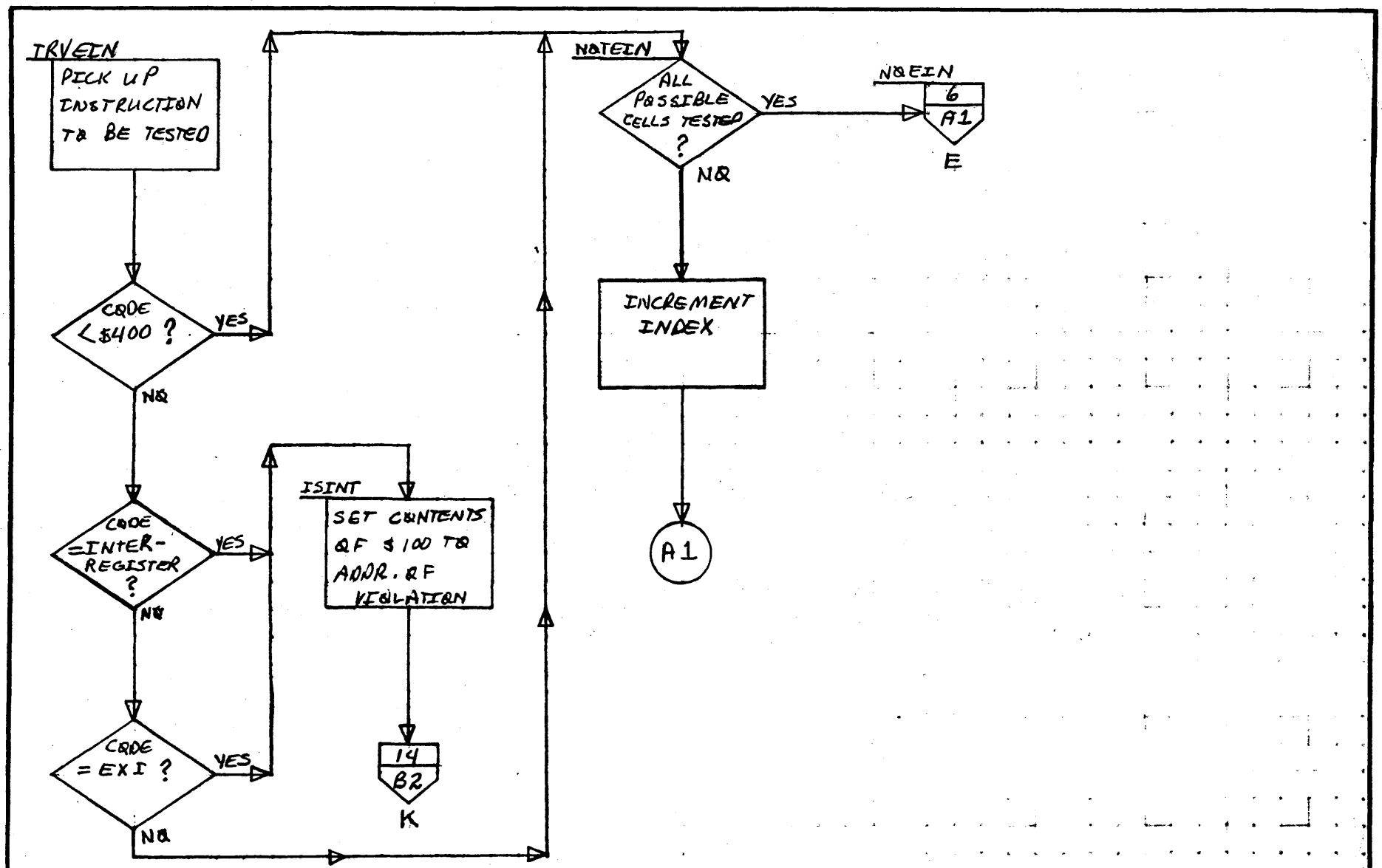
29.20



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 176	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.		
FLOWCHART ☐	PROCESSOR	PAGE 4 OF 18	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29.21



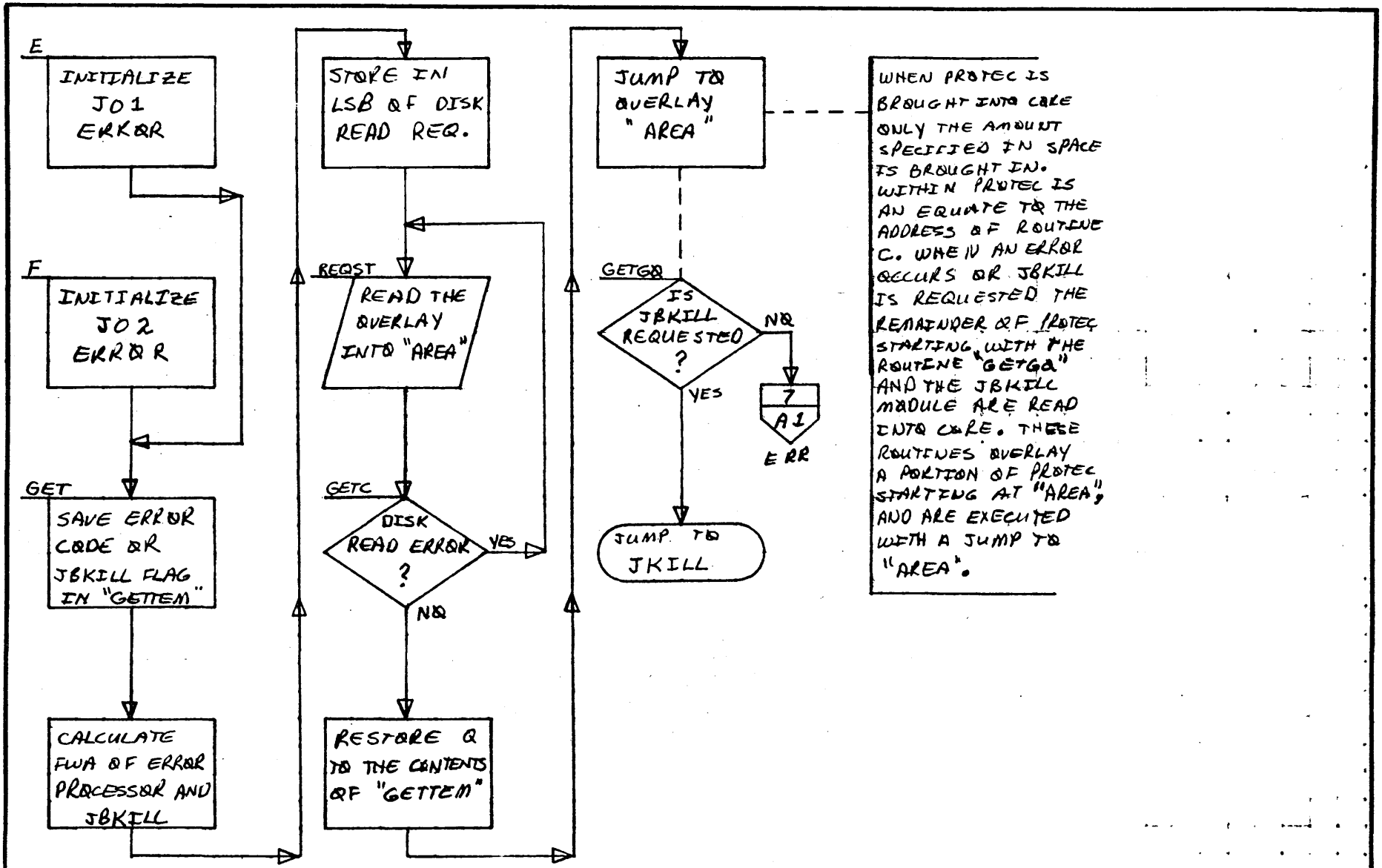
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.		
	PROCESSOR	PAGE 5 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29.22

AA1385 (FORMERLY CA127-1)

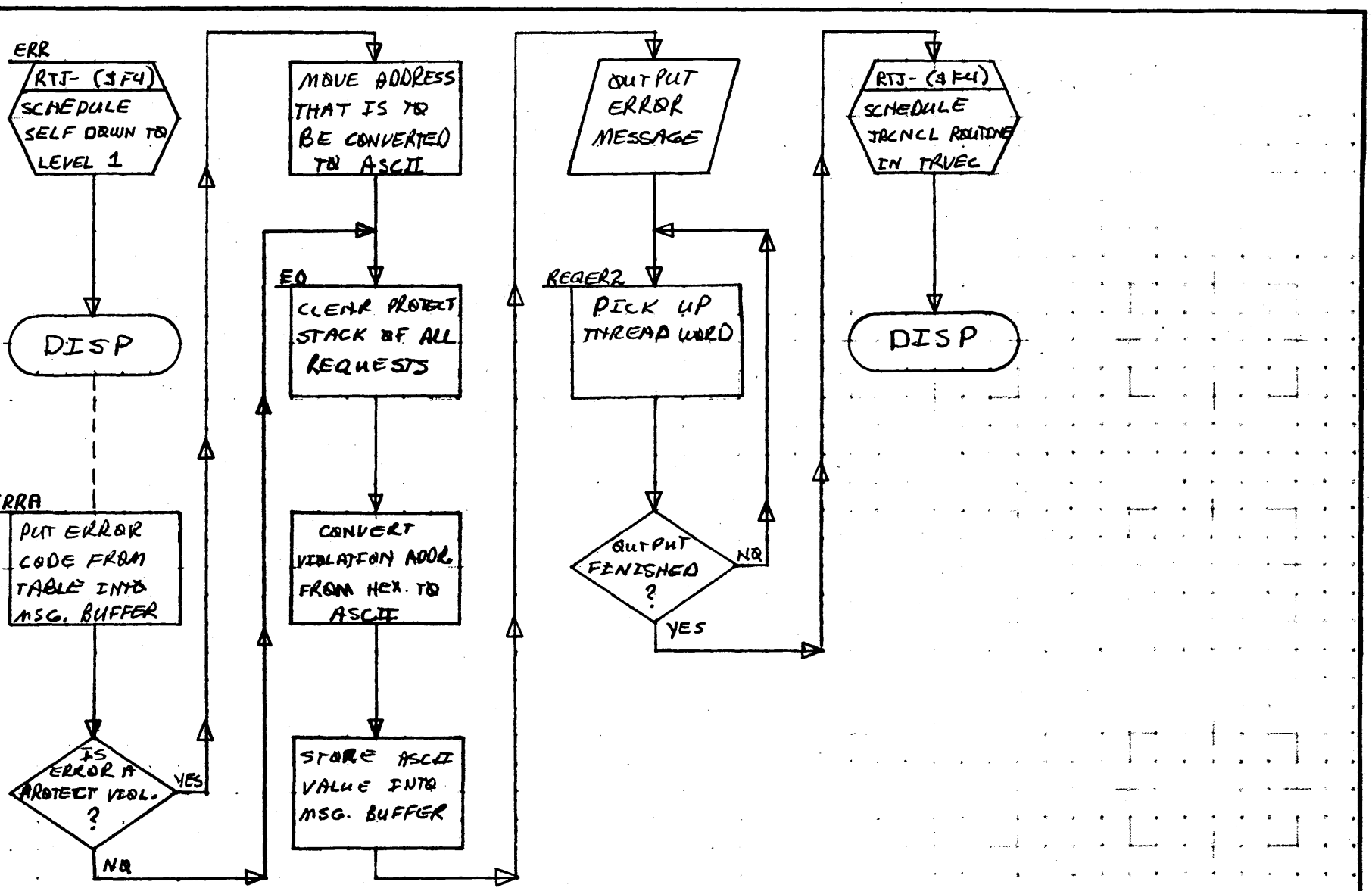
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PROTEC - PROTEC 1		PROJECT MGR.			
FLOWCHART ☐	PROCESSOR	PAGE 6 OF 18	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

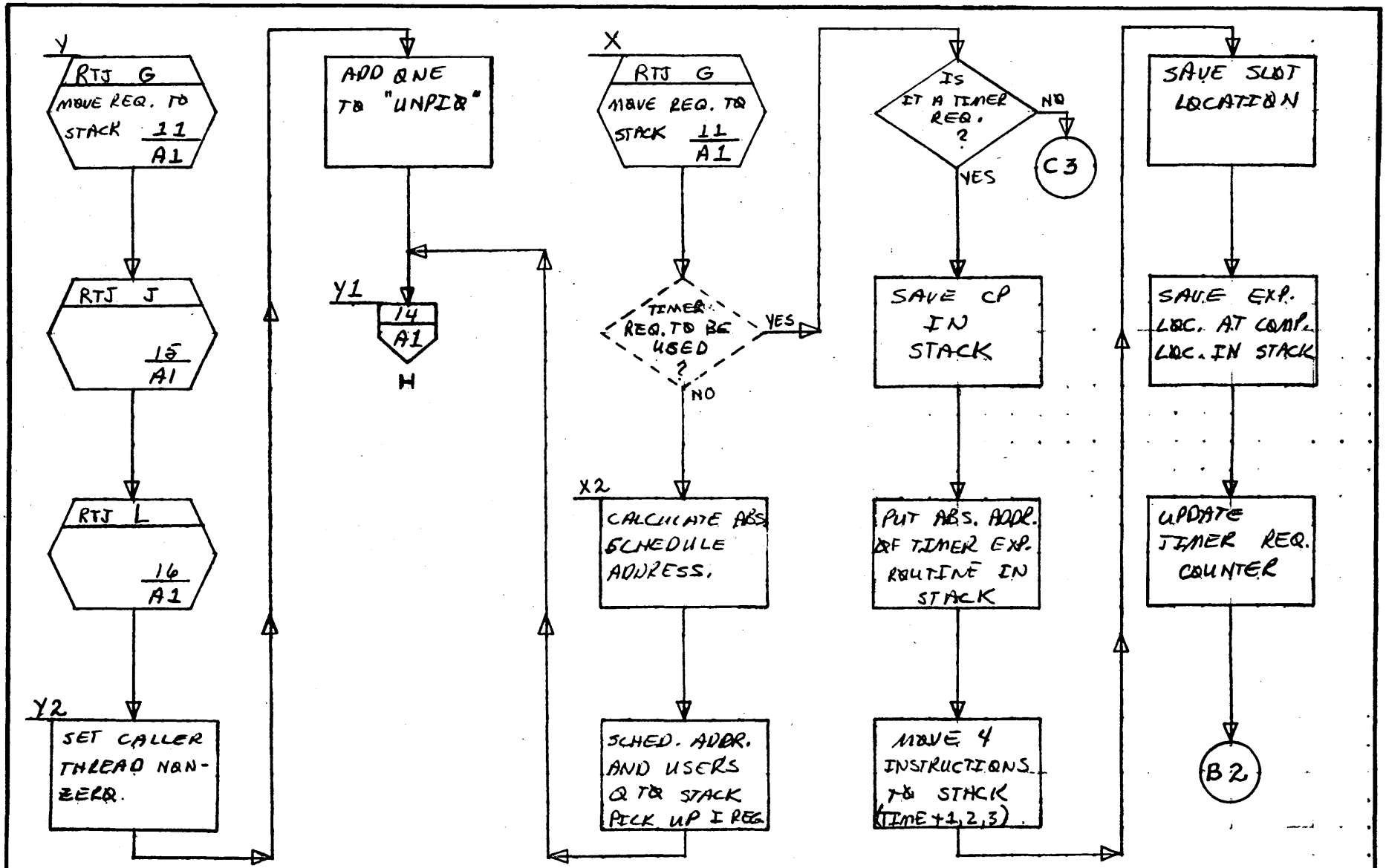
29.23



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: PROTEC - PROTECT		PROJECT MGR.		
	PROCESSOR	PAGE 7 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

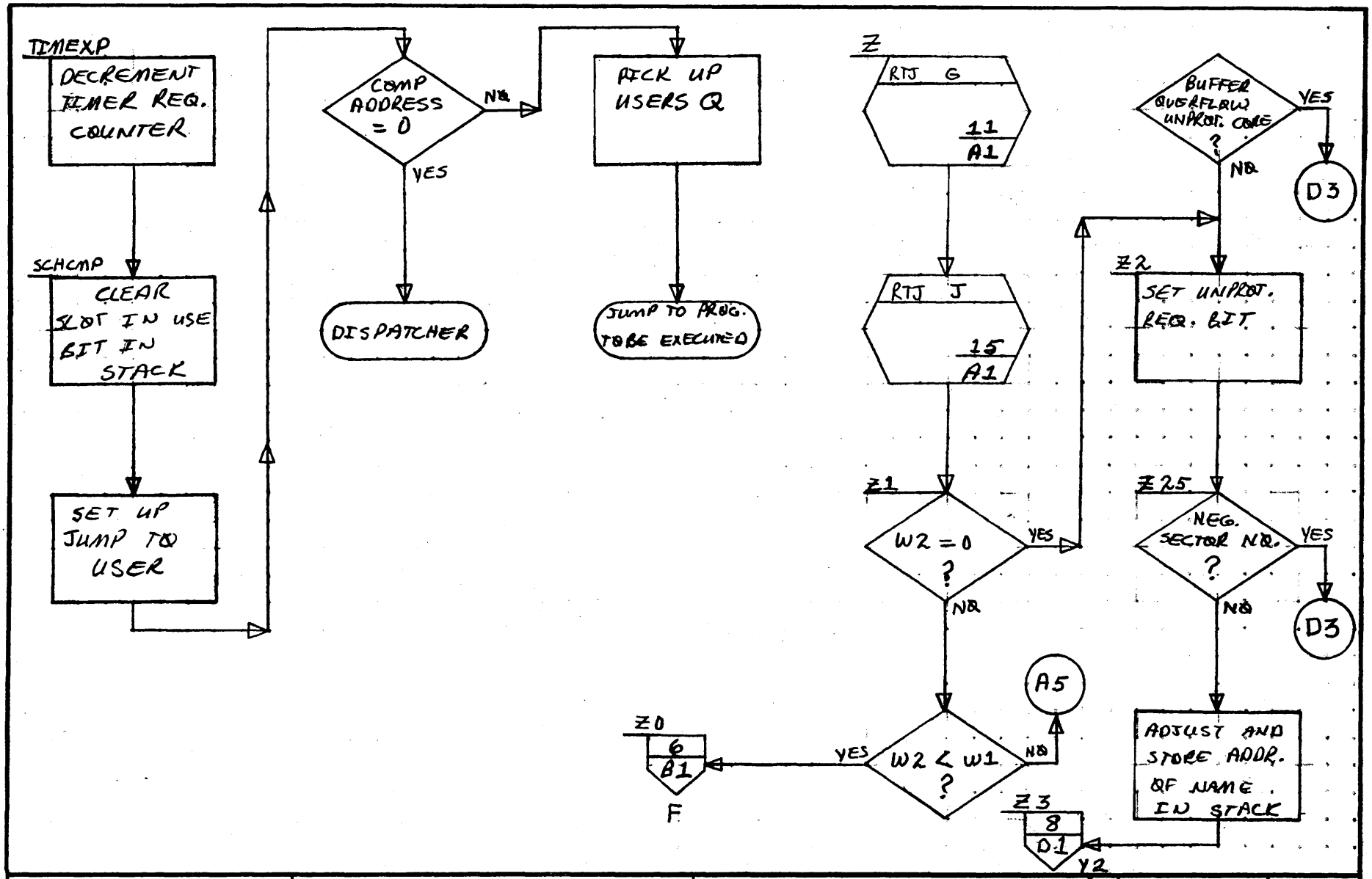
29.24



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC - PROTECT PROCESSOR	PAGE 8 OF 18	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

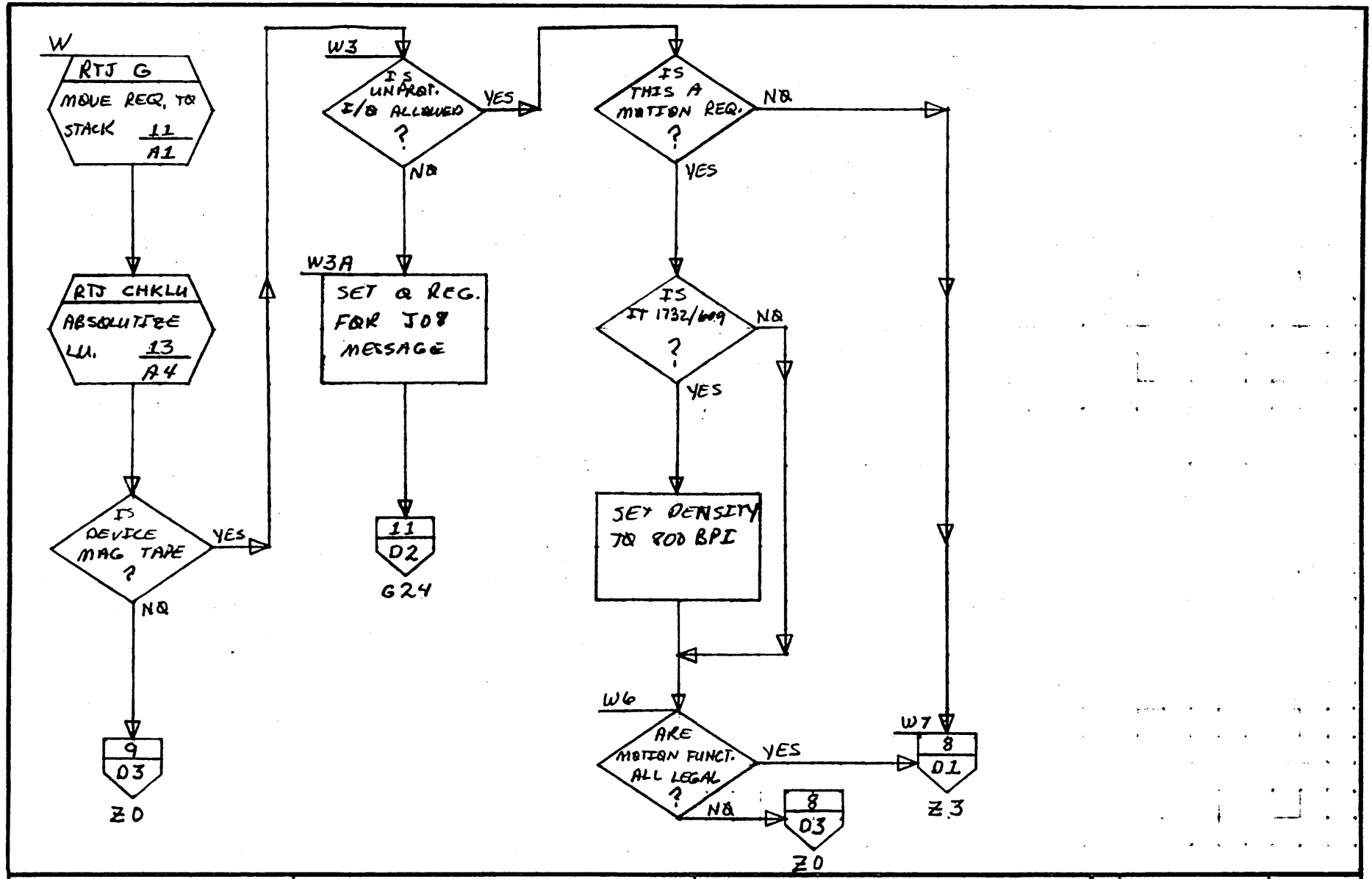
29.25



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.		
	PROCESSOR	PAGE 9 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29.24



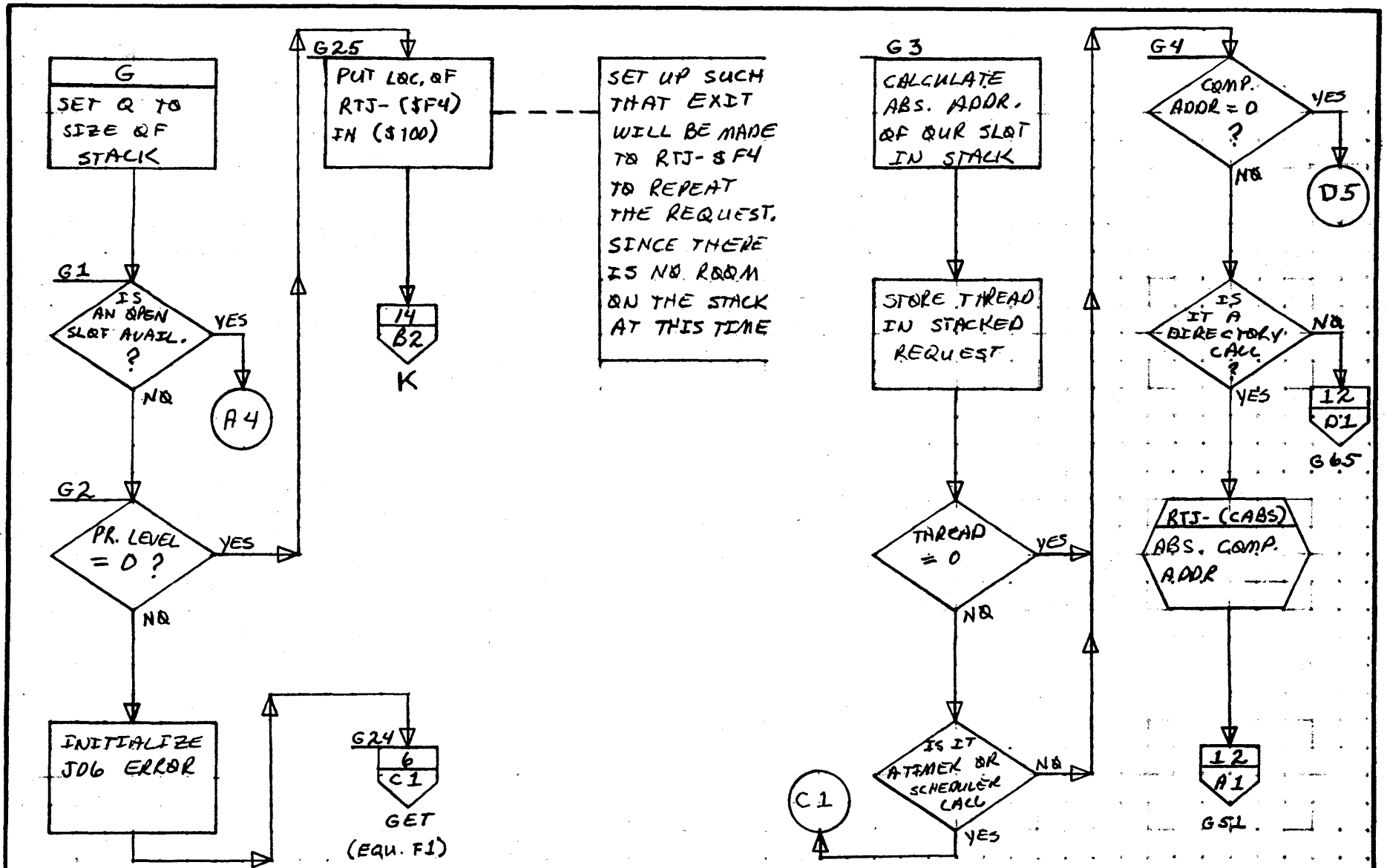
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR		
	PROCESSOR	PAGE 10 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29.27

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



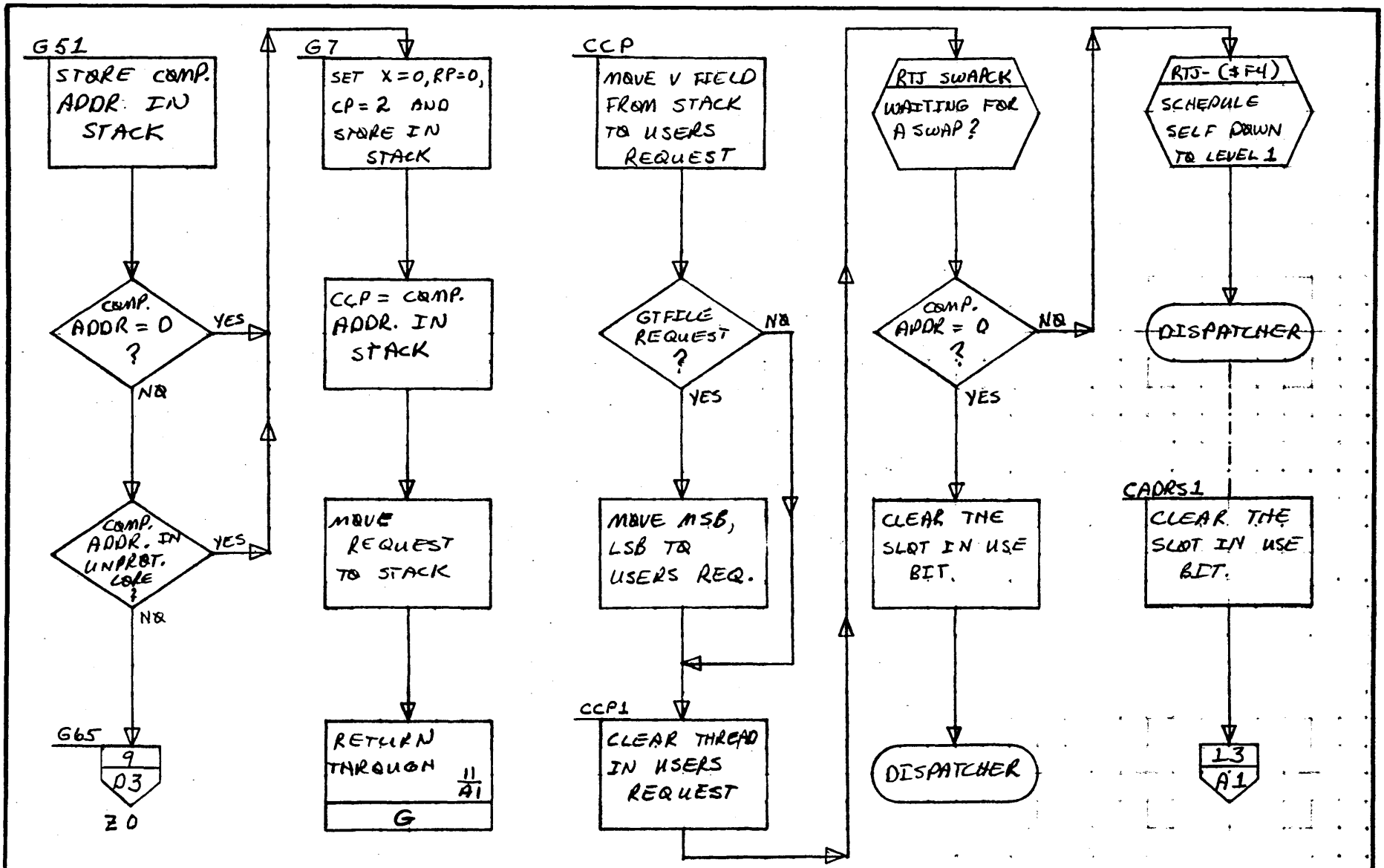
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.		
	PROCESSOR	PAGE 11 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29-28

AA1385 (FORMERLY CA127-1)

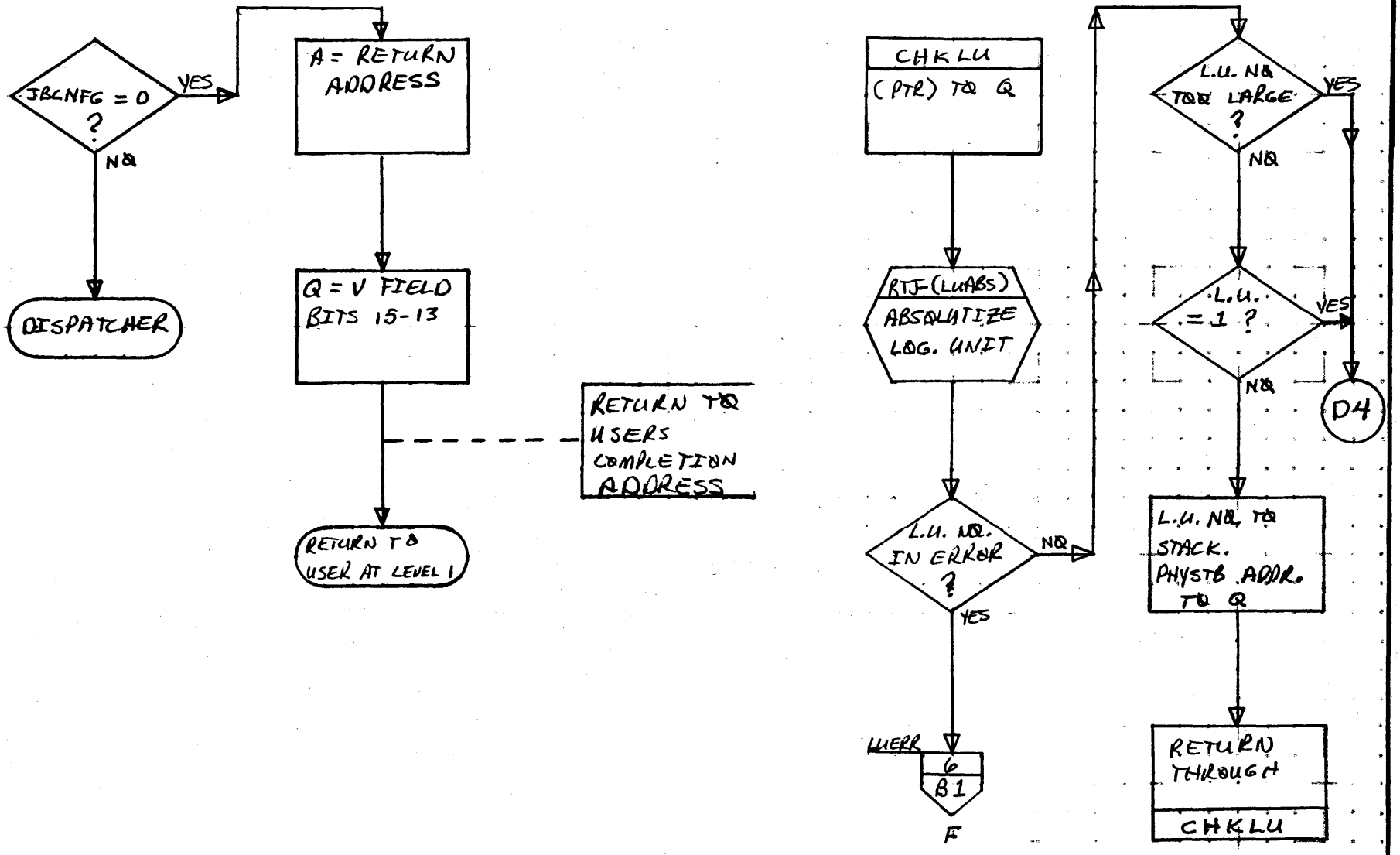
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PRQTEC - PROTECT		PROJECT MGR.			
FLOWCHART ☐	PROCESSOR	PAGE 12 OF 18	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

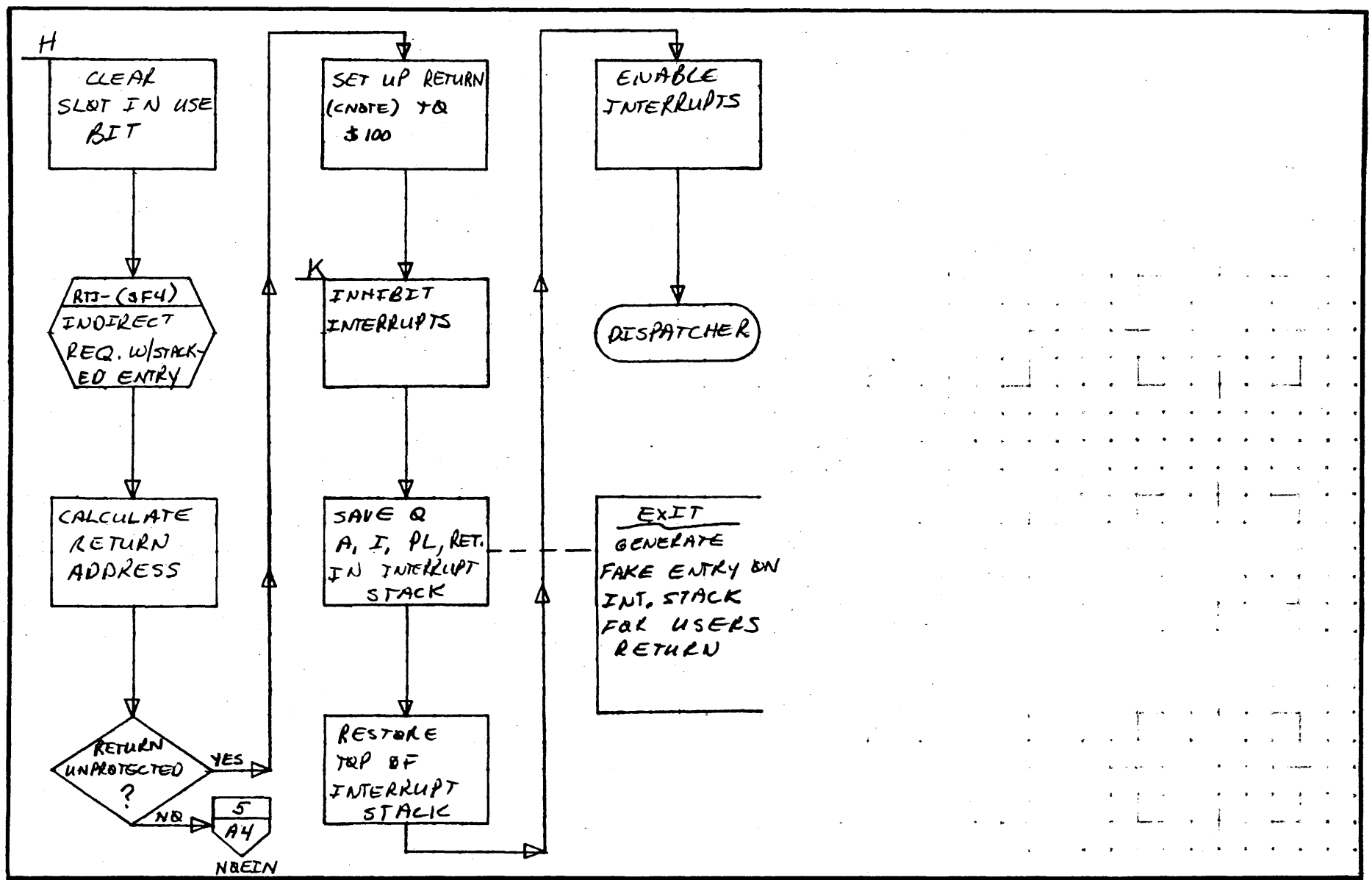
29.2



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE PROTEC - PROTECT PROCESSOR PAGE 13 OF 18 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

MAR 5 1971

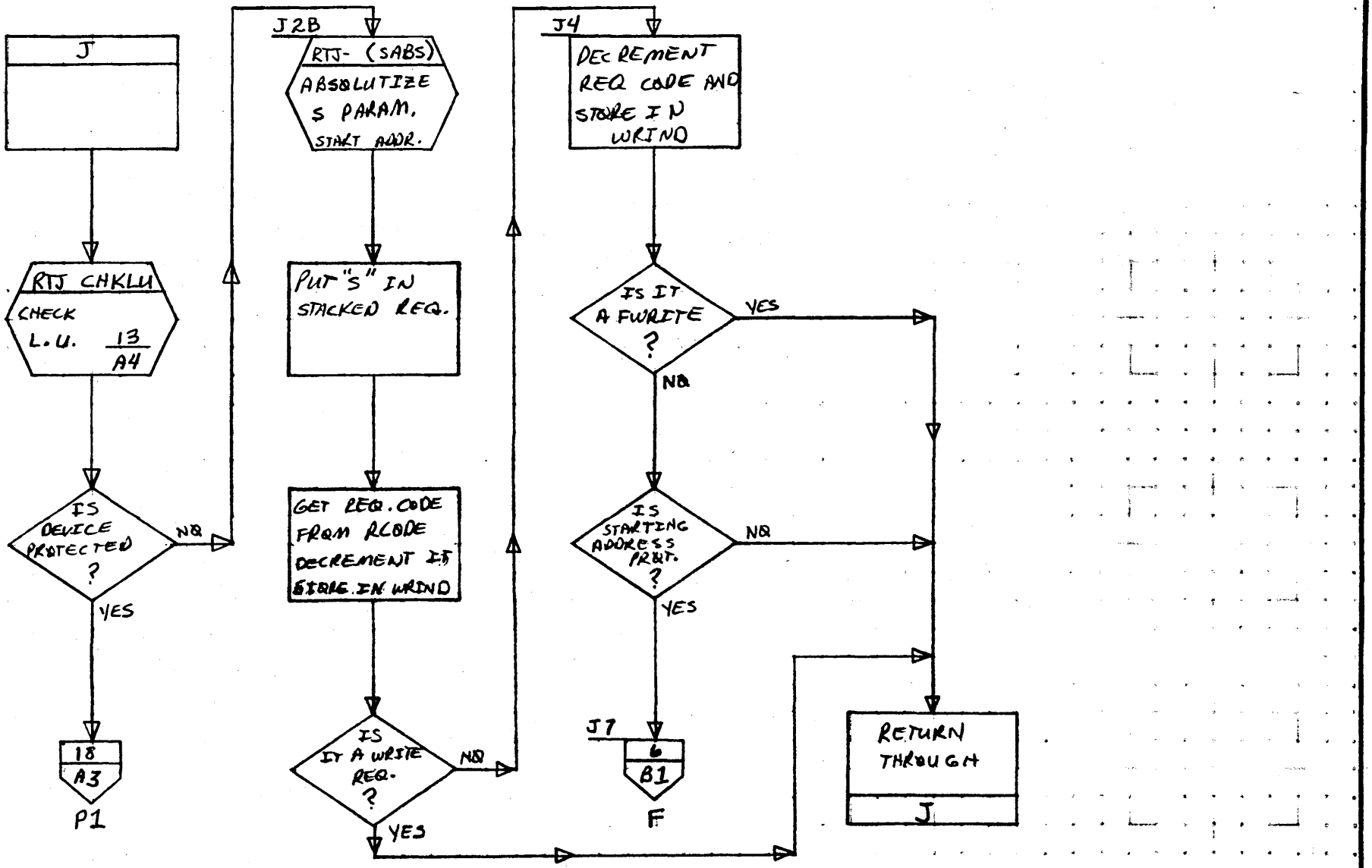
29.30



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS I/IS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC ~ PROTECT		PROJECT MGR.		
	PROCESSOR	PAGE 14 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29.31



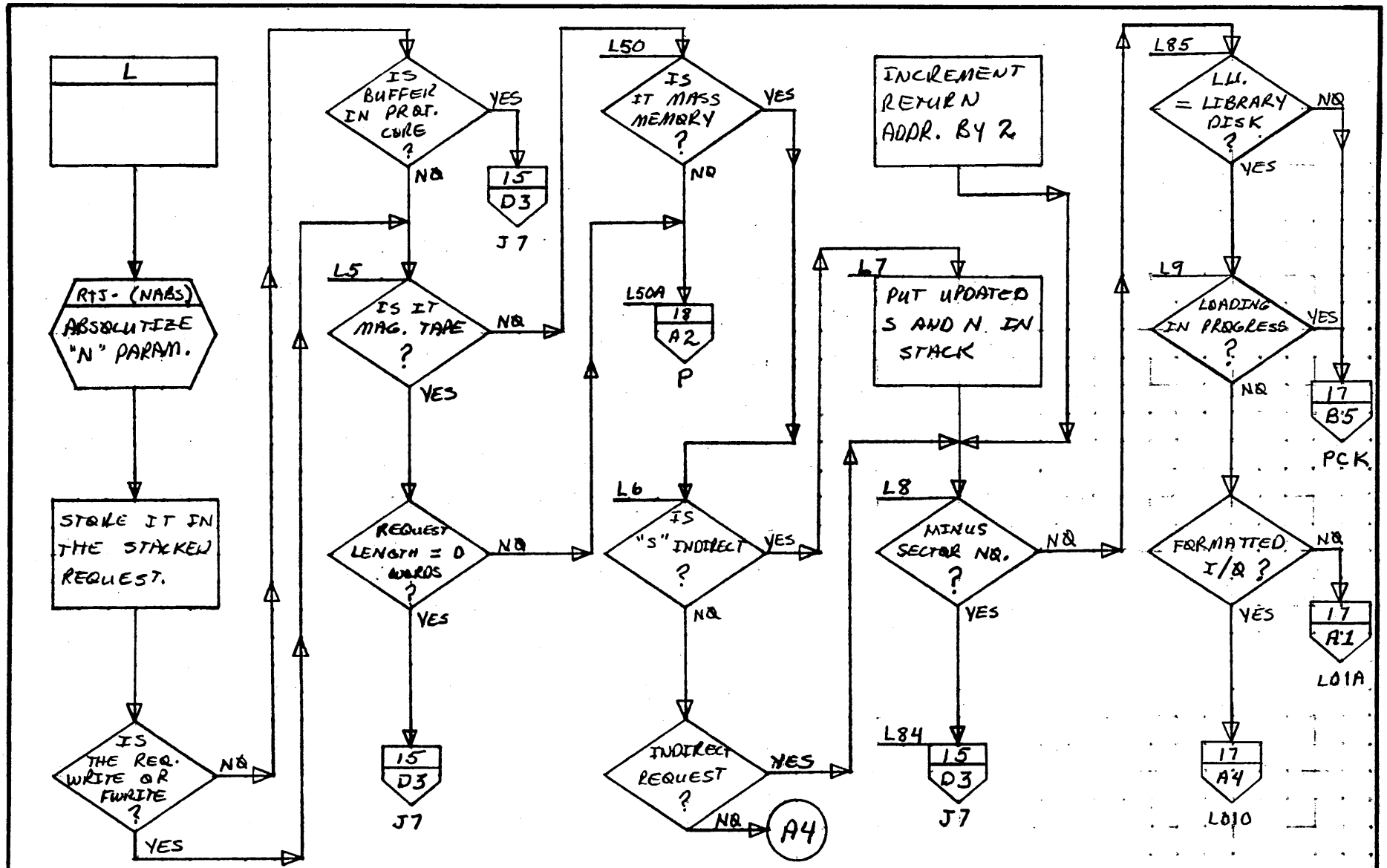
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.		
	PROCESSOR.	PAGE 15 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29.32

AA1385 (FORMERLY CA127-1)

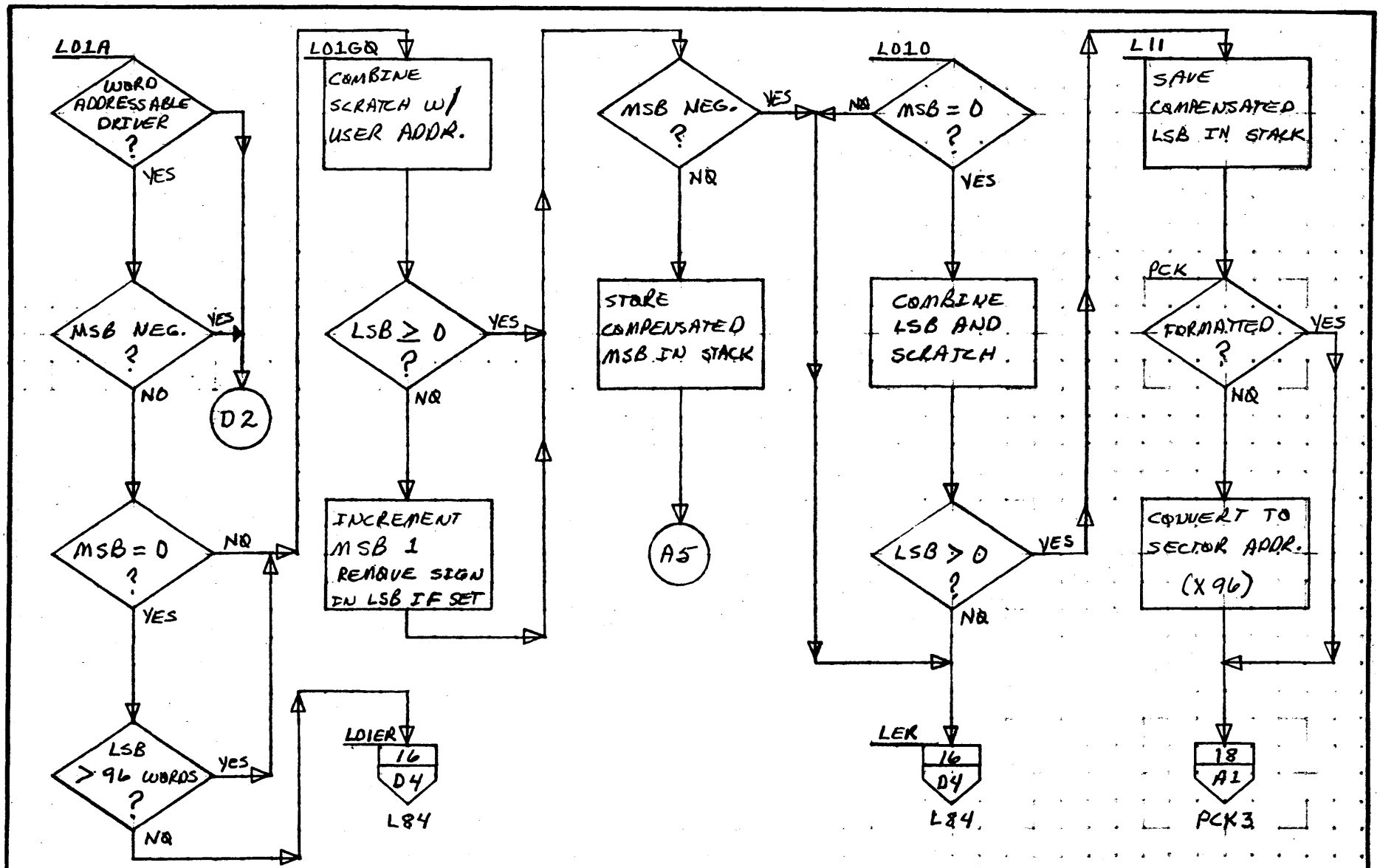
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IAIS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.		
	PROCESSOR	PAGE 16 OF 18	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

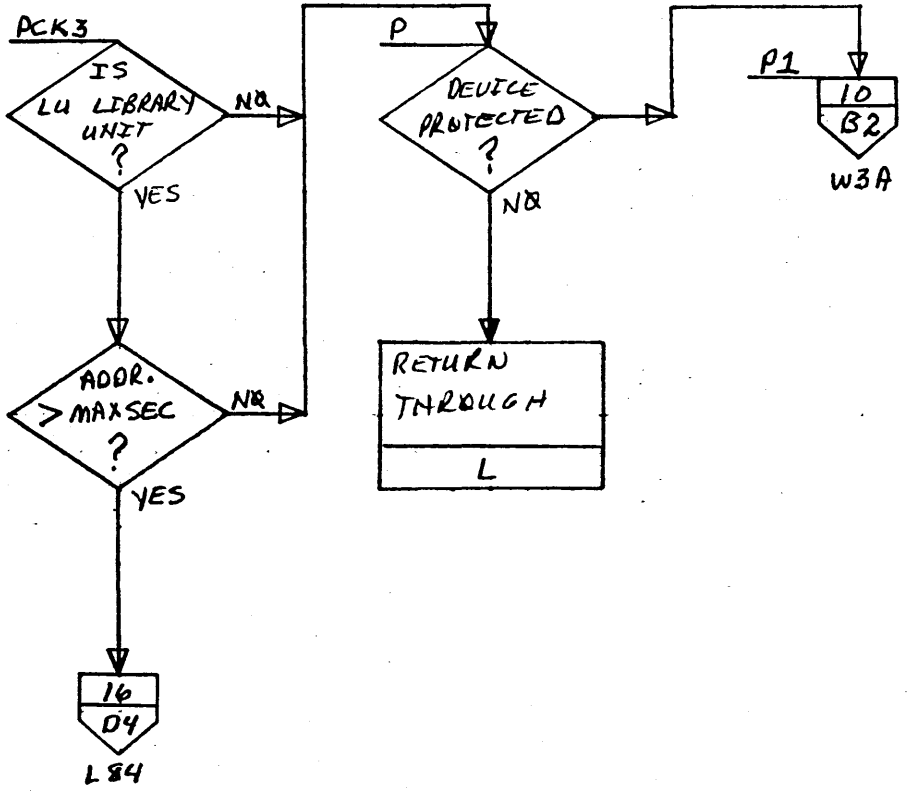
29.33



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PROTEC — PROTECT		PROJECT MGR.			
	PROCESSOR	PAGE 17 OF 18	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

29.34



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE PROTEC - PROTECT		PROJECT MGR.		
FLOWCHART ☐	PROCESSOR	PAGE 18 OF 18	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

29.35

30.0 JBKILL JOB KILL MODULE

30.1 FUNCTION

This module is used by the Job Processor to terminate processing and prepare for the running of the next job.

30.2 ENTRY POINT NAMES

JKILL

30.3 EXTERNALS AND DESCRIPTION

INTSTK	Location of Interrupt Stack {SYSBUF}
UNPTIM	Contains number of unprotected timer requests waiting. {TRVEC}
VRESET	*V Reset Flag {TRVEC}
FILE3	Contains address of PROTEC when JBKILL is in core {TRVEC}
IUP	Pointer to comment device {TRVEC}
LOADIN	Loader in core flag {TRVEC}
UNPIO	Number of unprotected I/O requests waiting {TRVEC}
RECOV	Abs. Addr. of RECOVR routine in JOBENT {TRVEC}
JBCNFG	JOB Cancel Flag {TRVEC}
TRNVEC	Contains address of TRNTBL buffer in JOBENT {TRVEC}
JBPROE	Contains Address of JBPRO routine in JOBENT {TRVEC}
JKIN	JBKILL in core flag {TRVEC}

30.4 ENTRY INTERFACES

This routine is entered from the Protect Processor when JBKILL is requested.

30.5 EXTERNAL INTERFACES

Normal exit from JBKILL is to JOBENT where JOBPRO is scheduled with an index to the RF3 routine within JOBPRO. If the Recovery Indicator {RI} is set, exit is to the RECOVR routine in JOBENT where the RCOVER module is scheduled.

30.6 GENERAL PROGRAM INFORMATION

JBKILL is brought into core only when the JBCNFG {Job Cancel Flag} is set. TRVEC checks this flag and if set, sets @ negative and schedules PROTEC. PROTEC determines the JBKILL is requested and reads its error processing routine and JBKILL into an overlay area within PROTEC. A jump is then made to the area to begin execution.

30.7 GENERAL DESIGN PECULIARITIES

JBKILL must be installed on mass storage immediately after the Protect Processor in order for it to be read in properly by PROTEC.

30.8 PROGRAM LOGIC

Whenever an unrecoverable error is detected in an unprotected program, this module is scheduled. The following errors cause this to happen:

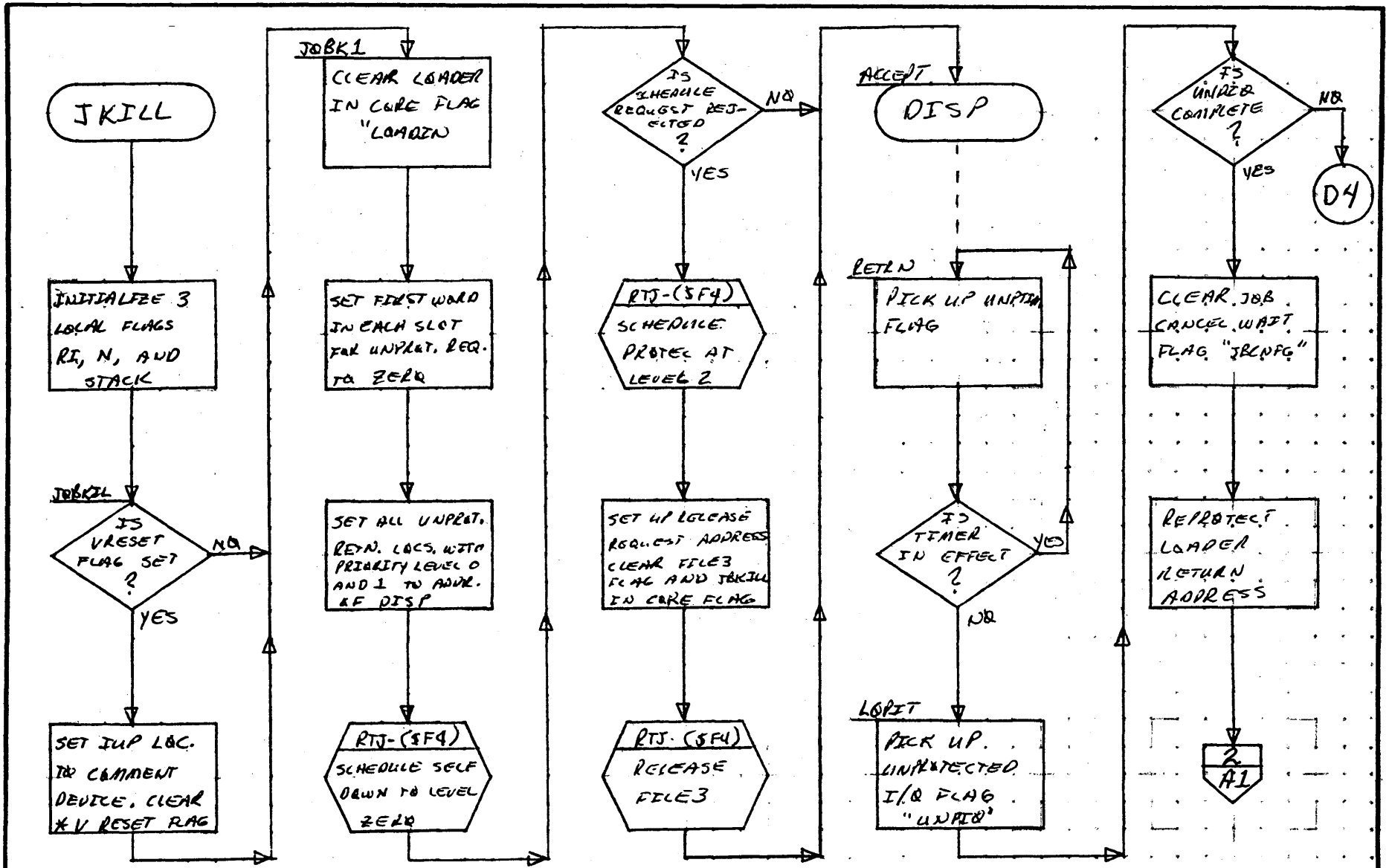
1. Illegal I/O format
2. Protect violation
3. Illegal parameter in a GTFILE, LOADER, CORE or STATUS request.
4. No transfer address when requested to execute a program.

The address JBCNCL in TRVEC is scheduled at level 2 by the PROTEC Processor or the JPLOAD module. JBCNCL in turn schedules PROTEC which reads in JBKILL. The following events occur when JBKILL is executed.

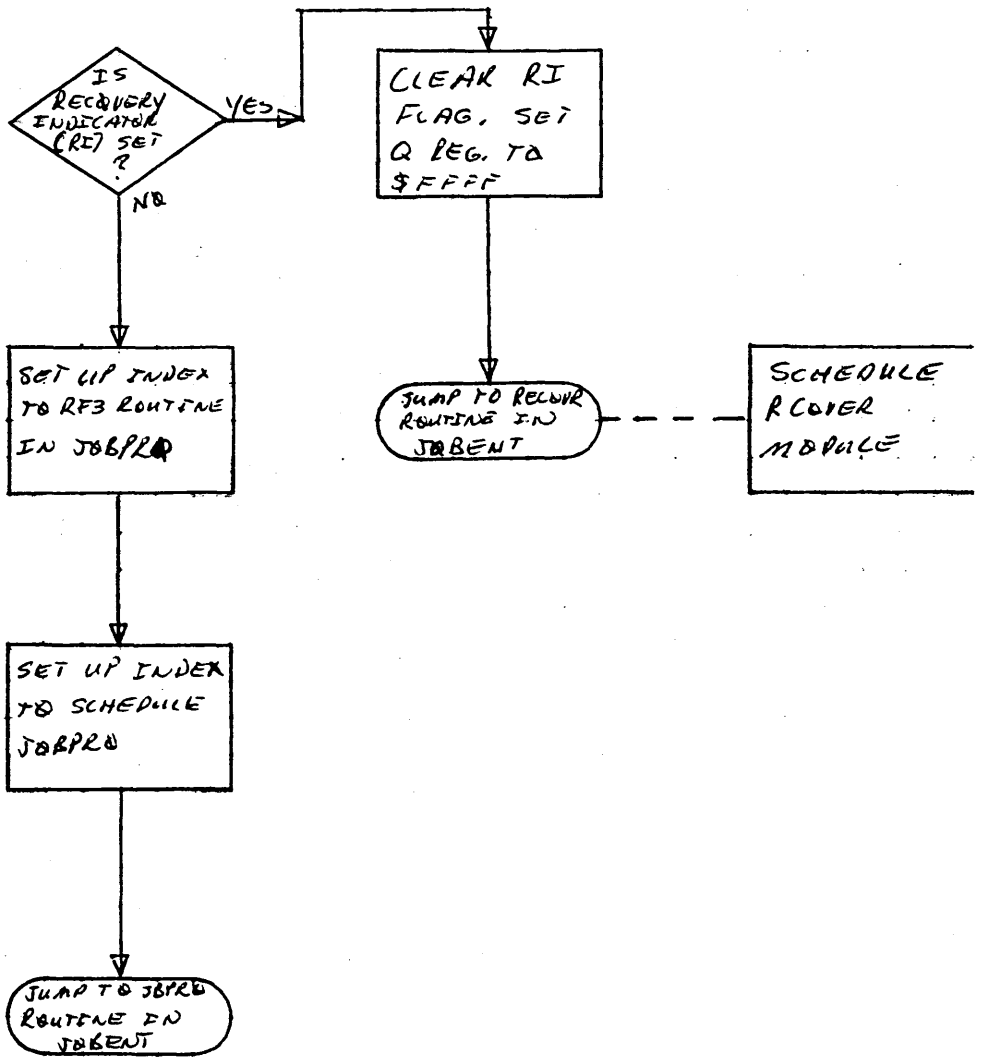
1. The Job Processor Loader in Core Flag {LOADIN} is set to zero, the *V flag {VRESET} is cleared if set and IUP is set to the comment medium.
2. The first word of each slot in the unprotected request slot in the Protect Processor is set to zero.
3. All interrupted stack entries referring to unprotected return locations, level zero and one priority are set to the address of the Dispatcher, except for request EXIT locations.
4. Schedule self down to level zero.
5. If the recovery switch {RI} is set, a jump to JOBENT to schedule the System Recovery module {RCOVER} is made after clearing the RI switch. If not, a jump is made to the JBPRO routine in JOBENT to schedule the JOBPRO module with an index set to the RF3 routine within JOBPRO.

30.9 SUBROUTINE LOGIC

NONE



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JBKILL - JOB KILL		PROJECT MGR.		
	PROCESSOR	PAGE 1 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE JBKILL – JOB KILL		PROJECT MGR		
FLOWCHART ☐	PROCESSOR	PAGE 2 OF 2	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

3.04

31.0 JPLOAD

31.1 FUNCTION

This module contains the Processing for all Loading Type Job Processor requests and does the scheduling of the Protect Processor if not in core. JOBENT schedules JPLOAD at priority level zero which allows these requests to be interrupted by other Job Processing Type statements. The statements processed in this module are *L, *P, *X, and * entry point.

31.2 ENTRY POINT NAMES

JBL - JPLOAD entry point
JP1 - Return from scheduling Protect Processor.

31.3 EXTERNALS AND DESCRIPTION

TRNVEC {TRVEC}	Abs. Addr. of TRNTBL buffer in JOBENT.
PRORET {TRVEC}	Return Addr. for PROTEC
JBPROE {TRVEC}	Abs. Addr. of JBPRO routine in JOBENT.
RELS1A {TRVEC}	Abs. Addr. of RELF routine in JOBENT.
PROTEC	Mass Resident Protect Rocessor.
LOG1A {SYSBUF}	Entry point to the LOG1A table in SYSBUF.
MIB {MINT}	Manual Interrupt Busy Switch. Zero = not busy. Non-zero = busy. Must be set to zero before another manual interrupt can be processed.
LIBET {TRVEC}	Holds the address of the routine in JOBENT that schedules LIBEDT.
JBCNCL {TRVEC}	Entry point to the routine that schedules PROTEC.
JBCNFG {TRVEC}	Job Cancel Flag which tells the Protect Processor to stop honoring requests from unprotected core.
TRANV {TRVEC}	Contains the absolute address of the Job Processor transfer table. {TRANTA}
IUP {TRVEC}	Pointer to the comment device. Used by the Job Processor transfer {initially set to $18FD}.
LOADSD	Contains an index to the FWA of the mass memory System Directory entries.
LOADIN {TRVEC}	Flag for the protect processor to allow the loader to read and write below the scratch area on mass storage.

JPRET {TRVEC}	Entry point to the table of preset entry points that allows entry to a protected routine from an unprotected routine.
JPRET1 {TRVEC}	Used to save the return address in T7 or in JPL0AD when either module calls the loader.
JPRETN {TRVEC}	Return is made from the loader through JPRETN which has been entered in the preset tables by T7 or JPL0AD.
FILE2 {TRVEC}	Contains the absolute location of JPL0AD as associated with this module.
FILE3 {TRVEC}	Contains the Absolute location of the Protect Processor when in core.
L0G1 {SYSBUF}	FWA of L0G1 table. Contains the largest legal Logical Unit Number.
BRKPT	Contains an index to the FWA of the mass memory System Directory entry for the Break Point Program.
UNPI0 {TRVEC}	Contains the number of I/0 calls pending.
SWAPCK {DRC0RE}	Entry point to the SWAPCK routine in DRC0RE. Used to decrement UNPI0 when unprotected I/0 is completed.

31.4 ENTRY INTERFACES

JBL - Entry to JPL0AD module is scheduled by J0BENT module.
JP1 - Entry to JP1 is made after scheduling in the Protect Processor.

31.5 EXTERNAL INTERFACES

The normal Exit from JPL0AD is to J0BENT where J0BPR0 is scheduled and executed according to the parameters passed from JPL0AD. If an error on loading was made, exit is to J0BENT to release FILE2 and jump to the Dispatcher awaiting J0B KILL execution.

31.6 GENERAL PROGRAM INFORMATION

31.6.1 Table of Job Processor Loader Type Requests

The following table is used to analyze the loader type statements.

TABl

CONSTANT	FUNCTION
$50FF	P ↵ statement
$4CFF	L ↵ statement
$58FF	X ↵ statement
$4C2C	L, statement
$582C	X, statement

31.6.2 Table of Job Processor Input Statement Subroutines

The following table is used in indexing to the loader subroutines as determined by the input statement.

TABLOC

Relative Address	Statement
PREJOB	*P ↵ statement
RBLOAD	*L ↵ statement
EXECUT	*L ↵ statement
RBLOAD	*L, statement
EXECUT	*X, statement
TRLOAD	*entry point state

31.6.3 Table of Absolute Locations in JOBPRO

The following table is transferred from JOBENT and is used to access its locations.

TRANTA - Abs. location of JOBTWO
Relative loc. of JOBP3 to JOBTWO
INPBUF - Absolute address of input buffer
Relative loc. of JOBP to JOBTWO
Relative loc. of JO4 to JOBTWO
Relative loc. of JO3 to JOBTWO
Lock out Flag - LO
BPS - Break-point load flag - BPS
RI - Recovery Indicator - RI
LOADEP - Loc. of loader entry point is present
@REG - Request no.
STCK - Abs. loc. of stack
NN - N Value - Number of stacked requests

31.7 GENERAL DESIGN PECULIARITIES

31.7.1 JPLOAD has its own decimal ASCII to HEX (ASCHEX) conversion routine. This routine is entered from the RBLOAD subroutine. Before entering ASCHEX the address +1 of the input buffer is multiplied by two. The product is used by ASCHEX to check for an input of two numbers.

31.7.2 LOADSD contains an index to FWA of Mass Memory System Directory.

31.8 PROGRAM LOGIC

Upon entrance to JPLOAD its FWA is saved in FILE2 in TRVEC. A check is then made to see if the Protect Processor is in core. If not, PROTEC is scheduled into core. Execution resumes at JP1 in either case. The locations from the transfer table (TRNTBL) in JOBENT are then transferred to JPLOAD.

The loader type input statements are then analyzed to determine which subroutine is to process the statement. A return jump is then made to the proper subroutine.

The input statements and the subroutines that processes them are as follows: *P cr to PREJOB, *L cr to RBLOAD, *X cr to EXECUT, *L, n cr to RBLOAD, *X,N cr to EXECUT and * entry point cr to TRLOAD.

The LOADER subroutine jumps to and is returned from the loader.

The ASCHEX routine is entered from RBLOAD to convert from decimal ASCII to HEX, the logical unit specified in the *L, n cr statement.

The exits from this program are to JOBENT where the proper module is scheduled according to the parameter passed from JPLOAD.

31.9 SUBROUTINE LOGIC

31.9.1 TRLOAD *entry point statement

Upon entry to TRLOAD a check is made to determine if the job processor statement is *LIBEDT. If true a jump is made to LIB in JOBENT to schedule LIBEDT, otherwise, pick up the address of the input buffer and return jump to LOADLR to load the program. If an error on loading was made schedule JBCNCL at level 2 and jump to JOBENT to release FILE2. If no error, save the transfer location, clear the address of the breakpoint program ($F3), pick up and save the address of the transfer vector table and jump to EX8 in the EXECUT subroutine to execute the job.

31.9.2 RBLOAD *L,N cr load relocatable binary program

This subroutine instructs the loader to load relocatable binary information. Once initiated, the loader continues loading until it reads an EOL block or a system control statement.

N specifies the logical unit number for the loading device, but if not specified, *L, the standard input device is used. The given logical unit number is checked for being negative or larger than the largest Logical Unit Number. If Logical Unit Number is in error, the 04, statement message index is set up and a jump is made to J0BENT. If logical unit specified is zero, the standard input device is used. The loader keeps track of the upper and lower limits of available core, and adjusts limits according to the amount of core allocated during input.

The general routine "LOADLR" is called to communicate between loader and statement processor. A and Q registers are set up for the specific type load operation and the "LOADLR" routine is returnjumped to. It protects the return location and stores the address of that address in the communication region {$EE} and the first preset entry {JPRET}. After returning from the loader, that location is protected again and interrupts are immediately enabled. The return flags are checked for any recoverable error, Job Processor statement read, or a successful operation. The index to RF3 in J0BPR0 is set up and a jump to J0BENT is made.

31.9.3 EXECUT *X,N execute the loaded program

This subroutine instructs the job processor to begin program execution; it must be used after an *L statement. If N is blank, the loader is directed to product a memory map after loading. If N is not blank, no memory map will be produced.

Upon entrance, a subroutine load is requested which detects any unpatched externals and searches the program library directory for a matching name. After patching all externals possible, the loader returns to "EXECUT". A check is made for irrecoverable errors; if any are found, control is transferred to J0BPR0 to read the next control statement.

A check is made to see if a map is desired. If so, it is requested. The I register is checked to see if the program has been loaded on mass storage; if so, it is brought into core.

The breakpoint switch is then tested. If this switch is zero, a jump is made to J0BENT to schedule JPT13 with the transfer location SAVA in the Q register. If the switch is set, the breakpoint routine is loaded from mass storage and stored in unprotected core. A check is made to see if it will fit into unused unprotected core; if not, a loader type error message is given:

E5

E10

BRKPT

and control is passed to the Job Processor to read a new statement.

After loading the breakpoint routine, its entry location is stored in $F3. The Q register will contain the transfer location of user's program, and a jump is made to J0BENT to schedule JPT13 and begin execution.

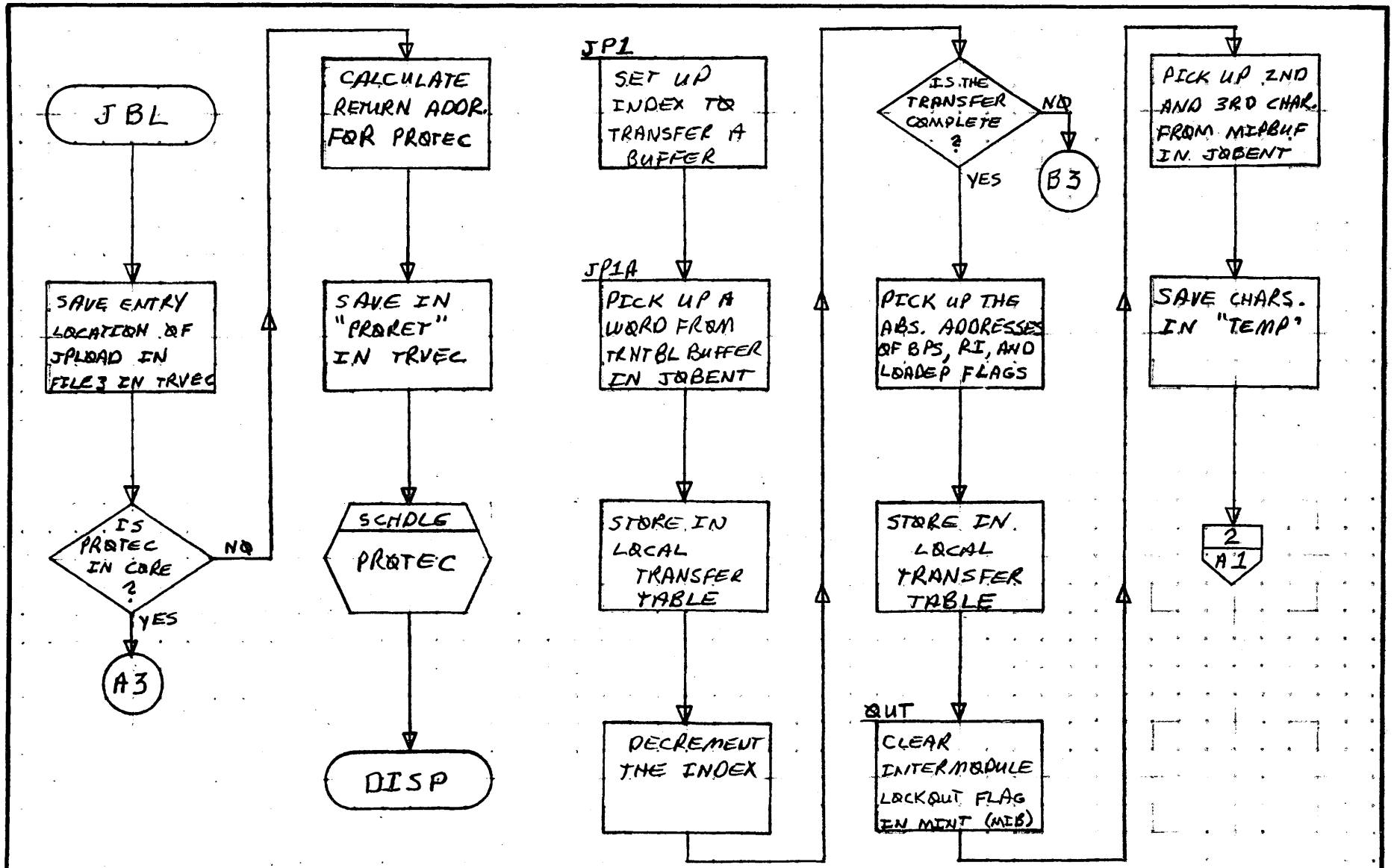
31.9.4 PREJ0B *P

This subroutine handles preload intiialization. That is, temporary upper and lower limits of available core are set to unprotected limits. The relocating loader is read into the highest unprotected core available. The entry location to the loader is stored in "L0ADEP" which is used by other routines to check if the loader is in core.

The *P statement must be used prior to any loading operation that is to be independent of previous loading operation.

31.9.5 ASCHEX Decimal ASCII to HEX conversions routine

This routine is entered from the RBL0AD subroutine. It converts the decimal ASCII Logical Unit Number specified in the *L,N statement, to hexidecimal. This routine checks for a legal ASCII numbers between $30 and $39 and also checks if larger than $39. This routine does not do the checking for a valid Logical Unit Number. This is done when control is returned to RBL0AD. If an error is detected, Q is set to $FFFF and control is returned to RBL0AD.



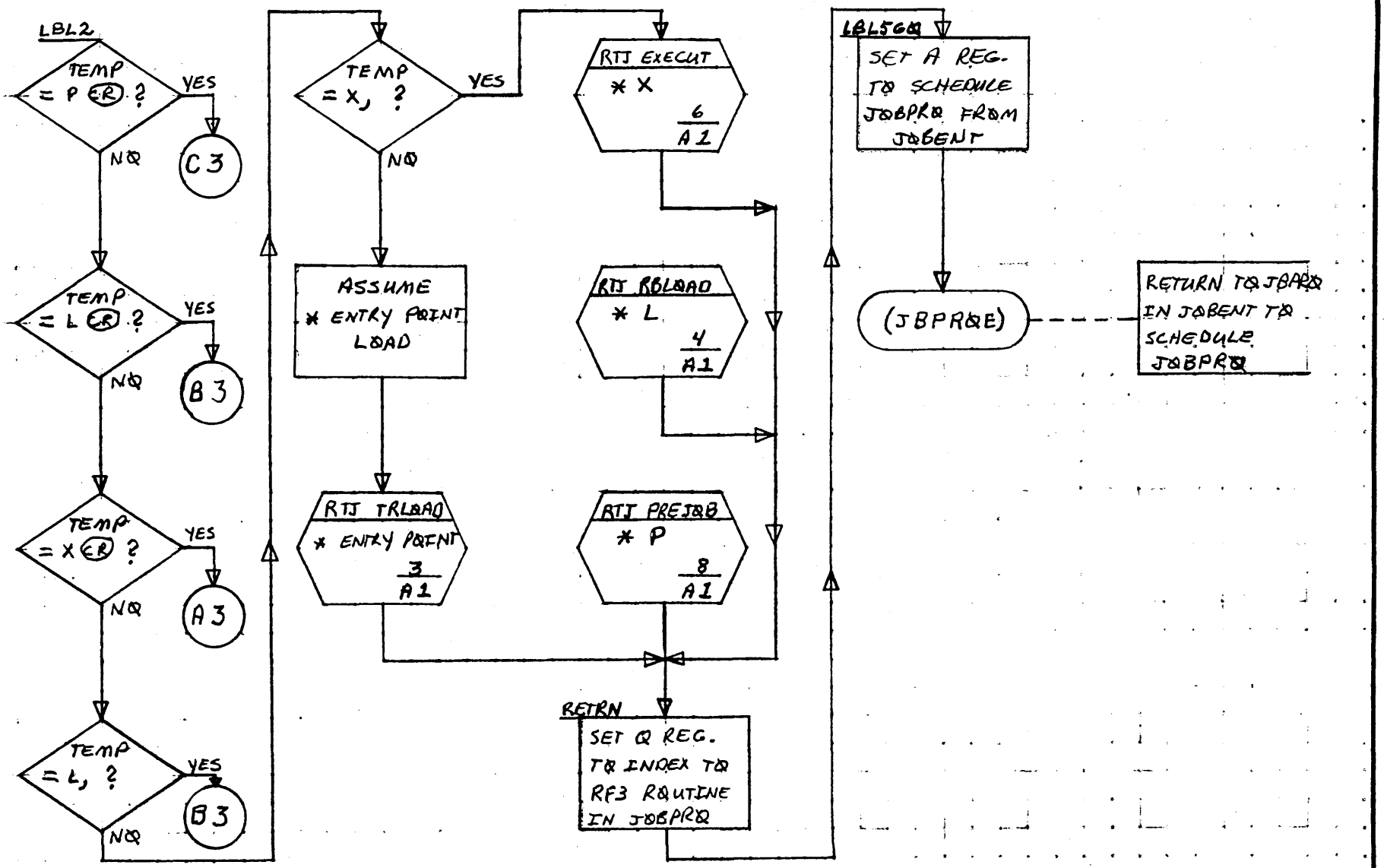
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JPLOAD		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

31.6



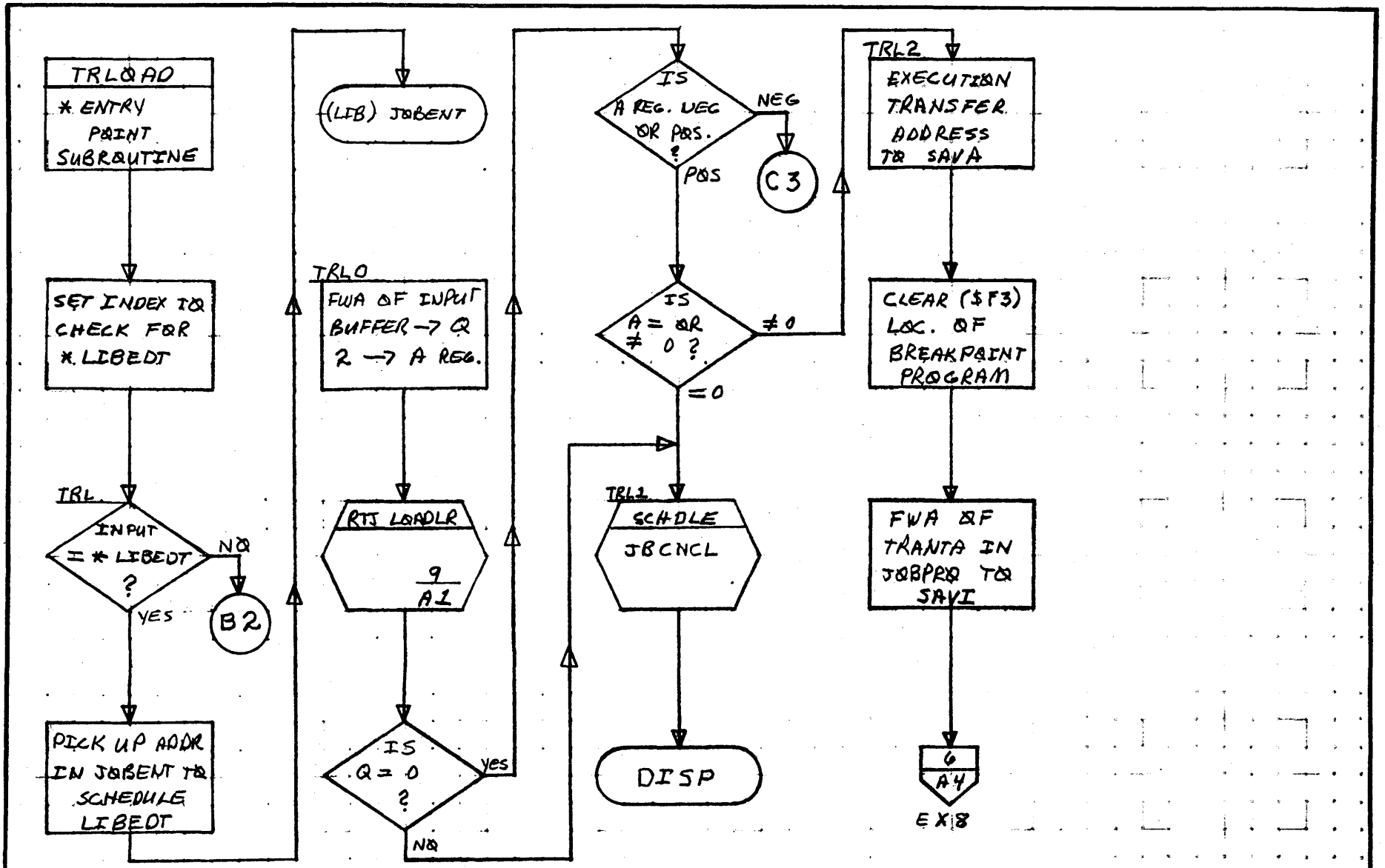
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JPLOAD		PROJECT MGR.		
		PAGE 2 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

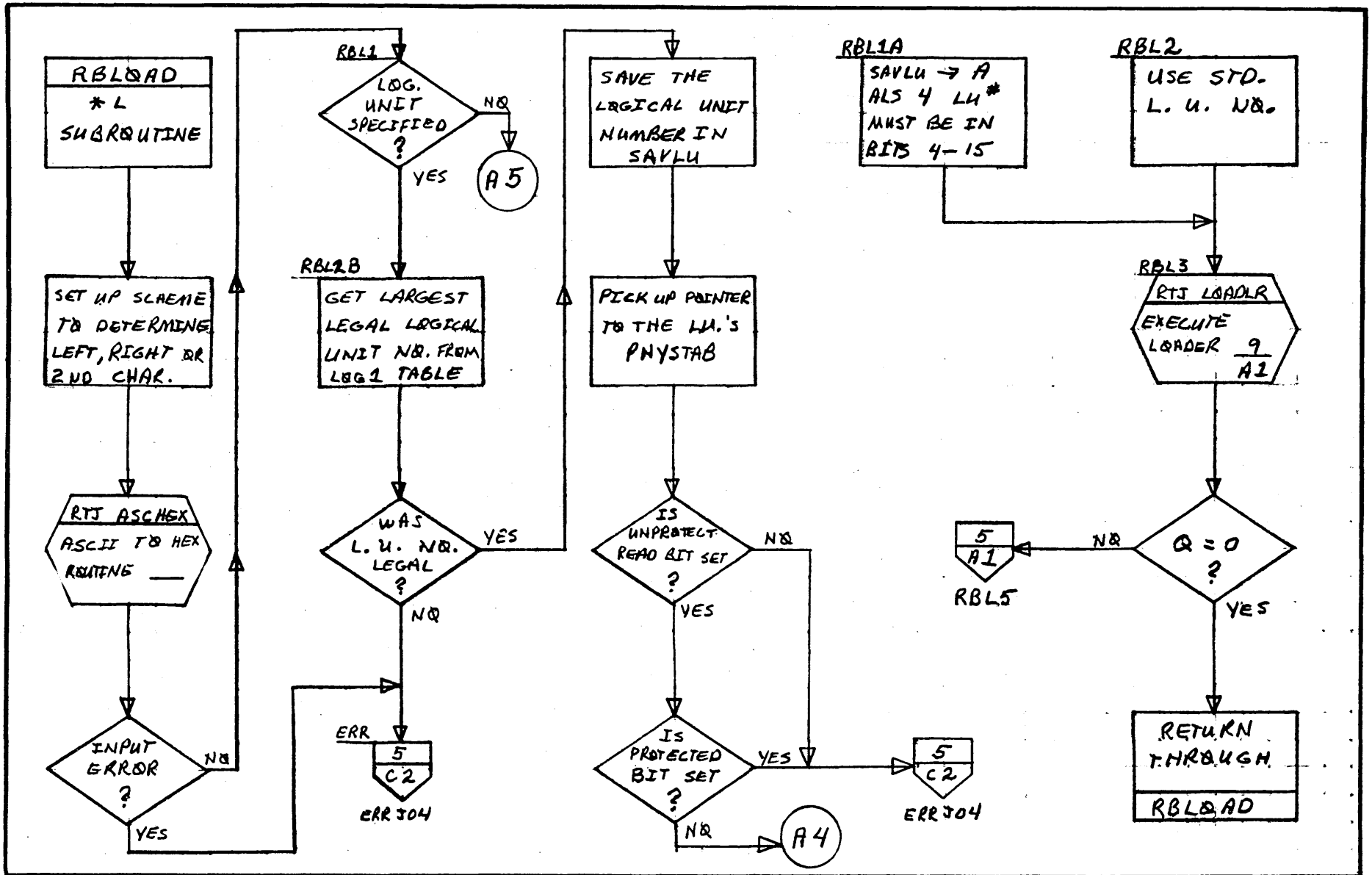
31.7



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE JPLOAD	PROJECT MGR.			
	PAGE 3 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

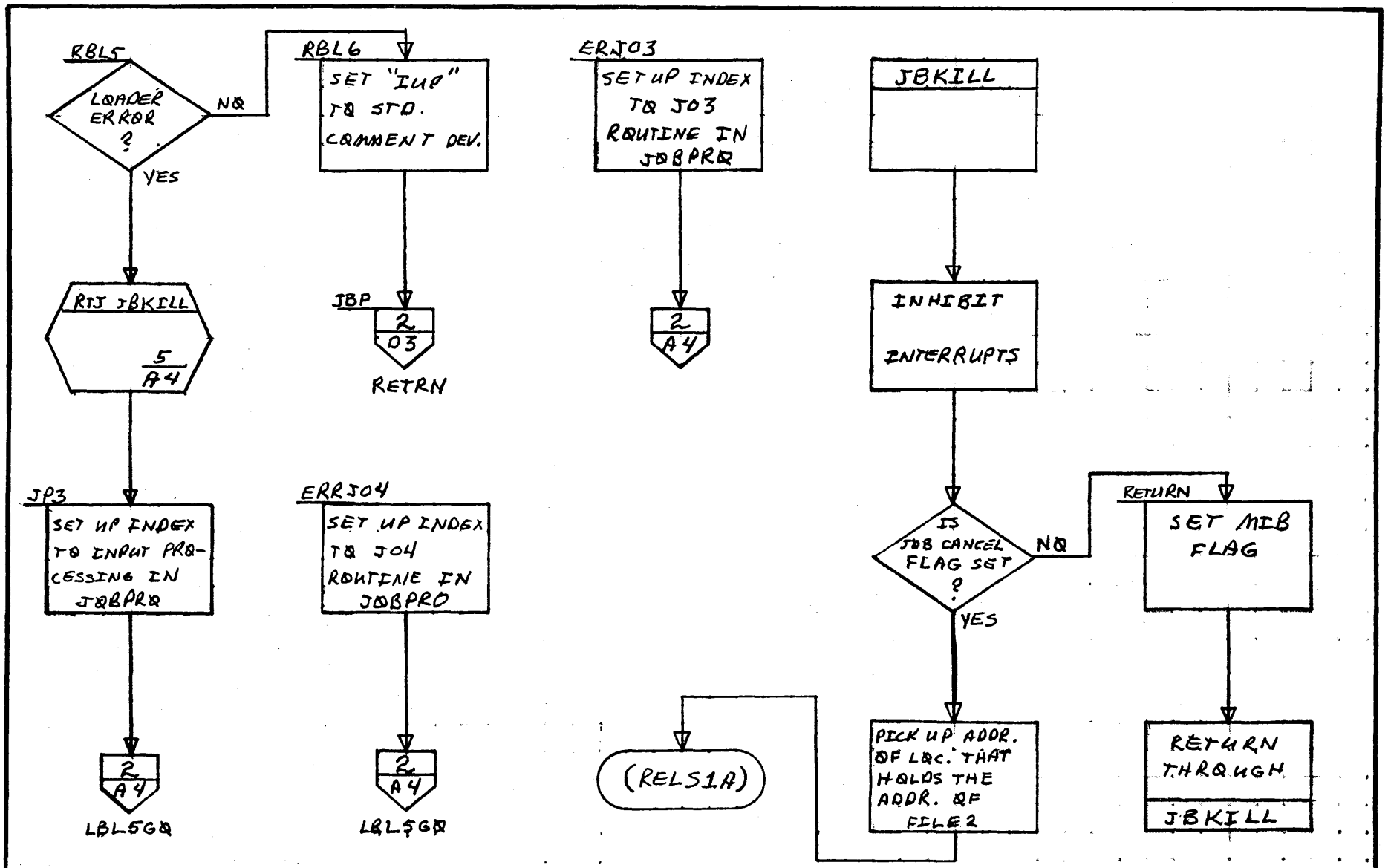
31.8



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE IPLRAD		PROJECT MGR.			
		PAGE 4 OF 11	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

31.5



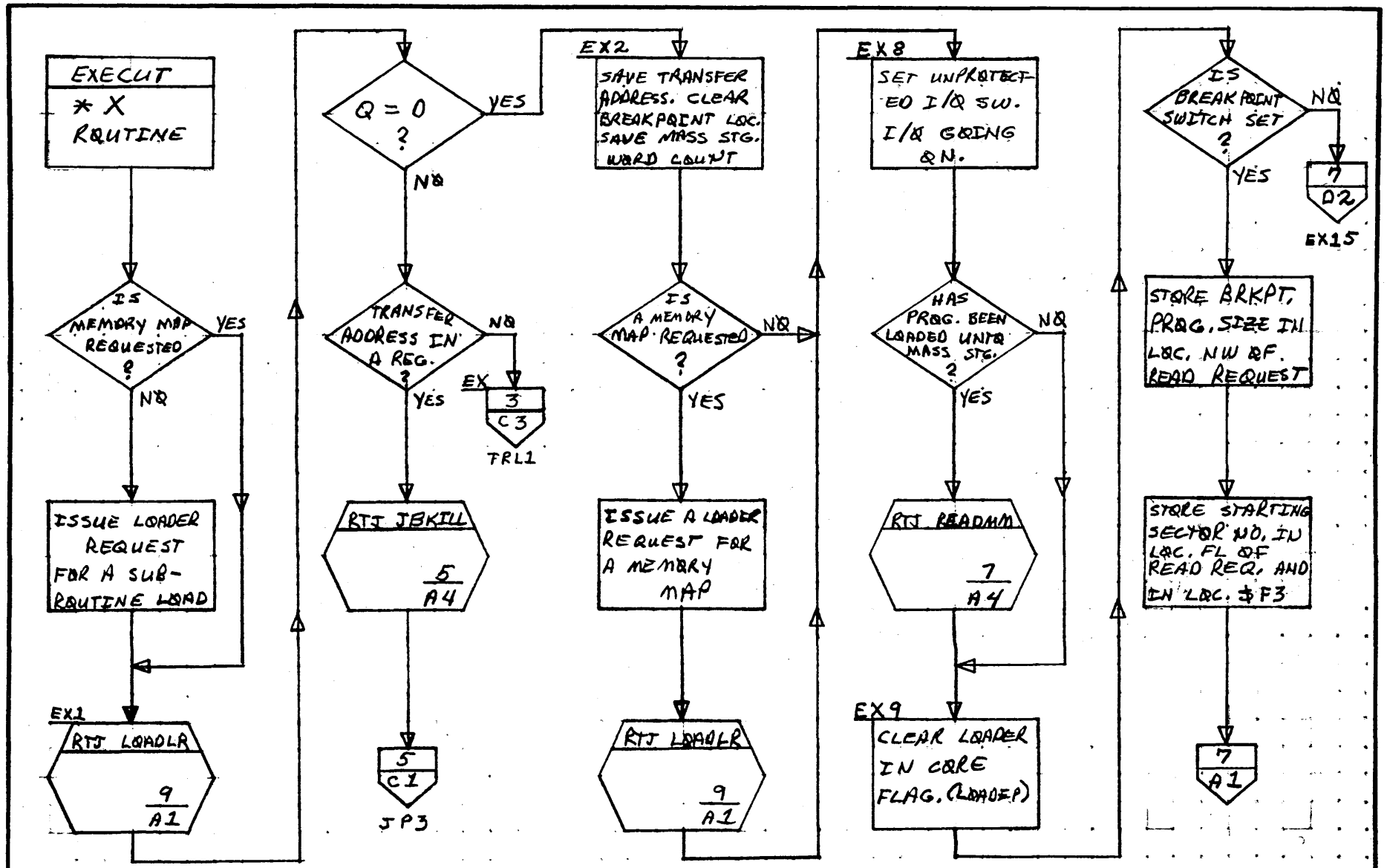
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JPLOAD		PROJECT MGR.			
FLOWCHART ☐		PAGE 5 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

31.10

AA1385 (FORMERLY CA127-1)

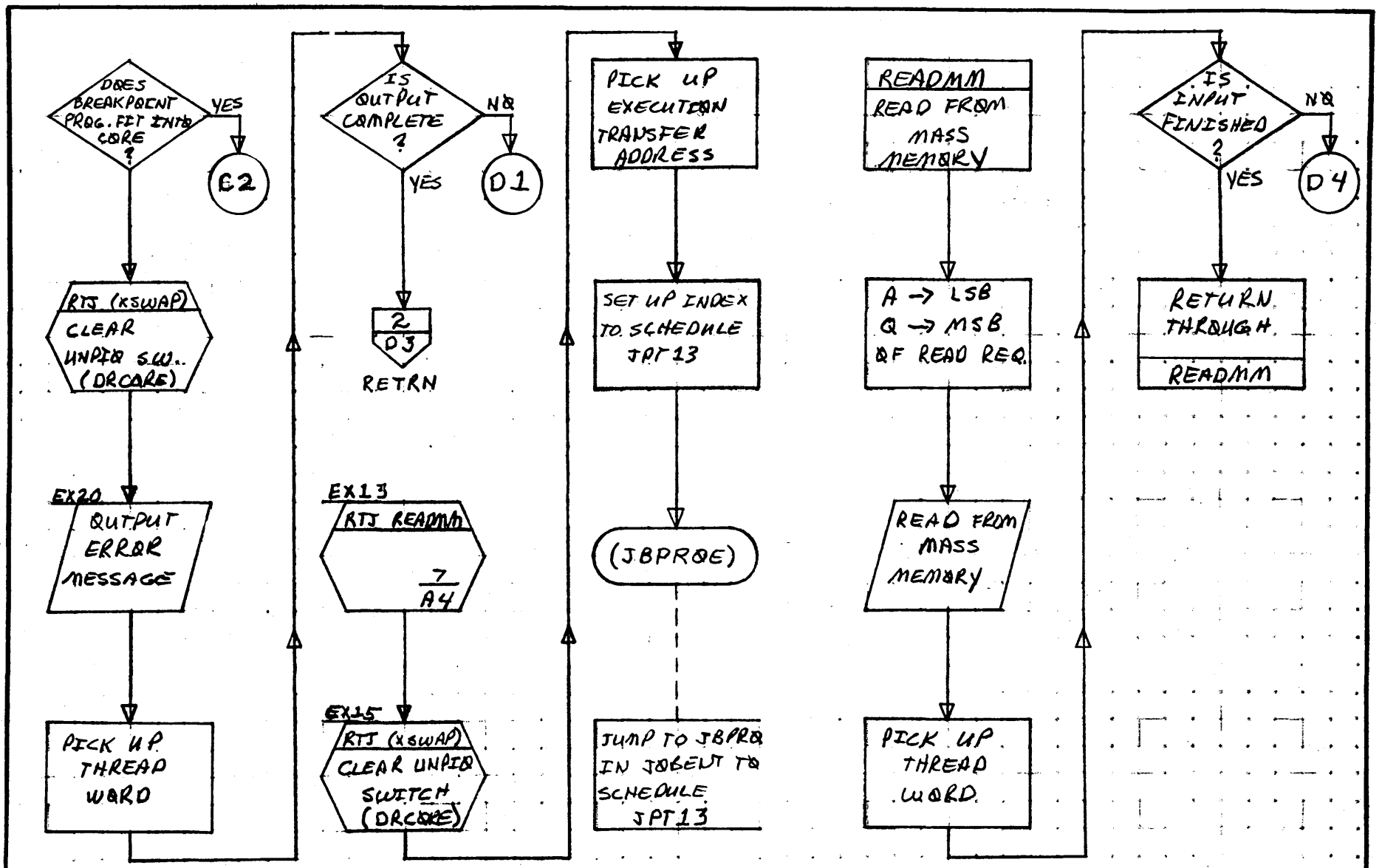
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JPLOAD		PROJECT MGR.			
FLOWCHART ☐		PAGE 6 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

31.11



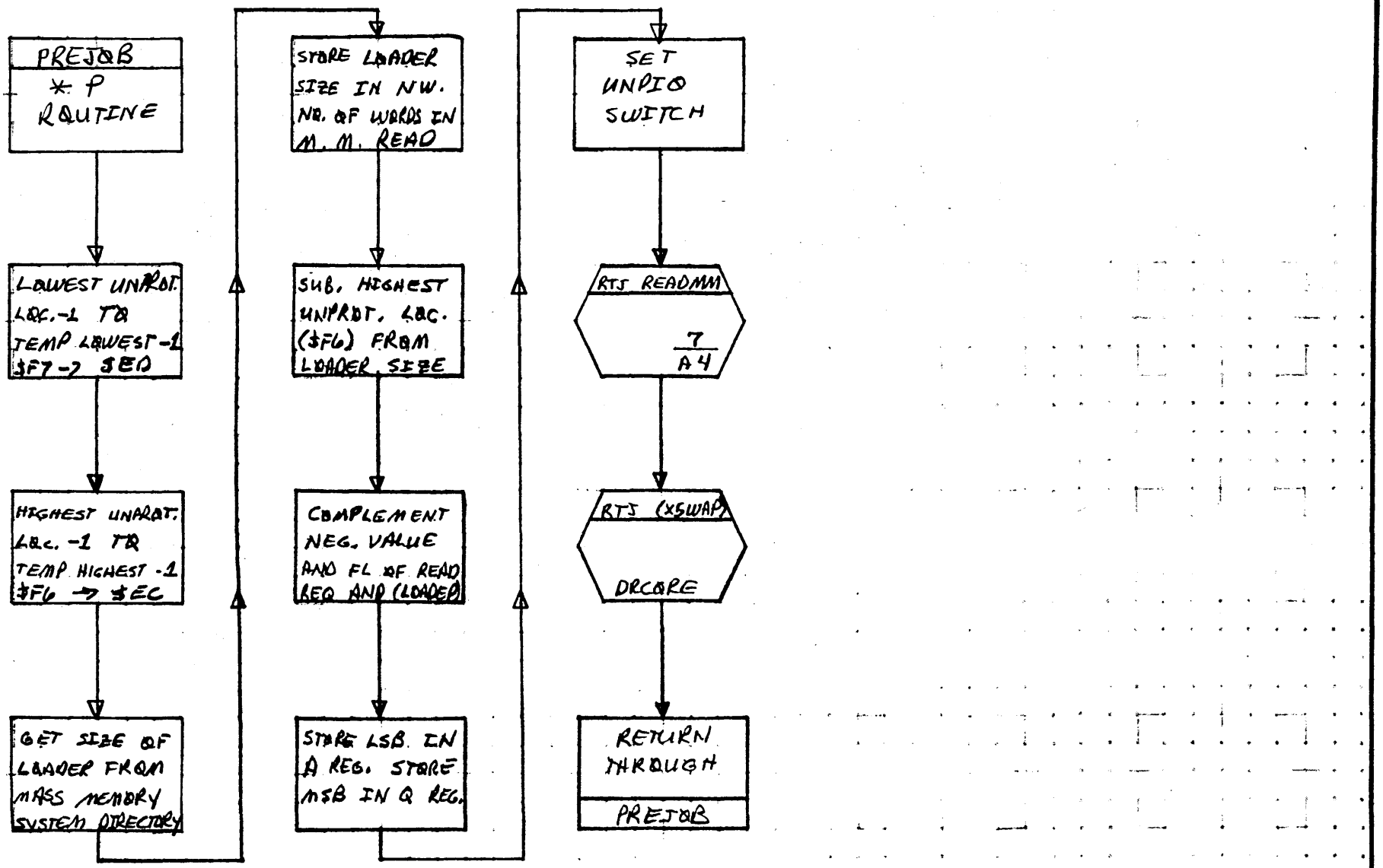
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JPLOAD		PROJECT MGR.		
		PAGE 7 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

31.12

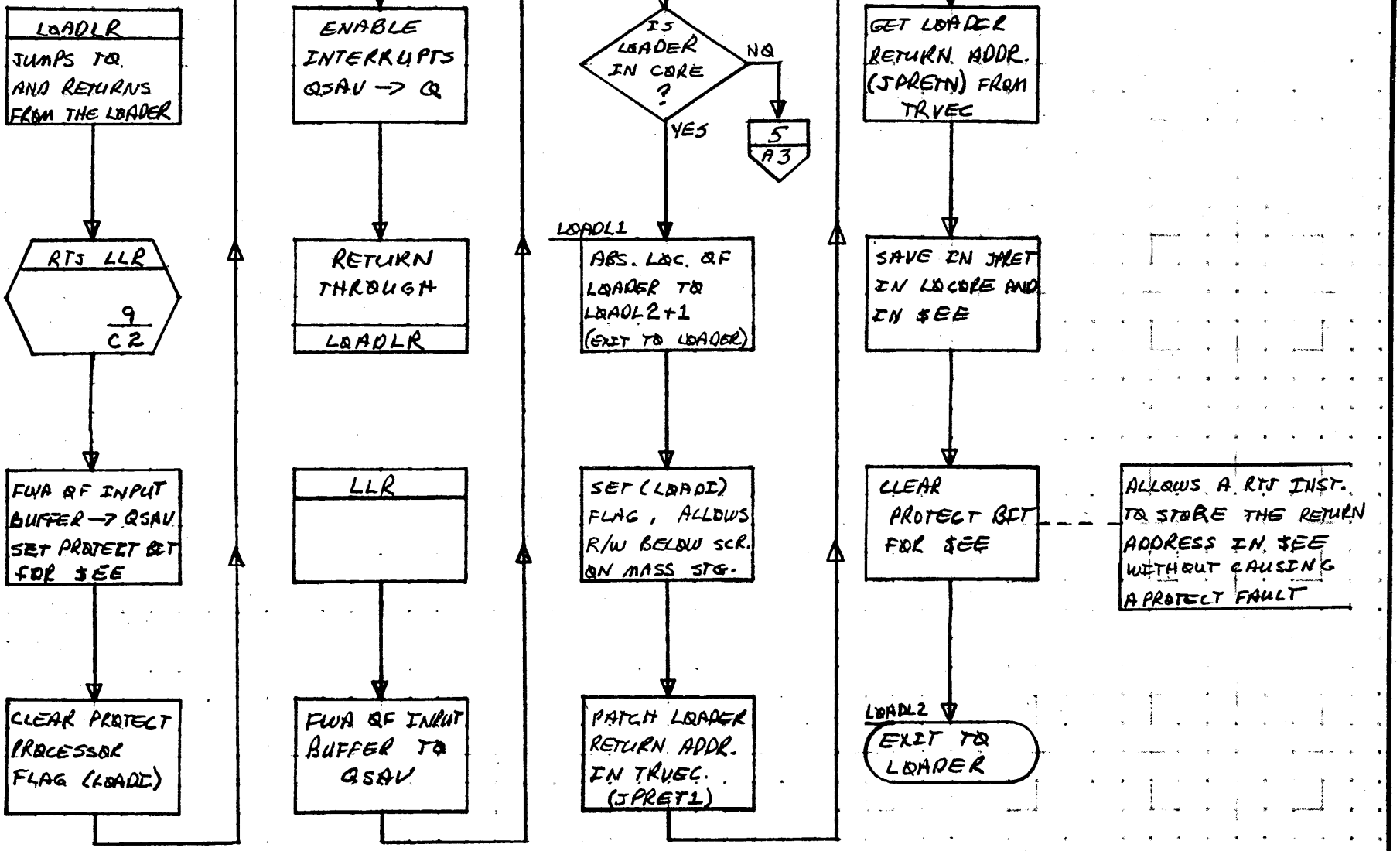
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE IPLOAD		PROJECT MGR.			
		PAGE 8 OF 11	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971



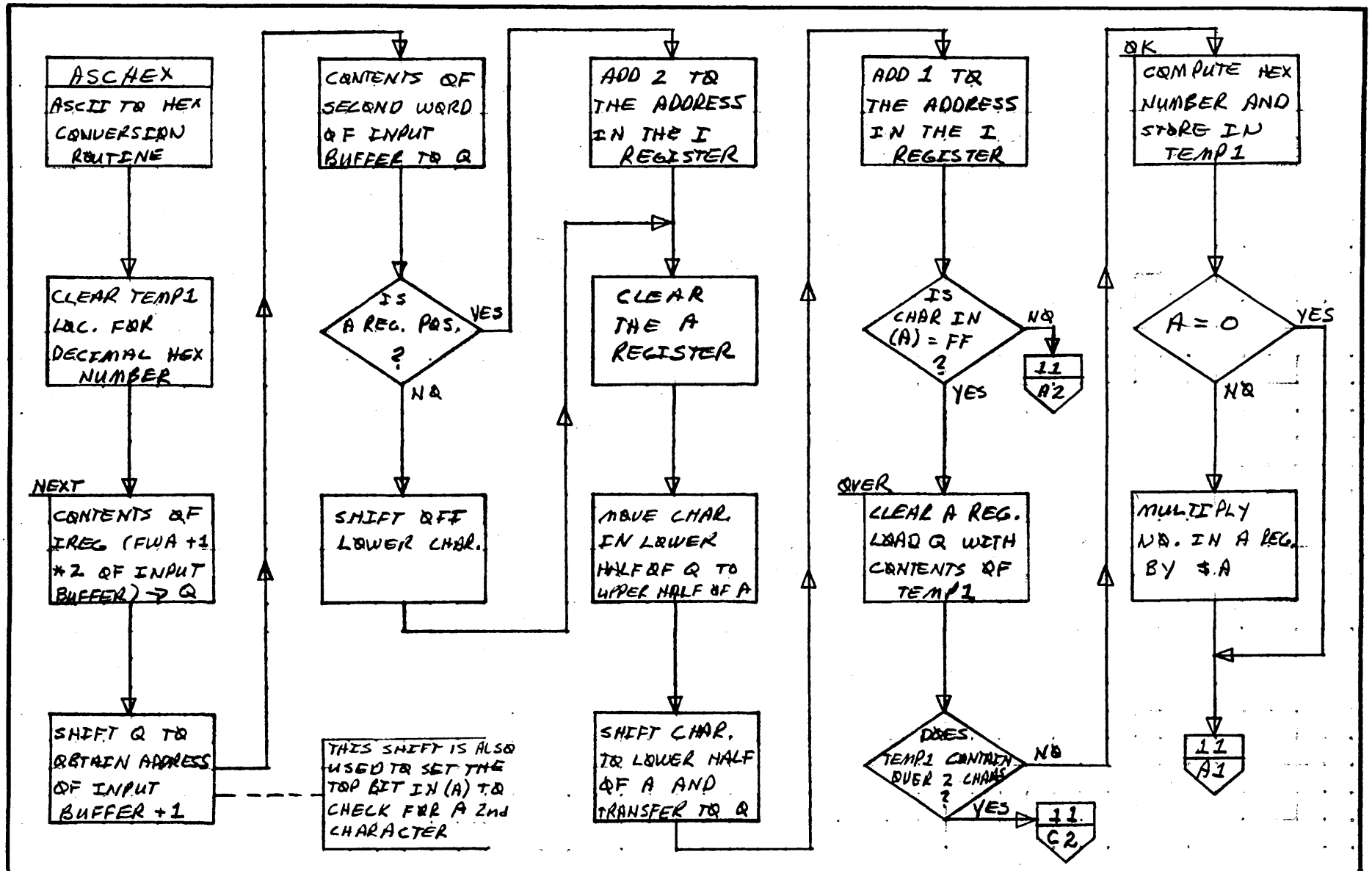
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JPLOAD		PROJECT MGR.		
		PAGE 9 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

31.14

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



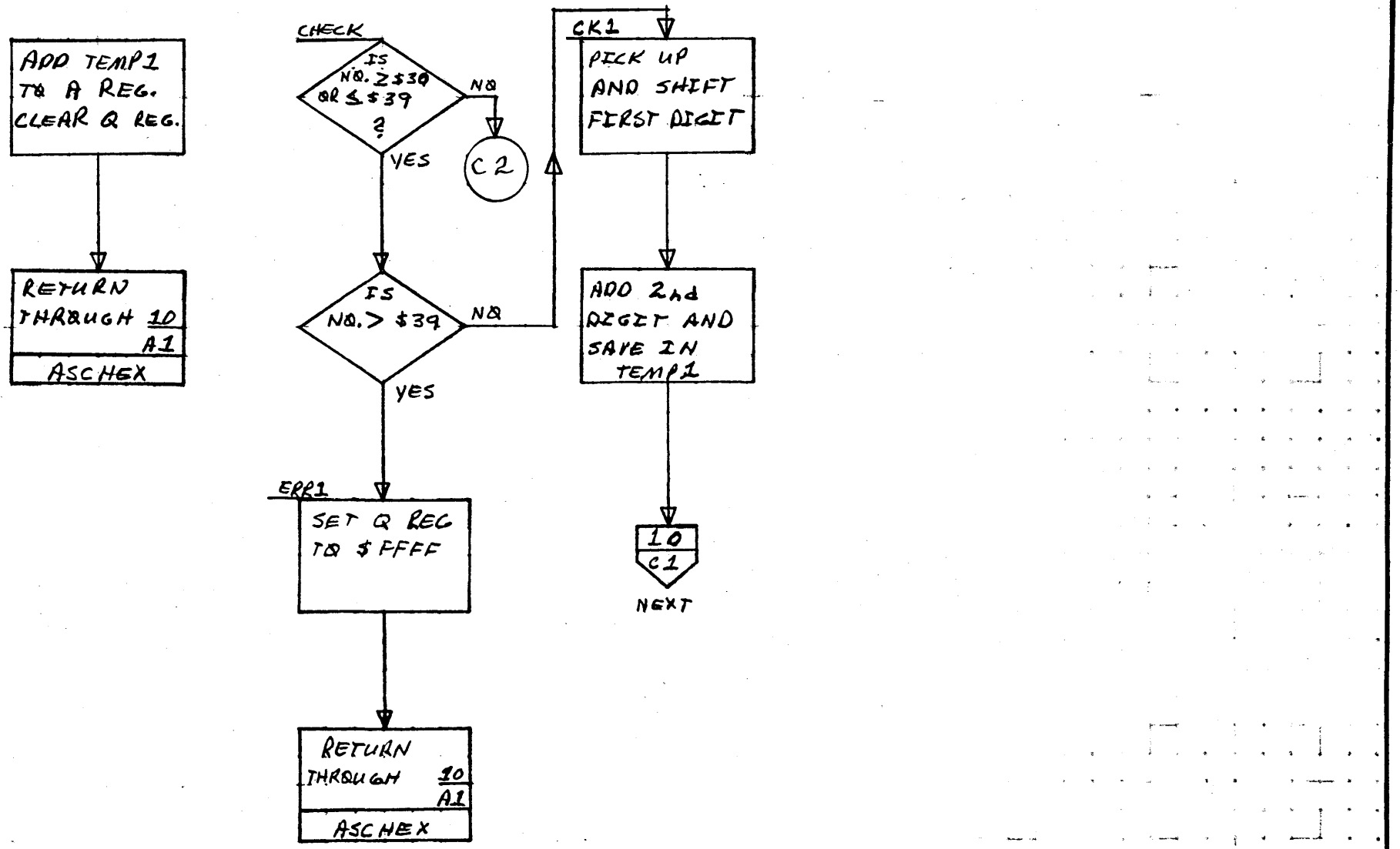
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE JPLOAD		PROJECT MGR.		
FLOWCHART ☐		PAGE 10 OF 11	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

31.15



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JPLRAD		PROJECT MGR.		
		PAGE 11 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

31.14

32.0 JPCHGE - ASCHEX

32.1 FUNCTION

This module processes ¤K and ¤V statements allowing the operator the option of selecting devices for system units other than those currently used.

32.2 ENTRY POINT NAMES

JPCG
CHANGE

32.3 EXTERNALS AND DESCRIPTION

NUMLU	Number of logical units in system {SYSBUF}
ASCHEX	Decimal ASCII to hex conversion routine
IUP	Contains location of input comment device {TRVEC}
TRANV	Contains location of TRANTA table in J0BPR0 {TRVEC}
FILE2	Contains absolute address of JPCHGE when in core.
L0G1A	Table containing Phystb addresses for logical units {SYSBUF}
MIB-MIBX	Flags in manual interrupt processor {MINT}
JBPR0E	Holds absolute address of JBPR0 routine in J0BENT {TRVEC}
MIBUF	Holds absolute address of MIPBUF buffer in J0BENT {TRVEC}

32.4 ENTRY INTERFACES

JPCG This entry point is scheduled by J0BENT

CHANGE This routine does the processing of the input statement.

32.5 EXTERNAL INTERFACES

Normal exit is to J0BENT where J0BPR0 is scheduled with an index to RF3. In case of an error, the error message is output within JPCHGE and exit is again to RF3.

32.6 GENERAL PROGRAM INFORMATION
Input Format

*K, I1, L1, P1 cr

l	logical unit number
I	system input unit
L	system print unit
P	system punch unit

*V, l, m cr

l	logical unit number
m	A or B A - formatted ASCII mode B - formatted binary mode

32.7 GENERAL DESIGN PECULIARITIES

None

32.8 PROGRAM LOGIC

This routine may be entered via the job processor {J0BPR0} or the manual interrupt processor {MINT}. If entry was via MINT the Q register will contain the location of the input buffer and exit will be made by clearing the MIBX flag and scheduling a release of this module with no return. If entry was via J0BPR0 the absolute location of JPCG will be put into FILE2, the location of input buffer will be

found from the transfer vector table buffer of J0BENT and the MIB flag will be cleared. The exit will be made to RF3 in J0BPR0 iva J0BENT. Entry determination is made by checking the priority level; if zero entry thru J0BPR0 is assumed.

The processing of the input statement is done thru the subroutine called "CHANGE" which in turn calls "ASCHEX" to translate the input statement. After the statement has been broken down to into characters and logical unit ordinals, each character is checked for equality with one of the expected characters. The logical unit ordinals are checked for valid numbers and a check is made in the physical device table, word 8, to insure the device is available to unprotected programs and is available for reading or writing from unprotected programs. In addition, the *V statement checks if a mode was selected and sets bit 12 in the specified communications region location if ASCII or no mode was selected. After completion of validation checking the ordinal is placed in the specified communication region location if no error occured.

If an error occured, a message is output and an exit is made bypassing any remaining parameters. Possible errors messages are:

J04 - Illegal or unintelligible parameters in the control statement.

J07 - Device not available to unprotected programs.

J08 - Either a read device has been assigned to a write only ordinal; or a write device has been assigned to a read only ordinal.

32.9 SUBROUTINE LOGIC

32.9.1 ASCHEX - this subroutine translates the input statement.

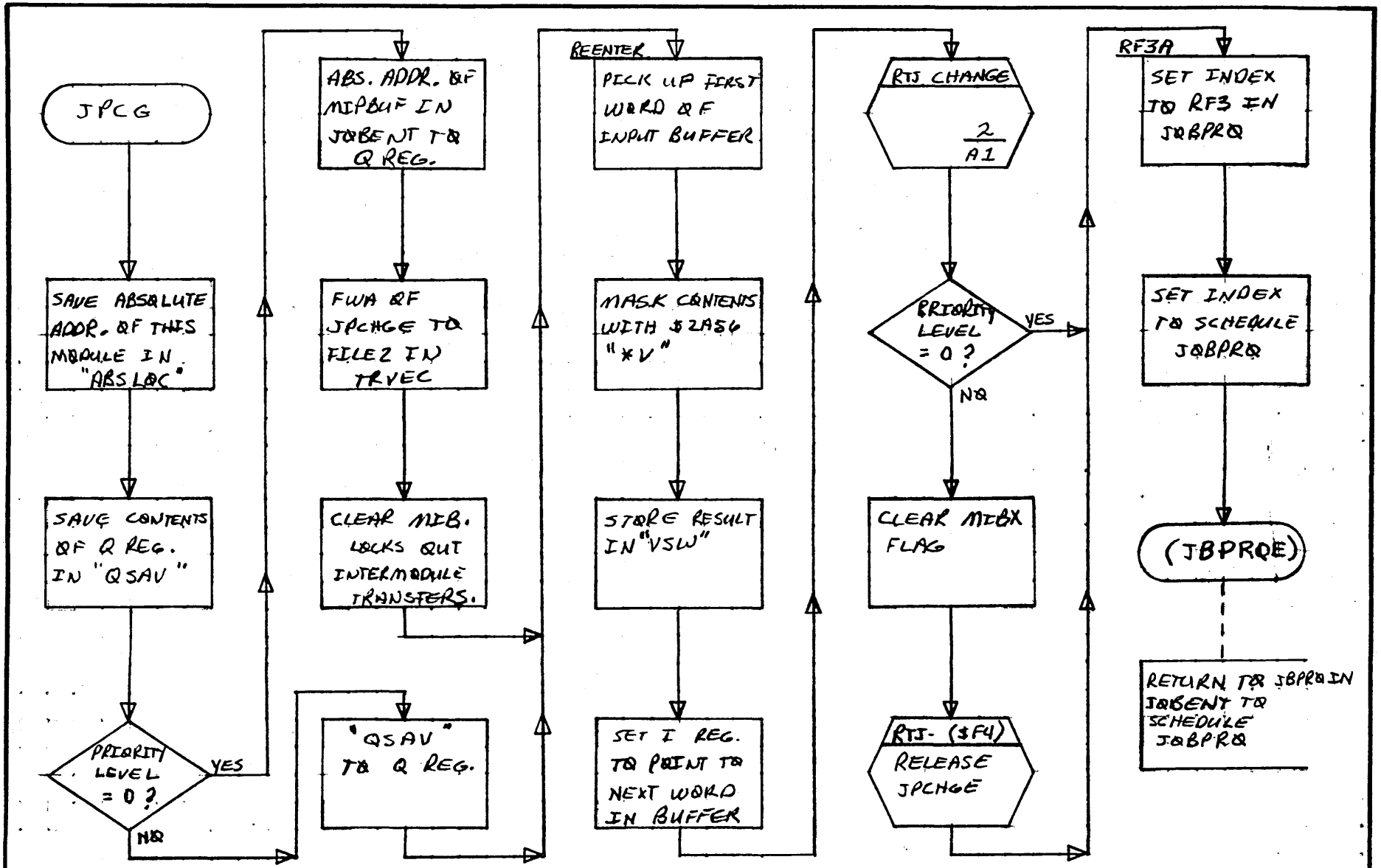
As passed by ▽CHANGE▽. It in turn passes one character at a time to ▽SUB▽ for interpretation.

Parameters for ASCHEX are:

Register	Upon Entrance	Upon Exit
A		+1 error 0 more words following -1 no words follow
Q	Loc to store ALPHA char	Hex number
I	Loc of char	Loc of next character

Parameters for SUB are:

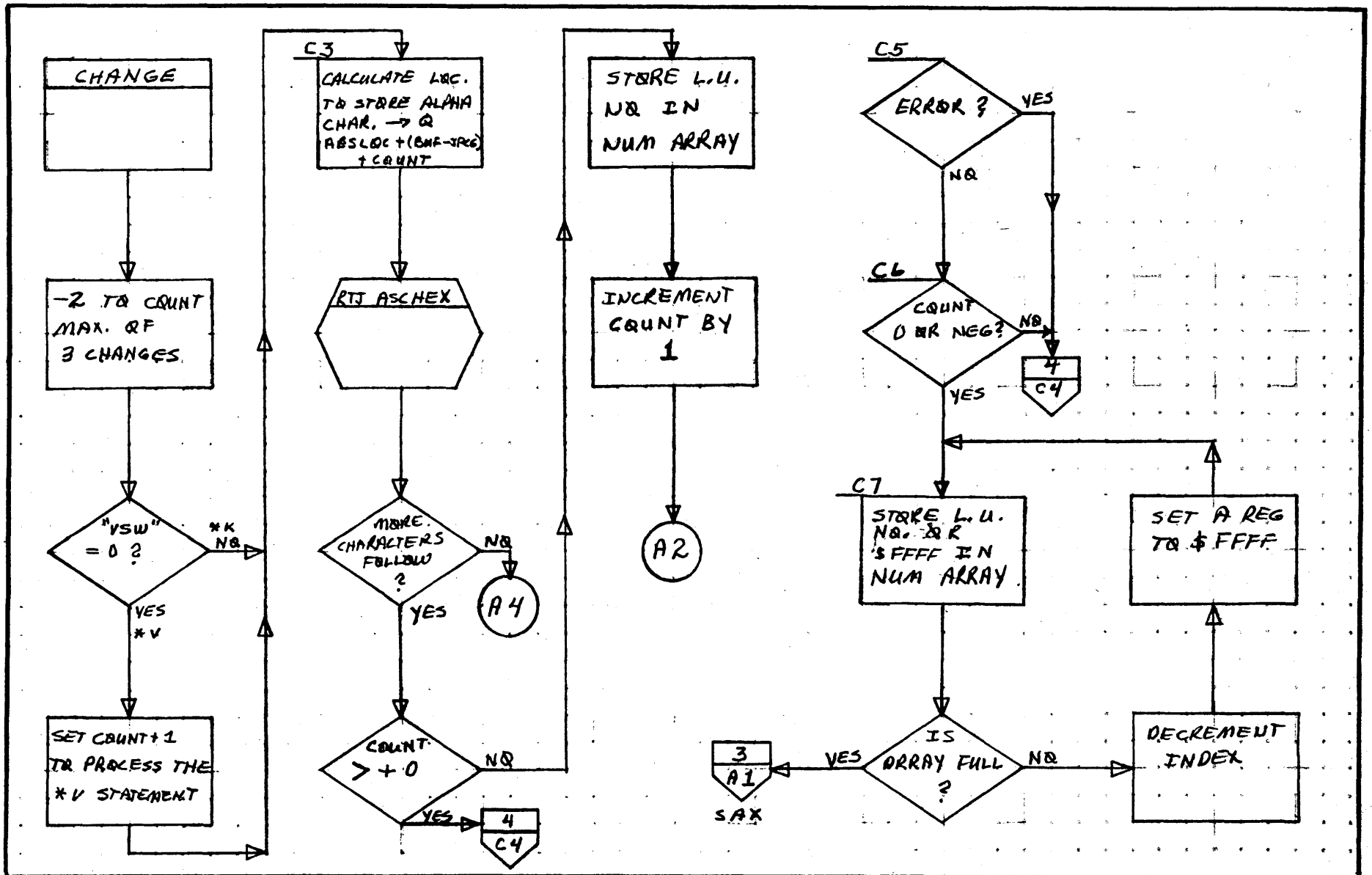
Character Found	Return Registers	
	A	Q
COMMA	0	0 1st char
	0	-1 2nd char
$FF	-1	-1
NO.	0	0
CHAR	0	0
ERROR	+1	-1



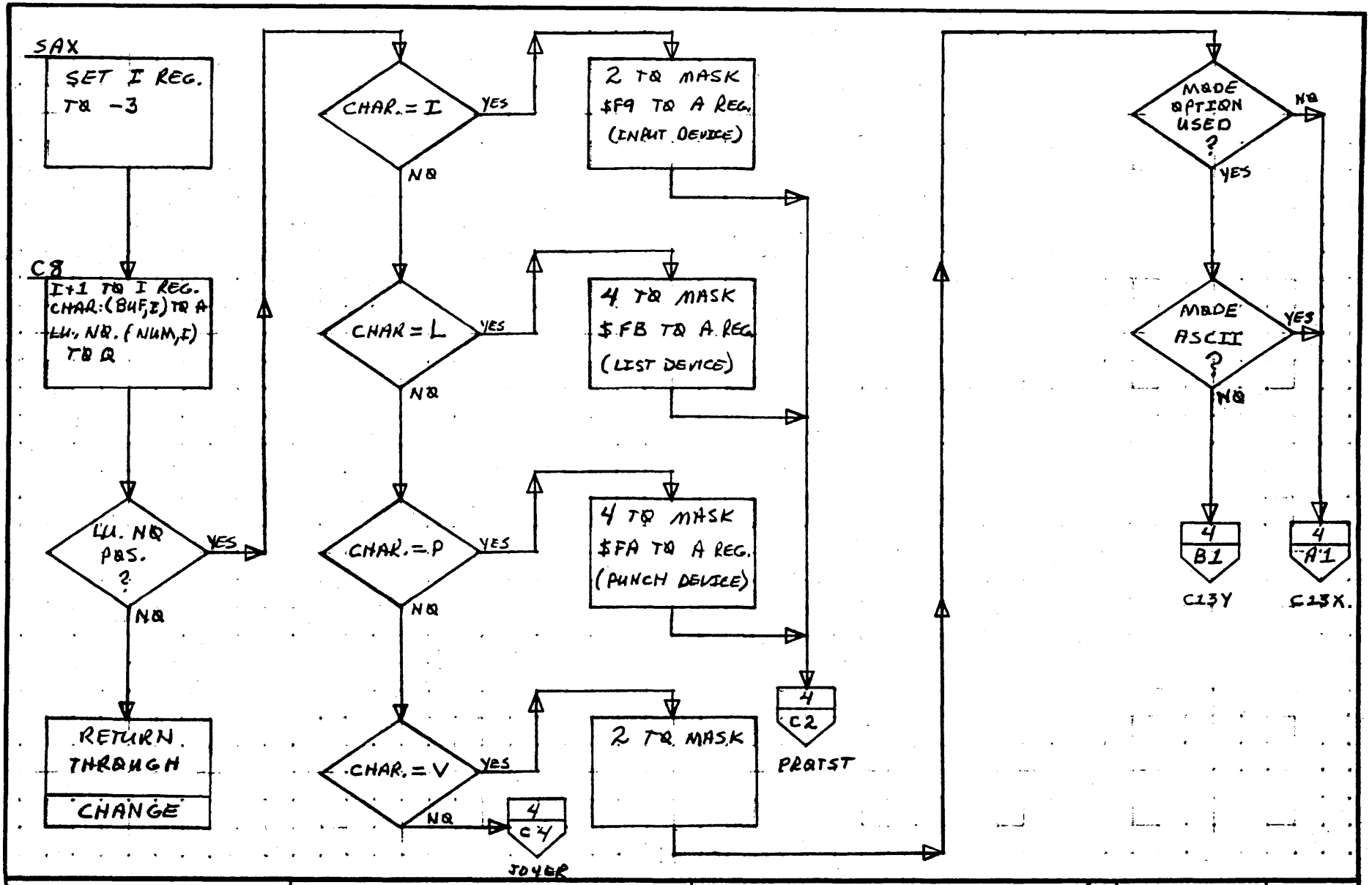
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS I113	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JPCHGE		PROJECT MGR.		
		PAGE 1 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

32.5



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JPCHGE		PROJECT MGR.		
		PAGE 2 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



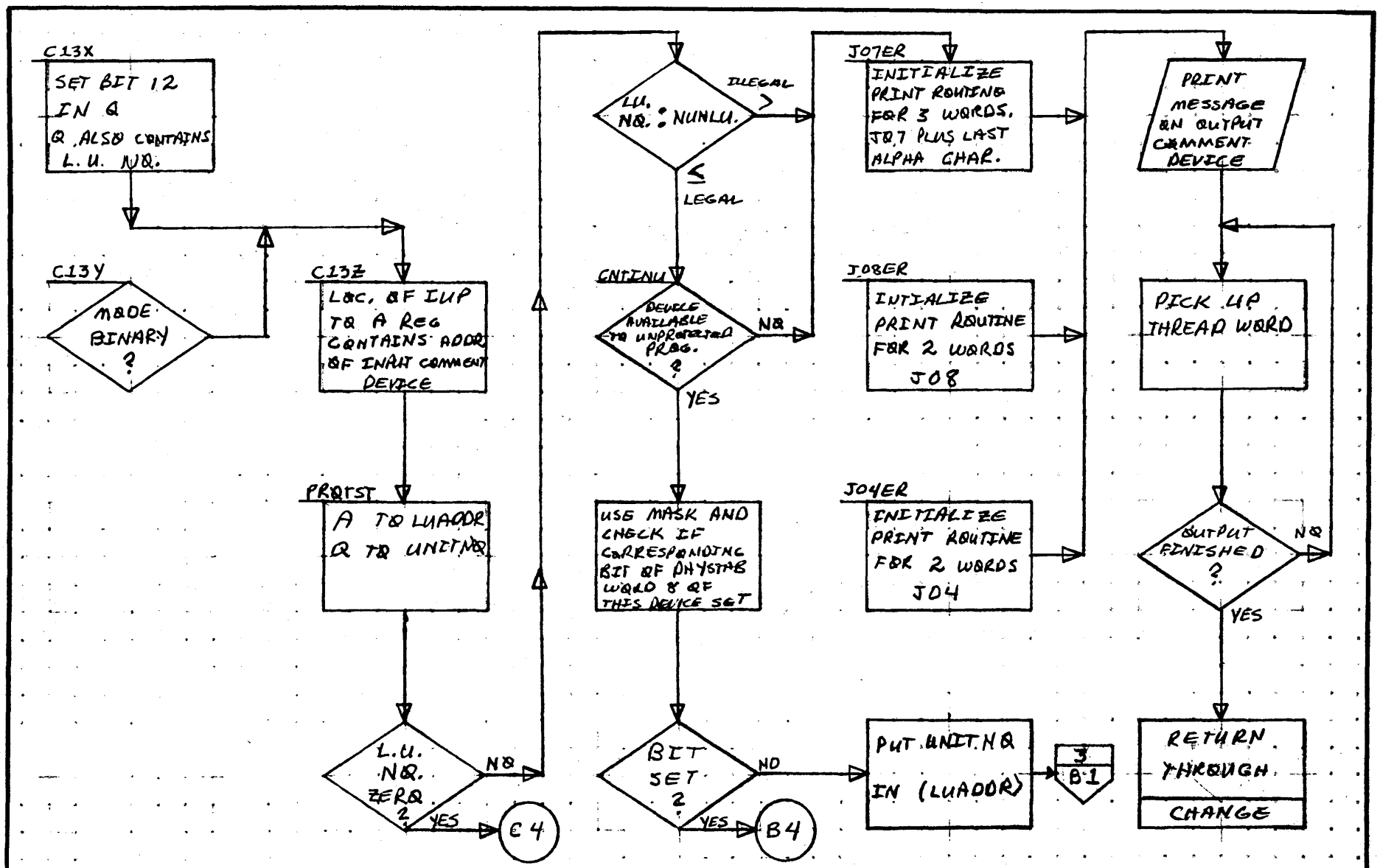
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE JPCHGE		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

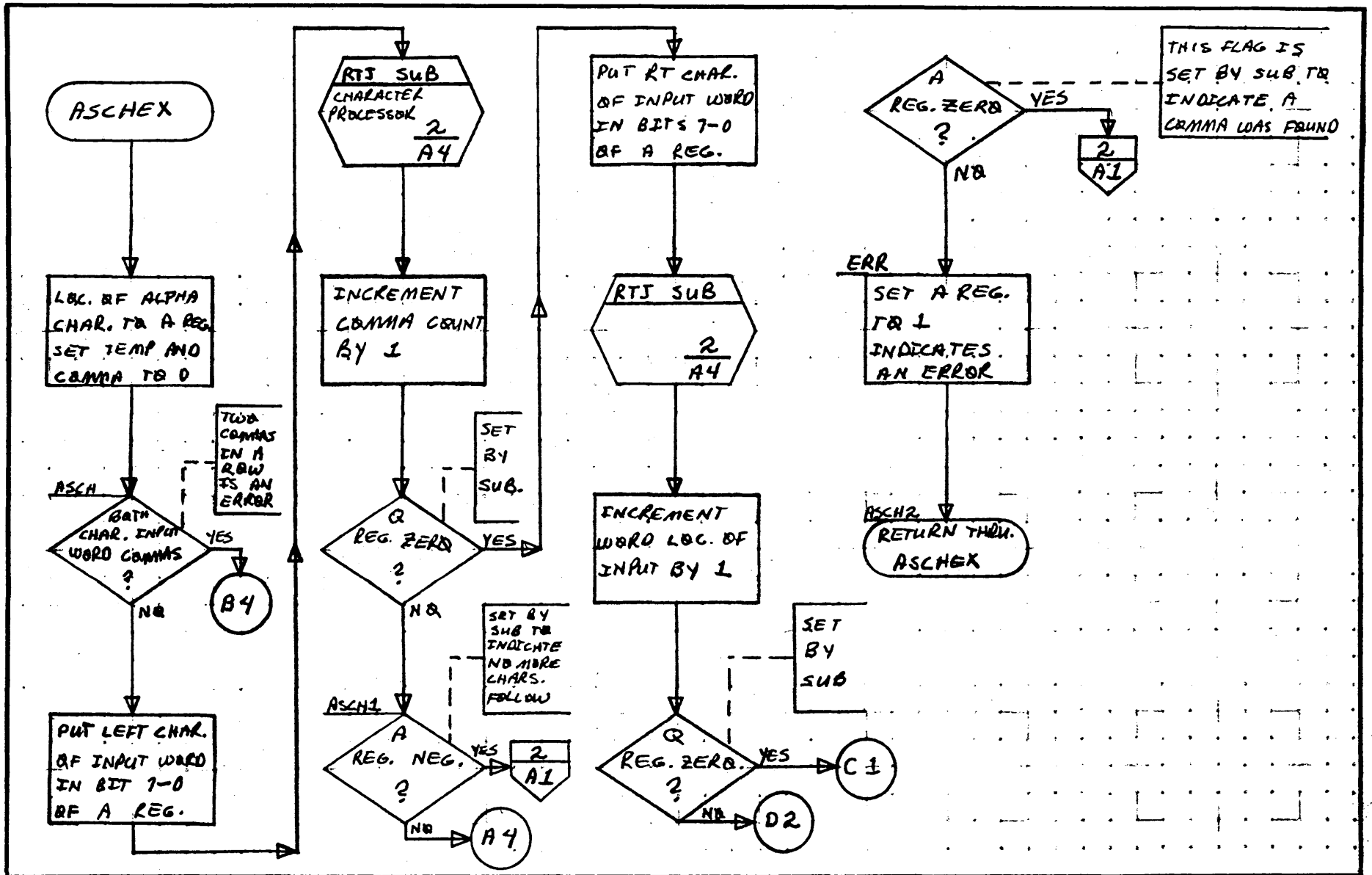
32.7



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE JPCHGE		PROJECT MGR.		
		PAGE 4 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

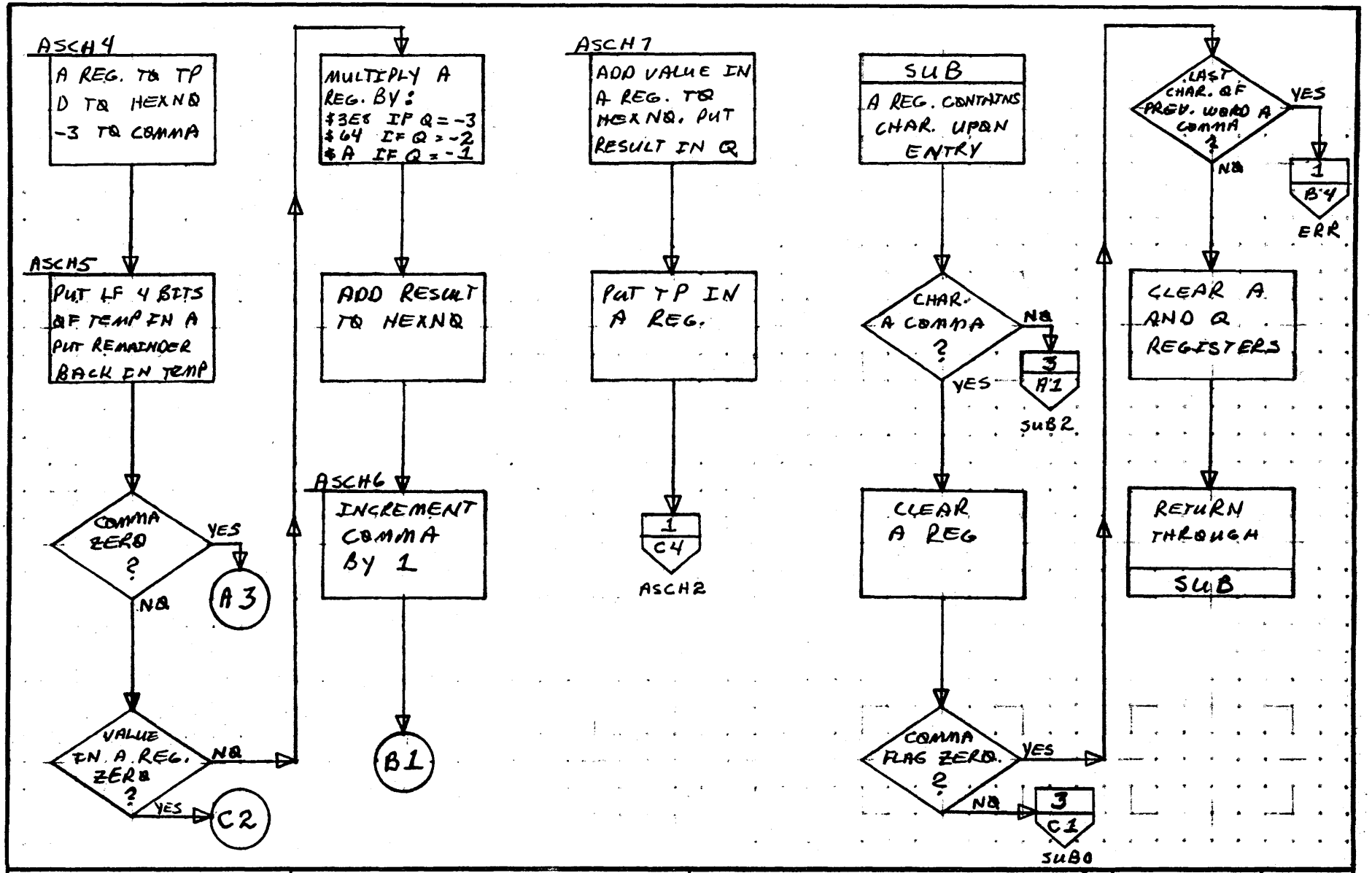
32.8



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ASCHEX		PROJECT MGR.		
		PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

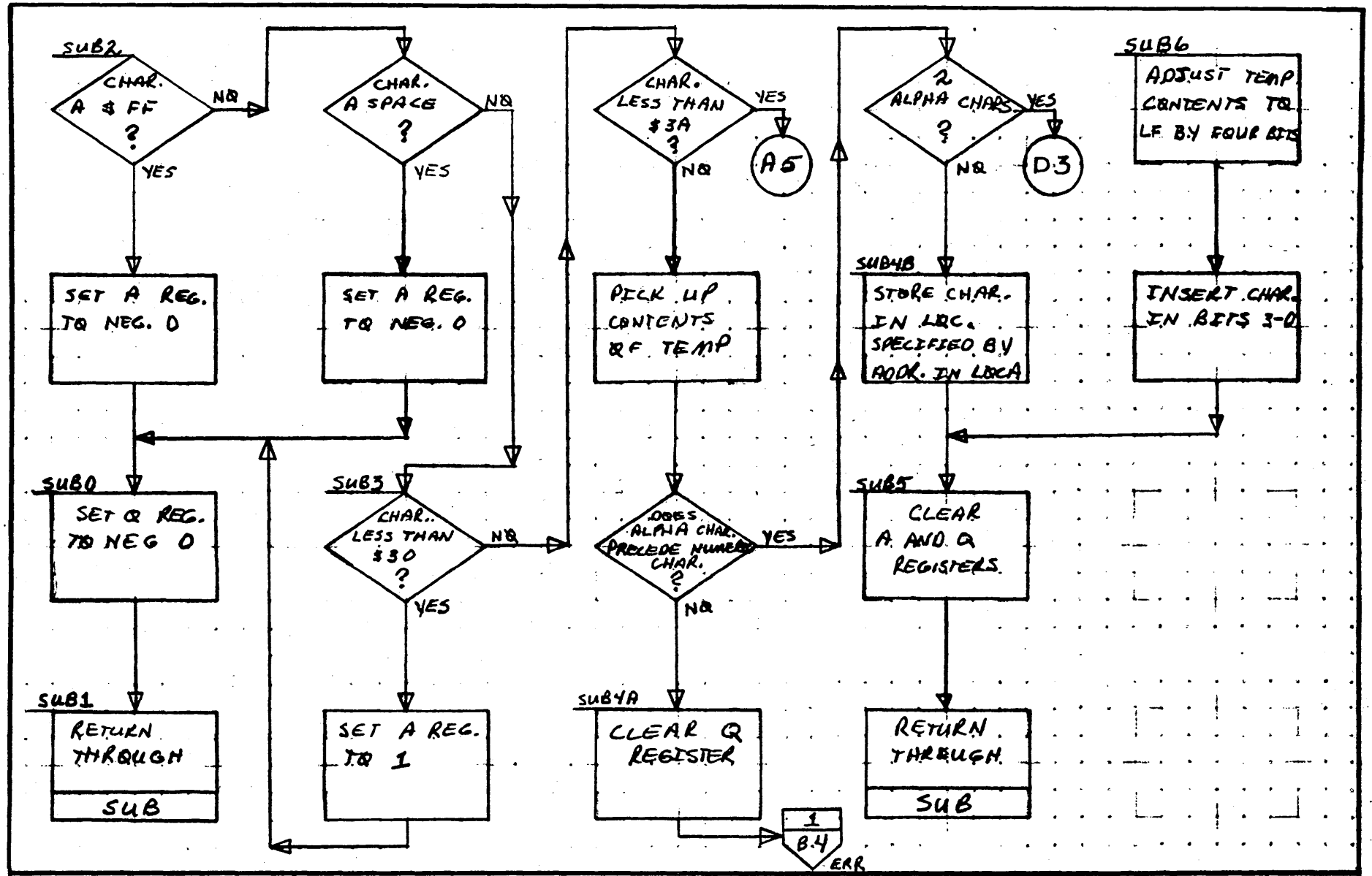
32.9



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ASCHEX		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

32.10



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ASCHEX	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

22.11

33.0 RESTOR RESTORE LOGICAL UNIT

33.1 Function

This routine informs the 1700 Operating System that action has been taken to restore the device to an operable state and it is now ready for I/O.

33.2 Entry Point Names

RSTR Restor Entry Point

33.3 Externals and Description

MIB Manual Interrupt Lookout flag {MINT}

MIBX Manual Interrupt Lookout flag {MINT}

LOGL Logical Unit table {indexed by logical unit} which contains the alternate logical unit number and various indications. {SYSBUF}

JBPROE

JBPROE Holds Abs. Addr. of JBPRO routine in JOBENT. {TRVEC}

MIBUF Holds Abs. Addr. of MIPBUF buffer in JOBENT. {TRVEC}

FILE2 Holds FWA of RESTOR when in core. {TRVEC}

LOGLA Logical Unit Table {indexed by logical unit} which contains the core location of the physical device table entry corresponding to each logical unit.

33.4 Entry Interfaces

This routine is scheduled into core by the JOBENT control module when it has been determined that a *R,lu statement has been entered.

33.5 External Interfaces

Normal exit is the JBPRO routine in JOBENT where JOBPRO is scheduled with an index to the RF3 routine within JOBPRO. If an error occurs a J04 message is output and exit is also through JOBENT to RF3 in JOBPRO.

33.6 General Program Information

If this program is entered at a priority level greater than zero the RESTOR routine releases itself before exiting back to the caller.

33.7 General Design Peculiarities

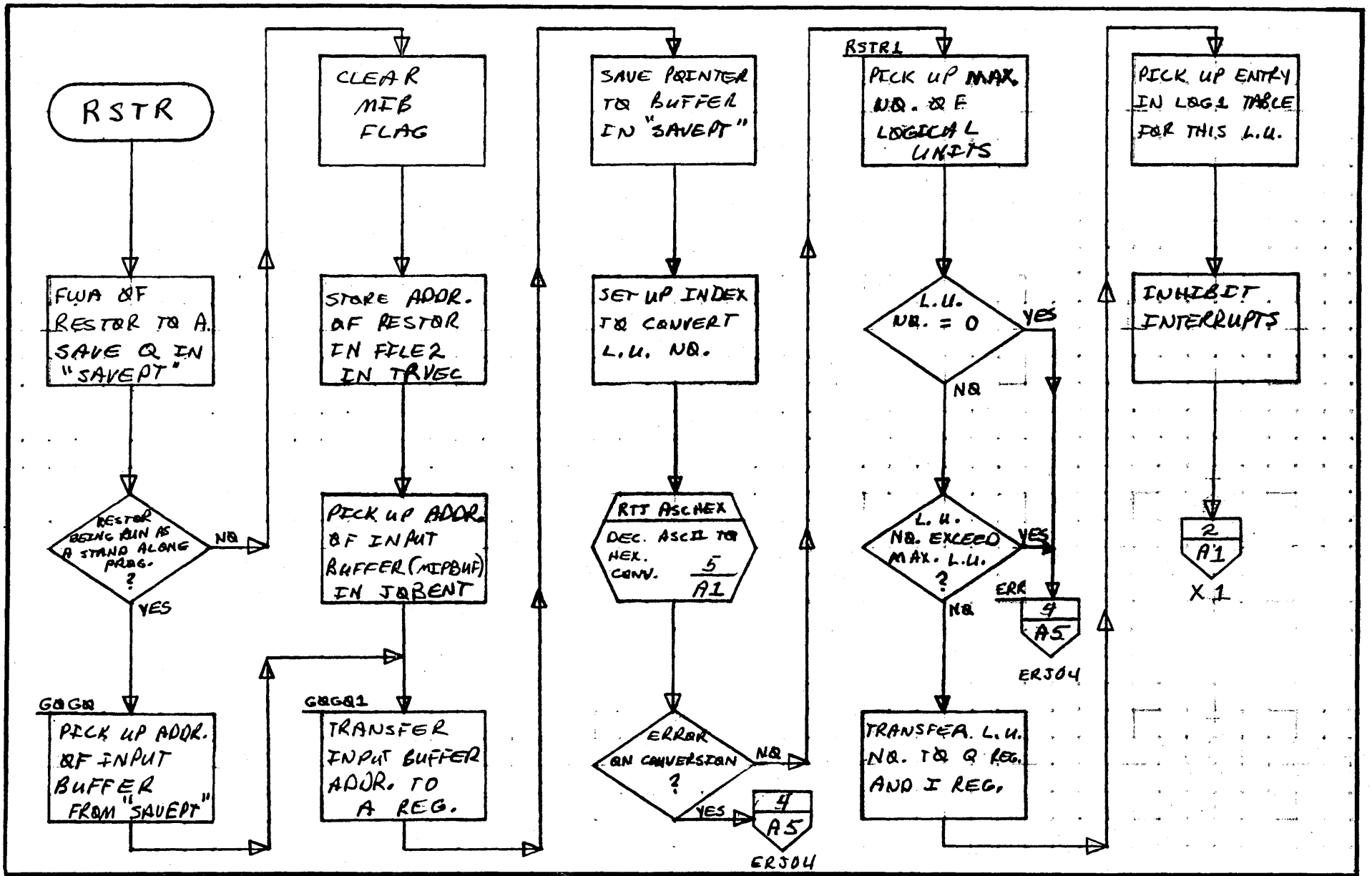
None

33.8 Program Logic

Upon entry to RESTOR the current priority level is checked and if greater than zero the input buffer address is in the Q register. If zero, pick up the buffer address in JOBENT and return jump to the ASCHEX subroutine to convert the logical unit number specified. Upon returning a check is made for a Logical Unit number error, and if so a J04 error message is output and control is returned to the caller. If no error the logical unit down bit is reset in the respective entry in the LOGI table in SYSBUF and the message flag bit (bit 8) in the first word of the logical units phystb is cleared. This message flag bit is reset for all devices that share this phystb. The restore device bit (bit 11), used by FRN, is then set in the LOGI table entry for this logical unit, and the message LU xx RESTORED is output. Control is then returned to the caller.

33.9 Subroutine Logic

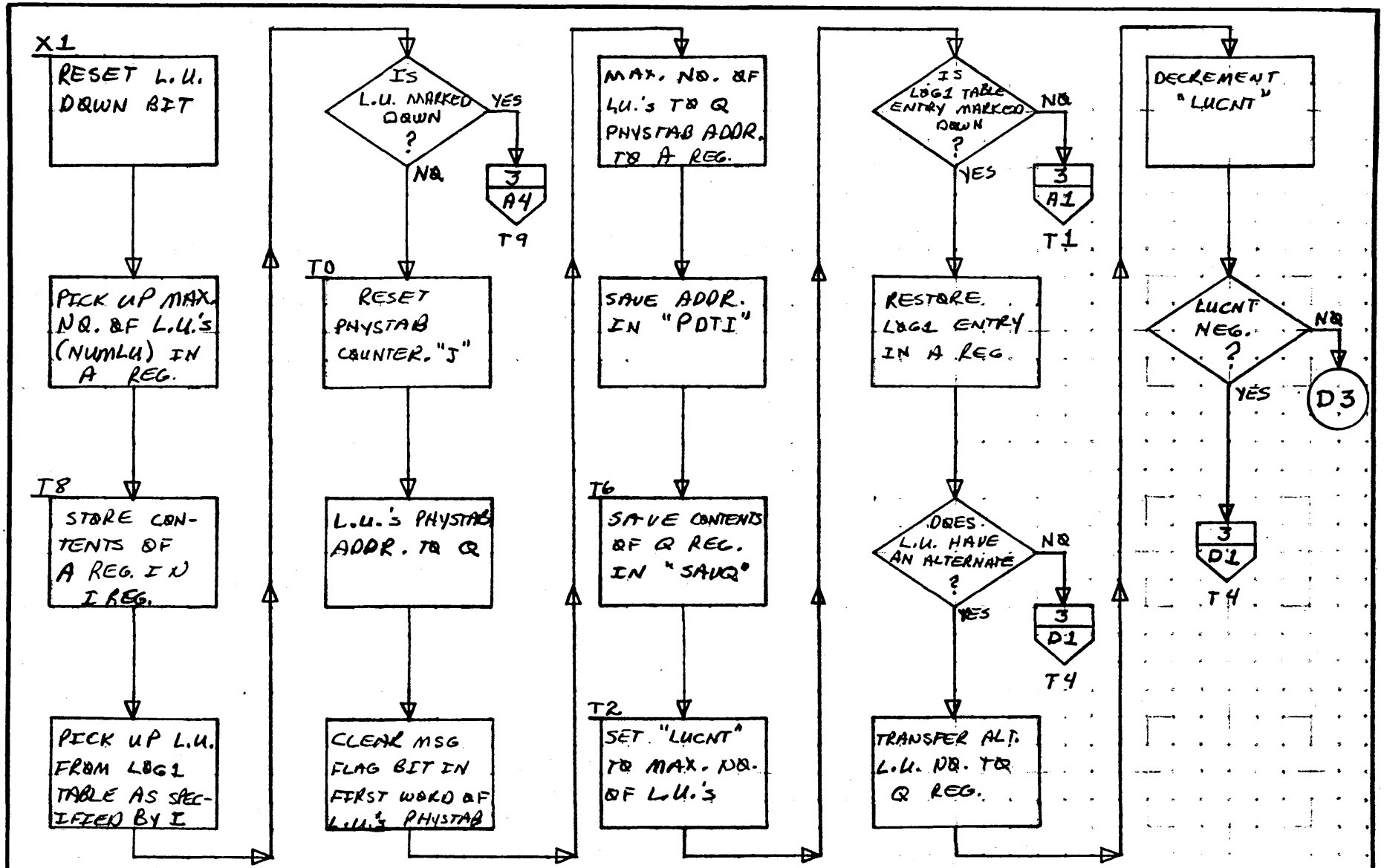
ASCHEX - This routine connects the decimal ASCII logical unit number entered to Hexadecimal. Error checks are made for more than two characters entered, characters less than $30 and greater $39. If an error occurs the Q register is set to negative zero and control returns to the caller. If no error is detected, the converted logical unit number is saved in the A register and control is returned to caller.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1100	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RESTOR		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

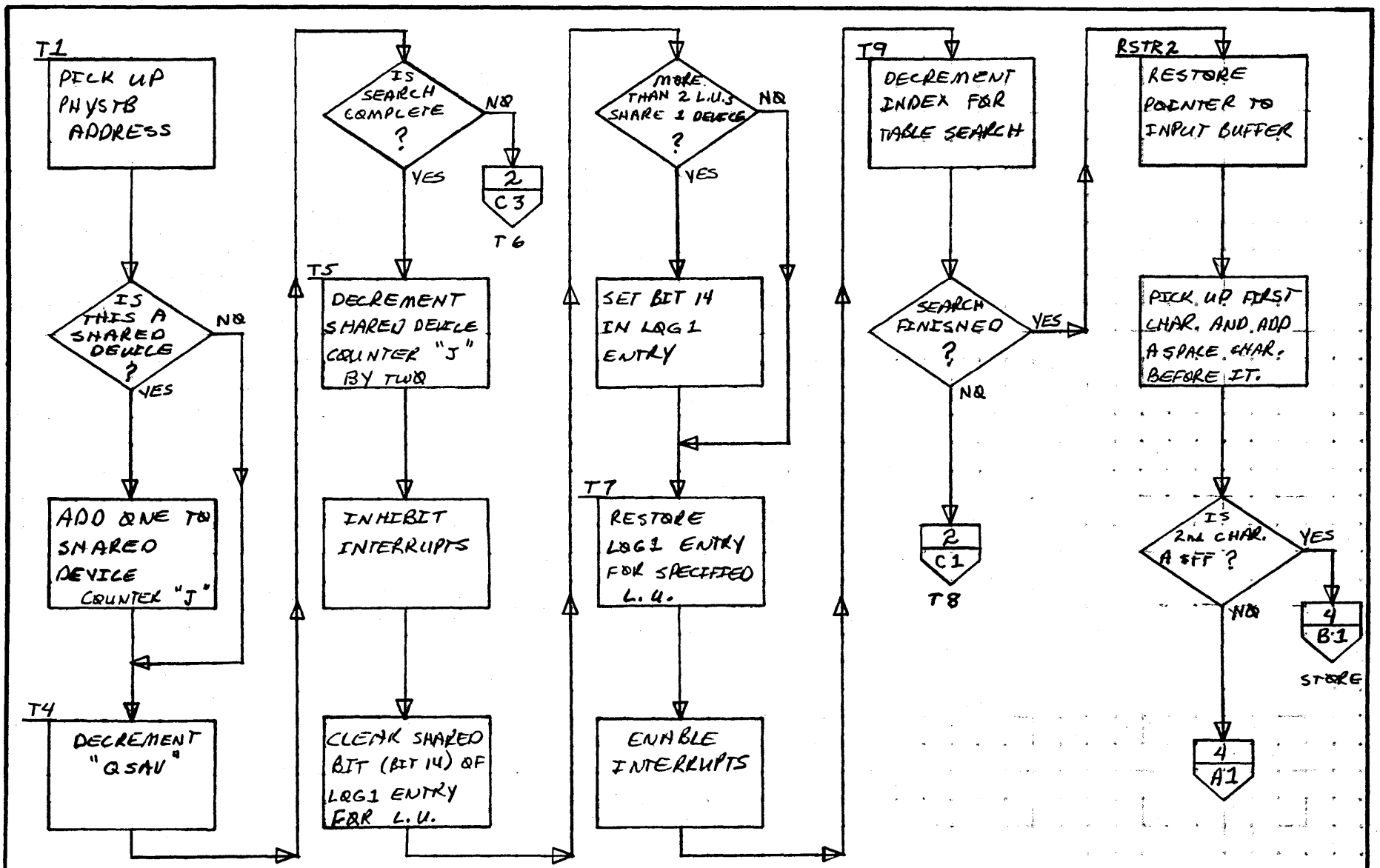
33.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RESTOR	PROJECT MGR.			
	PAGE 2 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

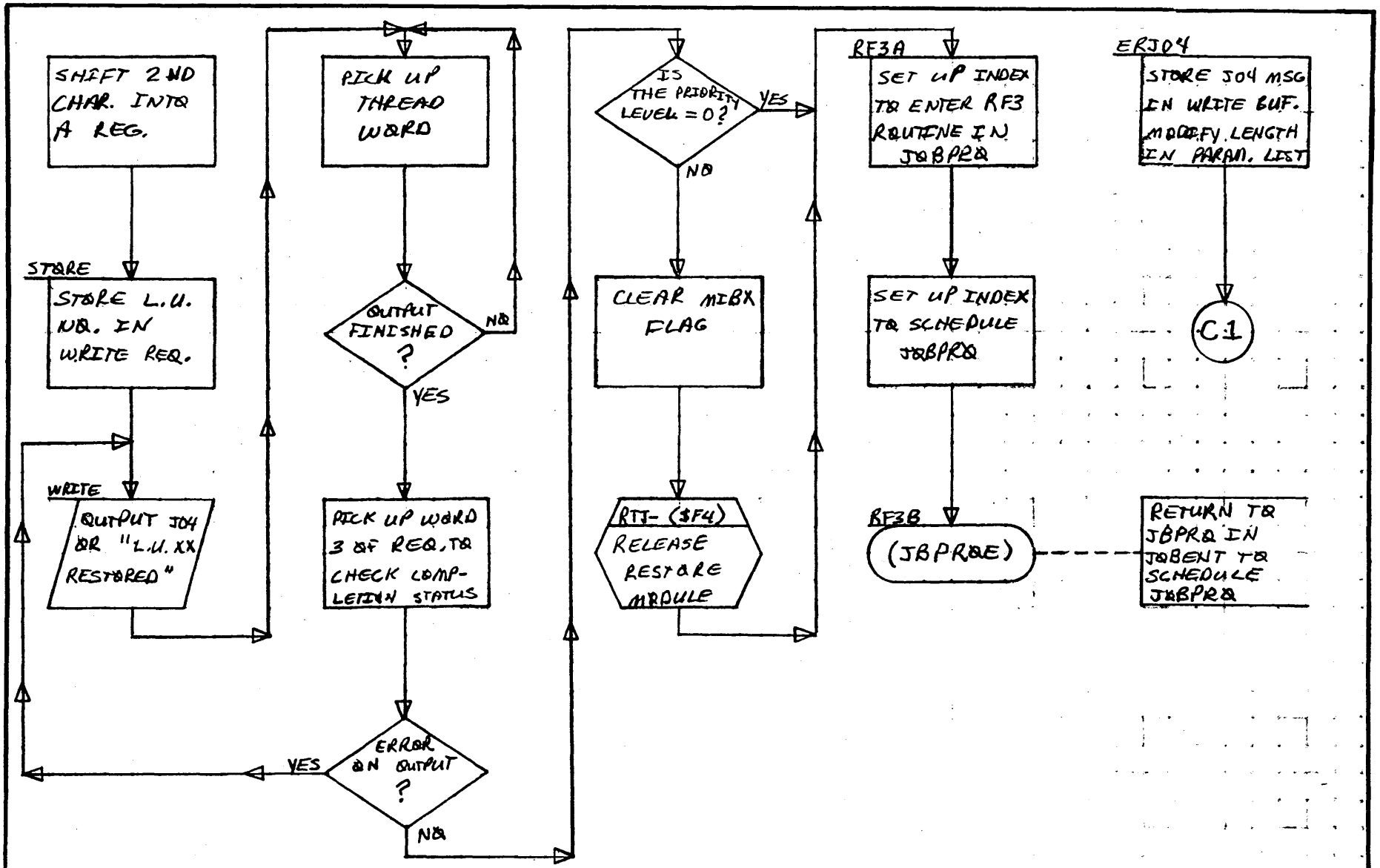
33.4



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RESTOR		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

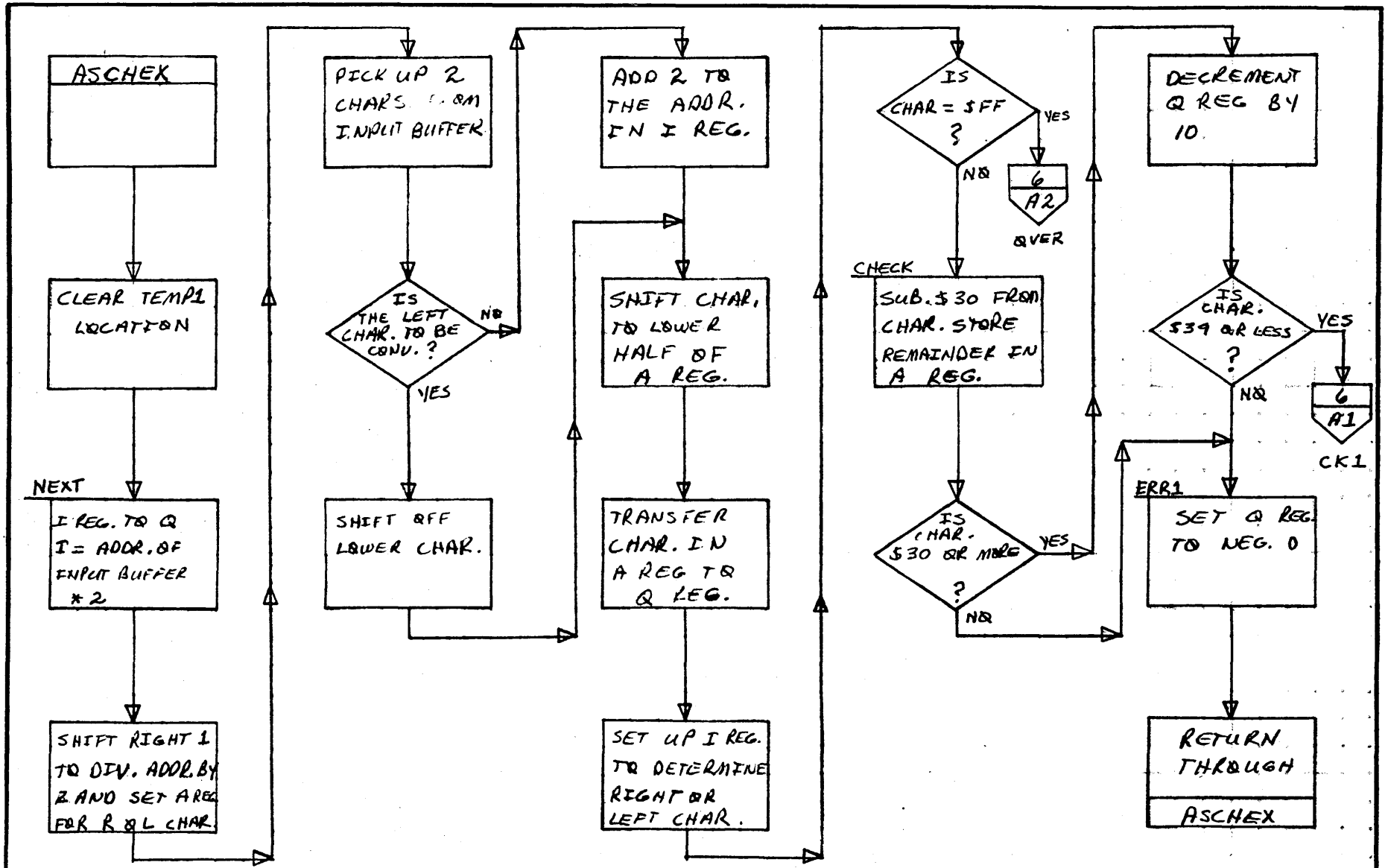
33.5



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RESTOR	PROJECT MGR.			
	PAGE 4 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

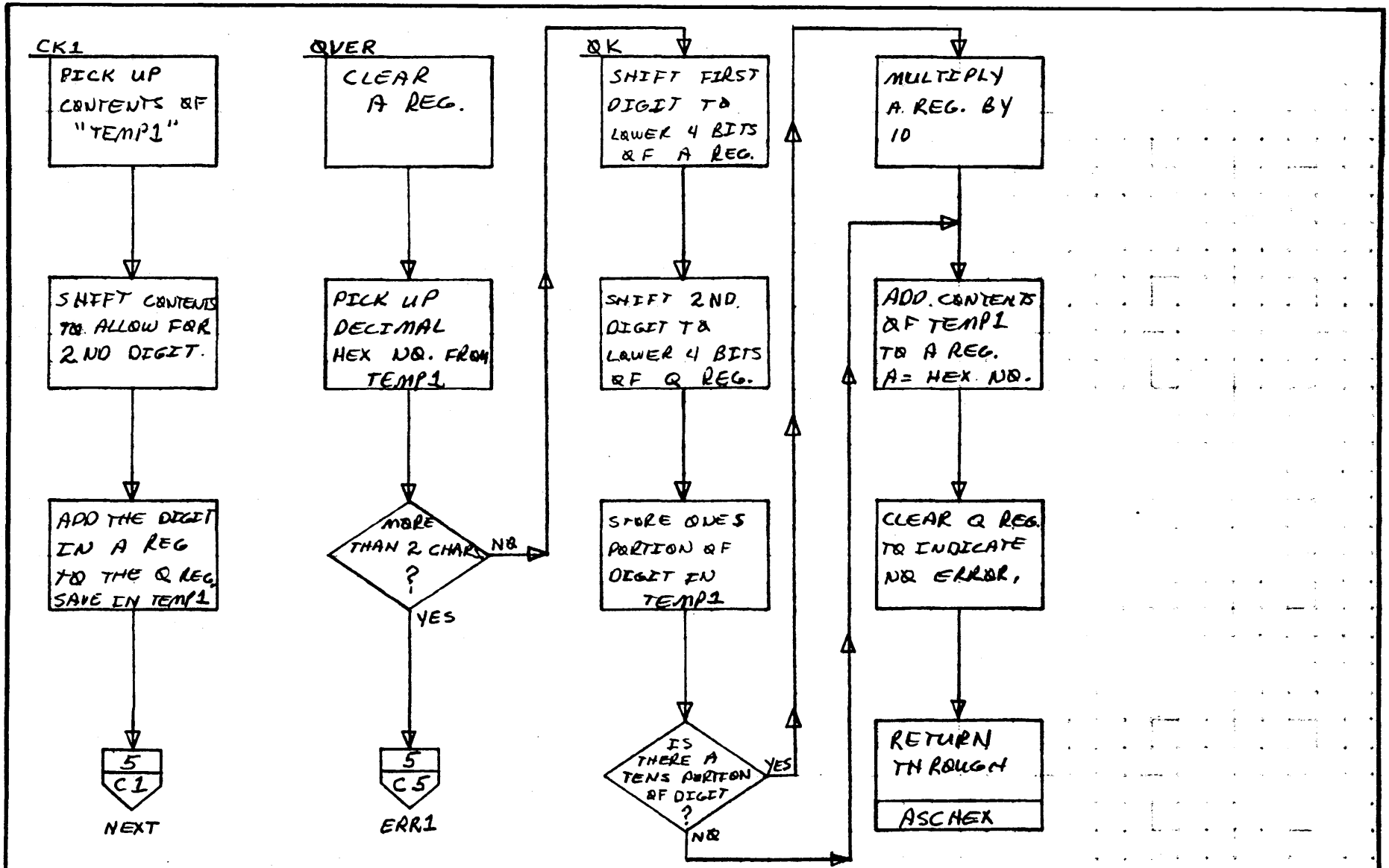
33.6



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE RESTOR		PROJECT MGR.		
FLOWCHART ☐		PAGE 5 OF 6	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

33.7



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RESTOR		PROJECT MGR.			
FLOWCHART ☐		PAGE 6 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

33.8

34.0 T13 GET-FILE REQUEST PROCESSOR

34.1 Function

This request is used to access permanent files in the program library. It is loaded under an *YM, JPT13 load, for mass storage resident programs. This routine will be brought in just before going into program execution and will be in core only during execution of a user's program, i.e., while *X Job Processor statement is in effect. Upon termination of a job, i.e., JOBKIL, normal EXIT, or SIGNOFF, "JPT13" module will be released.

34.2 Entry Point Names

EREQST - Initial Entry Point
T13 - Entry Point to Process GTFILE requests.

34.3 Externals and Description

SCHERR Error processor in SCHEDU

SWAPCK Routine in DRCORE which decrements UNPIO on completion of an unprotected I/O request.

LOCF Pointer to F routine in PROTEC {TRVEC}

LPTRS Pointer to PTRs in PROTEC {TRVEC}

FILE2 Holds FWA of T13 when in core {TRVEC}

MIB Manual Interrupt lockout flag {MINT}

RCTV Request Code Transfer Vector table {NMONI}

34.4 Entry Interfaces

The initial entry to T13 is to the EREQST routine. This routine calculates the absolute address of T13 and stores it in the RCTV table in NMONI and jumps to Breakpoint program if requested or to the program to be executed.

The T13 entry point is entered when a GTFILE request is made.

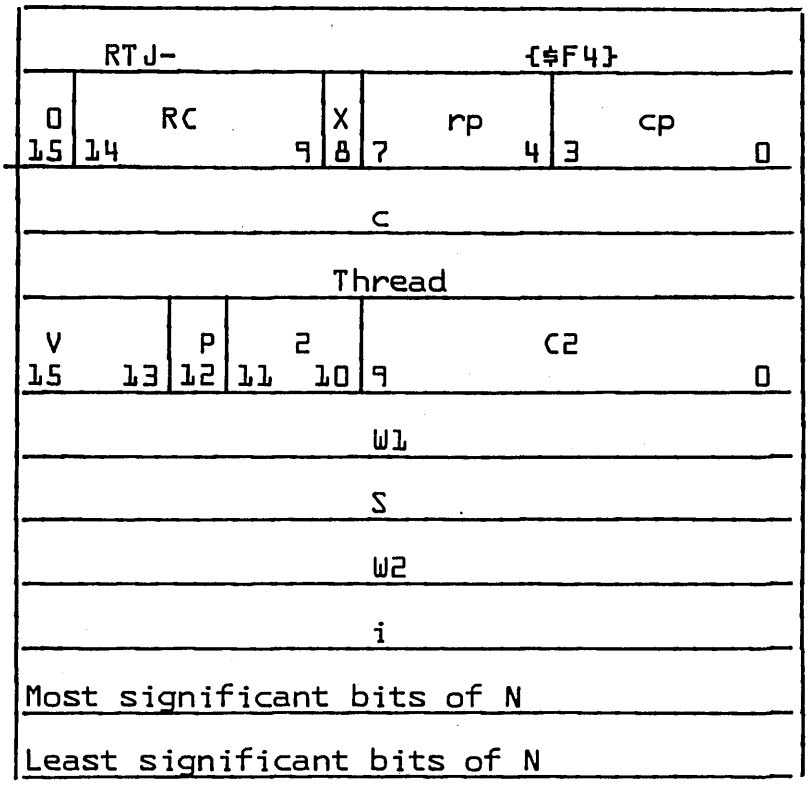
34.5 External Interfaces

Exit is made to REQXT after setting up the internal thread. After a GTFILE read request is initialized, exit is to the Dispatcher.

34.6 General Program Information

34.6.1 GET-FILE CALLING SEQUENCE

Request Code 13



where -

RC Request code for get-file is 13

c Completion address, address of core location to which control transfers when an I/O operation is completed. If omitted, no completion routine is scheduled. c in parentheses, represents an index to system library directory, indicating the program to be executed upon completion of an I/O operation. Use of this option {parentheses} by unprotected programs results in job termination.

Completion routines are operated by threading the I/O requests on the schedule thread.

x	c	Meaning of c
-	{c}	c in parentheses represents an index to the system directory; x has no meaning.
0 or blank	c	Completion address is c; a jump to c will be made.
≠ 0 and ≠ blank	c	c is a positive increment that is added to the address of the first word of the parameter list to form the actual completion address.

Address increment, a positive increment that is added to the address of the first word of the parameter list to form the address of the first word of a three-word block containing the ASCII name of the file. Locations before and after the request can be accessed because of wraparound in adding. If i is in parentheses, the parameter is the actual address of the first word of a three-word block containing the file name.

Starting address of the block into which the file, or portion of the file, is to be transferred. s and x are related as follows:

x	s	Meaning of s
= 0 or blank	s	s is the starting address.
≠ 0 and ≠ blank	s	s is the positive increment that is added to the address of the first word of the parameter list to form the starting address. Wraparound allows the starting address to be ahead of or behind the parameter list.
= 0 or blank	{s}	s is a core location which contains the actual starting address.
≠ 0 and ≠ blank	{s}	s is a positive increment that is added to the address of the parameter list to form the address of a location containing another positive increment. This second increment is added to the address of the first word of the parameter list to form the actual starting address. Both increments may reference locations ahead or behind the parameter list because of wraparound in adding. Return from the request processor is to the location following {s}.

w1,w2	First and last words, if only part of the file is desired. The parameters must be blank if the entire file is to be used. An attempt to read beyond the end of the file is honored if it does not destroy the system.
x	Relative/indirect indicator.
cp	Priority of completion address, the level at which the completion address is to be executed. Always one for unprotected requests.
rp	Priority of mass storage requests needed to complete a GTFILE request. Always Zero for unprotected requests.
p	Protect Processor sets bit 12 if unprotected request.

SYMBOL	Meaning
v	Error code passed to completion address in bits 13-15 of Q and set in the request by the system at completion. The settings of v are indicated below.
000	No I/O error - n words transferred
001	Unused
010	No I/O error - less than n words transferred
011	Unused
100	Unused
101	I/O error - n words transferred; error due to device failure
110	I/O error - less than n words transferred; error due to device not ready.
111	I/O error - less than n words transferred; error due to device failure

The last two words of a get file contain the sector number of the first word of the file. If this number is unknown, the two words should contain zero. In this case, the ▽GTFILE▽ searches for the file and stores the starting sector number in the two words.

Thread This contains a non zero value if stacked or currently being processed. The thread will be cleared before scheduling completion routine.

34.6.2 INTERNAL FLAGS

T13IOC Absolute location of get-file

GFBUSY Get-file busy switch, pointer to current parameter list location

GFTHED Get-file thread pointer to location of next request parameter list.

34.7 GENERAL DESIGN PECULIARITIES

The T13SUP routine will be overlaid by the GTFILE buffers.

34.8 PROGRAM LOGIC

The initial entry to T13 is made at EREQST. A jump is then made to the T13SUP routine which saves the absolute address of T13 in FILE2 in TRVEC. This entry point address of T13 is then calculated and stored in the RCTV table in NMONI. The MIB flag is cleared and control is transferred to the breakpoint program if requested. If breakpoint is not requested a jump is made to the execution address brought in the Q register.

Upon entrance to get-file, the thread in the parameter list is set to $FFFF. The get-file busy switch is checked; i.e., if non-zero, it contains the address of current get-file request processing. If busy, a check is made to see if requests are already threaded. If not, ▽GFTHED▽ is set to the request location. ▽GFTHED▽ defines the location of next request to be processed. If ▽GFTHED▽ contains an address pointing to a request, the thread in this request is checked for $FFFF, if address specified, next request is checked and so on. If thread contains $FFFF, the thread is replaced with address of request to be processed. After setting up Thread, an exit is made to the request exit (REQXT) module.

Execution of Get-file requests are broken down into three parts.

1. Get file parameters are computed. The Program Directory is searched for given name of a permanent file on the library. If W1 and W2 are specified, ▽COMP▽ address is set as Completion address and one

sector is read into the Get-file buffer. If the complete file is specified {no W1 and W2 given} ▽COMP1▽ address is set at the completion address and the complete file will be read in. After either Read request is initialized, control is returned to Dispatcher.

2. Upon entrance to ▽COMP▽ data from the get-file buffer will be transferred to request Buffer. If any more data needs to be transferred COMP1 address is set as completion routine and a Read request is made to bring rest of file into request buffer, after request is initialized, control is returned to Dispatcher.

3. ▽COMP1▽ calls request completion routine, checks for threaded requests, clears flags and returns to user's program if not threaded. If a request is pending it is set up to be processed and group one above is repeated.

ERROR CHECKS

The following errors will cause the message J02, hhhh to be printed out and control passed to the Job Kill portion of Job Processor.

1 No sector number and file name specified.

2 A sector number given but no file name or start and stop location specified.

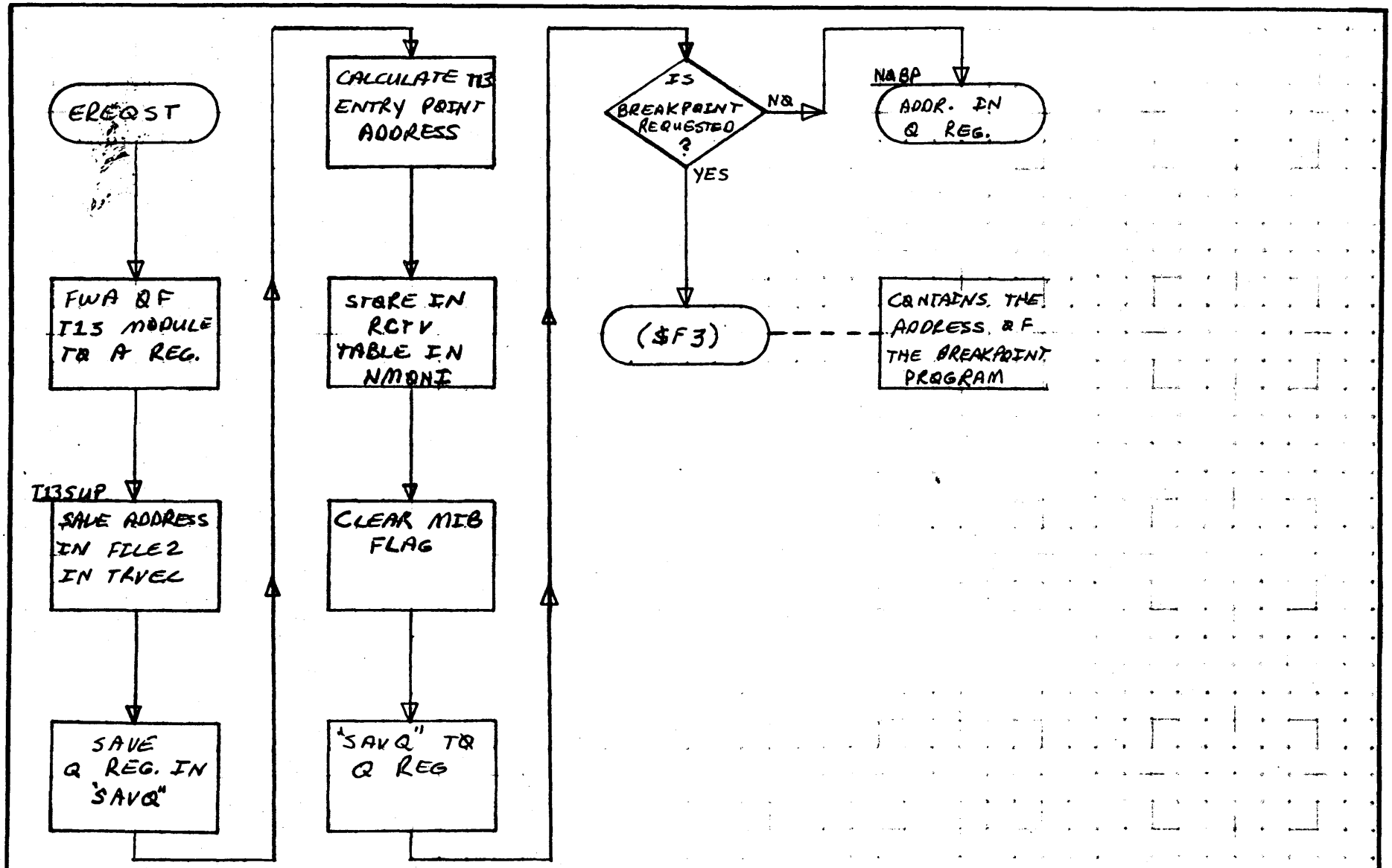
3 W2 specified {last word} and is greater than number of words in file.

4 Unprotected request exiting to protected locations.

5 No Library Directory.

6 Name not in library directory.

7 If request already threaded {SCHERR}



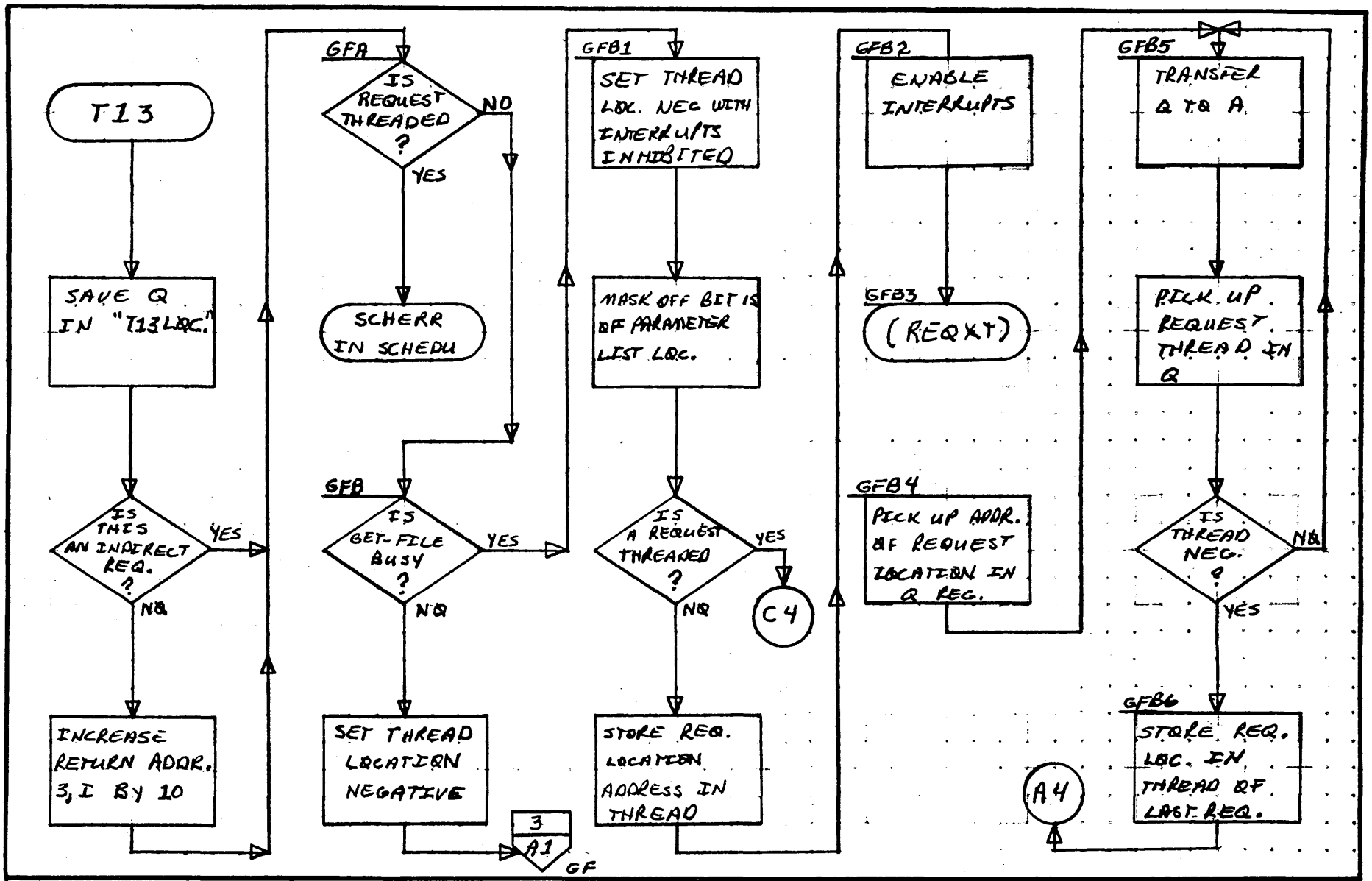
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE T13 - GETFILE REQUEST		PROJECT MGR.		
FLOWCHART ☐	PROCESSOR	PAGE 1 OF 9	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 3 1971

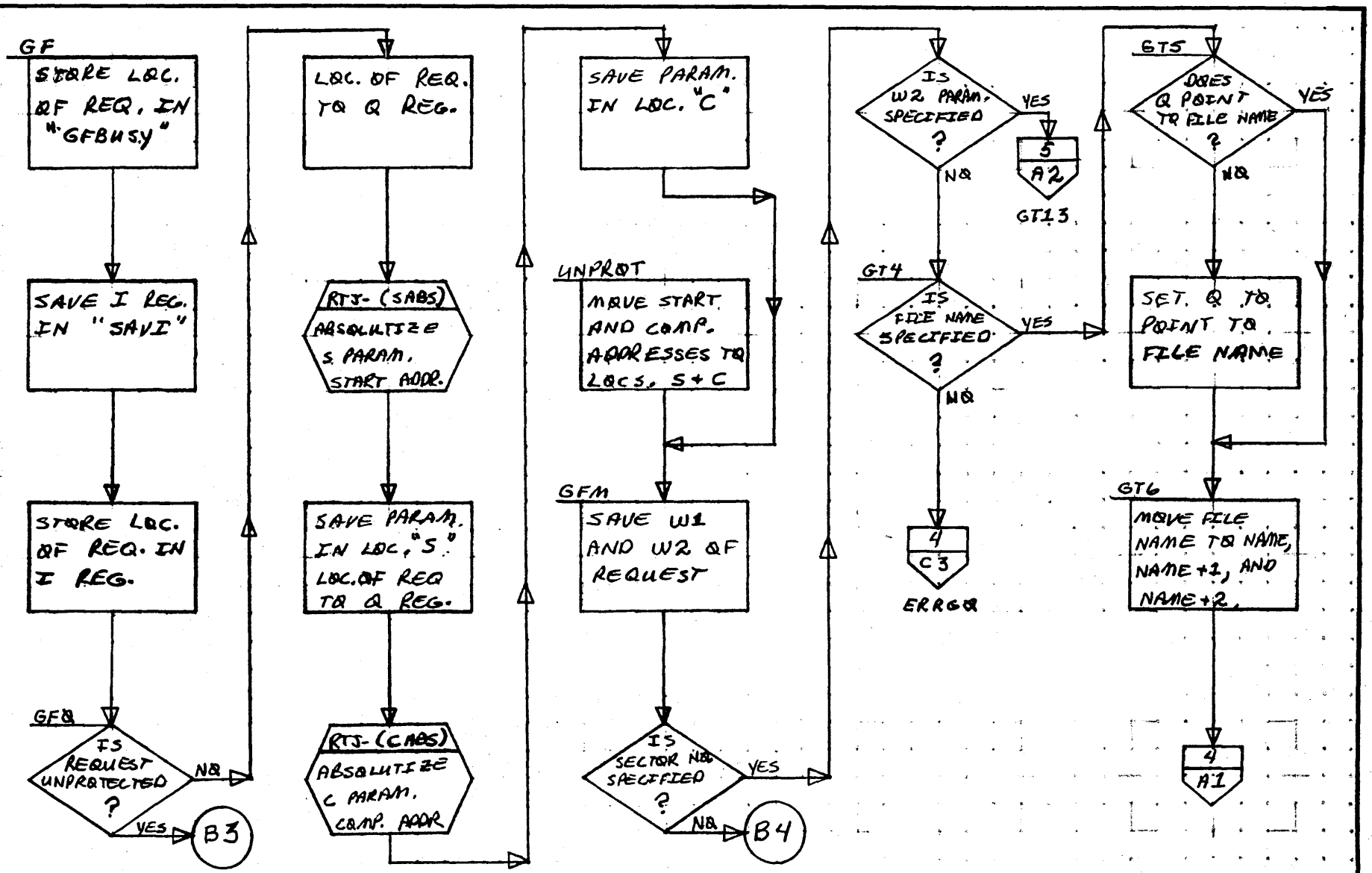
3h-7



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE T13 - GETFILE REQUEST	PROJECT MGR.			
	PROCESSOR PAGE 2 OF 9	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

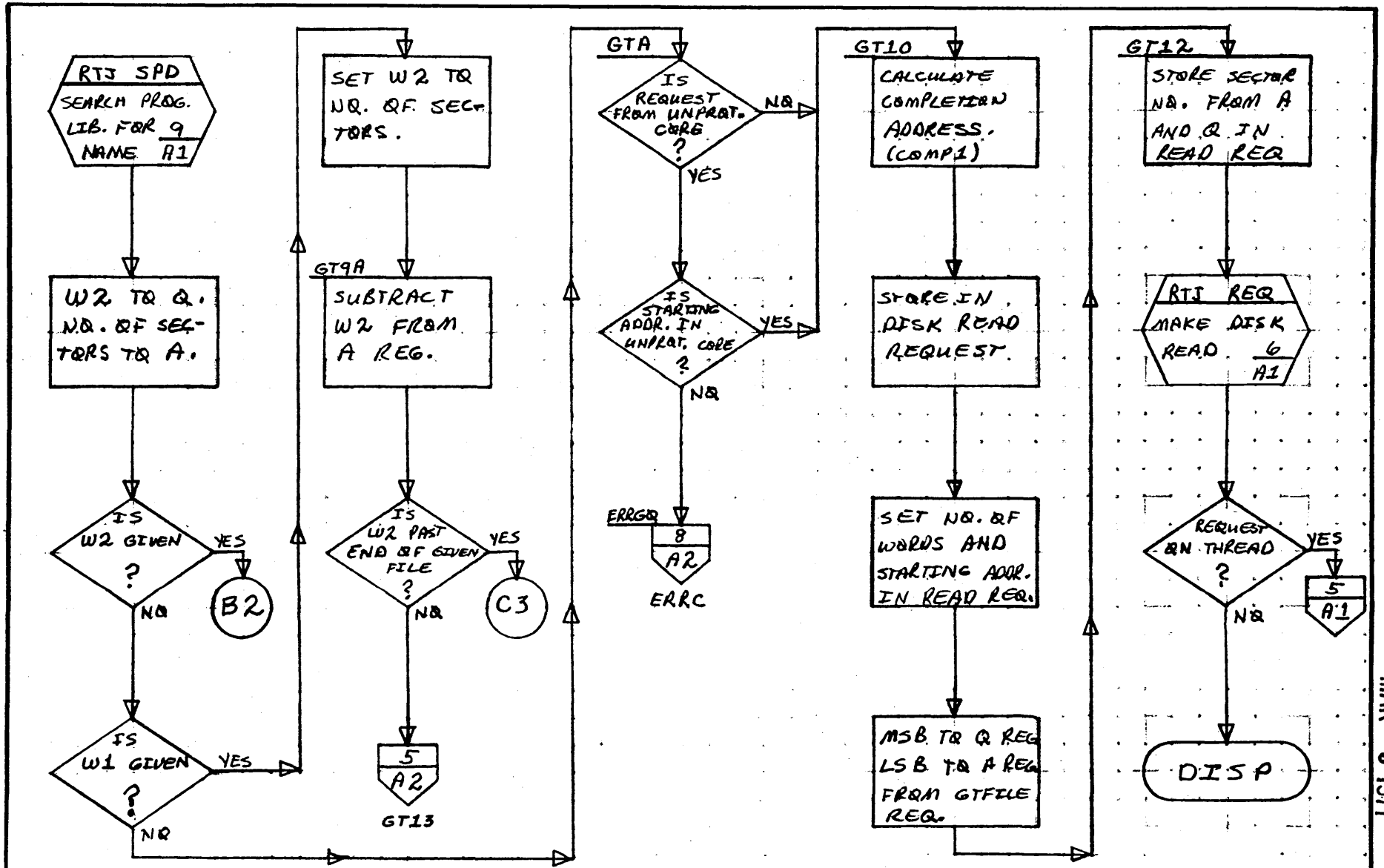
3h.8



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T13 - GETFILE REQUEST		PROJECT MGR.		
	PROCESSOR	PAGE 3 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

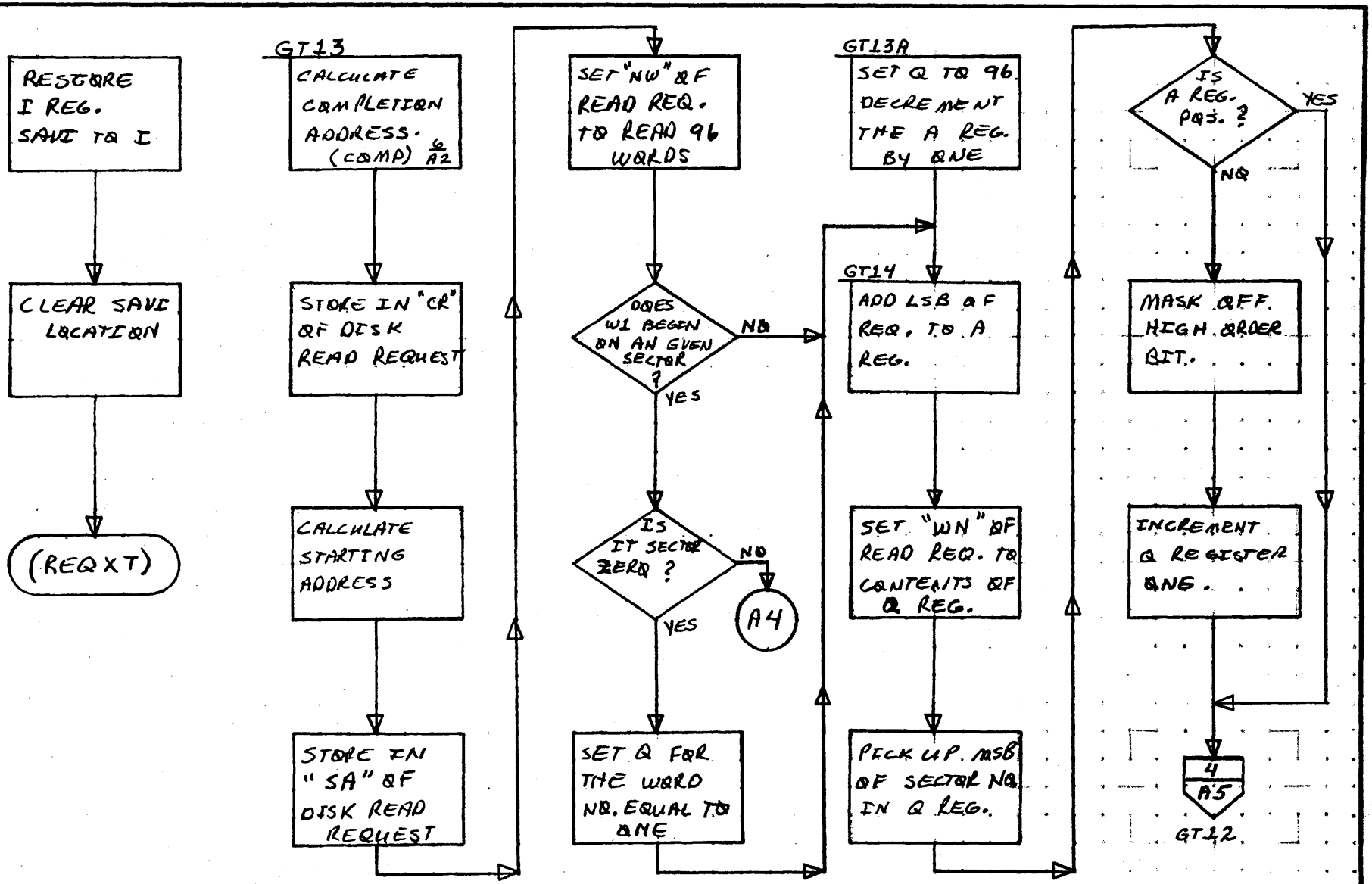
3h.9



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T13 - GETFILE REQUEST		PROJECT MGR.		
	PROCESSOR	PAGE 4 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

34.10



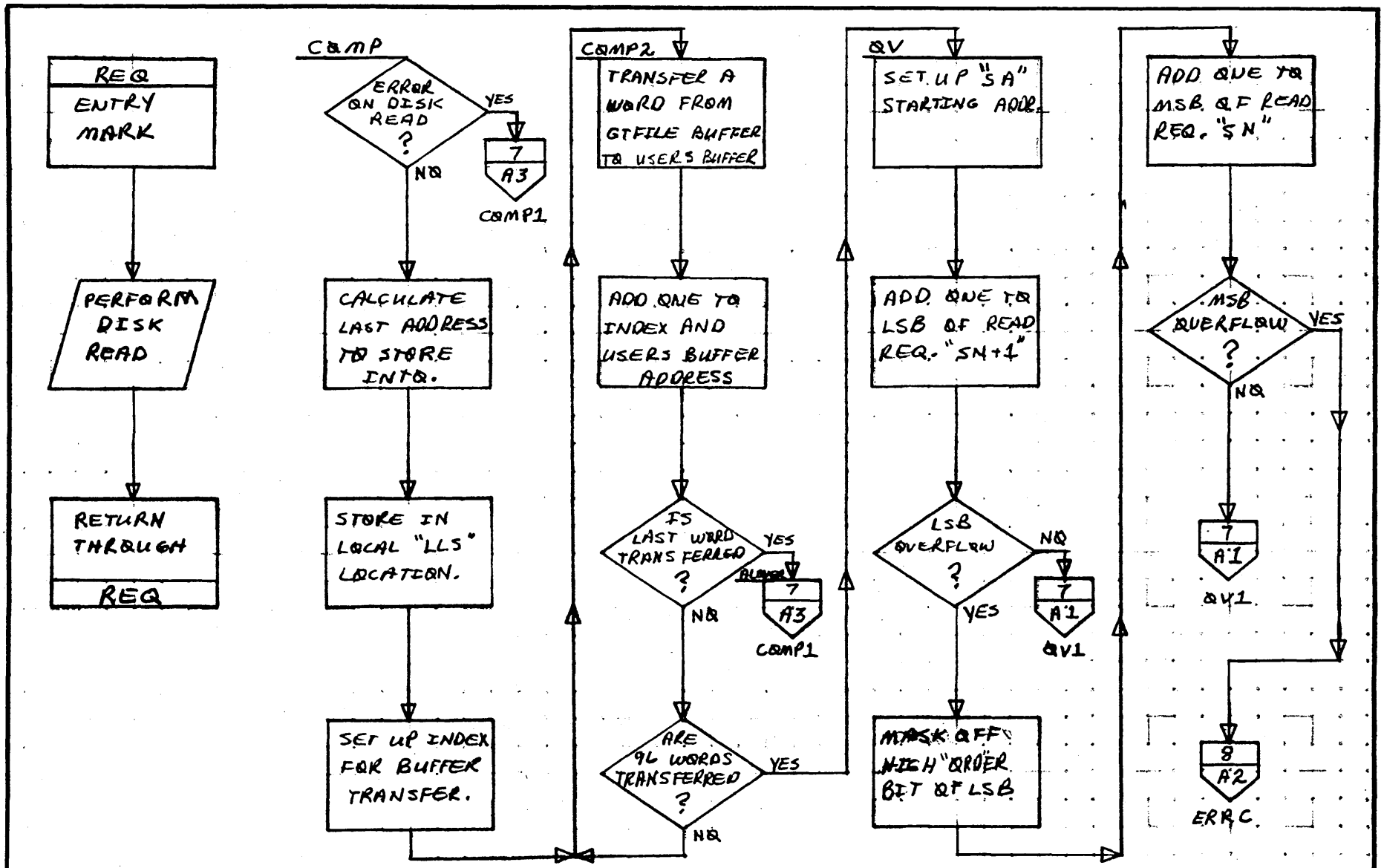
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T13 - GETFILE REQUEST PROCESSOR	PAGE 5 OF 9	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

34.11

AA1385 (FORMERLY CA127-1)

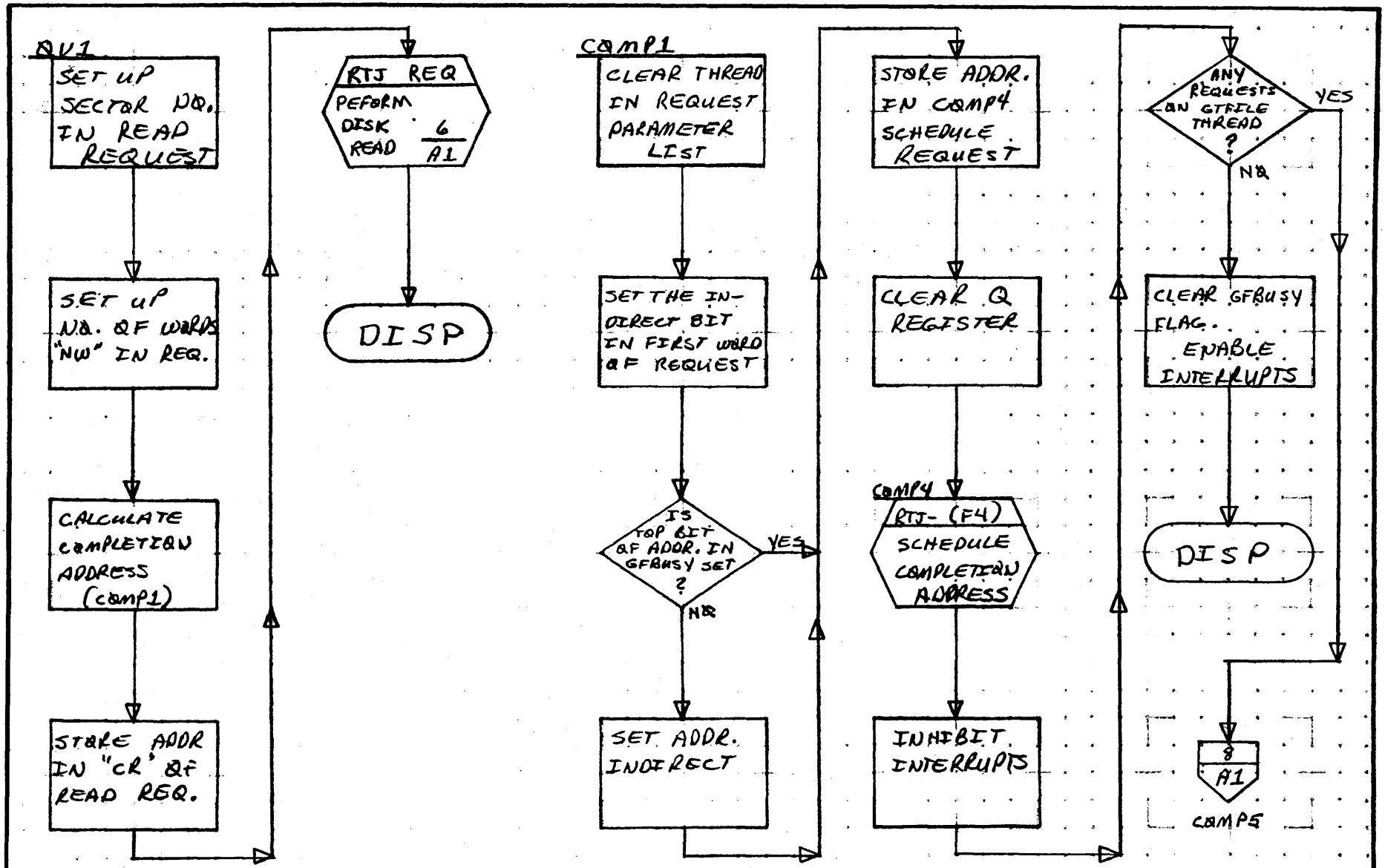
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE T13 - GETFILE REQUEST		PROJECT MGR.			
	PROCESSOR	PAGE 6 OF 9	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

34.12



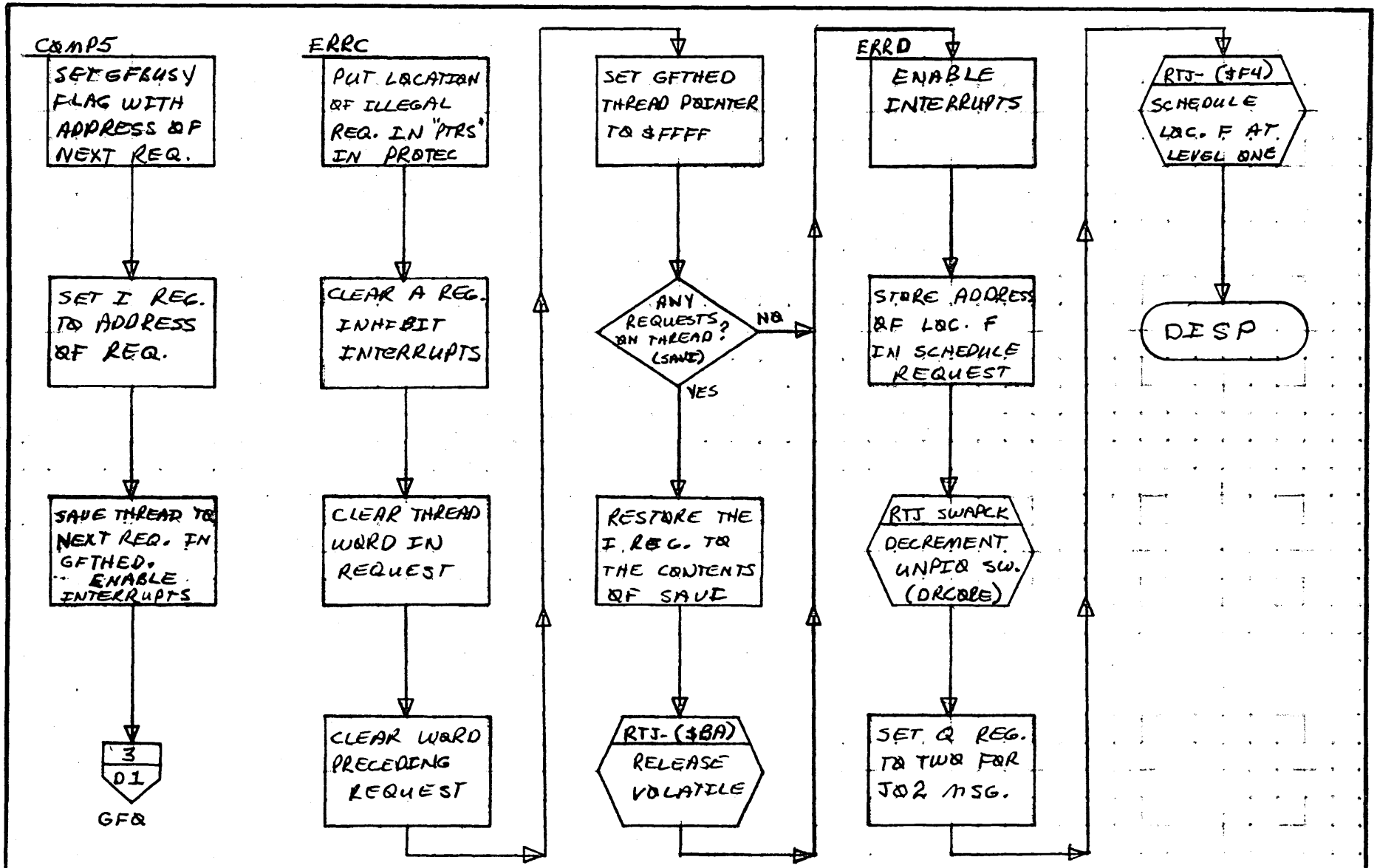
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE T13 - GETFILE REQUEST		PROJECT MGR.		
	PROCESSOR	PAGE 7 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

34.13

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



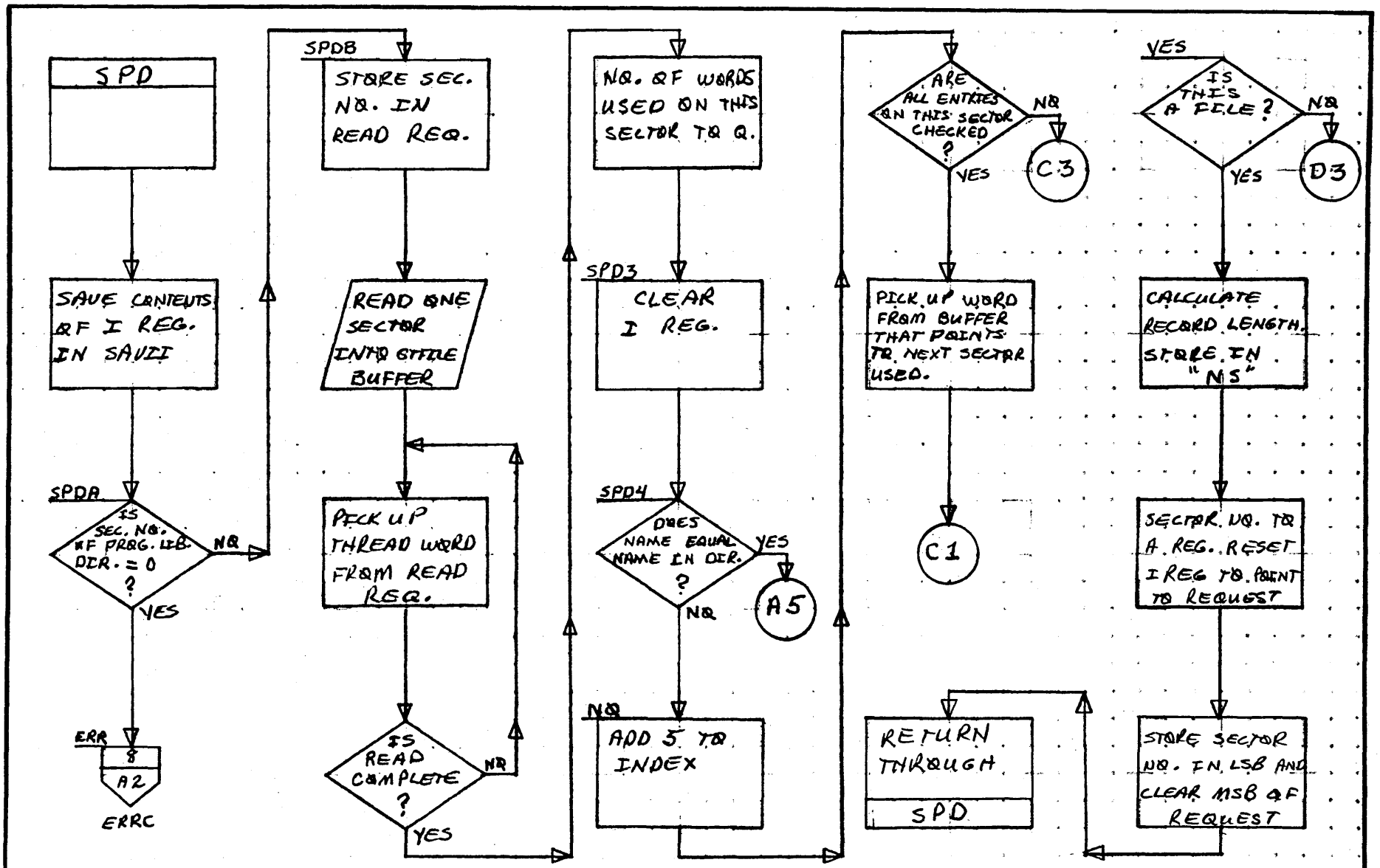
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE T13 - GETFILE REQUEST		PROJECT MGR.			
FLOWCHART ☐	PROCESSOR	PAGE 8 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

3h.1h

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE T13 - GETFILE REQUEST	PROJECT MGR.			
	PROCESSOR PAGE 9 OF 9	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

34.15

35.0 LIBRARY EDITING - LIBEDT

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued

35.1 PROGRAM FUNCTION

The library editing program is a mass storage resident program that is called into execution in protected core by the Job Processor to perform the following functions:

a. Listing the program library directory
b. Listing the system library directory
c. Absolutizing and punching relocatable binary input
d. Replacing programs in the system library directory
e. Deletion, replacement and addition of programs in the program library
f. Deletion of permanent files from the program library
g. Insertion and update of files in the program directory
h. Setting request priorities in the system directory.
i. Transfers of records between any two peripheral devices.

35.2 ENTRY-POINT NAMES

LBDT

35.3 EXTERNALS AND DESCRIPTION

FILE2 - Location in TRVEC which contains the absolute starting address of LIBEDT when it is in core.

SWITCH - Flag in TRVEC which is checked by LIBEDT before most input or output to determine if the job is to be terminated.

UNPIO - Location is TRVEC which contains the number of unprotected I/O requests in progress.

SWAPCK - Routine in DRCORE used to decrement the number of pending unprotected I/O requests.

LOGLA - Logical unit table (indexed by logical unit) which contains the core location of the physical device table entry corresponding to lack logical unit.

NUMLU - Number of Logical Units in SYSBUF

PCOMFL - Protected Common Flag in TRVEC used by the loader to insure Protected Common is not entered during a loader operation.

LIBEDT - Entry for LIBEDT in the System Library Directory

35.4 ENTRY INTERFACES

LIBEDT is scheduled for execution by JOBENT in response to a *LIBEDT statement from the Job Processor or the Manual Interrupt Processor. Upon entry LIBEDT's starting address is stored in FILE2 in TRVEC to be used by the JOB Processor to release LIBEDT when its operations are completed. The Q register contains the return address to the JOB Processor and is saved upon entry.

35.5 EXTERNAL INTERFACES

Exit from LIBEDT will occur for any of the following three conditions:

a) a *Z command is presented from the input comment device

b) the JOB KILL switch in TRVEC is set.

c) a fatal error during an overlay operation within LIBEDT.

In all cases exit is made by a jump to the return address in the JOB Processor saved upon entry.

35.6 GENERAL PROGRAM INFORMATION

35.6.1 CORE AND MASS STORAGE LAYOUTS OF LIBEDT

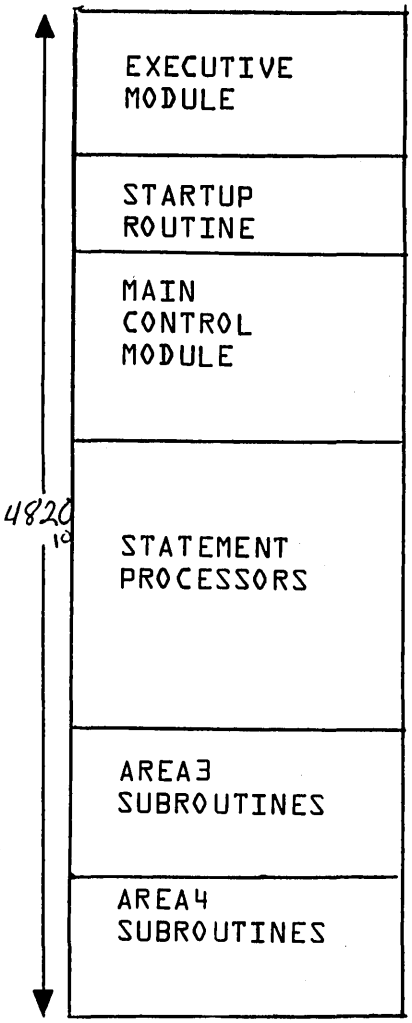


Figure #1
MASS STORAGE LAYOUT

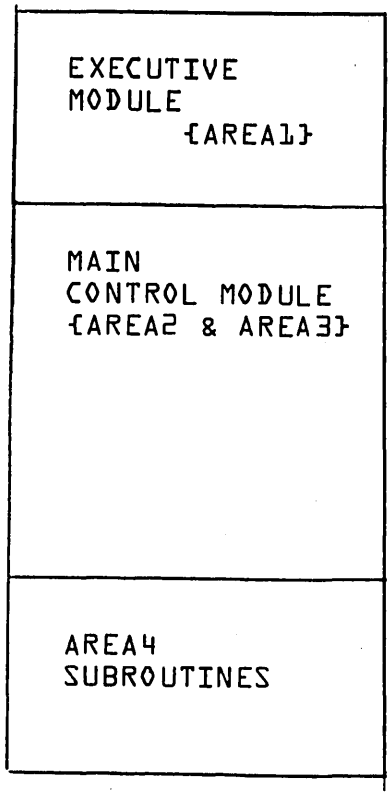


Figure #2
CORE LAYOUT WITH MAIN CONTROL MODULE ACTIVE

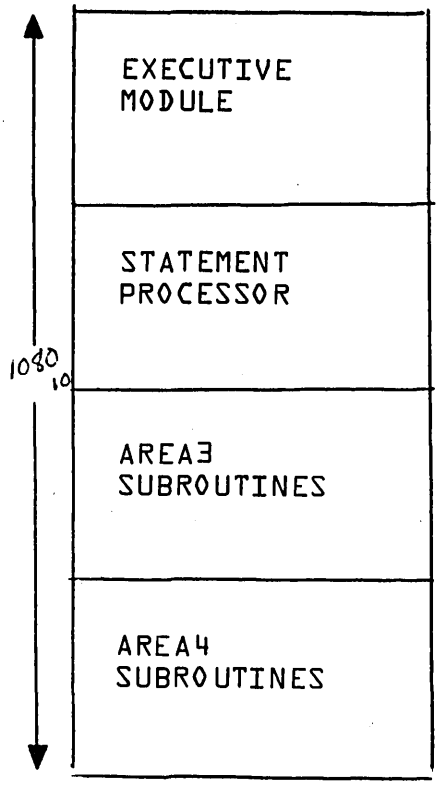


Figure #3
CORE LAYOUT WITH A STATEMENT PROCESSOR ACTIVE

35.6.2 AREA3 AND AREA4 SUBROUTINES

Group No.	Subr. M.S. ADDR.	RELOC. FACTOR	SUBROUTINE INCREMENT	SUBROUTINE NAME
1	MASLOC-LBDT	MSLOC	0	MASLOC
			MOVIT1	MOVIT
A			RDIN1	RDIN
R			SETCR1	SETCRR
E			GETCR	GETCOR
A			RBLOD1	RBLOD
3			LOAD1	LOAD
			MVCST1	MVCST1
			ST4Q6A	ST4Q6
2	INSERT-LBDT	INSREL	0	INSERT
			UDSET	UPDSET
			IOBLE1	JOBLE
			ZAES1	ZAES
			ZER1	ZER
3	SENDC-LDBT	SENDC1	0	SENDC
			LOGUT	LUCHK
4	SAC-LBDT	SACREL	0	SAC
			SACF1	SACF

Group No.	SUBR. M.S. ADDRESS	RELOC. FACTOR	SUBROUTINE INCREMENT	SUBROUTINE NAME
5	SEARCH-LBDT	SERCH	0	SEARCH
			FELD1	FELD
6	INPUT-LBDT	INPREL	0	INPUT
7	MASSUP-LBDT	MASUP	0	MASSUP
			SETC	SETC1
8	EES-LBDT	EESREL	0	EES
1	MFREAD-LBDT	MFRREL	0	MFREAD
A			MFWRT	MFWRITE
R			OPNEW1	OPNEW
E			RSAT1	RSAT
A			WSAT1	WSAT
4			SUR1	SUR
			ONSE	OINSE
2	PPRK-LBDT	PPKREL	0	PPRK
			ZOUT	ZRQOUT
3	READIN-LBDT	RINREL	0	READIN
4	KWRITE-LBDT	KWRIT	0	KWRITE

35.6.3 ACCEPTABLE LIBEDT CONTROL STATEMENTS

CONTROL STATEMENT	INTERPRETATION CODE	ENTRY POINT NAME
¤DL	0	LISTPD
¤DM	1	LISTSD
¤V	2	SYSINP
¤U	3	CONCTL
¤Z	4	LIBXIT
¤K	5	CHANGE
¤P	6	PGHABS
¤M	7	SLINSN
¤L	8	PLINSN
¤N	9	FILE
¤S	10	RPUPDT
¤T	11	COPY
¤R	12	PLINSN

35.6.4 LIBEDT ERROR MESSAGES

L01 More than six characters in a parameter name presented to the library editing program.

L02 More than 6 digits in a number presented to the library editing program.

L03 Improper system directory ordinal presented to the library editing program.

L04 Invalid control statement presented to the library editing program.

L05 Illegal field delimiter in a control statement presented to the library editing program.

L06 Illegal field in control statement presented to the library editing program.

L07 Errors in loading as a result of a library editing program control statement.

L08 A program to be added to the program library has an entry point duplicating one already in the directory.

L09 Standard input failed on first input record following an *N request.

L10 The operator is deleting a program which is not in the library.

L11 No header record on file input from mass storage.

L12 On an *L, entry statement, either there was an input error or the first record was not a NAM block.

L13 Common declared by the program being loaded exceeds available common.

L14 Program being loaded is longer than the size of unprotected core to the top of core.

L15 Illegal input block encountered, last program stored in library is not complete.

L16 I/O input error occurred last program stored is not complete.

L17 *L program being installed exceeds the capacity of LIBEDT to input from mass storage.

L18 Attempt to load a zero length program during a *M function.

35.6.5 INTER MODULE COMMUNICATION STORAGE IN THE EXECUTIVE MODULE

35.6.5.1 DEFINITION OF STORAGE

35.6.5.1.1 STORAGE ADDRESSABLE THROUGH I REGISTER

XXK LIBEDT statement deparameterizing buffer - 7 words in length

ØS Word count indicator for system directory listing processing

TS Word count indicator for program directory listing

EMTK Indicates if output is blocked or unblocked.

ERCA Absolute address of LIBEDT I/O tables {IOTAB}

WDCNT Word length indicator for punching absolutized programs

NOWLB Word length indicator for system update program - SLINSN

SECAD M.S. sector address for file inputs - MSB, also temporary storage

SECAD+2 M.S. sector address for file inputs - LSB

SIC Address of current program block being input -MSB

SIC+2 Address of current program block being input -LSB

WS Word length for mass storage FREAD and FWRITE requests

WS+1 Pointer for storage of entry name words in program directory

IOWC Not used

NOWLR Not used

ORCA Not used

ORCA+1 Not used

B1BABS	Temporary address of BUFF1B
NSIX	Constant to set 60 word read or write length
SSIX	Constant to set 96 word read or write length
PIBUFF	LIBEDT statement input buffer - 16 words in length
TERMSW	Terminating buffer indicator for reading input statement
RETAD	Return address in job processor for release of LIBEDT
RA	Bit counter for sector availability table usage
INTMP	Sector address compare for sector availability table usage - MSB
INTMP+1	Sector address compare for sector availability table usage - LSB
ZS	Absolute LIBEDT address of ZSEC
ZS+1	Absolute address of BUFF1C I/O buffer for program library input
ZS+2	Sector update counter
ZSEC	Beginning sector number of program library directory - MSB
ZSEC+1	Beginning sector number of program library directory - LSB
TTEMP	Word counter for statement deparameterizing and Sys. Directory output
TTEMP+1	Word counter for picking entry-points out of the input buffer
HR	Address pointer for directory listing in BUFF1A I/O buffer

ORD	Index to entry being updated in system directory SLINSN
WSK	Not used
WSK+1	Not used
LNOR	First available location in last program directory block allocated
LASTAD	Absolute address of BUFF1C I/O buffer
LSECAD	Absolute LIBEDT address of last
SYSTEM	Sector on which system resides
ADDCT1	Parameter pickup counter for input statement deparameterizing
ERRNNM	Storage for error code to be output by main control module
ENTSEC	Address of first program block input - MSB
ENTSEC+1	Address of first program block input - LSB
ENTSEC+2	Beginning sector address of file being removed for possible $C1 update
SECS	Beginning sector address of S.A.T. - MSB
SECS+1	Beginning sector address of S.A.T. - LSB
SECX	Sector address of currently used sector of S.A.T. - MSB
SECX+1	Sector address of currently used sector of S.A.T. LSB
IOSEC	Not used

IOSEC+1	Sector number of current program or file block being removed
BUF1S	Absolute address of S.A.T. core buffer
BUF1S+1	Not used
SATMP	Contents of SATMP+1
SATMP+1	S.A.T. input buffer word address containing empty sectors
BUFF1A	Absolute address of BUFF1A I/O buffer
BUFF1B	Absolute address of BUFF1B I/O buffer
BUFF1C	Absolute address of BUFF1C I/O buffer
BUFF1D	Absolute address of BUFF1D I/O buffer
SICID	Absolute LIBEDT address of SIC
IOSAD	Absolute address of BUFF1D I/O buffer
NENT	Not used
NORD	Mass storage sector address for reading routine MSB
NSEC	End of program input indicator for PLINSN
PDSN	Mass storage sector address for READIN routine LSB
REMOV	Signals removal operation to PLINSN
REMFIL	Signals file removal to PLINSN
UPDATE	Set if program replacement active
POINT	Pointer to program library entry for this replacement
SAT	Sector address for S.A.T. operations
DUP	Duplicate entry point flag

INPPAR	Holds a 0 or 1 for S.A.T. update operations
IOSEAD	Absolute LIBEDT address of IOSEC
WORD	Current word in S.A.T. sector being examined
SECXID	Absolute LIBEDT address of SECX
CSSAVE	M.S. sector no. of previous program directory sector.
EXTLKA	Indicator for library load to clean up unpatched externals.
W1	First word for file update
W2	Last word for file update
RDREQ	Storage for file input mode indicator
MSINP	Mass storage input indicator
UPDL	Number of sector to be released or reserved in S.A.T.
UPDL1	Compare counter for sector release or reservation S.A.T.
UPDSAT	Beginning sector no. for release or reservation in S.A.T.
TRISEC	Number of update sectors input for file update operation
TRISEC+1	Not used
NUSECT	Program directory entry for new program indicator
CSCNT	Sector counter for multiple input of update sectors
FILIN	Indicator for file module operation

LAST	Eventual M.S. address of last program directory sector - MSB
LAST+1	Eventual M.S. address of last program directory sector - LSB
ASAVA3	Temporary storage of A register for AREA3 overlays
QSAVA3	Temporary storage of Q register for AREA3 overlays
ASAVA4	Temporary storage of A register for AREA4 overlays
QSAVA4	Temporary storage of Q register for AREA4 overlays

35.6.5.1.2 ADDRESSABLE BY PROCESSOR OR SUBROUTINE RELOCATION FACTOR

35.6.5.1.2.1 LIBEDT INPUT/OUT BUFFER ADDRESS°s AND SYSTEM ADDRESS°s

INBFAD - Temporary storage for BUFF1A Address

BUF1A - Temporary storage for BUFF1A Address

BFA58 - Absolute address of word 59 in BUFF1A I/O Buffer

BFA59 - Absolute address of word 60 in BUFF1A I/O Buffer

BFC93 - Absolute address of word 94 in BUFF1C I/O Buffer

BFC94 - Absolute address of word 95 in BUFF1C I/O Buffer

BFC95 - Absolute address of word 96 in BUFF1C I/O Buffer

BFC96 - Absolute address of word 96+1 in BUFF1C I/O Buffer

BFB93 - Absolute address of word 94 in BUFF1B I/O Buffer

BFB95 - Absolute address of word 96 in BUFF1B I/O Buffer

BFA93 - Absolute address of word 94 in BUFF1A I/O Buffer

BFA94 - Absolute address of word 95 in BUFF1A I/O Buffer

BFA95 - Absolute address of word 96 in BUFF1A I/O Buffer

BFC4 - Absolute address of word 5 in BUFF1C I/O Buffer

BFC15 - Absolute address of word 16 in BUFF1C I/O Buffer

BF1BA - Entry location indicator in program directory of duplicate entry point

TTTEMP - Absolute LIBEDT address of TTEMP

35.6.5.1.2.2 LIBEDT INPUT/OUTPUT LOGICAL UNITS TABLE

IOTAB - System scratch logical unit

IOTAB+1 - System library logical unit

IOTAB+2 - Standard input logical unit

IOTAB+3 - Standard binary output logical unit

IOTAB+4 - Standard print logical unit

IOTAB+5 - Standard output comment device

IOTAB+6 - Standard input logical unit and mode from job processor

35.6.5.1.2.3 JOB-KILL ROUTINE ADDRESSES

JKILLA - Storage for return address when JOBKILL not active

JKILAD - Storage for return address when JOBKILL active

35.6.5.1.2.4 OVERLAY PROCESSORS CONTROL LOCATIONS

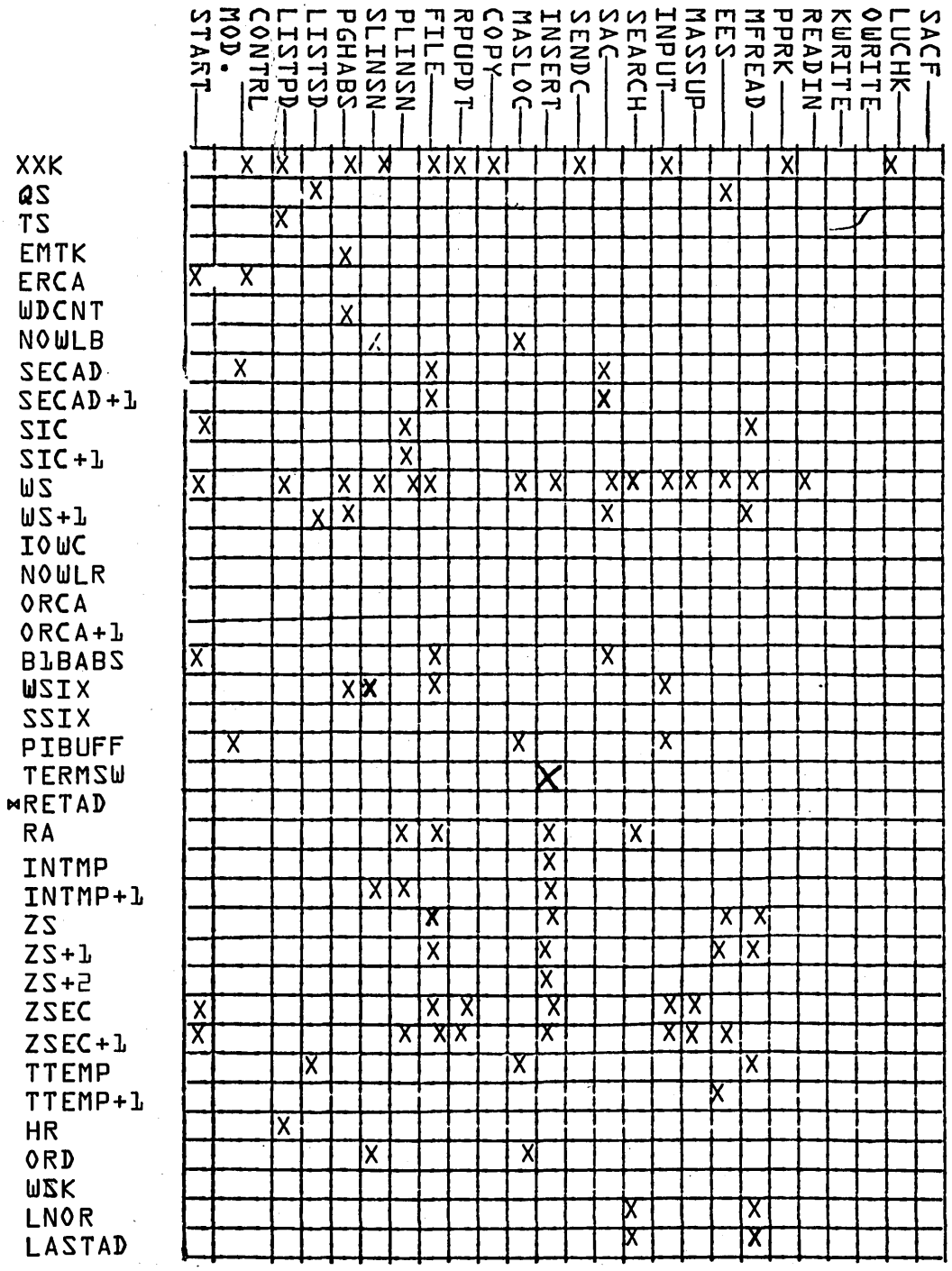
QTEMP - Storage for AREA3 Overlay Subroutine Increment

PROG2 - Subroutine group in AREA3 indicator

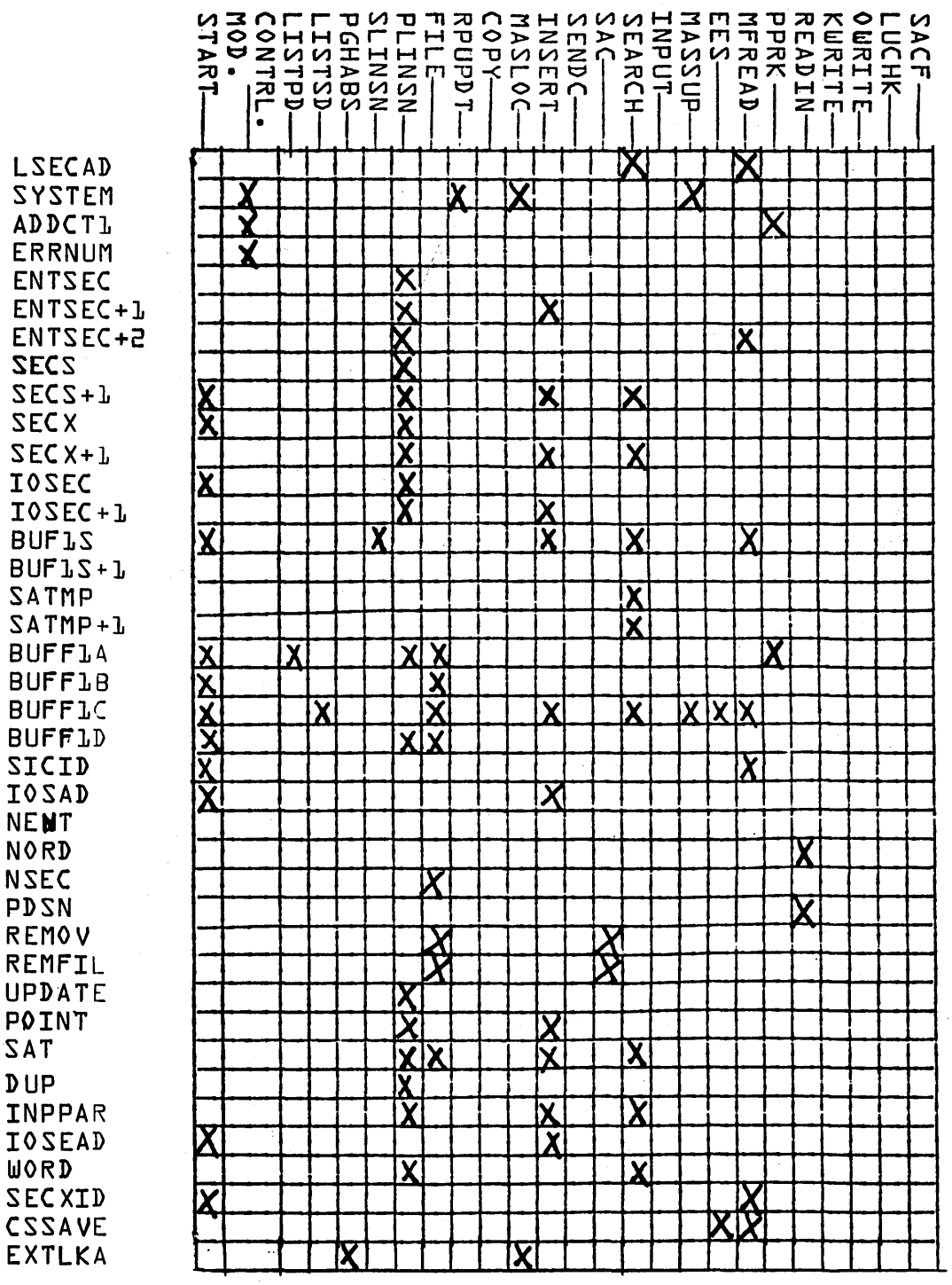
PROG3 - Subroutine group in AREA4 indicator

@TEMP1	-	Storage for AREA4 Overlay Subroutine Increment
PROG1	-	Processor in AREA2 Indicator
@TEMP3	-	Entry indicator for main LIBEDT control module

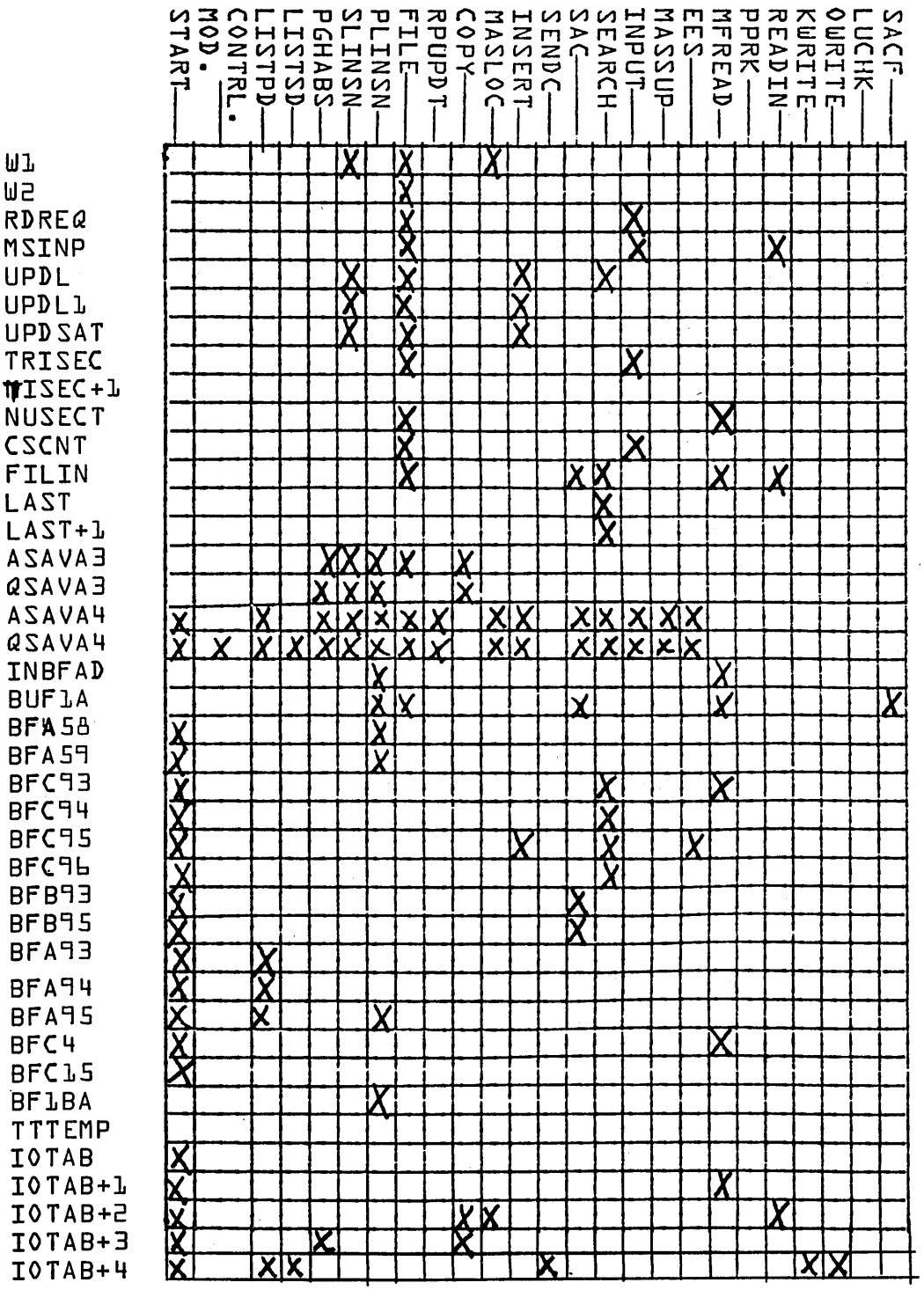
35.6.5.2 CROSS-REFERENCE OF STORAGE USAGE



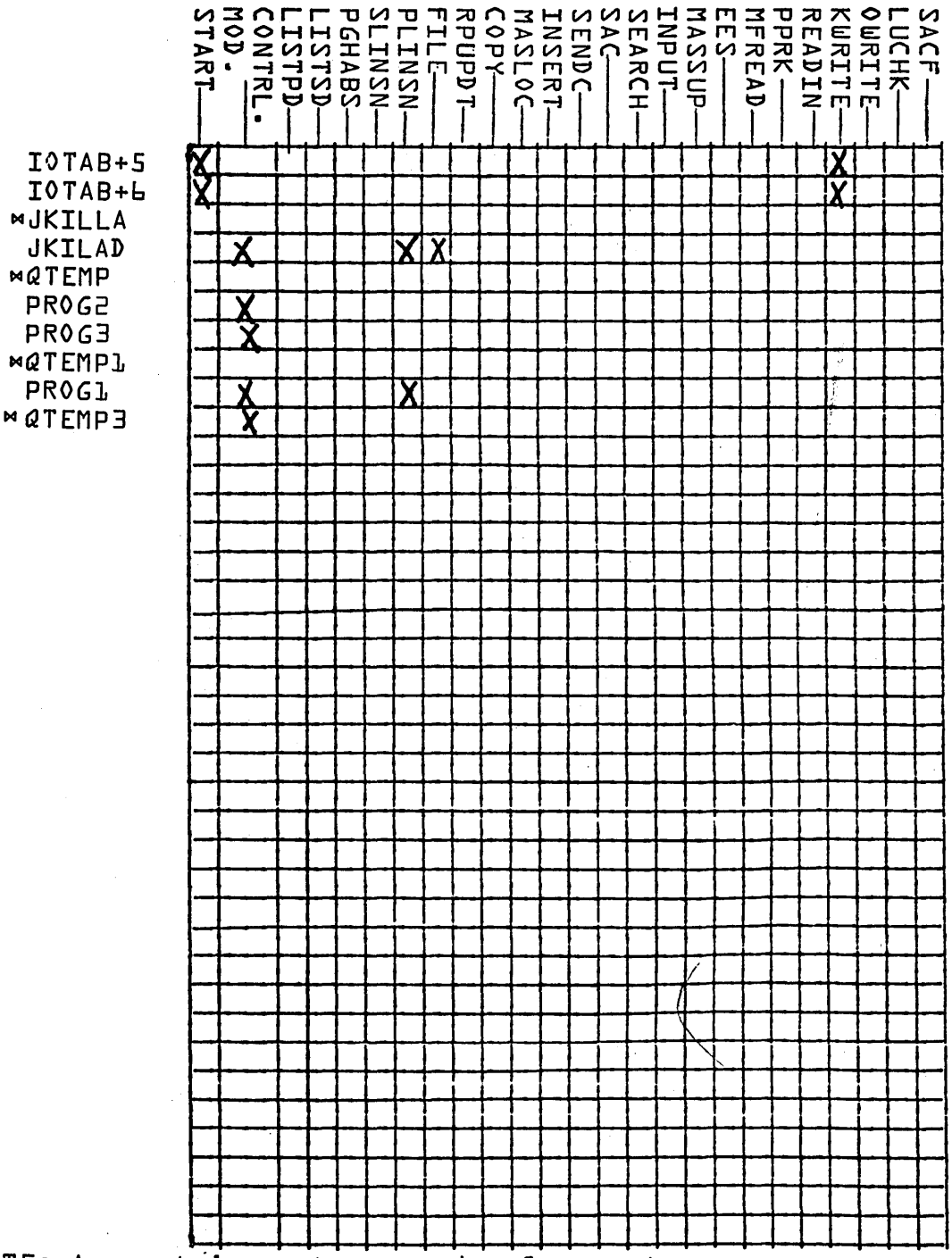
35.6.5.2 CROSS-REFERENCE OF STORAGE USAGE (Continued)



35.6.5.2 CROSS-REFERENCE OF STORAGE USAGE (Continued)



35.6.5.2 CROSS-REFERENCE OF STORAGE USAGE (Continued)



*NOTE: Any entries not crossed referenced complete all their operations within the executive module. These may however contain valuable debugging information.

35.7 GENERAL DESIGN PECULIARITIES

LIBEDT in an effort to maximize its utilization of protected core uses a series of overlays to cause only the called for statement processor and a limited number of its subroutines to be core resident at any one time during execution. LIBEDT is assembled as one program and loaded on the library by the initializer. To insure installation of LIBEDT in 16K of core, it should be loaded immediately after all essential core resident modules. Once loaded on the library LIBEDT length in the system directory is changed to an actual operating length each time the system is autoloaded through the space module.

The LIBEDT program consists of a resident executive module, a main LIBEDT processing module, statement processors, and subroutines. The executive routines and tables are resident as long as LIBEDT is operational. The main processing module is overlaid into core after LIBEDT has been started up to determine the type of operation to be performed, and to set up the I/O table properly before executing a statement processor. This module is also brought in to process any errors and to pick up the next statement to be analyzed. This module runs in both AREA2 and AREA3. The statement processors are read into AREA2 by the main control module as each is called for and remains there until the main control module is brought back in. All subroutines used by the statement processors run either in AREA3 or AREA4 as they are called for. In most cases, a subroutine group consists of several subroutines that are executed together or frequently by a particular processor. Refer to Figure 1 for LIBEDT mass memory layout and Figure 2 and 3 for its allocatable core structures.

35.7.1 EXECUTIVE MODULE INTERFACE

The executive module which remains in AREA1 until all LIBEDT functions are completed serves as the communications vehicle between the active statement processor and subroutine groups. Since the relative execution locations are not the same in core as they are on mass storage for the main control module, the statement processors, and subroutines; the executive module is addressable in only two ways.

The first method is through the "I" register which serves as a pointer to a series of buffers, tables, counters and mass storage and core address. When referencing any information in this area the reference can be made absolutely through "I".

For communications with any location other than those controlled by the I register it is necessary to use a relative corrective increment which is added to the desired address in the control module. This relative increment must appear at the beginning of the area that is to be overlaid, and serves to make operations in overlays relative to the area in LIBEDT where it is to be executed.

A second feature of the executive module communication necessitating observance is the method of overlaying and passing information to subroutines from main processors or from subroutine to subroutine. In the executive module there are two tables (section 35.6.2) which contain the relative address of each subroutine group to run in AREA3 and AREA4. To bring the proper subroutine into execution in either AREA3 or AREA4 the following procedure must be performed.

First any information to be passed to the subroutine can be saved in LIBEDT's executive table locations labeled QSAVA3 and ASAVA3 for AREA3 and QSAVA4 and ASAVA4 for AREA4. The A and Q registers will be loaded with these values after the subroutine has been overlaided. Once this operation is completed the A register is loaded with the number of the subroutine group desired in the particular area and Q is loaded with the increment to the entry point of the subroutine requested within this subroutine group. The first subroutine in each group, since it is at the beginning of the area does not necessitate an increment and therefore the Q register should be entered with zero (0).

After these parameters are set entry can be made to the overlay processor for the desired area through a return jump. The return address will be set in the first word of the selected subroutine for exit back at P+1 of the location from which the subroutine overlay call was made.

35.7.2 PRECAUTION TO OBSERVE DURING MODIFICATION TO LIBEDT

LIBEDT with its present structure is designed to utilize a minimum amount of allocatable core without serious degradation in performance. Since this is the case, strict observance should be given to certain key parameters whenever system modification or program corrections are implemented.

Besides proper addressing to the executive module, the primary factor to observe is that the size limits for the various areas are set correctly. The limits for MSOS 3.0 release version of LIBEDT are 1) 300 for AREA2, where the statement processor runs; 2) 200 for AREA3 subroutines; and 3) 200 for AREA4 subroutine. The lengths of these areas are set by EQU's in the beginning of the program and can be changed if necessary. It should be noted however, that whenever these EQU's are expanded or when the size of the executive module is expanded the total length of LIBEDT will be increased and may require changing the operating length of the program put in the system directory at autoload time through the space module. Expansion it in excess of the present size of the Job Processor's requirements (approximately 1650 words) may require increase the size of allocatable core where LIBEDT runs. The release version of LIBEDT uses 1090 exclusive of JOBENT.

The main LIBEDT control module can be the size of AREA2 and AREA3 in LIBEDT which at present is 500 words. This length is not controlled by on EQU although one exists for convenience in identifying it. It may, however, be modified by changing the length inserted in the Overlay request at entry point TRANF in the executive module.

The total core length of LIBEDT is calculated by a EQU at the end of the startup routine, as well as the starting address for each area. These EQU's can be checked for convenience in determining correct core allocation.

35.7.3 ADDITION OF NEW STATEMENT PROCESSOR AND/OR SUBROUTINES

The capability of adding new processing operations to LIBEDT can be accomplished with relative ease by observing the following procedures:

1) Set up the mass storage address of the new statement processor at the end of the ADCT address table in the main control module. This address is of the following form:

```
e.g.  ADC     Process-LBDT     14
```

2) Supply an interpretation code at the end of the interpretation code table. This table in the control module has entries M1,M2 .....M13. Two instructions must be placed in the table. A jump to the routine responsible for overlaying the statement processors and an instruction for setting the interpretation code.

```
e.g.  M14     ENQ   13
              JMPM  MG0
```

NOTE: These two instructions will be placed after the M13 entry.

3) Insert a one character control statement name into the table FUNTAB in the control module. If the control statement name is two characters it must be placed in FEND. In either case a jump to the interpretation code routine must also be supplied.

```
e.g.  ALF     1,MW
      JMPM    M14


e.g.  ALF     1,MG0
      NUM     8 47FF
      JMPM    M15
```

To add new subroutines for a new processor or for existing processors the rules to observe are:

For subroutines in AREA3 set the mass memory address of the subroutine in the table of AREA3 subroutines in the executive module.

```
e.g.  ADC    LOADUP-LBDT     5
```

The position in the table will determine the value entered in the A register (this example 5) to call the subroutine group. The rules for AREA4 subroutines are the same except the mass storage address is placed in the table of AREA4 subroutines in the executive module.

35.8 PROGRAM LOGIC

35.8.1 EXECUTIVE MODULE

After being scheduled by the JOB PROCESSOR the executive module of LIBEDT is entered to initialize all tables, buffer areas, and address to be used in inter-module and system communications.

Upon entrance the starting core address of LIBEDT is saved in FILE2 in TRVEC to be used by the JOB PROCESSOR for the release of LIBEDT. The return address in the JOB processor where the release is performed is passed in the Q register and saved. The I register is next setup to serve as an index to a series of buffers, tables, and address storage locations used between processors and subroutines. (section 35.6.5). Exit is next made to a one time startup routine which sets essential information to commence processing. This includes LIBEDT's I/O logical unit table, the input/output buffers in unprotected core and necessary addresses within these buffers, temporary unprotected core limits, and initialization of the Sector Availability Table parameters. LIBEDT next sets the correct job mode indicator in the I/O table and prints the message LIB on the standard comment device. Before continuing to overlay the main control module the startup routine completes it's final task which is to calculate the sector on the disk where the system resides and save this address.

35.8.2 MAIN LIBEDT CONTROL MODULE

This module which resides in both AREA2 and AREA3 is responsible for processing communications between the operator and LIBEDT. This includes notifying the operator that the program is ready to initiate a new process by output the IN message, notification of any error conditions, interrogation of comment device and processing of all input statements, and execution of MK, MV and MU statements which alter standard logical units.

Once in core this module can be entered at three different locations. The actual entry point is determined by the value entered in the Q register when exiting to perform the overlay to bring in this module. A zero (0) in Q signals that processing is proceeding as specified and entry should be at RPSA. A one (1) in Q notifies the operator that the job kill switch was set during processing and entry should be made at RPSX to exit directly to the Job Processor. A three (3) in Q means an error condition occurred and entry is to be made at NEXT1 to process the error before returning with the IN message.

On initial entry to the main control module the IN message is output to the comment device when LIBEDT is ready to read a control statement from the comment or input device. If reading the device causes an error return, the comment device will automatically be re-read for a correct or new control statement.

The acceptable control statements to LIBEDT along with their entry points and operation interpretation codes supplied by LIBEDT are given in section 35.6.3. The interpretation code is stored in a executive module cell called PROG1, which can be checked to determine which processor is in during debugging. This cell is also used when a statement processor is overlaid.

All control statement names are stored in the table FUNTAB along with a jump to a routine for setting the interpretation code. The interpretation code is also responsible for determining the overlay of the proper processor. A list of statement processor entry-point addresses is supplied in the table ADCT. Using the interpretation code as an index, one of these entries is selected and used to calculate the beginning mass storage address. The length for the read is set as 300 words and a jump is made to entry point BRIPR1 to perform the overlay of the correct statement processors. If the size of AREA2 is ever changed this 300 word read length must also be changed.

35.8.3 LIBEDT STATEMENT PROCESSORS

35.8.3.1 LIST PROGRAM DIRECTORY - LISTPD

The routine responsible for listing the program directory is LISTPD. Each entry point and file in the program directory is listed in the following format:

```
ENTRY     SECT.   LSB
FILE      SECT.   LSB     FILE
```

The name of the entry point or file appears first followed by the word °SECT.° To the right of °SECT.° appears the least significant bits of the first mass storage sector onto which the information was stored. A file is distinguished by the word FILE appearing to the right of the least significant bits.

Prior to reading a program directory sector, a test is made to determine if the last program directory sector has been processed. If the last sector has been processed, the message FINI is output on the comment medium and control returns to the main control module. LIBEDT in turn types IN and anticipates a new control statement. If the last sector has not been processed, each entry will be interrogated to determine if it is an entry point, file or null. Entry points and files will be output in the format described previously and null entries will be ignored. A null entry is one in which all words of the entry are zero.

35.8.3.2 LIST SYSTEM DIRECTORY - LISTSD

LISTSD is the routine responsible for listing the system directory. The system directory is listed on the standard output LIST device. The format of the output depends on the contents of the directory. If core resident {4 word} entries exist, they will be listed prior to mass storage resident {7 word} entries. Only those parts of the directory which actually exist will be listed. The main control locations for this routine are:

1. $EB - Core location of the system directory
2. $E7 - Index to first mass storage entry
3. $E6 - Length of the system directory

The number of core resident entries is obtained by dividing the contents of $E7 by four. The number of mass storage resident entries is obtained by computing the difference {$E6} - {$E7} and dividing by seven. If either are zero, they will not be listed.

35.8.3.3 PUNCH ABSOLUTE RECORDS - PGHABS

PGHABS is responsible for reading in, absolutizing and punching out programs in a single record of program length or in multiple records of length 96 words. Upon entering PGHABS, the next field is scanned for an F. If an F is found, multiple records of length 96 words will be punched. Relocatable binary programs are loaded by the relocating loader which has the capability of loading onto mass storage. Programs on the input device are first loaded via a relocatable binary load as a result of a request to the loader. A subroutine load is then performed in an attempt to patch all unpatched externals. Any unpatched externals at the conclusion of subroutine load are then listed by the loader. At this point the operation may be continued by typing in an M cr or may be deleted by typing in an MT cr.

After loading has occurred, control returns to PGHABS which must punch the information. PGHABS first determines whether the loader placed the program{s} on mass storage. If the program{s} reside{s} on mass storage it will be moved from there to unprotected core. If the program {s} was not placed on mass storage, it already appears in unprotected core. In either case a test is next made to determine if one record or multiple records are to be punched. If one record is to be punched, the starting address of the output buffer is set to the lowest unprotected location {$F7}+1 and the length of the record is set to the length of the load {$ED} - {$F7}. A test is then made to determine if the job has been killed by the job processor. If the job has been killed punching will not occur. If the job has not been killed PGHABS determines if the absolutized file is to be output on mass storage. If this is the case, PGHABS outputs a three word header record containing the word count followed by a two word identification field containing ^LIBE^. The header record is used by the FILE

module in determining the number of sectors to read from mass storage. The punch driver is then entered and the program reads output.

If 96 word records are desired, the length of load is divided by 96 yielding "n" 96 word blocks and one block R where 0<R<96. The same procedure described for a single block is applied when punching multiple 96 word blocks, except that prior to punching a new block, the output buffer pointer is advanced by 96 words. After all punching has occured, control returns to the main control module. This is signalled by IN appearing on the comment medium.

35.8.3.4 SYSTEM LIBRARY UPDATE - SLINSN

The SLINSN program is responsible for updating programs in the system directory. On entry to SLINSN, a control statement test is made to determine if a numeric ordinal to an entry in the system directory appears. If a non-numeric or no ordinal is found, L6 appears on the comment medium. If a numeric ordinal is found, a control statement test is made for a numeric sector number. If a non-numeric sector number , L6 appears on the comment medium. If no sector number or a sector number is found, it is saved and the next field interrogated.

The next parameter checked for is the {d} data base. If no field is specified processing passes to an examination of the M parameter. When the {d} is present, however, the program first sets a flag to signal that the job kill switch should be unset upon return from the loader to allow processing of this parameter. The job kill flag is set since E4 loader errors occur when setting the data base. These errors are not, however, critical to performing this function. The list output device should be checked to make sure no other fatal loader error{s} occurred. The actual setting of the data is accomplished by making a loader call with a function seven {7} in the A register and the converted data base in the Q register. The data base must be specified to LIBEDT in decimal due to conversion in the parameter unpacking routine PPRK.

If an M appears in the next control statement field, the system directory ordinal is assumed to be in the mass storage part of the directory. If no M appears, a test is made to determine if a sector number appeared in the control statement. If one did, L6 appears on the comment medium. If no sector number was supplied, it is assumed that the system directory ordinal is in the core resident part of the directory and control is transferred to the routine CORRES.

35.8.3.4.1 UPDATE OF CORE RESIDENT ENTRIES

The CORRES routine converts the ordinal input on the control statement to an index to an entry in the system directory.

This is done as follows:
(ordinal number - 1)*4 ORD

If this index exceeds the index to the first mass storage entry in $E7, the L6 appears on the comment medium. If the index is valid, a relocatable binary program load is started. Once all programs have been loaded from the input device, a subroutine load is performed to patch any unpatched externals. At the conclusion of subroutine load all unpatched externals will be printed. To terminate the operation at this point, type an *T cr. Otherwise, type an * cr.

Once control has returned to SLINSN from the loader, a test is made to determine if loading took place onto mass storage. If it did, the program is brought into unprotected core and stored. If not the length of the program is checked to make sure a zero length program was not loaded in which case an error 18 is output.

The program in unprotected core is next moved, with interrupts off, to some area of protected core. Exit is then made from SLINSN.

35.8.3.4.2 UPDATE OF MASS STORAGE RESIDENT ENTRIES

Updating entries in the mass storage resident part of the directory is handled in a fashion similar to its core resident counter part. These exceptions are noted:

1. An index to an entry in the mass storage resident part of the directory is computed.

   (ordinal number -1)* 7 + ($E7) ORD

2. Loading occurs as previously described except that upon completion of the load the N parameter of the LIBEDT call is checked to see if linking to the program library was requested. If requested, make a second loader call to complete this task. The load length is also checked as described for core resident entries.

3. A test is made to determine if the program just loaded will fit on the mass storage sectors of the old program. If the program fits and it is currently in unprotected core, it is moved from unprotected core to the specified sector. If the program was loaded onto mass storage, it must first be moved to unprotected core prior to outputting it onto mass storage on the specified sector.

4. If the program just loaded will not fit onto the sectors of the old program, and loading took place on mass storage, then the scratch sector will be updated to point around the newly taken sector, and the Sector Availability Table (S.A.T.) will be updated to reserve these sectors from any other LIBEDT action. The sectors for the old program will be released from the S.A.T., thus making them available for other LIBEDT action. If the program was loaded into unprotected core, it will be moved from unprotected core to the scratch area and the scratch sector will be updated. In both of the cases under 4., the mass storage sector on which the program resides, as indicated by the system directory entry, must be updated. The mass storage resident $C0 and $C1 are also updated.

5. Under 3. and 4. the word count in the system directory entry is modified to reflect the new program length.

35.8.3.4.3 PROCEDURE WHEN SECTOR NUMBER IS SPECIFIED

When a sector number appears in a control statement, it is converted from decimal to hexadecimal and stored in the system directory entry along with the program length. No tests are made to determine where the program is being stored and no attempt is made to modify C0 and C1. Loading and transferring then occurs as described in 35.8.3.4.2

35.8.3.5 ADD-REMOVE-REPLACE PROGRAMS AND REMOVE FILES-PLINSN

The PLINSN program is responsible for updating the program library. It provides for deletion, insertion and addition of programs to the library. It is also responsible for deleting permanent files from the library. This module is entered as a result of a *L, entry point name or a *R, name, F submitted to LIBEDT. The first function performed by PLINSN is determining the type of control statement presented to it. If the control statement is a *R, a remove flag is set. The next function is the scanning of the name field for an entry point name. If no entry point name is supplied, L6 appears on the comment medium. If the remove flag is on, PLINSN then scans the next field to see if the file parameter was input. If it is present, a flag is set indicating that a permanent file is to be removed. PLINSN then uses the entry point name as a search argument in the program directory, taking into account whether it is searching for a program name or permanent file name. If a match is found, PLINSN enters either the REPL routine or the REPL20 routine depending on whether a program or permanent file is to be operated on. If no match is found, PLINSN checks the remove flag; if it is on L10 appears on the comment medium and PLINSN returns control to LIBEDT. If the remove flag is off PLINSN overlays the NBEK routine.

35.8.3.5.1 ENTRY POINT FOUND IN PROGRAM DIRECTORY

If an entry point name is found to be in the Program Directory, PLINSN enters the REPL routine. REPL first clears all entries in the program directory which reside on the same sector as the entry point found as a result of scanning the Program Directory. REPL then frees all sectors used by the program from the Sector Availability Table. If the remove flag is off, REPL then enters the NBEK routine (see Section 35.8.3.5) to string the new program in the library. If the remove flag is on, REPL enters the XXTA routine to write out the S.A.T. table and update $C0 and $C1 on the mass storage resident system.

35.8.3.5.2 REMOVING PERMANENT FILES FROM THE LIBRARY

If a permanent file is to be removed from the library, PLINSN enters the REPL20 routine. REPL20 obtains the starting sector and file length from the program directory and computes the last sector of the file. REPL20 then deletes the file name from the Program directory and releases all sectors on which the file resides from the Sector Availability Table. REPL20 then enters the XYTA routine to write the S.A.T. table back on mass storage and to update $C0 and $C1 on the mass storage resident system.

35.8.3.5.3 ENTRY POINT NOT IN PROGRAM DIRECTORY

For new programs PLINSN enters the NBEK routine. NBEK searches the Sector Availability Table for empty sectors to string the program blocks through. Each block occupies one sector with words 59 and 60 of the block set to the sector number containing the next block of the program. When the last block (XFR block) is read, words 59 and 60 are set to zero to denote the end of the program string. Each entry point name of the program is compared with all the program entry point names in the directory. If it is unique (no match) the entry point name is placed in the directory along with the sector onto which the first program block was stored. If a duplicate entry point is encountered during the directory scan, NBEK sets an error flag indicating a

duplicate entry point name has been encountered and continues inserting the program blocks into the library until an XFR block is found. If a new program directory sector is needed as a result of placing a new entry point name into the program directory, the Sector Availability Table is searched for an empty sector and the program directory is strung through this new sector. Before exiting, NBEK checks the error flag for duplicate entry point names. If the flag is off, NBEK enters the XYT routine. If the error flag is on, NBEK sets the remove flag on and enters the REPL routine to delete the new program from the Program LIBRARY.

When adding new programs to the library it should be noted that the process for inserting new program blocks on the library is altered somewhat if the new program is being input from mass storage. To insure that program library directory blocks do not wipe out entry point blocks when processing these blocks from mass storage the following alternate procedure is used.

1. All blocks prior to the first entry point block are output block by block as previously described.

2. Once the first entry point block is encountered, all entry point blocks, external blocks and the XFER block are read into unprotected core.

3. These blocks are now put back on the library one block at a time from unprotected core. A search for an available sector is done prior to output of each sector. The SEARCH routine also zeros the sector out in the Sector Availability Table (S.A.T.).

4. If a new program library directory sector is now required when processing the entry point blocks, this block can be obtained from mass storage without destroying any program blocks.

35.8.3.5.4 EXITING AFTER INSTALLING NEW PROGRAM

To exit back to the main control module after installing a new program the XYT routine is entered. This routine first writes out the S.A.T. sector being used to mass storage. Next a check is made to see if the error flag for duplicate entry points is on. If this is the case an overlay is performed to bring the removed portion of this processor back into core to remove the program just installed. (see Section 35.8.3.5.11). After removal an L08 error will appear on the comment medium and control returned to the main control module.

If no duplicate error occurred XYT proceeds to update $C0 and $C1 on the mass storage resident system. Before exiting back to the main control module a check is made to determine if an input error or illegal block error occurred during the input of the program. If either error 15 or 16 occurred the error message will be output. Another attempt can be made to replace or remove this program since the next sector pointer in the last block output was set to zeros before exiting with the error.

35.8.3.6 REPLACE, ADDITION, AND UPDATE OF PERMANENT FILES - FILE

FILE module is responsible for inserting and updating files in the program library. This module is entered as a result of submitting an *N, FILNAM, W1, W2, N statement to LIBEDT.

35.8.3.6.1 CONTROL STATEMENT PROCESSING

The first field processed by FILE is the file name field. This field must contain a non-numeric name of length 6 characters or less. If this field is omitted or is all numeric, LIBEDT responds with an L6 error message. Otherwise, control statement processing continues.

The next fields processed are the W1 and W2 fields. These fields contain the first and last words being updated in a file. These fields may be omitted. If non-numeric information is found in W1 and W2, the message L6 is written. If W1 and W2 are specified, they will determine the portion of the file being updated. Furthermore, if W2 appears, a test is made to insure that W2 is greater than W1. If this error is detected, L6 appears on the comment medium.

The final field processed is the input mode field. This field must contain either an A or B. If the input mode field is A, records in ASCII format will be read from the standard input device. If the mode is B, Binary records will be expected from that device. Any other symbol will cause L6 to appear.

35.8.3.6.2 BRANCHING TO FILE SUB-PROCESSOR

Once all control statement fields have been processed, one of two routines is entered to update, or insert or replace a file. If no W1 field is specified this is a clue to the FILE module that a file is being inserted or replaced and that the routine INSREP is to be entered. If a W1 field is specified, the file must of necessity appear in the program library. In this case the UPDATE portion of this processor is overlaid into core. Also at this point, if W1 is zero W2 is checked for zero. If W2 is not zero an L6 error is output since it would appear that an UPDATE was desired but the W1 field did not get specified.

35.8.3.6.3 THE FILE INSREP ROUTINE

Upon entering INSREP all replacement blocks will be read from the input device and placed on mass storage scratch sectors.M A test is then made to determine if the file name from the control statement appears in the program

library directory. If the file name does not appear in the directory, the routine XXXX2 is entered. At XXXX2 the length of the new file is picked up and exit made to the SEARCH routine to find the first group of consecutive available mass storage sectors large enough to contain this file. Upon return to the processor, a check is made to see if the first sector of this group is the first scratch sector {$C1}. If this is the case {$C1} is updated by the length of the file and the new length's complement and {$C1} are inserted in the program directory. The Sector Availability Table is then updated by adding to {$C1} the length of this file to determine the number of sectors to set as occuppied. After setting the file's sectors as occuppied in the S.A.T. a check is made to see if a new file name is to be placed in the directory. If necessary the routine NEWFIL is entered by a return jump. The program directory is next written back to mass storage. The contents of $C0 and $C1 on the mass storage resident system is also updated before exiting back to the main control module.

When the group of consecutive available sectors found does not begin at {$C1} then it is necessary to move the file to its new location. A check is first made to see if any portion of the file after being moved will exceed {$C1}. If this is the case, this increment is saved and the processor proceeds to move the file to its new location. The new starting mass storage address and the complement of the file's length are put in the directory and the directory is output to mass storage. {$C1} is then updated if necessary, and the new sectors occuppied are zeroed out in the S.A.T. $C0 and $C1 are also updated if required on the mass storage resident system.

If the file name from this control statement appears in the program library directory, a test is made to determine if the replacement file will fit into the area provided by the file being replaced. If the replacement will not fit, all the sectors for the old program are released from the Sector Availability Table and the routine XXXX2 is entered. Otherwise, the routine REPLAC is entered.

The REPLAC routine first checks the length of the new file; if it is zero L09 is output to the comment medium and control is given to the LIBEDT module. If the length is not zero, the routine first updates the directory entry by specifying a new sector length for the file. The updated program directory sector is then output back on mass storage. The replacement file is next read into unprotected core, sector by sector, and transferred to the mass storage area occupied by the file being updated. If the new file is shorter than the old file, the unused sectors of the old file are released from the Sector Availability Table. Control then returns to the main control module.

35.8.3.6.4 FILE INPUT FROM MASS STORAGE

If the input device is mass storage, the routine INPUT checks to see if there is a header label with the identification ᵛLIBEᵛ in the second and third word of the first sector in the scratch area. If the record does not contain the identification ᵛLIBEᵛ L10 appears on the comment medium and control is given back to LIBEDT. If the header record is identified the first word of the header is used for the word count of the file, and the contents of $C1 are then incremented by one to point to the first record of the file. Entry is then made at XXXX2 to see if the file is to remain in its present location or to be moved. If a move is not required, the file will remain where it was input to mass storage and the process will continue as mentioned in the description of the XXXX2 routine (see Section 35.8.3.6.3).

35.8.3.6.5 THE FILE UPDATE ROUTINE

Immediately after entering UPDATE, all update file blocks are read from the standard input device and placed on mass storage.* A test is then made to determine if the file name from the control statement appears in the program library directory. If the name does not appear in the directory, L7 appears on the comment medium. If the name does appear in the directory, a W2 field test is made. If W2 is not present, only one word in the file must be updated and the routine ONONLY is entered. Otherwise W2-W1 words of the binary file must be updated provided the update does not exceed the library fileᵛs

length. If W2-W1 overflows the file, L8 appears on the comment device. If overflow does not occur, the first W2-W1 words are removed from the mass storage resident update file and inserted into the library file starting at word W1. After inserting W2-W1 words into the library file, exit is made from FILE to the main control module. This is signalled by IN appearing on the comment device.

When one file word is to be updated, the routine ONONLY is entered. This routine computes the word position in the file, of the word being updated, as follows:

SS = Starting Sector for File
Q+R= $\frac{\{W1\}}{96}$
UDS= Update Sector = {SS} + {Q}
UDW= Update word = {R}-1

ONONLY then reads the update word from the input device and places it in unprotected core. The update word is next inserted into word UDW on sector UDS. After update is complete, control returns to the main control module.

35.8.3.7 SET SYSTEM DIRECTORY REQUEST PRIORITIES - RPUPDT

RPUPDT is the statement processor responsible for setting the request priorities of all entries in the system directory. The request priorities set by RPUPDT will determine the area of allocatable core where each of these programs may operate.

The first operation performed by the processor is to fetch the ordinal number and request priority from the call. Checks are made to assure that these values are not non-numeric. If non-numeric an L6 error is output. Next a test is made to assert that the specified request priority is not greater than sixteen {16}. The largest allowed. If greater a L1 error is output. Two final tests are made before inserting the specified request. The first of these checks is to see if the entry to be updated is mass memory or core resident. The second check is to assure that no more parameters are present. If so, an L6 error is output.

Once all parameters have been deciphered RPUPDT proceeds to insert the request priority in the directory based on whether it is for a mass storage or core resident program; as previously determined from the call. In each case the specified ordinal is checked against those in the directory to determine if it is a legal ordinal. If not a L3 error is output. If correct the request priority is stored in the correct directory entry, the position of the entry in the system calculated, and the correct value inserted in the directory on the mass storage resident system.

35.8.3.8 COPY BINARY AND ASCII RECORDS - COPY

The COPY statement processor is a utility program within LIBEDT which allows the transfer of information between any two peripheral units. This transfer may be in any combination of ASCII or Binary specified. The maximum number of records to be transfered may also be specified.

The actual input and output of the information to be transfered by this processor is done through an FREAD and FWRITE request in the processor. All processing prior to these two operations is concerned primarily with removing the information from the call and inserting this information in the correct locations in the input and output requests. The following sequence of events accomplish this task:

1. Set the input logical unit in the FREAD request

   A. Check the following features if a unit is specified

      1. Issue an L1 error if unit not numeric

      2. If the unit specified is not an input unit output an L6 error from subroutine LOGUT.

   B. When no unit is specified insert the standard input logical unit in the request.

2. Pickup $C0 and $C1 and save for possible use if the mass storage devices are selected.

3. Set starting address of READ Buffer to first unprotected +1 and the length of the READ to the size of unprotected core.

4. Get the input mode

   A. If the value is not non-numeric output a L1 error.

   B. Check if ASCII or Binary and save

5. Set the output logical unit in the FWRITE request.

   A. Check the following feature if a unit is specified.

      1. Issue an L1 error if unit non-numeric

      2. If the unit specified is not an output unit issue an L6 error from subroutine LOGUT.

   B. When no unit is specified insert the standard output logical unit in the request.

6. Check if the maximum record count is specified

   A. If the value obtained is not numeric output a L1 error

   B. If value is numeric complement and save it.

   C. When no value is specified the value saved is zero.

7. If the input or output is the library or scratch mass storage units their mass storage addresses are setup for the requests.

8. Transfer all information as now specified.

9. Exit to the subroutine SENDC to output to the standard list device the number of records transferred.

10. From SENDC exit back to the main control module.

35.9 SUBROUTINE LOGIC

35.9.1 EXECUTIVE MODULE SUBROUTINES

35.9.1.1 LIBEDT PROTECT PROCESSOR - B

This routine is used to monitor protect faults which occur during LIBEDT operations with the loader. If the fault is caused by a loader entry to the Job Processor or the monitor, processing continues as specified. All other entries will kill the job and release LIBEDT.

35.9.1.2 EXIT LIBEDT - LIBXIT

When it has been determined that the LIBEDT job should be terminated entry is made to LIBXIT where a direct jump is made to the Job Processor to release LIBEDT. The *Z statement and JKILL routine are the primary users of LIBXIT.*

35.9.1.3 JOB KILL - JKILL

In all instances when it is beneficial to check if the LIBEDT job has been killed before continuing, entry is made to this routine. JKILL will determine if exit back to the statement processor is necessary before returning to the Job Processor. The protected common flag used by the loader is also clear in case if it was being used by the *M processor.

35.9.1.4 AREA3 OVERLAY ROUTINE - READA3

On entry to the overlay handler the subroutine group increment is saved and a check is made to determine if the correct subroutine group is already in. If in the increment is added to the start of AREA3 to get the entry location. The return address is then put in the first word of this subroutine. A, Q, and I are setup and a jump is made to the subroutine.

When the called for subroutine's group is not in AREA3 it is first read into the area and the process then continues as specified above. If a read error occurs while reading the overlay in, the job is aborted and exit made back to the Job Processor.

35.9.1.5 AREA4 OVERLAY ROUTINE - READA4

The logic flow of the overlay handler for AREA4 subroutine groups is the same as that for AREA3 . See the description for the AREA3 Handler {35.9.1.4}.

35.9.1.6 OVERLAY HANDLER FOR STATEMENT PROCESSORS AND MAIN CONTROL MODULE - TRANF

When overlaying the main control module into AREA2 and AREA3 entry is made at TRANF. This subroutine first save an index which determines where entry is to occur in the control module and any error number being passed for processing. The length for the read in of the control module is put in the request as well as its calculated starting mass storage address.

At BRIPR0 and BRIPR1 the actual mass storage read operation to bring in the main control module or statement processors occur. If a statement processor is to be overlaid, its number is saved. This number has no significance other than aiding debugging. Once the overlay is completed a jump is made to the beginning of AREA2 {PROGIN} to beginning processing.

35.9.1.7 MASS MEMORY ADDRESS CALCULATION - MMADDR

This routine gets the mass storage address of LIBEDT from the system directory and converts this sector number to word address. The address of the sub-routine is then added to this address and supplied to the proper overlay routine.

35.9.2 MAIN CONTROL MODULE SUBROUTINES

35.9.2.1 KREAD ROUTINE

This routine causes a control statement to be read from the comment or input device. Interrogation of the statement buffer will continue until a statement is input.

35.9.2.2 ERROR PROCESSING - NEXT, NEXT1

Entry is made in the main control module at NEXT1 if an error message is to be output that occurred outside of this module. All error processing from which the module enters at NEXT. After output of the error message the input statement buffer is interrogated for another control statement.

35.9.2.3 *V AND *U PROCESSORS - CONCTL - SYSINP*

The control statement input device is determined by CONCTL and SYSINP. When SYSINP is entered the input is selected (via $F9). Control statements will be read from the specified standard input device until a *U occurs which sends control back to the standard comment device for output of the IN message.

35.9.2.4 *K PROCESSOR ROUTINE - CHANGE*

This routine processes the *K statement. Standard input, print or binary output logical units may be changed in the I/O table (IOTAB), thus allowing the operator the option of selecting devices for systems units other than those currently used.

*NOTE: Since the *Z, *V, *U, and *K are indirect LIBEDT functions that service the major processors; separate AREA2 processor programs do not exist to execute these functions. Their present locations reduce the need for continuous overlaying when setting up to execute a major statement function.

35.9.3 AREA3 SUBROUTINES

35.9.3.1 GROUP NO.1 - MSLOC

35.9.3.1.1 MASLOC

This subroutine checks to see whether the loader when processing a *P or *M LIBEDT statement left the absolutized file out on mass storage or if it was left in core. If it is on mass storage it is brought into core. If it is in core it is left there. If the file is on mass storage and must be brought into core a check is made to insure that it will fit. If it is too large a error 14 is output.

35.9.3.1.2 MOVIT

The MOVIT routine is used by the *M processor to perform a core to core transfer of data from unprotected core to the specified starting address in the system directory of this core resident program. Interrupts are inhibited during the move operation.

35.9.3.1.3 RDIN

This is the calling subroutine for the *M processor to execute the MASLOC and MOVIT subroutines.

35.9.3.1.4 SETCRR

SETCRR is used to set temporary unprotected core limits for those processors that use the loader.

35.9.3.1.5 GETCOR

After a load operation is completed GETCOR can be executed to obtain the length of the load in unprotected core.

35.9.3.1.5 RBLOD

This routine is used to execute a relocatable binary load from the loader. Once the actual load has occurred the routine makes a second call to the loader to attempt to clean up all unpatched externals before returning to the calling processor.

35.9.3.1.6 LOAD

All calls to the loader from any other subroutine or processor is directed through LOAD. Upon entry the A register contains the loader operation code and the Q register the core address of the first of three sequential locations containing an entry point name or an address for data base operations. Upon return if normal termination occurred the mass storage or core load indicator is saved before processing continues. If termination was via a control statement, the statement is moved to the statement input buffer and the mass storage or core indicator examined. A check is finally made before exiting back to the processor to unset the job kill switch if a data base operation is in progress.

35.9.3.1.7 MVCST

If the loader operation performed by the LOAD subroutine terminated with a control statement this subroutine is entered to move the statement to the statement input buffer from where the next operation can be interpreted.

35.9.3.1.8 ST4Q6A

This routine is entered from the *M processor to insert the length and starting sector number of the core or mass storage resident program replaced in the system directory on the mass storage core image of the system.

35.9.3.2 GROUP NO. 2 - INSREL

35.9.3.2.1 INSERT

Whenever programs or files are added, replaced, or removed from either the system or program directories it is essential that the sector availability table be updated to reflect current disk storage thus avoiding destruction of information. The INSERT routine is entered for this purpose. The sector number whose representative bit is to cleared or set is passed to the routine in the A register. Whether the sector bit is to be cleared or set is determined by the value passed in INPPAR. A zero passed signifies that the sectors bit is to be set as not available and a one causes the sector to become available for future use.

35.9.3.2.2 UPDSET

This subroutine is used in conjunction with the INSERT subroutine when blocks of sectors are to be set or cleared in the Sector Availability Table. The routine is used by the *M, *R, and *N processors.

35.9.3.2.3 IOBLE

When a program is to be replaced or removed from the program library, IOBLE is entered to read up the old program blocks to get the sector number of the next program block. This sector number is then used by the processor to clear the sector in the sector availability table.

35.9.3.2.4 ZAES

This routine is used by the *L and *R processor to zero out all directory entries which have as a beginning sector number the same number as the starting address of the program being removed or replaced.

35.9.3.2.5 ZER

ZER is used by ZAES to perform the actual zeroing of the program library directory entry.

35.9.3.3 GROUP NO. 3 - SENDC1

35.9.3.3.1 SENDC

The *T processor upon completion of its read and write operation exits to this routine to convert the record count from hexidecimal to ASCII and to output a message to the standard print device specifying the number of records transferred. Only a four digit number can be output by the routine although there is no limit to the number transferable by the *T processor.

35.9.3.3.2 LUCHK

When transfers are to be performed by the * T processor this routine checks the specified input and output device for their legality. Checks are made of the following features: 1) that the unit specified is not the core allocator (L.U.*1), 2) that the device is a legal read or write device, and 3) that the logical unit number does not exceed the number of units in the system as specified by location NUMLU in SYSBUF.

35.9.3.4 GROUP NO. 4 - SACREL

35.9.3.4.1 SAC

This routine searches the program directory for a match to an entry point. If a match is found the sector number of directory block containing the entry point will be saved. The Q register is returned to the processor with the index to the first word of the match entry in this sector and the A register is set to zero. If no match is found the A register is returned with a non-zero value. Entry at SAC is made when a program entry point is being checked to determine if a replace operation is to be made or when the legality of a remove operation is required.

35.9.3.4.2 SACF

Entry is made to this routine to execute SAC for file operations. This includes determination of file replacement and validity of file removals.

35.9.3.5 GROUP NO. 5 - SERCH

35.9.3.5.1 SEARCH

The insertion of new programs, program library, or file blocks on mass storage will cause SEARCH to be entered. Program and program library blocks are chained together on mass storage. Therefore, SEARCH will locate the next available sector, zero out its bit in the Sector Availability Table, and return to the calling routine with the sector number.

Files unlike program or program library blocks must reside contiguously on mass storage. The Sector Availability Table is therefore, interrogated to find the first group of sectors available large enough to contain this file. When found return is made to the file processor with the beginning sector number for the file. These sectors will be set as occuppied by a call to the UPDSET (see section 35.9.3.2.2) subroutine by the file processor.

35.9.3.5.2 FELD

The addition of program or file entry points to the program library may necessitate the acquisition of an addition program directory sector . When this is the case FELD is entered to find the current last directory block such that the sector address of the new block obtained by executing SEARCH can be stored for chaining as the directory. If the new block obtained is in the scratch area of the disk the $C1 scratch pointer is updated. Before exiting the new block is also zeroed out.

35.9.3.6 GROUP NO. 6 - INPREL

35.9.3.6.1 INPUT

This routine reads from the input or comment device ASCII or Binary Format Records of 96 words or less for input to the file processor. Reading is terminated by an error return from the input driver or by the next control statement. If termination is via the control statement the statement is moved to the LIBEDT statement input buffer to be processed upon completion of the current request. The blocks of information are input to unprotected core. If unprotected is not large enough to contain all the input, unprotected is then moved to mass storage each time it is unable to hold more information.

Upon completion of all input operations return to the file processor is made with the A register specifying the number of sectors of information read. If input was from mass storage the number of sectors of information in the file is obtained from the file header record thus negating the necessity of reading up all the information to obtain a sector count.

35.9.3.7 GROUP NO. 7 - MASUP

35.9.3.7.1 MASSUP

Upon completion of processor tasks that add or remove information from mass storage, or when a job kill occurs, it is essential that the beginning scratch sector get properly updated as not to cause destruction of information. MASSUP accomplishes this by reading up the system sector containing the scratch address, changing its value if necessary and then writing the system sector back to mass storage.

35.9.3.7.2 SETCL

This routine is used to set the word length, input buffer address, and sector number location address for read of a system's sector from mass storage.

35.9.3.7 GROUP NO. 8 - ESSREL

35.9.3.7.1 ESS

The insertion of new entry point names in the program library requires that a search of the directory be made to locate and utilize any empty entries in existing program library sectors. If an empty entry is found the Q register points to the first available word in the entry and the A register is set to zero. If no entries are found the A register is returned with a non-zero value. When no entries are found subroutine FELD (See Section 35.9.3.5.2) must be executed to get a new library block before returning to EES.

35.9.4 AREA4 SUBROUTINES

35.9.4.1 GROUP NO. 1 - MFRREL

35.9.4.1.1 MFREAD

This routine reads one sector from a mass storage unit. The address of the sector number must be in the Q register and the address of the input buffer in the A register. Q15 = 1 signifies a separate scratch unit. Q15 = 0 signifies that the library and scratch units are the same.

35.9.4.1.2 MFWRITE

MFWRITE writes N words on a mass storage device the address of the word containing the sector number must be in the Q register and the address of the output buffer in the A register. The number of words to write is passed from storage location WS.

35.9.4.1.2 OPNEW

When outputting program blocks OPNEW sets up the parameters for the mass storage write.

35.9.4.1.3 RSAT

RSAT sets the input buffer and sector address location for reading a block of the sector availability table into core.

35.9.4.1.4 WSAT

WSAT sets the output buffer address and sector number location for writing a block of the sector availability table back to mass storage.

35.9.4.1.5 SUR

SUR sets the input/output and sector address locations for reading from and writing to the program library.

35.9.4.1.6 OINSE

This routine inserts entry points in the program library directory one entry point at a time. This subroutine is used in conjunction with ESS {section 35.9.3.7.1} which locates empty entries and FELD {section 35.9.3.5.2} which obtains additional program library blocks if necessary. If an additional directory block is required this routine will write the S.A.T. out to mass storage before exiting.

35.9.4.2 GROUP NO. 2 - PPKREL

35.9.4.2.1 PPRK

Each time a control statement field must be processed PPRK is entered. This routine will extract up to 6 characters and interpret them as follows:

1) If all characters are numeric {0---9}, they are assumed to be decimal and converted hexidecimal. On return from PPRK the following will be set:

   a) A = 1

   b) Q = number of digits

   c) XXK = converted number

2) If any character is alphabetic, all characters are assumed to form a name. On return from PPRK the following will be set:

   a) A = 0

   b) Q = number of characters

   c) XXK, XXK+1, XXK+2 = packed characters

35.9.4.2.2 ZRQOUT

This routine is used by PLINSN and FILE to move the entry point name input from the XXK buffer to the BUFF1A input buffer thereby allowing the de-parameterizing of the remaining portion of the control statement.

35.9.4.3 GROUP NO. 3 - RINREL

35.9.4.3 READIN

READIN reads records from the input or comment device for the *L and *N processors. The input buffer address must be passed in the A register and the read length in WS. If mass storage is being used the scratch sector address must also be passed in PDSN and NORD. The Q register upon entry will pass a first time flag to avoid having to set up the I/O information on each entry when the subroutine is being used by the file processor the read mode is checked and stored in the request.

35.9.4.4 GROUP NO. 4 - KWRIT

35.9.4.4.1 KWRITE

This routine writes two words on the standard output comment medium. The two words to write immediate follow the return jump used to exit to the subroutine. The primary calls to the routine are to output error messages and the IN messages.

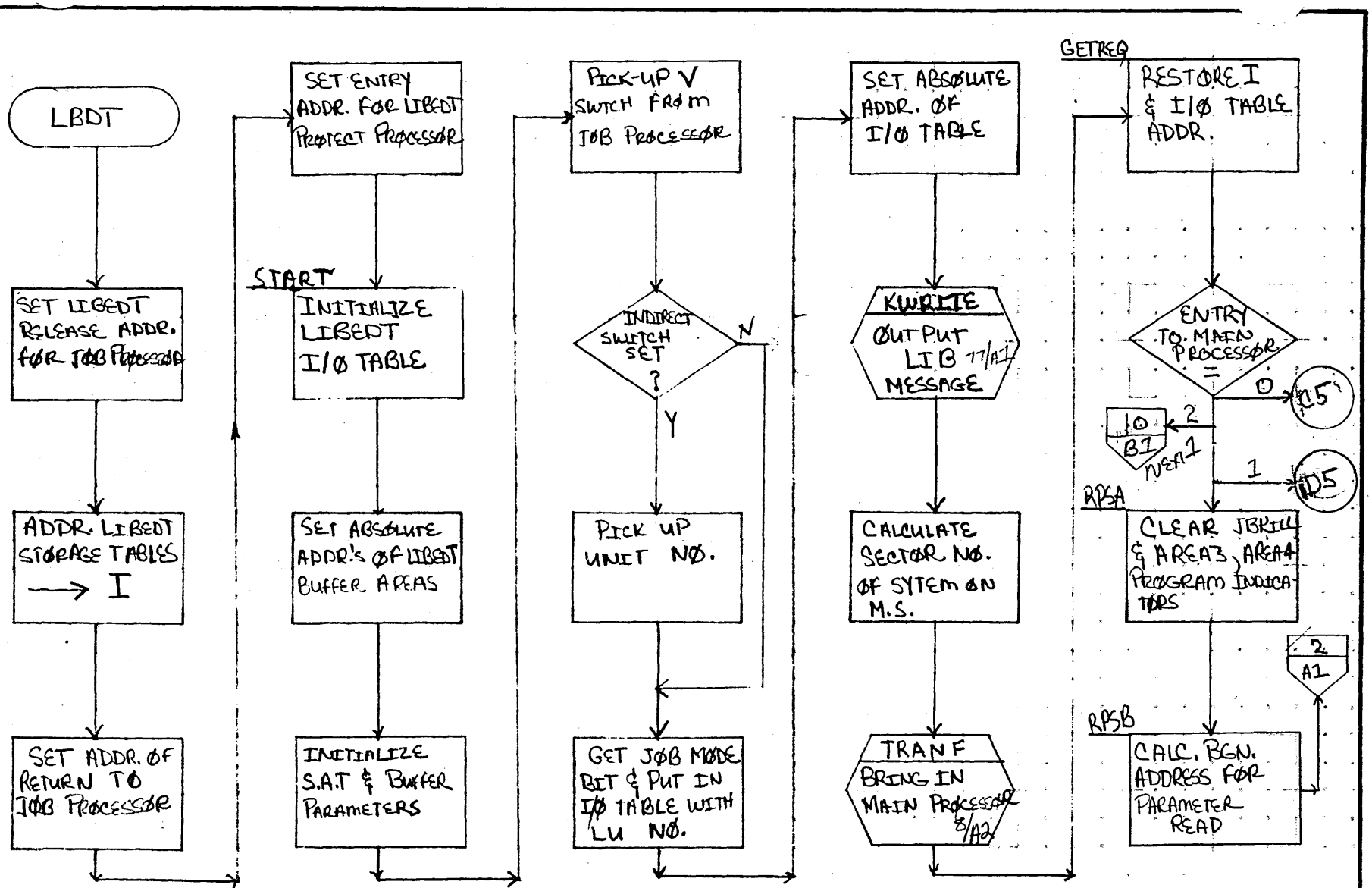
35.9.4.4.2 OWRITE

OWRITE functions through KWRITE and differs only in that the output device is set to the standard print device.

LIBEDT FLOWCHART INDEX

LIBEDT FLOWCHART INDEX {Continued}

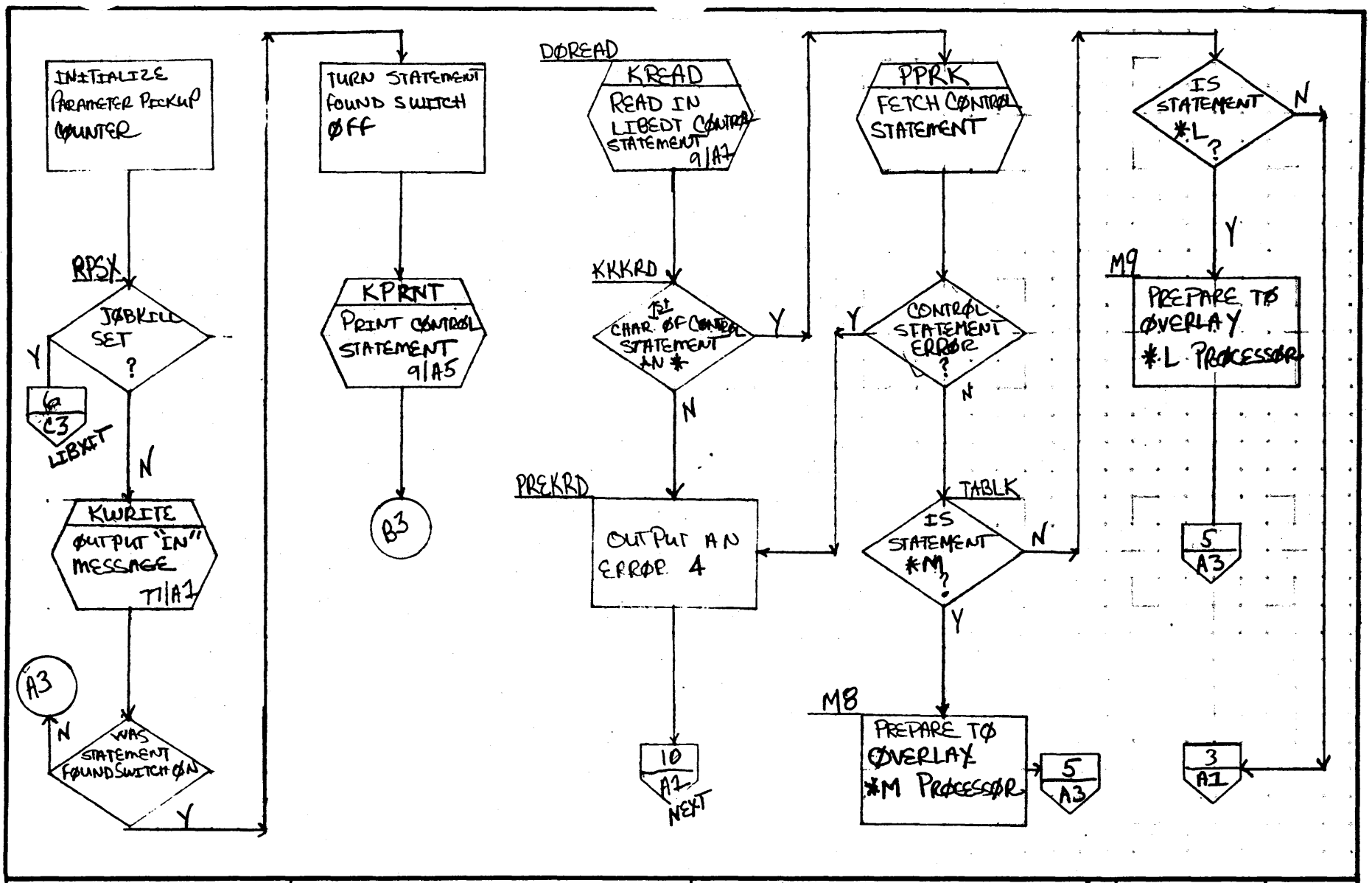
ROUTINE	PAGE NO.	COORDINATES
MFREAD	70	A1
MFWRITE	70	A4
OPNEW	71	A3
RSAT	71	A4
WSAT	71	A5
SUR	72	A1
OINSE	72	A2
PPRK	73	A1
ZRQOUT	75	A2
READIN	75	A3
KWRITE	77	A1
OWRITE	77	A3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LIBEDT	PAGE 1 OF 79	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

35.57



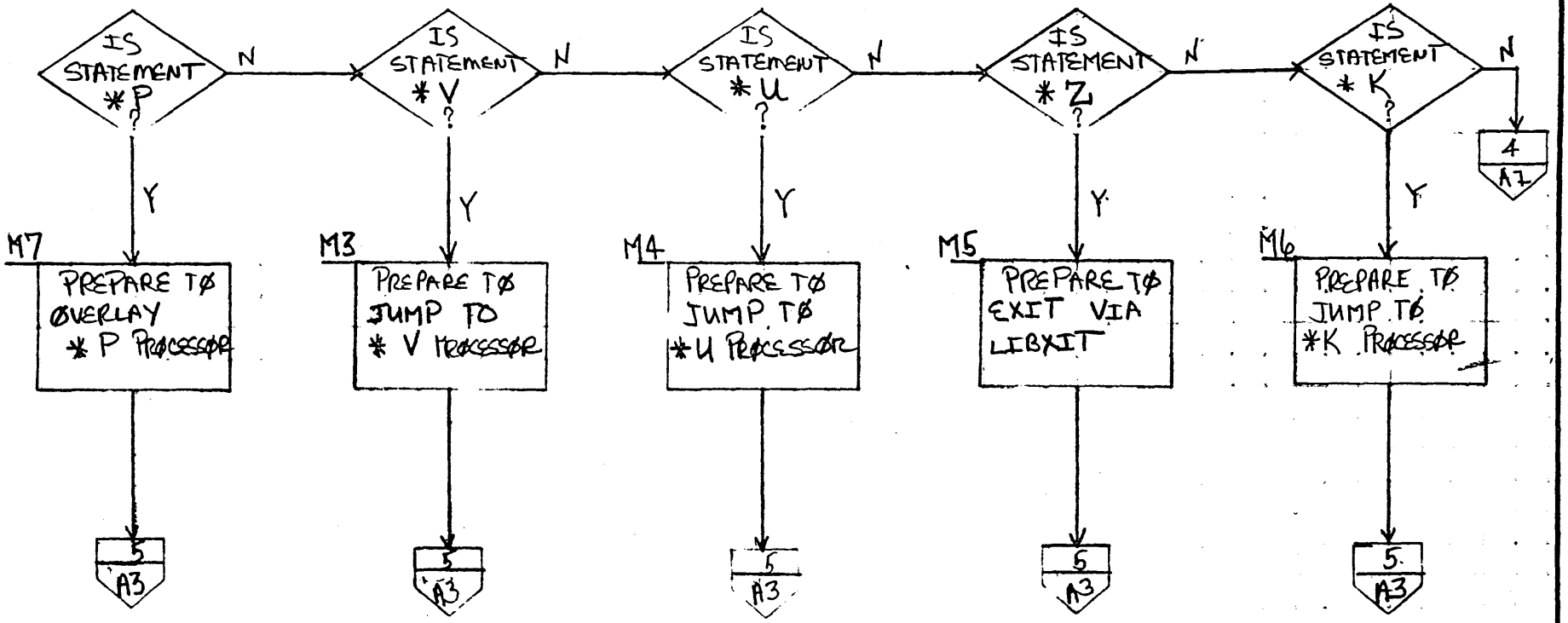
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	LIBEDIT	PAGE 2 OF 77	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.58

CA1385 (FORMERLY CA127-1)

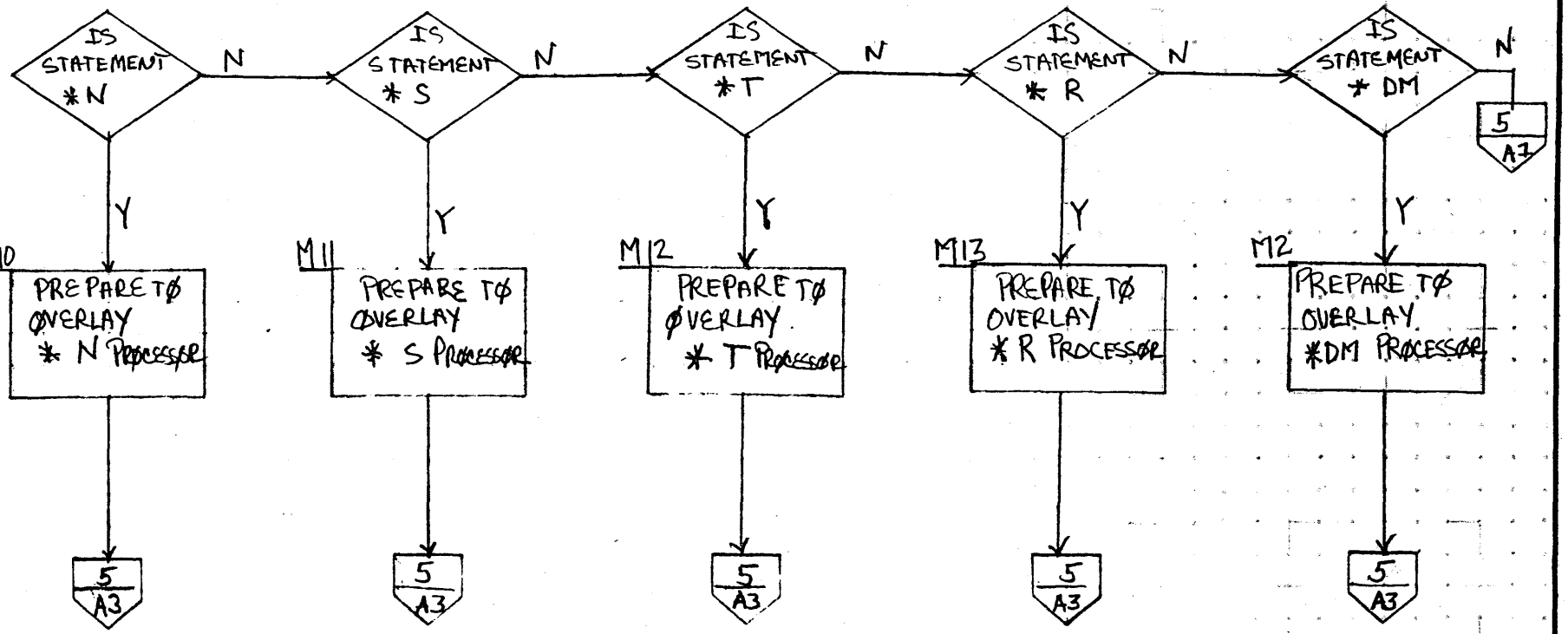
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 3 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.59



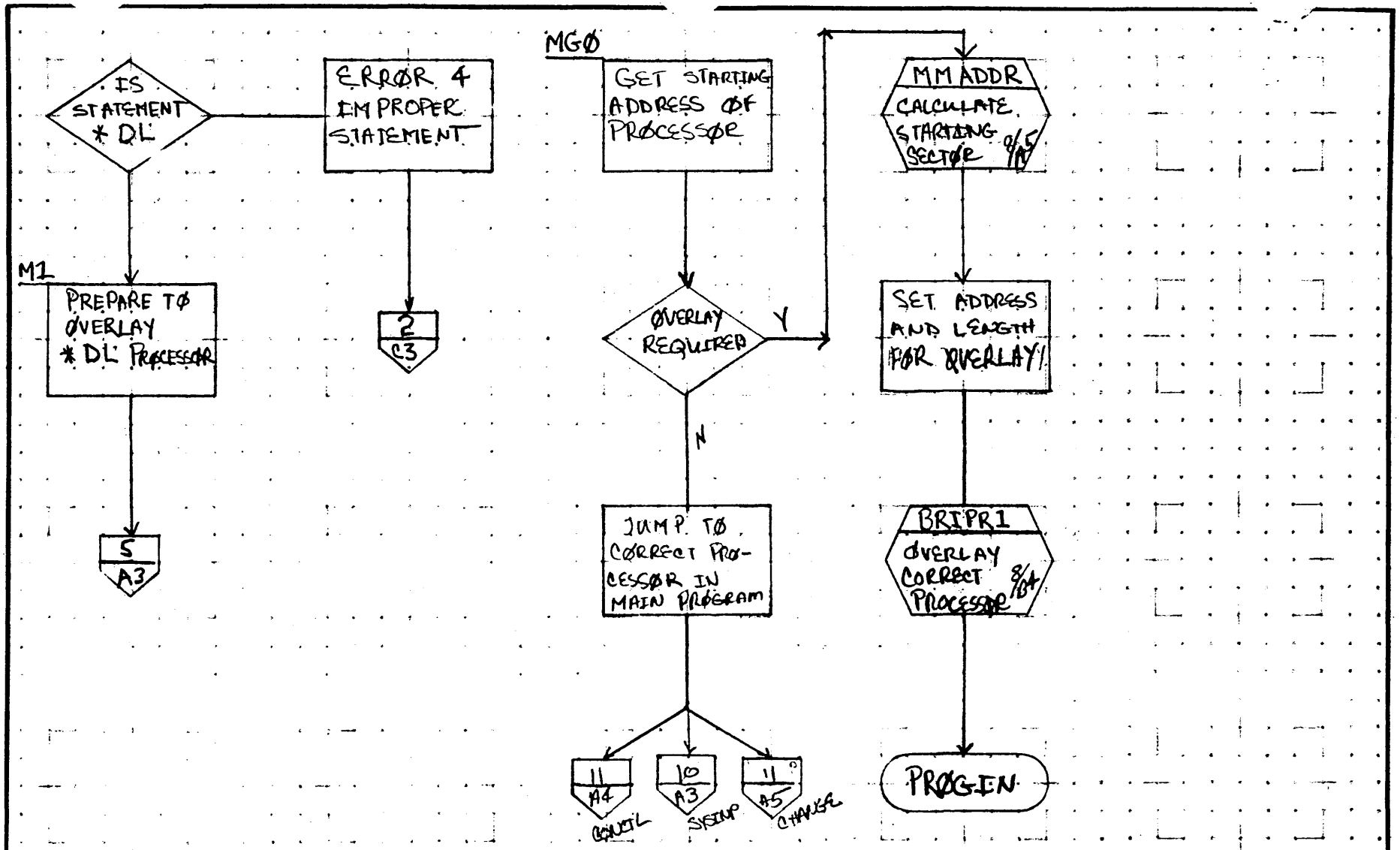
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 4 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35-60

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



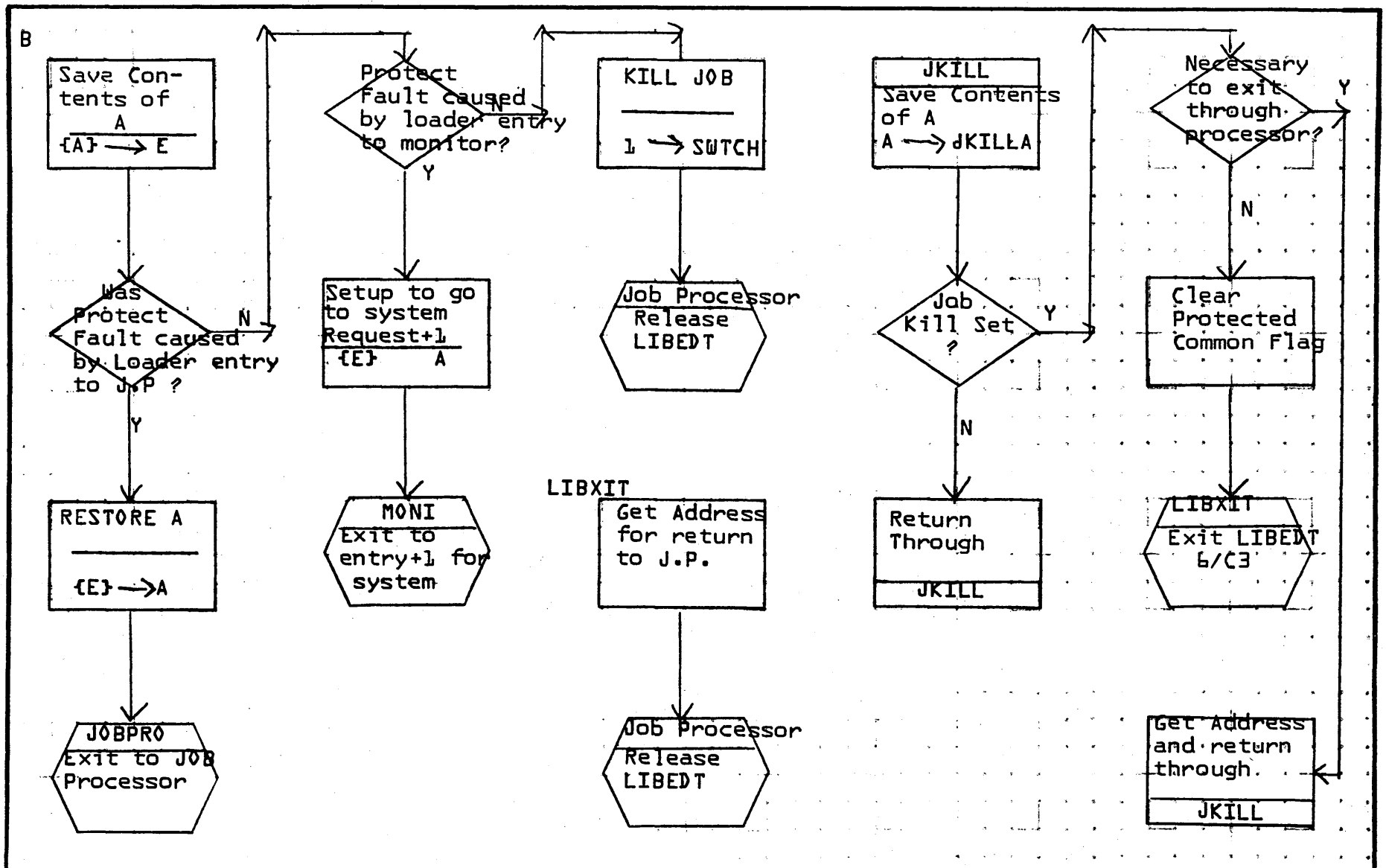
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	LIBEDT	PAGE 5 OF 77	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.61

AA1385 (FORMERLY CA127-1)

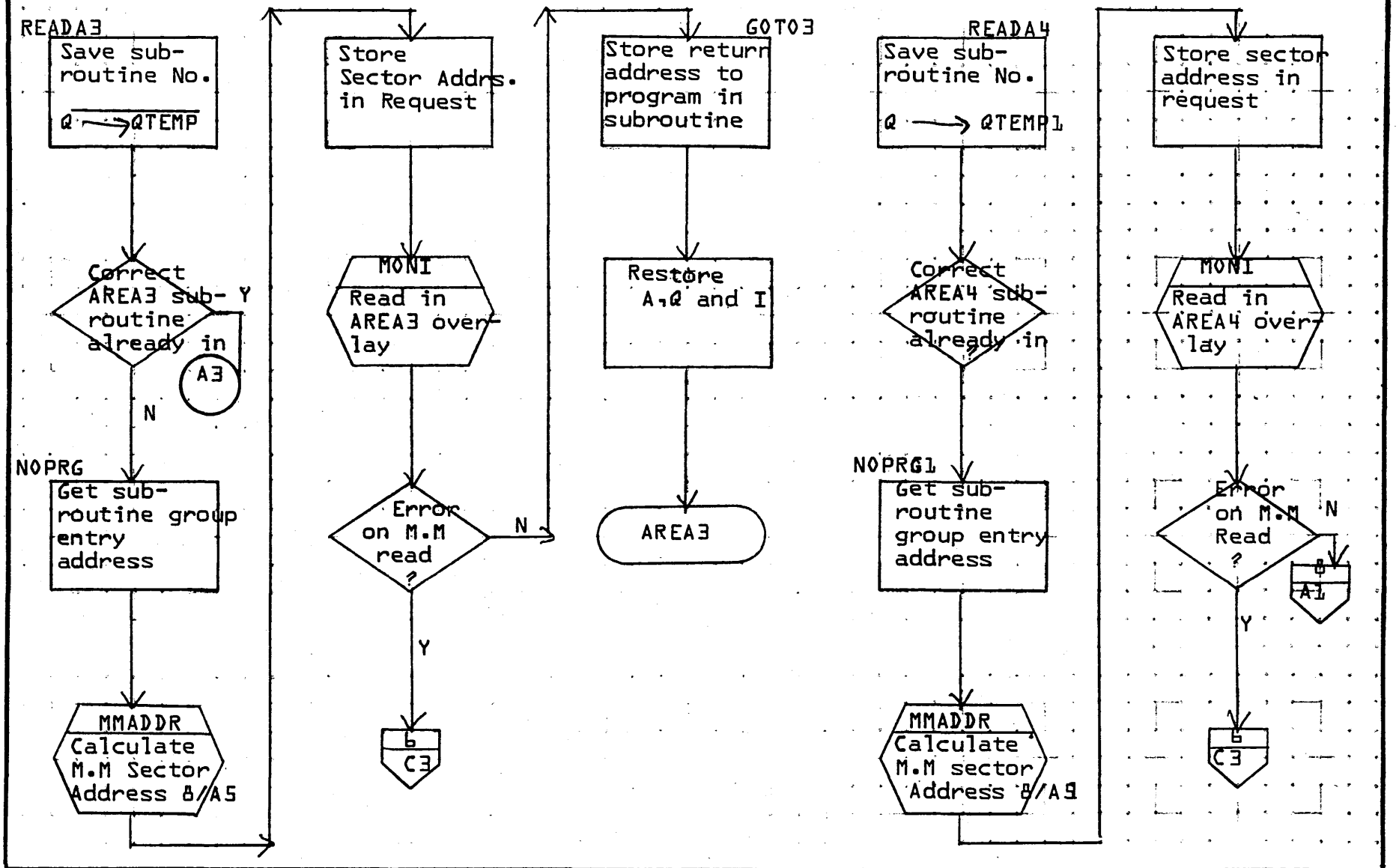
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS, MACH. TYPE 7600	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
	PAGE 6 OF 17	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

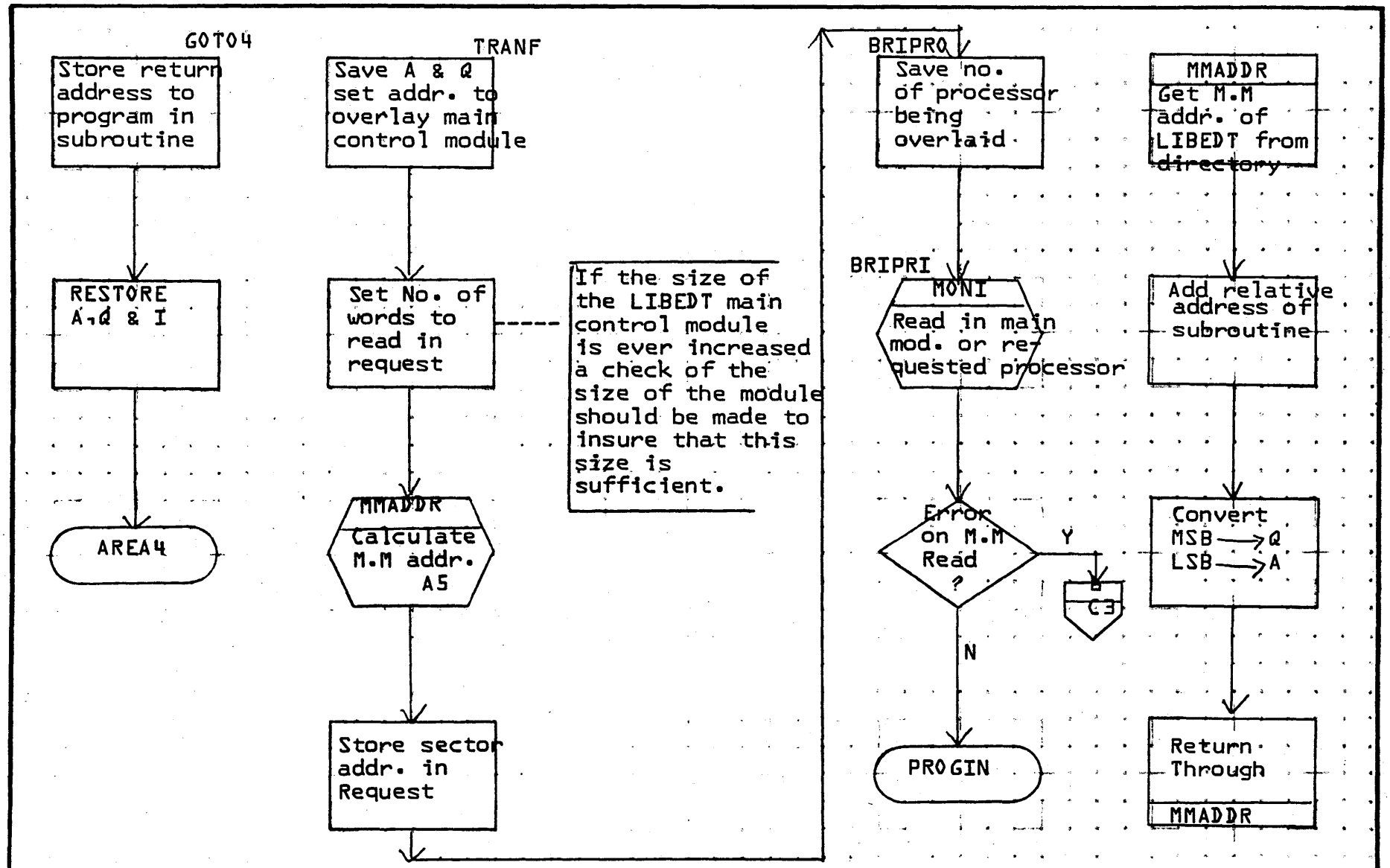
35.62



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 7 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.63



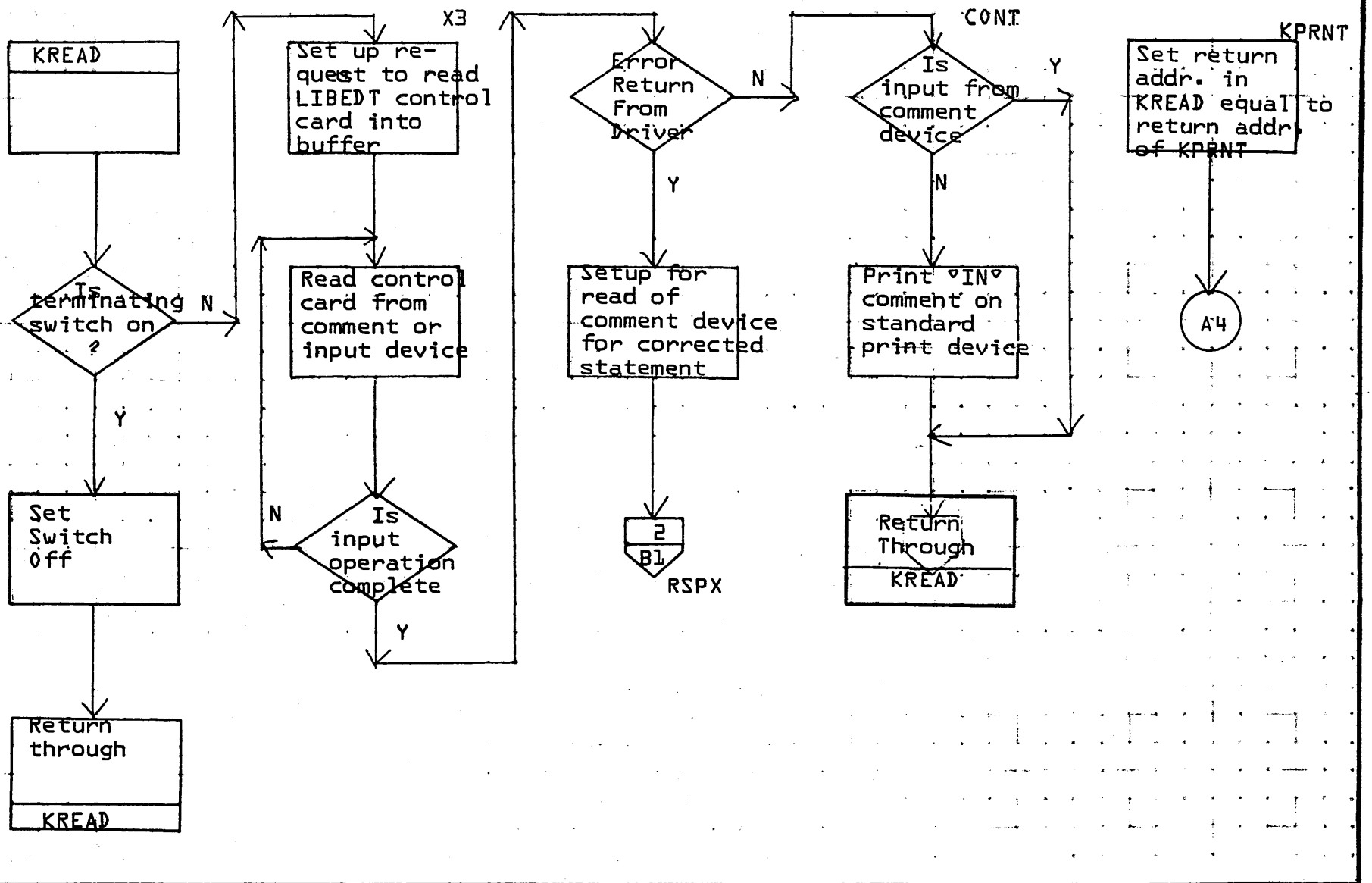
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS LMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 8 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.64

AA1385 (FORMERLY CA127-1)

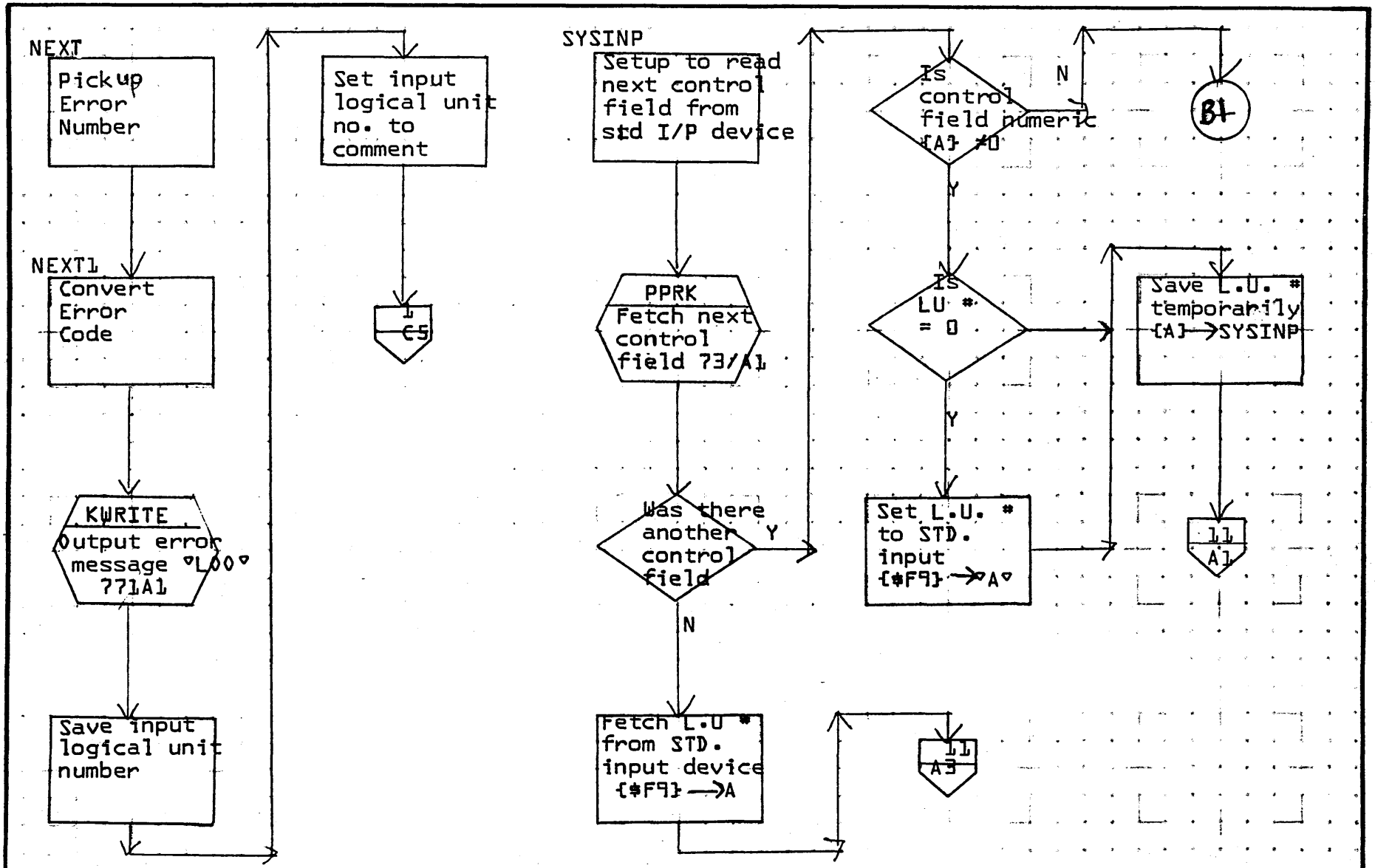
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 9 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.65



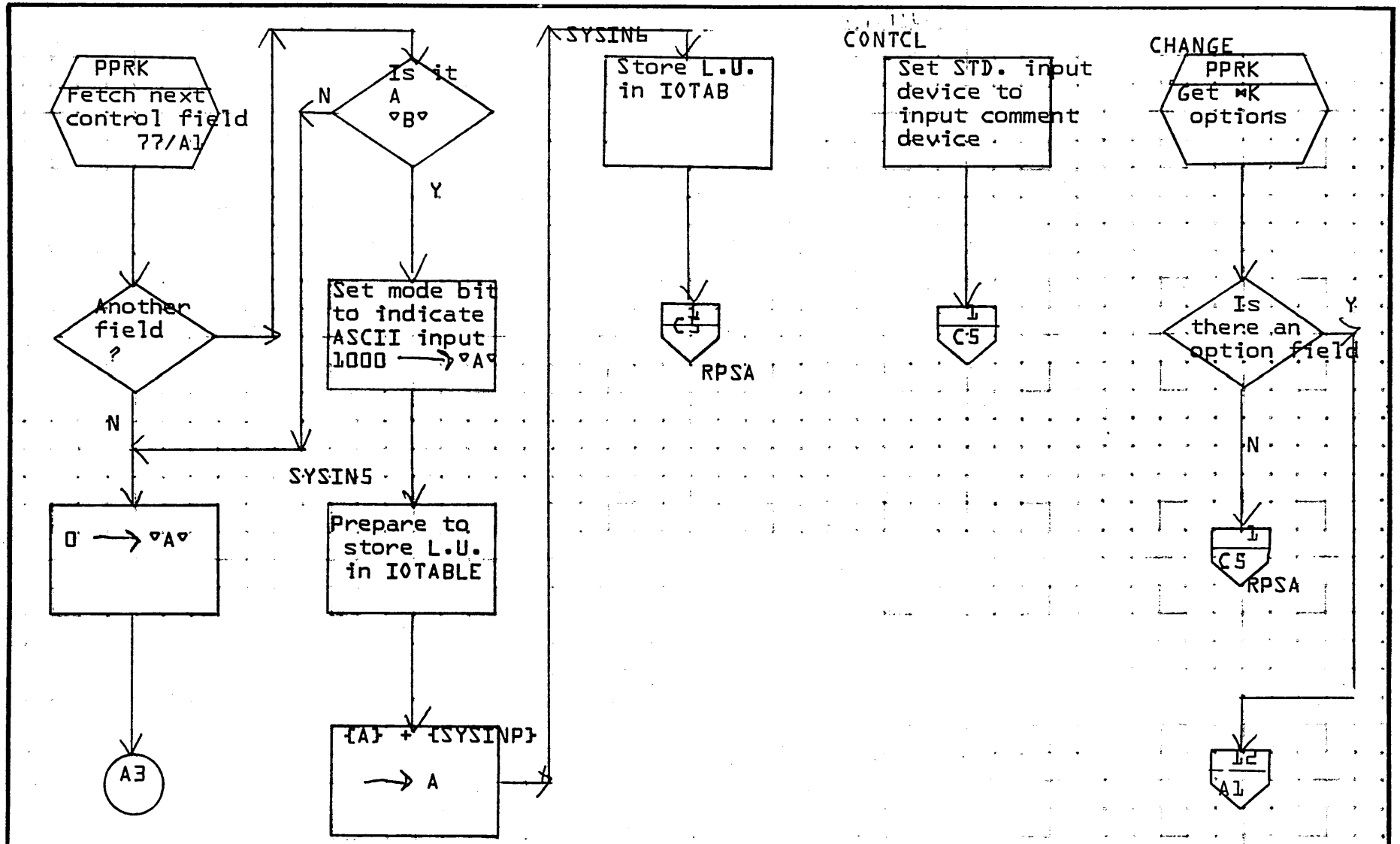
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	LIBEDT	PAGE 10 OF 77	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.66

AA1385 (FORMERLY CA127-1)

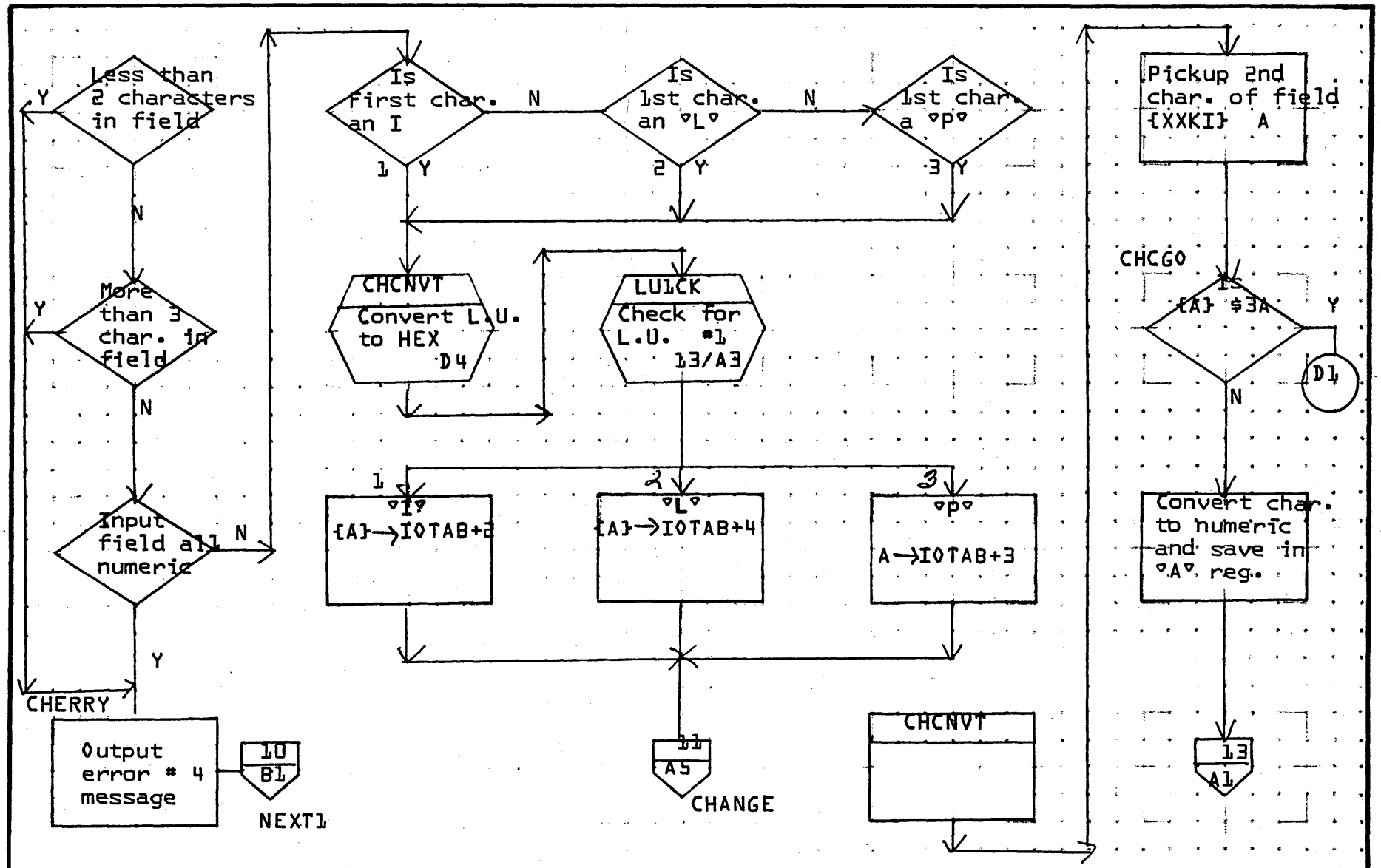
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS 7MS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	LTBIOT PAGE 11 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

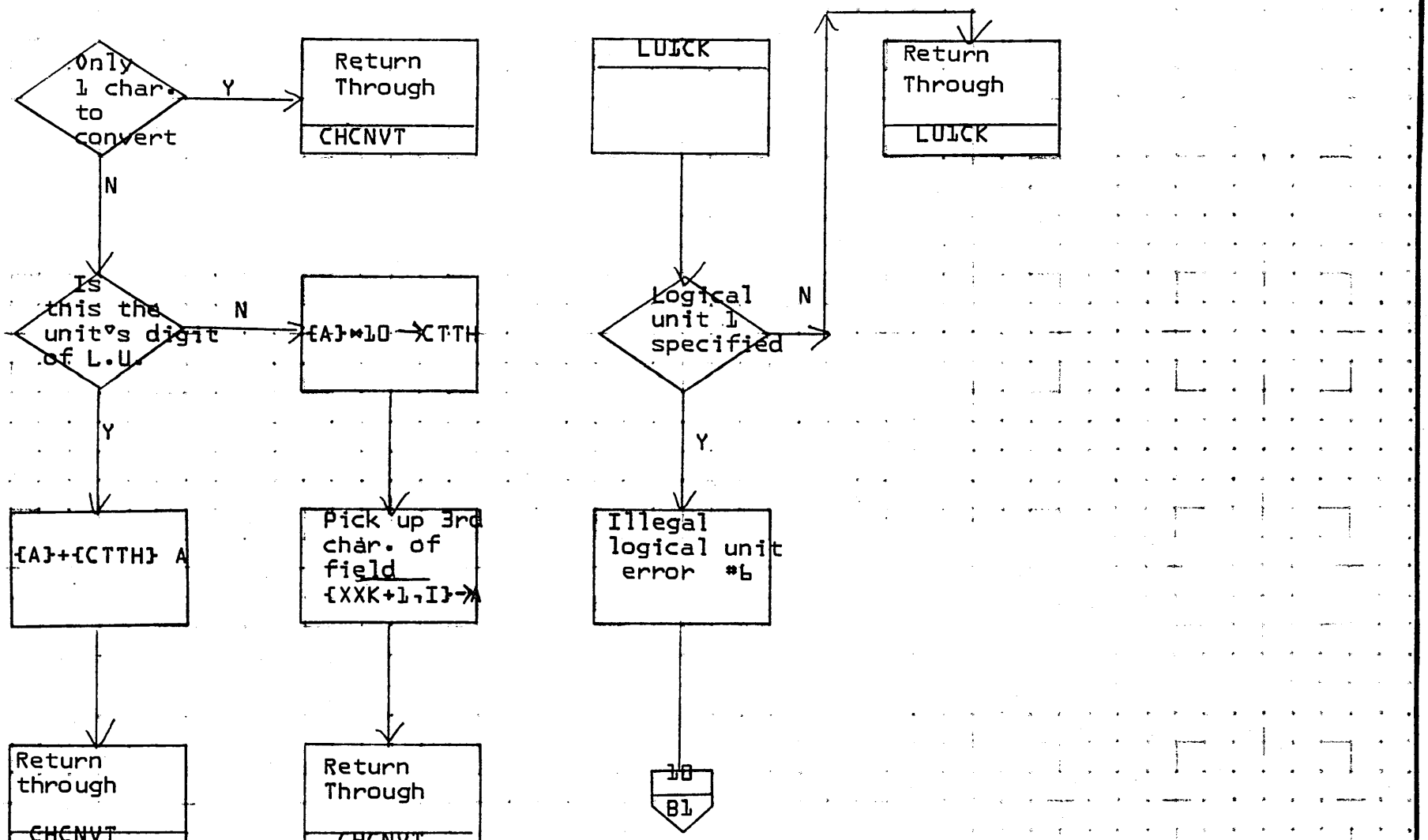
35.67



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS 1MS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 12 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.68



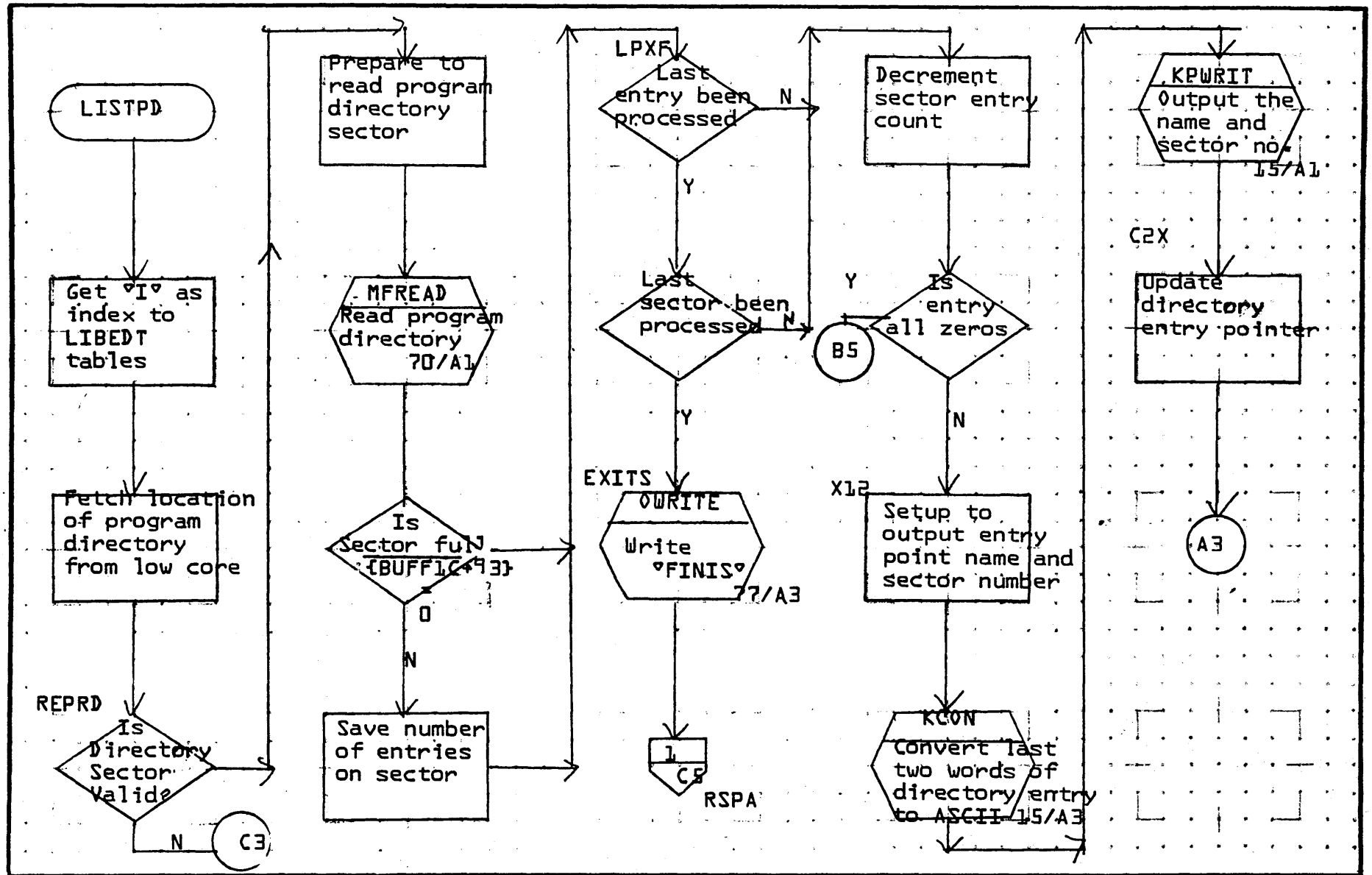
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	LIBEDT	PAGE 13 OF 77	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.69

AA1385 (FORMERLY CA127-1)

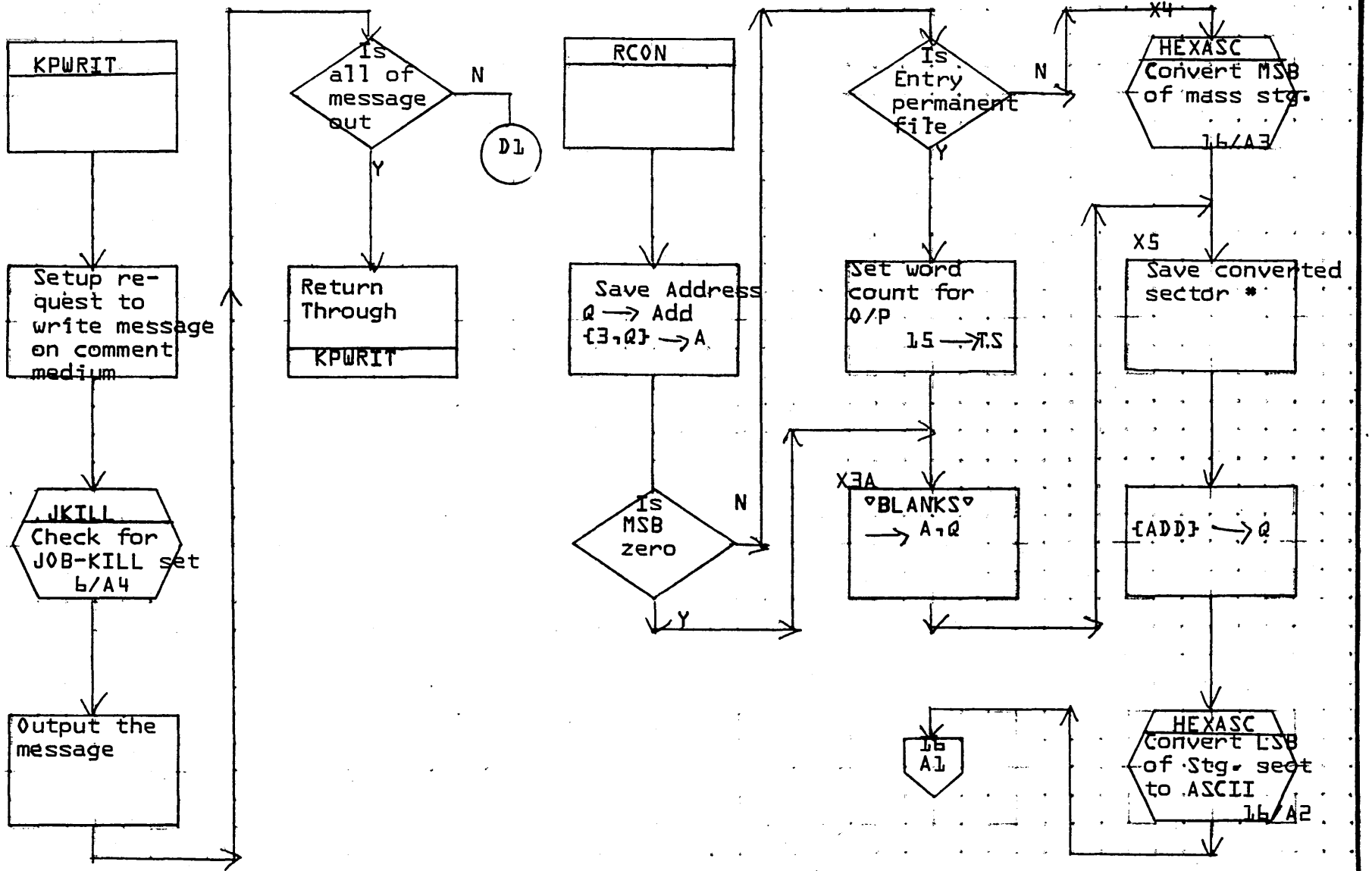
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS Ims MACH. TYPE Dmn	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 14 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.70



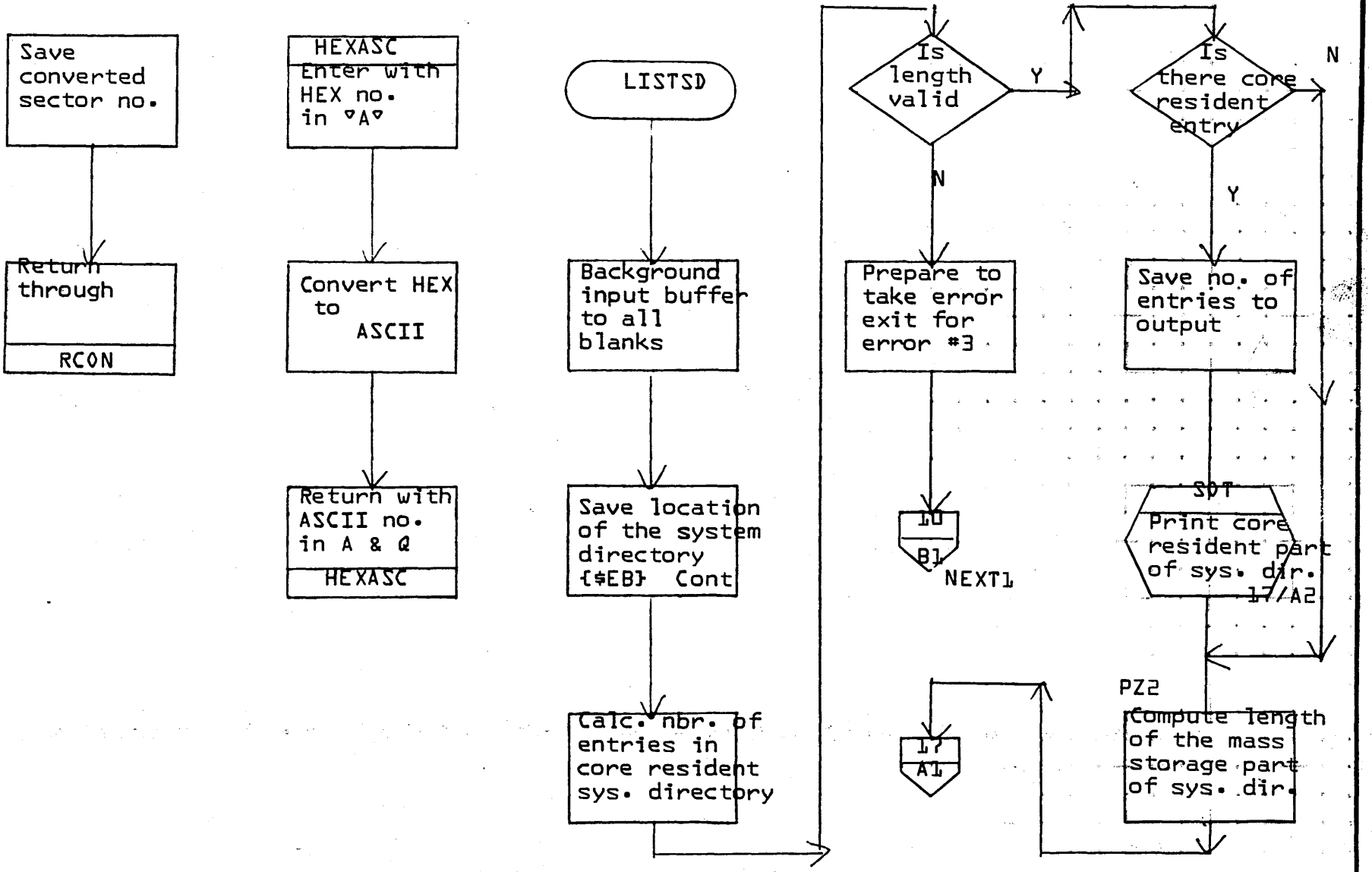
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1760	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 15 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

35.71



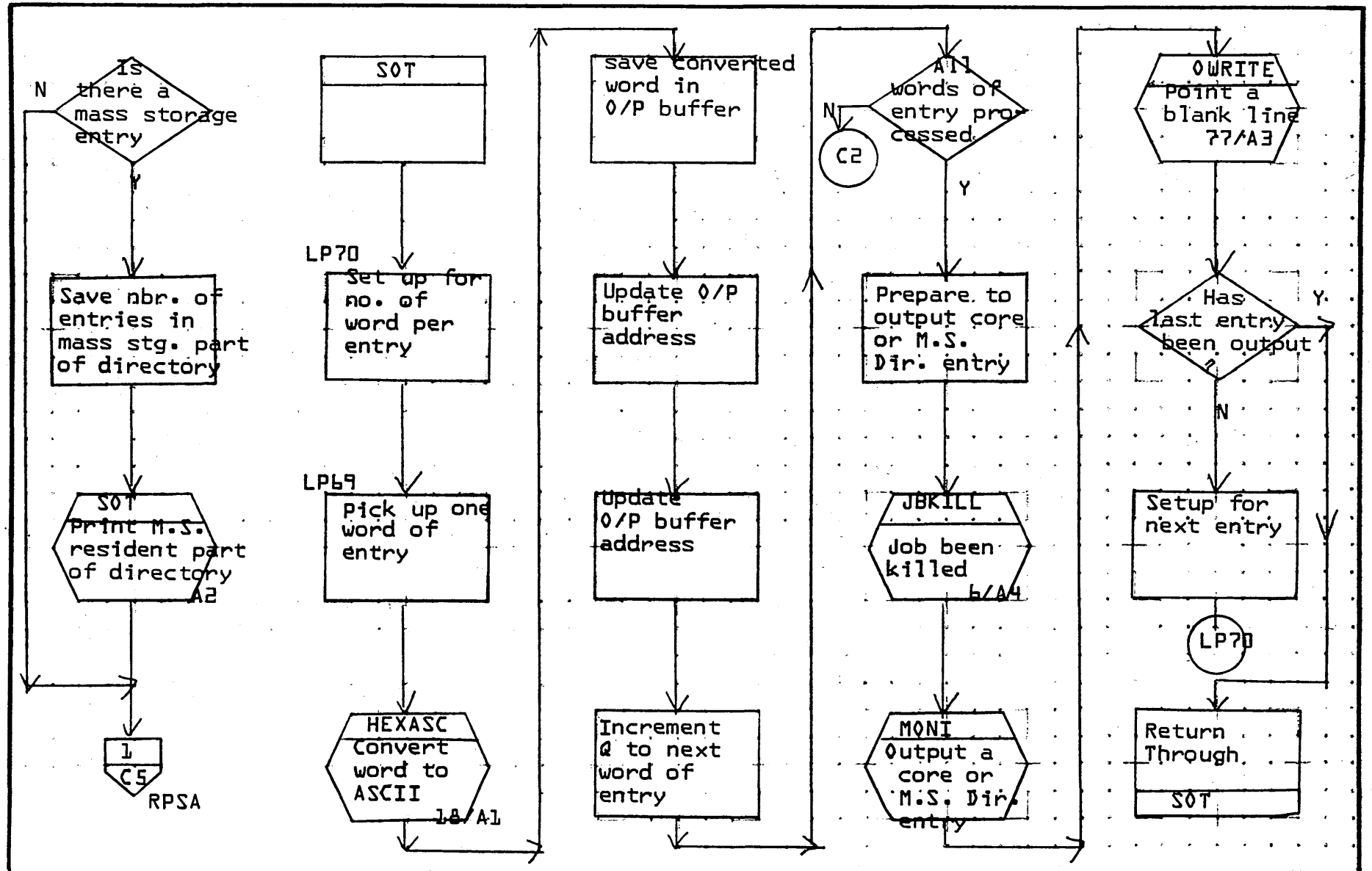
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1 pnS	MACH. TYPE 1760	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
		PAGE 16 OF 77	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.72

AA1385 (FORMERLY CA127-1)

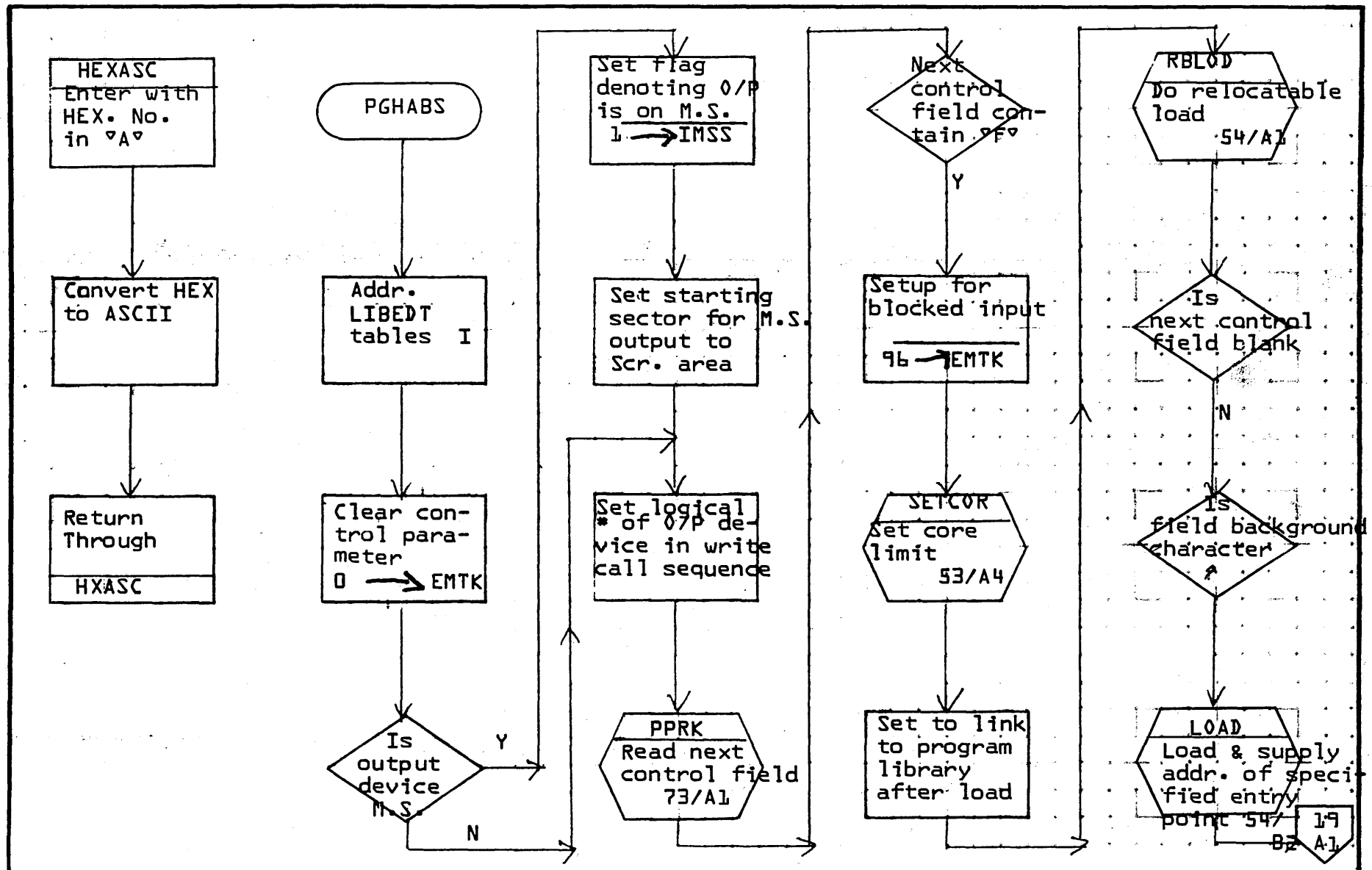
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 11	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	LIBEOT	PAGE 17 OF 77	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.73



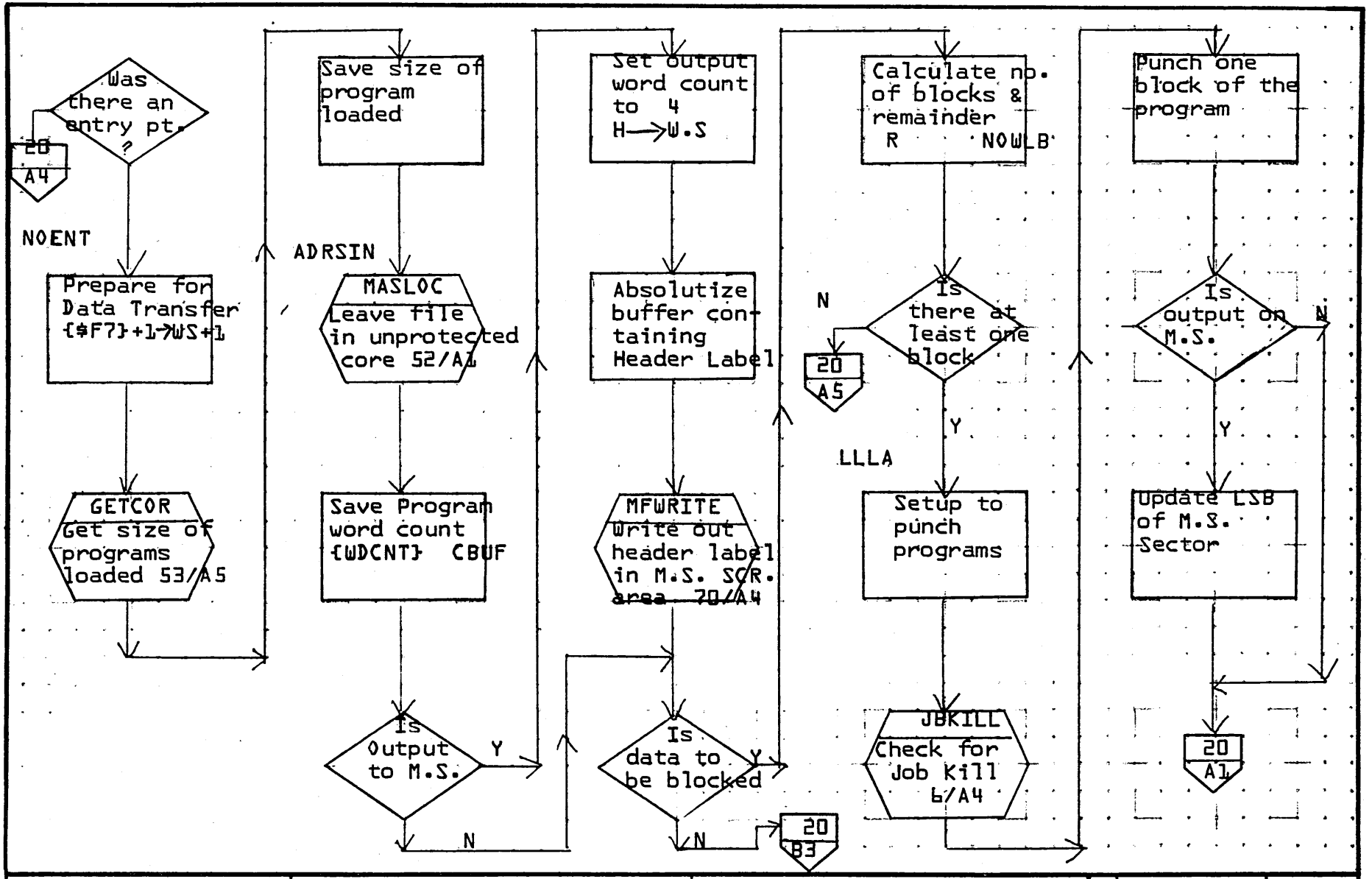
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	LIBEDT PAGE 18 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.74

AA1385 (FORMERLY CA127-1)

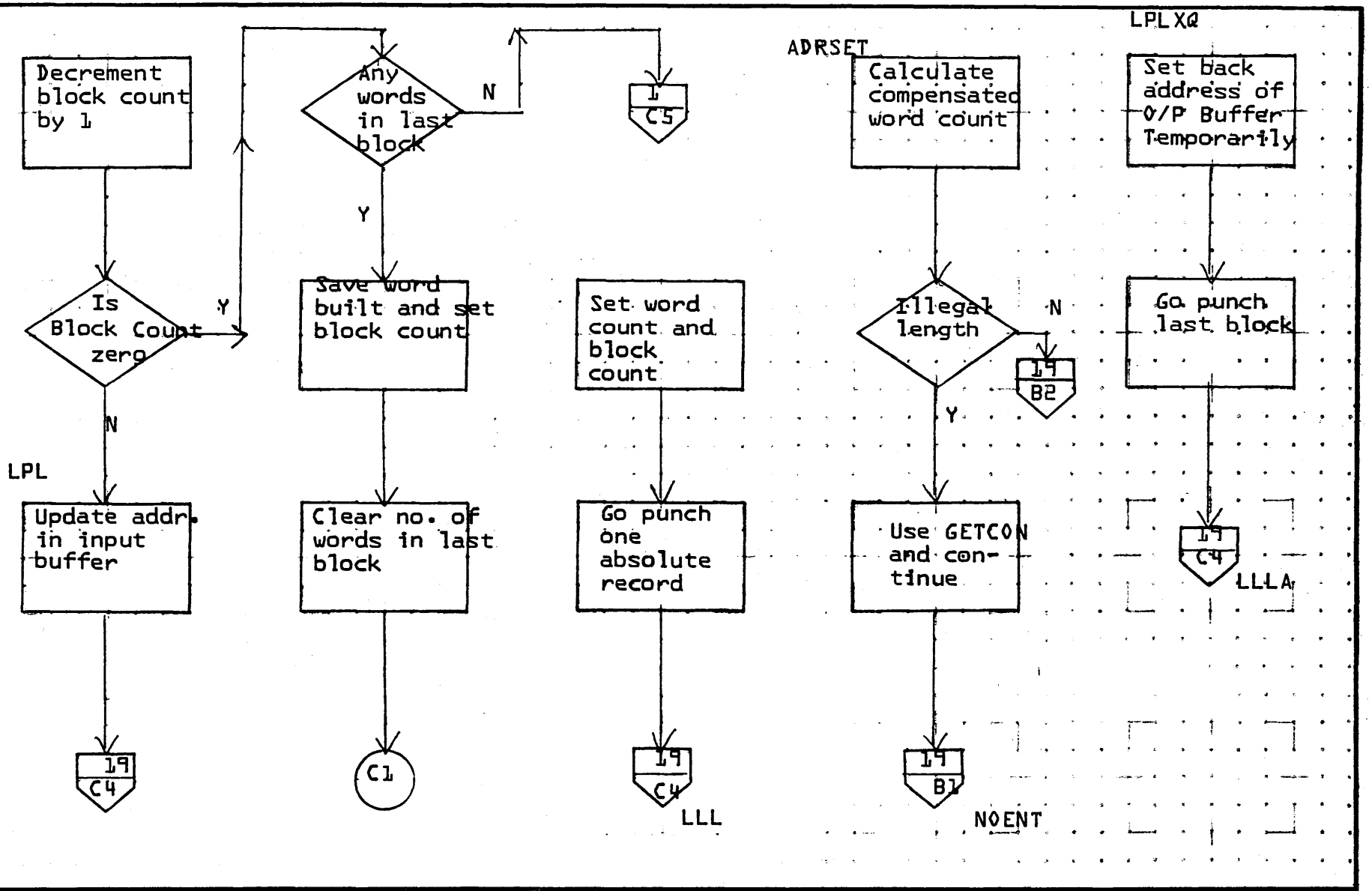
PRINTED IN U.S.A.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐	LIBEDT	PAGE 19 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.75



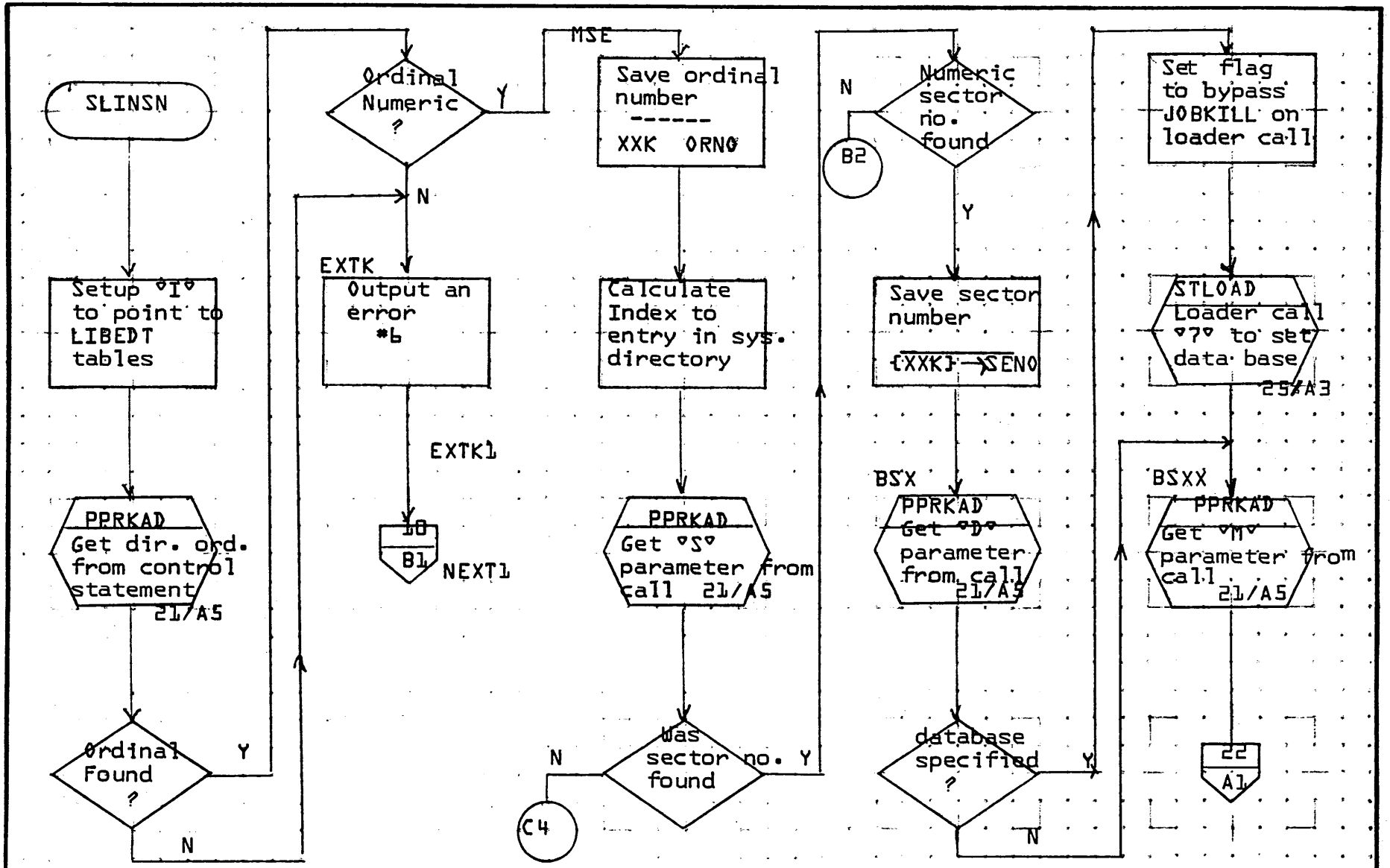
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1 MS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 20 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.74

AA1385 (FORMERLY CA127-1)

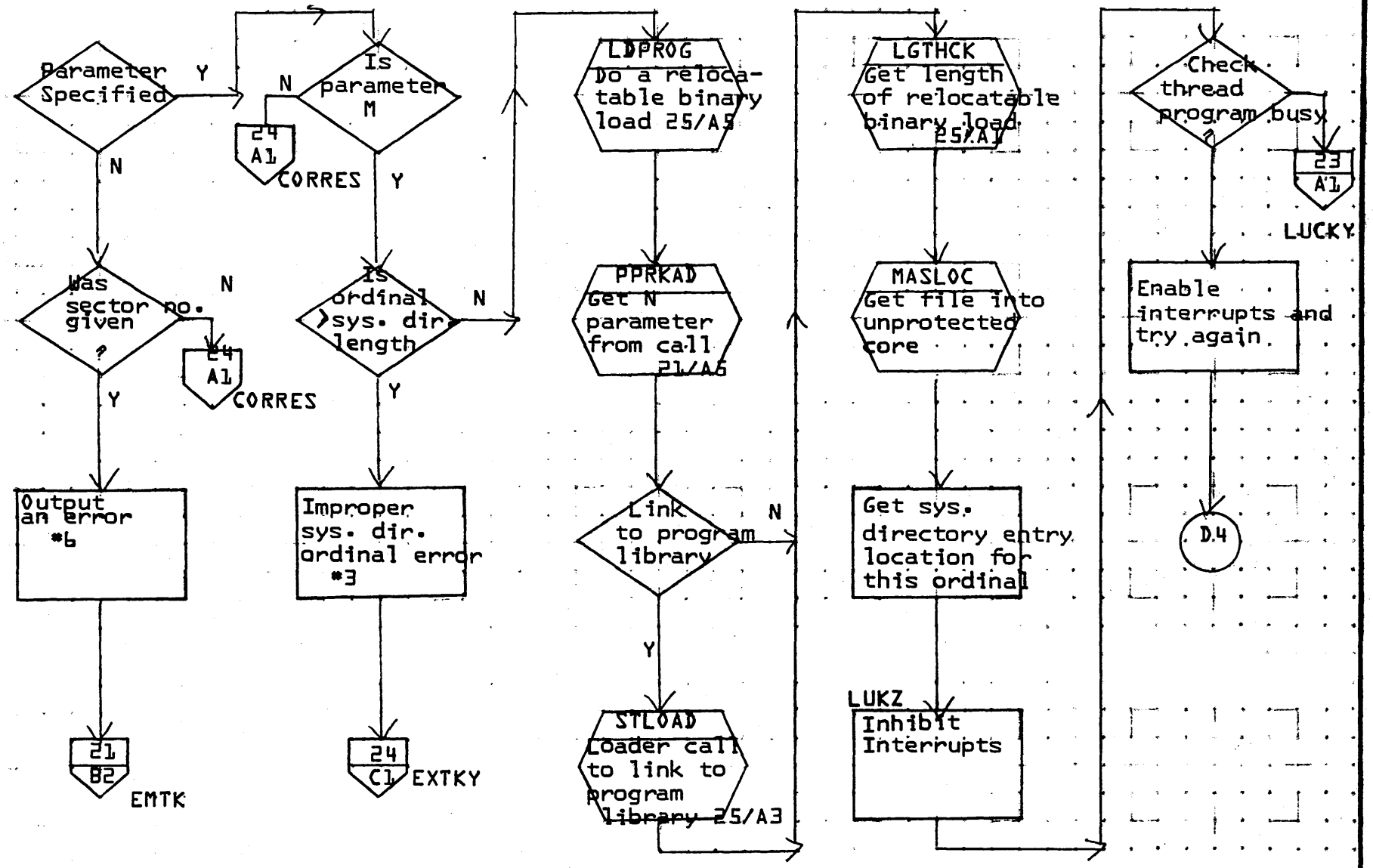
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	LIBEDT	PAGE 21 OF 77	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

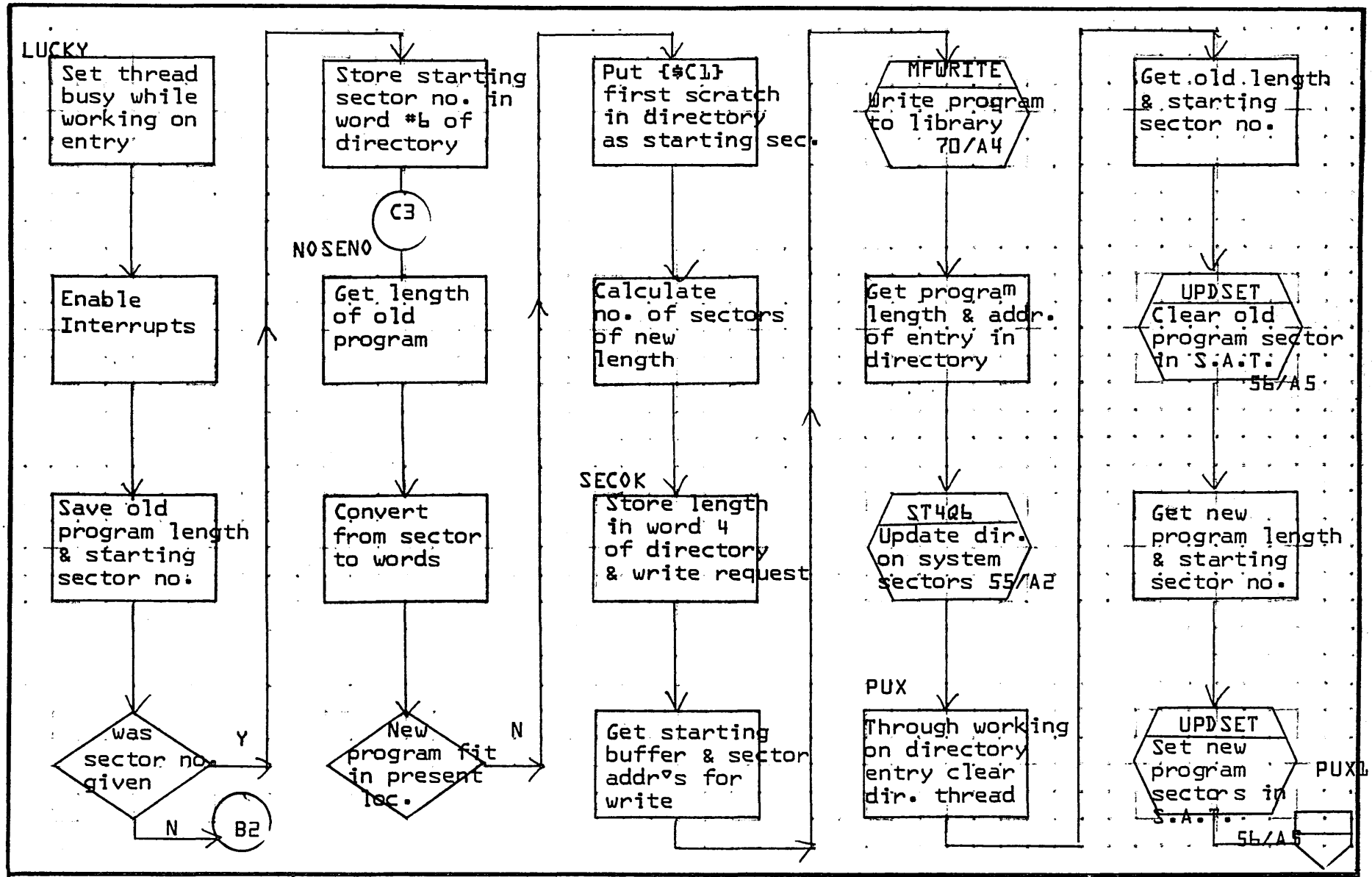
35.77



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 7000	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	LTBEDT	PAGE 22 OF 77	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.78



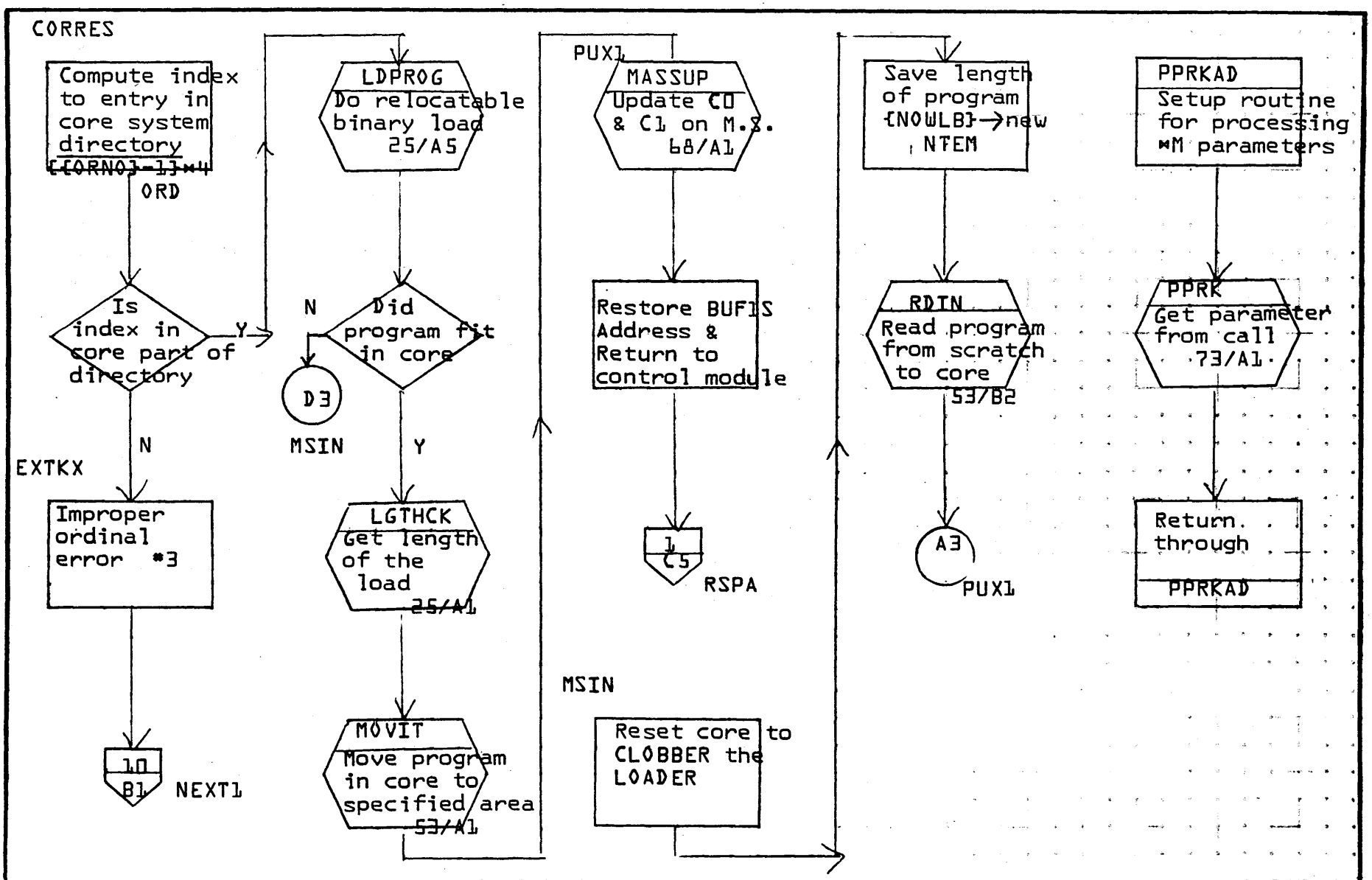
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS [illegible] MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	LIBEOT PAGE 23 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.79

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



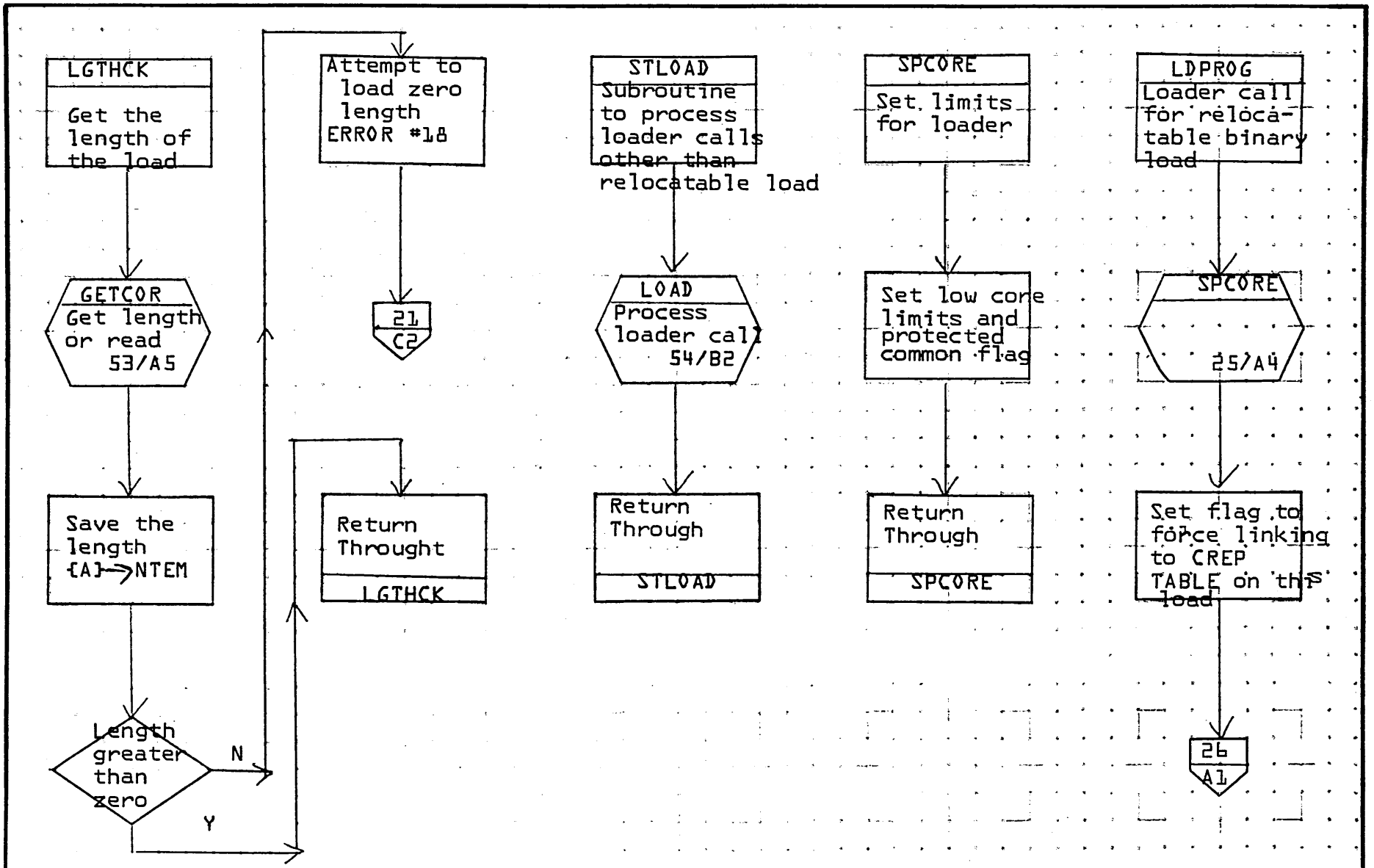
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBLOT	PAGE 34 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.80

AA1385 (FORMERLY CA127-1)

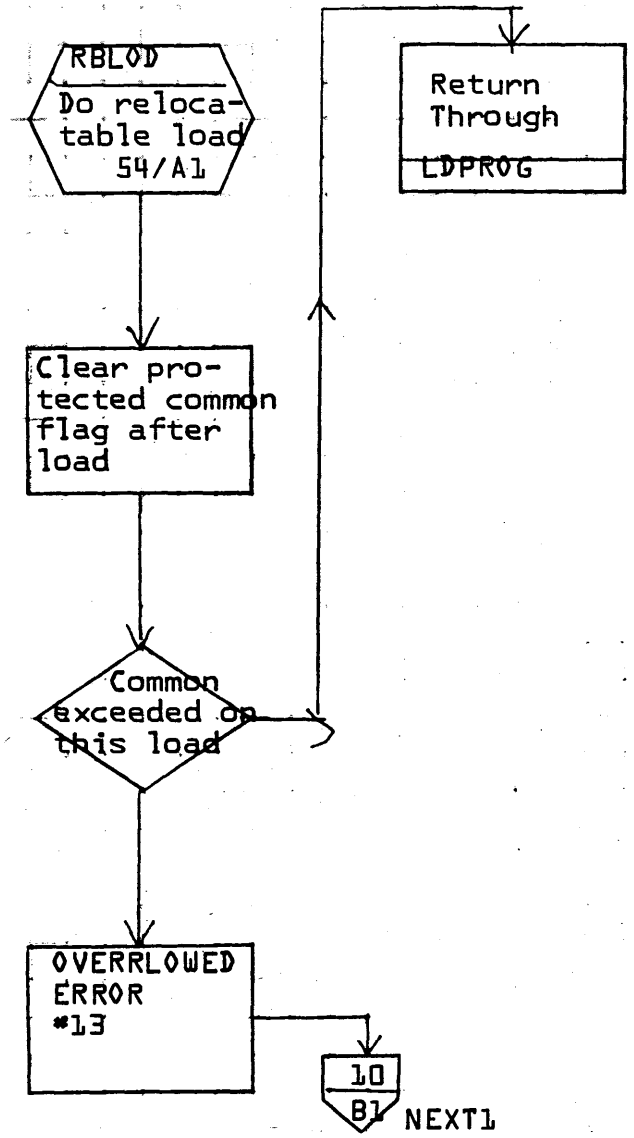
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LTBEDT	PAGE 25 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.81



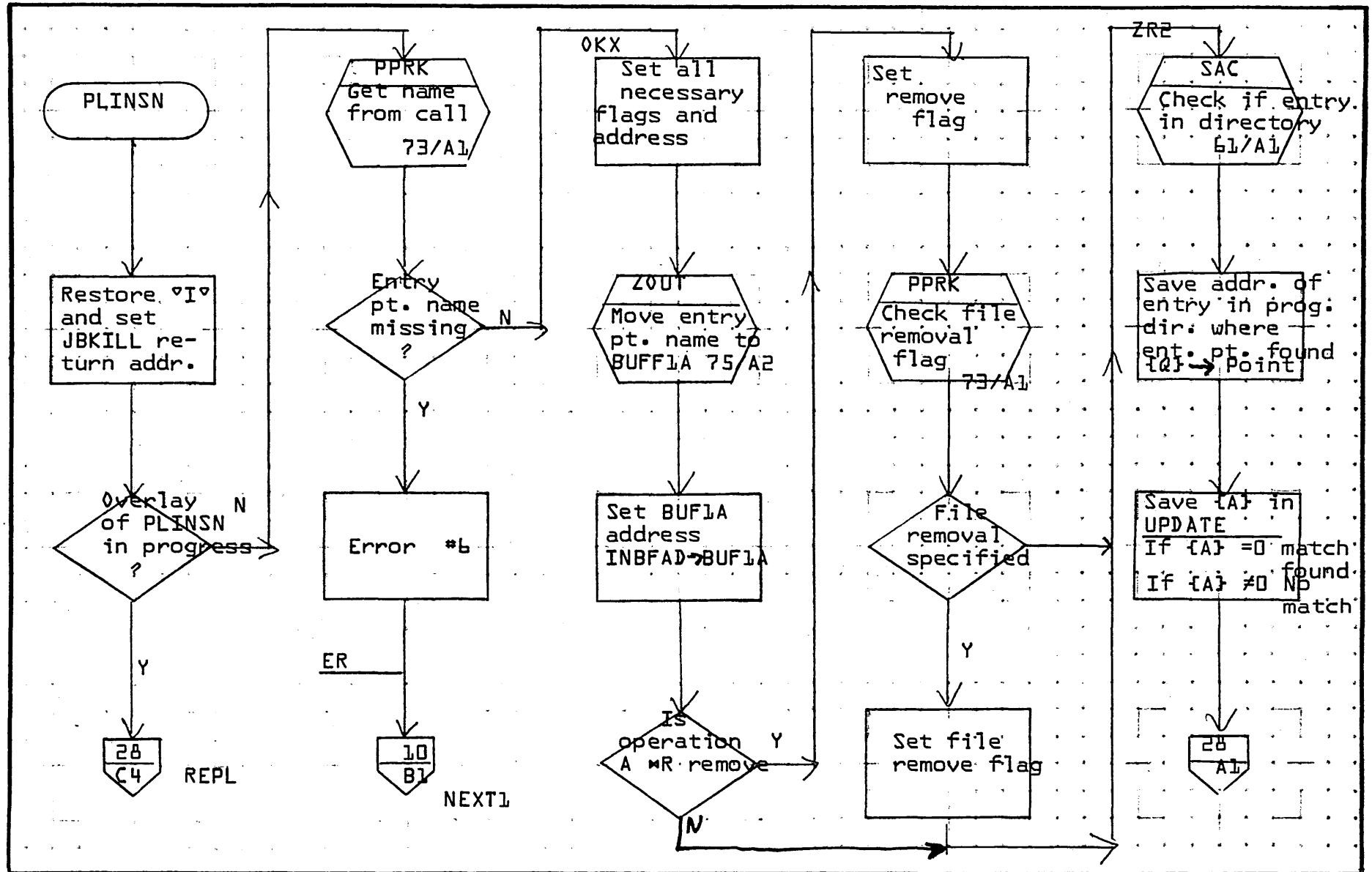
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐	LIBEDT	PAGE 26 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.82

AA1385 (FORMERLY CA127-1)

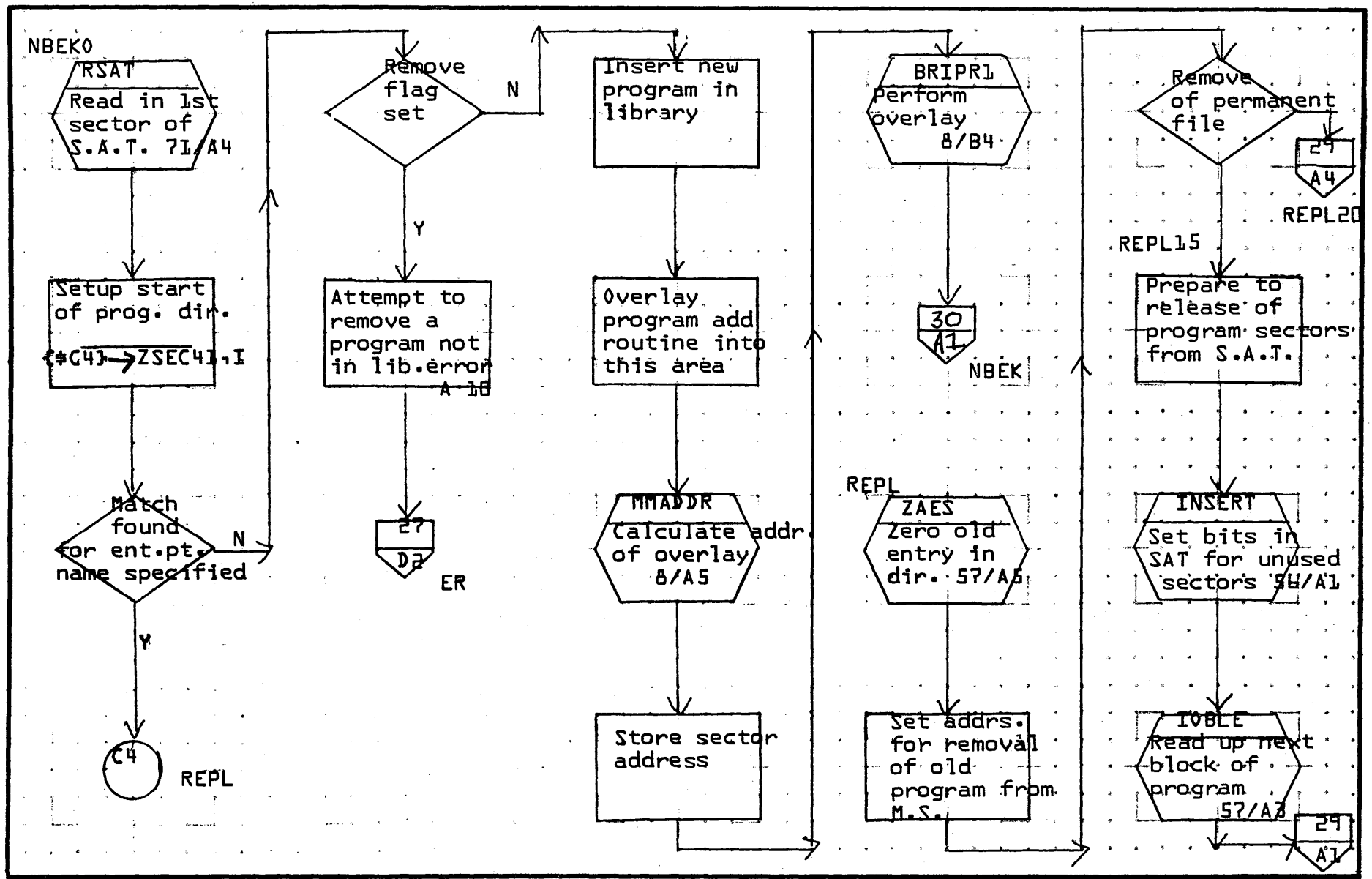
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	LIOEOT	PAGE 37 OF 77	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

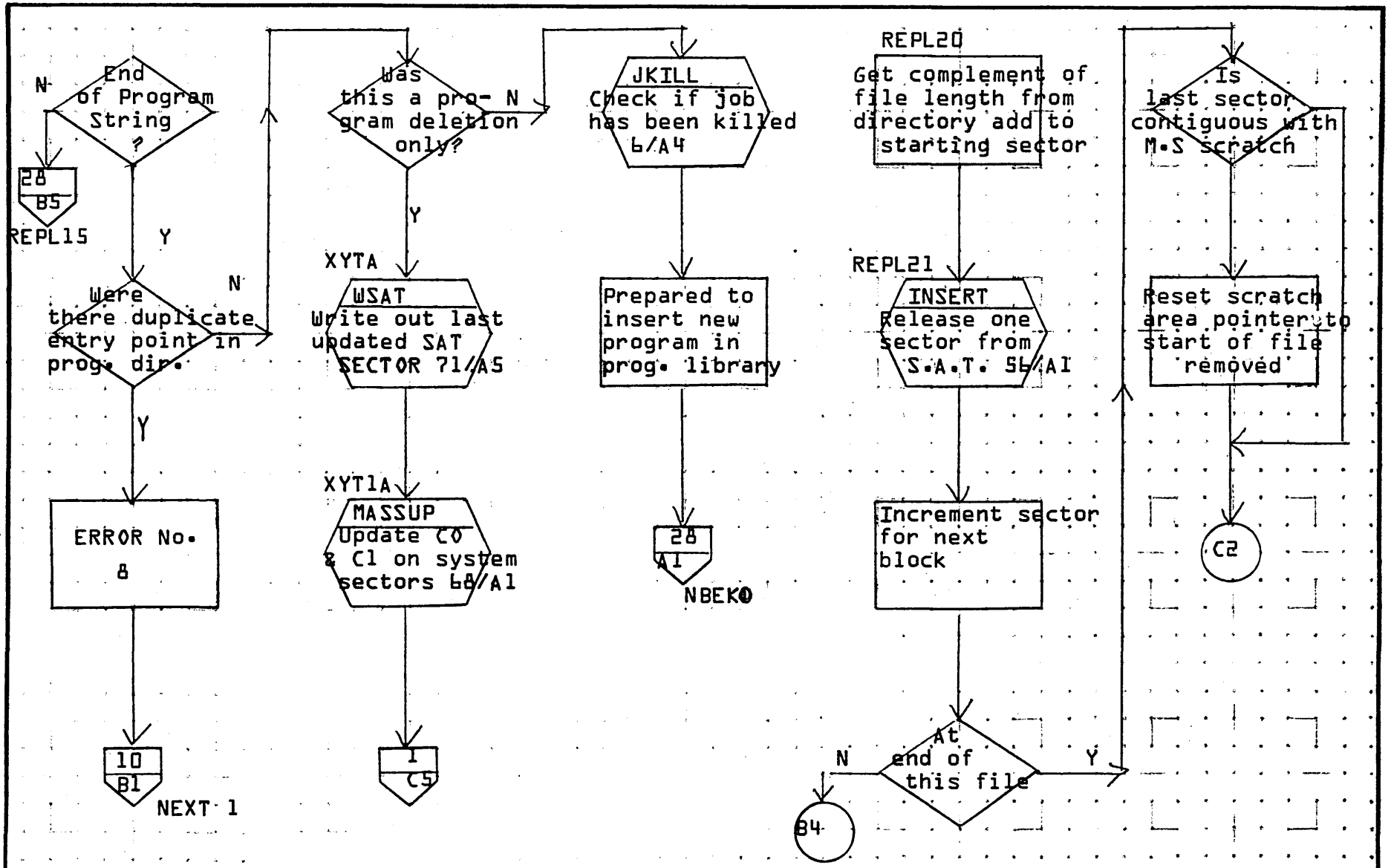
35.83



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDIT PAGE 28 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.84



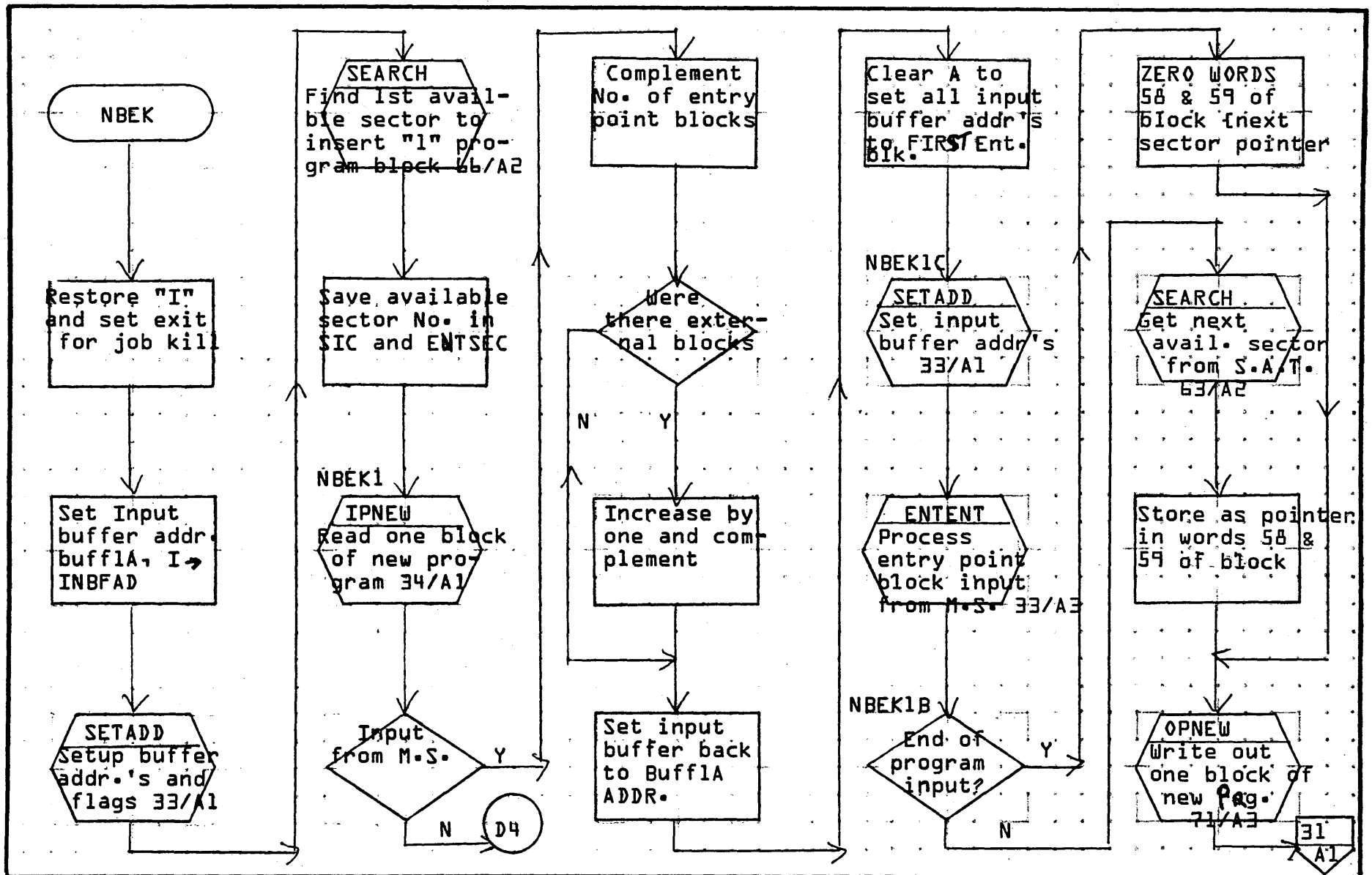
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS FMS	MACH. TYPE 7600	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐		PAGE 89 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.85

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



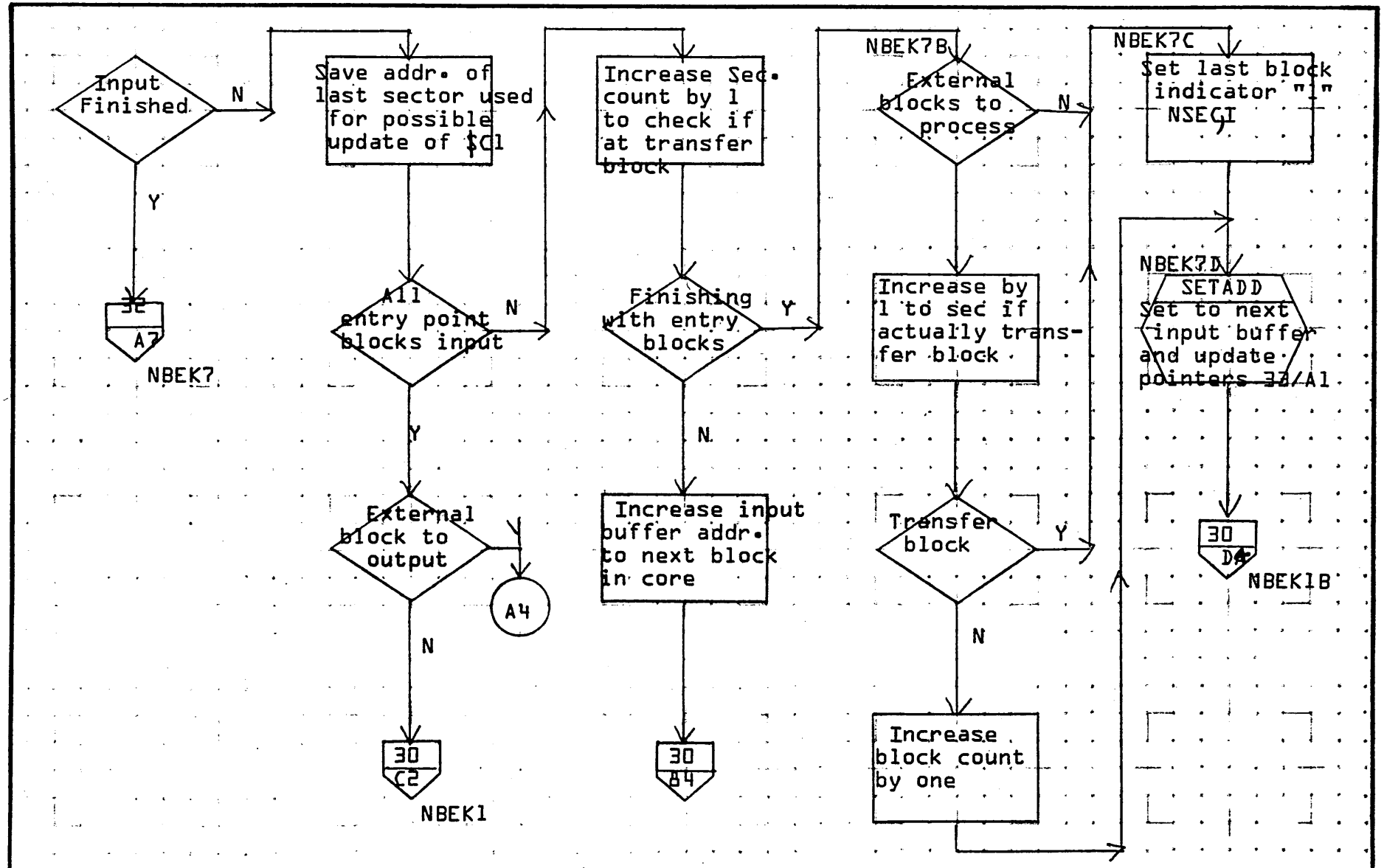
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐		PAGE 30 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

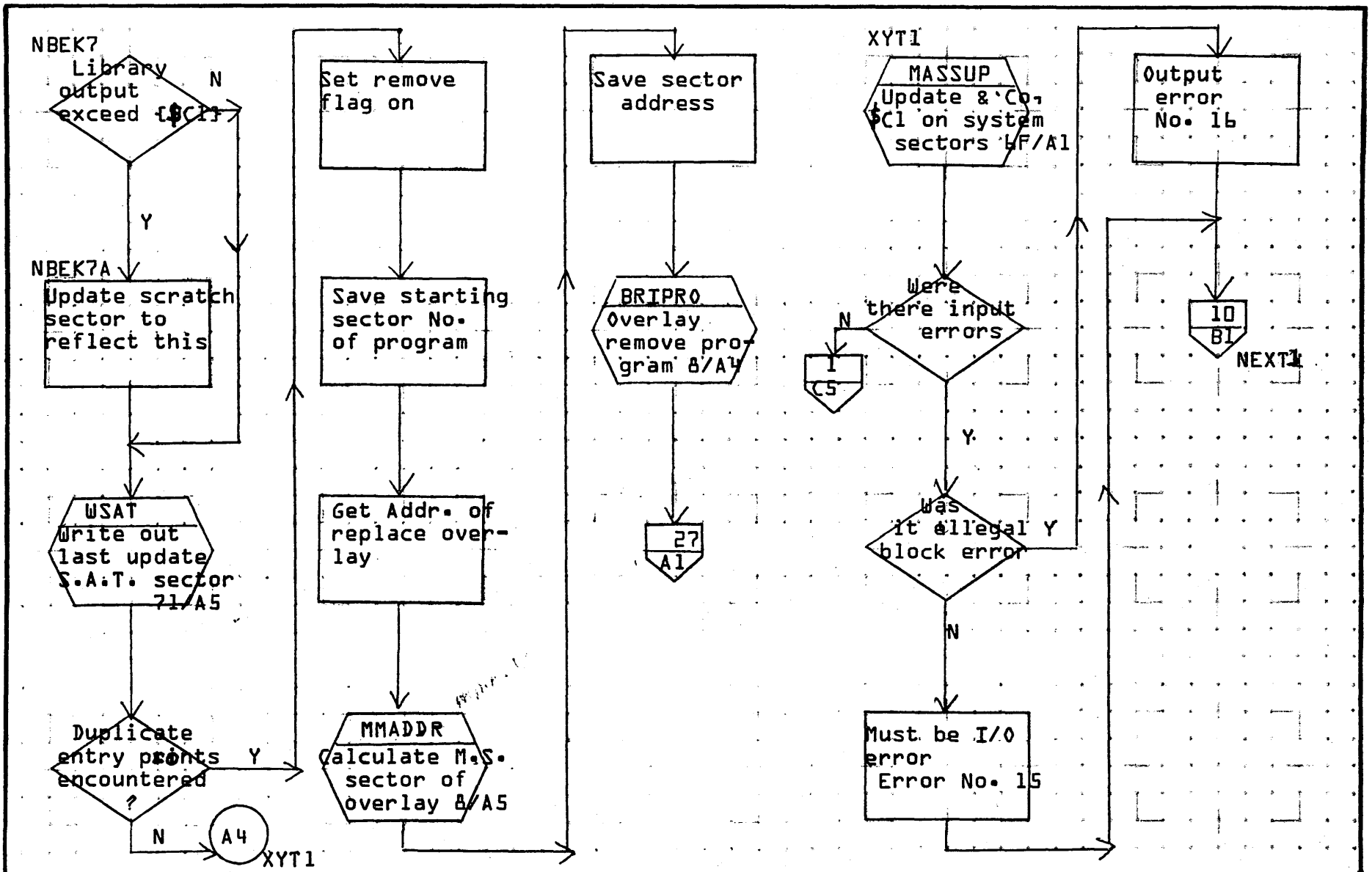
35.86



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Tribute	PAGE 31 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.87



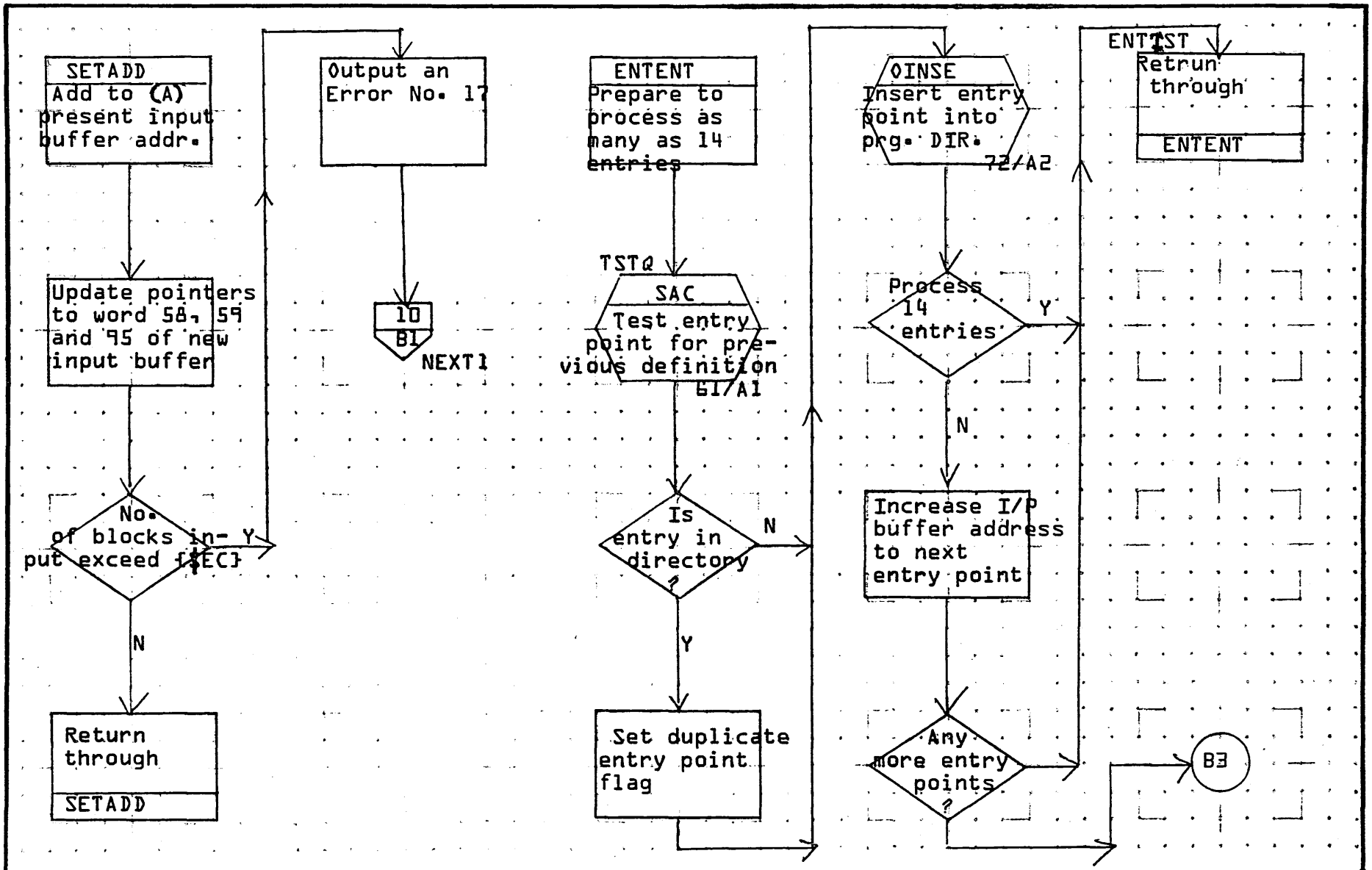
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS MACH. TYPE 1700 DOCUMENT TITLE PAGE 32 OF 77 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

MAR 5 1971

35.88

AA1385 (FORMERLY CA127-1)

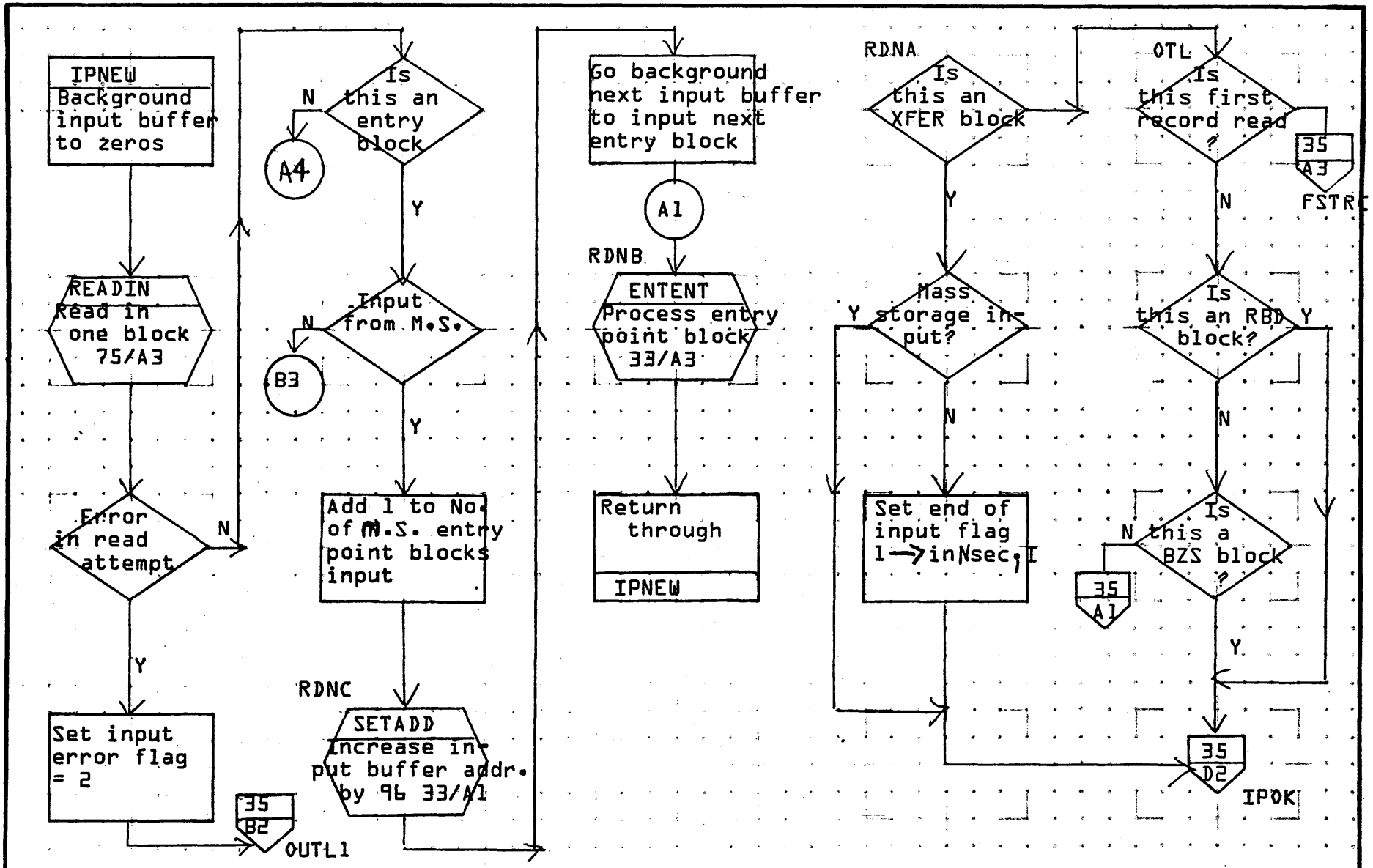
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Tibedt	PAGE 33 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.89



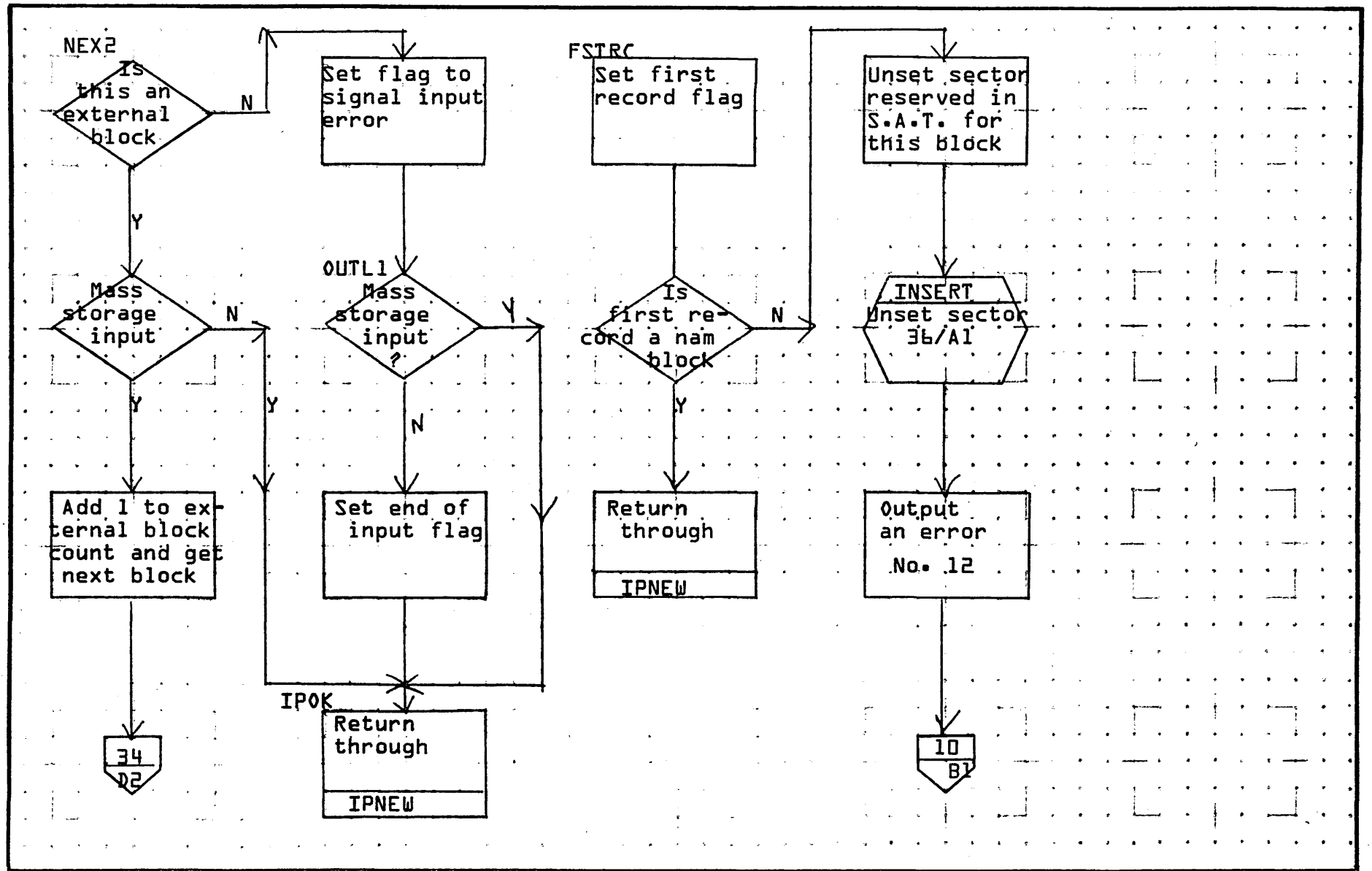
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐		PAGE 34 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.90

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



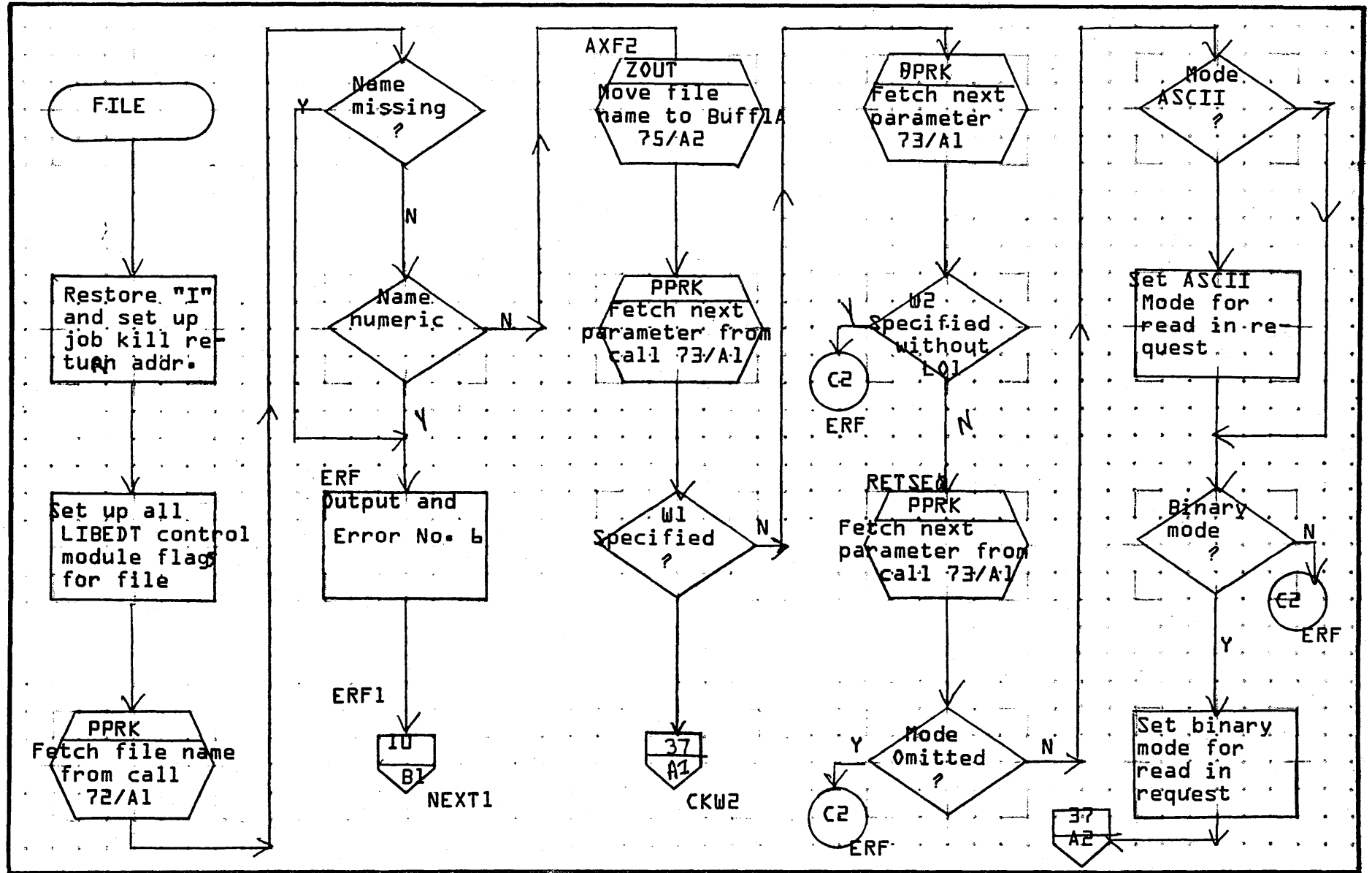
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 7000	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐		PAGE 35 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.91

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.



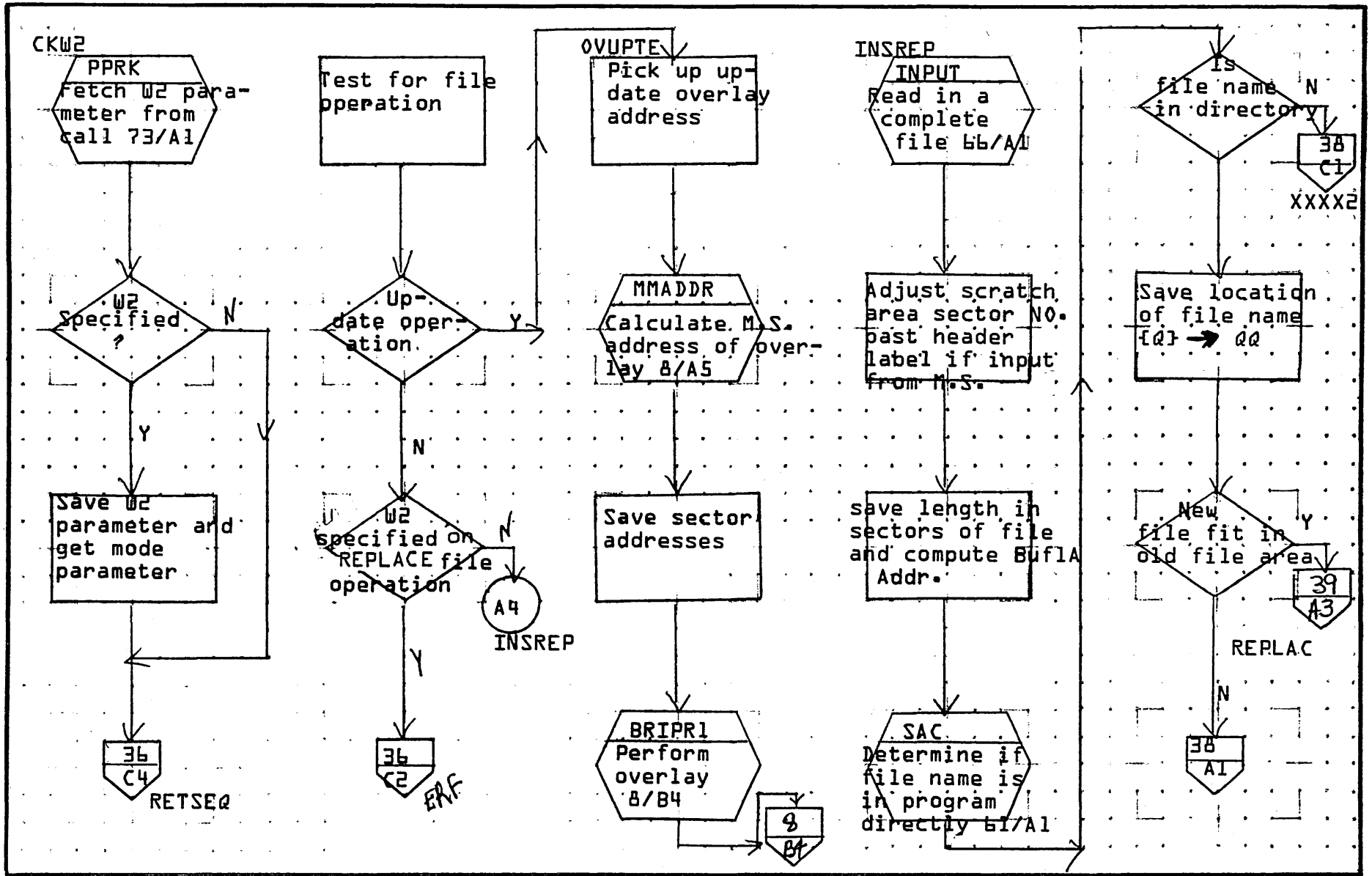
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: EMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐		PAGE 36 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.92

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

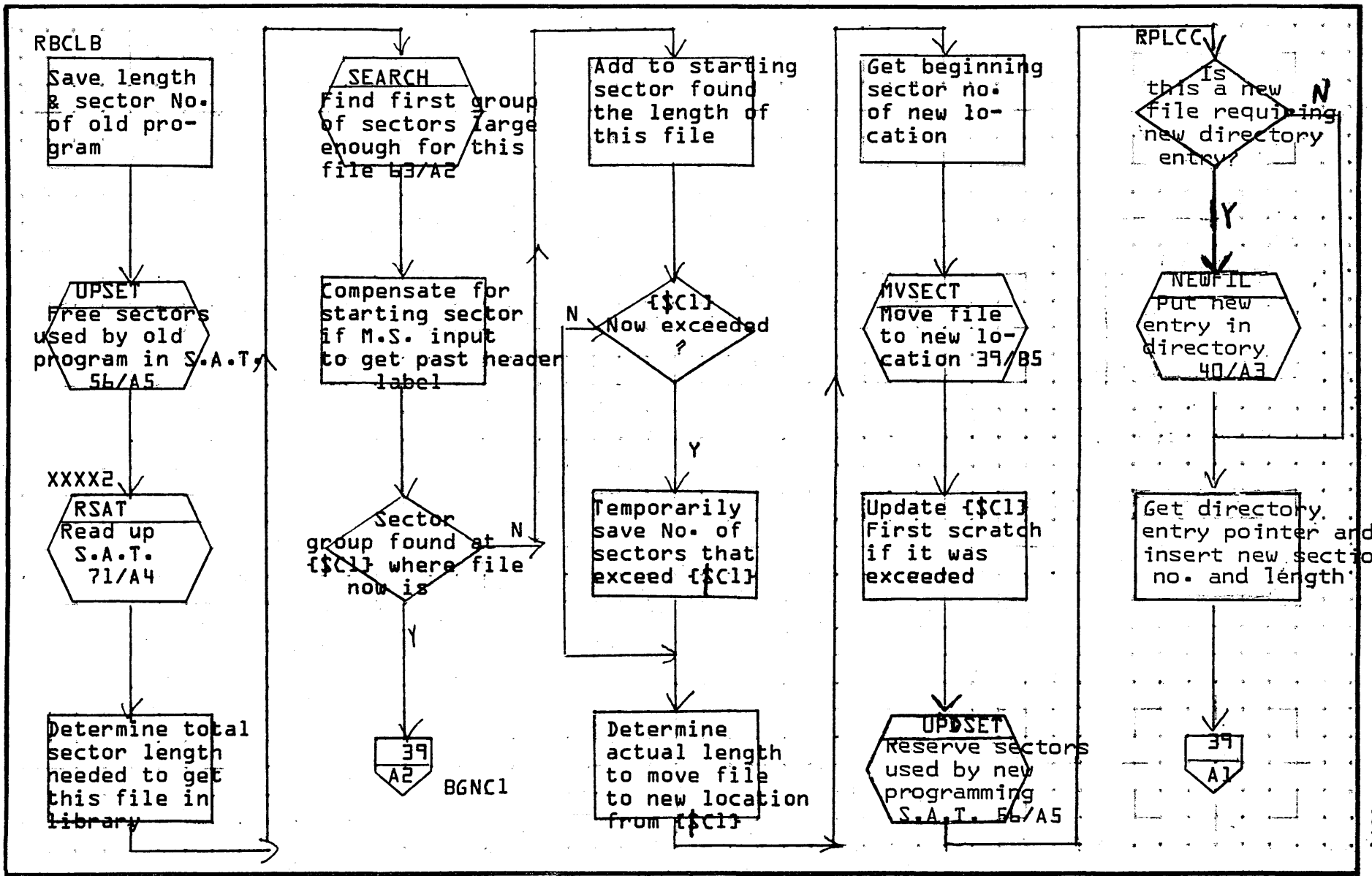


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Subset	PAGE 37 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

35.93



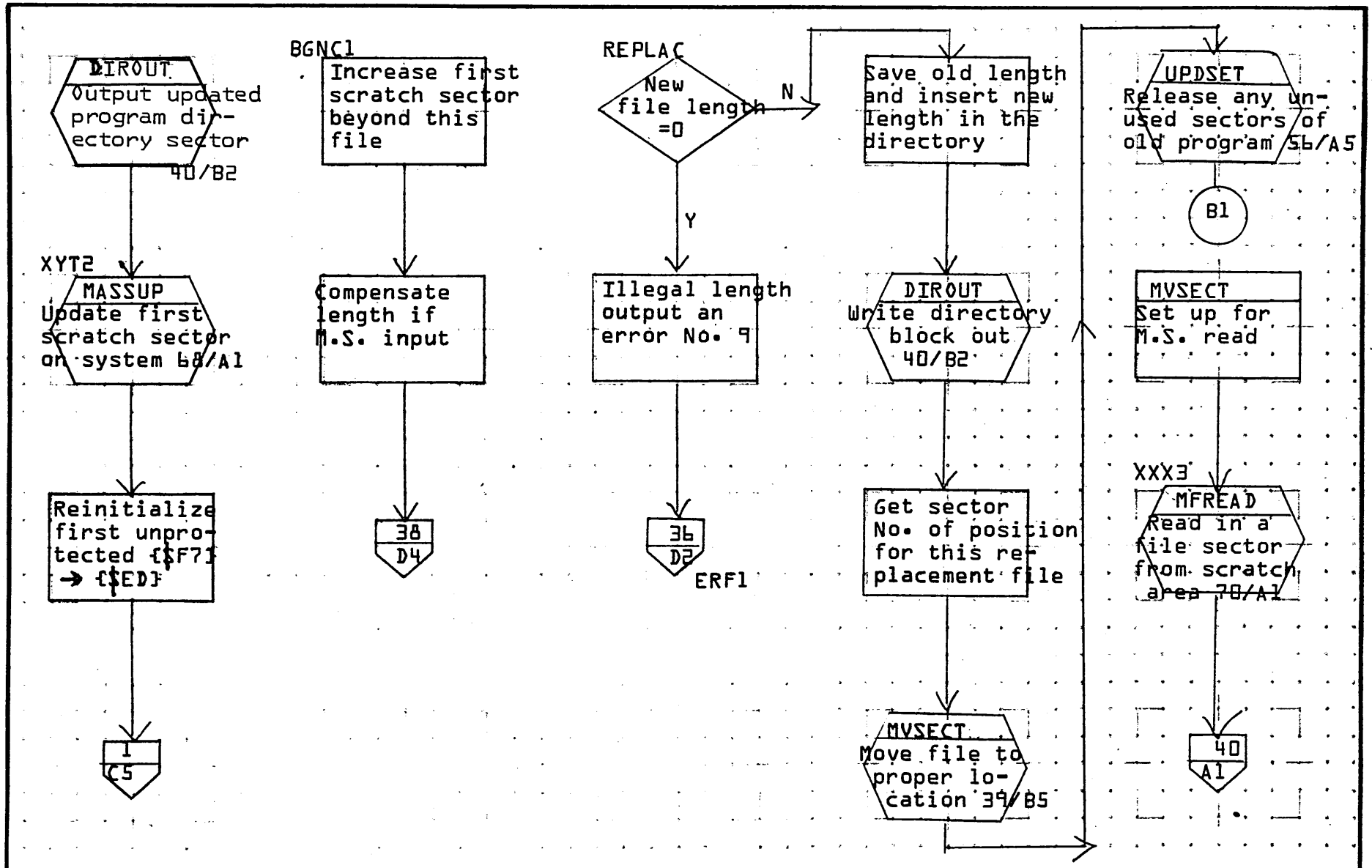
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐		PAGE 38 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.94

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



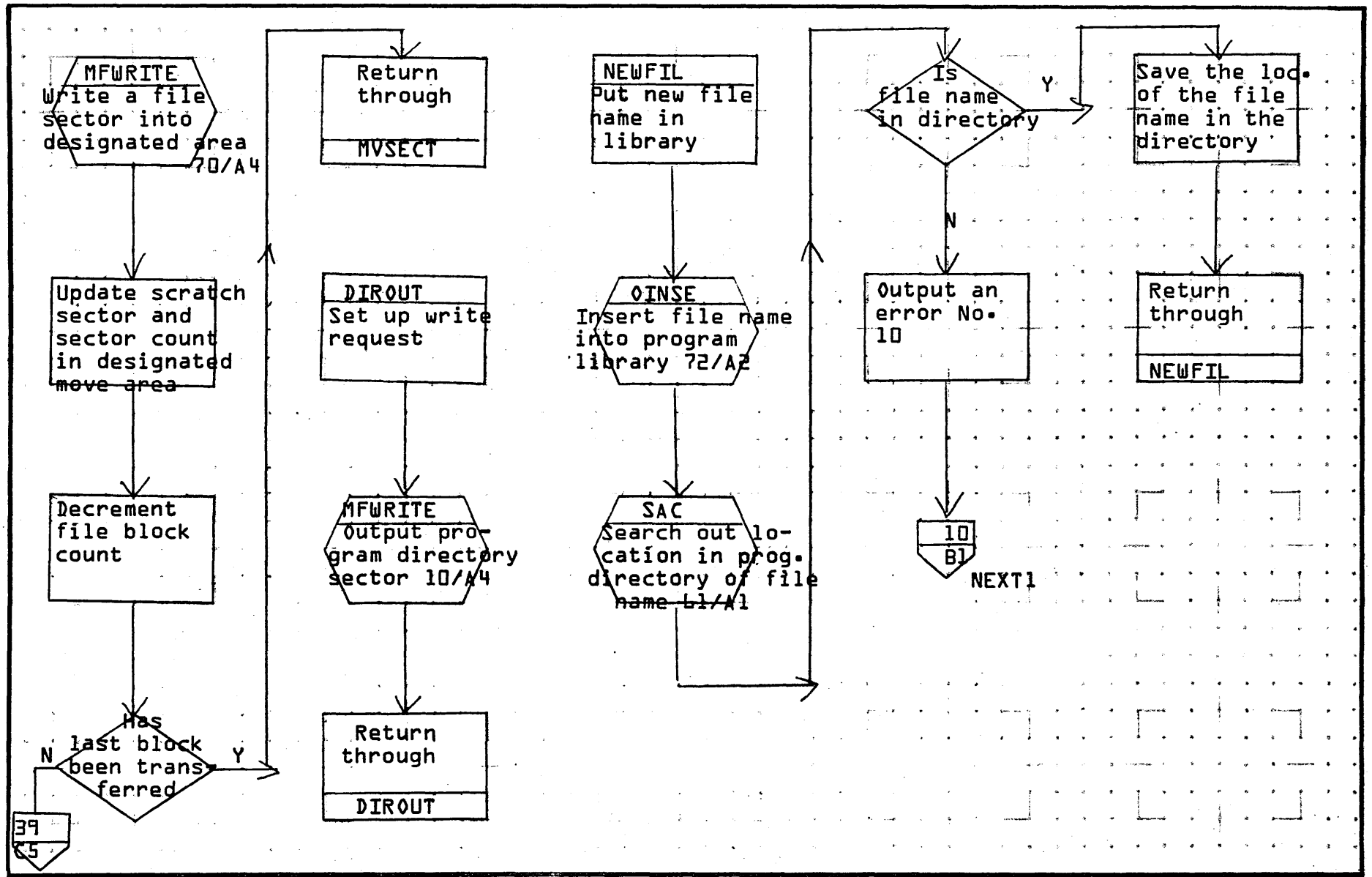
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	Title Of PAGE 37 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.95

AA1385 (FORMERLY CA127-1)

PRINTED IN USA



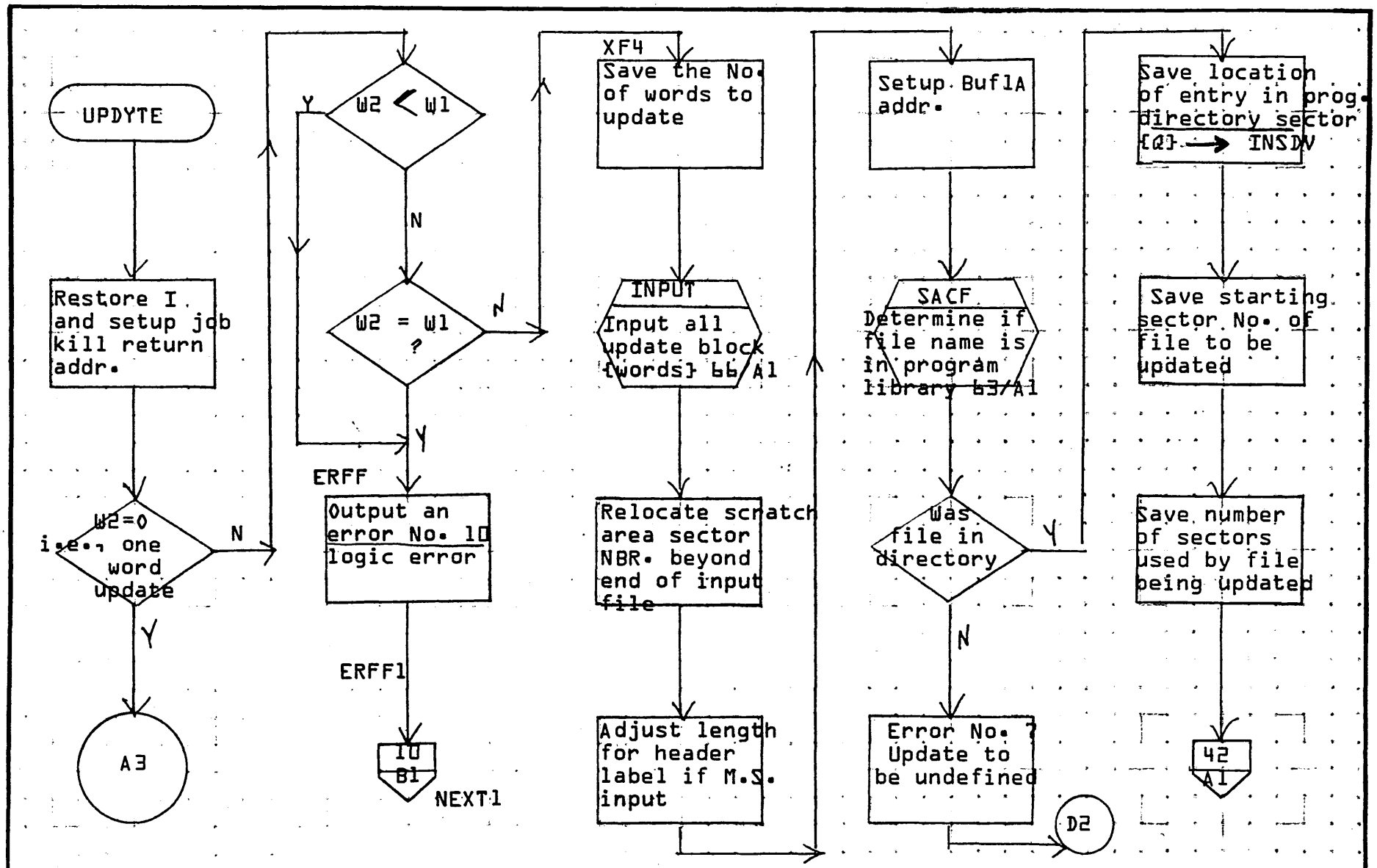
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 7700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	Subset	PAGE 40 OF 77	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.96

AA1385 (FORMERLY CA127-1)

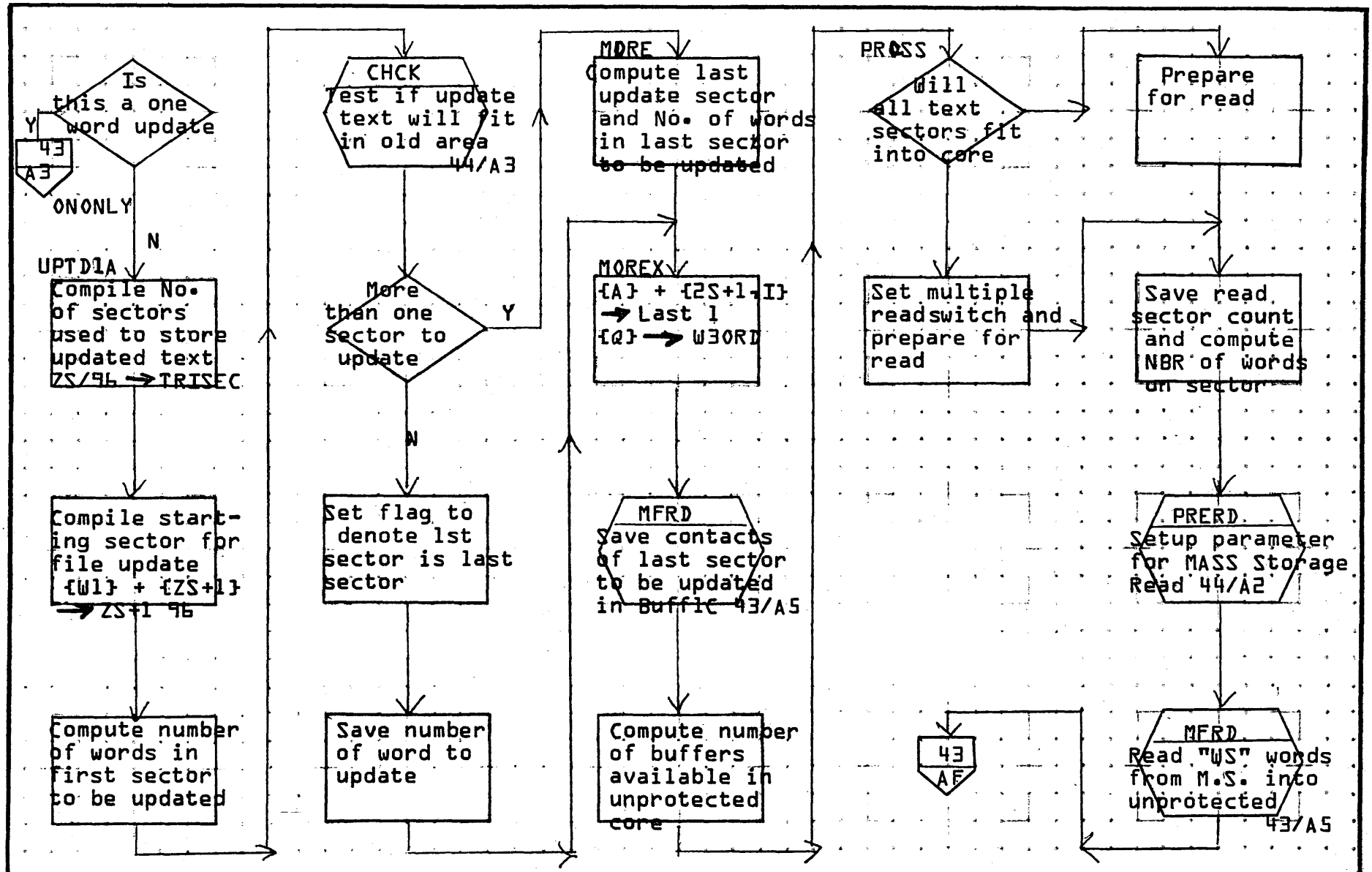
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS JMS	MACH. TYPE 7600	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	Tube Ct	PAGE 44 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.97



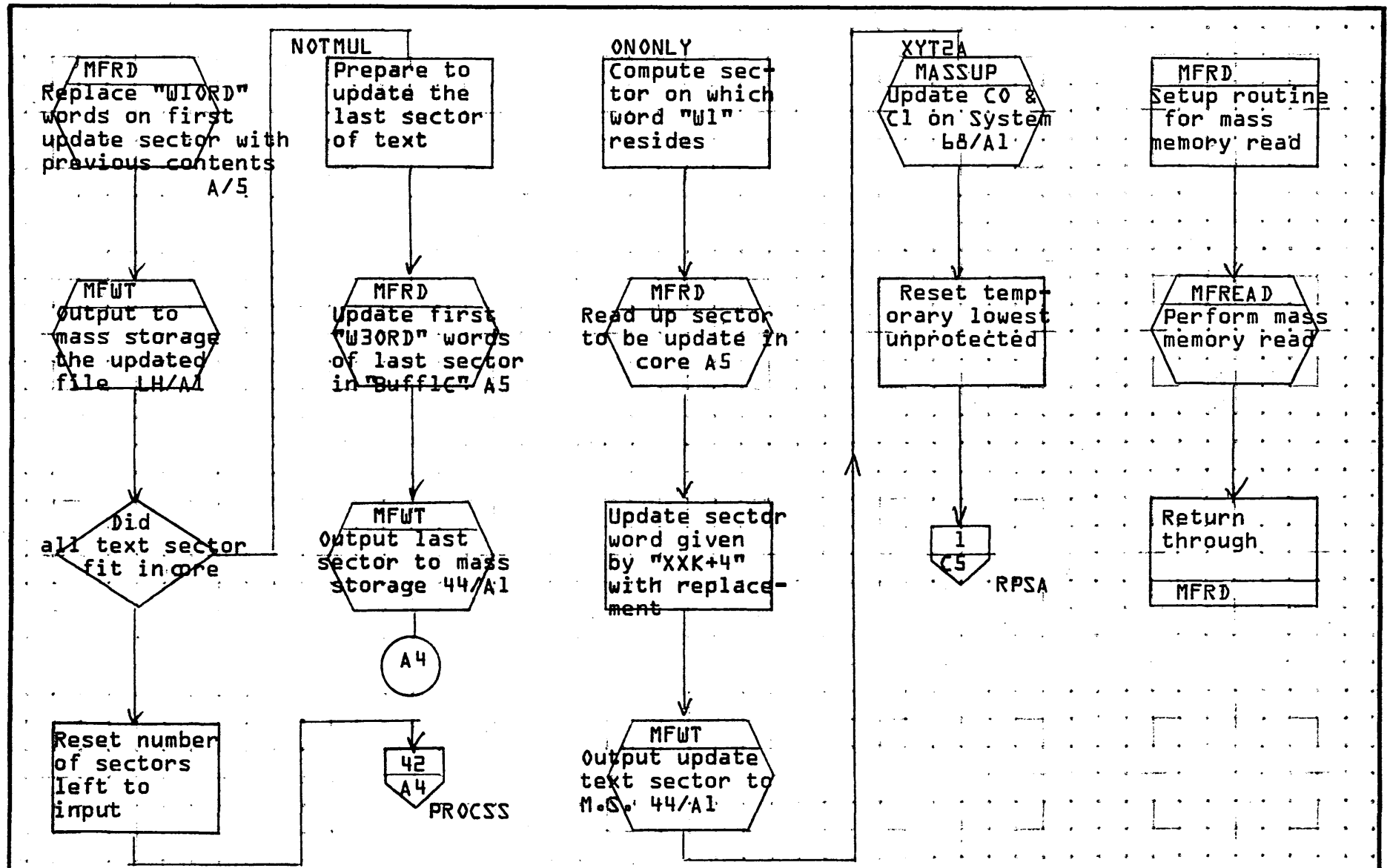
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 42 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.98

AA1385 (FORMERLY CA127-1)

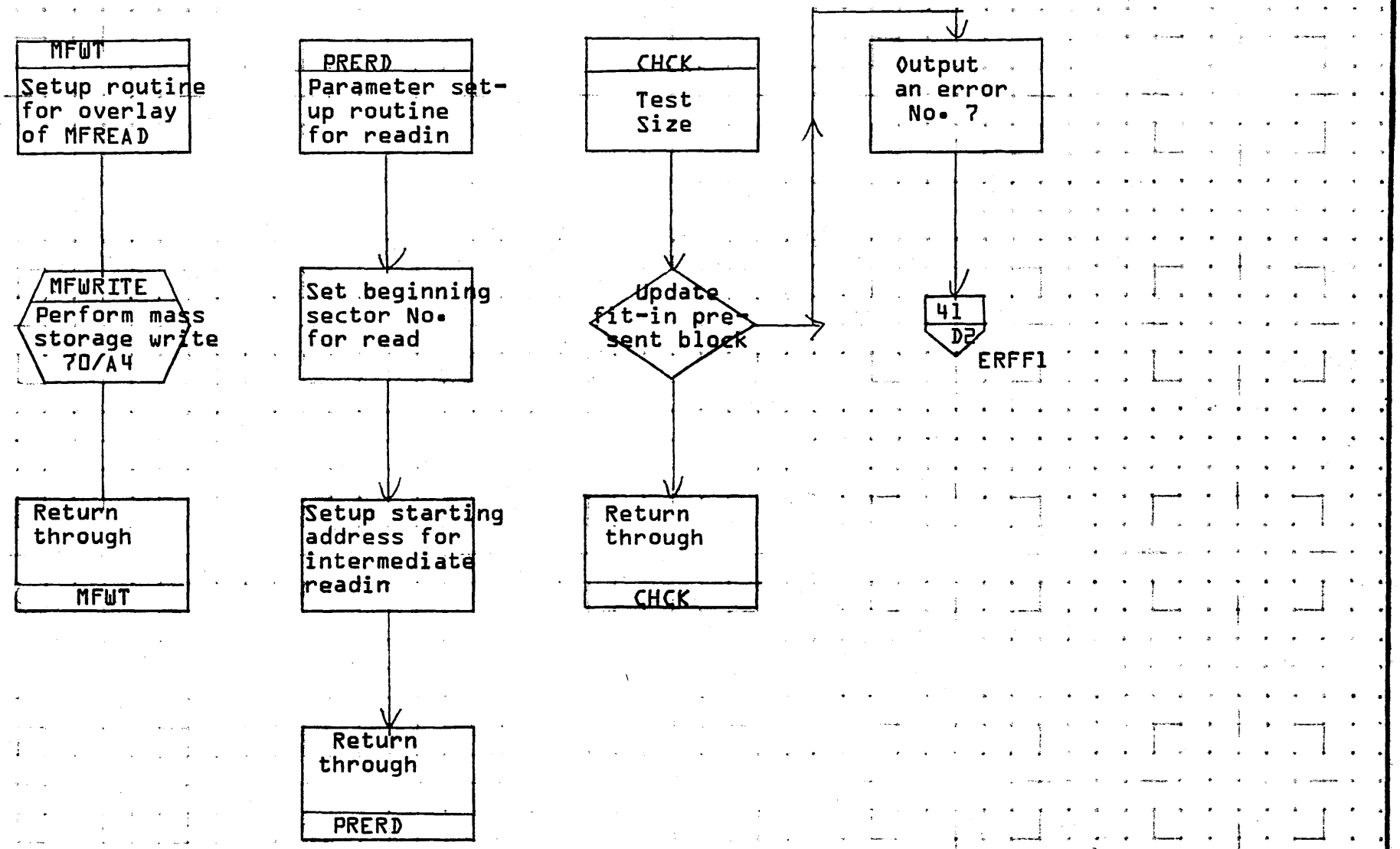
PRINTED IN U.S.A.



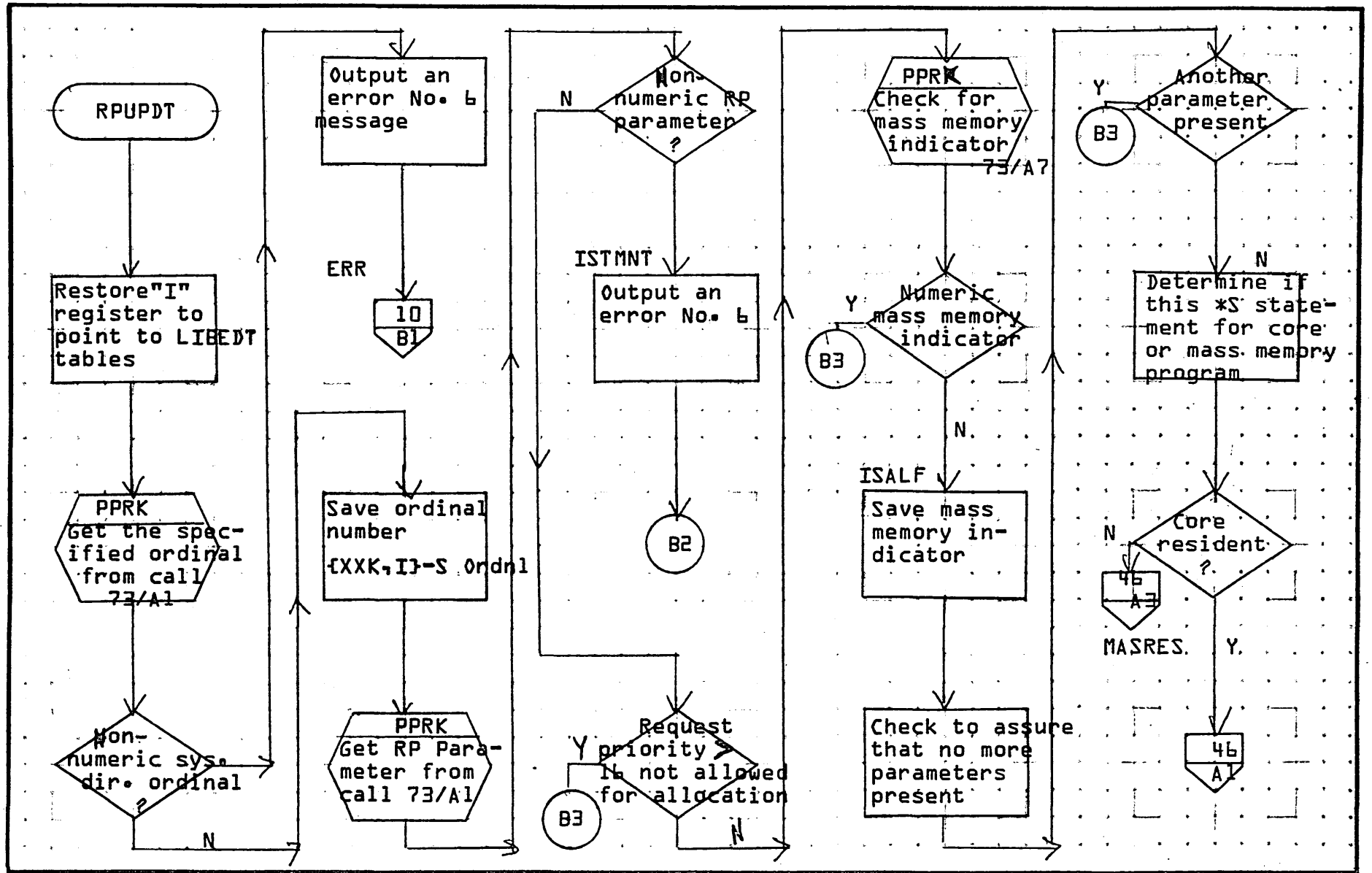
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE M00	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBDT	PAGE 48 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.99



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 44 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			



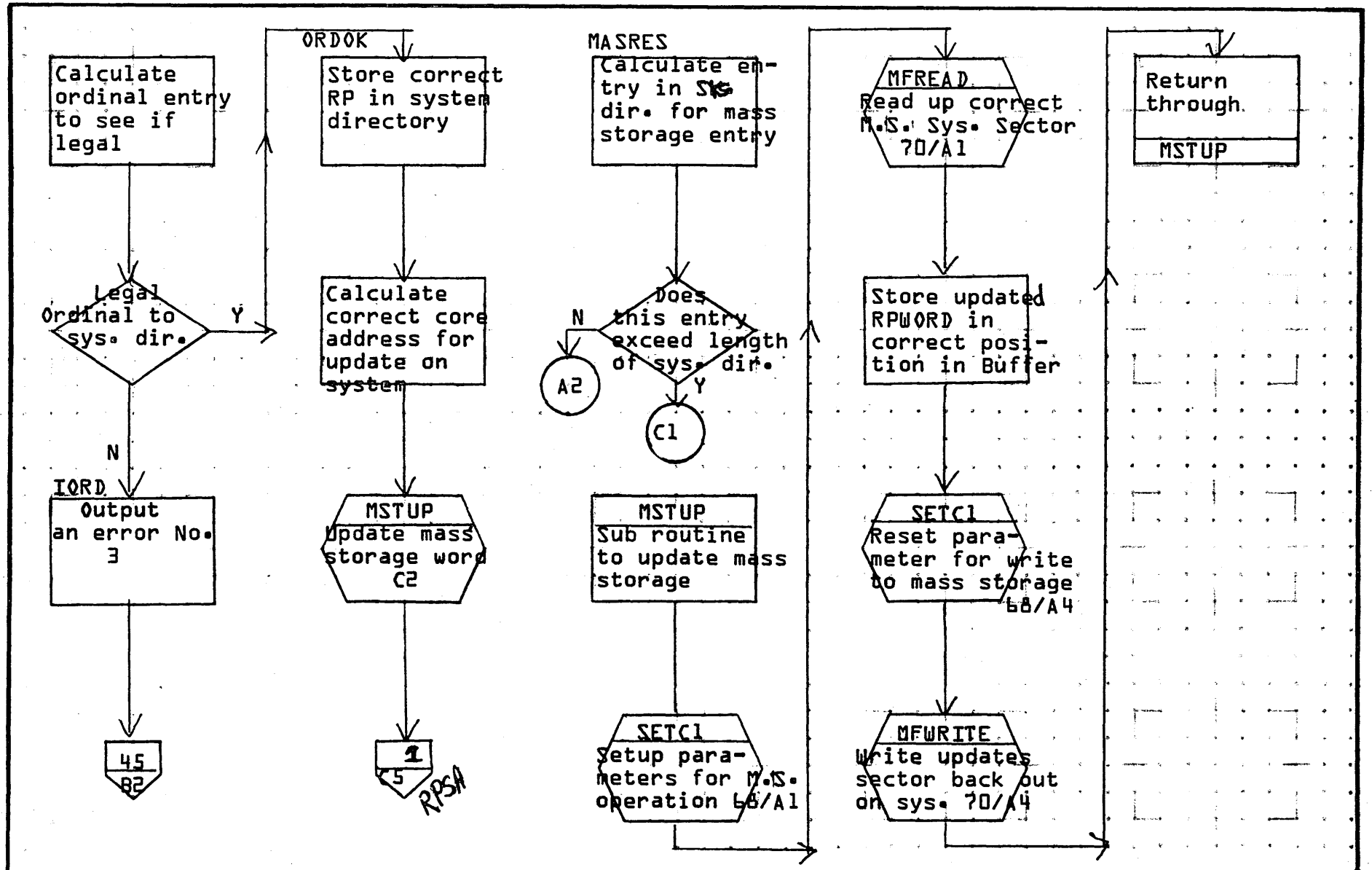
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 46 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.101

AA1385 (FORMERLY CA127-1)

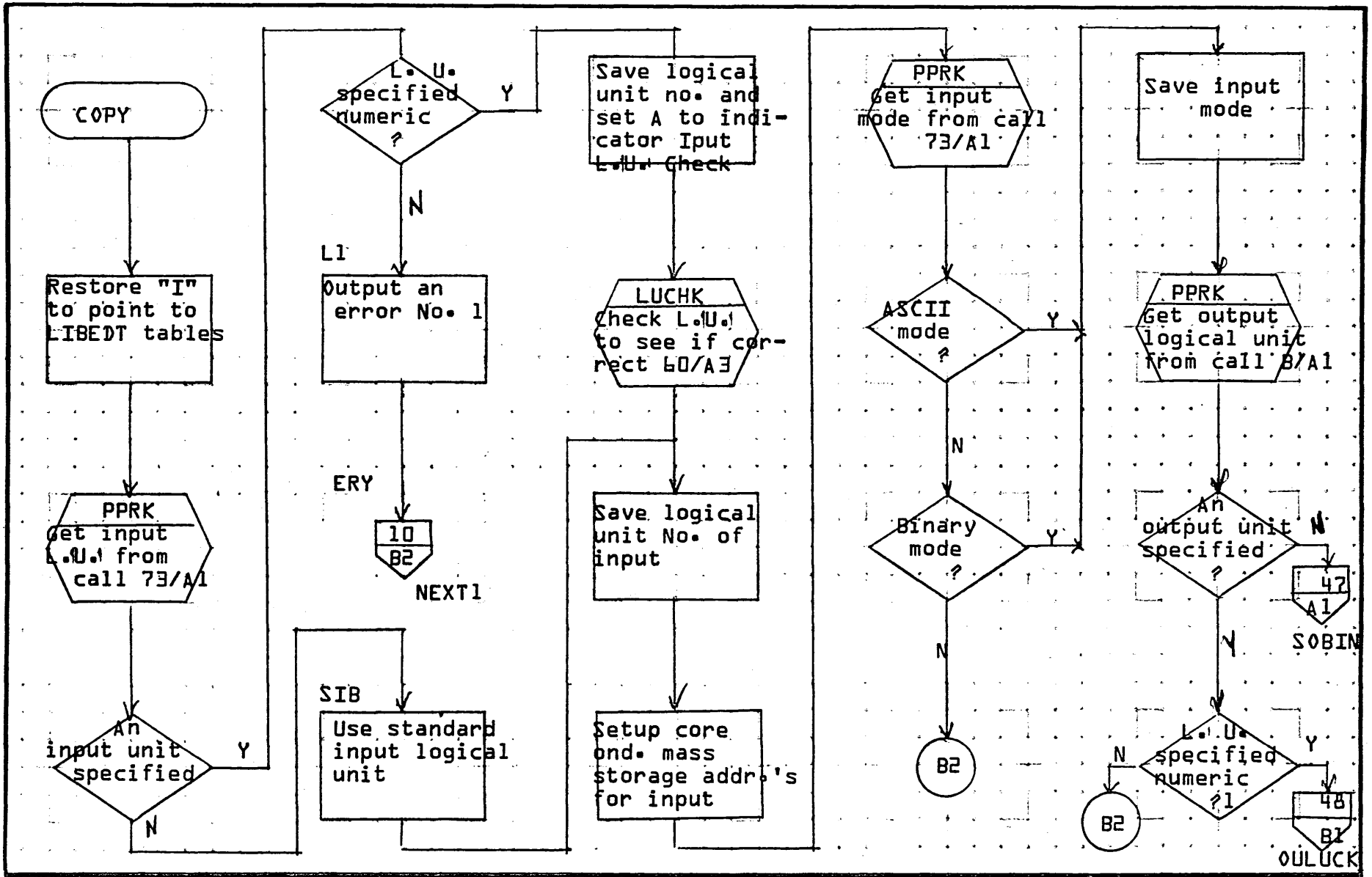
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 7600	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 46 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.102



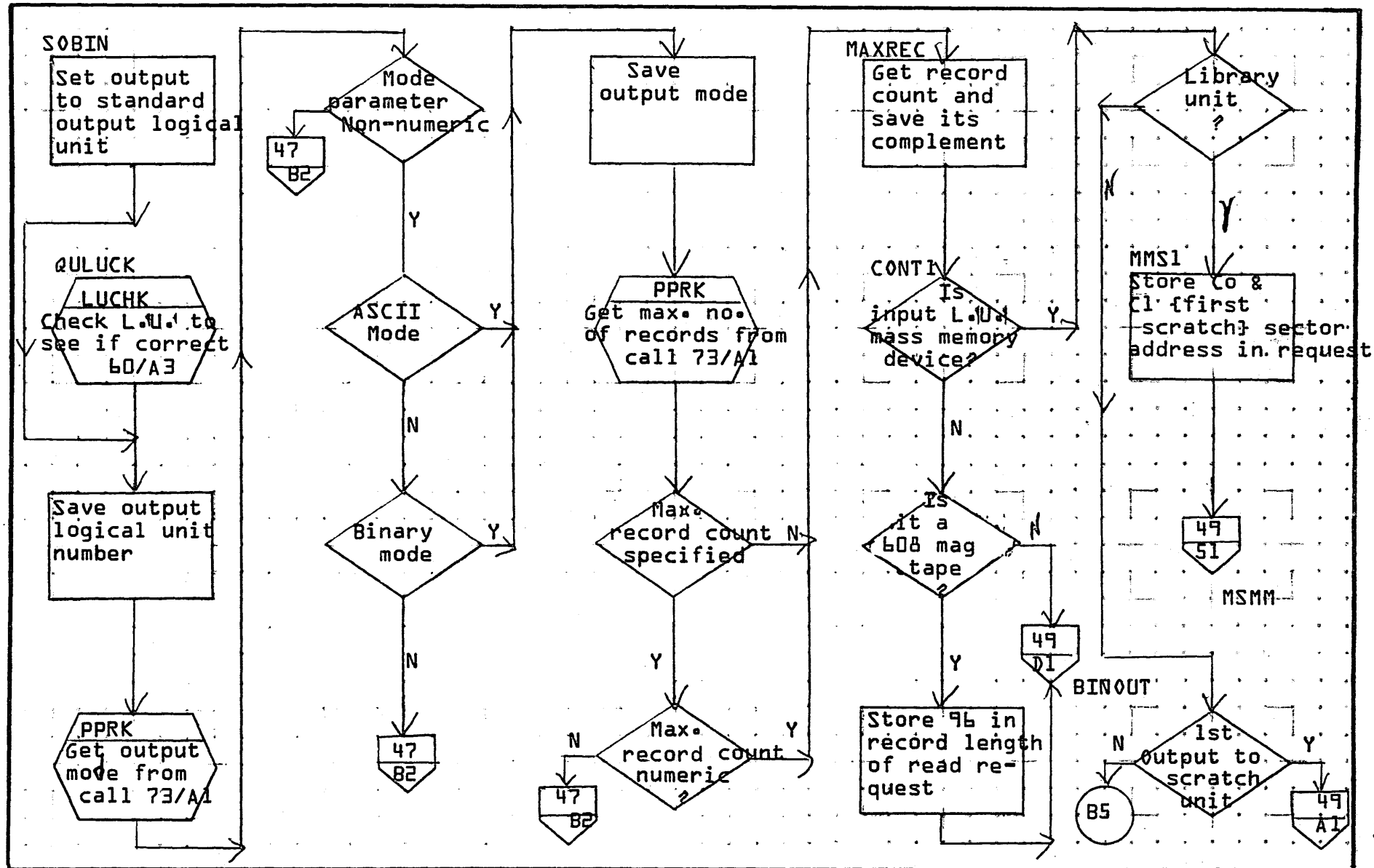
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 47 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.103

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



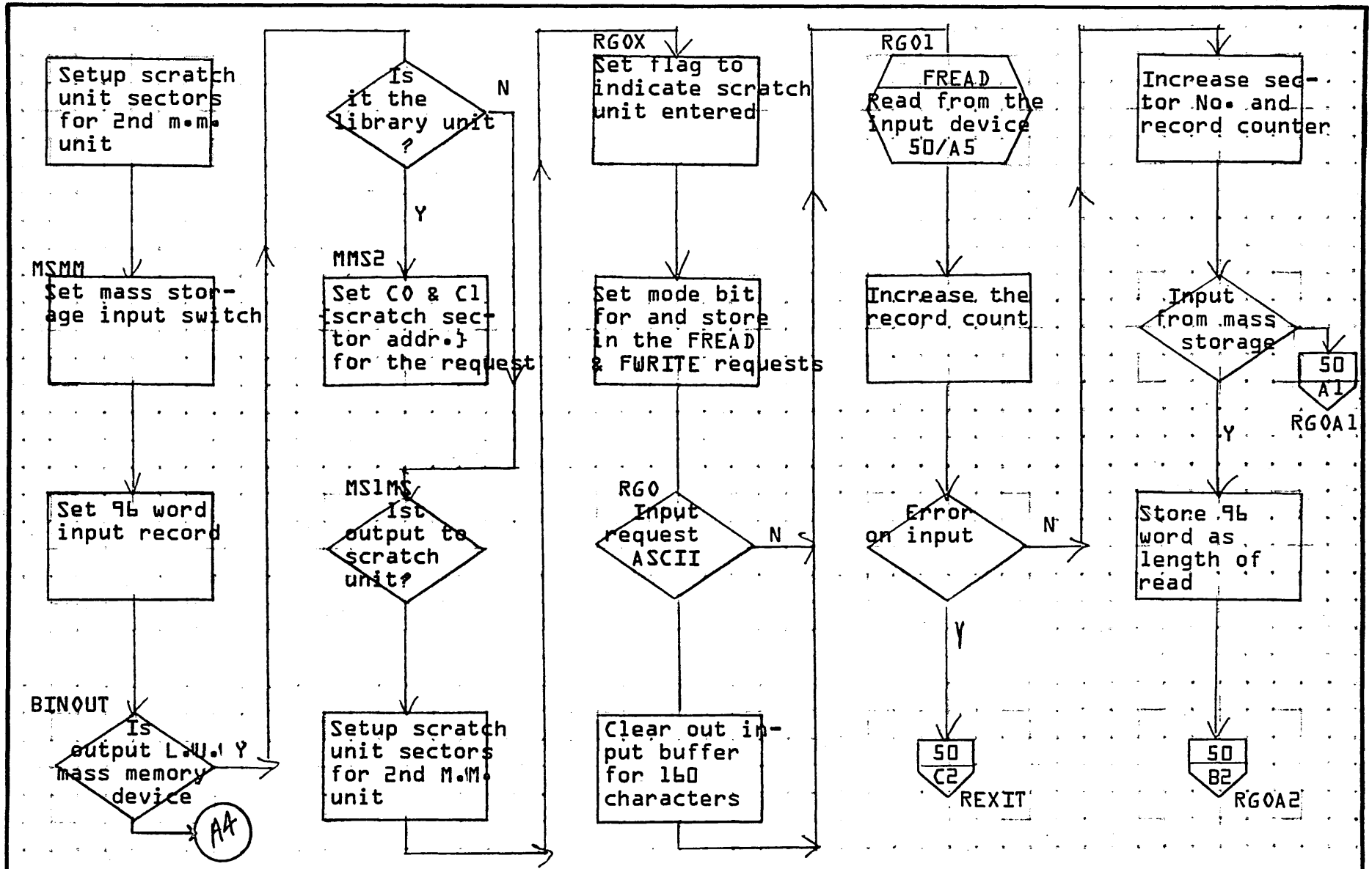
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 48 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.104

AA1385 (FORMERLY CA127-1)

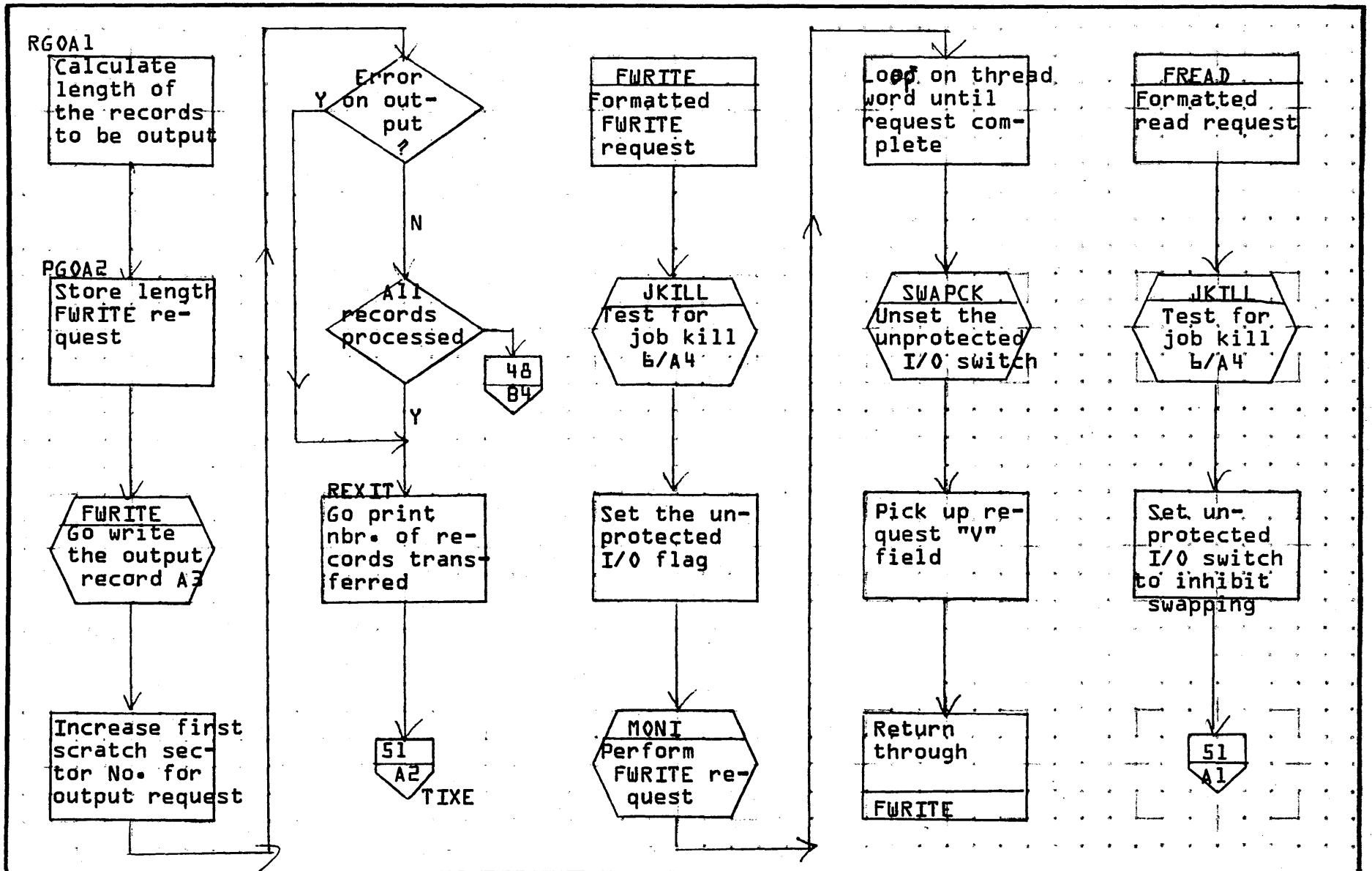
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 113 OF 177	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.105



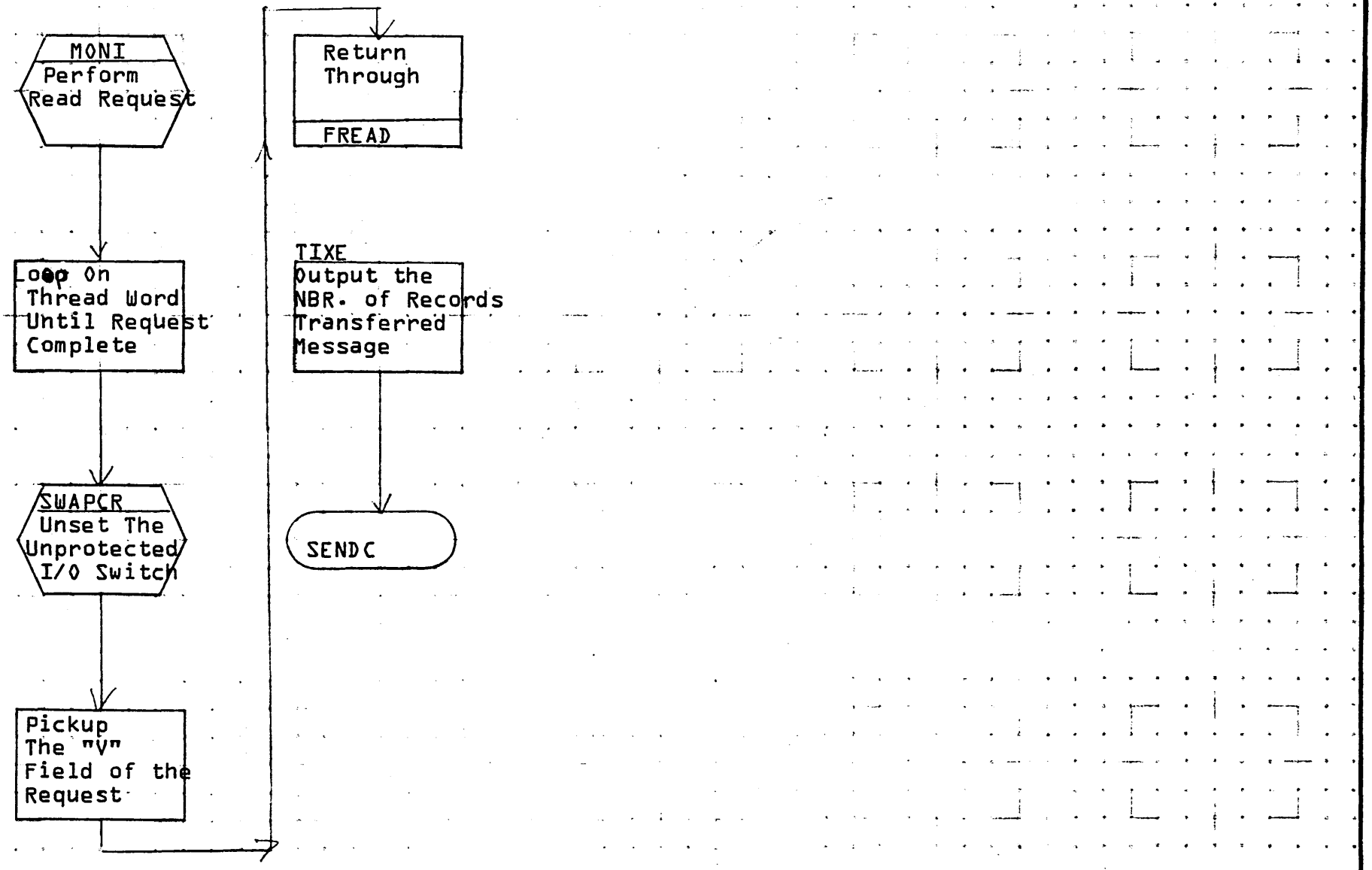
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LFBEDT PAGE 50 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.106

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

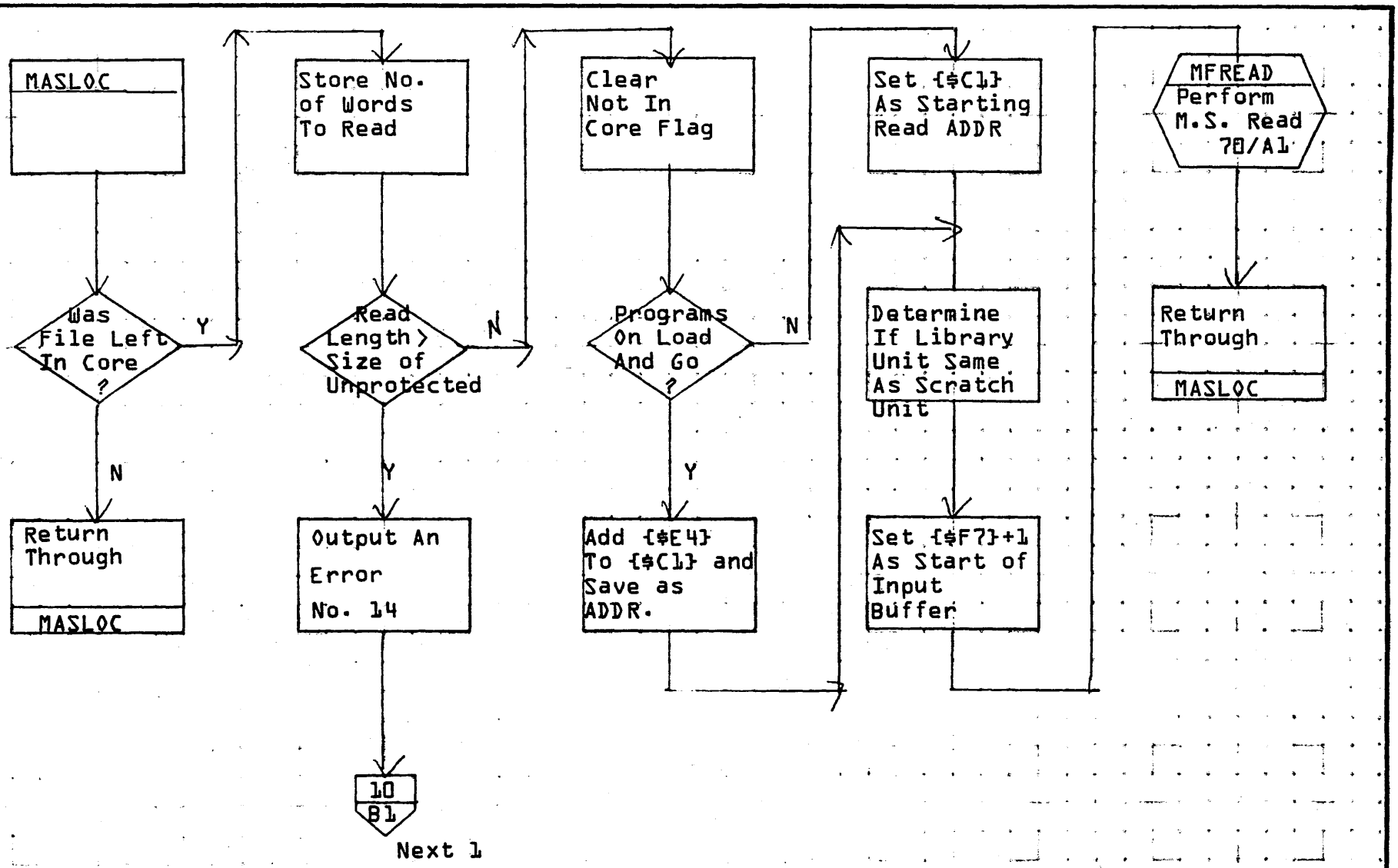


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	LIBGDT PAGE 51 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

60.102.00

AA1385 (FORMERLY CA127-1)



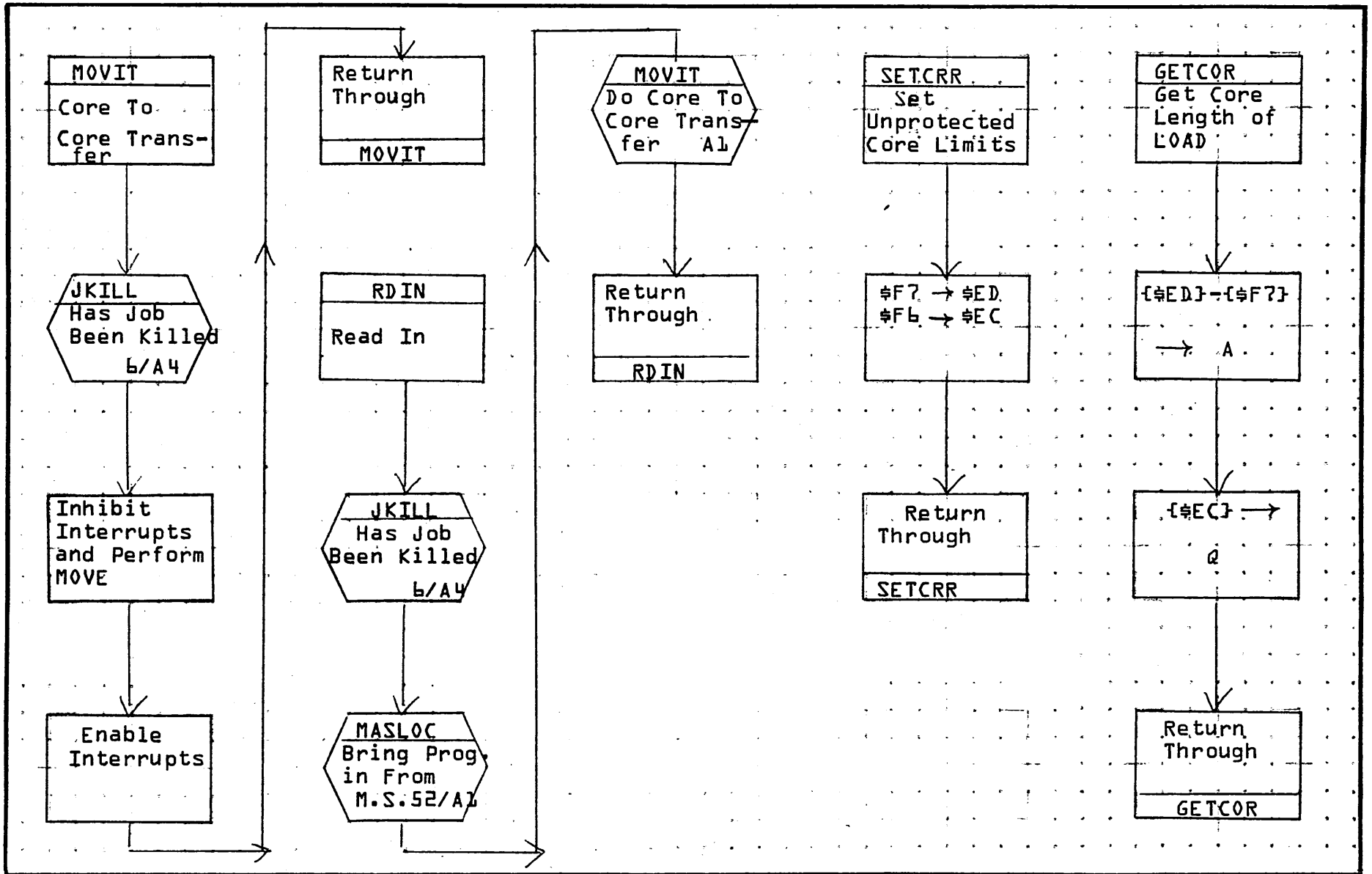
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 52 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

35.108



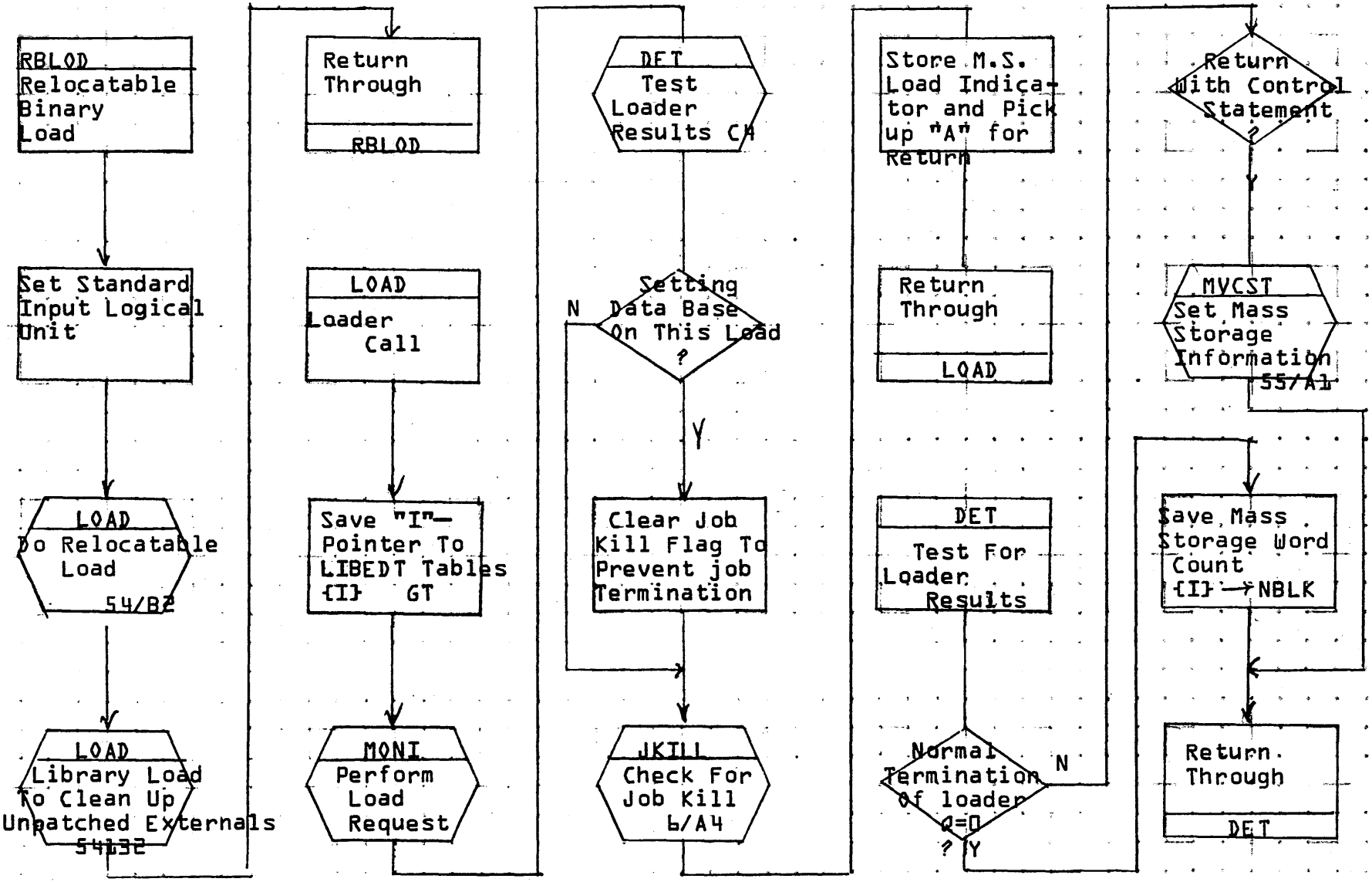
CONTROL DATA CORPORATION	DOCUMENT CLASS JMS	MACH. TYPE 1708	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE		PROJECT MGR.			
SAMPLE CODE ☐	LIBEDT PAGE 53 OF 77		PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

35.109



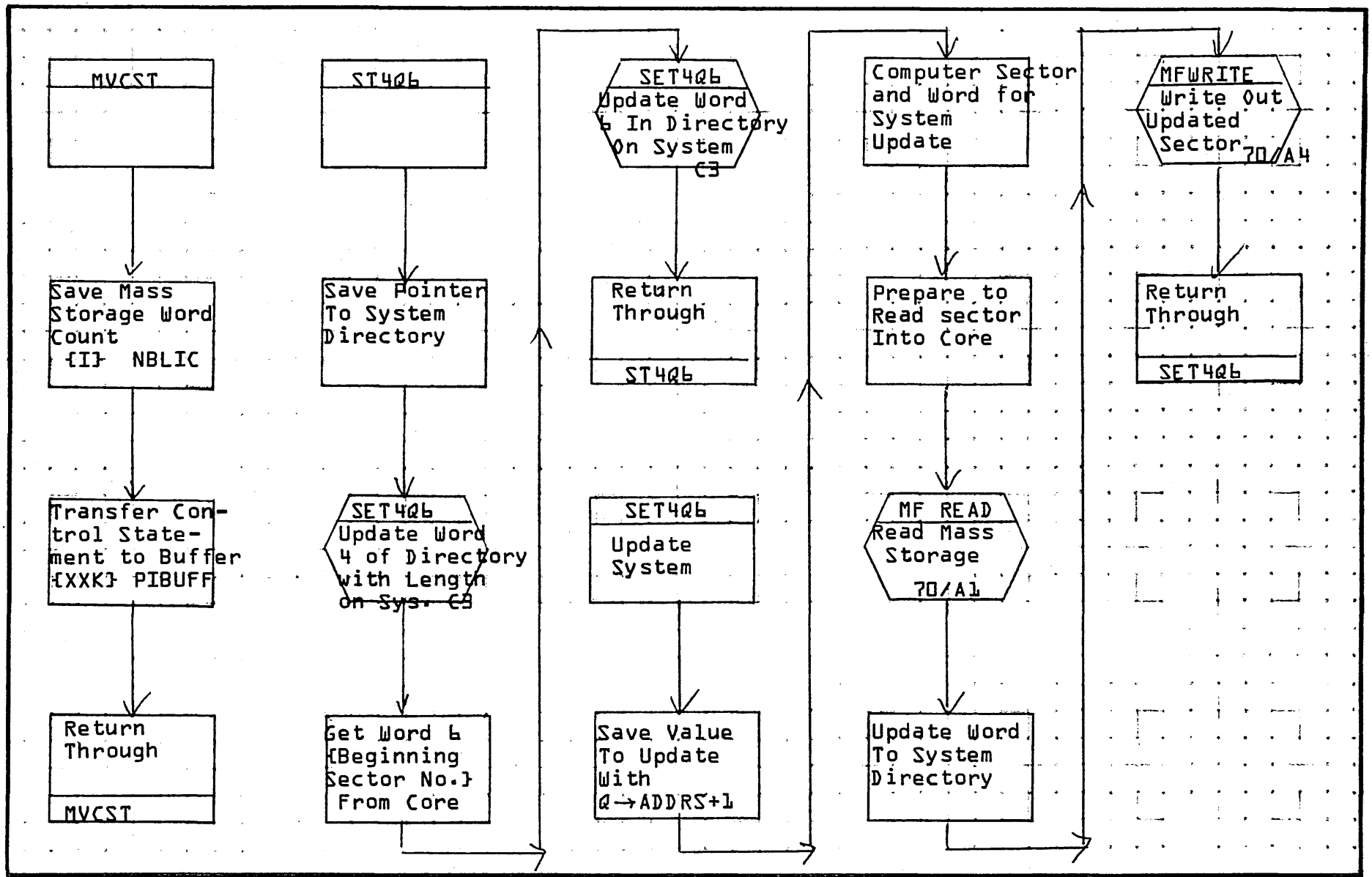
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS MACH. TYPE 1708	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBEDT PAGE 54 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.110

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

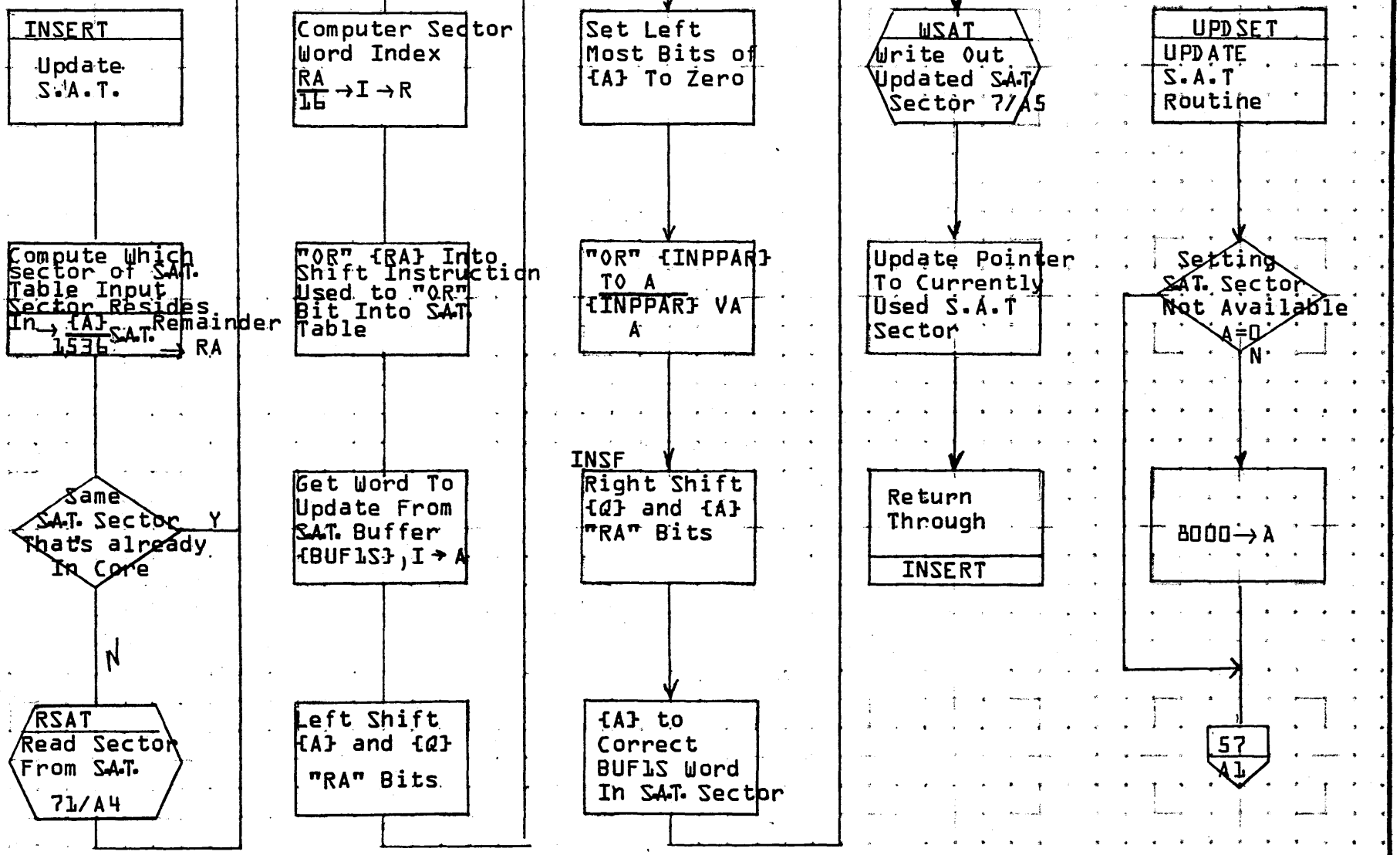


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LFBEDT PAGE 55 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.111

AA1385 (FORMERLY CA127-1)



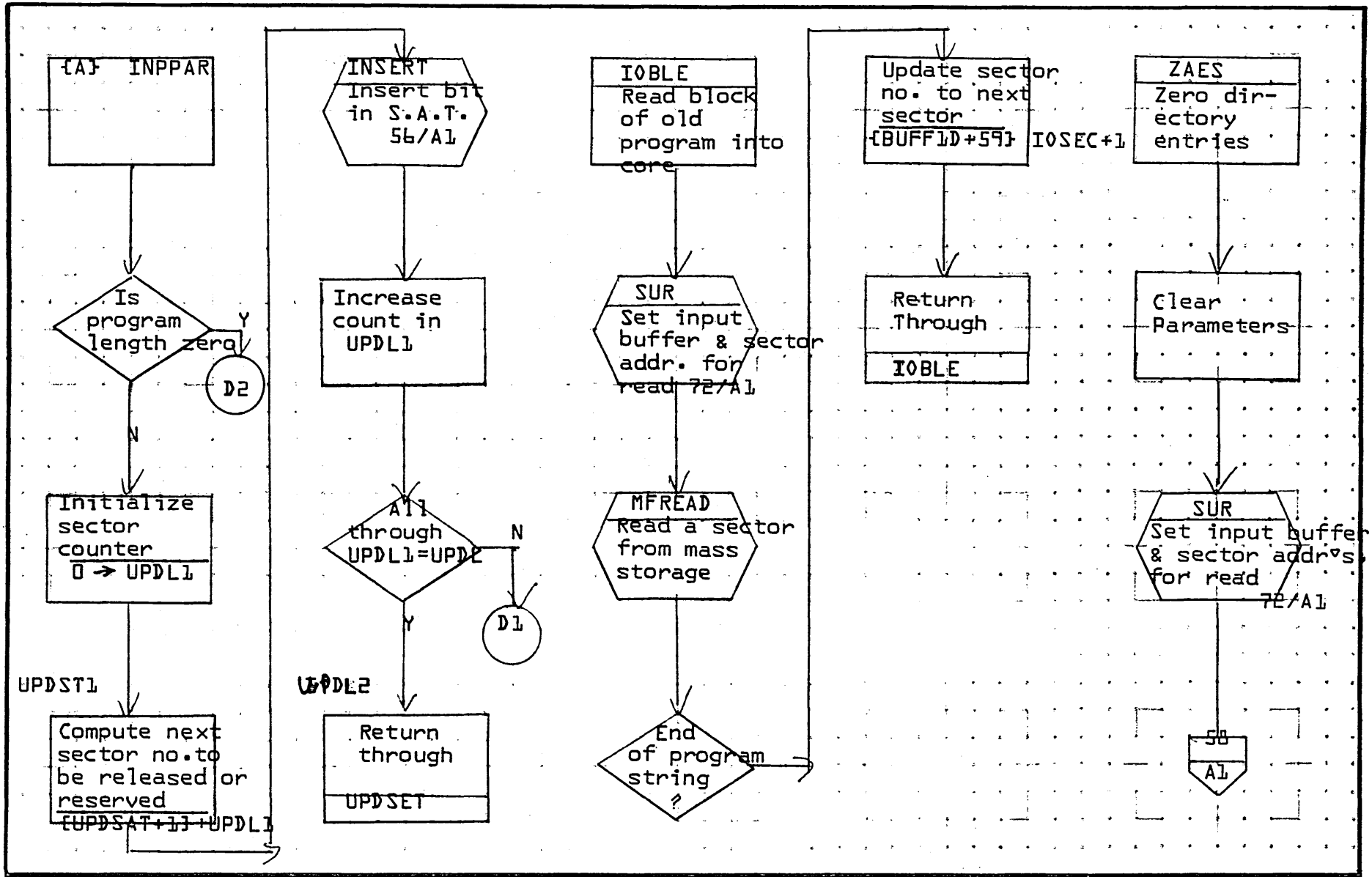
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 7600 DOCUMENT TITLE LIBEDT PAGE 56 OF 77 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

MAR 5 1971

60435112

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

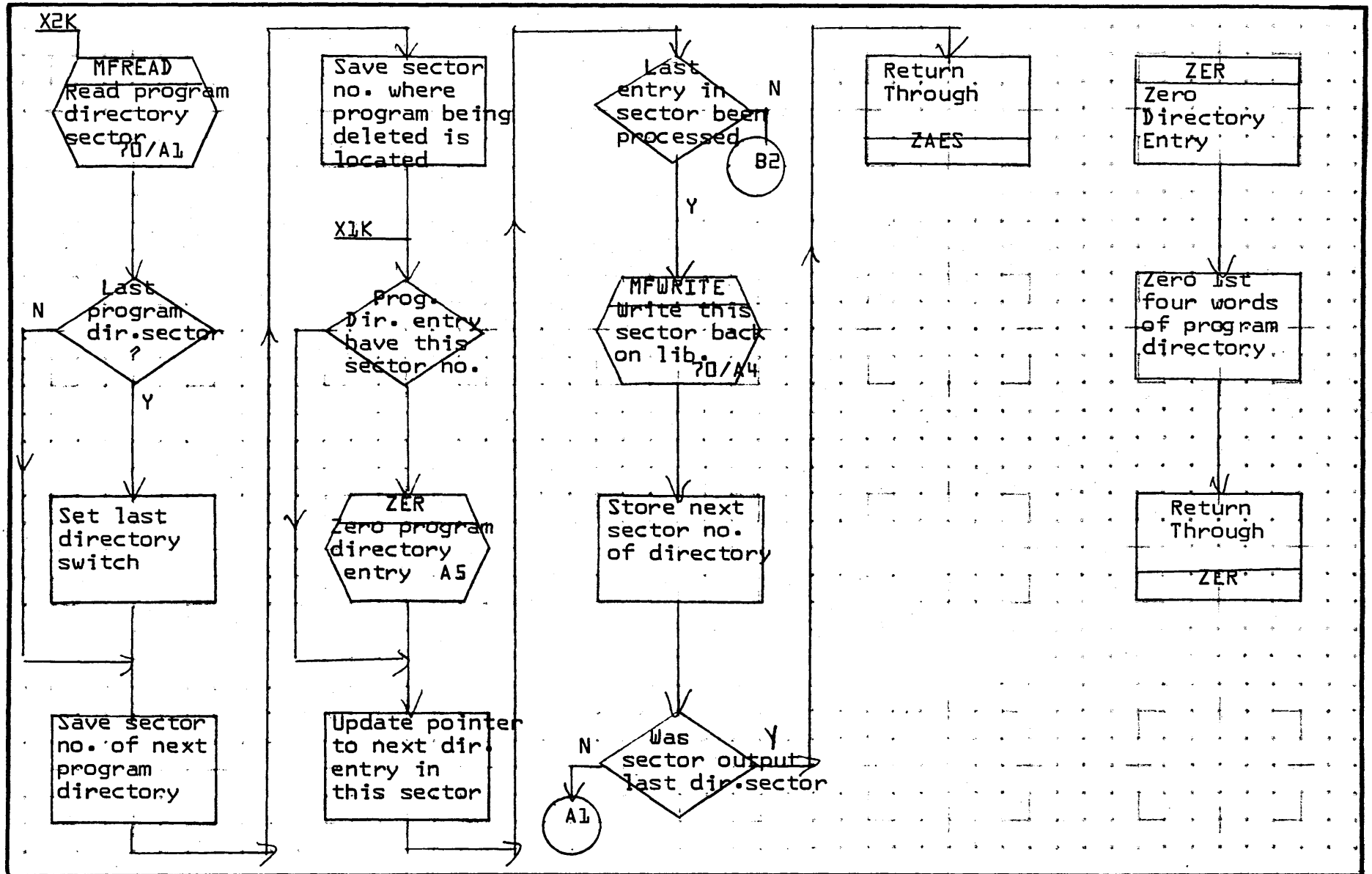


CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE		PROJECT MGR.			
SAMPLE CODE ☐	LIBEDT	PAGE 57 OF 77	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

AA1385 (FORMERLY CA127-1)

MAR 5 1971

35.113



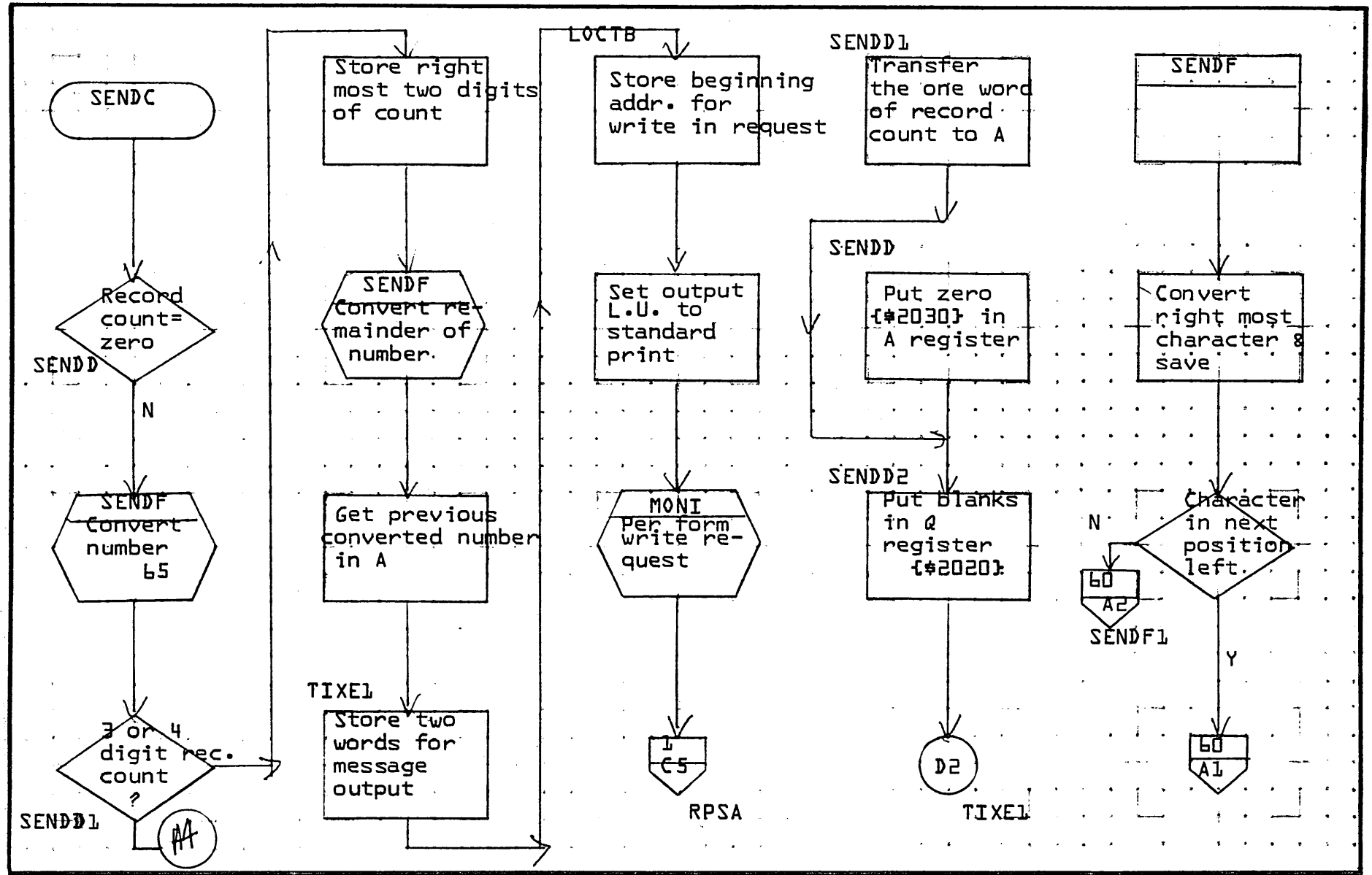
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 58 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.114

AA1385 (FORMERLY CA127-1)

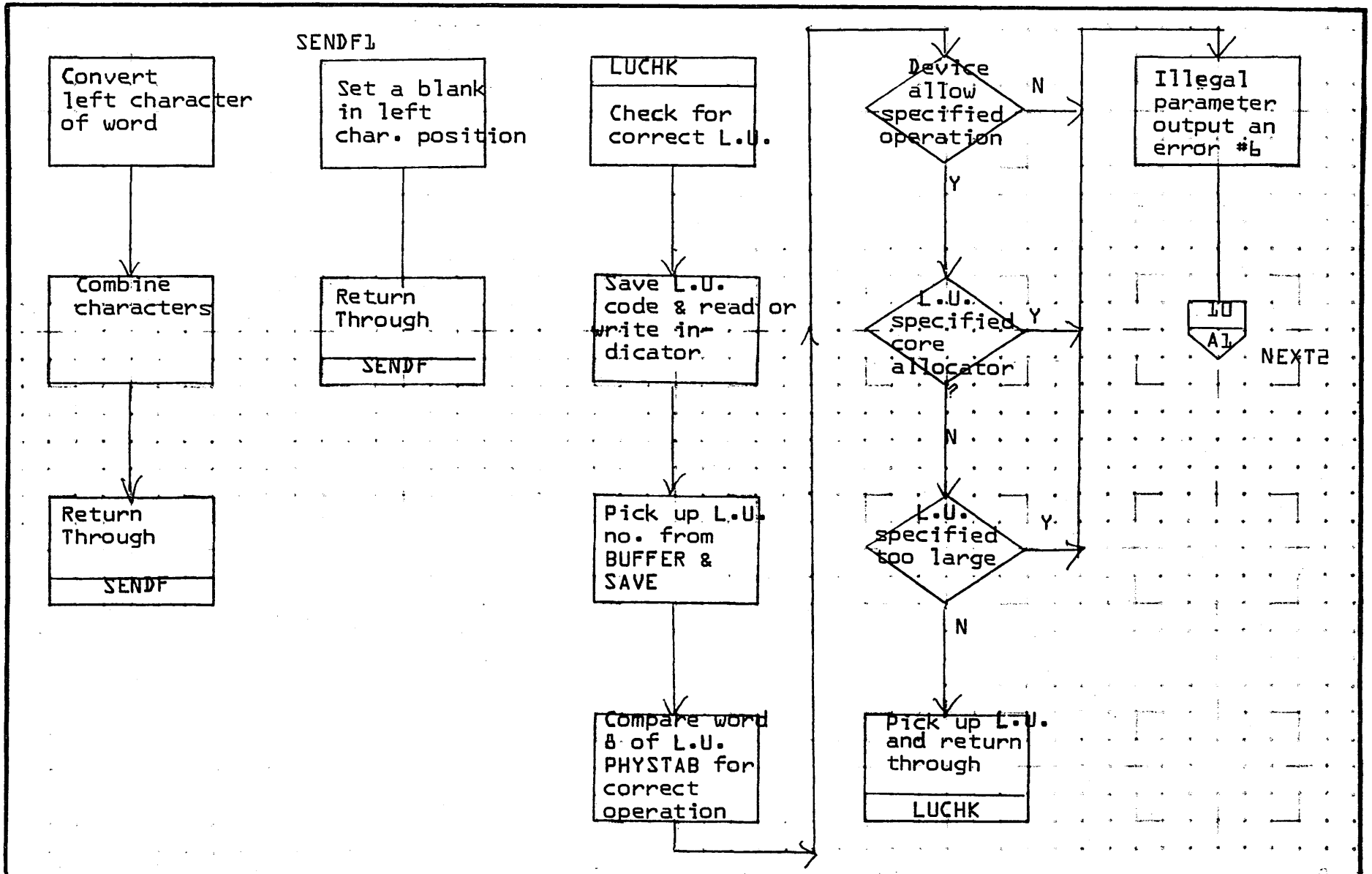
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1708	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LIBDT	PAGE 59 OF 79	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

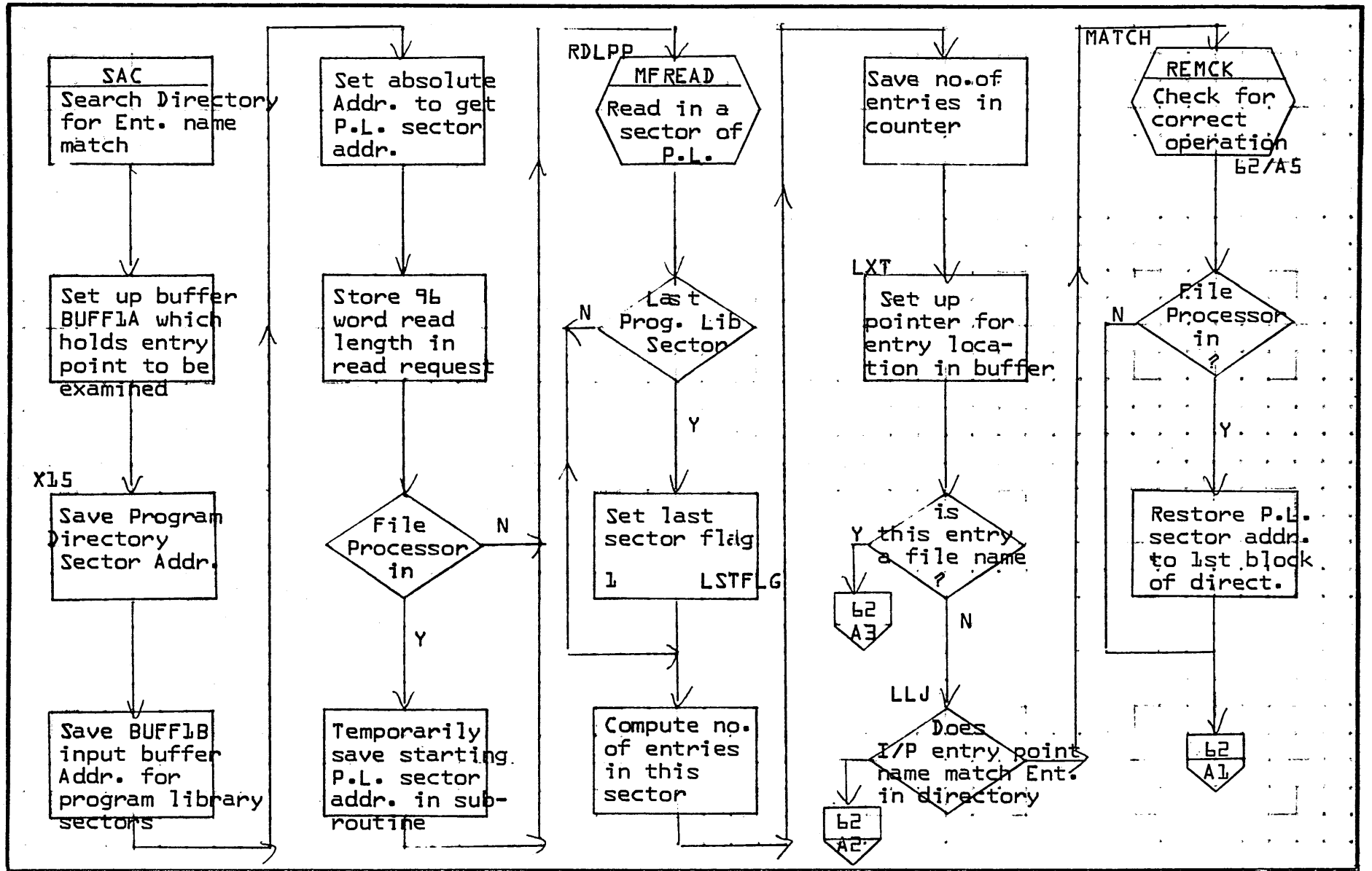
35.115



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐	LIBEDT	PAGE 10 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

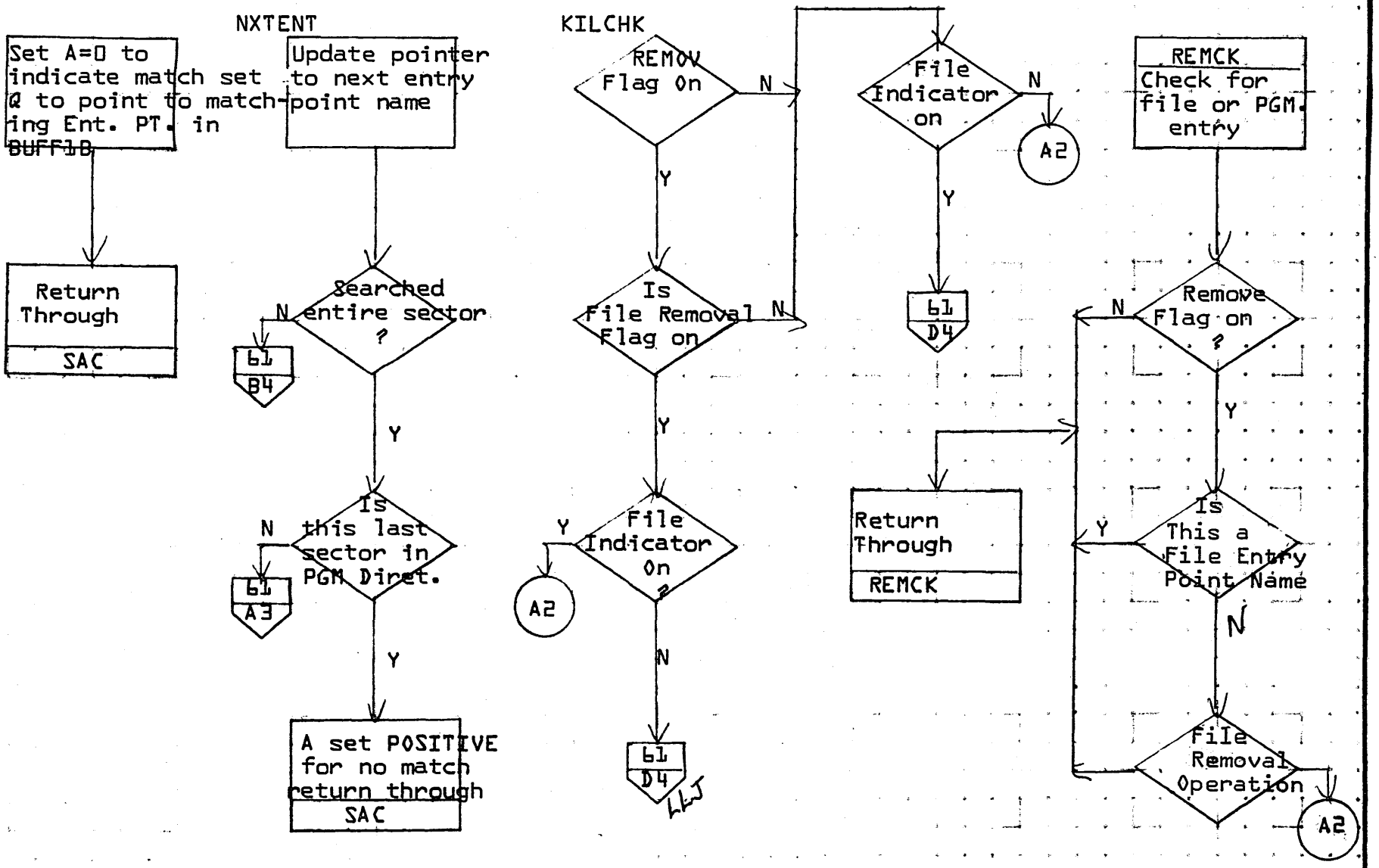
35.116



CONTROL DATA CORPORATION - SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LFBEDT PAGE 61 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

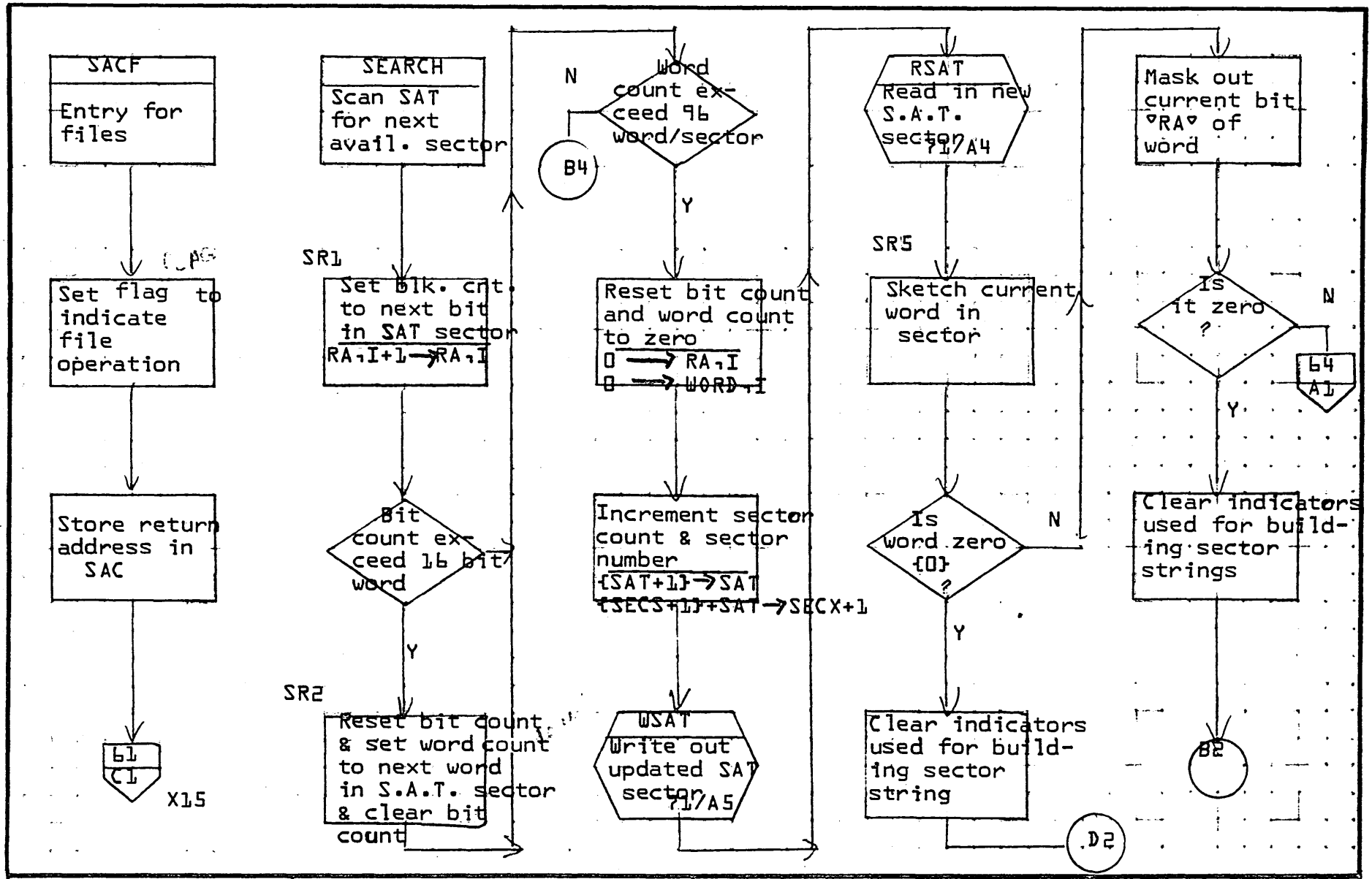
35.117



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1708	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐	LABEDT	PAGE 62 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

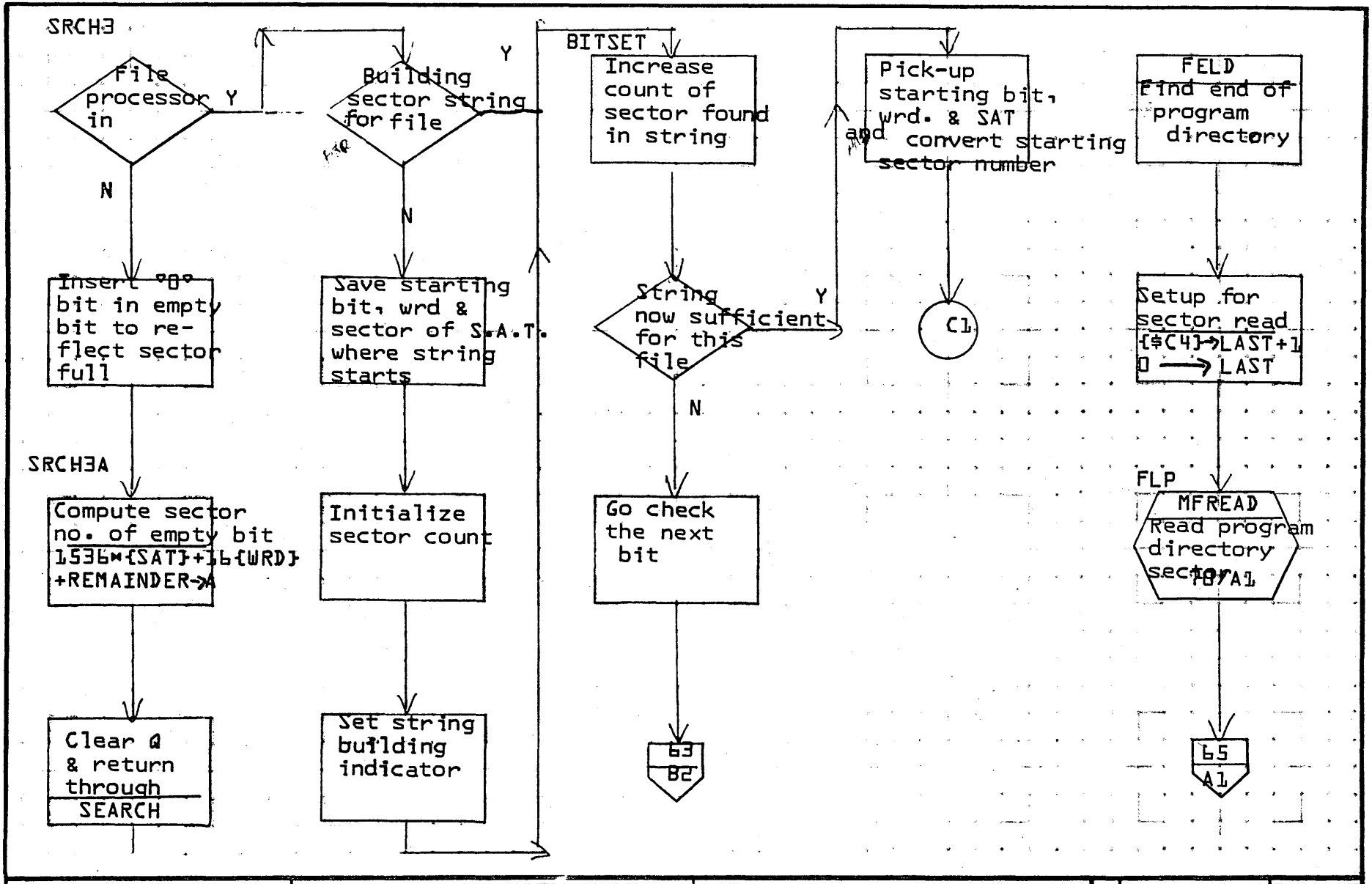
35.118



CONTROL DATA CORPORATION	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE		PROJECT MGR.			
SAMPLE CODE ☐	LFIBEDT	PAGE 63 OF 77	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

MAR 5 1971

35.119



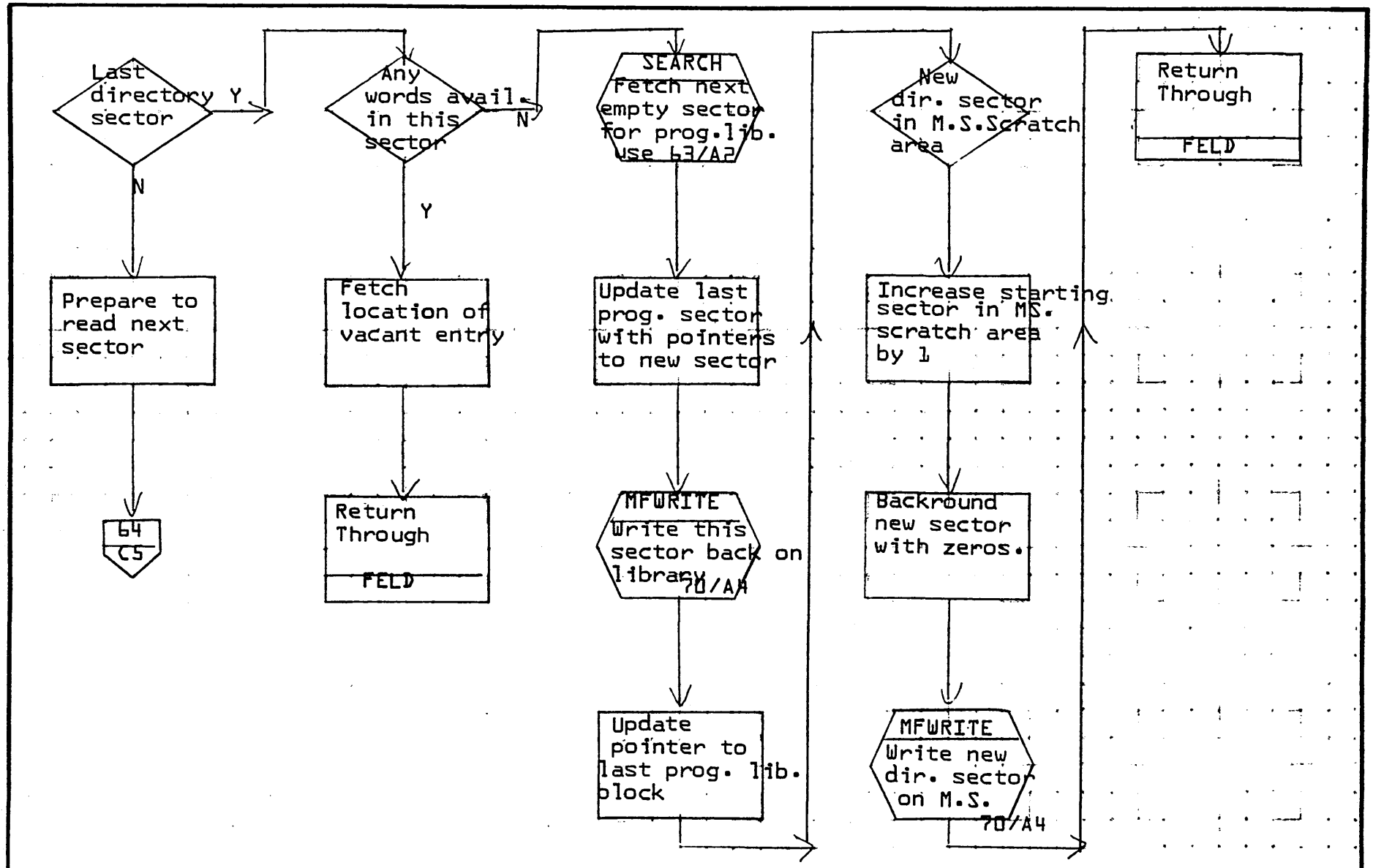
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS HMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LFBEDT PAGE 64 OF 77	PROJECT MGR.			
	NUMBER ISSUE DATE	PROJECT NAME			
	DRAWN BY DATE	TASK NO.			
		TASK NAME			

MAR 5 1971

35.120

AA1385 (FORMERLY CA127-1)

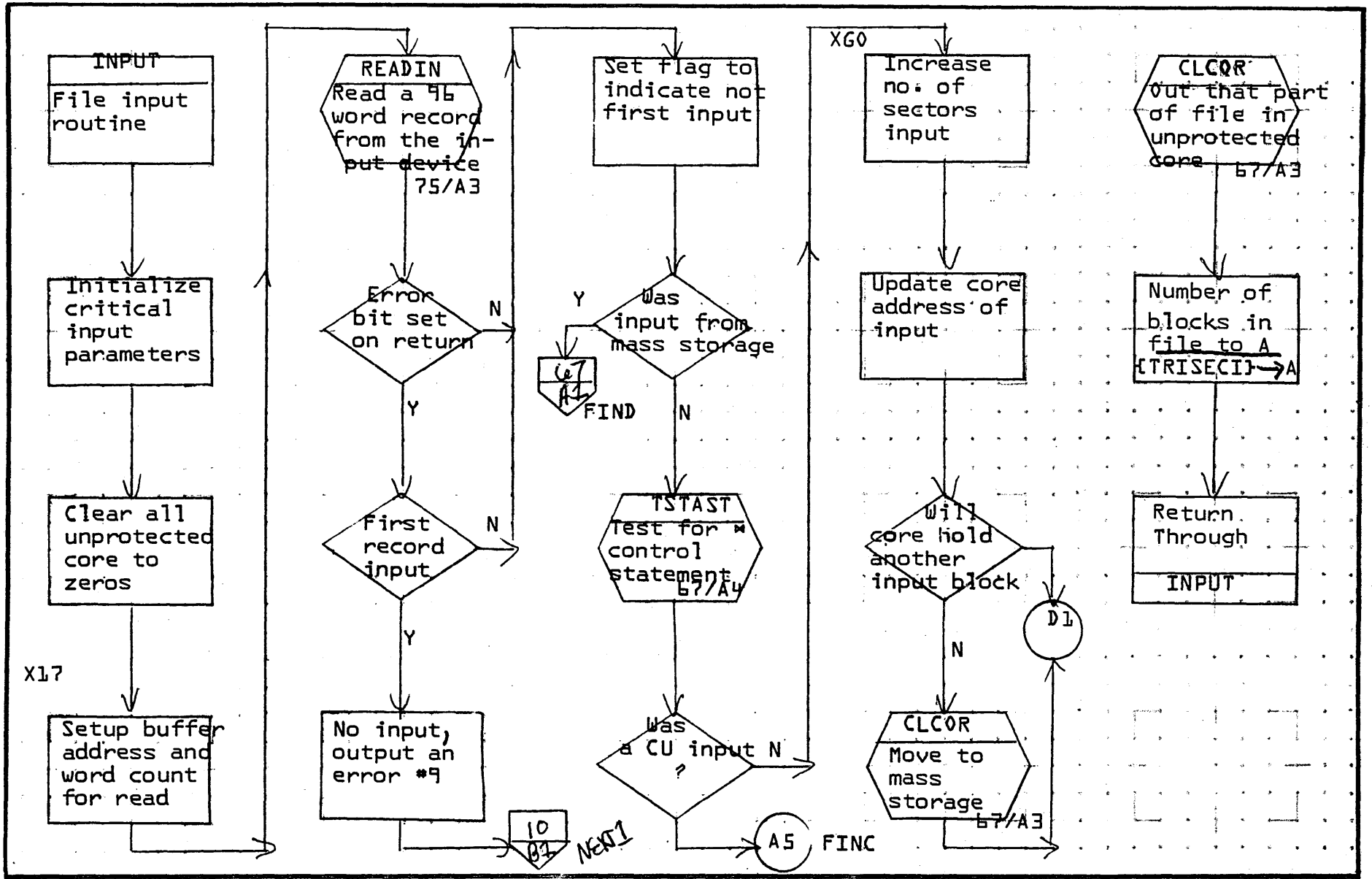
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 6 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 19/1

35.121



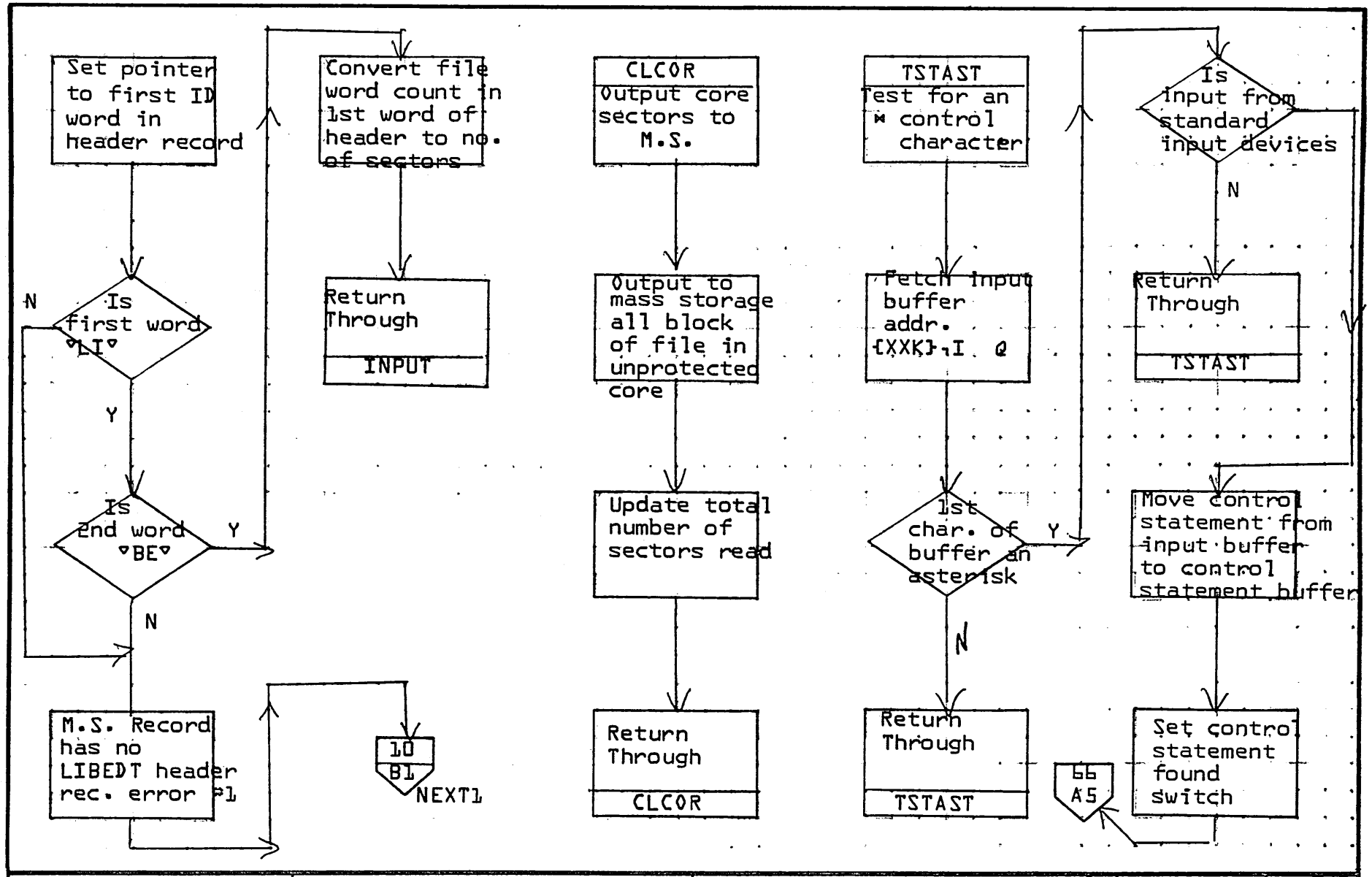
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LIBSDT PAGE 66 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.122

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



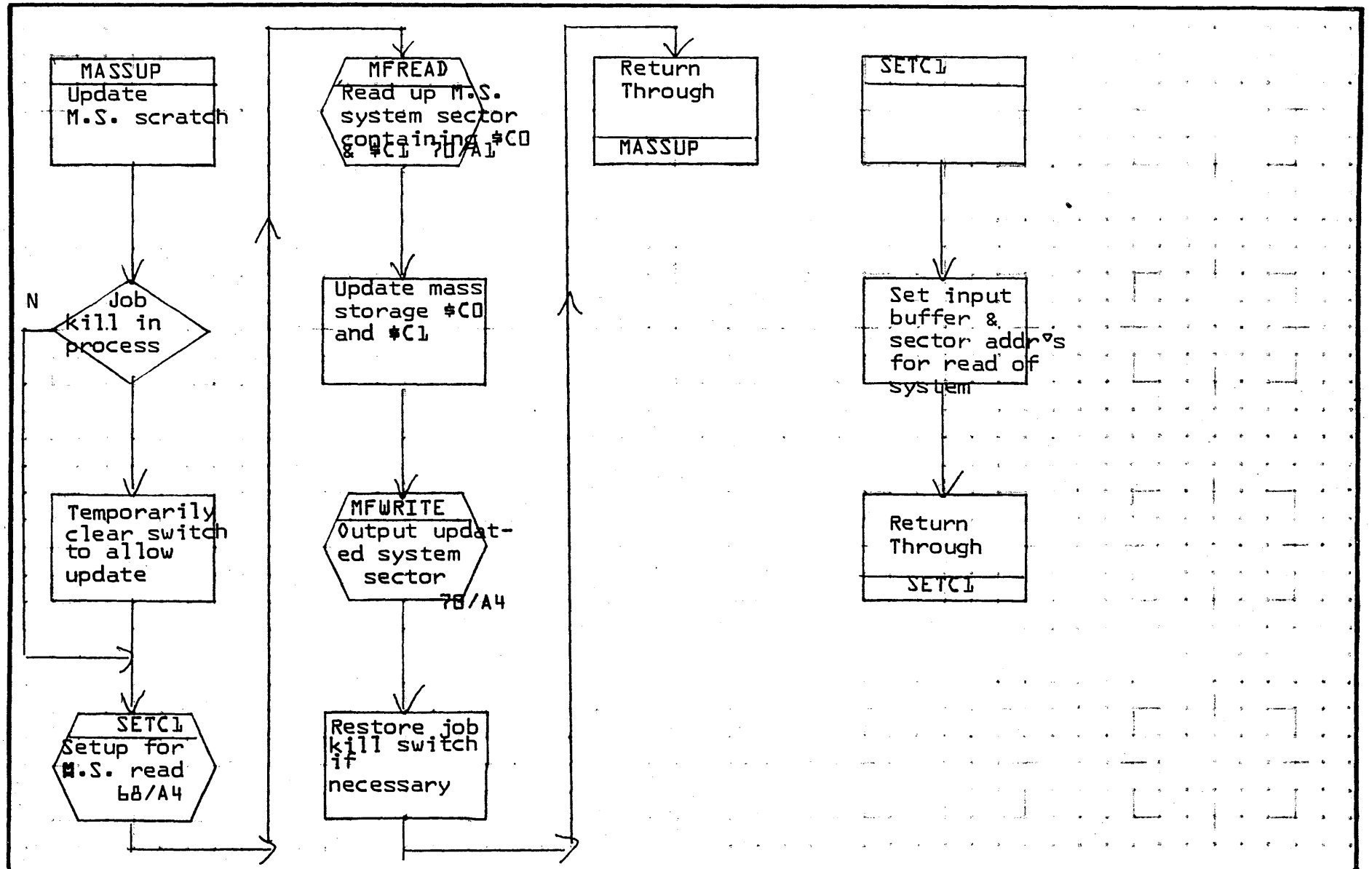
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1708	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐	LIBEDT	PAGE 67 OF 77	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

35.123

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



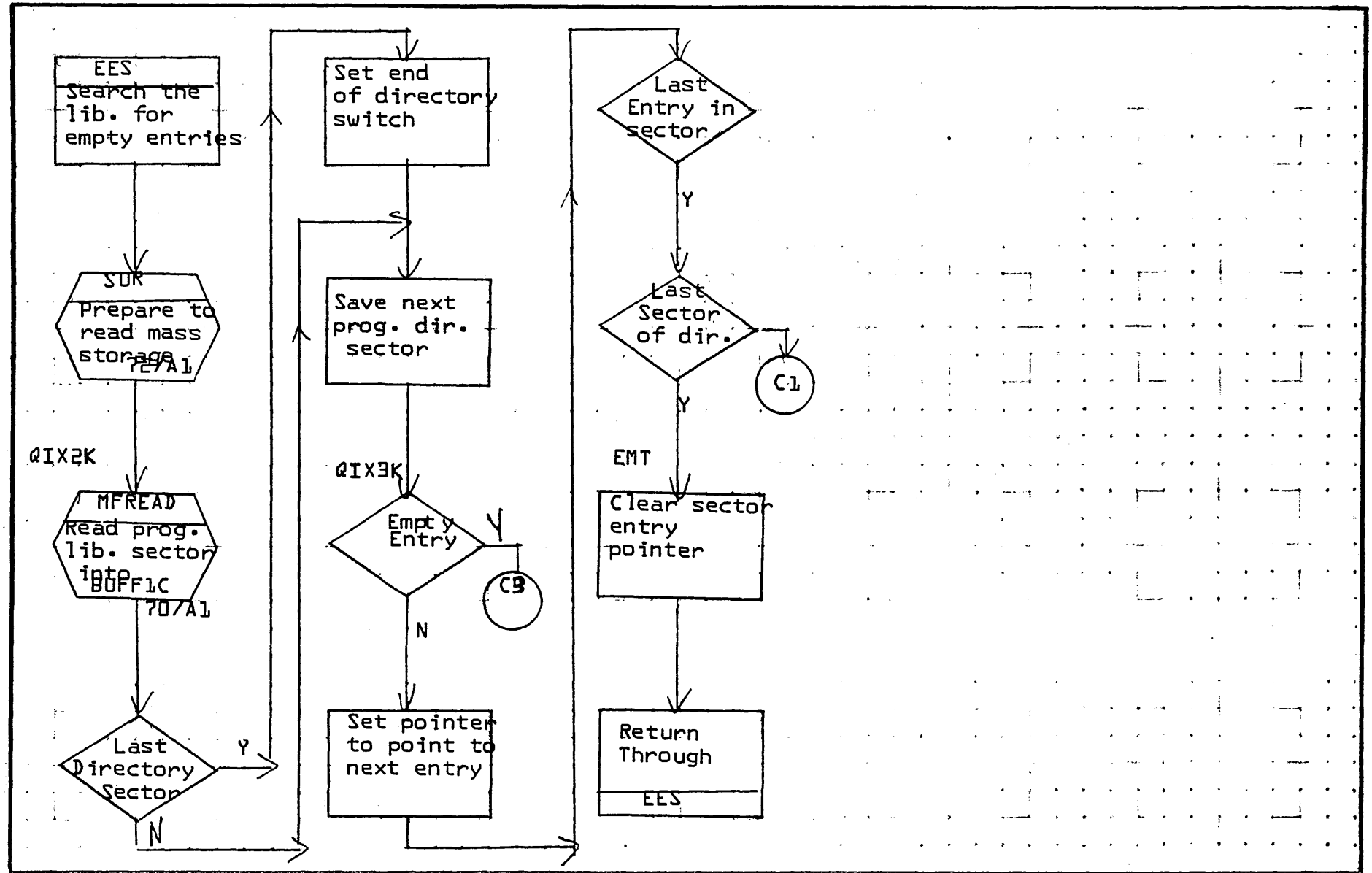
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LFBEDT PAGE 18 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.124

AA1385 (FORMERLY CA127-1)

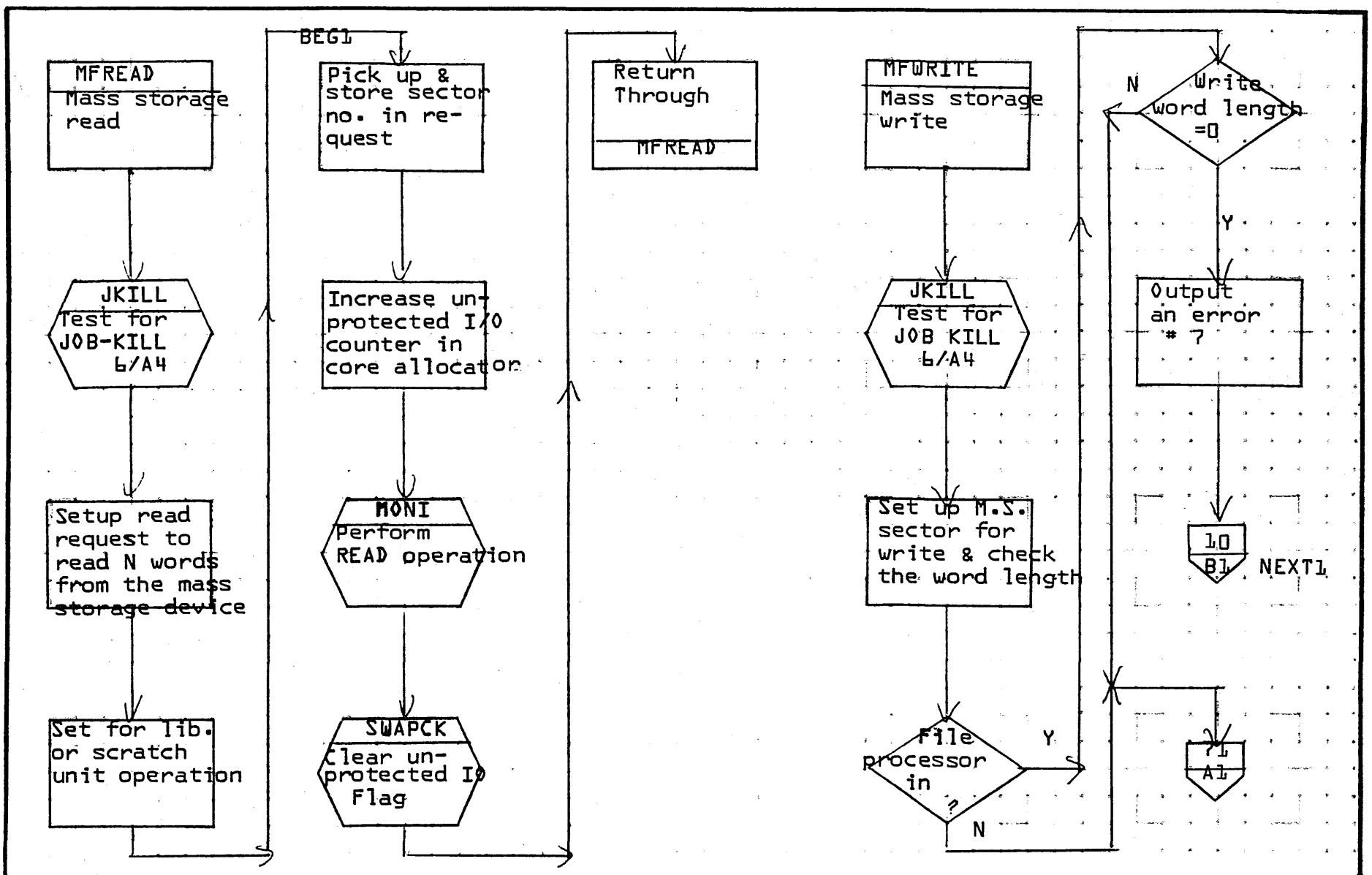
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LFIB2DT PAGE 69 OF 77	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.125



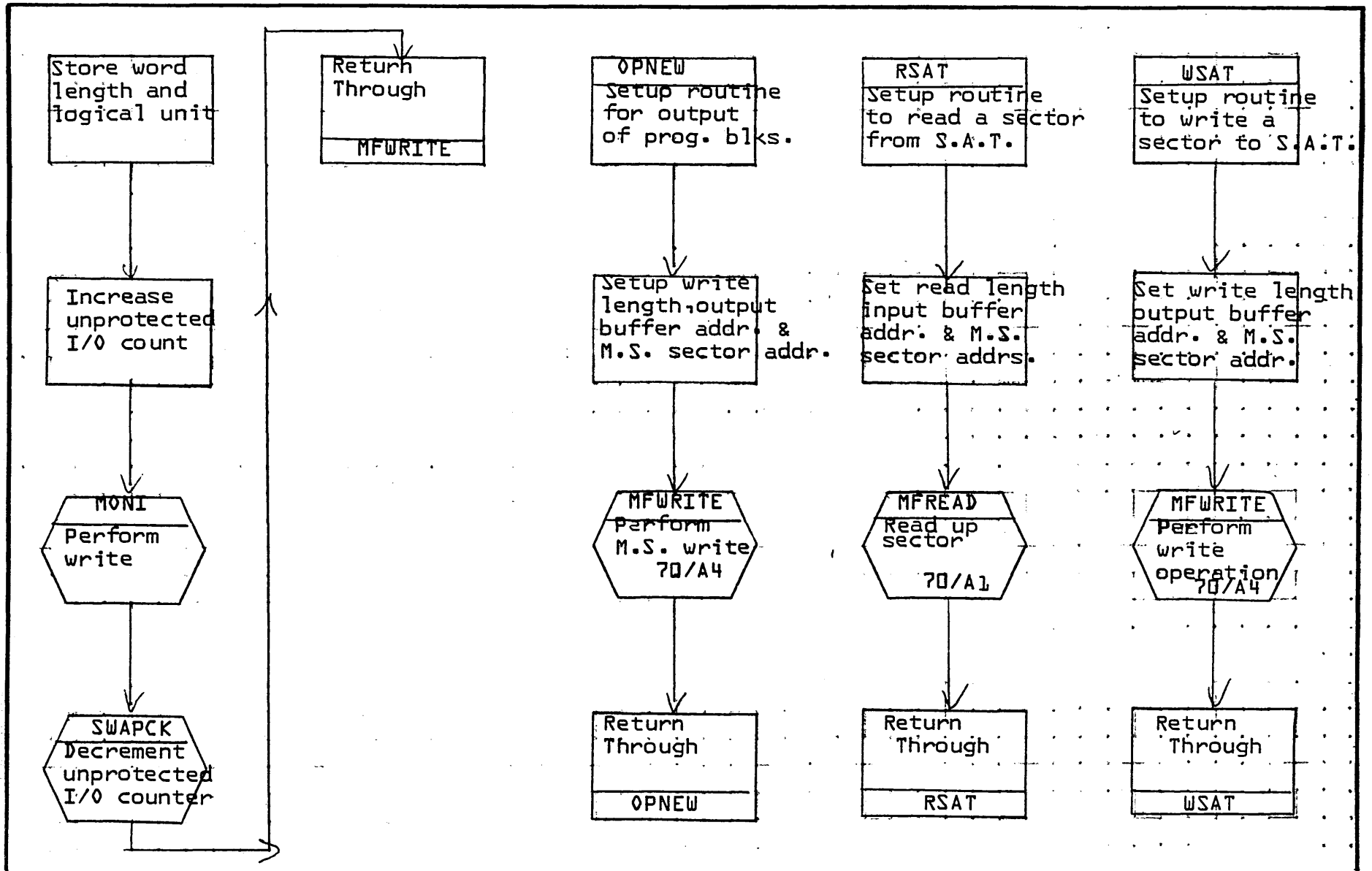
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEOT	PAGE 70 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.124

AA1385 (FORMERLY CA127-1)

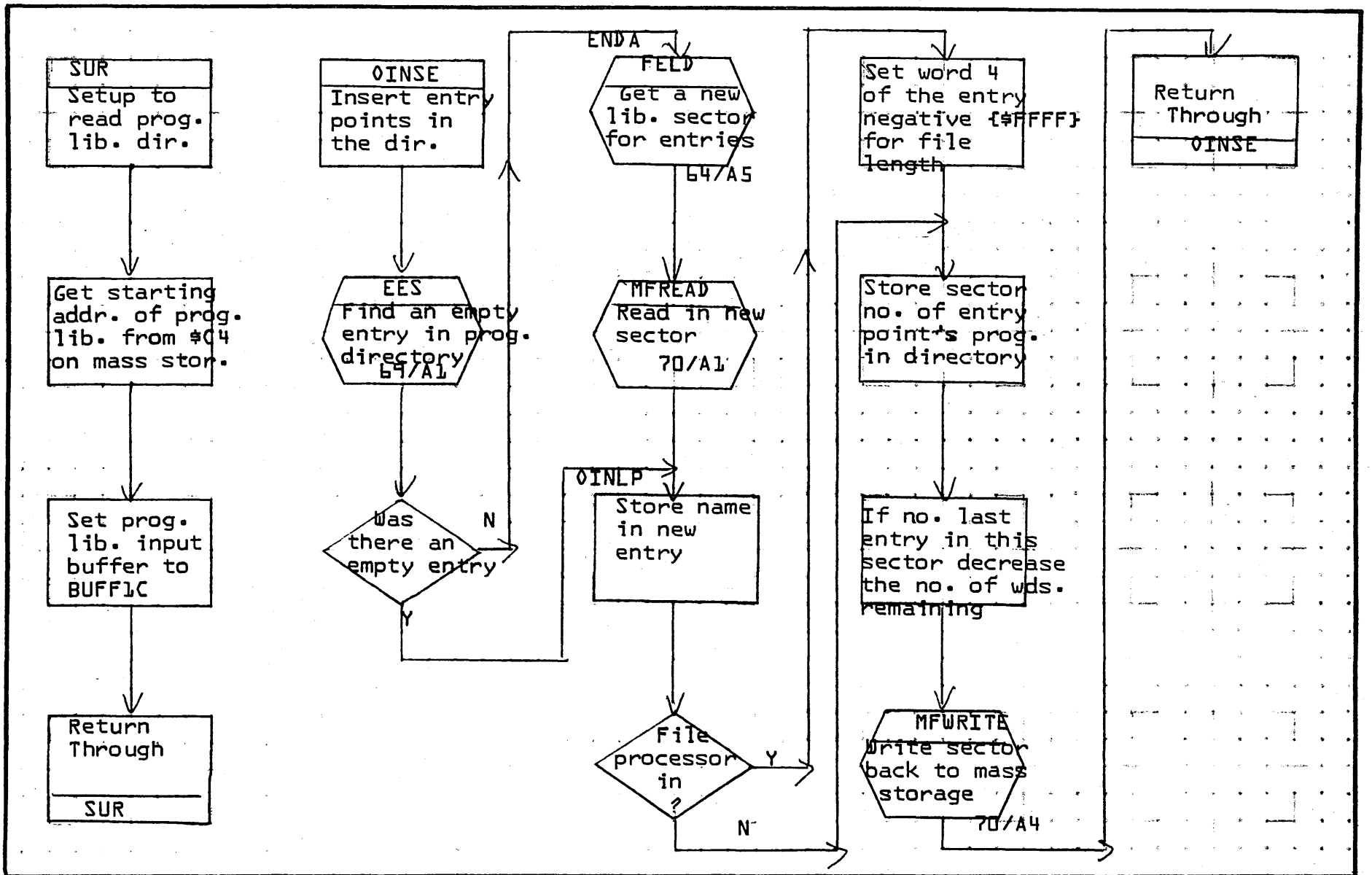
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 7700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LFIBEDT PAGE 71 OF 77	PROJECT MGR.			
		PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.127



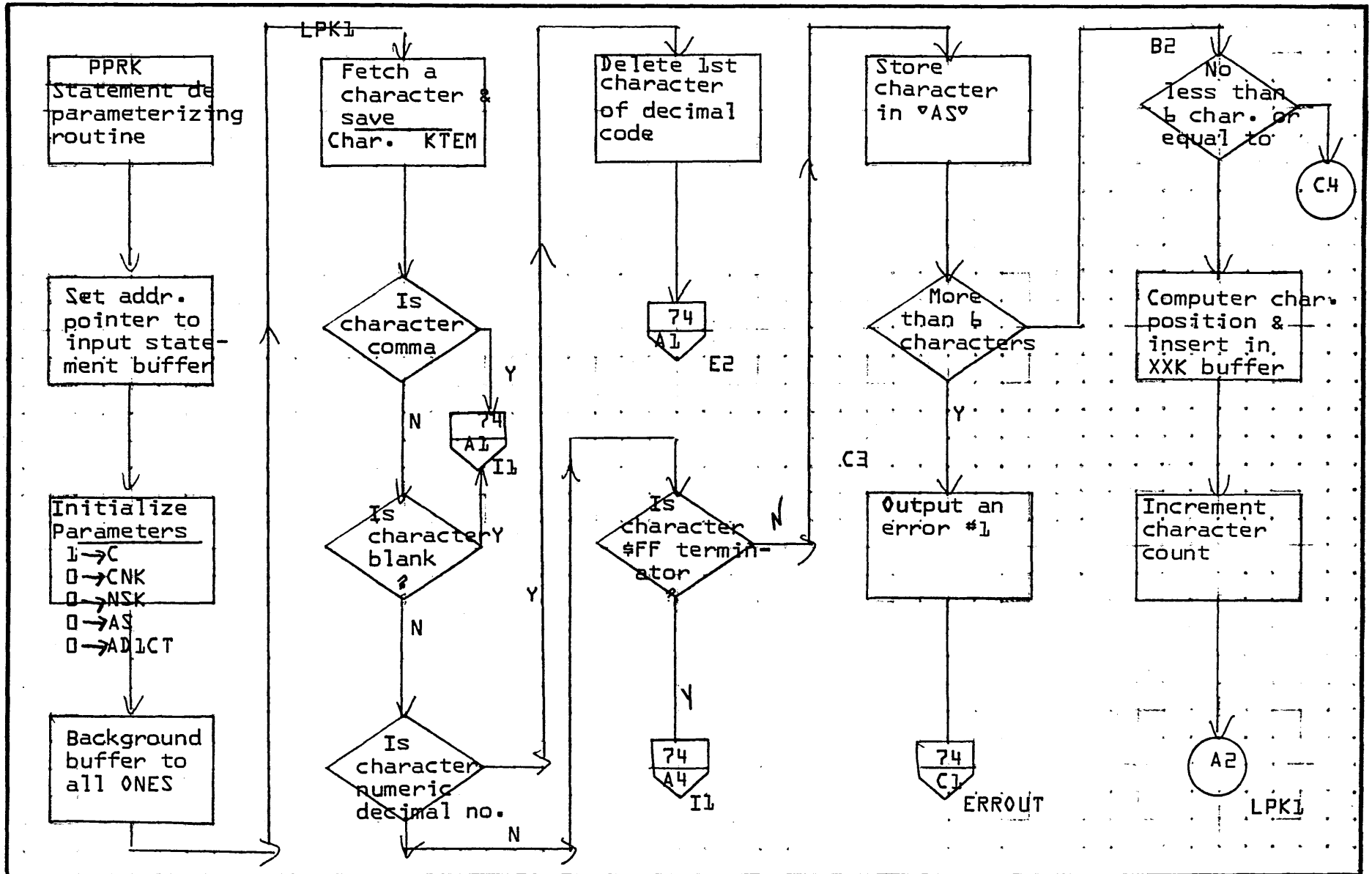
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 7600	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	LIBDT PAGE 72 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.128

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



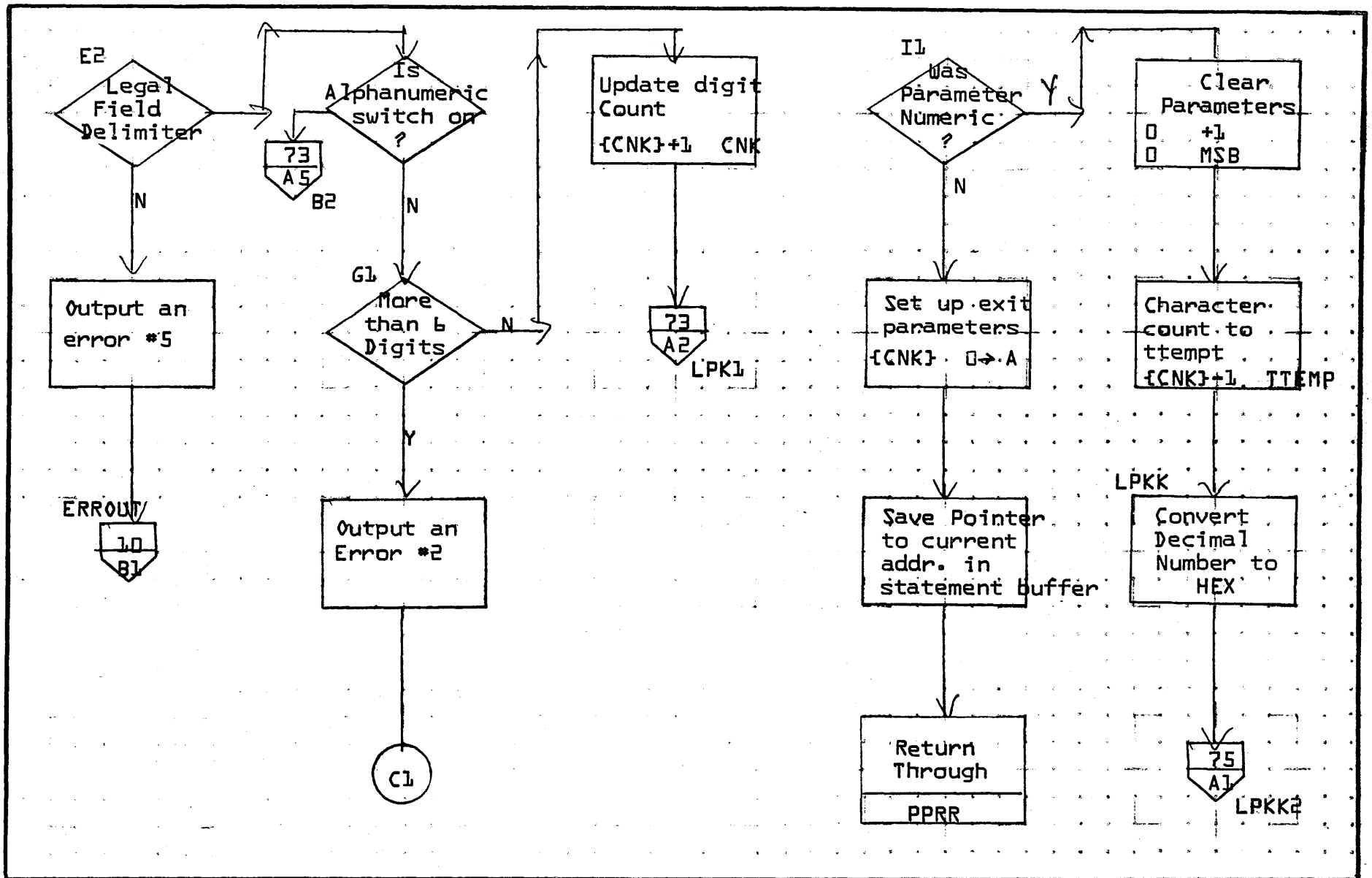
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 73 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.125

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



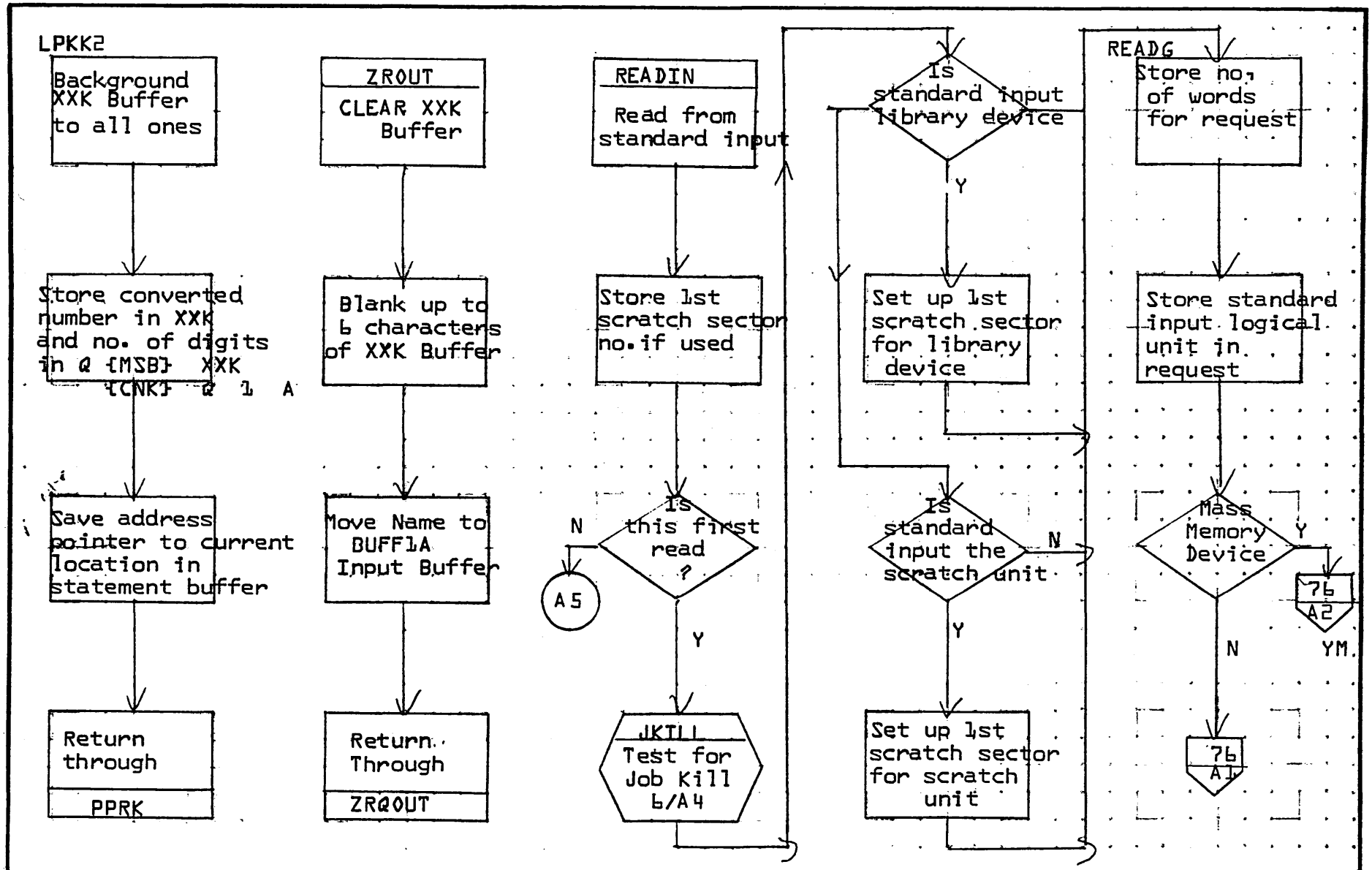
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 74 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.130

AA1385 (FORMERLY CA127-1)

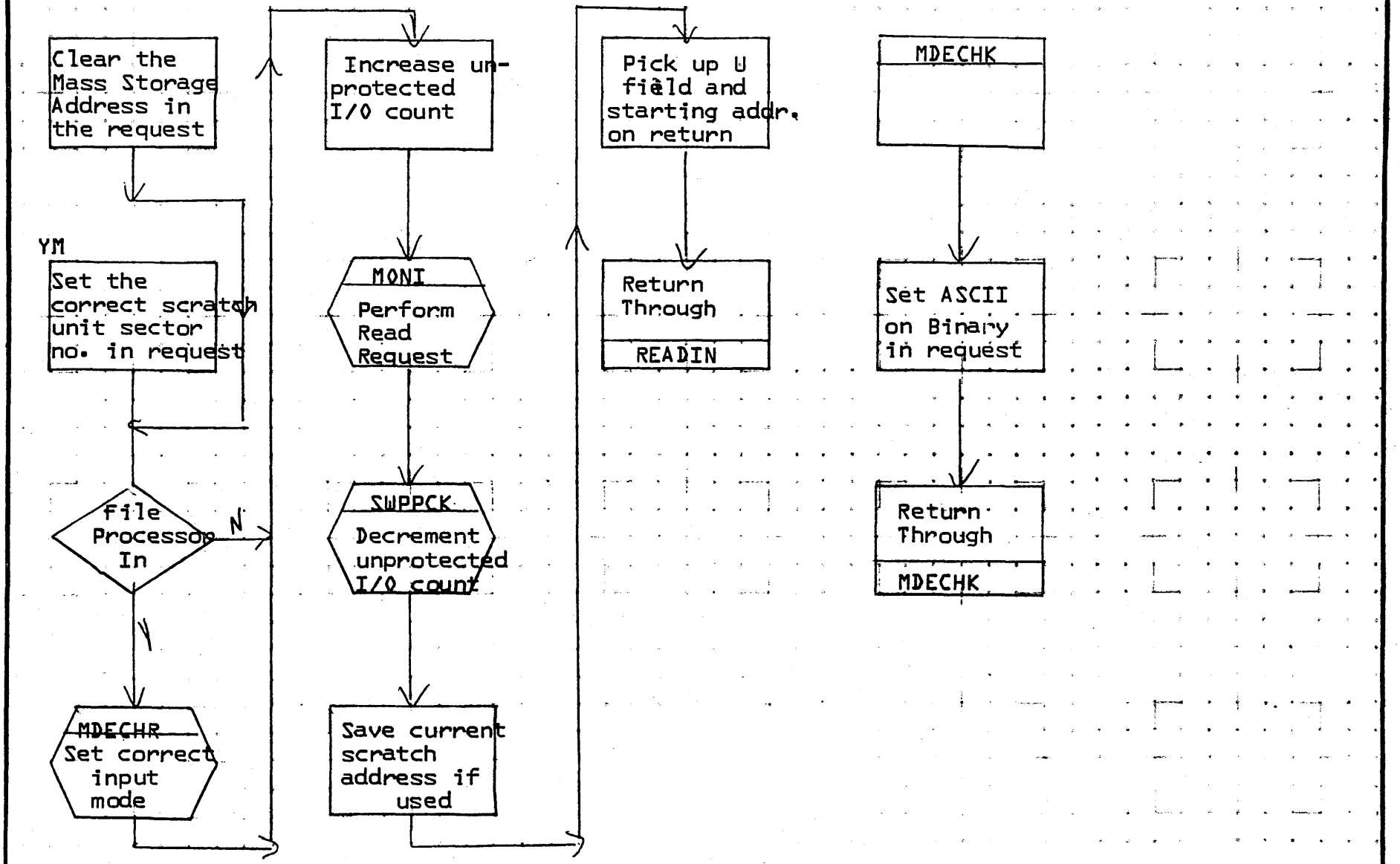
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 7000	PROJECT NO.	REV.	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	LAB2D1 PAGE 75 OF 99	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

35.13.1



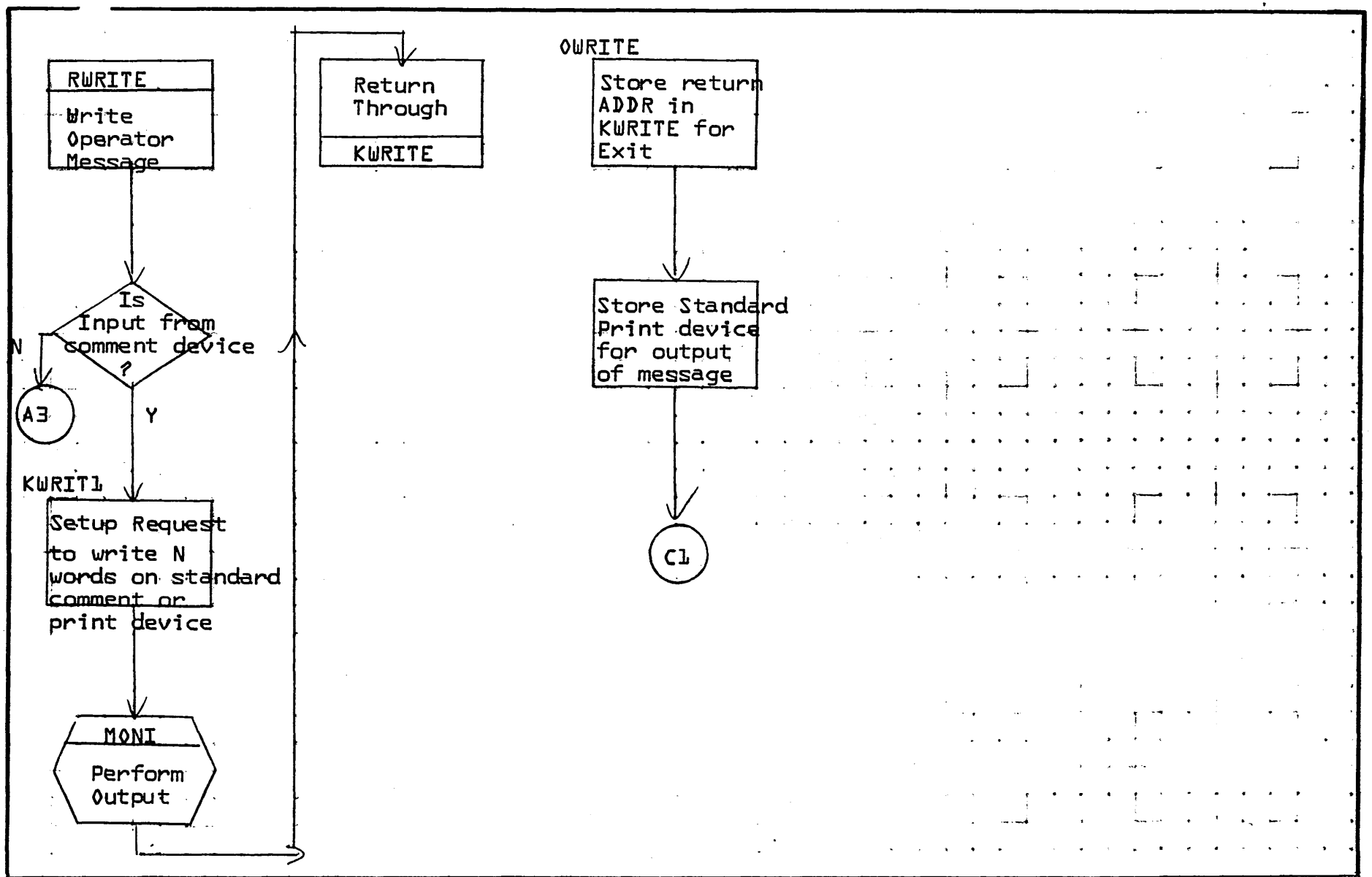
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LIBEDT	PAGE 76 OF 77	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

65.132

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS EMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐	LEIBERT PAGE 77 OF 77		PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

35.133

AA1385 (FORMERLY CA127-1

PRINTED IN U.S.A.

36.0 RECOVERY PROGRAM

36.1 PROGRAM FUNCTION

By using the recovery program, the programmer may determine the state of core and mass storage at the end of job execution. It is operated under the direction of the job processor. Whenever the job processor detects an *SR statement prior to job execution, it sets the recovery indicator switch. The recovery program is operated at job termination when the switch is set.

36.2 RECOVR ROUTINE

36.2.1 Internal Description

Upon entering the recovery program, the comment "RE" is typed on the comment medium. The operator is then free to direct the operation of the recovery program by entering control statements.

All control statements must be entered from the comment medium. The output device may be selected. Control statements to the recovery program consist of statements in format records. The information is placed in a buffer internal to recovery program and is recorded in the internal code of the computer (ASCII).

The first character of a control statement must be an (*), and the last character must be a carriage return. The intervening characters of the statement identify the type of statement and consequently, the prescribed action desired.

Internal to the recovery program is a list of acceptable control statements. A control statement read by the recovery program must match some entry in this list. With each entry in this list there is associated an address to which the recovery program will transfer in order to carry out the operation directed by a control statement. The standard list of input statements for the recovery program consists of the following:

INPUT STATEMENT	TRANSFER ADDRESS TO BEGIN ACTION
*Dhhhh, HHHH	DMPCOR
*MS1, W1, S 2, W2, N	MASDMP
*T	TERMIN
*Unit Number	OUTSEL

Unacceptable control statements are indicated by a printout (ERR) and the recovery program waits for another control statement.

A function module is entered via a jump from the control module. Return to the control module is via a jump to "REC" in the function module.

The following subroutines are contained in the "RECOVR" module.

1. WR WRITE Routine

   Upon entrance, Q contains number of words to write. The MARK contains location of buffer. Program returns to next location following buffer.

2. SIFT This routine translates input messages into an array. Parameter checking is made on the following:

   a. No more than 4 characters between commas allowed
   b. No more than 5 words inputed.
   c. Only Hex characters read.

3. BIASCI Binary to ASCII code conversion

   Upon entrance, A contains Binary number, exits with ASCII code in A and Q registers.

4. SP Stores Spaces ($2020) in IBUF (Working output buffer)

5. GET Gets one character from "IBUF"

   The I register contains character number, and upon return A and Q register contain character right adjusted.

6. TERMIN Terminates control of the Recovery program. The location specified in $EE is scheduled at level zero. The recovery module is released and control is returned to the Dispatcher.

36.2.2 Additional Statements

In order to process control statements, in addition to those normally included in the list, the following must be done:

1. Add the external name of user's program to process statement to "EXT" list.

2. Add four words defined in coding (see coding of RECOVR)

36.2.3 Entry Interfaces

RECOVR - Entry from "JOBENT" Module
REC - Entered from "RCOVER" Modules, Starts processing procedures for next input statement.
ERR - Error statement entry. Type out error message "ERR".
TERMIN - Terminate Recovery routine.
GET - Gets character from "IBUF".
SIFT - Entry to routine that places input information into an array.

SP - set spaces in IBUF
BIASCI - Binary to ASCII code conversion

36.2.4 External Symbols

FILE2 - Contains absolute location of ▽RCOVER▽ routine.
SWITCH - Set to +1 when ¤Z read by manual interrupt routine.

36.2.5 Internal Symbols

LUNO - Logical Unit number of output device.
ABIBUF - Absolute location of IBUF.
IBUF - 96 word input and output buffer.
BUF - 5 word hex array of input data, set by SIFT routine.

36.3 OUTSEL MODULE

36.3.1 Internal Description

This module allows the operator the option of selecting any device to be used as output medium by the recovery program. The unit number must be legal and the device must be capable of output. If no ▽¤ unit number▽, statement is made, the standard output unit for print information will be used. If an error is detected, control is returned to ERR in the RECOVR module, which prints out ▽ERR▽ and asks for the next statement.

36.3.2 Entry Interfaces

OUTSEL - Entry to module

36.3.3 External Symbols

LOGLA - Logical Unit Table

36.4 DUMP CORE ROUTINE

36.4.1 Internal Description

The control statement ¤Dhhhh, HHHH instructs the recovery program to dump the contents of core beginning at hexadecimal address hhhh and terminating at hexadecimal address HHHH. If the terminating address is less than the starting address, no data will be printed. If the stop address is not given, only one word will be printed. Ten words will be printed on a line together with the starting address of the first word on the line. If the output device is magnetic tape or a printer, 16 words

will be printed on a line. If the last word of a line is equal to each word of the next line, only an asterisk will be printed and no further data will be printed until a variation is detected. The printout uses

only hexadecimal digits. Upon completion of the dump, exit will be made to the control module.

"DMPCOR" routine is so set up that the mass storage dump program will use the "DUMPC" portion of "DMPCOR" routine. The following core locations will be switches used by both routines: COREL, WLOC, STOP, COMPE, ASKWR, B.

36.4.2 Entry Interfaces

DMPCOR - Entry to module
DUMPC - Dump core routine
PRINT - Prints one line of information

36.4.3 Internal Symbols

START - First location to dump.
STOP - Last location to dump or last word.
WLOC - Current word location dumping or word number.
COMPE - Word compare check flag.
ASKWR - Line compare check flag.
WORDC - Word count
LWORD - Last word
COREL - Location of current word dumping
B - Absolute location of IBUF

36.5 MASS STORAGE DUMP

36.5.1 Internal Description

The Control Statement *M S1, W1, S2, W2, N instructs the recovery program to dump mass storage scratch area. Mass storage is dumped beginning with word W1 in sector S1 and terminating with Word W2 in sector S2. The following rules apply for values assigned to S1, W1, W2 and S2.

1. If W1 is omitted, the first word to be dumped is the first word of the first sector to be dumped.
2. If S2 is omitted, it is assumed that S2 has the same value as S1.
3. If W2 is omitted, the last word to be dumped is the last word in sector S2.

Therefore -

1. A single sector will be dumped if only a value of S1 is specified.
2. No information will be dumped if S1 is greater than S2.

3. No information will be dumped if S1 equals S2 and W1 is greater than W2.

If N does not appear, the library unit is assumed. If N appears, it must be the logical unit number of some mass storage device.

The starting scratch sector specified will be read into core. The communication flags will be set and the buffer dumped by DUMPC in "DMPCOR" routine.

The first line printed will defined sector number followed by dump of sector in hex. If the contents of the last word of a line equals the contents of each word in the following line, only an asterisk will be printed. No further printout, except for sector number will be printed until a variation is detected. Upon completion of the printout, exit will be made to the control module.

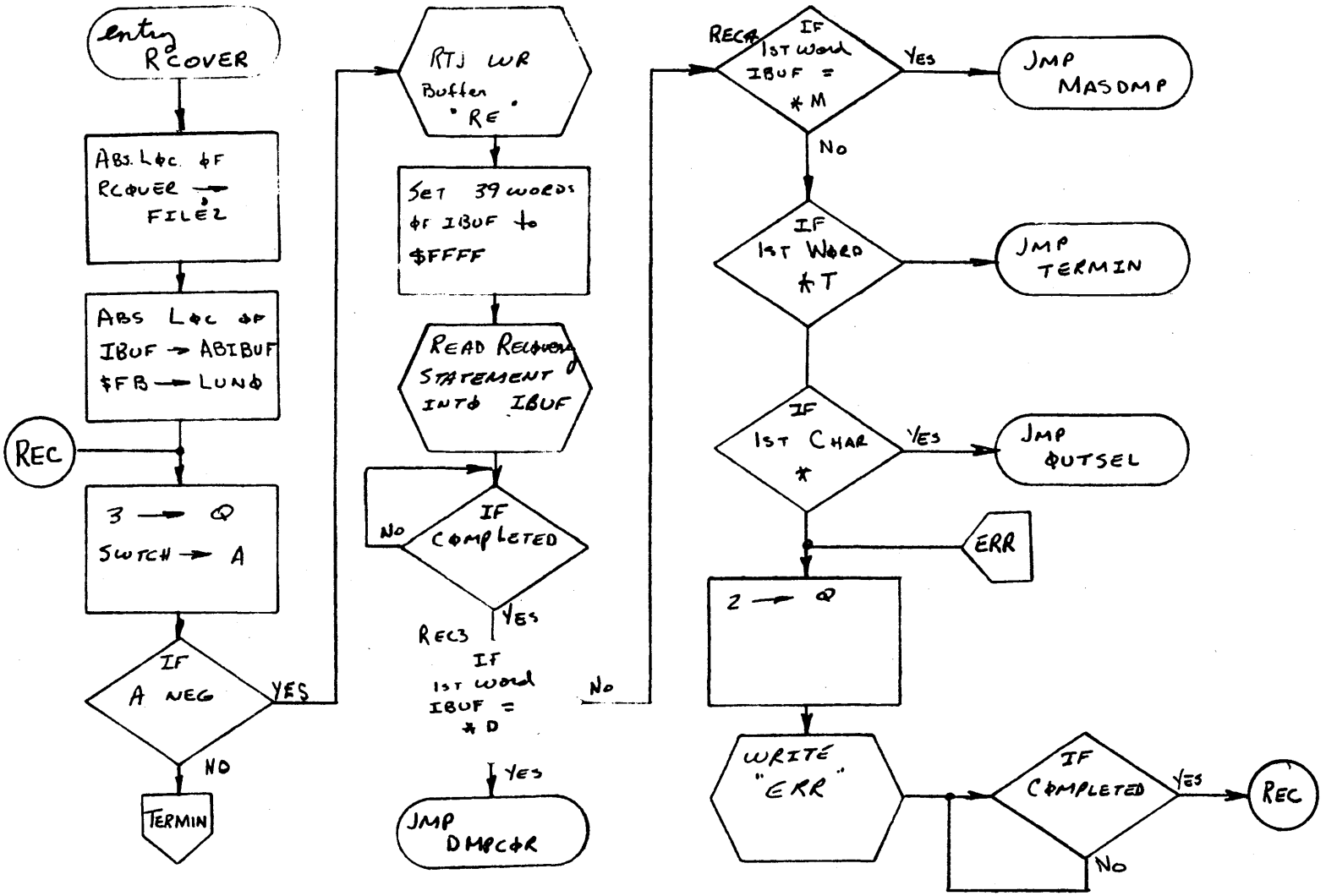
36.4.2 Entry Interfaces

MASDMP- Entry to module
SECNO - Sequence number printout routine
GETFIL- Get mass storage sector to be output.

36.4.3 Internal Symbols

S1 - Starting Sector No.
W1 - First word of starting sector
W2 - Last word of last sector
S2 - Last sector
N - Logical Unit Number
LCSWT - Switch set $\geq \neq$ absolute sector number wanted.
MBUF - 96 word buffer

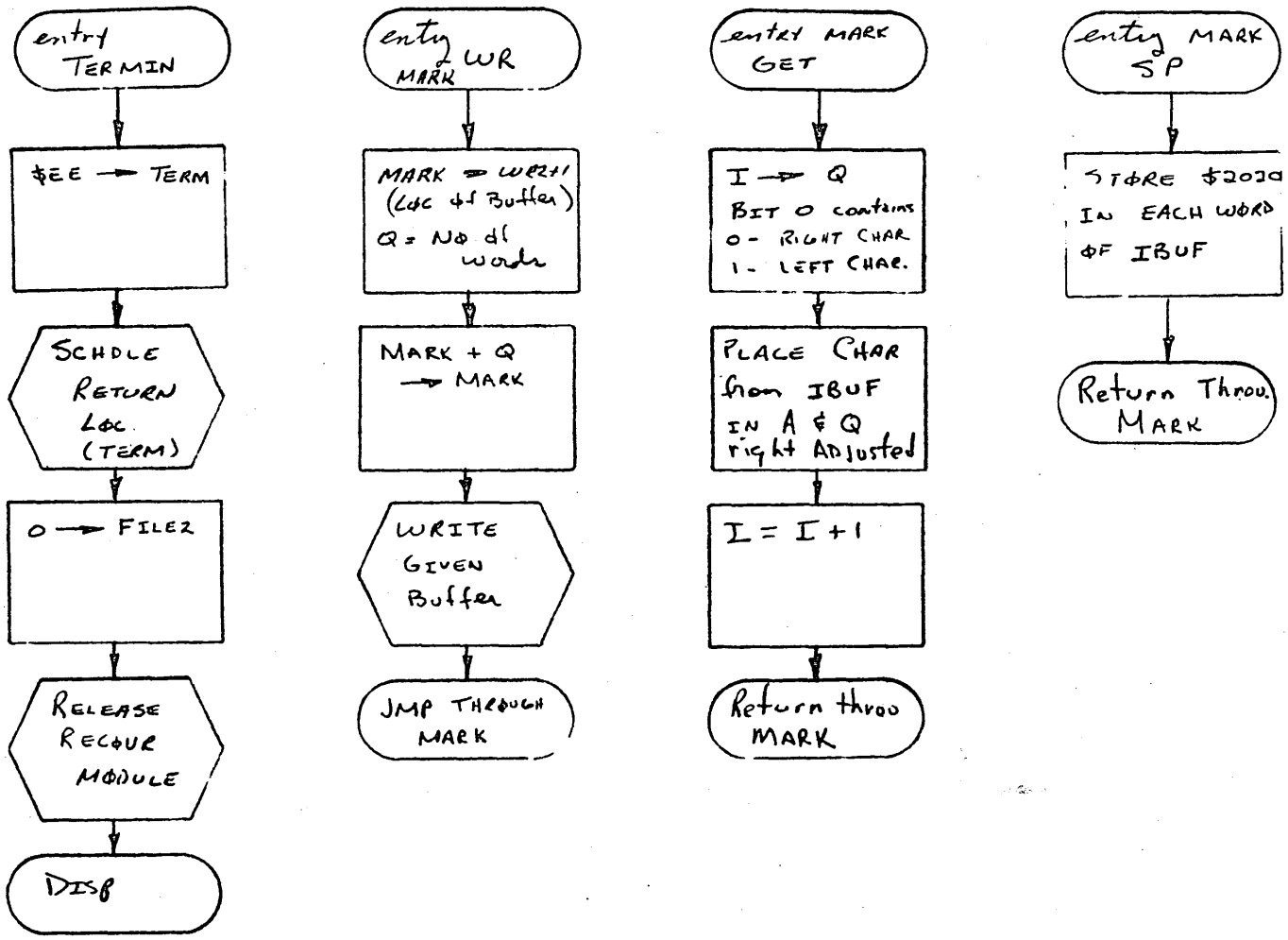
RCOVER



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE RECOVER		PROJECT MGR.		
	PAGE 1 OF 4		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

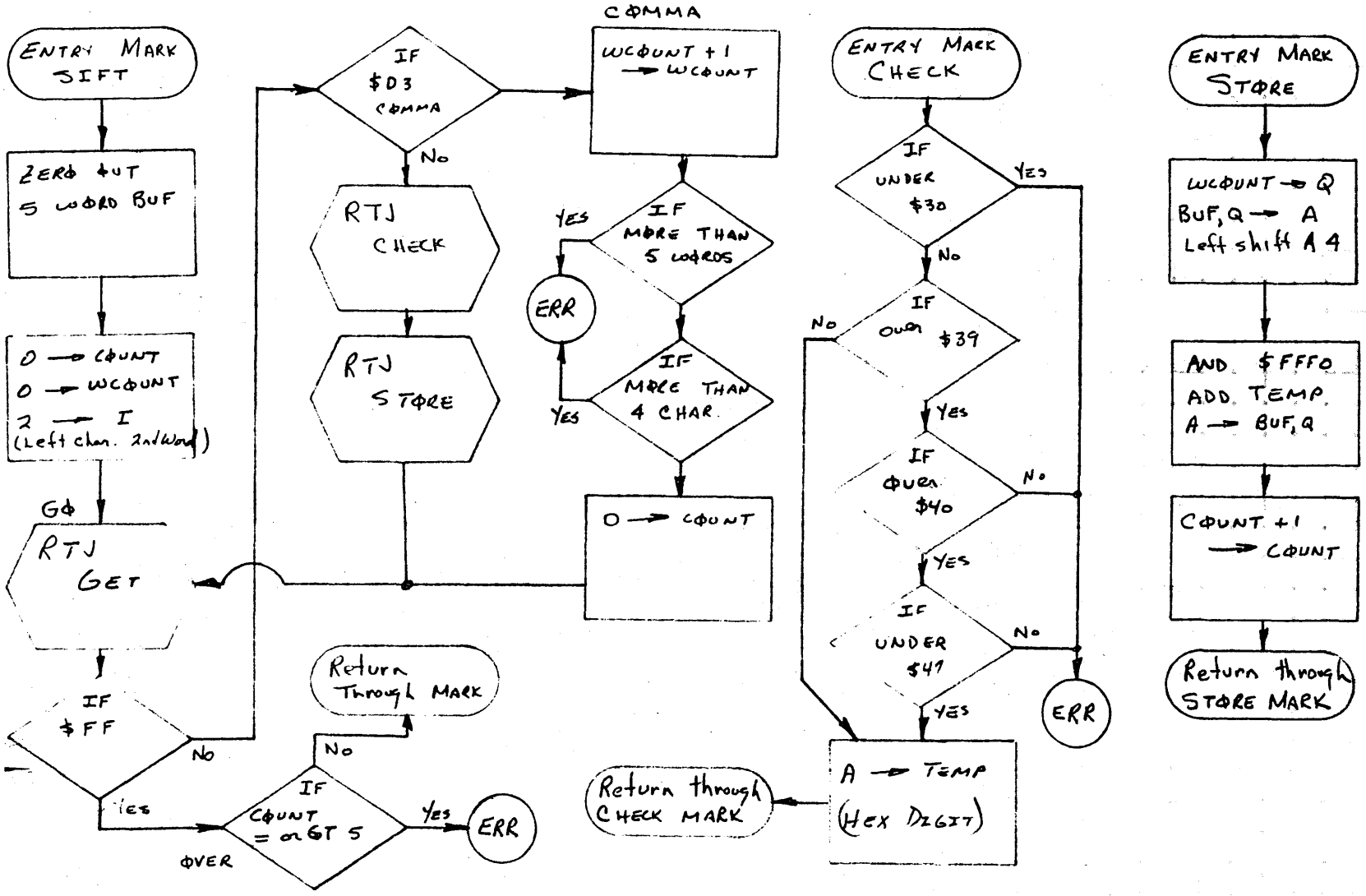
MAR 5 1971

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RECOVERY		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

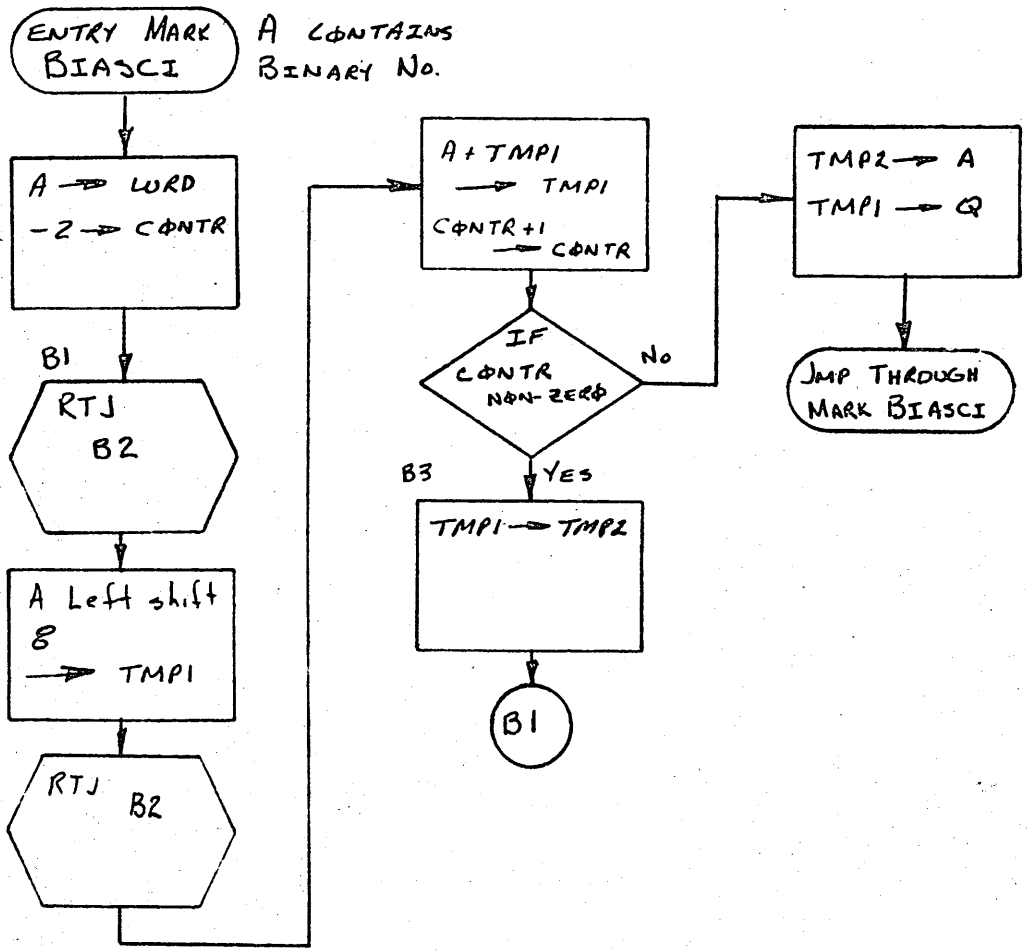
MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: Ims	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: RECOVERY		PROJECT MGR.		
	PAGE 3 OF 4		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

AA1385 (FORMERLY CA127-1)

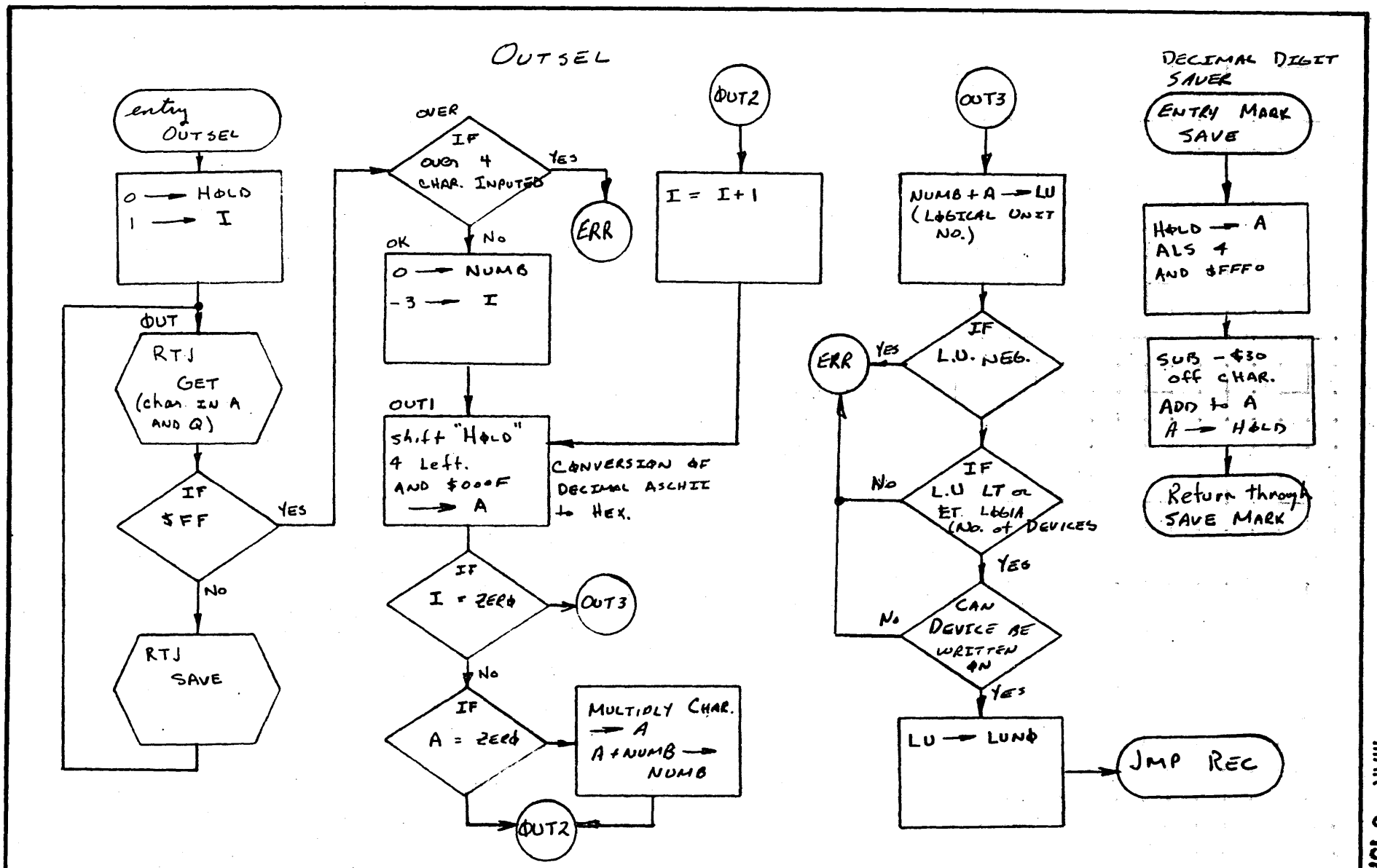


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RECOVERY	PROJECT MGR.			
	PAGE 4 OF 4	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1970

E4.6

AA1385 (FORMER CA 27)



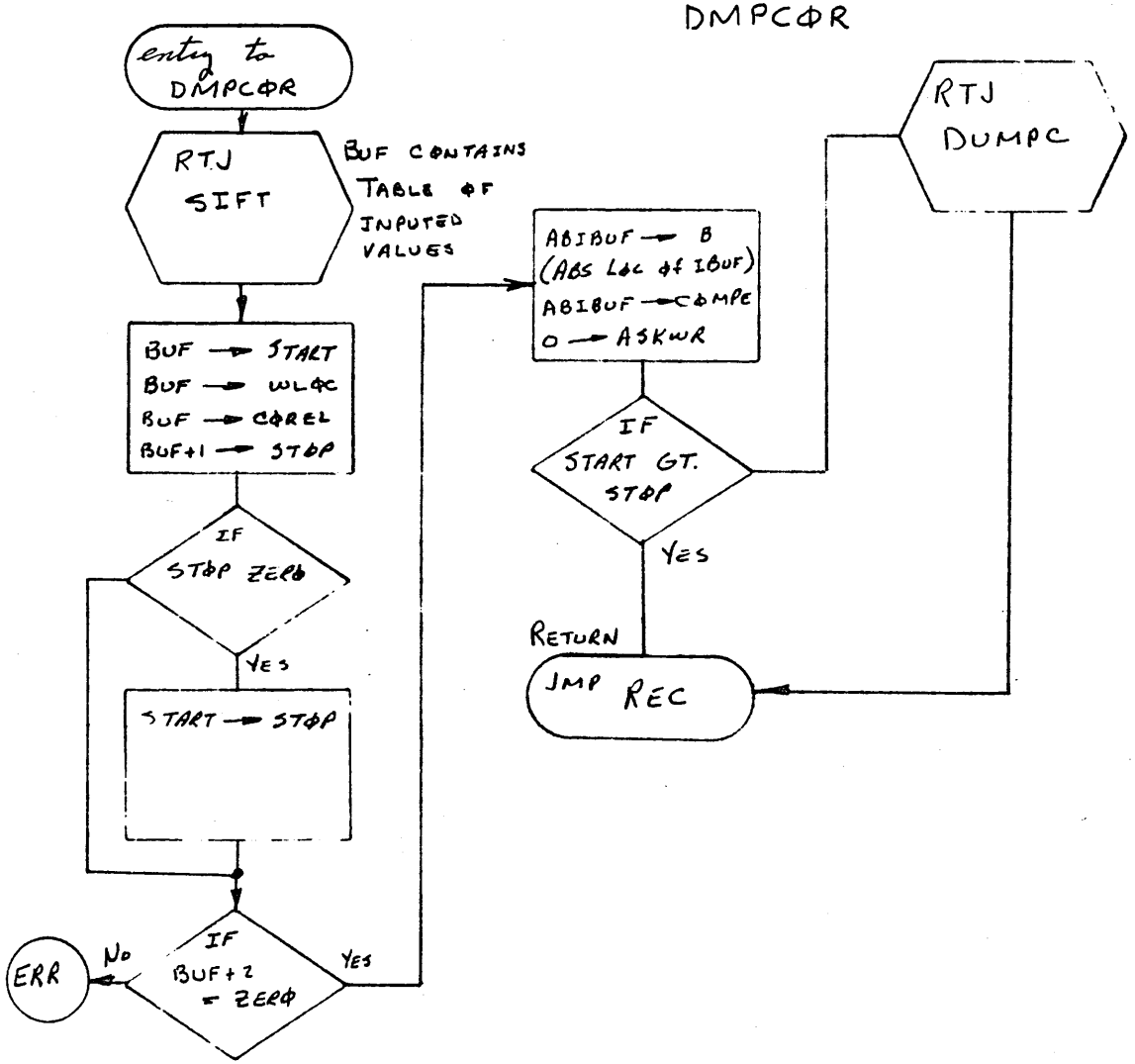
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE RECOVERY		PROJECT MGR.		
	PAGE 1 OF 1		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

3b.10

AA1385 (FORMERLY CA127-1)

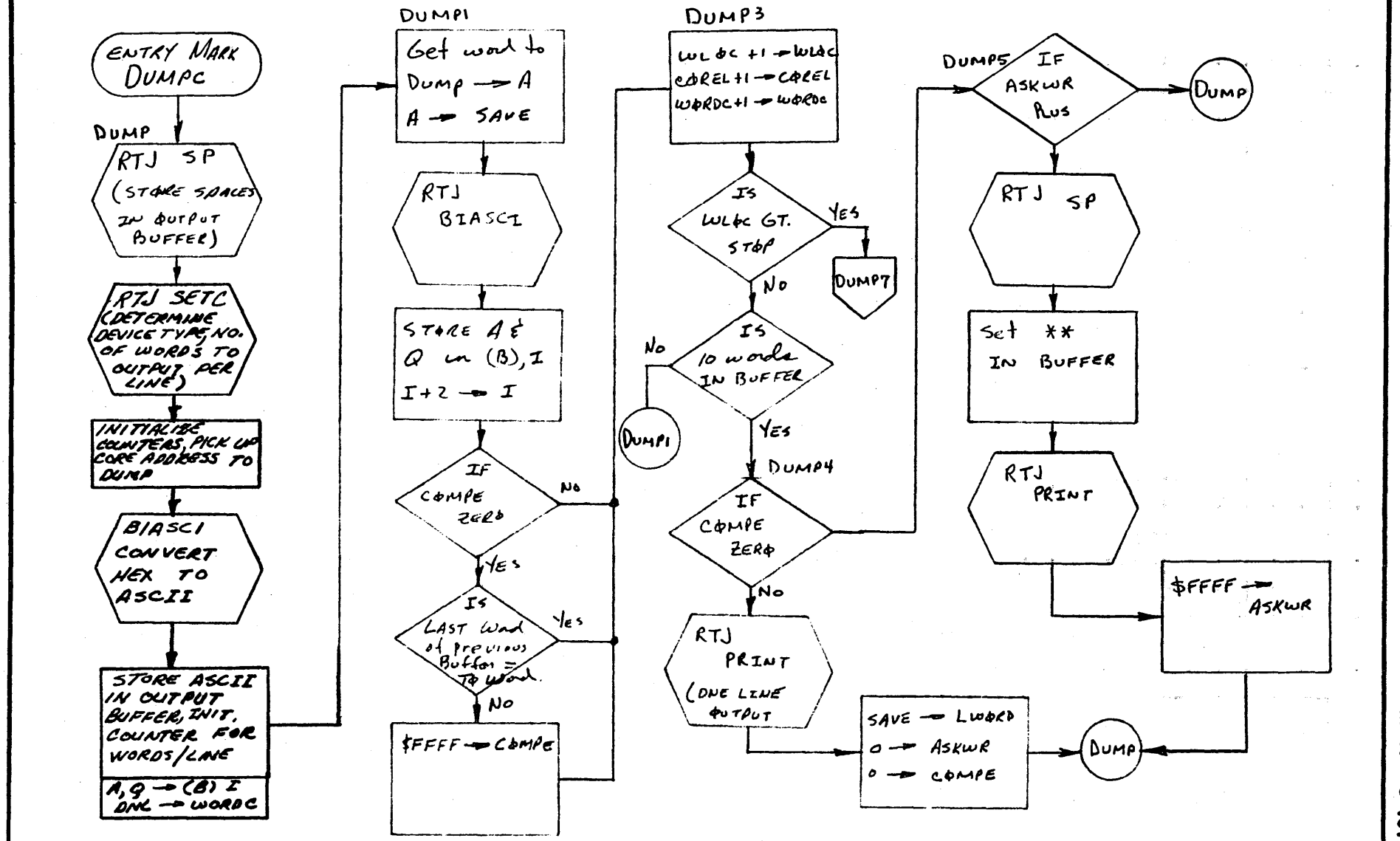
DMPCDR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RECOVERY		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

36.11

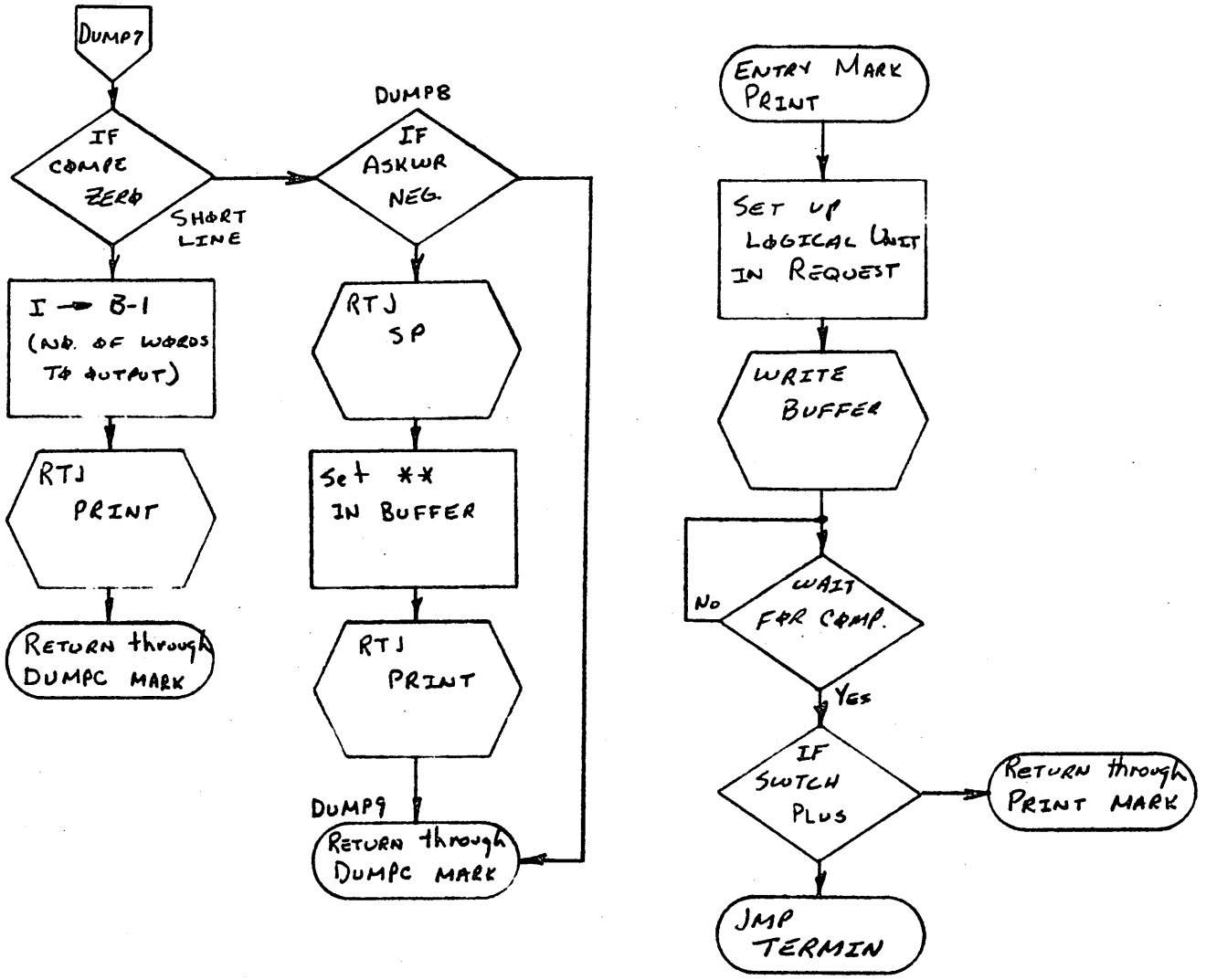


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS Ims	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE RECOVERY		PROJECT MGR.		
		PAGE 2 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

34.12

AA1385 (FORMERLY CA127-1)

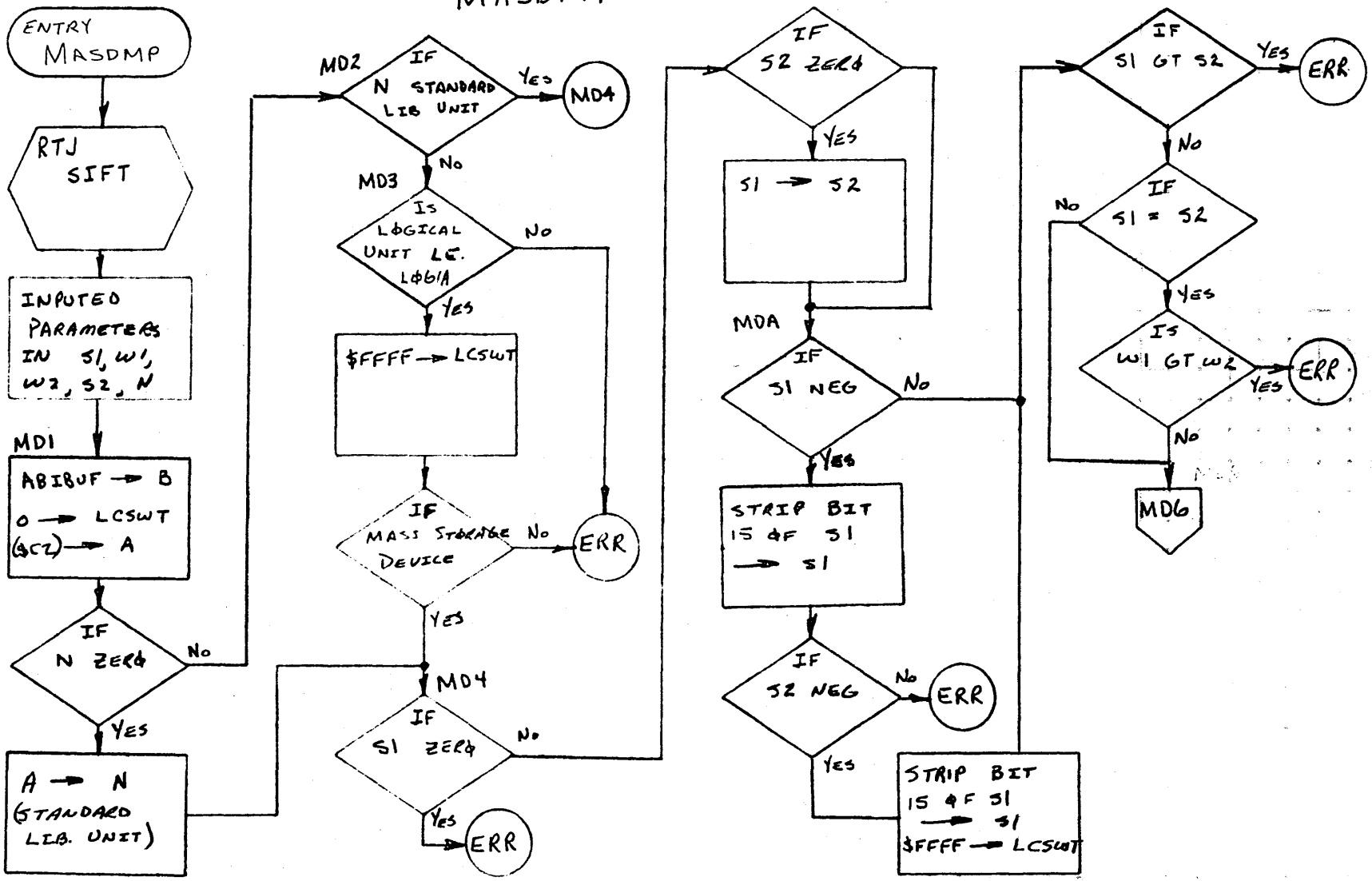


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RECOVERY		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

36 13

MASDMP

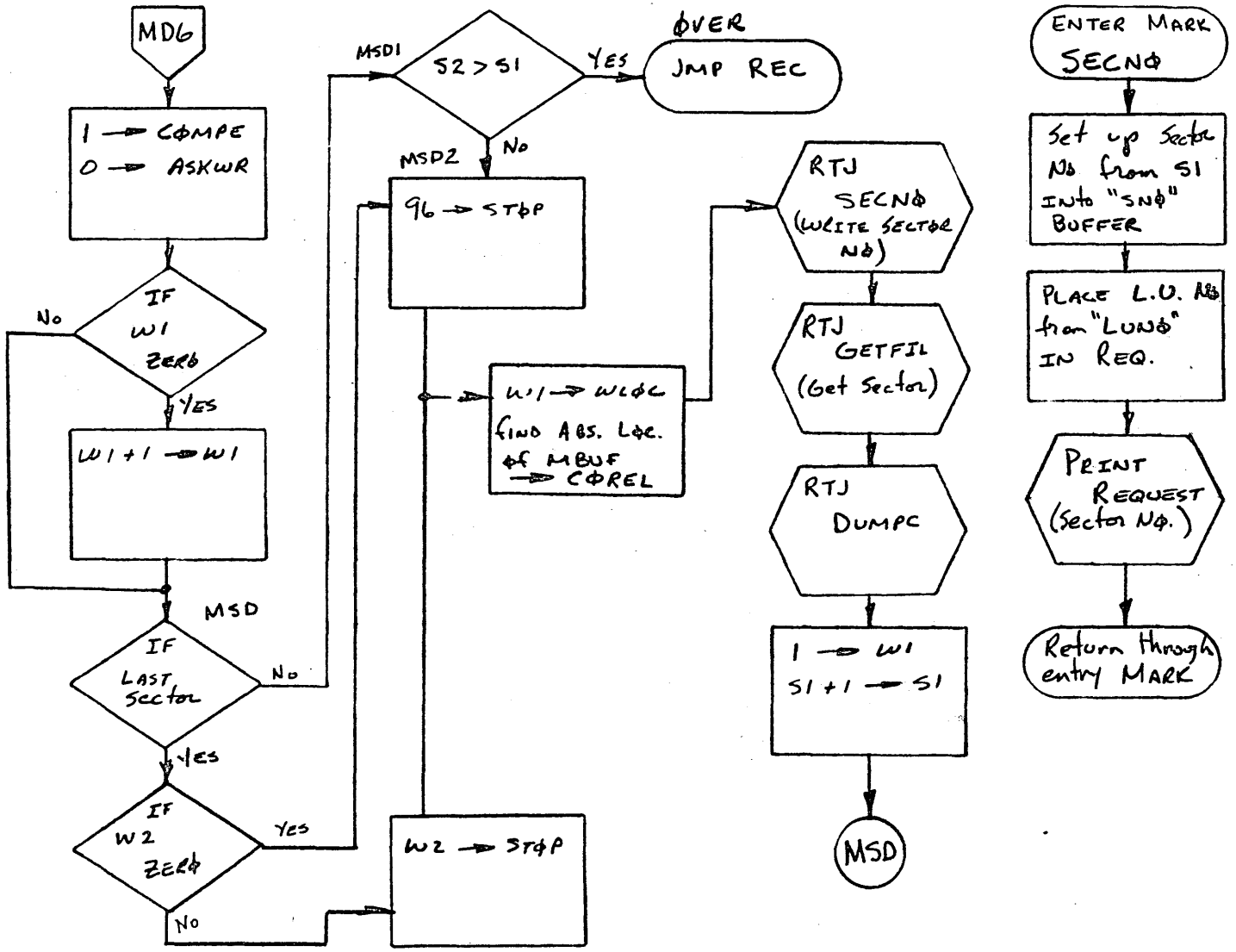


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE RECOVERY		PROJECT MGR.		
		PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

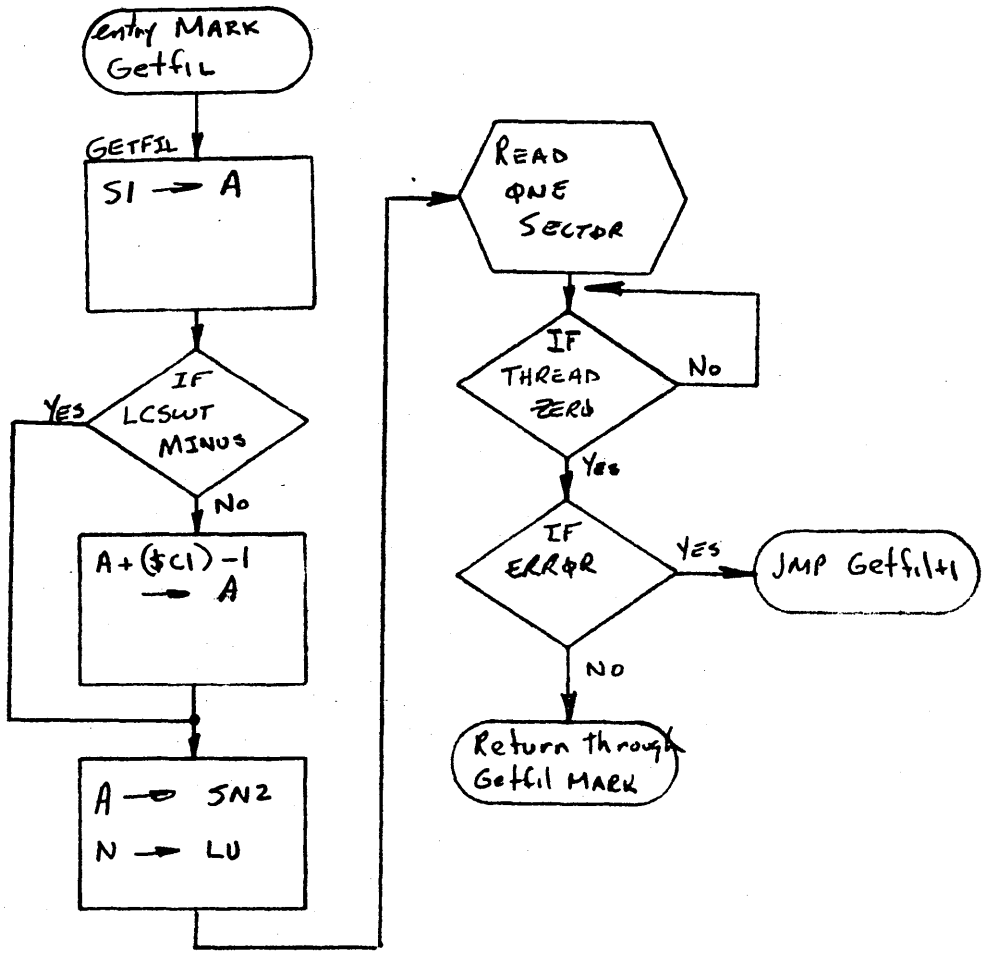
39.14



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RECOVERY	PROJECT MGR.			
	PAGE 2 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1979

AA-385 (FORMER CA-37)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE RECOVERY		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

36.16

AA1385 (FORMERLY CA127-1)

BREAKPOINT PROGRAM

37.0 PROGRAM FUNCTION

The breakpoint program enables the operation of a job to be controlled from the COMMENT medium. A breakpoint is set initially to a core address in the command sequence of a program and the program is interrupted prior to execution of the instruction stored at the breakpoint location.

The breakpoint program is placed in the system library on mass storage by the system initializer. A *B statement instructs the Job Processor to turn on the breakpoint load switch. If this switch is on when an execute statement (*X) is detected, the Job Processor places the breakpoint program after the job in unprotected core.

Before executing the job, the breakpoint program is entered. The entry point to the breakpoint program is stored in $F3, (unprotected core location). A printout, BP, on the comment device indicates entrance to the program. The operator can then direct the operation of the breakpoint program by entering input statements through the comment device.

When the job attempts to execute an instruction at an address to which a breakpoint has been set, the breakpoint program is entered with an indirect return jump through $F3. BP, location of breakpoint, is printed on the comment device.

37.2 BRKPTD ROUTINE

37.2.1 Internal Description

"BRKPTD" is entered from Job Processor (JPT13) module with the Q register containing the job entry location. $F3 is changed to the address of BRKP and control is passed to the READY internal entry point, which prints out BP. Upon entrance through BRKP the A, Q and I registers are saved. BP, (location where breakpoint set) is printed out. The breakpoint statement is then requested on the comment medium. Upon completion of the inputed statement, the first word is compared with a list of expected values. If none compare, a B01, statement is printed. If a match is obtained, the input statement is broken down into an array (BUF1). Upon jumping to the requested processor, the A register contains number of words in BUF1 and the Q register contains the absolute location of BUF1.

The B01 entry causes the error message B01 statement to be printed out on the comment medium. Upon completion another breakpoint statement is requested.

37.2.2 Control Statements

The standard list of control statements for the breakpoint program are:

INPUT STATEMENT	COMPARE WORD	TRANSFER ADDRESS
*Ahhhh cr	*A	ENTERA
*C cr	*C	RESUME
*Dhhhh,HHHH cr	*D	DMPCOR
*Ehhhh,hhhh,... cr	*E	ENTCOR
*Ihhhh cr	*I	ENTERI
*Jhhhh cr	*J	JUMPTO
*M,S1,W1,S2,W2,N cr	*M	MASDMP
*P cr	*P	PRTREG
*Qhhhh cr	*Q	ENTERQ
*Rhhhh cr	*R	RETJMP
*Shhhh,hhhh,... cr	*S	SETRRP
*T,hhhh,hhhh,... cr	*T	TERMIN

The transfer addresses are declared as external names in the control module of the breakpoint program and as entry point names in the function modules. Each function is entered via a jump from the control module (BRKPTD). The return to the control module is via a jump to the entry point (READY) in the function module.

37.2.3 Illegal Input Statement

The breakpoint program will reject any control statement read for which there is no entry in its own list of control statements. Unacceptable control statements will be indicated via the comment B01, statement. The breakpoint program will then wait for another control statement.

37.2.4 Additional Statements

To add or subtract function modules, the first two characters of the input statement must be added or deleted from the BPTAB table. The proper transfer address must be added or deleted as an external, and added or deleted from the transfer table BPREL.

37.2.4 Entry Interfaces

BRKPTD	Entry from Job Processor (JPT13) module
BRKP	Entry from Job Address set in $F3
READY	Entry to process next Breakpoint instruction
B01	Entry to B01, statement error message processor
BUF	A 40 word temporary buffer used for input and output
REG	First word address where the A, Q, I and M registers upon entrance are stored

37.2.5 External Symbols

RBKP05 - SIFT Places input statements into array
BRKP07 - BIASCI Binary to ASCII conversion
BRKP10 - BPL Table of breakpoint locations
(BPL - 1) contains location of breakpoint currently processing.
BRKP11 - INST Table of breakpoint instructions
(INST - 1) contains instruction at breakpoint, currently processing
BRKP12 - BPCOUNT Number of breakpoints set

37.2.6 Internal Symbols

USERP Address of user's entry. If set to address first time to breakpoint
LOC Location of breakpoint
LASTAB Must be last card for "BPTAB' table

37.3 SIFT ROUTINE

37.3.1 Internal Description

This routine transfers the ASCII input buffer to a hexadecimal array. Upon entrance the array BUF1 is set to zero, flags are set and the input is scanned to find either a $FF or a comma ($2C). The preceeding characters are formed into a word with the following checks made.

1. Hex ASCII character only.
2. No more than 4 characters read.

The Hex characters in BUF are transferred to binary and stored consecutively in the BUF1. Upon reading a $FF, the last word is formed, the A register is set to the number of words in BUF1 and the Q register contains the absolute location of BUF1.

If more than 15 words read or one of the above errors detected, a jump to B01 error message is made.

37.3.2 Entry Interfaces

SIFT Program entry
GET Gets one character from BUF

37.3.3 External Symbols

BRKP03 - BUF The buffer which contains Input statement.
BRKP04 - B01 Error message B01 routine.

37.3.4 Internal Symbols

COUNT	Character counter
WCOUNT	Word counter
TEMP	Temporary storage of HEX number
BUF1	40 word hexadecimal buffer of input data, also used as output buffer

37.4 BIASCI ROUTINE

37.4.1 Internal Description

This routine takes the binary number in the A register and transfers it to an ASCII character which are stored in A and Q registers upon return.

The SP routine stores $2020 in 40 word BUF1. The PROTEC routine checks the location specified in the A register to see if it is in unprotected core; if so, control is returned to call routine with A and Q registers set to entry condition. If an error is detected, the error message B02,hhhh is printed on the comment medium, where hhhh is the location in protected core. Control is returned to the calling program with the A register set to $FFFF.

37.4.2 Entry Interfaces

BIASCI	Binary to ASCII code conversion
SP	Stores ASCII code for spaces in BUF1
PROTEC	Checks if location in the A register is unprotected

37.4.3 External Symbols

BRKP06 - BUF1	Input output buffer

37.5 RETJMP ROUTINE

37.5.1 Internal Description

The *Rhhhh statement instructs this module to return jump to location hhhh and begin execution at hexadecimal address hhhh+1. If the return jump is made to a closed loop, the breakpoint routine will be re-entered when an indirect jump is made to the address stored in location hhhh.

The address hhhh is checked by PROTEC to see if unprotected. If the A register is negative upon returning, control is returned to READY. If not, the A, Q, I registers are restored, and control is returned jumped to hhhh.

37.5.2 Entry Interfaces

RETJMP	Program entry

37.5.3 External Symbols

BRKP01 - READY	Prints BP and Requests next statement
BRKP02 - REG	Registers upon entrance stored here
BRKP04 - B01	Error return
BRKP09 - PROTEC	Checks if contents in A unprotected

37.6 JUMPTO ROUTINE

37.6.1 Internal Description

The *Jhhhh statement instructs this module to jump to location hhhh and begin execution at hexadecimal address hhhh.

Upon entrance the input array BUF1 is checked for one value only. If not, a B01 error message is printed.

The address hhhh is checked by PROTEC to see if unprotected. If the A register is negative upon returning, control is returned to READY. If not, the Q, A and I registers are restored, and control is jumped to hhhh.

37.6.2 Entry Interfaces

JUMPTO	Program entry

37.6.3 External Symbols

BRKP01 - READY	Prints BP and requests next statement
BRKP02 - REG	Register upon entrance to breakpoint
BRKP04 - B01	Error return
BRKP02 - PROTEC	Checks if contents in A unprotected

37.7 ENTER ROUTINE

37.7.1 Internal Description

These instructions *Ahhhh, *Qhhhh, *Ihhhh instruct the enter program to place the hexadecimal value of hhhh in the A, Q or I registers respectively. Upon entrance, the input buffer array BUF1 is checked for one term. If one not present, the error message B01 is called. Upon finding a value, it's stored in REG array for the appropriate register.

37.7.2 Entry Interfaces

ENTERQ	Enter in Q register routine
ENTERA	Enter in A register routine
ENTERI	Enter in I register routine

37.7.3 External Symbols

BRKP01 - READY	Prints BP and requests next statement
BRKP02 - REG	Registers upon entrance to breakpoint
BRKP04 - B01	Error return

37.8 ENTER CORE ROUTINE {ENTCOR}

37.8.1 Internal Description

The statement ¤EHHHH,hhhh,...hhhh (cr) instructs ENTCOR routine enters the consecutive hexadecimal constants, hhhh, into sequential memory locations of unprotected core beginning with HHHH. Consecutive constants must be separated by commas and a carriage return must follow the last constant. Consecutive commas indicate the contents of the location that would receive the missing constant are not to be disturbed. If a location is to be set to zero, a zero must be indicated via a constant in the statement. A maximum of 15 locations may be altered by a single ¤E statement.

The array following the 16 word array in BUF1, contains $FFFF for each entry where two commas are not separated by a constant.

37.8.2 Entry Interfaces

ENTCOR	Program entry

37.8.3 External Symbols

BRKP01 - READY	Prints BP and requests next statement
BRKP04 - B01	Error return
BRKP09 - PROTEC	Checks if contents in A register unprotected

37.9 REGISTER PRINTOUT {PRTREG}

37.9.1 Internal Description

The contents of the A, Q, I, M and P registers are printed with tags specifying each register. i.e. A=HHHH, Q=HHHH, etc.

Upon entrance the input buffer array BUF1 is checked for values, if any exist, the error message B01 is requested. If none exist, the values in REG are transferred to ASCII and stored in print buffer (BUF1).

Upon transferring all values to ASCII and storing them in BUF1, it is printed out.

37.9.2 Entry Interfaces

PRTREG	Program entry

37.9.3 External Symbols

BRKP01 - READY	Prints BP and requests next statement
BRKP02 - REG	Registers upon entrance to breakpoint
BRKP04 - B01	Error return
BRKP07 - BIASCI	Binary to ASCII conversion routine
BRKP08 - SP	Stores spaces characters in BUF1

37.10 SET BREAKPOINT ROUTINE (SETBRP)

37.10.1 Internal Description

The control statement *Shhhh,hhhh,... (cr) instructs the SETBRP program to set a breakpoint switch at each of the hexadecimal addresses specified. Each heaxdecimal address, hhhh, must be a memory location in umprotected core. If an address in protected core is specified, it is rejected and an error message B02, illegal address, is printed on the comment device. The breakpoint program then processes the next address in the input block.

Up to 15 hexadecimal addresses may appear in a control statement. The total number of breakpoints which may be set at any time is represented by the value in HOBP. The value of this parameter is fixed at the time the breakpoint program is assembled. It may be changed only by reassembly of the program.

The SETBRP program will ignore a hexadecimal address hhhh in the input block of a breakpoint had previously been set at this address. No indication of this action will be given.

Locations were breakpoints are set and stored in the BPL table. Instructions that were replaced by breakpoints are stored in INST table. The number of breakpoints set is defined by BPCOUT. If BPCOUT is greater than NOBP (number of breakpoints that can be set) the error message B03,hhhh is printed and the rest of the statement is ignored. hhhh is defined as the first location breakpoint will not be set.

37.10.2 Entry Interfaces

SETBRP	Program entry
BPL	Table of breakpoint locations
INST	Table of instructions replaced by breakpoints
BPCONT	Number of breakpoints set

37.10.3 External Symbols

BRKP01 - READY	Prints BP and requests next statement
BRKP04 - B01	Error return
BRKP07 - BIASCI	Binary to ASCII conversion routine
BRKP09 - PROTEC	Checks if contents in A unprotected

37.11 REMOVE BREAKPOINTS (TERMIN)

37.11.1 Internal Description

The statement *Thhhh,hhhh,... (cr) instructs TERMIN to remove the breakpoint from the hexadecimal address hhhh. If a breakpoint is not set at a given address, the TERMIN program processes the next address. If a *T statement is not accompanied by any hexadecimal addresses, all breakpoints previously set are removed. Up to 15 addresses are allowed in a single *T statement.

To remove all breakpoints, location defined in BPL are stored with values in INST followed by setting BPCONT to zero.

Upon removing specified breakpoints, BPL is set to $FFFF. Upon completion of a *T statement, the BPL and INST tables are packed to delete removed breakpoints. BPCONT is changed to the number of breakpoints left.

37.11.2 Fntry Interfaces

TERMIN	Program entry

37.11.3 External Symbols

BRKP01 - READY	Prints BP and requests next statement
BRKP10 - BPL	Table of breakpoint locations
BRKP11 - INST	Table of instructions replaced by breakpoints
BRKP12- BPCONT	Number of breakpoints set

37.12 DUMP CORE ROUTINE (DMPCOR)

37.12.1 Internal Description

The control statement *Dhhhh,HHHH instructs the DMPCOR program to dump the contents of core beginning at hexadecimal address hhhh and terminating at hexadecimal address HHHH. If the terminating address is less than the starting address, no data will be printed. If the stop address is not given, only one word will be printed. Ten words will be printed on a line together with the starting address of the first word on the line. If the last word of a line is equal to each word of the next line, only an asterisk will be printed and no further data will be printed until a variation is detected. The printout uses only hexadecimal digits. Upon completion of the dump, exit will be made to the control module (READY).

DMPCOR routine is so set up that the mass storage dump program will use the DUMPC portion of DMPCOR routine. The following core locations will be switches used by both routines: COREL, WLOC, STOP, COMPE, ASKWR, B.

37.12.2 Entry Interfaces

DMPCOR	Entry to module
DUMPC	Dump core routine
PRINT	Prints one line of information

37.12.3 Internal Symbols

START	First location to dump
STOP	Last location to dump or last word
WLOC	Current word location dumping or word number
COMPE	Word compare check flag
ASKWR	Line compare check flag
WORDC	Word count
LWORD	Last word
COREL	Location of current word dumping
B	Absolute location of BUF1

37.12.4 External Symbols

BRKP01 - READY	Prints BP and requests next statement
BRKP04 - B01	Error return
BRKP07 - BIASCI	Binary to ASCII conversion routine
BRKP08 - SP	Store space character in BUF1

37.13 MASS STORAGE DUMP

37.13.1 Internal Description

The control statement *M,S1,W1,S2,W2,N instructs the MASDMP program to dump mass storage scratch area. Mass storage is dumped beginning with word W1 in sector S1 and terminating with word W2 in sector S2. The following rules apply for values assigned to S1, W1, W2 and S2.

1. A single sector will be dumped if only a value of S1 is specified.
2. No information will be dumped if S1 is greater than S2.
3. No information will be dumped if S1 equals S2 and W1 is greater than W2.

If N does not appear, the library unit is assumed. If N appears, it must be the logical unit number of some mass storage device.

The starting scratch sector specified will be read into core. The communication flags will be set and the buffer dumped by DUMPC in DMPCOR routine.

The first line printed will be defined sector number followed by dump of sector in hex. If the contents of the last word of a line equals the contents of each word in the following line, only an asterisk will be printed. No further printout, except for sector number will be printed until a variation is detected. Upon completion of the printout, exit will be made to the control module.

37.13.2 Entry Interfaces

MASDMP	Entry to module
SECNO	Sequence number printout routine
GETFIL	Get mass storage sectors to be output

37.13.3 Internal Symbols

S1	Starting Sector Number
W1	First word of last sector
W2	Last word of last sector
S2	Last sector
N	Logical unit number
MBUF	96 word buffer

37.13.4 External Symbols

BRKP01 - READY	Print BP and request next statement
BRKP04 - B01	Error return
BRKP07 - BIASCI	Binary to ASCII conversion routine
BRKP08 - SP	Store space character in BUF1
BRKP20 - DUMPC	Dump core routine
BRKP21 - COREL	Location of current word dumping
BRKP22 - WLOC	Current word location dumping or word number
BRKP23 - STOP	Last location to dump or last word
BRKP24 - COMPE	Word compare check flag
BRKP25 - ASKWR	Line compare check flag
BRKP26 - B	Absolute location of BUF1

37.14 RESUME MODULE

37.14.1 Internal Description

This statement *C instructs RESUME to exit the breakpoint routine and to resume execution of the job. If the breakpoint program had been initially entered from the Job Processor (JPT13) immediately after having been loaded, the breakpoint program will put the job into execution by transferring control to the address passed to the breakpoint program in Q.

If the breakpoint program had been entered from the job, the instruction stored at the address of the last breakpoint will be executed.

The location and instruction will be stored in the location preceding BPL and INST tables. Upon entrance to breakpoint, the instruction to be interpreted will be broken down into 5 groups.

1. Skip type instruction
2. Register type instruction
3. Indirect relative mode
4. Absolute and indirect absolute
5. Direct relative

The last three groups of instructions will be translated into an indirect absolute instruction before executing this new instruction. It is checked for being a return jump; if so, it is interpreted as a return jump instruction.

The A, Q and I registers are returned, and the generated instruction is executed and control is returned to the job.

37.14.2 Entry Interfaces

RESUME	Program entry

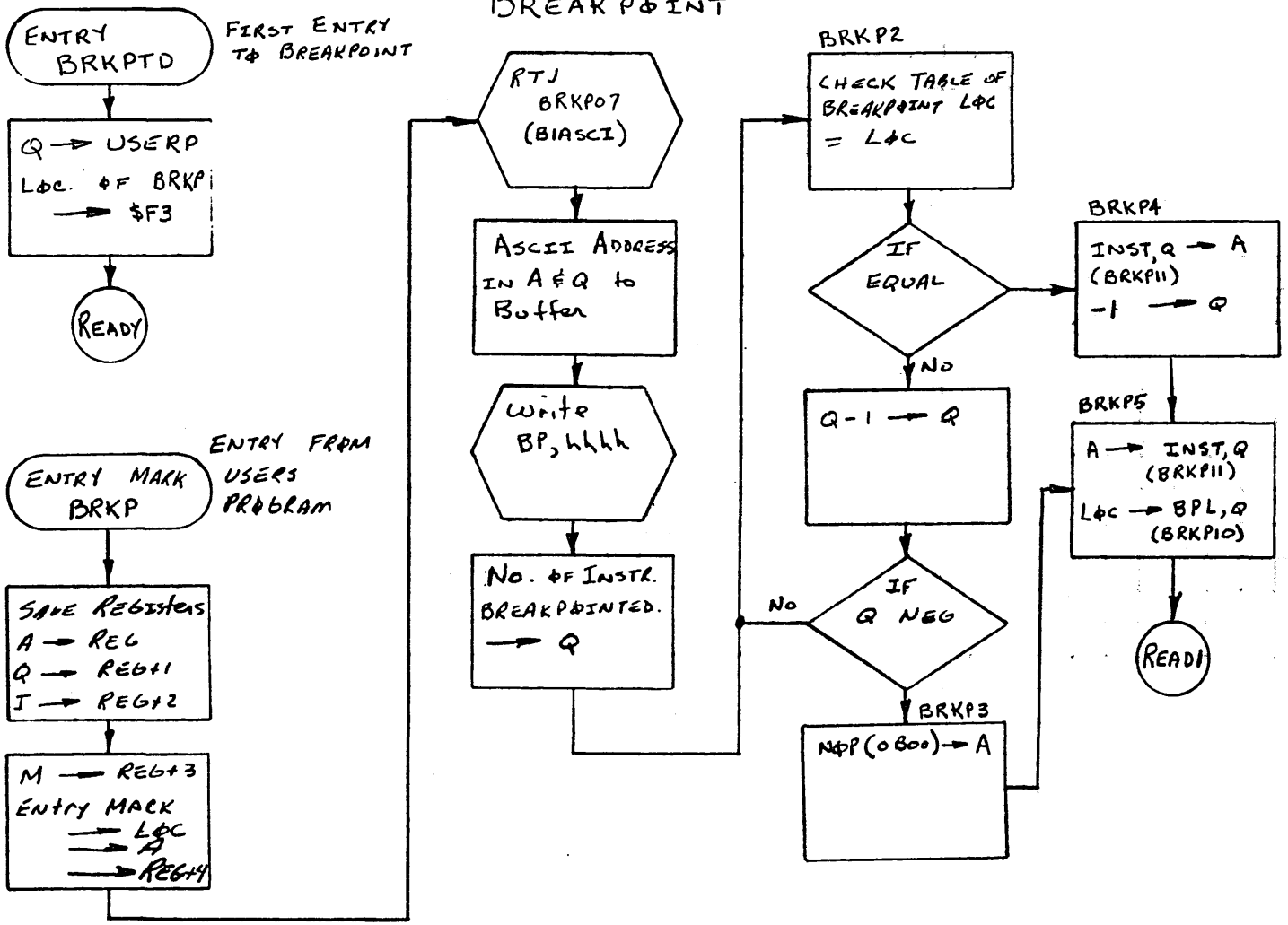
37.14.3 External Symbols

BRKP02 - REG	Register upon entrance to breakpoint
BRKP04 - BO1	Error routine
BRKP10 - BPL	Table of breakpoint locations
BRKP11 - INST	Table of instructions replaced by breakpoint
BRKP12 - BPCONT	Number of breakpoints set

37 14.4 Internal Symbols

LOC	Location of instruction
LOC1	1st word location following instruction
LOC2	2nd word location following instruction
INSTR	Instruction
INSTN	First 4 bits of instruction
MODE	Mode of operation
DELT	Delta field
REGA	A register saved
REGQ	Q register saved
REGI	I register saved

BREAKPOINT

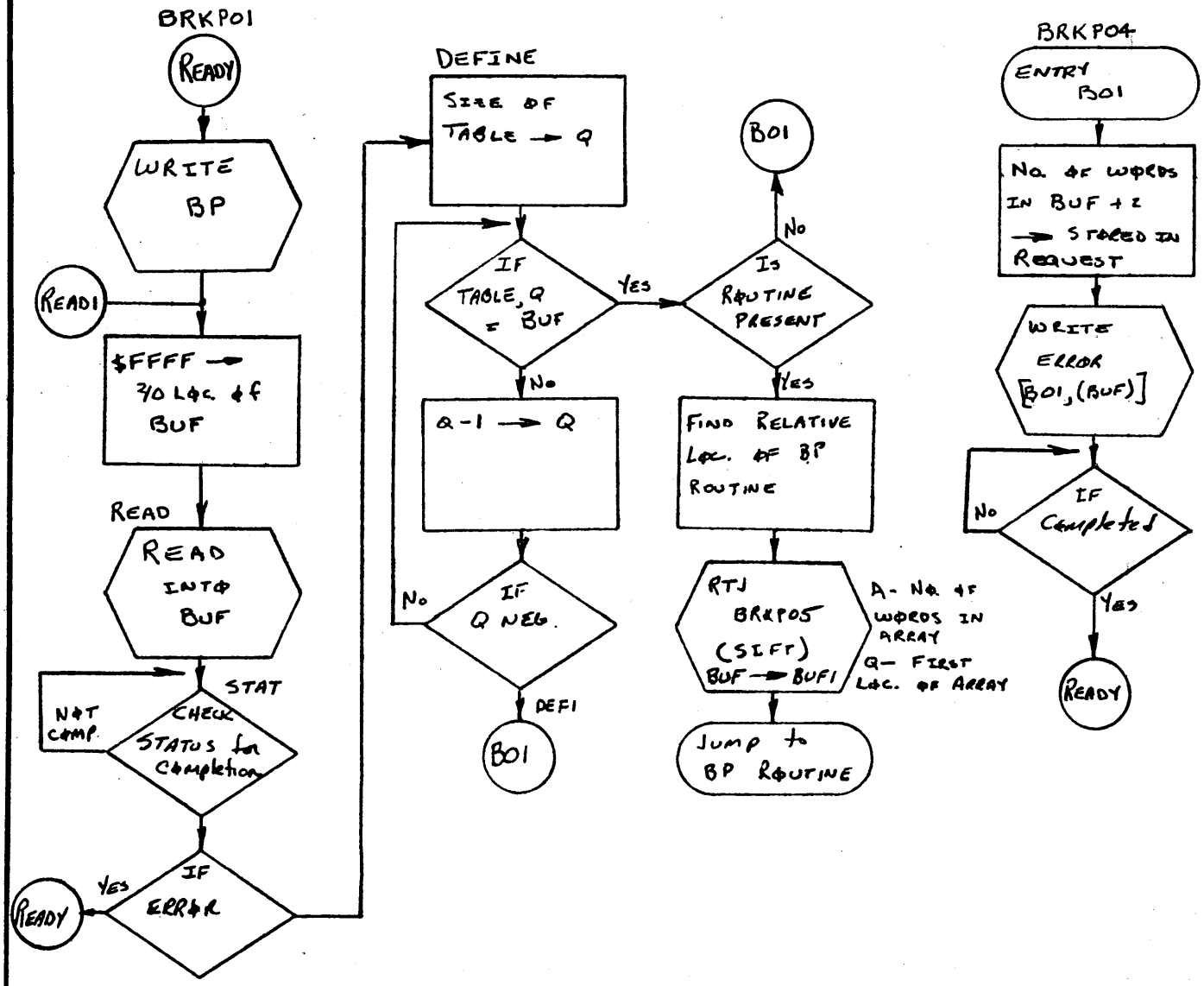


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BREAKPOINT (BRKPTD)	PAGE 1 OF 2	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

37.13



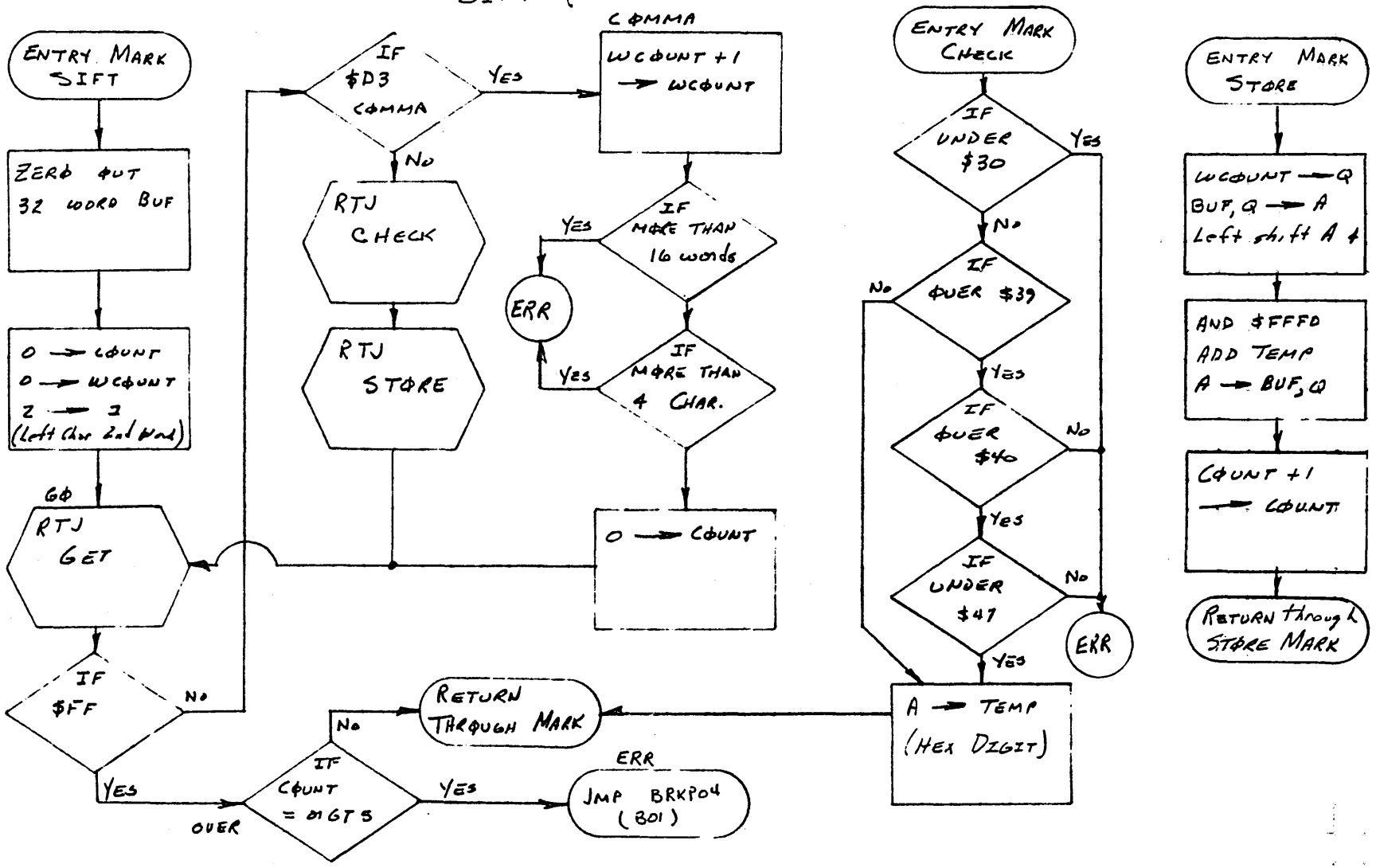
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.			
	BRKPTD	PAGE 2 OF 2	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

3214

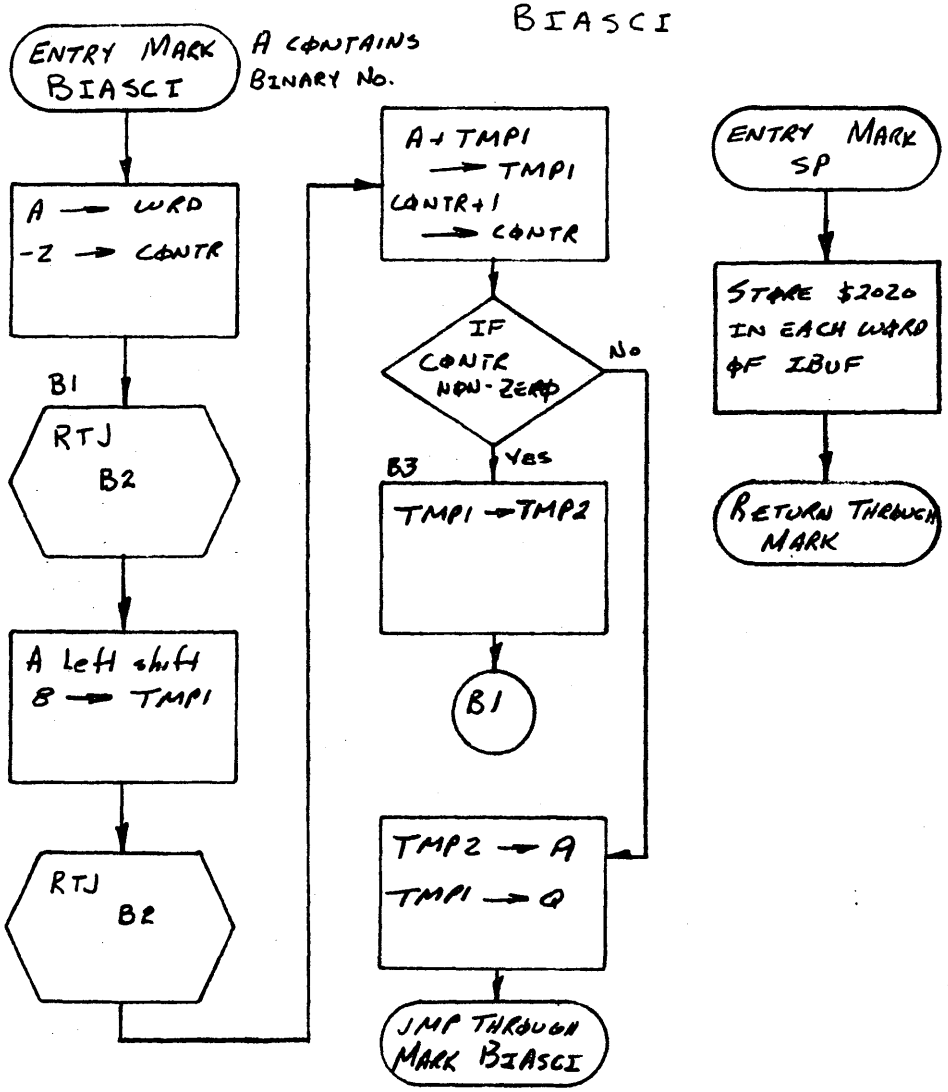
SIFT (BRKP05)

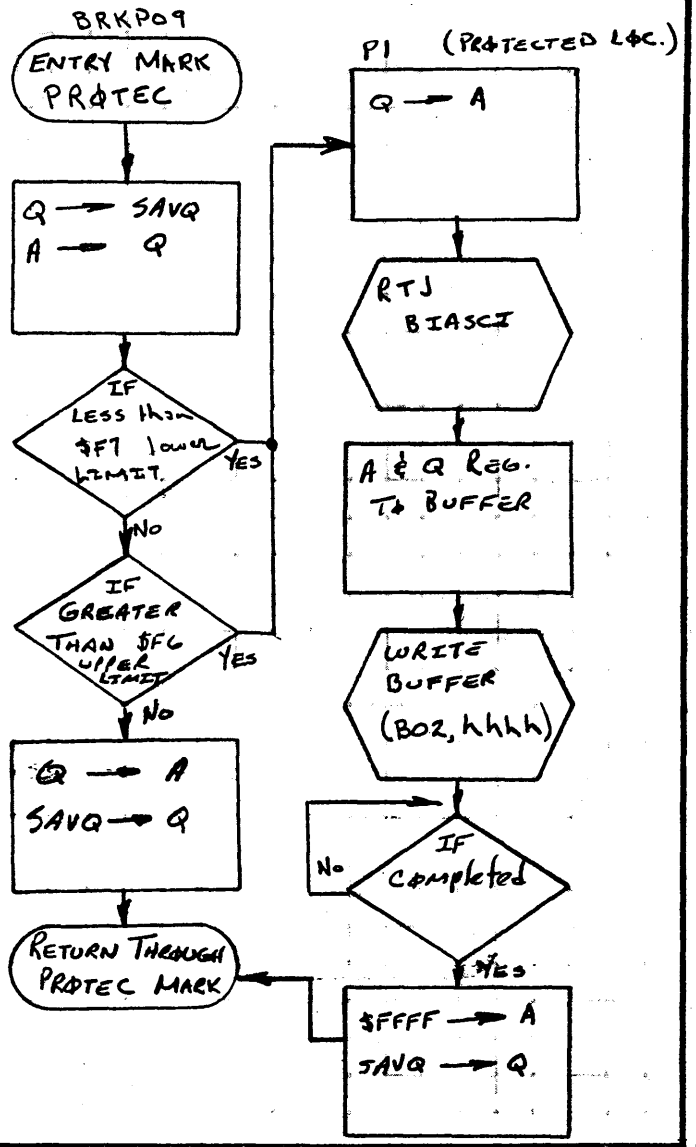


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.		
	SIFT	PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

BIASCI



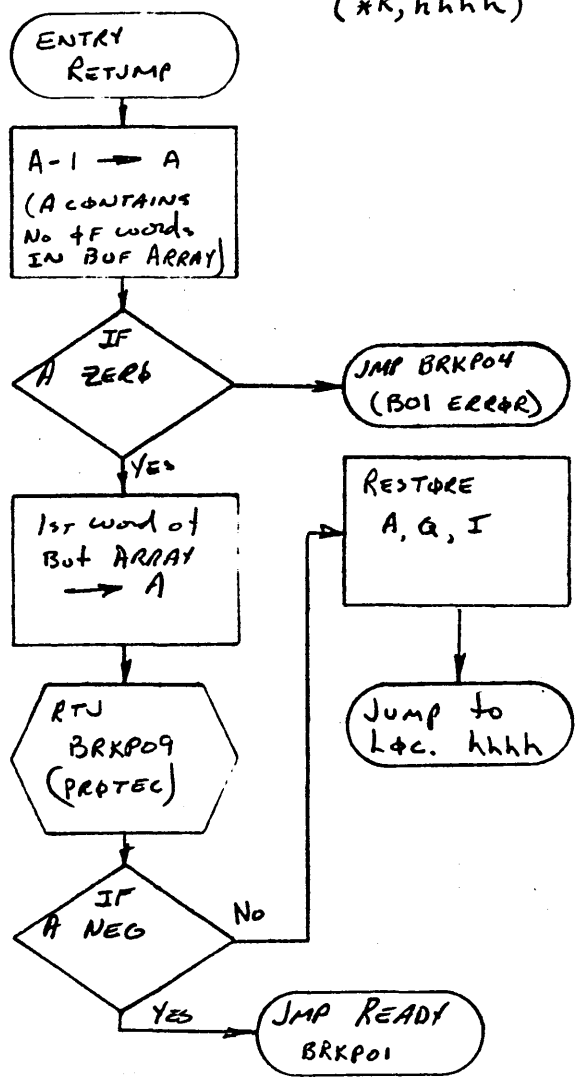


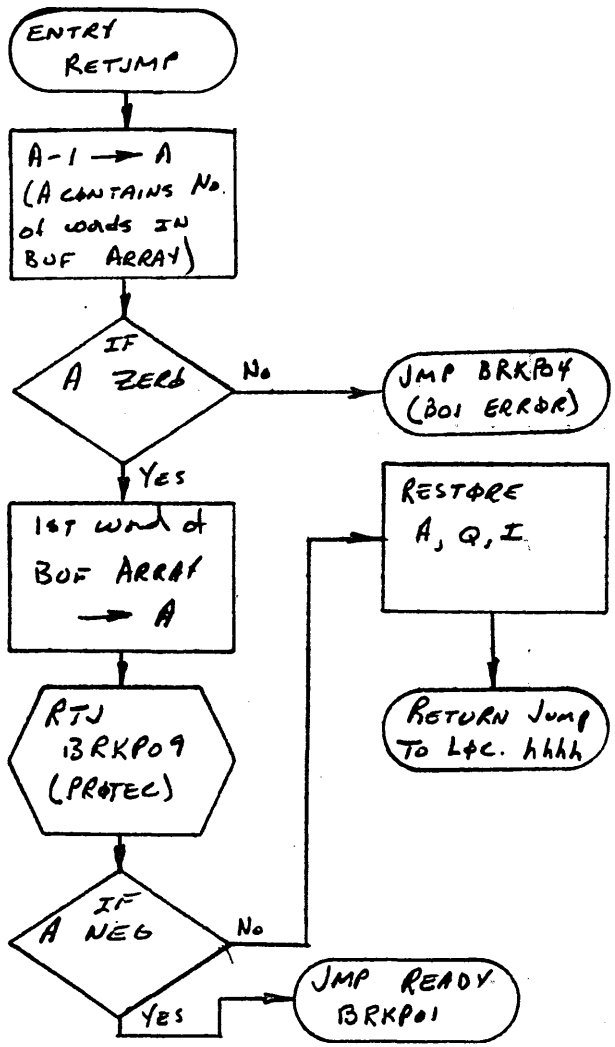
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
	BIASCI PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

37.16

AA1385 (FORMERLY CA127-1)





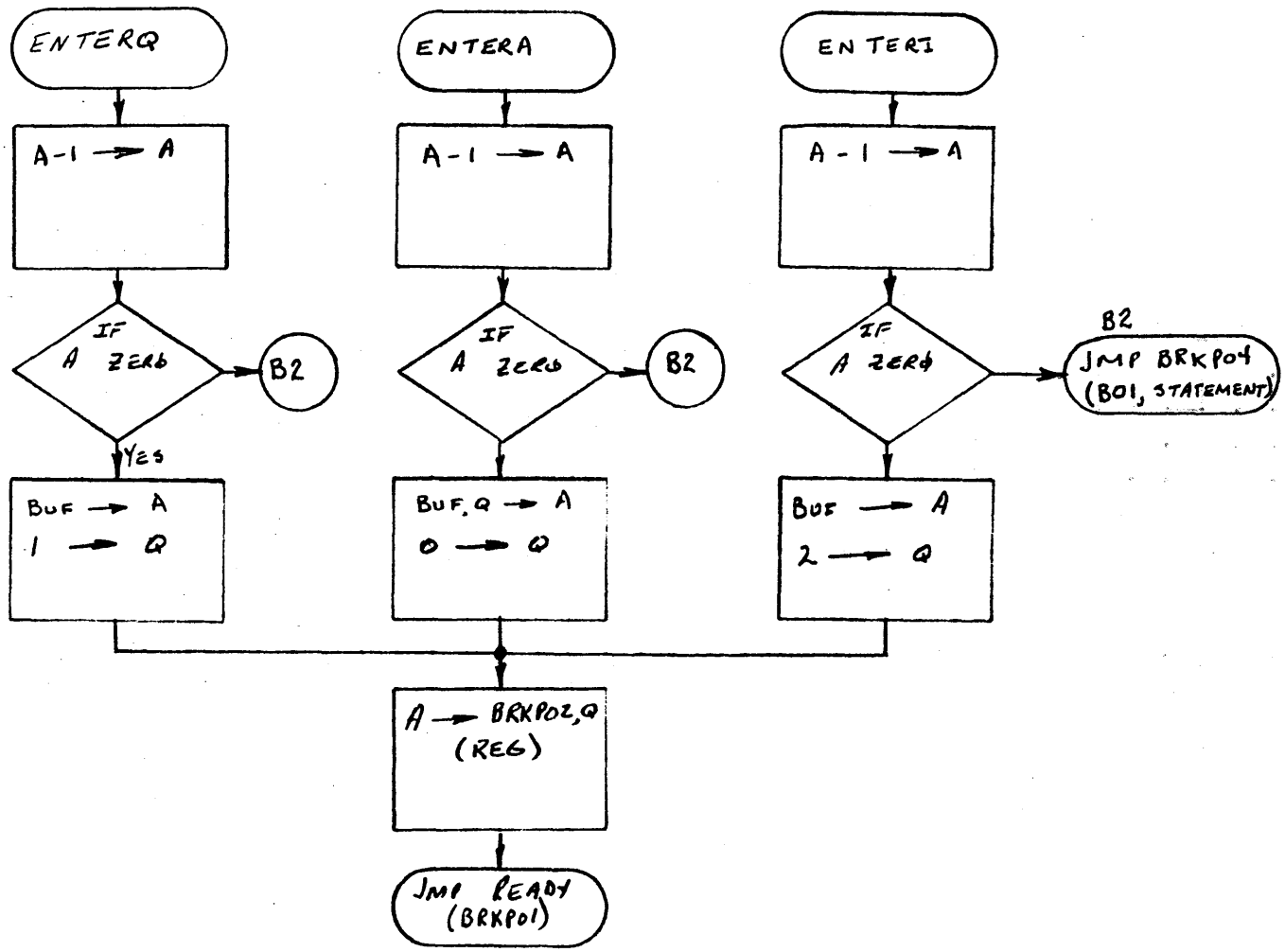
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
	RETJMP PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1

MAR 5 1971

37.17

ENTER

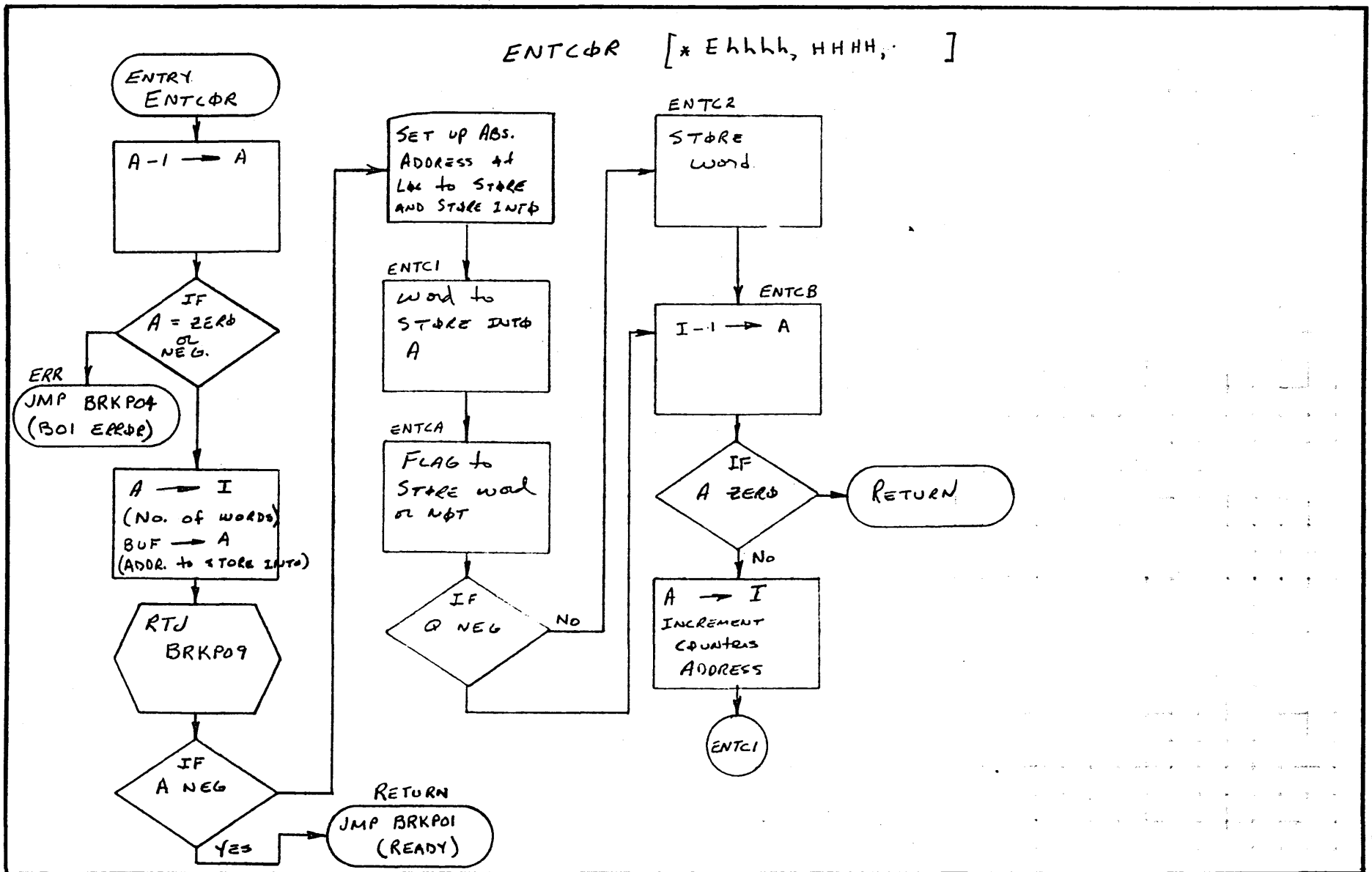


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

3718

AA1385 (FORMERLY CA127-1)

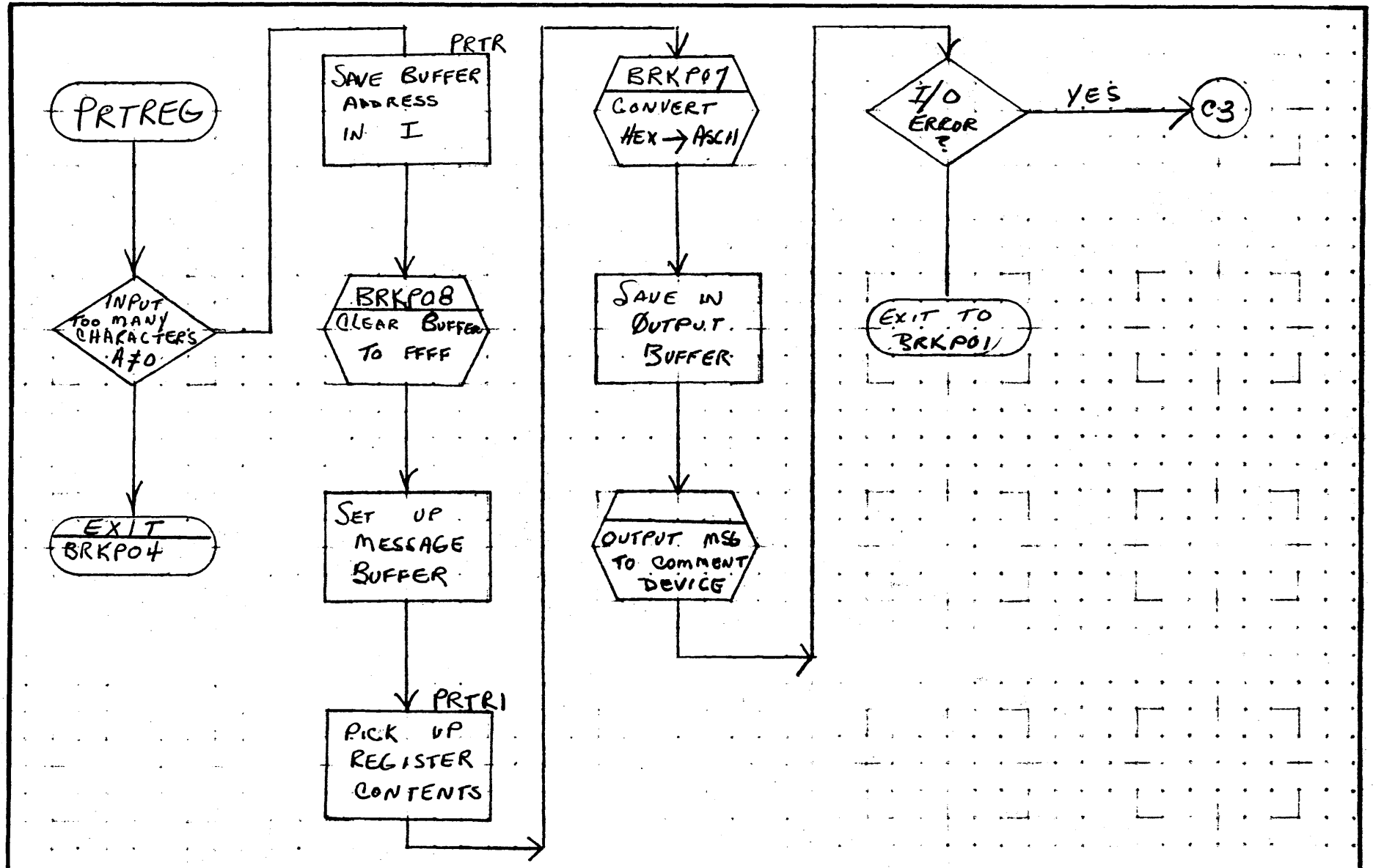


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

37.19

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
	PRTREG PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

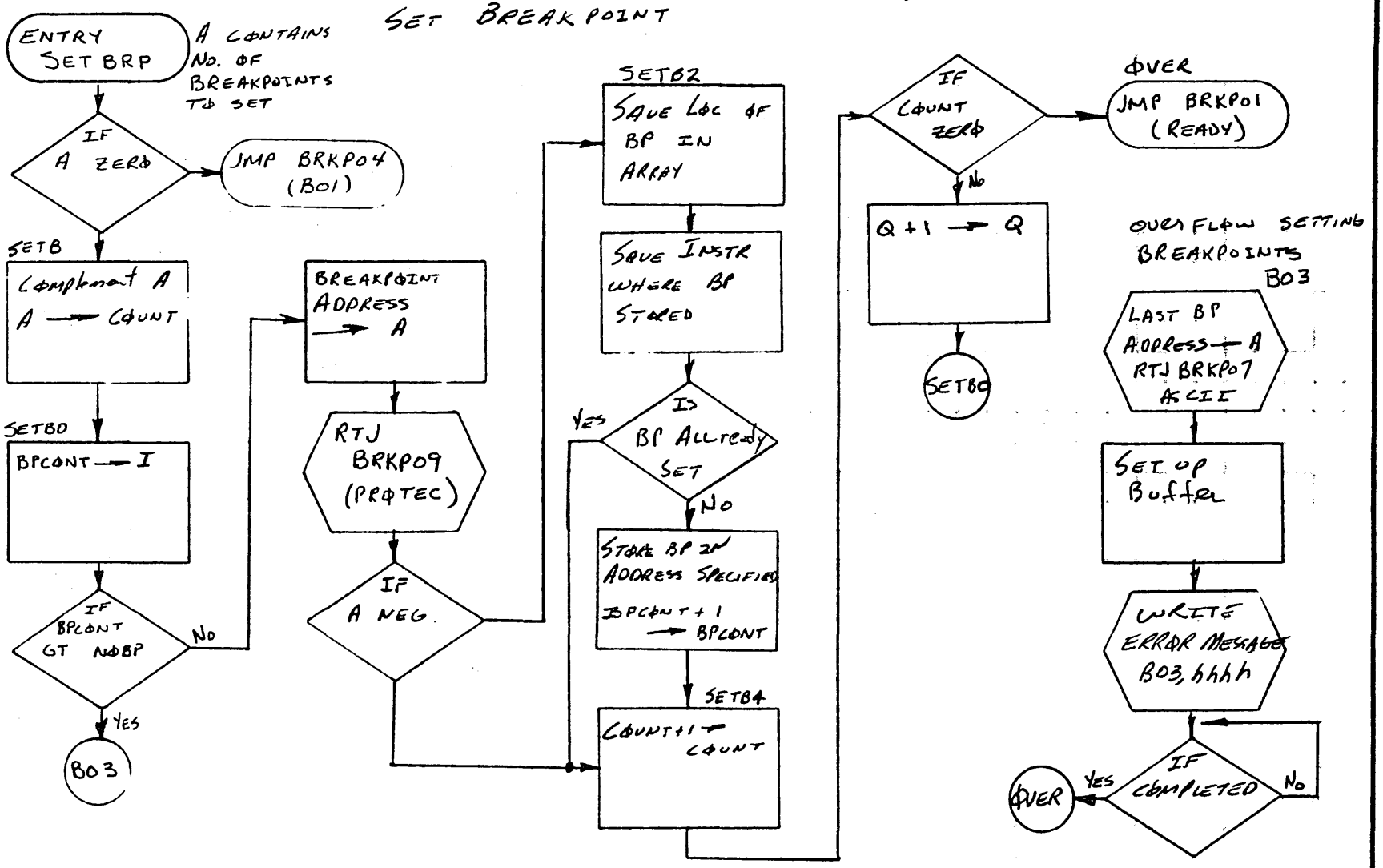
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

37.20

SETBRP (*SHHHH, HHHH, ...)
SET BREAKPOINT



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

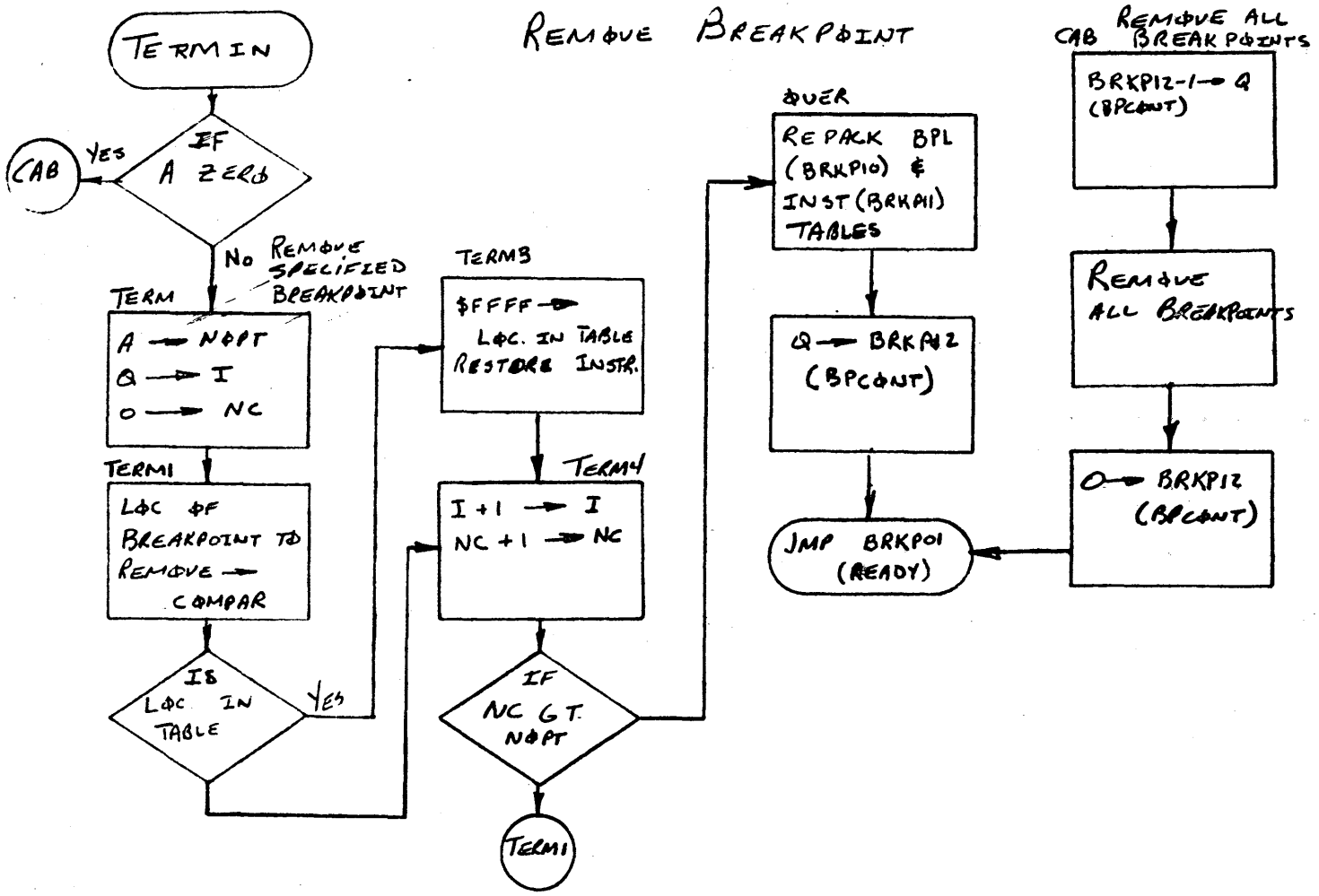
MAR 5 1971

37.21

AA1385 (FORMERLY CA127-1)

TERMIN (*T, HHHH, HHHH, ...)

REMOVE BREAKPOINT



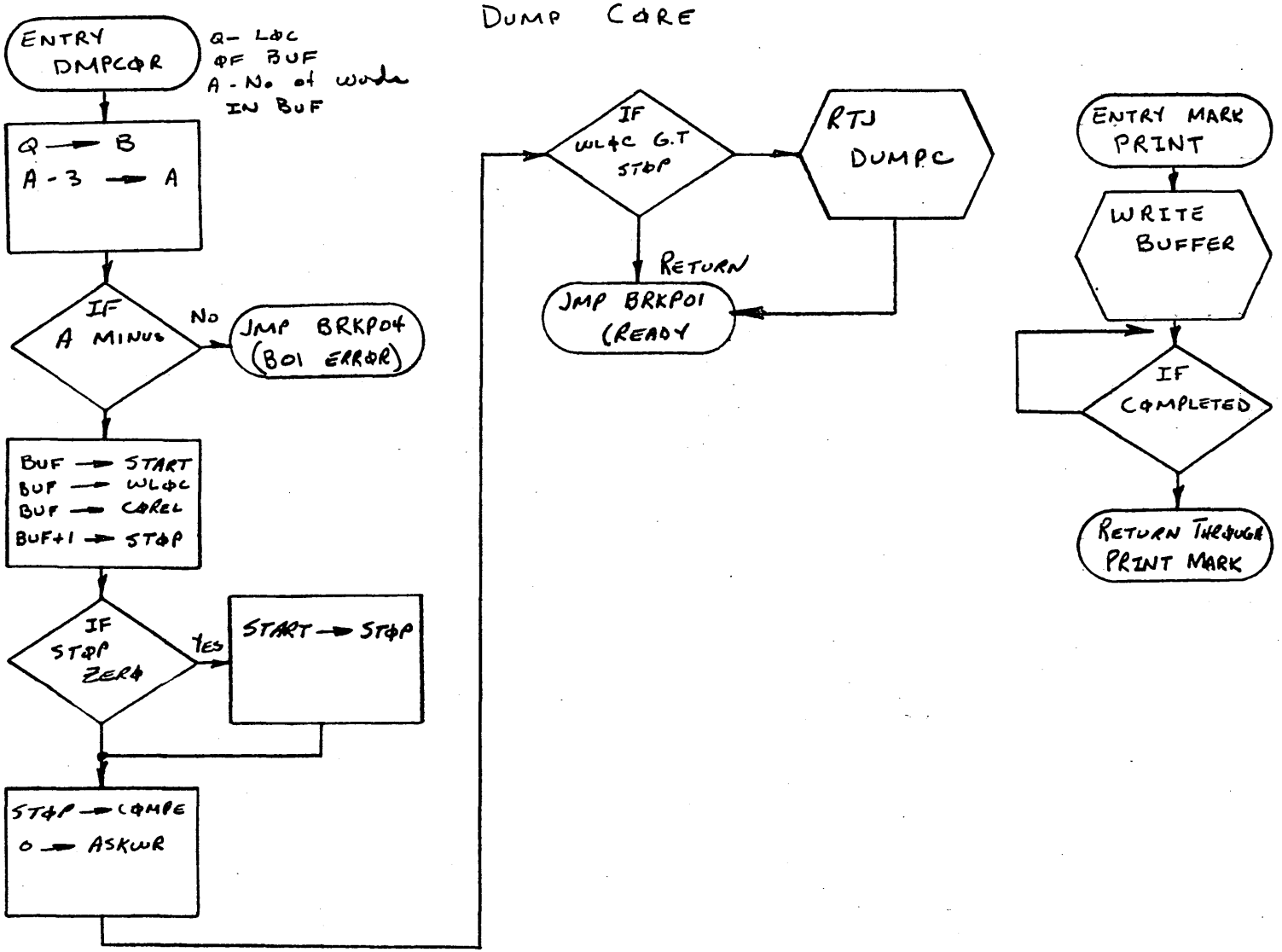
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
FLOWCHART ☐	PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

34.22

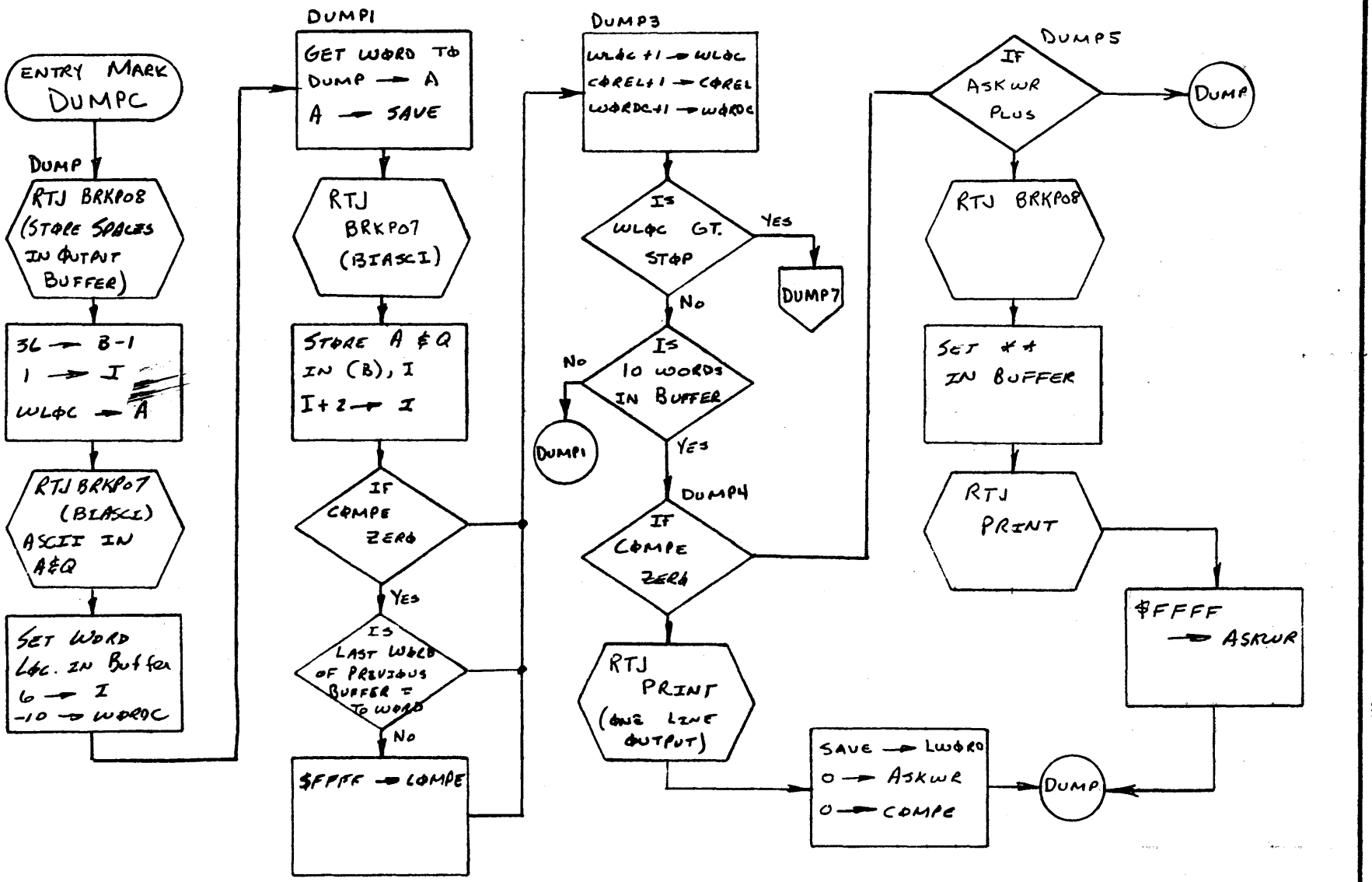
AA1385 (FORMERLY CA127-1)

DUMP CORE



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.		
	DMPCOR	PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971 37.23

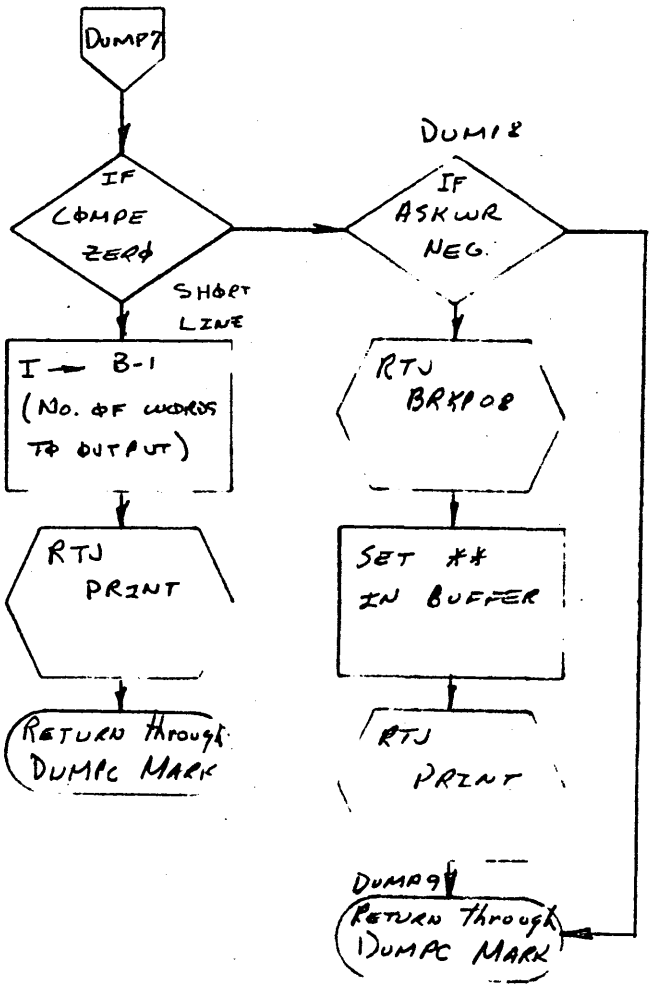


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.		
	DMPC⊄R	PAGE 2 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

37.24

AA1385 (FORMERLY CA127-1)



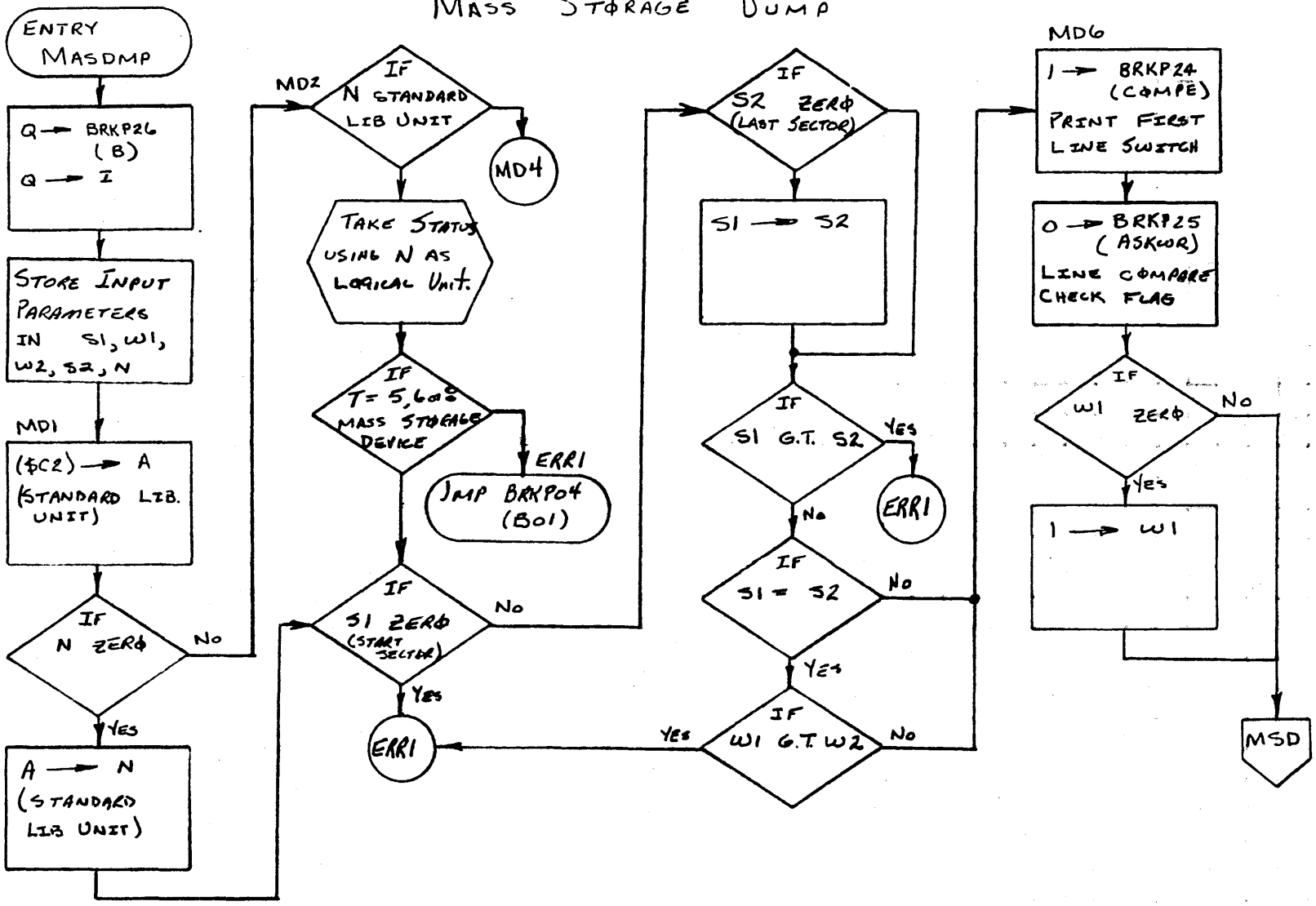
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.			
FLOWCHART ☐☐	DMPC&R	PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

37.25

AA1385 (FORMERLY CA127-1)

MASS STORAGE DUMP

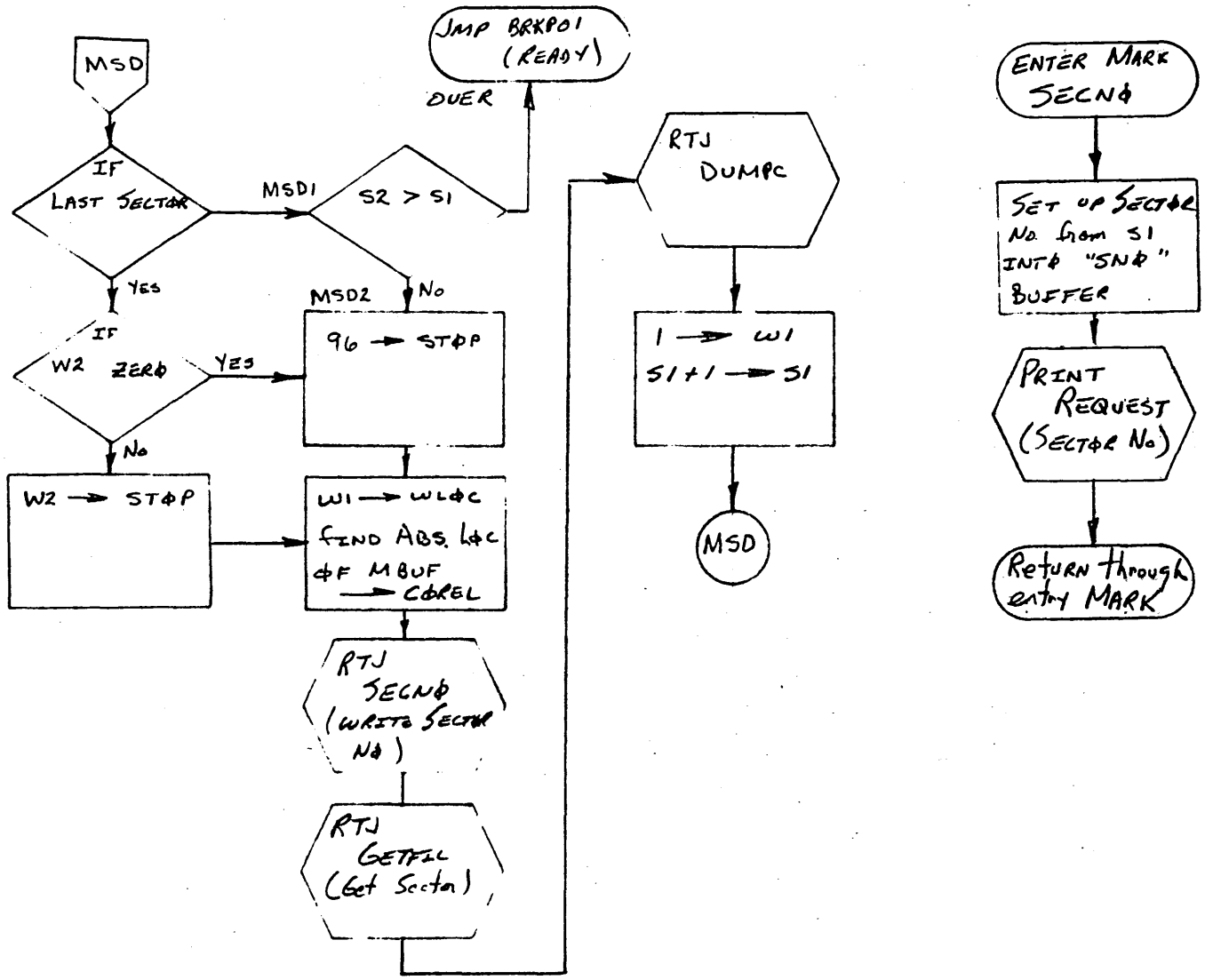


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.		
	MASDMP	PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

37.25

AA1385 (FORMERLY CA127-1)

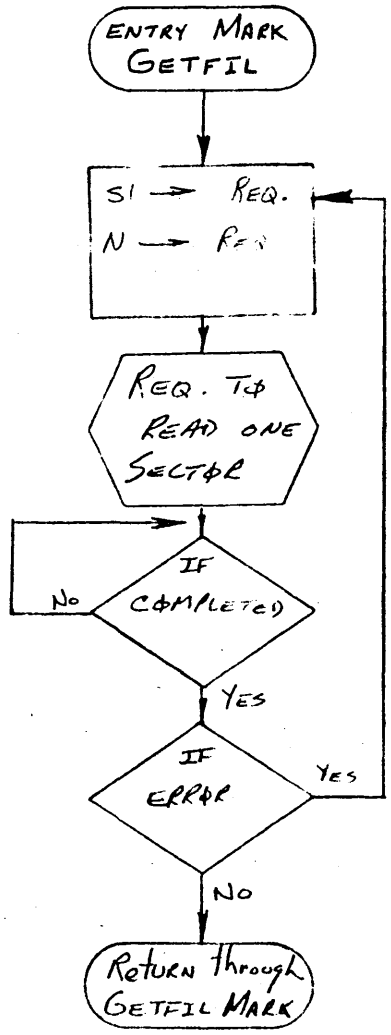


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
	MASDMP PAGE 2 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

37227

AA1385 (FORMERLY CA127-1)



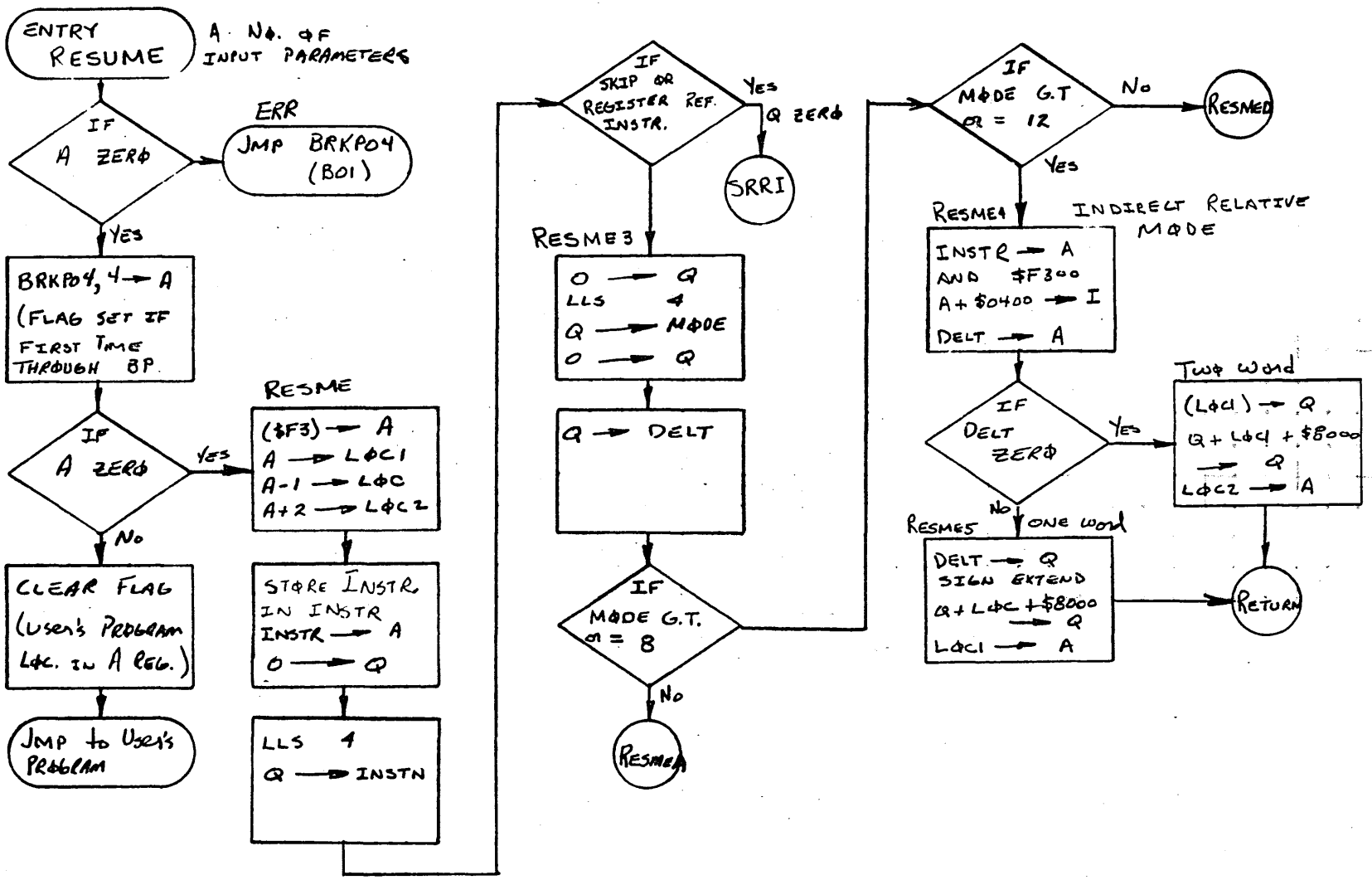
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.		
	MASDMP	PAGE 3 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

37.28

AA1385 (FORMERLY CA127-1)

RESUME (*C)

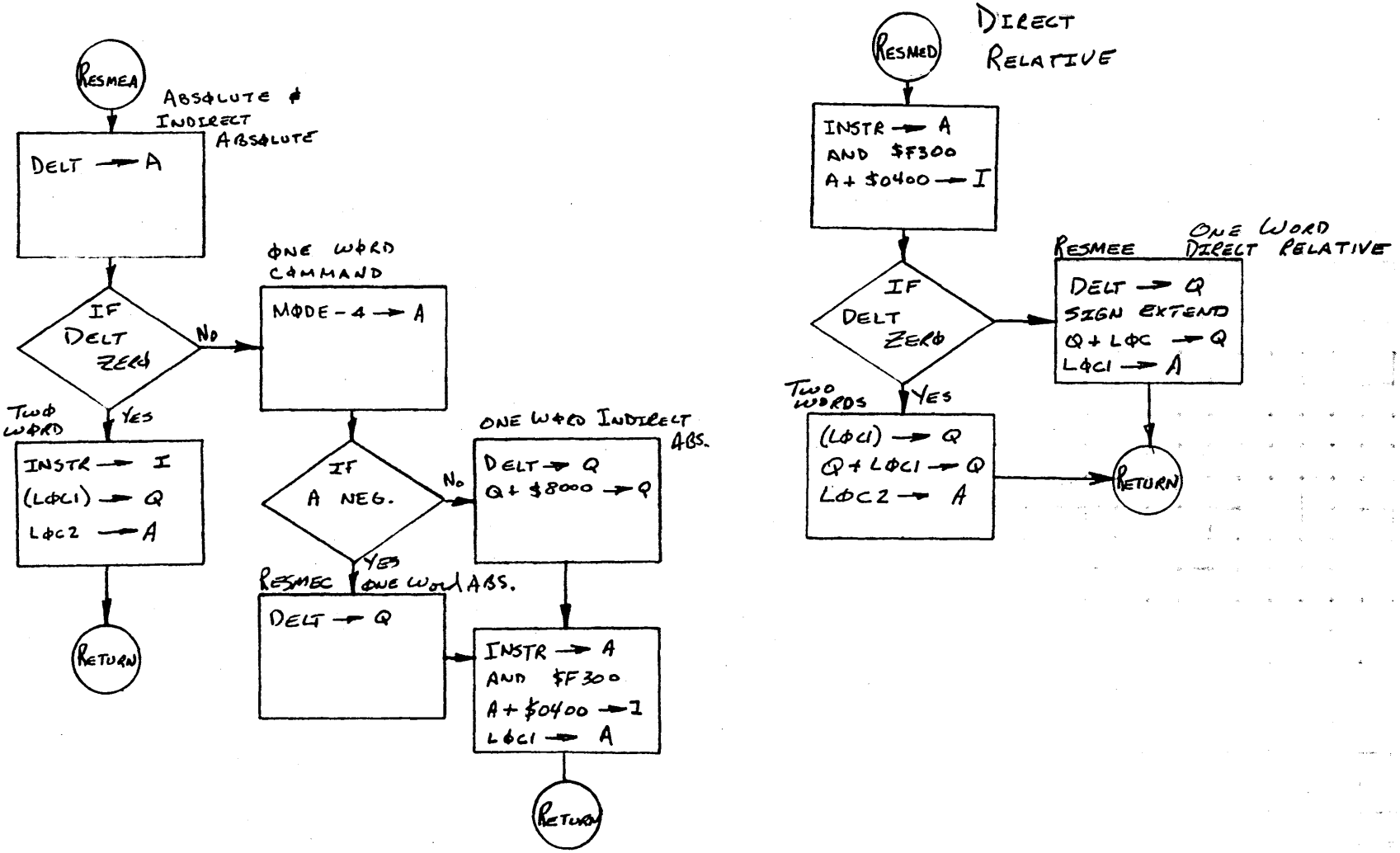


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE BREAKPOINT		PROJECT MGR		
	RESUME	PAGE 1 OF 4	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

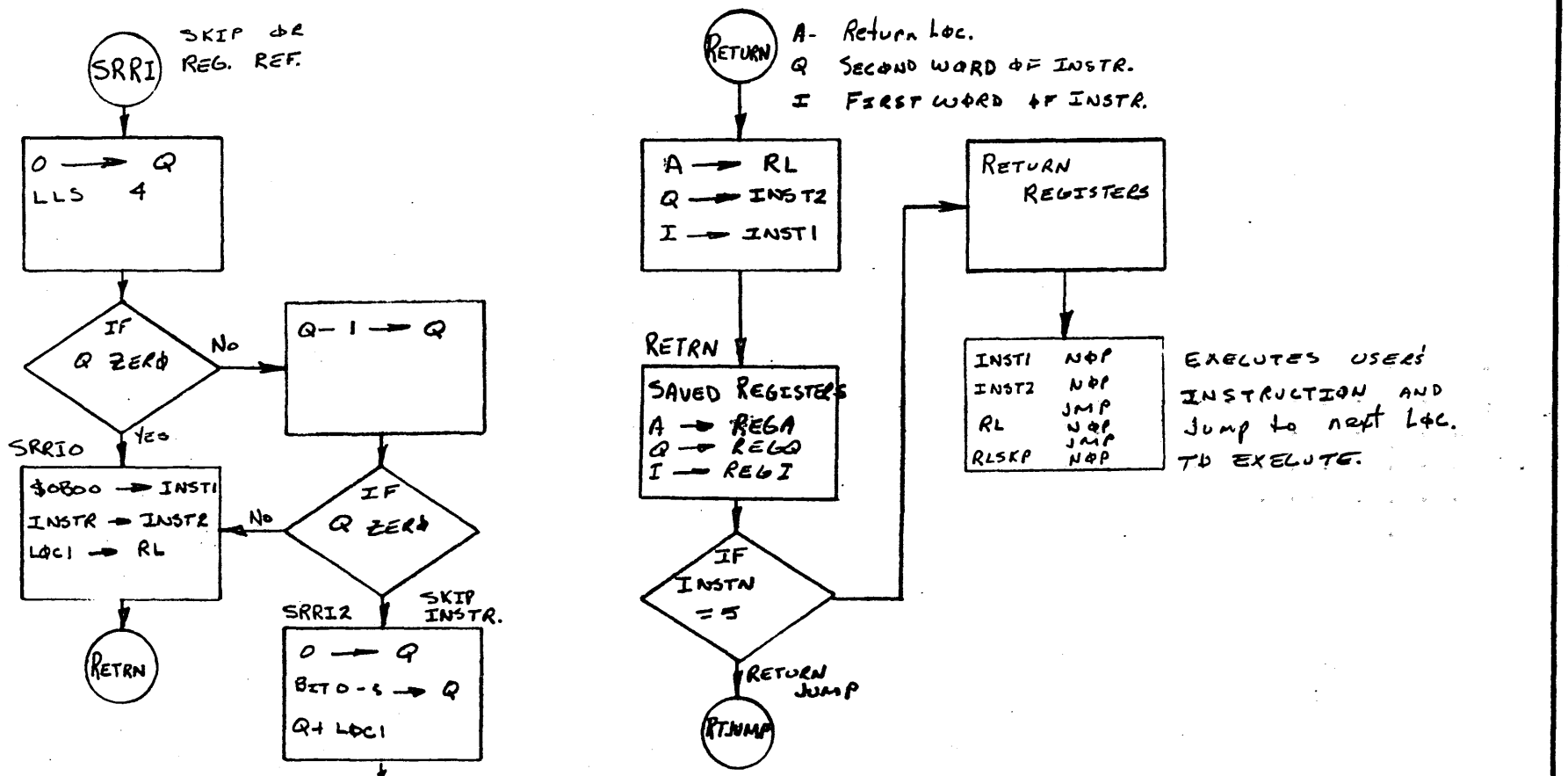
37.29

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.			
FLOWCHART ☐	RESUME	PAGE 2 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

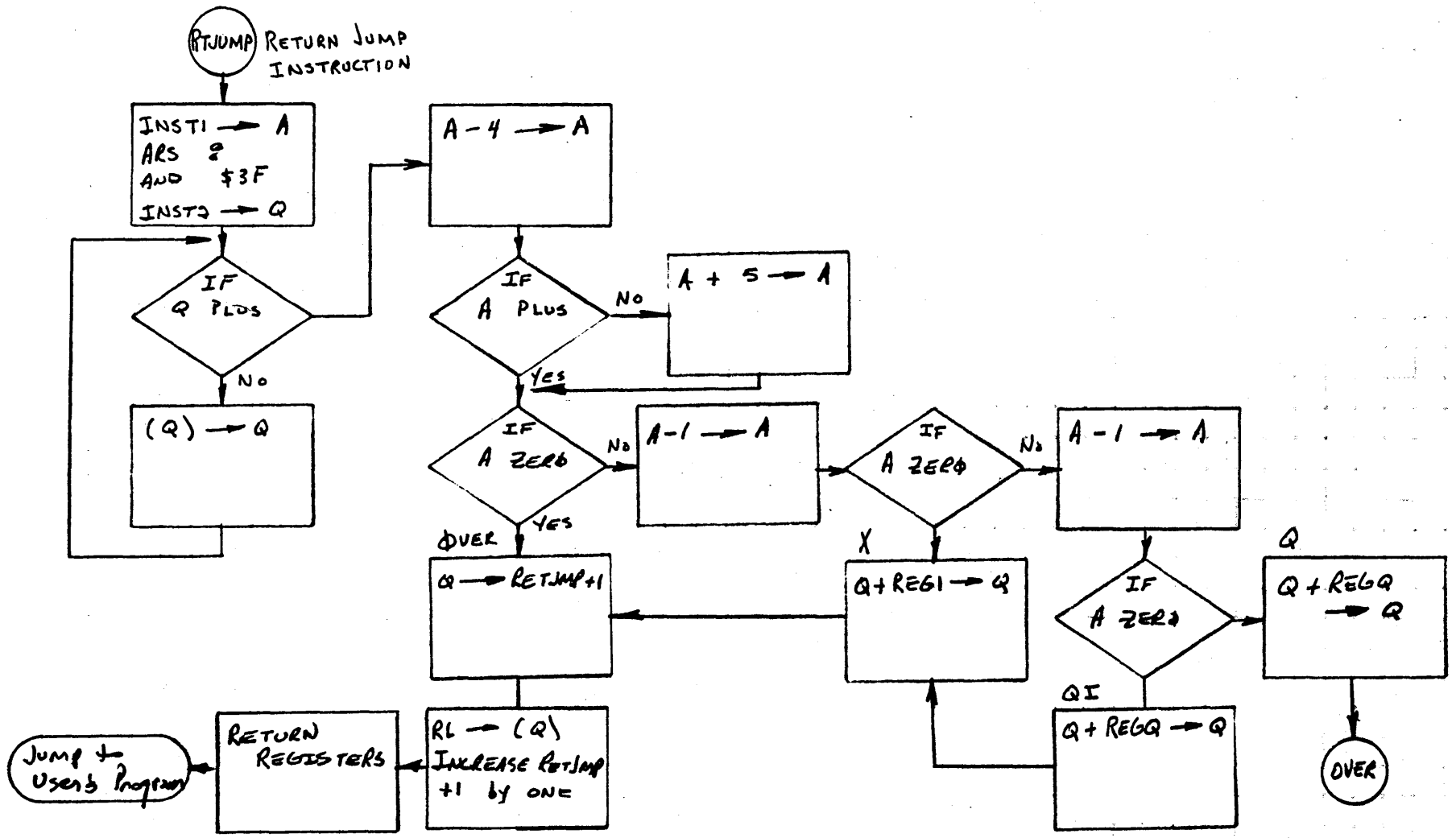


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BREAKPOINT		PROJECT MGR.			
FLOWCHART ☐	RESUME	PAGE 3 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

37.3.1



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BREAKPOINT	PROJECT MGR.			
FLOWCHART ☐	RESUME PAGE 4 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

37132

38.0 ON-LINE DEBUG PACKAGE

The On-Line Debug Package (ODP) is a mass storage resident program with the entry name ODEGBUG in the System Directory. It is initiated with a manual interrupt. The manual interrupt program schedules the Manual Input for Process (MIPRO) program which in turn schedules the on-line debug program.

ODP is assembled as one program and loaded onto mass memory by the Initializer. When the program is scheduled by MIPRO, its length in the System Directory entry is changed to the actual amount of core used.

The program consists of executive routines, processors, and subroutines. The executive routines are core resident as long as ODP is active (Area 1). The processors are dynamic and in core only when the request associated with processor is being processed (Area 2). The subroutines are also dynamic and in core only at the request of the processor requiring it (Area 3).

Figure 1 and 2 show the lay out of the program on mass memory and the program in allocatable core respectively. All communications to the executive routines by the processors and subroutines must have a relative corrective increment added because the relative position in core is not the same position on mass memory.

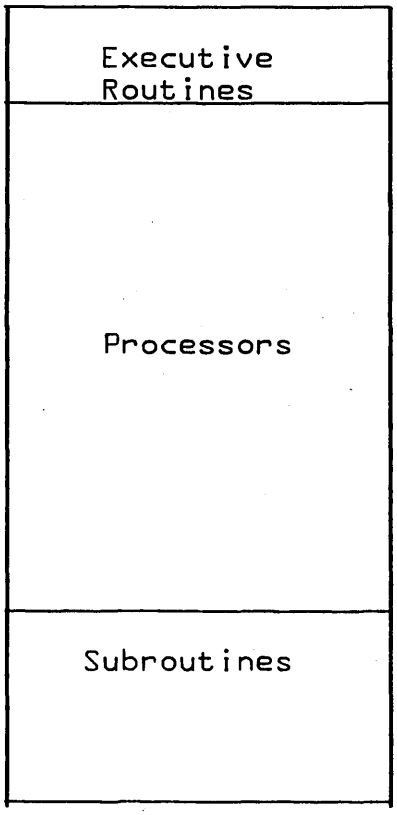


Figure 1

Mass Memory Layout

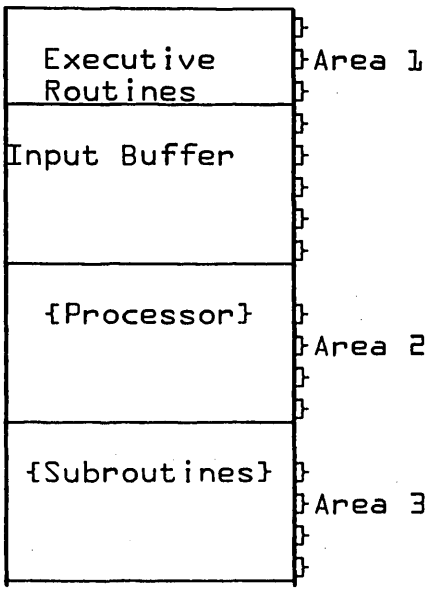


Figure 2

Allocatable Core Layout

38.1 Executive Routines {Area 1}

The following functions {with entry names} are executed by the executive routine when called by the processors or subroutines.

1. Initialize {ODP}

   Initialize program by setting in-process flag, obtaining the standard output comment logical unit and printing DEBUG IN comment.

2. Another Request {LOKMOR}

   Background input buffer and request input from input comment device. Transfer Get Request Processor to area 2.

3. Transfer Processor to Area 2 {BRIPRI}

   Read processor from mass memory and go to processor.

4. Transfer Subroutine to Area 3 {HANDLE}

   If subroutine not already in Area 3, transfer subroutine to Area 3 and go to it.

5. Recoverable I/O Errors {IOERR}

   Print message and check for ODP not-active. If active, go request more.

6. Irrecoverable I/O Errors {ERROR}

   Reset ODP in-progress flag and release core.

7. Return from Processors {SOMMOR}

   Check for ODP not-active. If active, go request more.

8. Request More {SOME}

   Print NEXT? comment and go to get another request.

9. Common Subroutine to Calculator Mass Memory

   Address of Processors and Subroutines {MMADDR}. The location of the program on mass memory is obtained from the system directory entry for ODEBUG. This location plus the relative address of the processor or subroutine results in the mass memory address.

38.2.0 Processors {Area 2}

The following processors with the exception of the Get

Request Processor are brought over into Area 2 by the Get Request Processor. The transfer is always done even though the processor may already be in Area 2. The Get Request Processor is brought over by the executive program.

38.2.1 Get Request Processor {GETREQ}

This processor is used to bring into area 2 the processor corresponding to the request desired.

A three character field is obtained. A check for a valid field terminator is made. {Valid terminators are zero, comma, and $FF}. A format incorrect message is initiated if the terminator is not valid. Otherwise, a search is done for the mnemonic. If the mnemonic is not found, an illegal request message is initiated. Otherwise, the index is used to obtain the relative address of the processor. The mass memory address is calculated and the transfer initiated by jumping to BRIPRL in the executive routine. If it is the OFF mnemonic, control is transferred to the off logic in the executive routines and no processor is initiated.

38.2.2 Load Hexadecimal Processor {LHXREQ}

This processor loads hexadecimal values into core. Fields are obtained and their values summed until a slash field terminator is obtained. The sum is the starting core location at which the load is to begin. The asterisk field terminator is illegal if the slash field terminator has not

been encountered. Following the slash, fields are obtained and assumed to be data which is stored into core. An asterisk field terminator causes the preceding field to be saved as an op code. The following field is then calculated as a relative address. If the op code is not zero, an eight bit relative address is obtained and combined with the opcode to form a one word relative instruction. More than one slash field terminator will result in a format incorrect message.

38.2.3 Dump Core Processor (DPCREQ)

This processor dumps the specified contents of core in hexadecimal. Three fields are obtained. A check for valid field terminator is made and a format incorrect message initiated if not valid. The fields are converted to hexadecimal. A check for non-negative addresses is made. Negative addresses result in a format incorrect message. If the starting address is greater than the last address, the first and last are set equal. As many calls as are required are made to the Print Line subroutine dump one line (8 cells) in the comment output device.

38.2.3.1 Search Core Processor (SCNREQ)

This processor searches core for a particular bit configuration as defined by number and mask field. Core is searched in increments as specified. An index is set to get five four character fields. Each field is checked for a valid

field terminator and converted to hexadecimal. An invalid field terminator results in a format incorrect message. Each field is checked for being blank. Blank fields are replaced with their substitute. Only the mask and increment have substitutes with meaning, the others are set to zero. After all fields have been obtained, the search is begun.

If the desired configuration is obtained, a heading is printed once, followed by the location and contents. When the search is complete, a search finished message is initiated.

38.2.3.2 Set Core Processor {SETREQ}

This processor fills the specified core with the specified value. An index is set to get three fields. Each field is checked for a valid field terminator and converted to hexadecimal. An invalid field terminator results in a format incorrect message. The core locations specified are checked for being non-negative. The pattern is then stored into core.

38.2.3.3 Move Block of Core Processor {MBCREQ}

This processor moves a block of core to another location in core. An index is set to get three fields. Each field is checked for a valid field terminator and converted to hexadecimal. A non-negative check is made on each core location specified. A check is made to determine if a backwards move is required. If so, the move is made starting with the last location rather than the first.

38.2.4 Scheduler Request Processor {SCHREQ}

This processor creates a scheduler request and makes a call to the monitor for its execution. An index is set to get three four character fields. Each field is checked for a valid field terminator and converted to hexadecimal. The scheduler call is constructed and a call made to the Monitor for its execution. Note that for the request priority field only the last digit is used even though more may be entered.

38.2.5 Search Core for Parity Error Processor {SPEREQ}

This processor searches core for a parity error condition. One field is obtained and converted to hexadecimal. A valid field terminator check is made. A blank field will be replaced by $7FFF {end of core with full compliment of memory}. A negative address check is also made. Interrupts are inhibited during each parity error check. Thus no systems parity errors are obtained. The location of each parity error is printed.

38.2.6 Clear Protect Bit Processor {CPPREQ}

This processor clears the protect bit for the cells specified. An index is set to get two fields. Each field is checked for valid field terminator and converted to hexadecimal. The core addresses are checked for being non-negative. The specified protect bits are cleared.

38.2.6.1 Set Protect Bit Processor {SPPREQ}

This processor is identical to clear Protect Bit Processor except that the bit is set.

38.2.7 Add Hexadecimal Processor {ADHREQ}

This processor adds a maximum of 8 hexadecimal numbers. An index is set to get 8 fields. Each field is checked for a valid control character and converted to hexadecimal. It is then added into the accumulation. The result is printed.

38.2.8 Subtract Hexadecimal Processor {SBHREQ}

This processor subtracts two hexadecimal numbers.

An index is set to get two fields. Each field is checked for a valid control character and converted to hexadecimal. The two results are subtracted, first field minus second field, and the result printed.

38.2.9 Generate Scratch Area Processor {GENREQ}

This processor allocates an area of allocatable core. An index is set to get two fields. Each field is checked for a valid control character and converted to hexadecimal. A validity check is made on the request priority {greater than two}. The core request is constructed and a call to the Monitor to allocate core is made. If allocation is not possible, a NO CORE AVAILABLE comment is printed. If the core allocation was made, a comment indicating its first and last location is printed.

38.2.10 Release Core Processor {RELREQ}

This processor releases core starting at the location specified. One field is obtained and converted to hexadecimal. The result is checked for being non-negative and stored in the release request. A call is then made to the Monitor to release core.

38.2.11 Dump Allocatable Core Map Processor {DACREQ}

This processor dumps a map of allocatable core. A ▽snap shot▽ of allocatable core is made with interrupts inhibited. The time that interrupts are locked-out will depend on the amount of core allocated but will normally exceed the 50 microsecond limit placed on interrupt lock-out. The snap shot executes the following logic. Each piece of allocatable core is checked for being empty or filled. If filled, the starting address and length are stored in a buffer. If empty, an asterisk flag is stored in place of the length. When all of allocatable core has been checked or the buffer is full, the snap shot is complete. A message is then constructed to print out the map. The subroutine hex to ASCII is used to convert hex to ASCII and insert it in the message. A call to the Monitor is made to initiate the print out of the map.

38.2.12 Print Thread Processor {PTHREQ}

This processor prints the location and the first two words of the first ten entries or less on a thread. An index is set to get two fields. Each field is checked for a valid

control character and converted to hexadecimal. The two fields are added to obtain the location of the top of thread. Under interrupt lock-out a ▽snap shot▽ of ten entries or less on the thread is taken by going down the thread and storing the location of the entry and the first two words in a buffer. A heading is printed and followed by a line at a time containing the entry location and its first two words. A subroutine is used to add a carriage return to each line, store the length of the message into the request parameters, and call the Monitor.

38.2.13 Magnetic Tape Motion Processor {MTRREQ}

This processor handles all tape motion requests. The logical unit field is obtained and converted to hexadecimal. It is then checked for being assigned to a magnetic tape device. If the logical unit is not properly assigned, an illegal logical unit comment is initiated. The number of files {records} field is obtained and converted to hexadecimal. It is checked for being a non-negative number. If this is an advance record request, special coding reads a 10 word record into core the number of times specified to accomplish the advance.

Otherwise, a tape motion request is initiated for the desired motion. This request is repeated until the specified number of executions have been done. A subroutine is used to check for valid field terminator and convert

from ASCII decimal to hexadecimal. A table contains the motion control codes used in the request and is indexed by the position of the request mnemonic in its table in the Get Request Processor.

38.3 Subroutines {Area 3}

The following subroutines are brought over into Area 3 by the Processors as needed. If the subroutine is already in Area 3, the transfer is not done.

38.3.1 Print Subroutine {PRINT}

This subroutine contains message skeletons used by the various processors. A table contains the relative starting location and length of the message skeleton thus allowing the skeletons to be of different lengths.

38.3.2 Get Field Subroutine {GETFLD}

This subroutine finds the next field in the input buffer that is terminated by a comma, slash, asterisk or end-of-text character. It has the following characteristics:

1. The number of characters desired is in the Q-register on entry. For hex, this is four or less, for decimal this is five plus sign or less.
2. If the field contains more than the desired number of characters, only the desired number preceding the control character will be used.
3. If the field contains less than the desired number of characters, only the actual number of characters will

be used.

4. If the field is not present {i.e., short form of a request may be used}, blanks will be returned to the caller. The control character will be set to zero.
5. If a slash control character has been encountered, blanks are returned until the condition is reset by the caller. {This permits fields to be not present preceding a slash}.
6. The control character is placed in the cell labeled FIELD in Area 1. For a hexadecimal number, characters are placed in FIELD +2,3. For a decimal number, the characters are placed in FIELD +1,2,3.

38.3.3 ASCII to Hexadecimal Conversion Subroutine {ASCHEX}

Converts the ASCII characters in FIELD +2,3 to hexadecimal. Non-hexadecimal characters cause the accumulation to be cleared.

38.3.4 Print Line Subroutine {DMPBUF}

This subroutine prints one line on the comment output device containing the location and contents of eight cells or less. On entry the Q-register contains the number of cells to be printed. A negative Q-register indicates that only the location is to be printed. Two forms of the call permit the location of the cells to be the actual core location or a specified location. In the latter case, a relative address defines the actual location. Before the call to

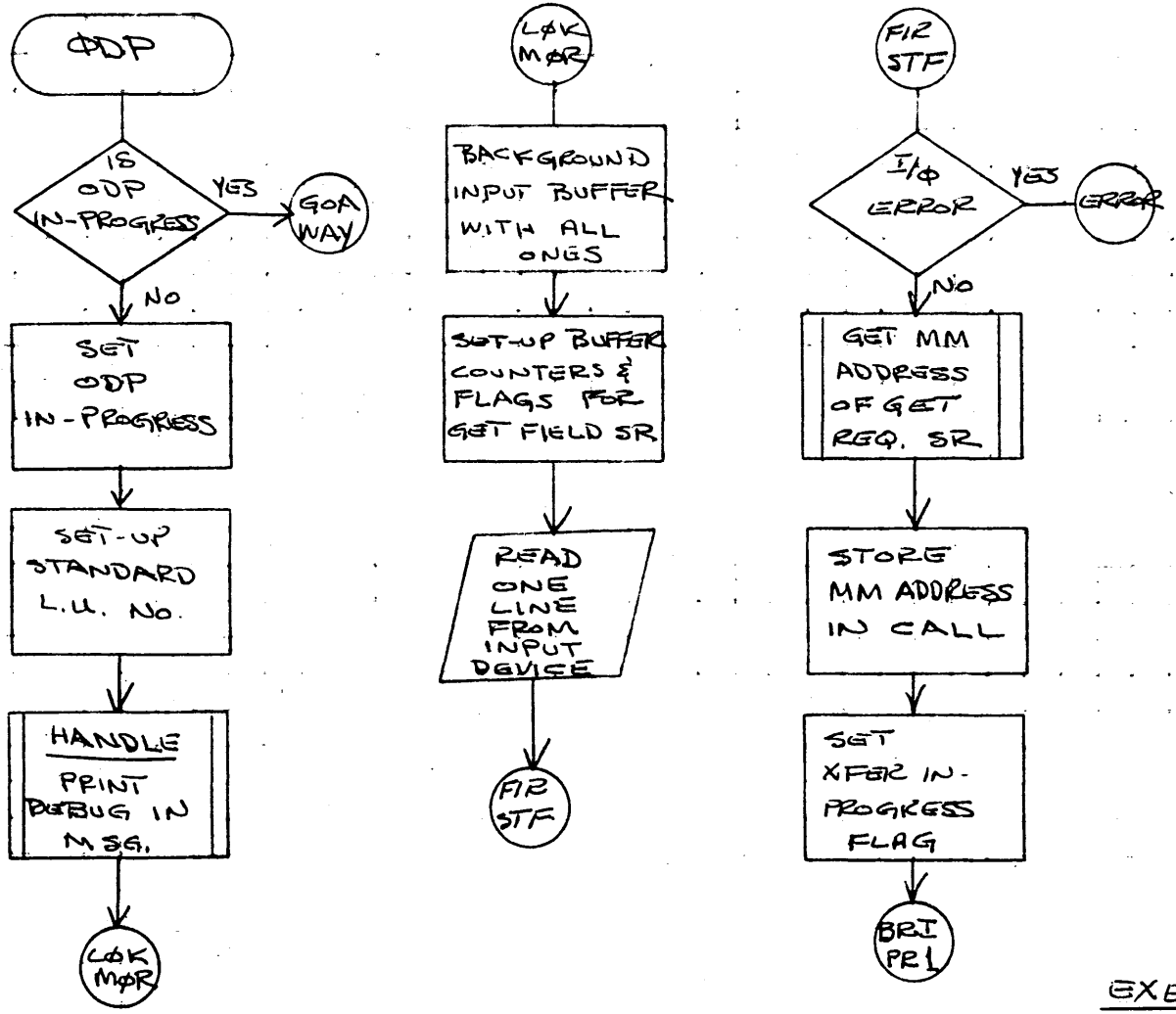
the Monitor is made, a check is made of the in-progress flag. If off, the package is turned off without printing the line.

38.3.5 ASCII Decimal to Hexadecimal Subroutine {ASCDEC}

This subroutine converts ASCII characters in locations FIELD +1,2,3 to hexadecimal via a decimal conversion. The result is return to the A-register. Legal characters are +,-,0-9. Illegal characters and plus and minus clear out the accumulation. Values greater than $2^{15}-1$ will result in an error.

38.3.6 Hexadecimal to ASCII {HEXASC}

Converts the hexadecimal number in the Q-register to two words of ASCII characters and stores them starting at the location specified by the relative address following the call. Return is made to second word following the call.



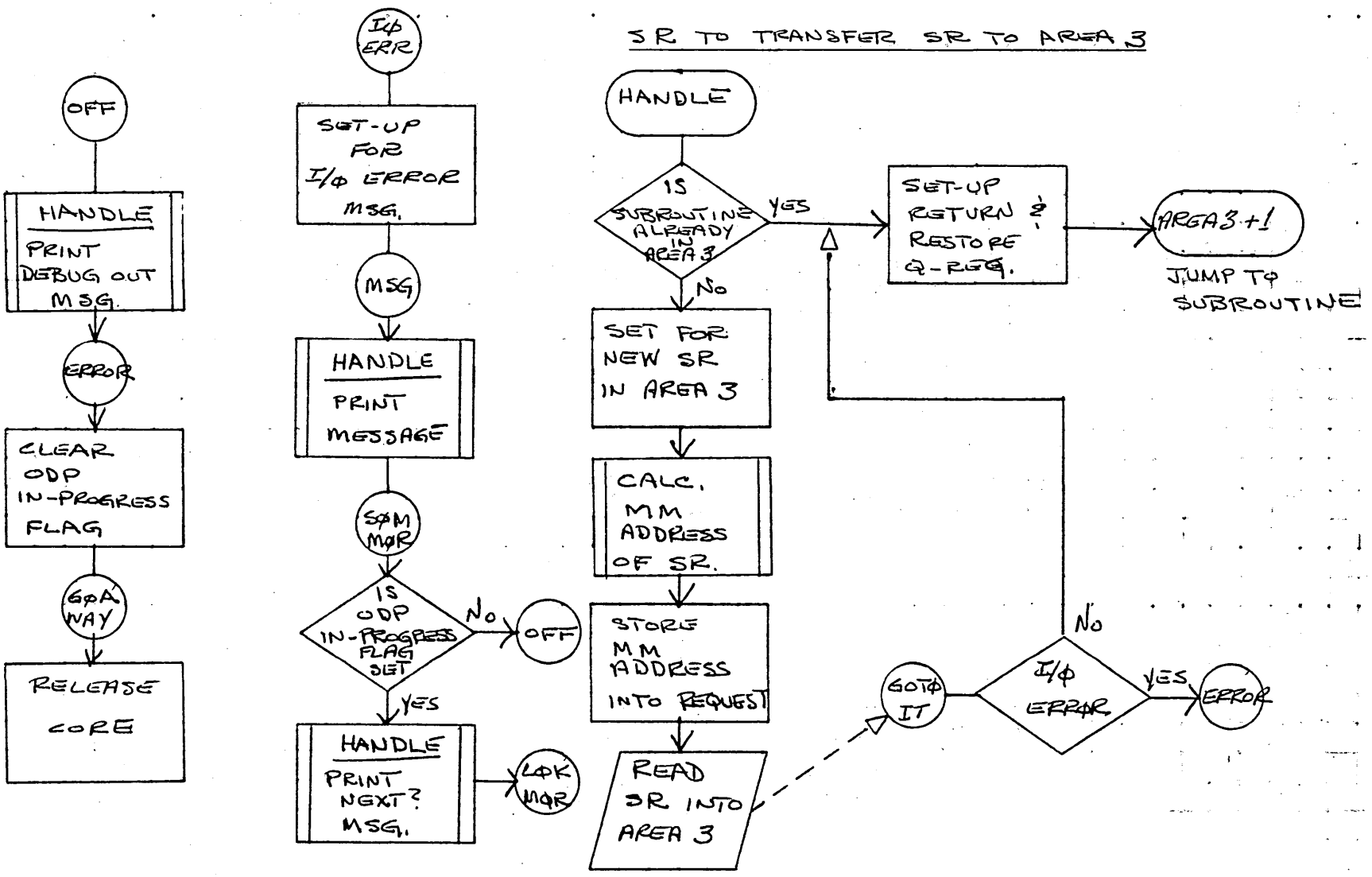
EXECUTIVE ROUTINES

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ON-LINE DEBUG PKG.		PROJECT MGR		
	PAGE 1 OF 34		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

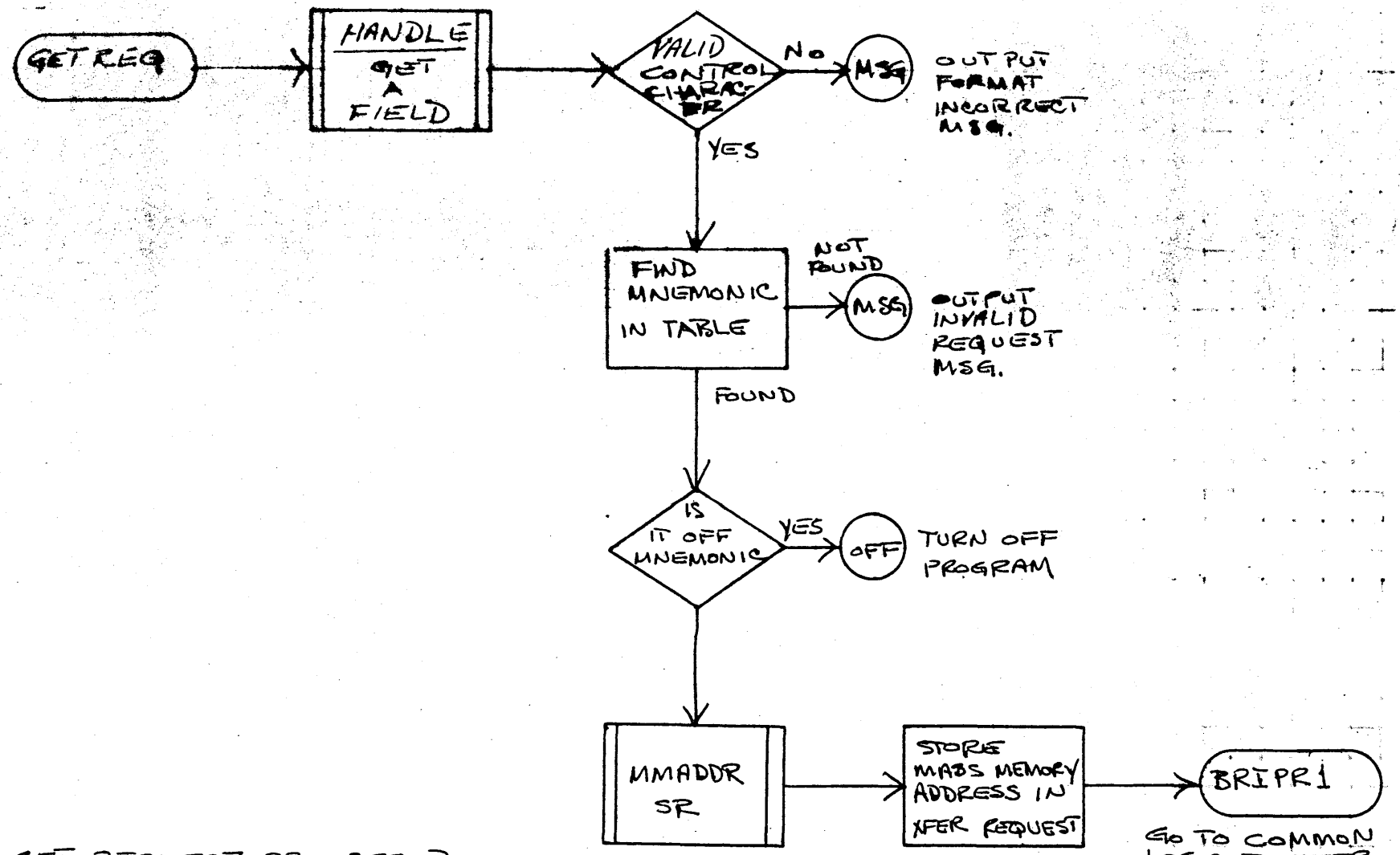
3.6.14



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

3B 715



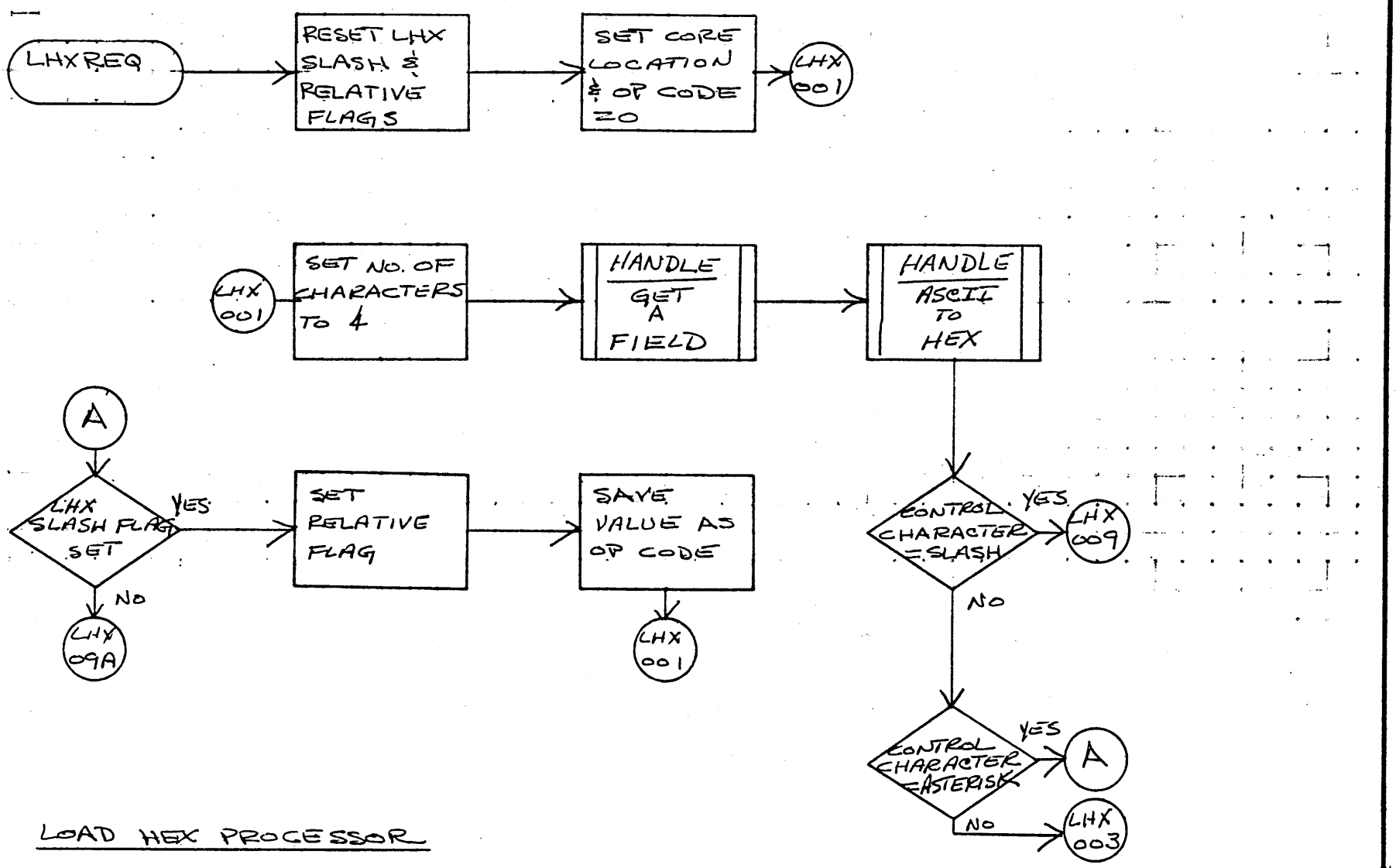
GET REQUEST PROCESSOR

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODEBUG	PROJECT MGR.			
	PAGE 3 OF 34	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORM (A127-1)

MAR 5 1971

3B.16

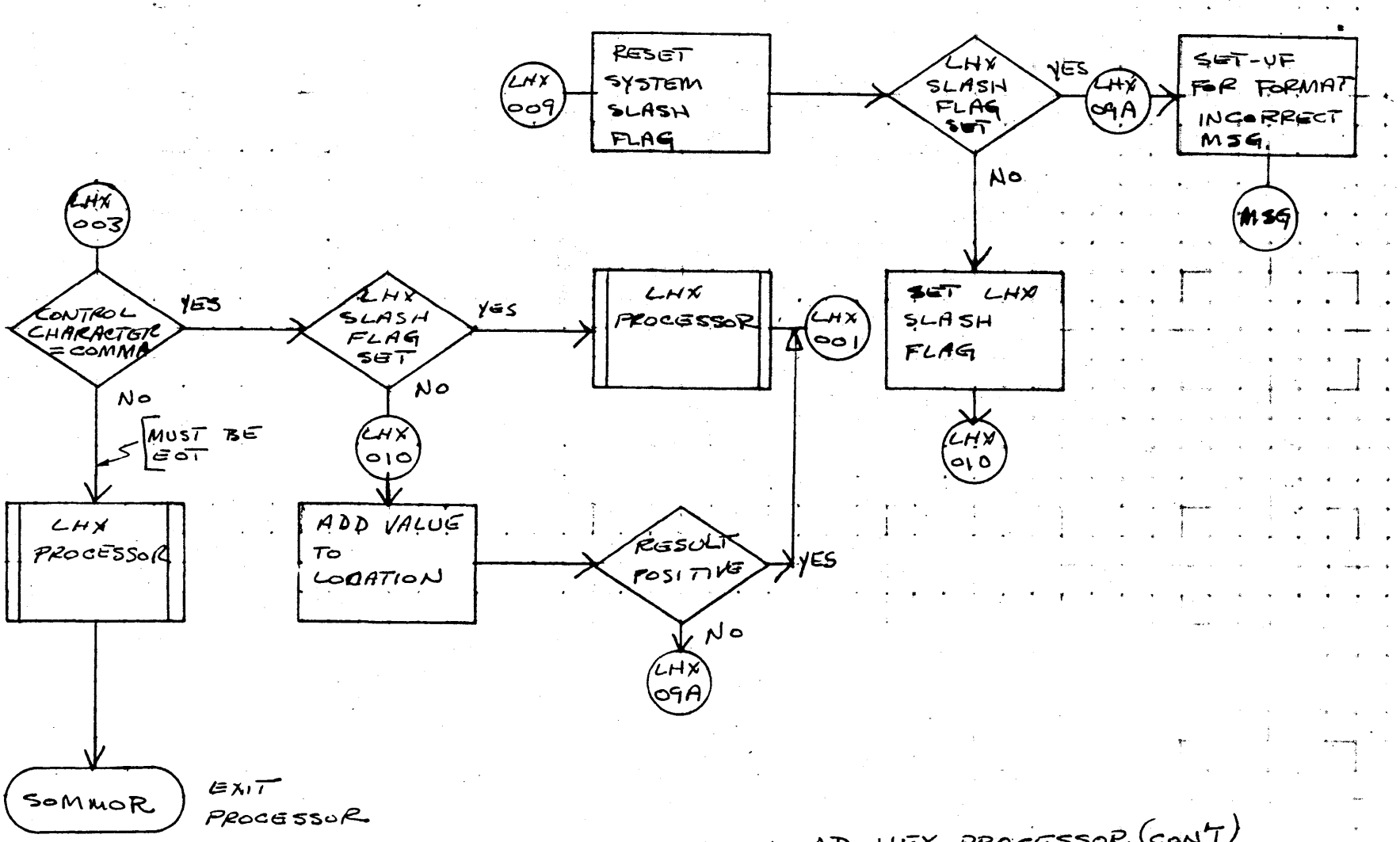


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 4 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

38.17

AA1385 (FORMERLY CA127-1)



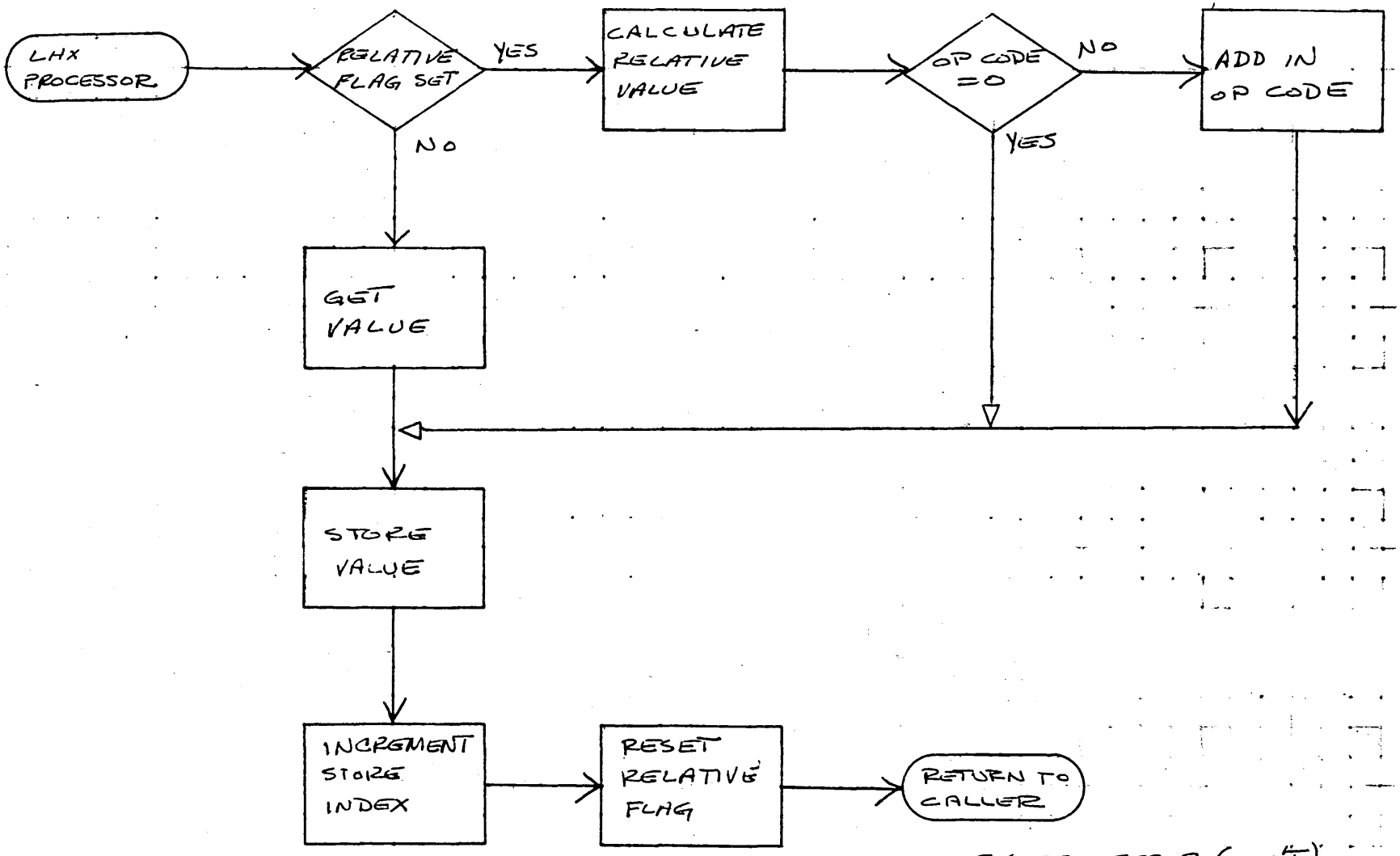
LOAD HEX PROCESSOR (CONT)

CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
FLOWCHART ☐		PAGE 5 OF 34	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

36.18



LOAD HEX PROCESSOR (CON'T)

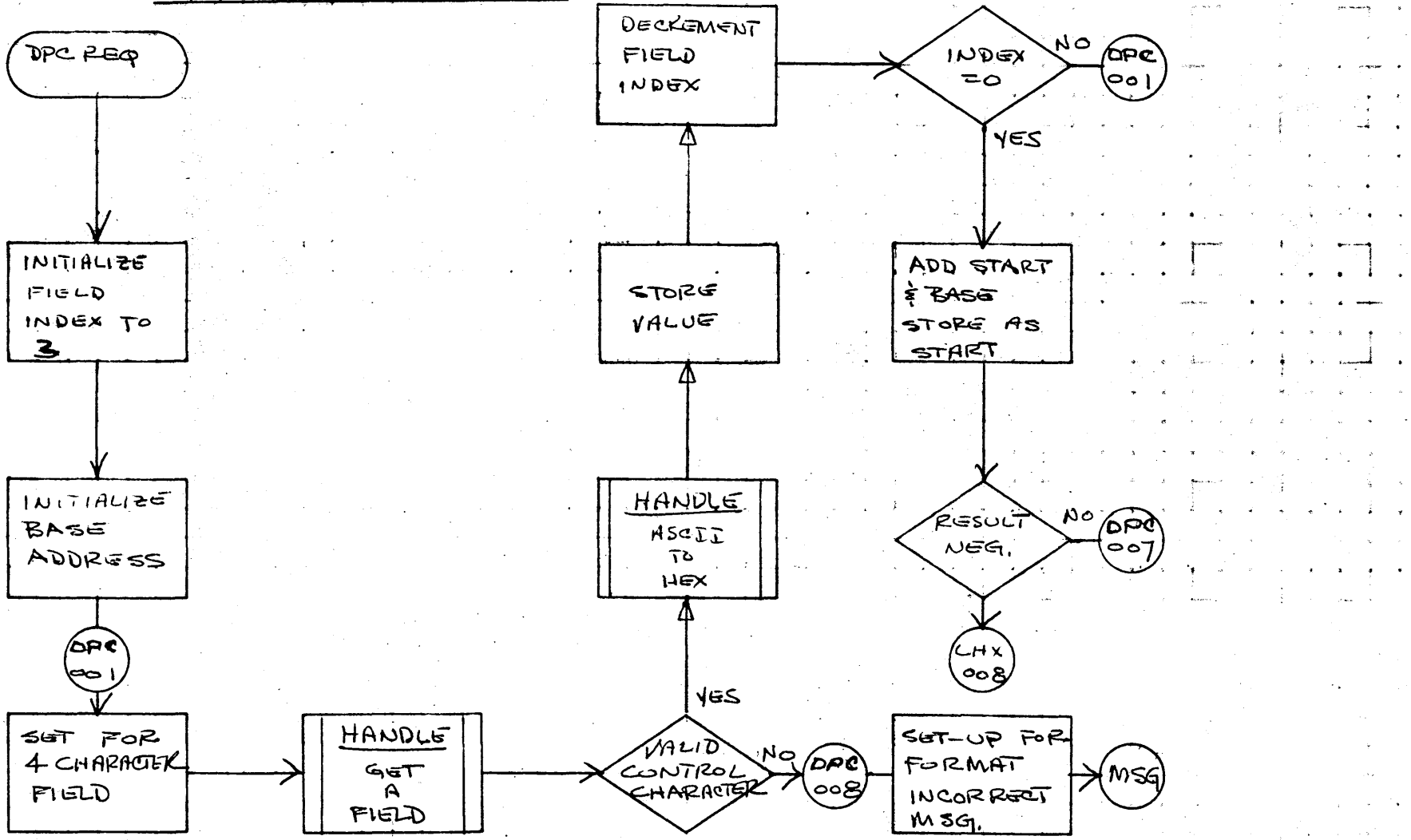
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 6 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.19

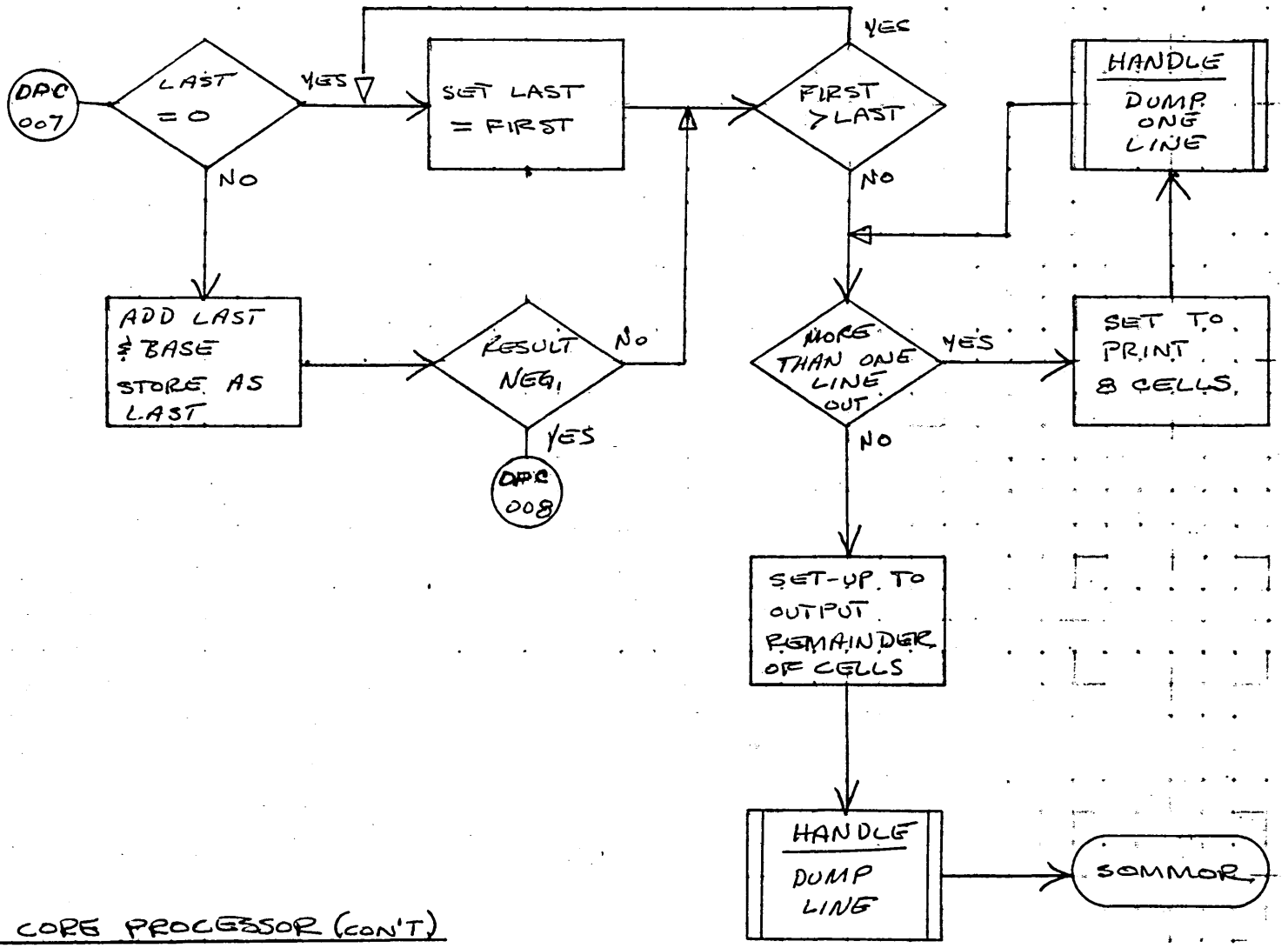
DUMP CORE PROCESSOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
		PAGE 7 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971



DUMP CORE PROCESSOR (CON'T)

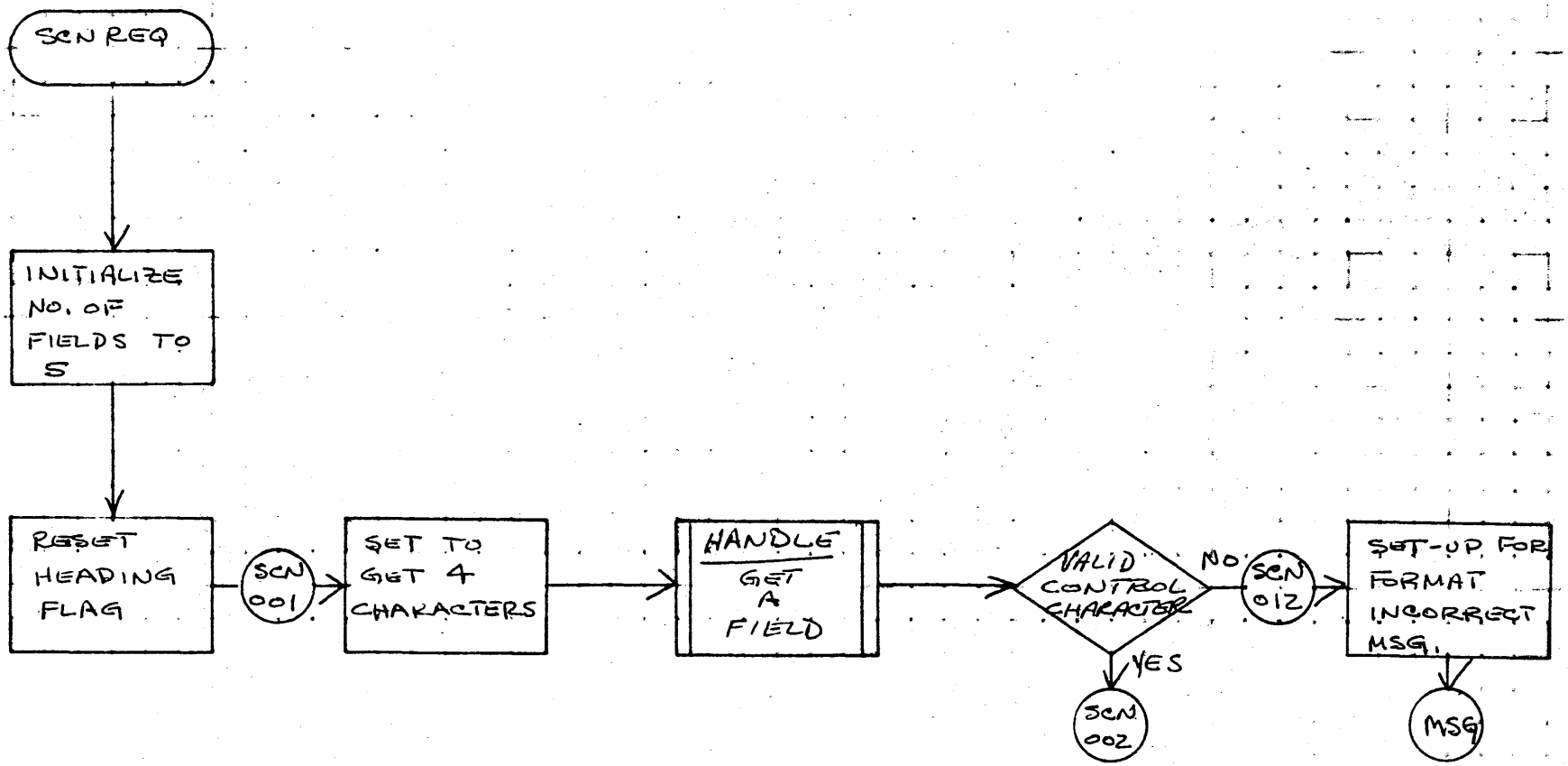
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODER16	PROJECT MGR			
	PAGE 8 OF 34	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127 1)

MAR 5 1971

38-21

SEARCH FOR NUMBER PROCESSOR

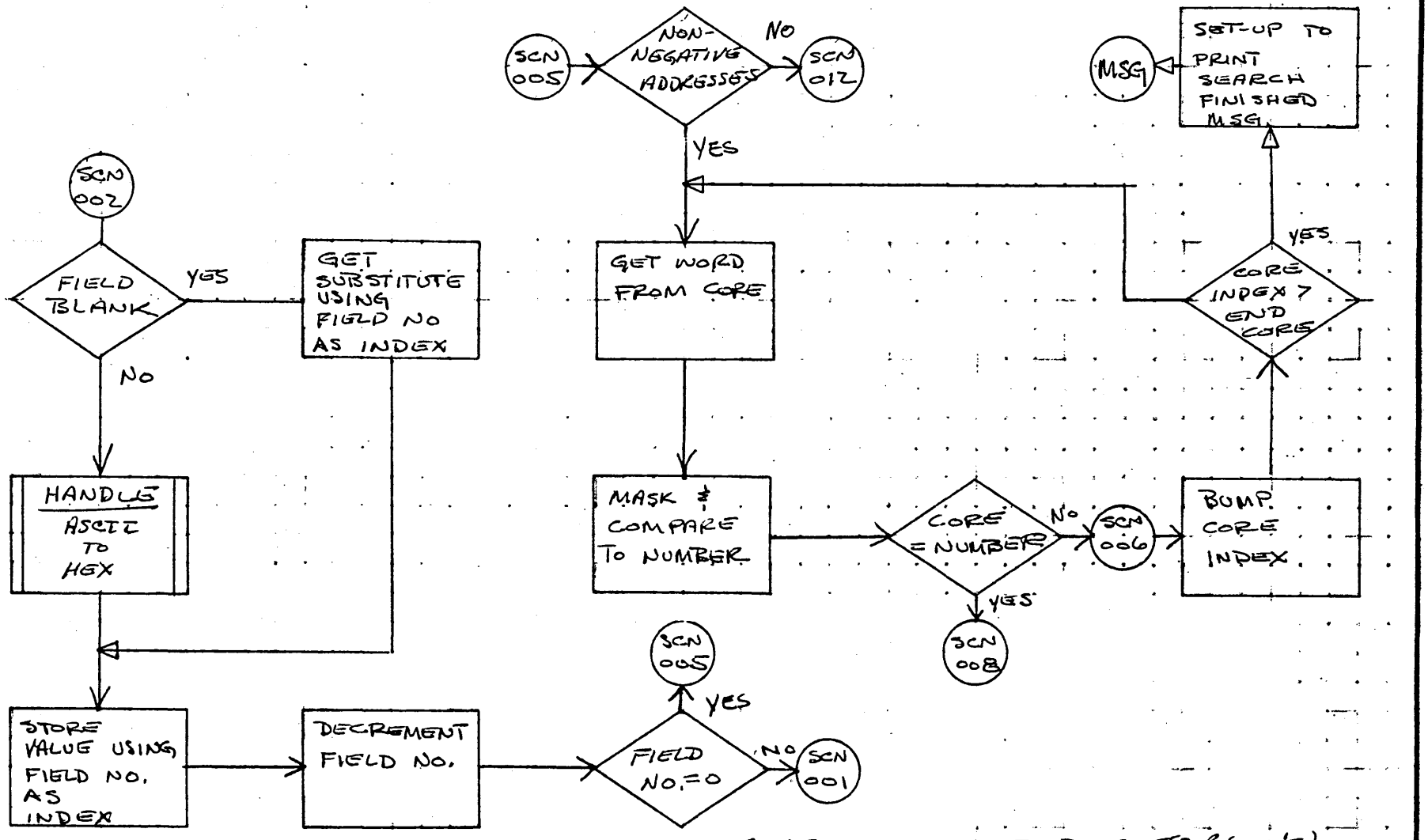


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODEBUG	PROJECT MGR.			
	PAGE 9 OF 34	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.22



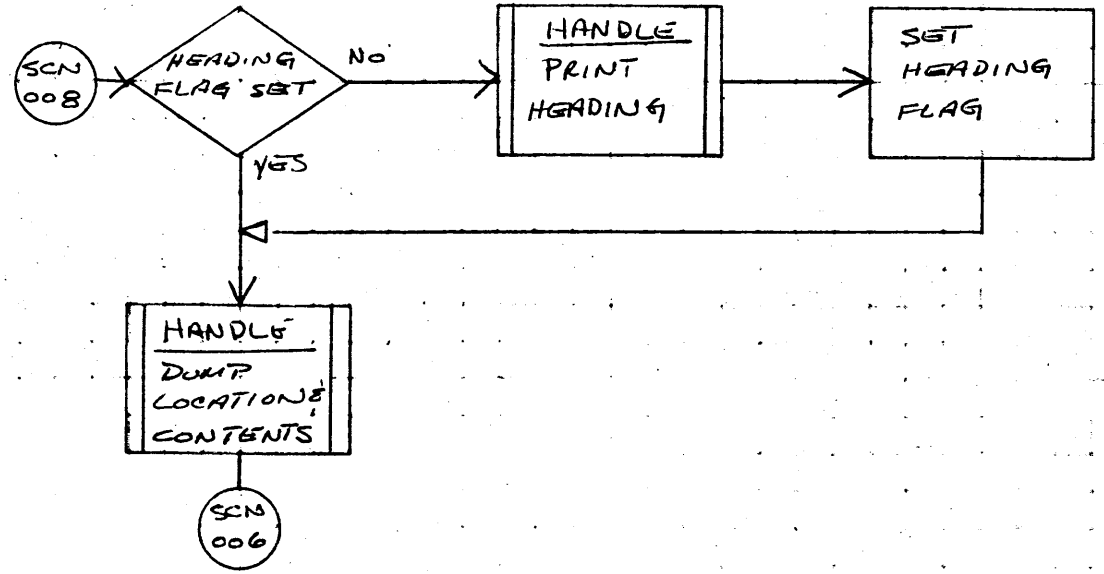
SEARCH FOR NUMBER PROCESSOR (CON'T)

CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
FLOWCHART ☐		PAGE 10 OF 34	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

E2.33



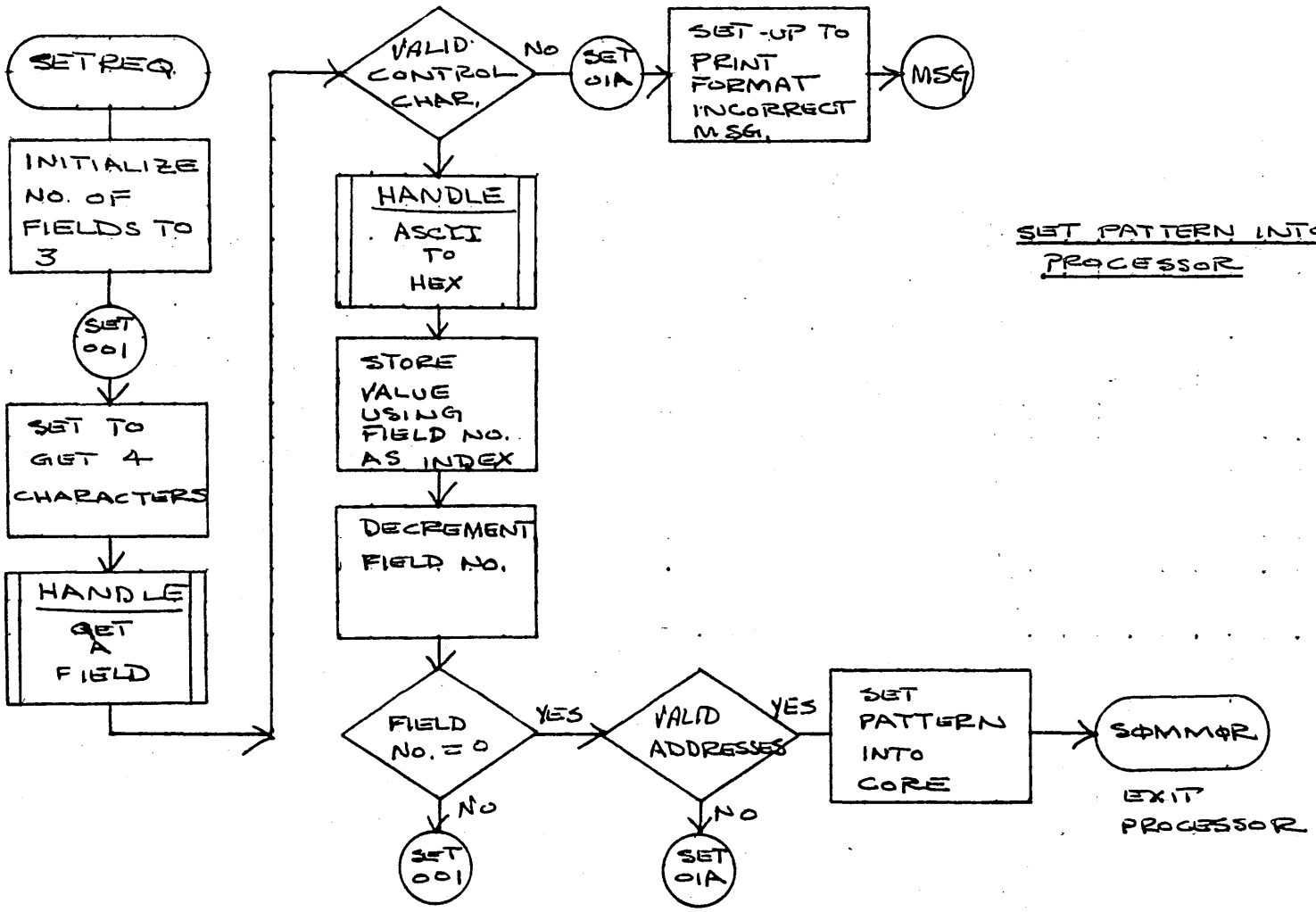
SEARCH FOR NUMBER PROCESSOR (CONT)

CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODEBUG	PROJECT MGR.			
	PAGE 11 OF 34	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.24

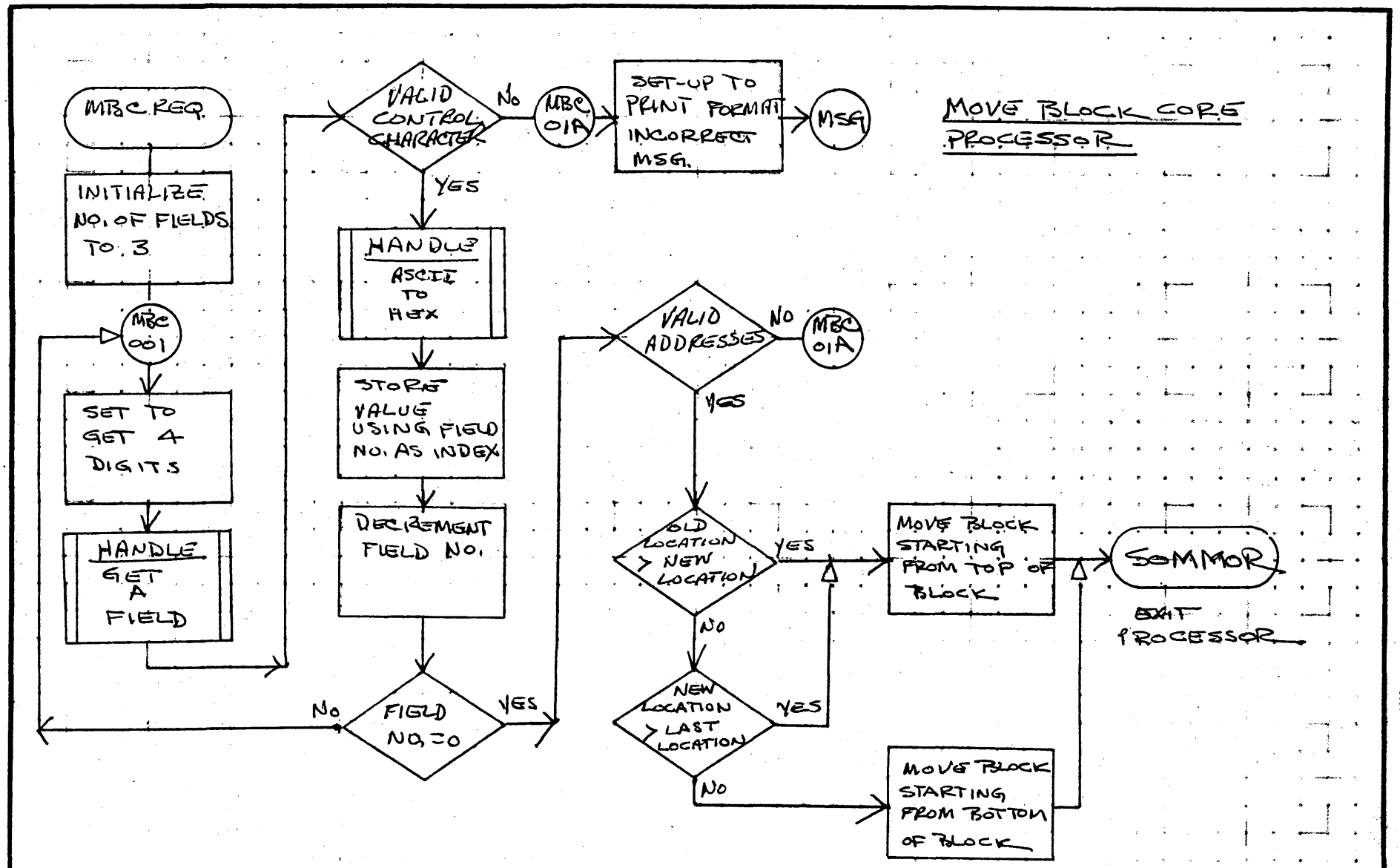


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 12 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127

MAR 5 1971

38.25

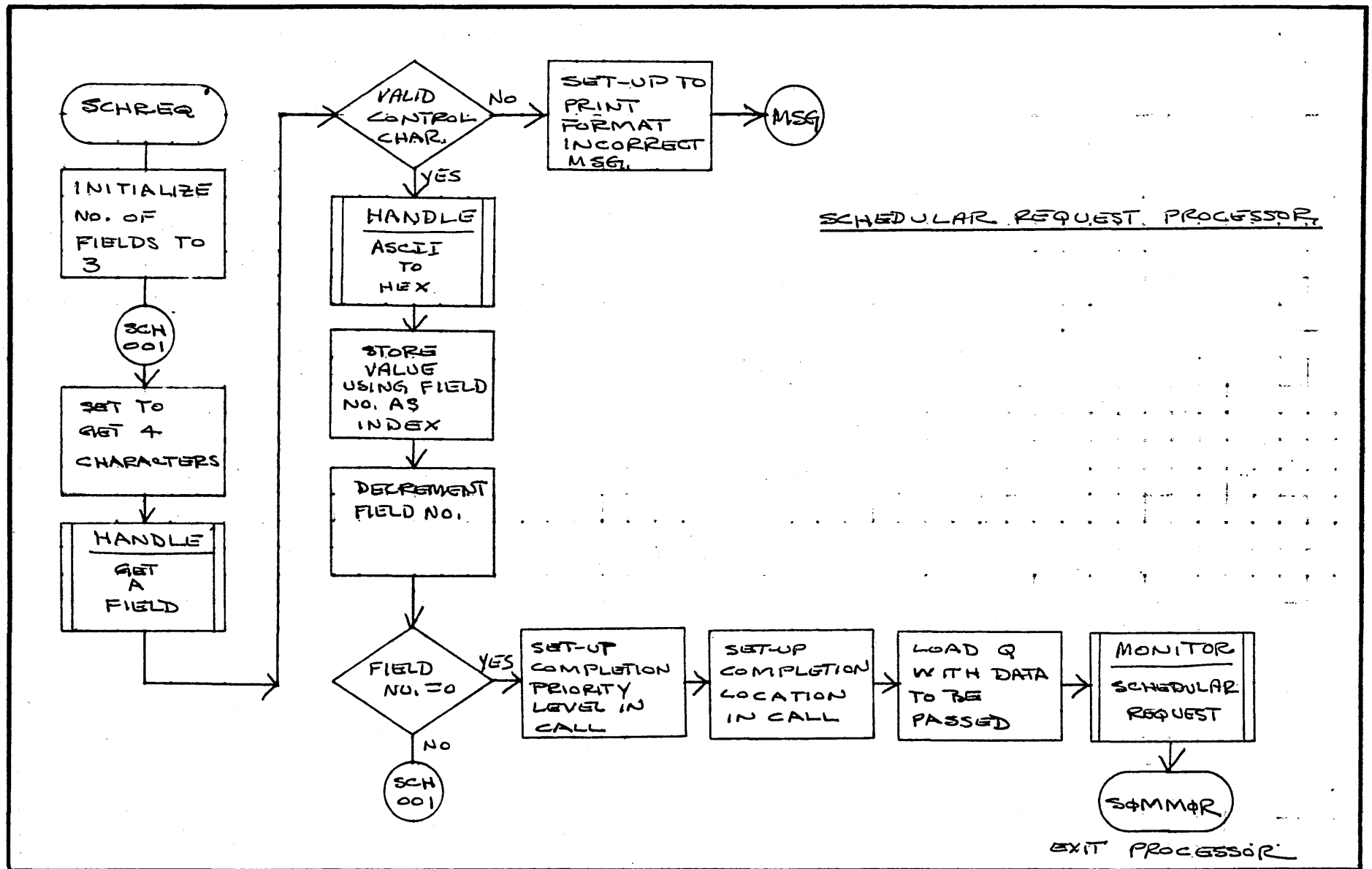


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
		PAGE 15 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.29

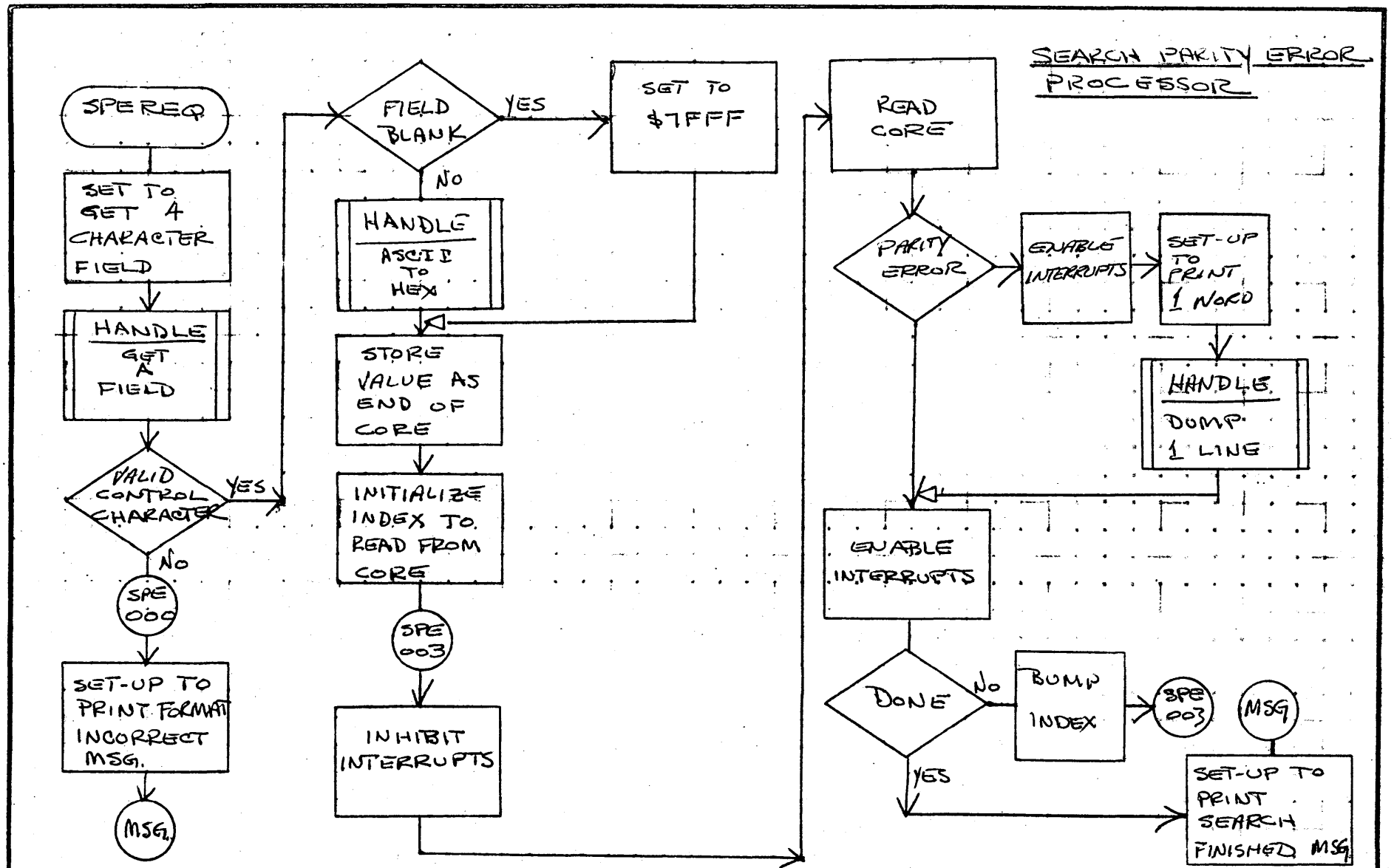


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
		PAGE 14 OF 94	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA 385 (FORMERLY CA12

MAR 5 1971

38-27

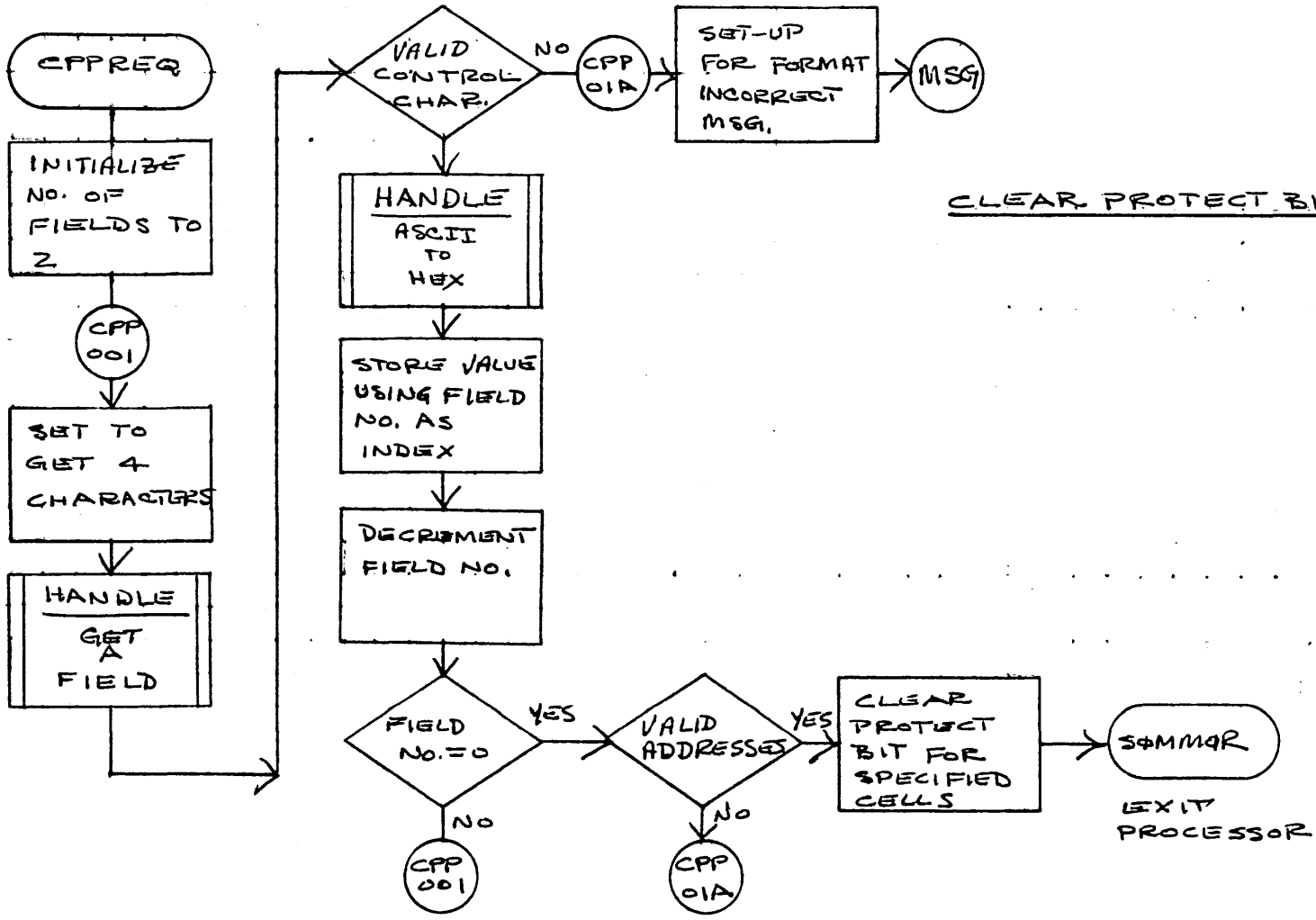


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 16 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38,28



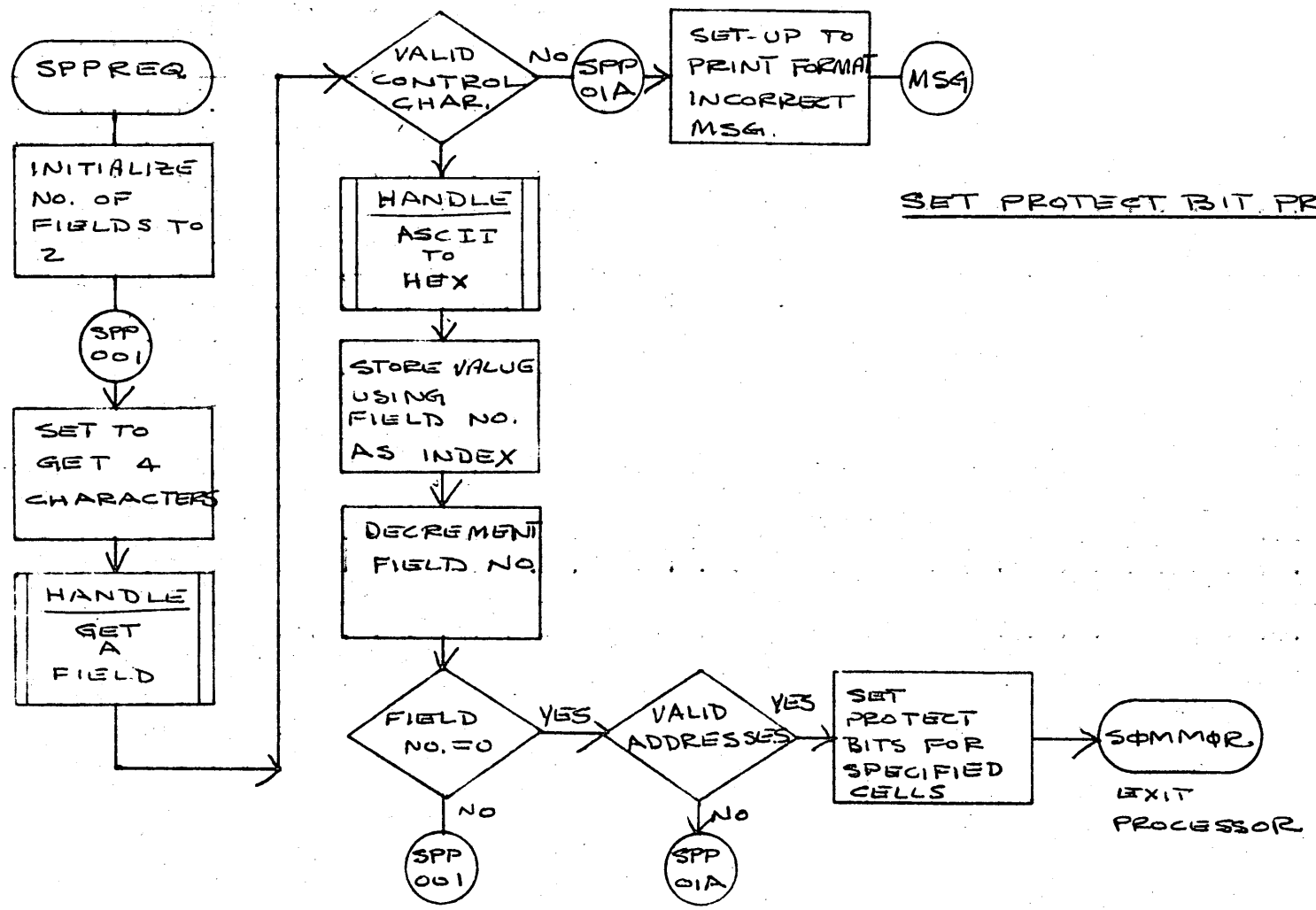
CLEAR PROTECT BIT PROCESSOR

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 0 DEBUG		PROJECT MGR.		
		PAGE 16 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.29



SET PROTECT BIT PROCESSOR

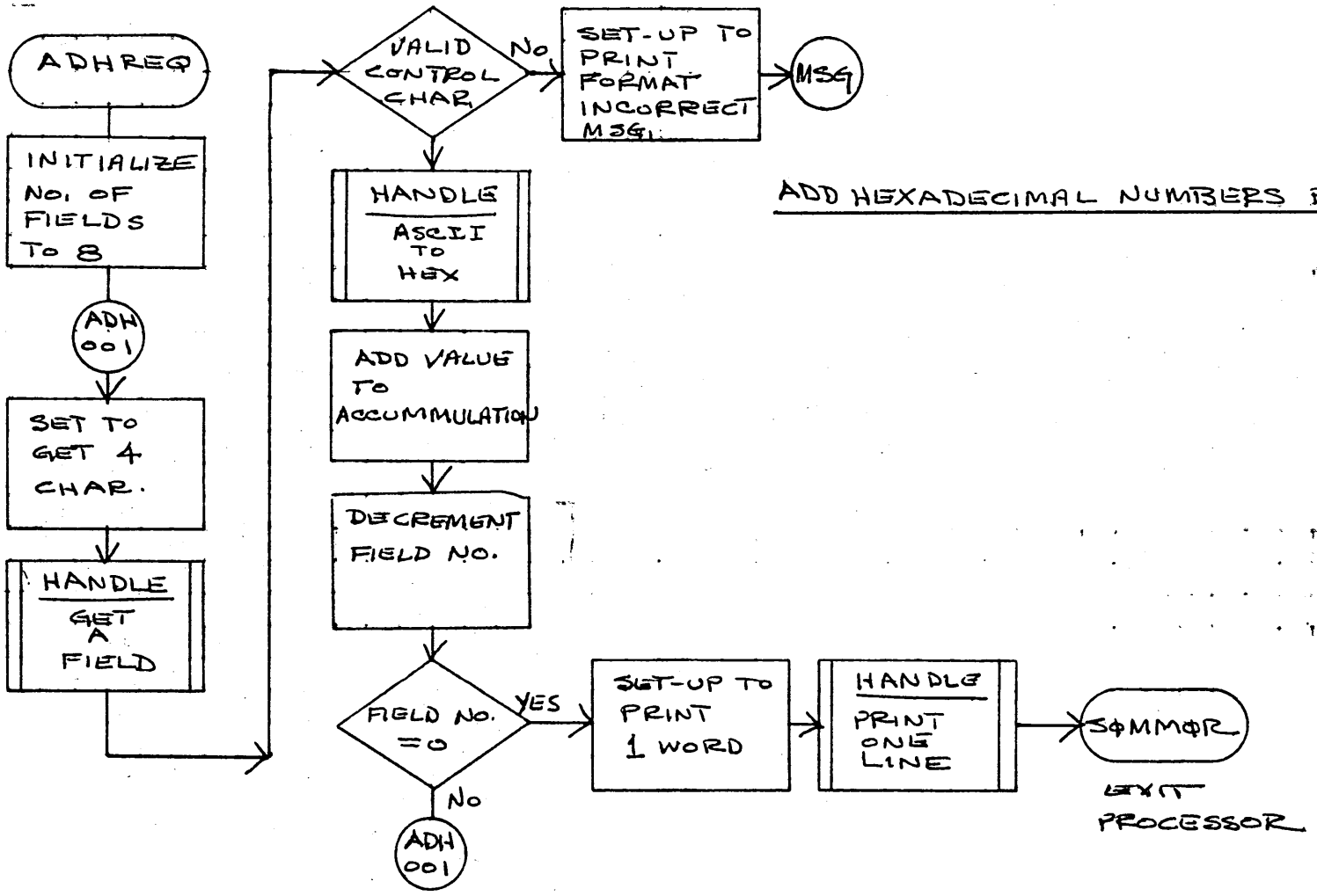
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODE306		PROJECT MGR.		
		PAGE 17 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

3.30

A B C D



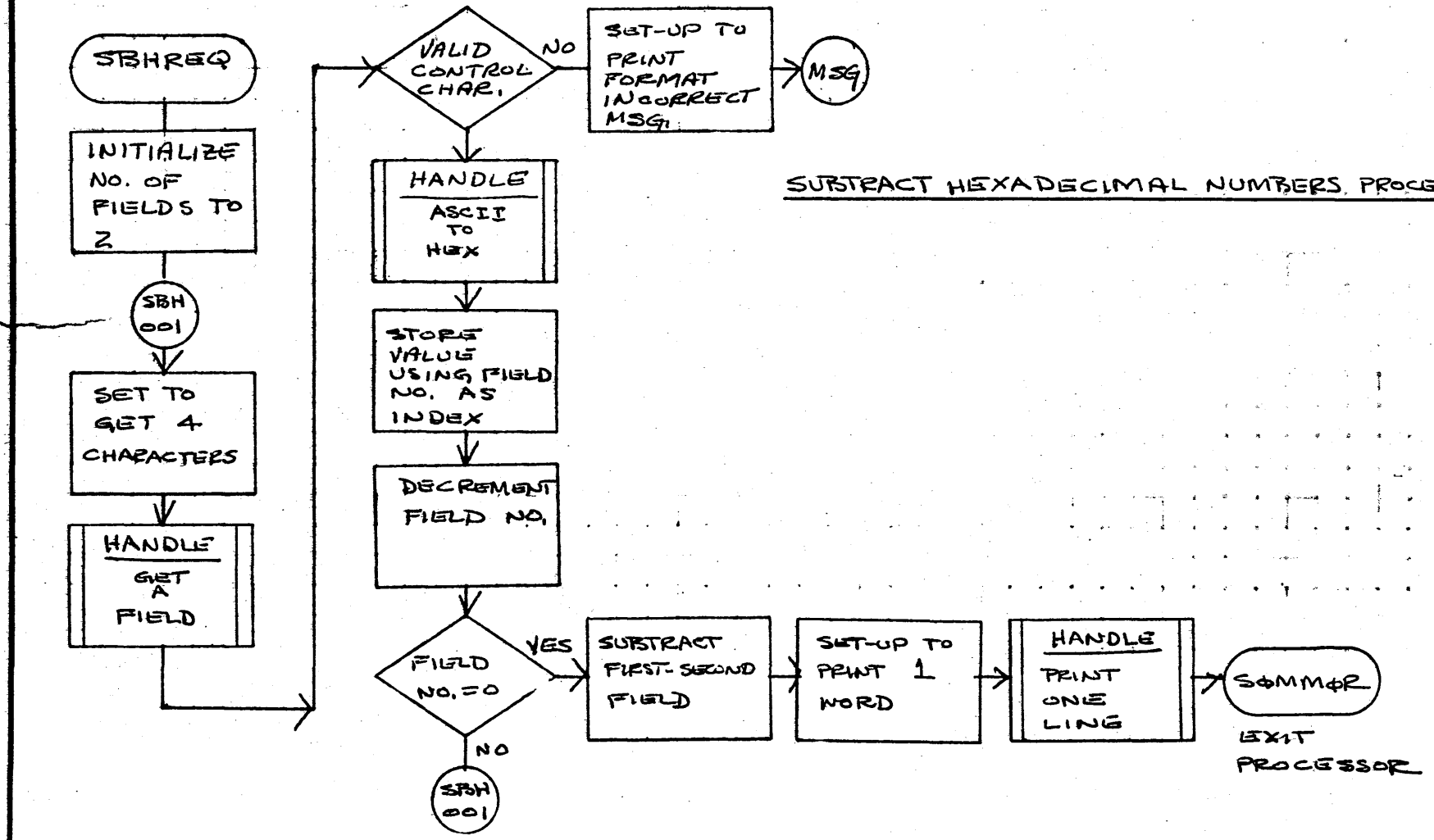
ADD HEXADECIMAL NUMBERS PROCESSOR

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODE306		PROJECT MGR.		
		PAGE 18 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

38.33

A1385 (FORMERLY CA127)

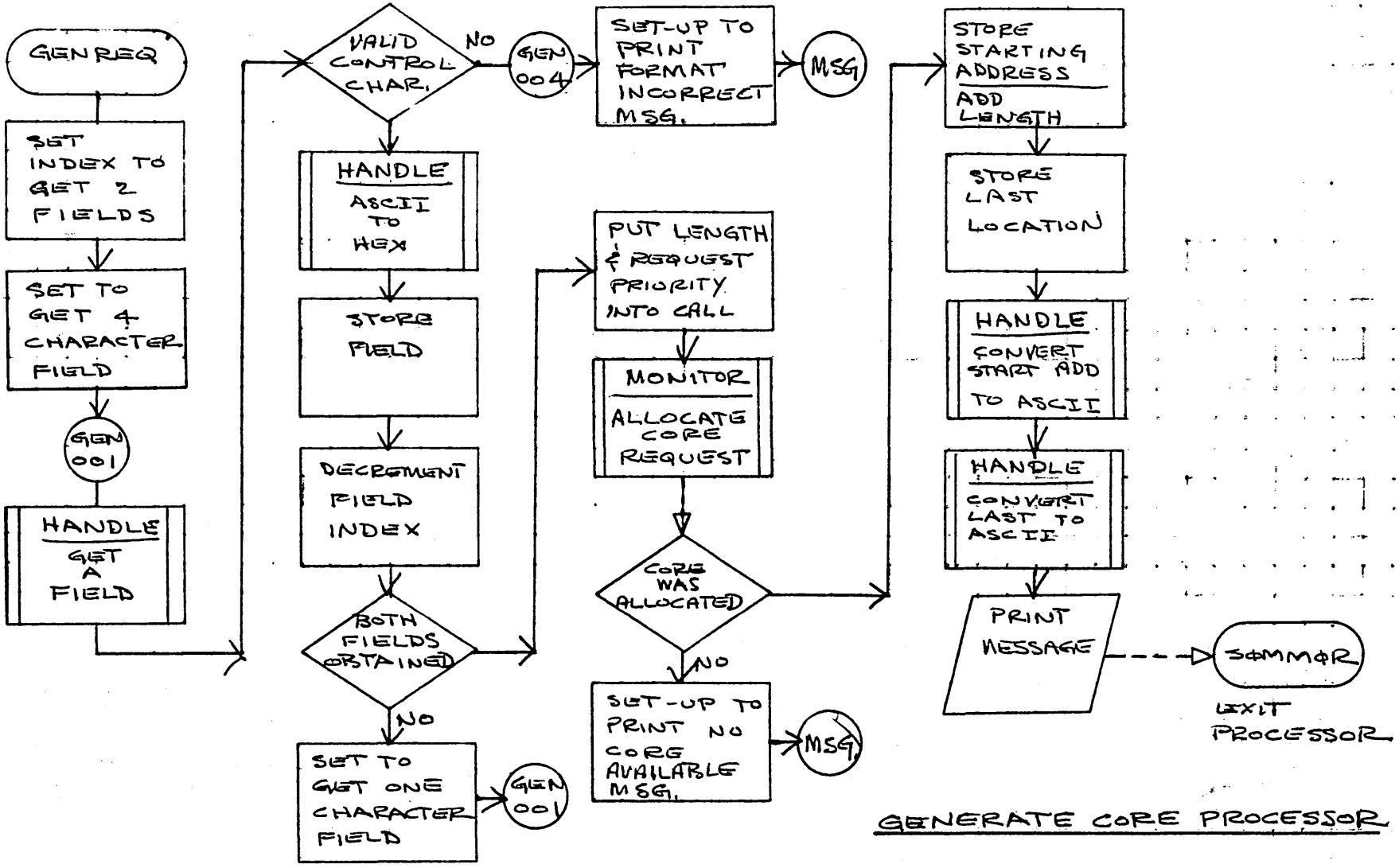


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
		PAGE 19 OF 34	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.32



GENERATE CORE PROCESSOR

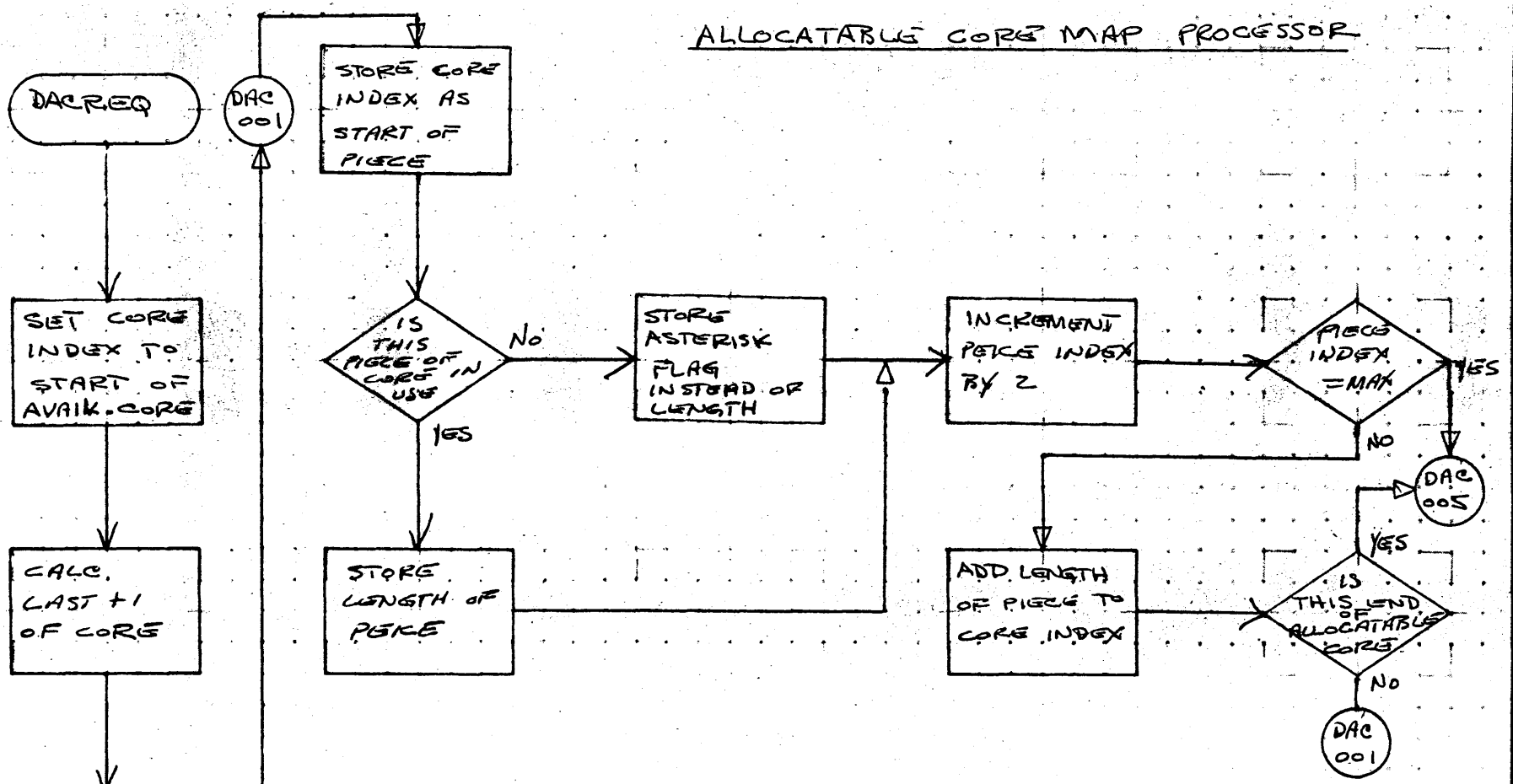
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODEBUG	PROJECT MGR.			
	PAGE 20 OF 34	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

EB, EE

AA1385 (FORMERLY CA127-1)

ALLOCATABLE CORE MAP PROCESSOR

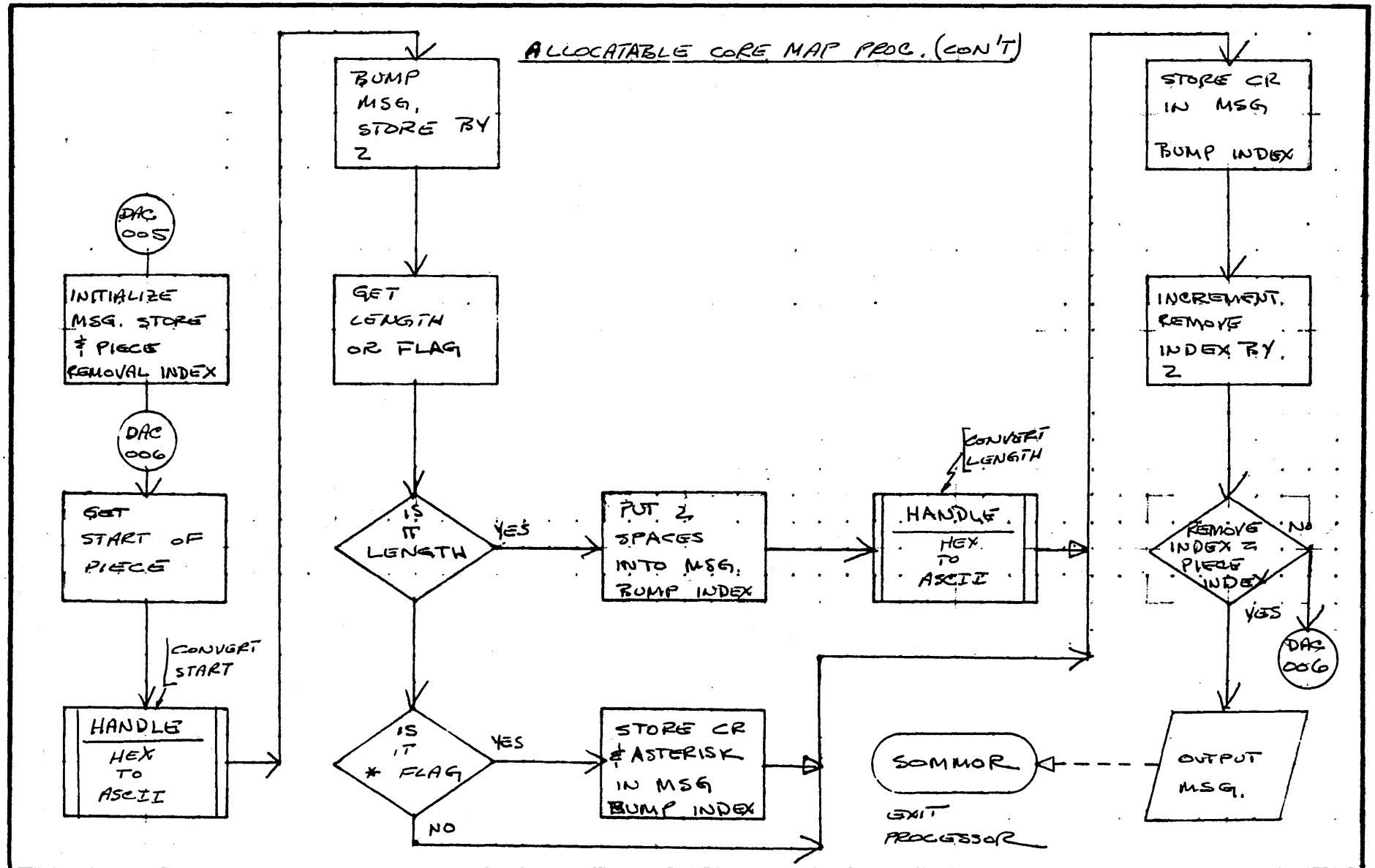


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
		PAGE 21 OF 34	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

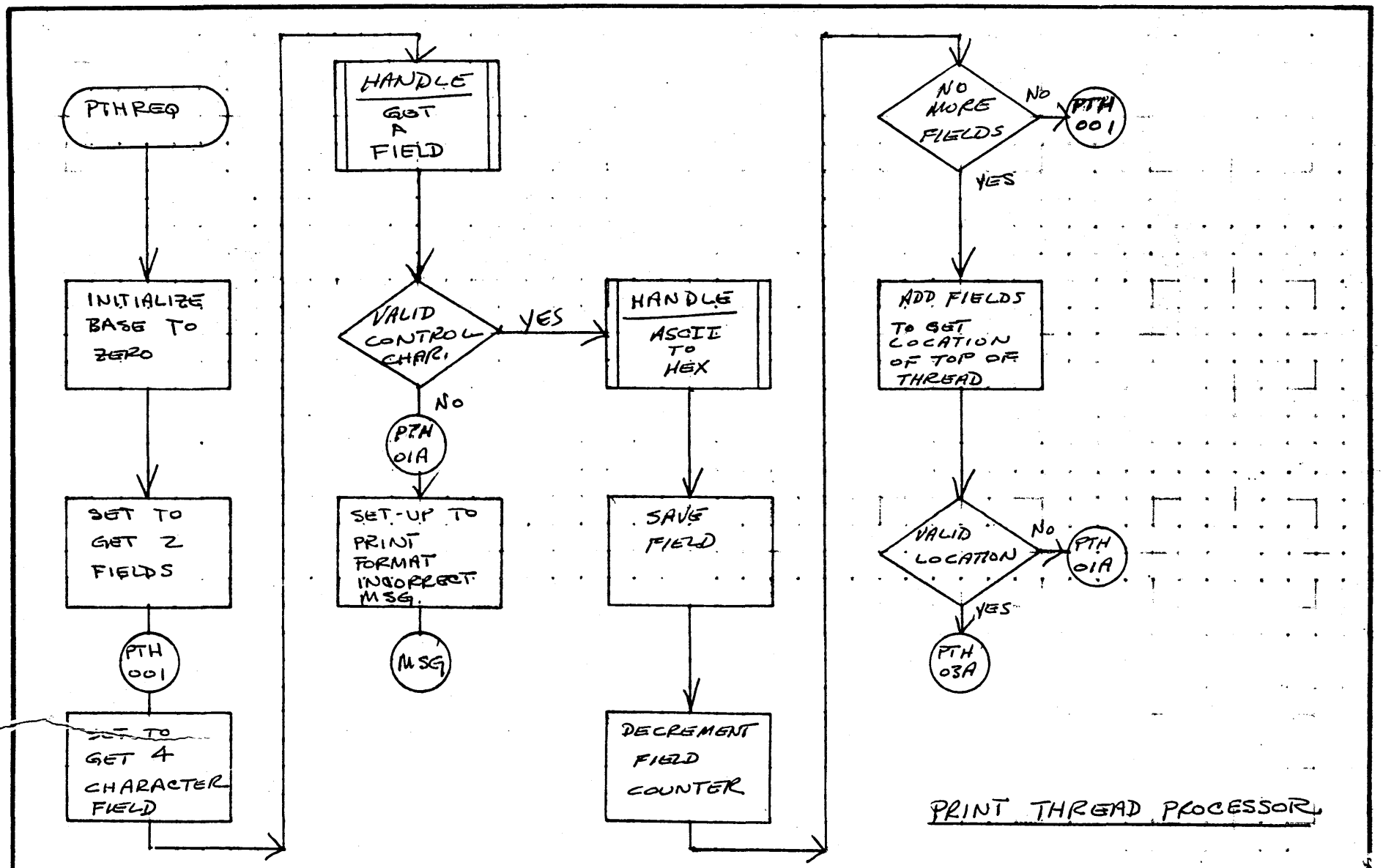
MAR 5 1971

3.3.34



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
		PAGE 22 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA 27-1)

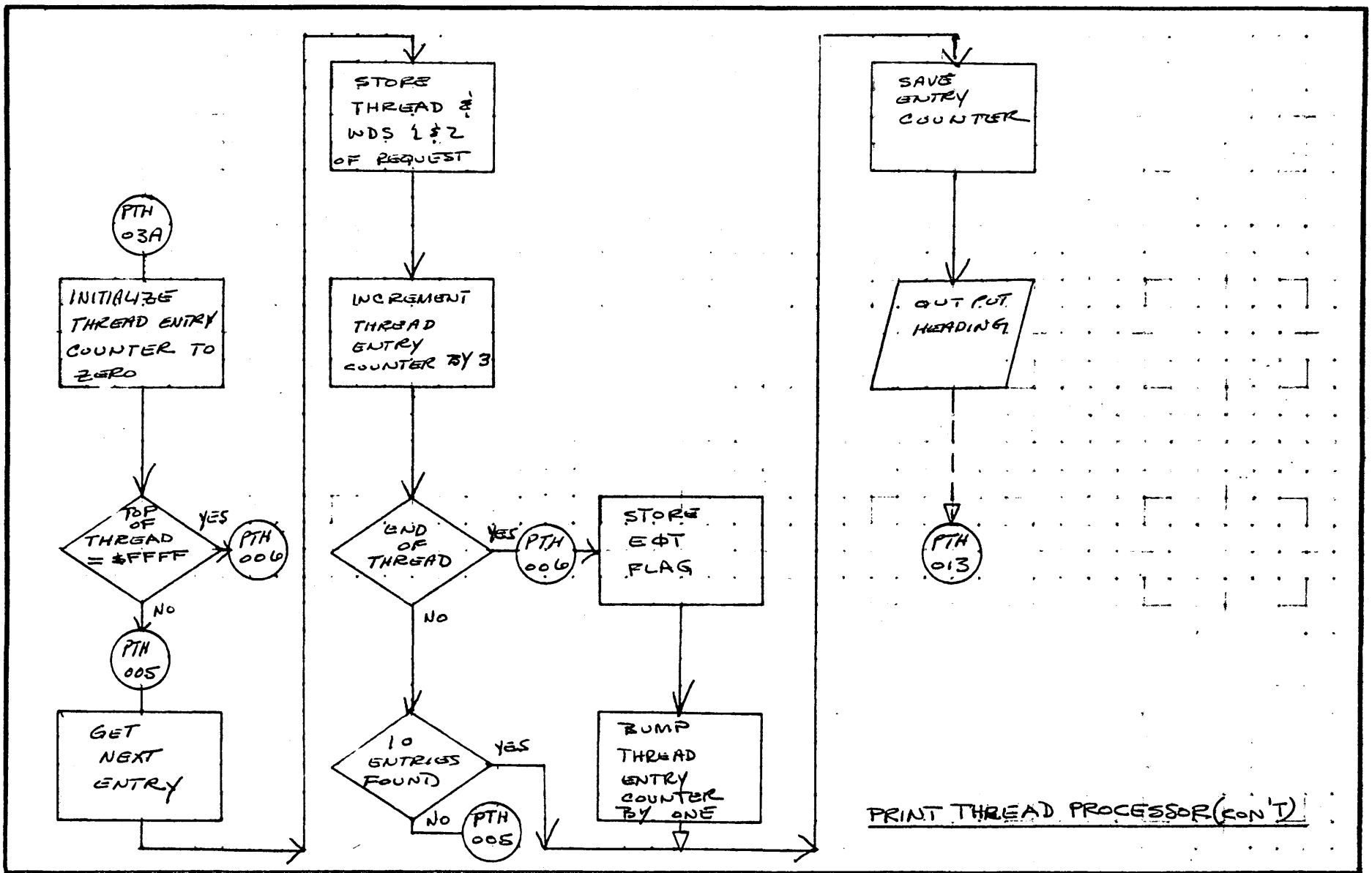


PRINT THREAD PROCESSOR

MAR 5 1971

CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODERUG		PROJECT MGR.			
		PAGE 25 OF 34	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)



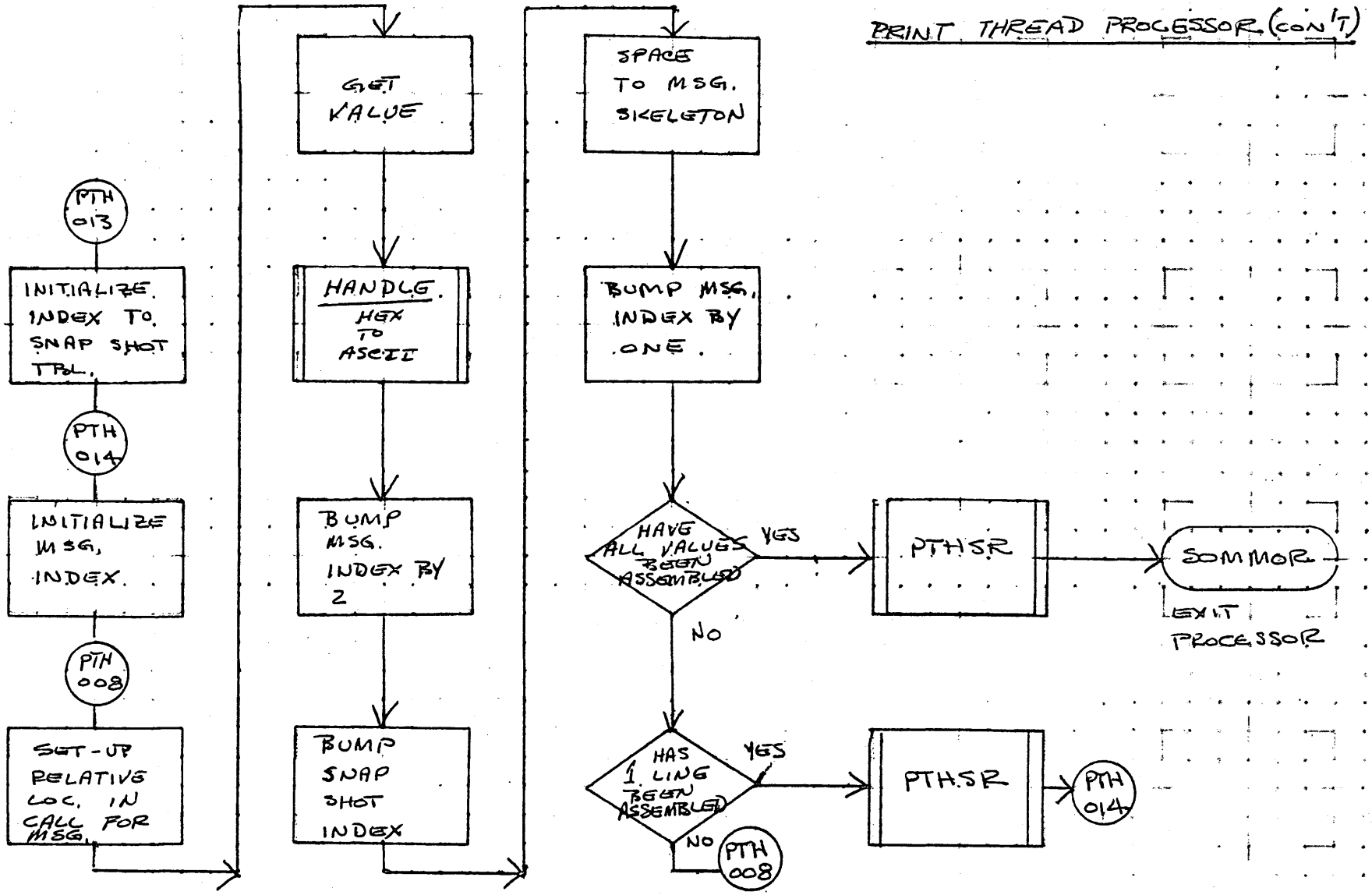
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 24 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

EB-37

AA1385 (FORMERLY CA127-1)

A B C D



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1200	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
		PAGE 25 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

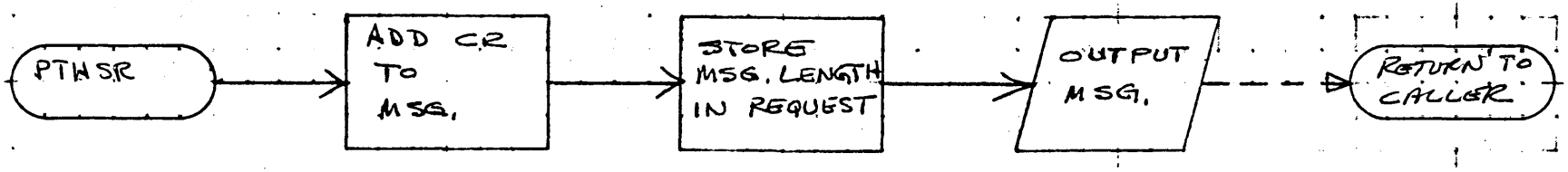
AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.38

PRINT THREAD PROCESSOR (CON'T)

OUTPUT SR

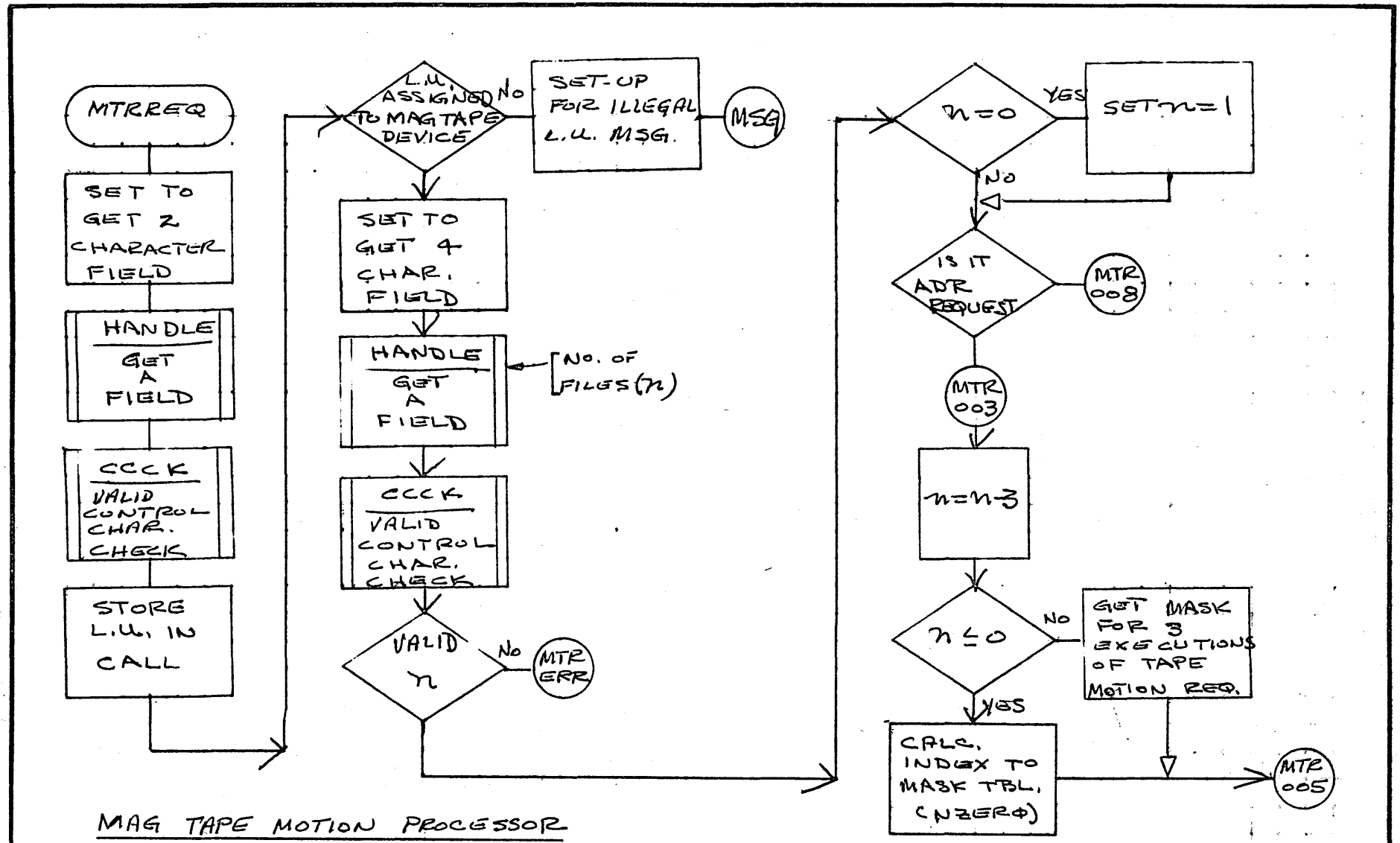


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 26 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

3a.39



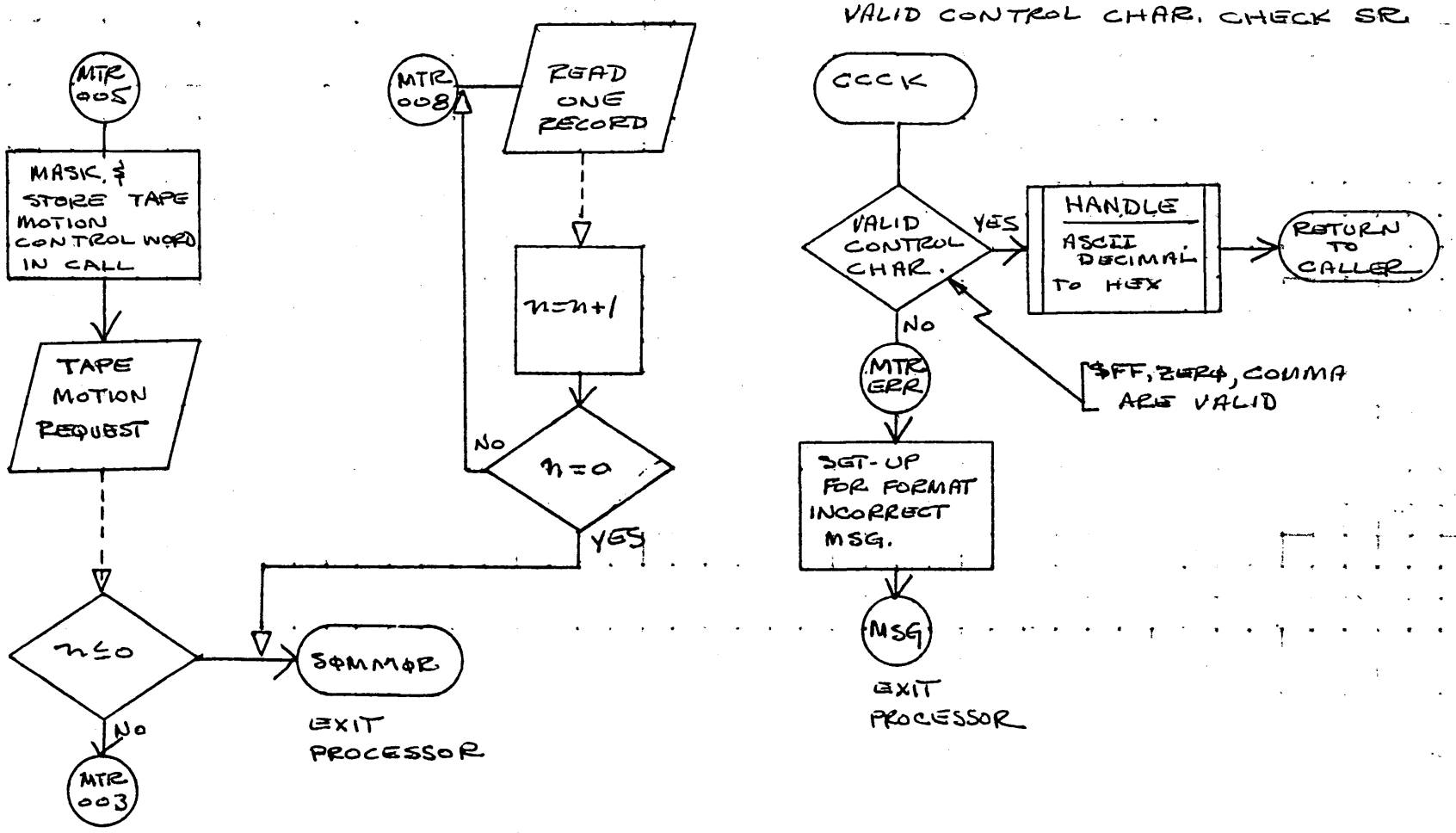
MAG TAPE MOTION PROCESSOR

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ODERG	PROJECT MGR.			
	PAGE 27 OF 34	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

3.40



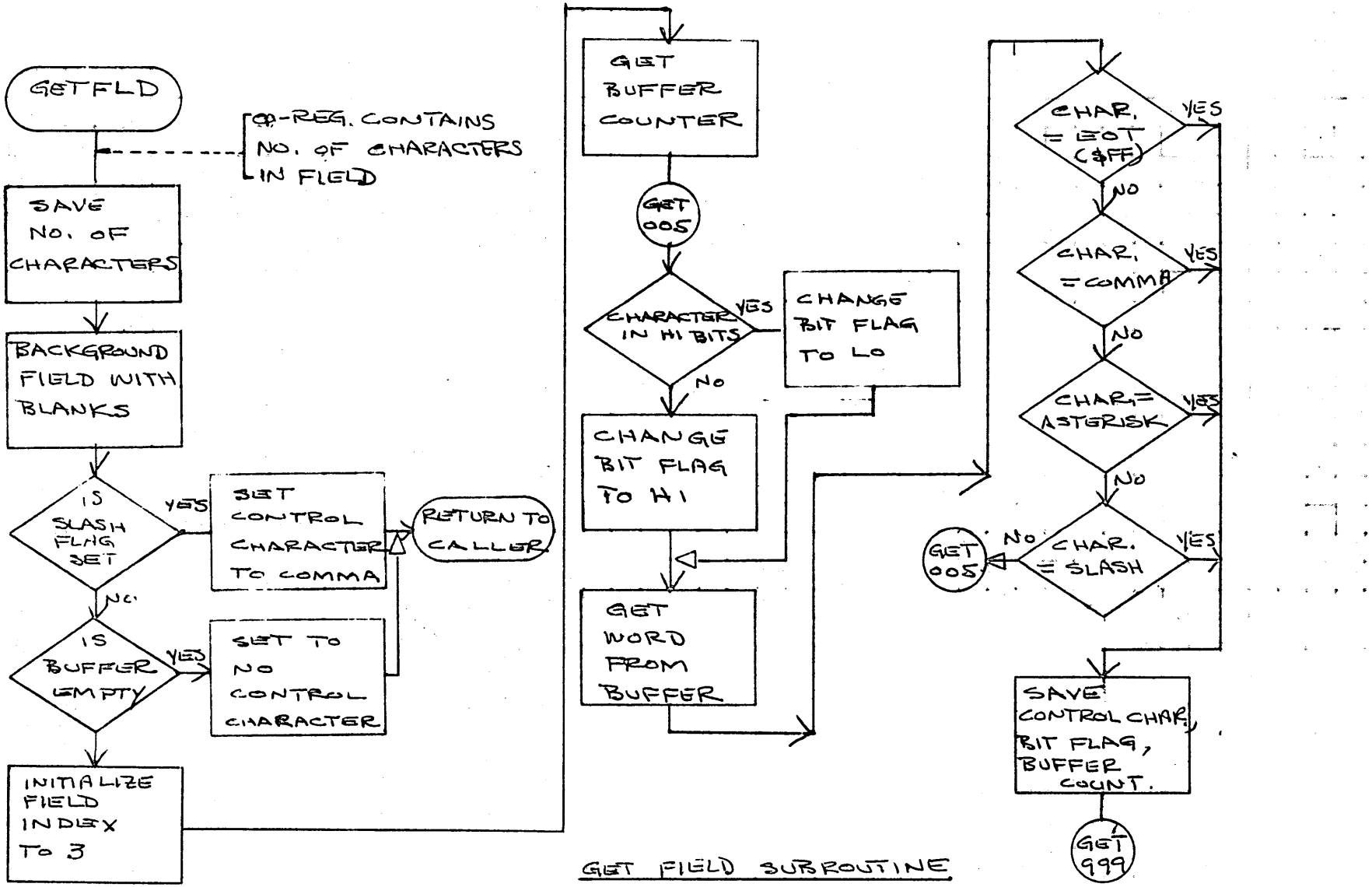
MAG TAPE MOTION PROCESSOR (CON'T)

CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
	PAGE 28 OF 34		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

EB.41



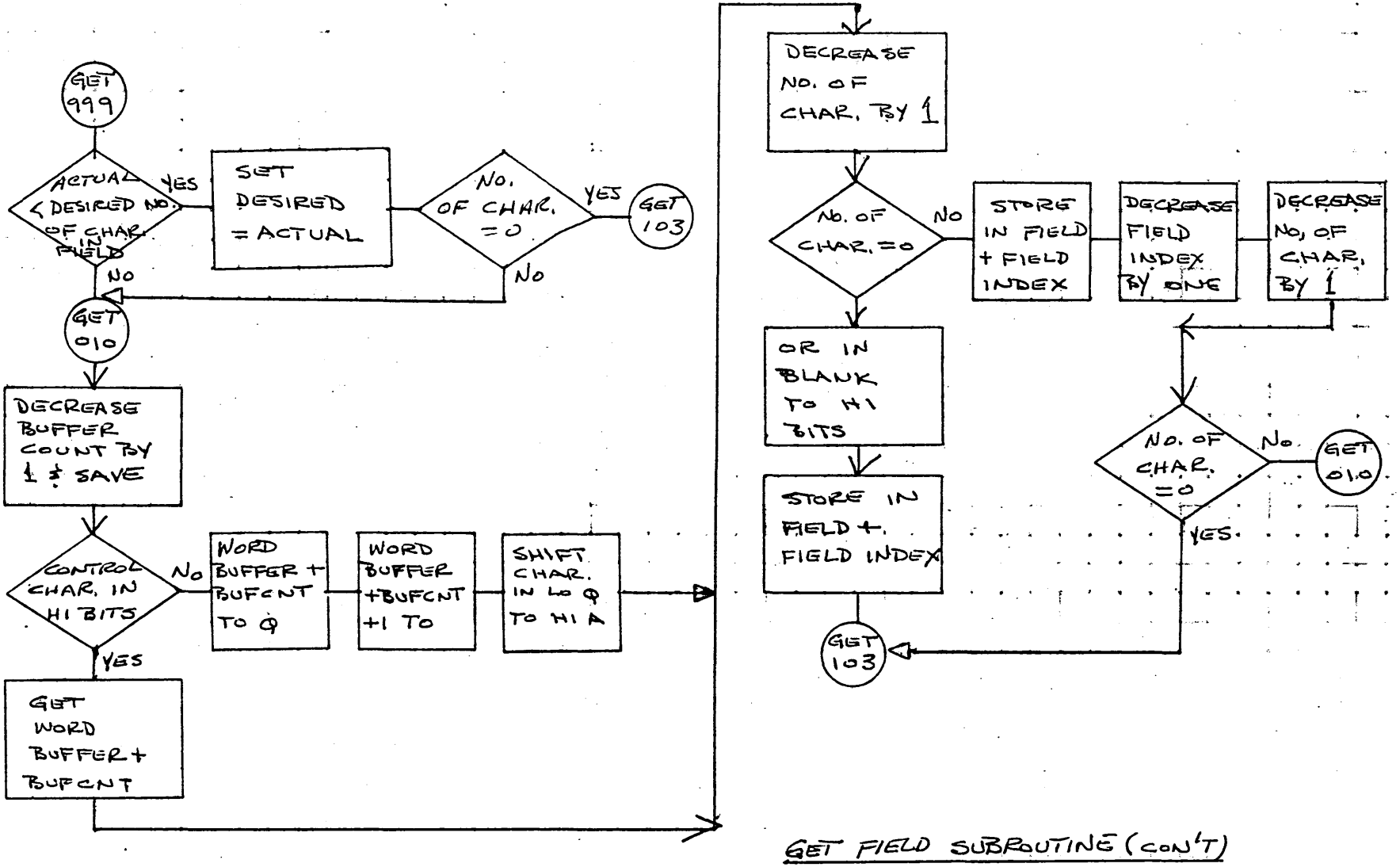
GET FIELD SUBROUTINE

CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
SAMPLE CODE ☐		PAGE 29 OF 34	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

AA1385 (FORMERLY CA127-1)

MAR 5 1971

38.42

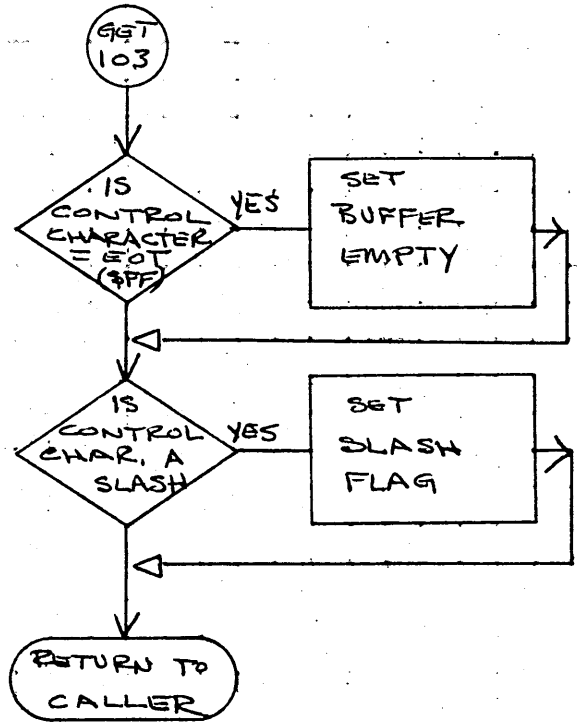


GET FIELD SUBROUTINE (CON'T)

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ODEBUG		PROJECT MGR.		
		PAGE 39 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971



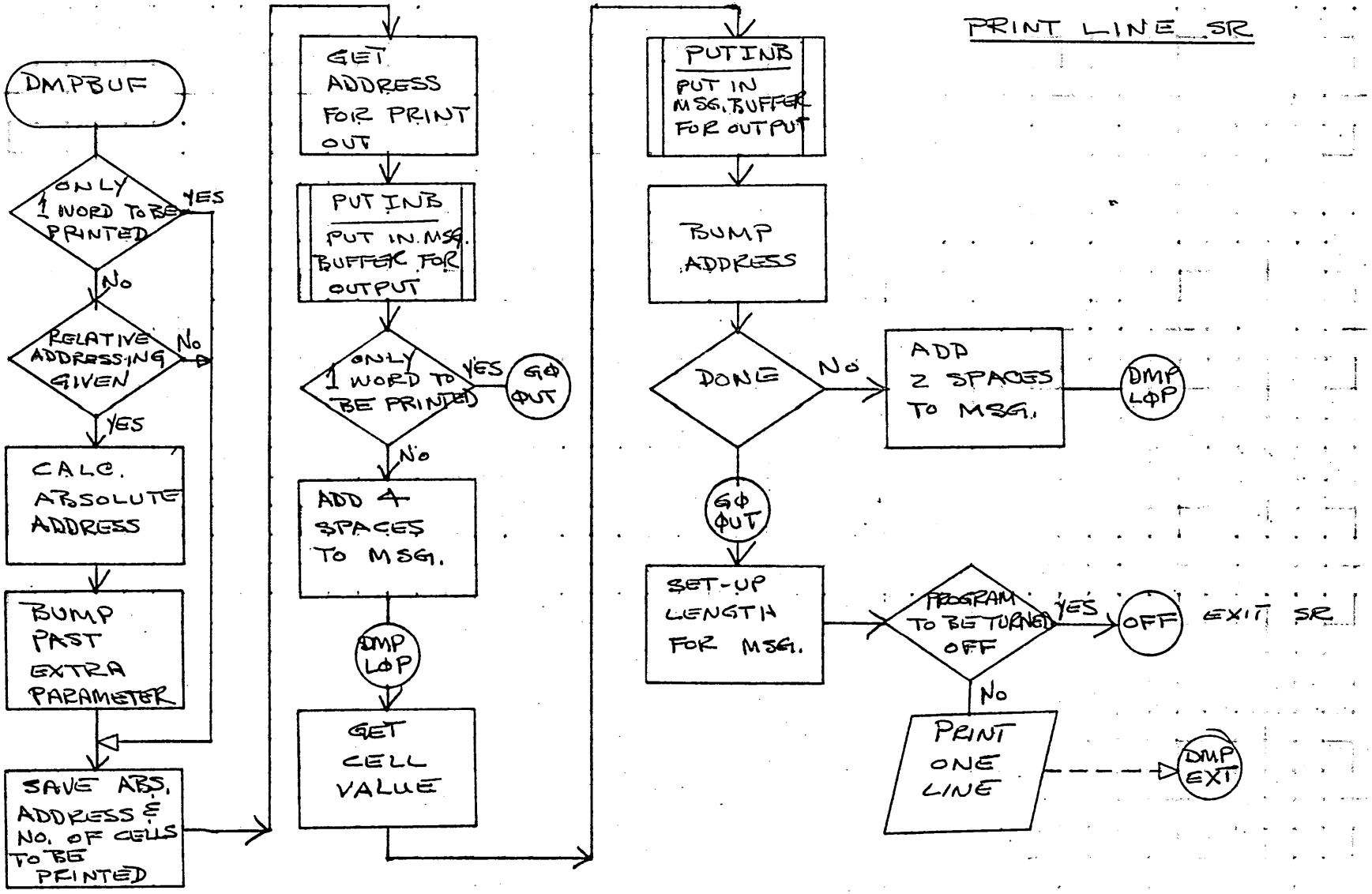
GET FIELD SUBROUTINE (CONT)

CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE ODERUG	PROJECT MGR.			
	PAGE 31 OF 34	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

3B-44

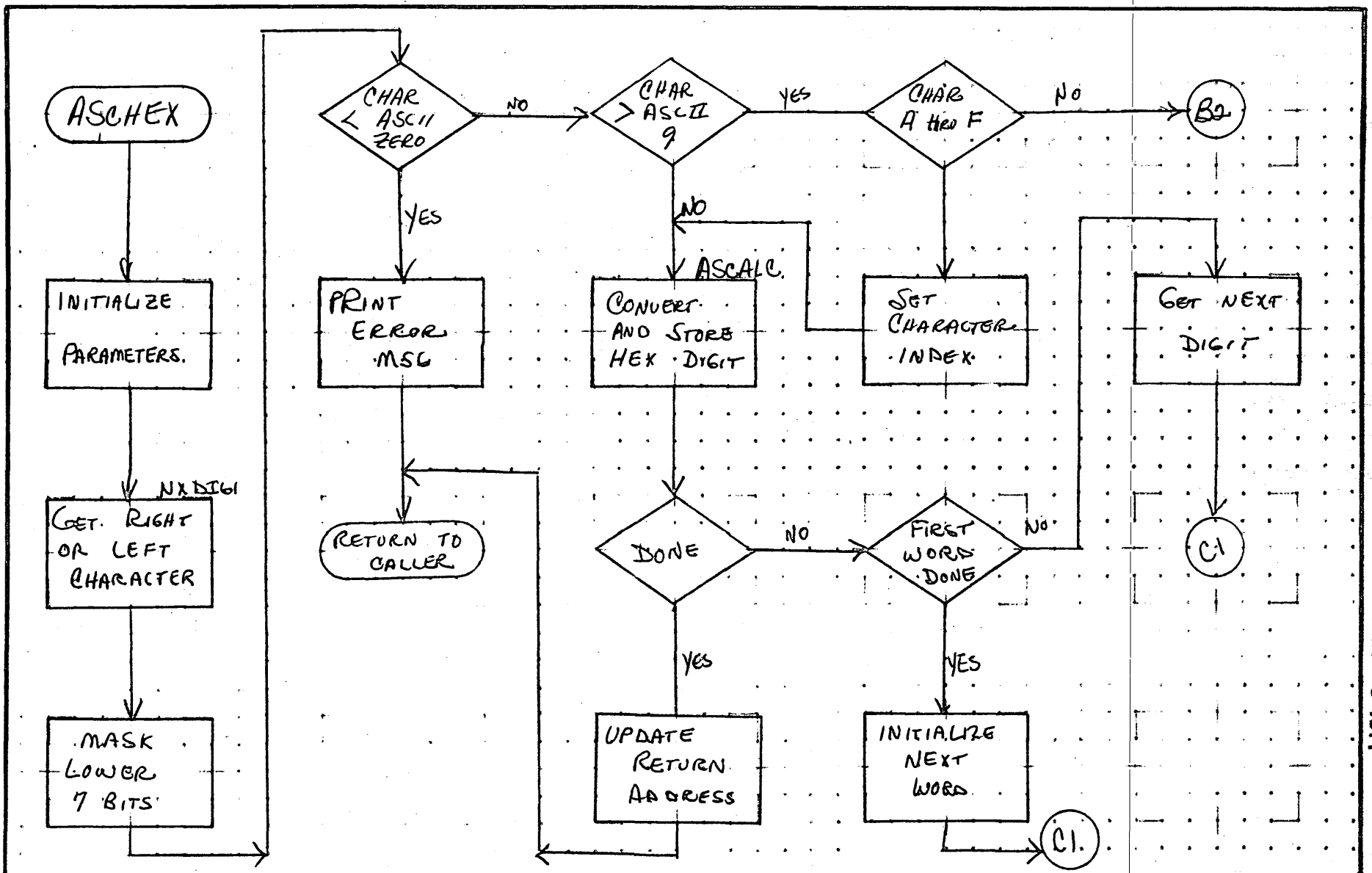


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 32 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

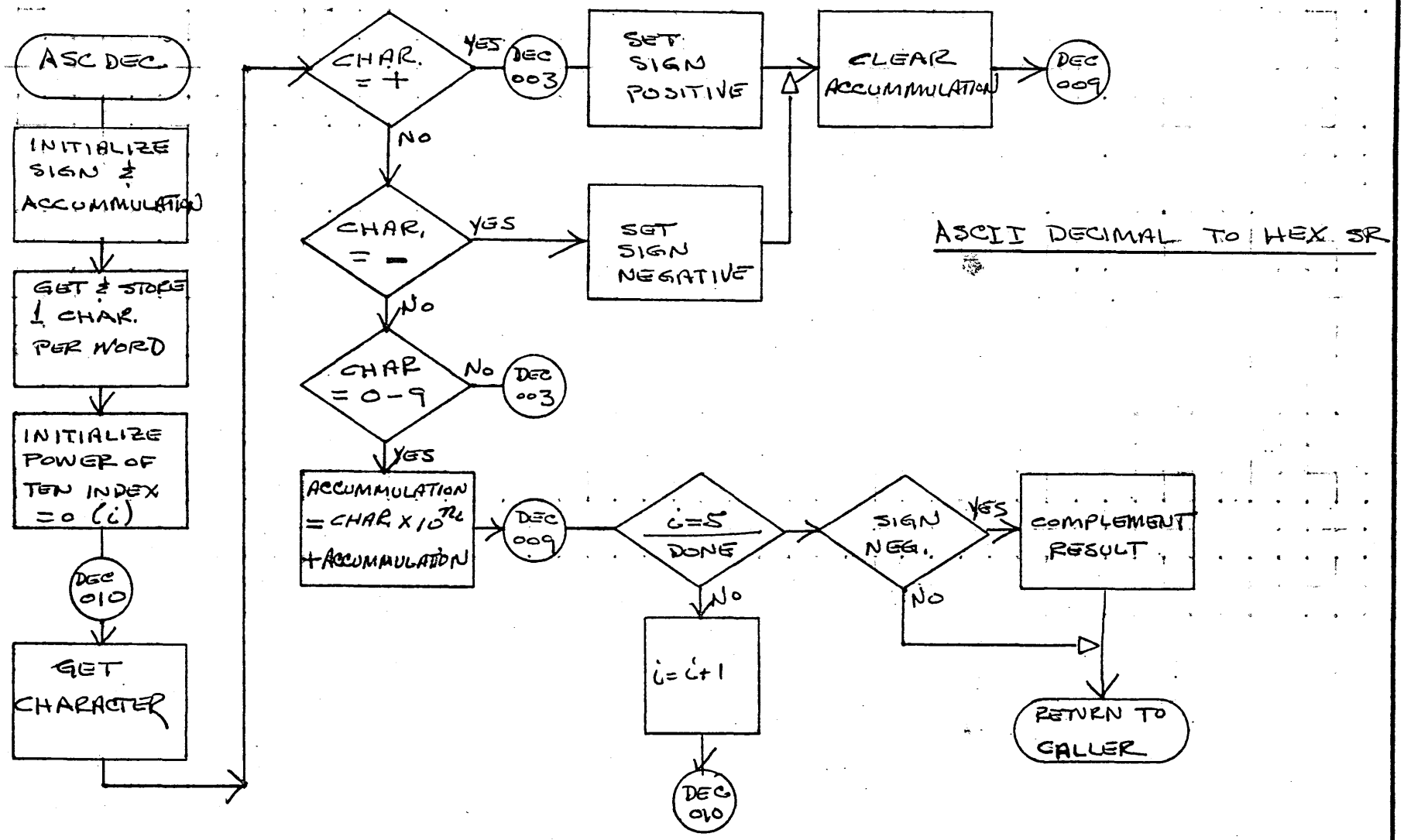
MAR 5 1971

38-45



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ØDEBUG		PROJECT MGR.		
		PAGE 33 OF 34	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ODEBUG		PROJECT MGR.			
FLOWCHART ☐		PAGE 34 OF 34	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

EB.47

AA1385 (FORMERLY CA127-1)

39.0 RELOCATING LOADER

The Loader is a non-resident sub-program of the 1700 Operating System. The Loader is capable of loading relocatable binary programs produced by the 1700 Assembler. The design of the Loader is independent of the I/O configuration of the hardware for the system on which it operates. A single version of the Loader accepts input from any device, whether buffered or unbuffered. One of these devices may be a mass storage device such as the Library Unit or the Scratch Unit.

39.1 STORAGE OF THE LOADER

The Loader is stored in relocatable binary form on some external medium. The Loader is placed in the System Library as an absolute record during the System Initialization procedure.

39.2 LOADING OF THE LOADER

The Loader may be brought into core in one of two ways:

1. Pre-job initialization by the Job Processor.
2. A formal Loader Request within the user's program.

The Loader is read as an absolute record from the System Library and placed in the upper most part of unprotected core. The Loader is accessed in the System Library through an entry in the System Library Directory. A word in this entry contains the length of the Loader. The length of the Loader is subtracted from the address which is the upper limit of unprotected core. The resulting address is the 1st core location to be occupied by the Loader.

The upper limit of unprotected core is contained in location $F6. It is defined as the highest {toward $7FFF} unprotected address +1. The lower limit of unprotected core is contained in location $F7. It is defined as the lowest {toward 0} unprotected address -1.

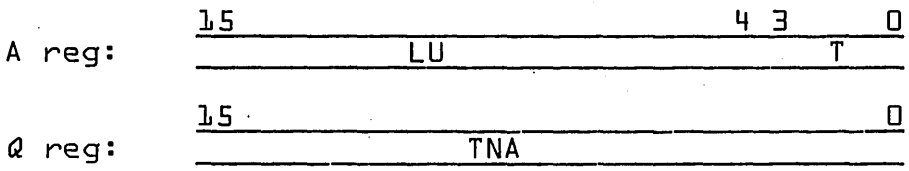
Pre-job initialization by the Job Processor is performed whenever the Job Processor reads an ▽*P▽ input statement. Refer to Chapter 26 for further information concerning pre-job initialization. Refer to Chapter 11 for further information concerning Loader Requests.

39.3 OPERATION OF THE LOADER

A Loader operation is initiated with a jump to the lowest {toward 0} core location occupied by the Loader.

39.3.1 Types of Loader Operations

The particular Loader operation to be performed is determined by the information passed to the Loader from the monitor via the A and Q register as illustrated:



The "T" field indicates the type of Loader operation to be performed.

T Field	Loader Operation
0	Relocatable Binary Load
1	Subroutine Load
2	Program Load
3	Memory Map List
4	Entry Point Lookup
5	Subroutine Loading
6	Patch to Core Resident
7	Set Data Base

The "LU" field contains the logical unit number of the input device to be used for this operation. This field is ignored if the Loader operation is Memory Map List.

The TNA field contains the core address for storage of an entry point name. This information is significant only if the Loader operation is Program Load. The Q register is ignored in all other cases.

39.3.1.1 Relocatable Binary Load Operation

The purpose of this operation is to load relocatable binary programs from any peripheral device. The Loader call to load relocatable binary input requires that the T field is set to zero. The LU field contains a number which refers to an ordinal in the equipment table. If the left most bit of the LU field is one, the Loader will assume the input device is the standard input device. In this case, the Loader will address the standard input device indirectly via the address $F9 in the communication region, which is the location containing the ordinal for this device in the equipment table. The Q register is ignored for relocatable binary loading operations.

39.3.1.2 Relocatable Binary Load Operation for Load-and-Go Input

The purpose of this operation is to load relocatable binary programs from the mass storage unit which contains the scratch area. The Loader call to bring in LOAD-AND-GO input requires that the T field is set to zero. The LU field is set to the number of the equipment table entry for the mass storage device containing the scratch area. The Q register is ignored.

For LOAD-AND-GO input, the Loader will address the mass storage device containing the scratch area indirectly via the address $B3 in the communication region. This is the location containing the ordinal for this input device in the equipment table.

39.3.1.3 Subroutine Load Operation

This operation is performed subsequent to relocatable binary loading. As a part of the relocatable binary loading procedure, the Loader will attempt to match external names in the Loader Table with entry point names in the Program Library Directory. Unpatched externals are those for which there are no corresponding entry poing names in the Program Library Directory.

For every match that is made, the Loader will load the appropriate routine from the Program Library. If any unpatched externals are not matched by entry names in the Program Library, the names for these unpatched externals are printed on the comment medium. The operator must then type in ×{carriage return} to resume operations, or ×T {carriage return} to abandon the job. For subroutine loading, the T field is 1 and LU field is ignored. The input device is understood to be the mass storage device containing the Program Library. The Q register is ignored in subroutine loading.

The Loader will address the mass storage device containing the Program Library indirectly via the address $C2 which contains the ordinal of this device for the equipment table.

39.3.1.4 Program Load Operation

This operation is used to load a program from the Library unit and enter it immediately for execution.

For program loading, the T field is 2 and LU field is ignored. The input device is understood to be the mass storage device containing the program library. The Q register contains the core address where the program name is stored in the internal core of the computer {ASCII}.

The input device is addressed by the Loader in the same manner as for a SUBROUTINE LOAD OPERATION.

The program name is an entry point name of the program to be loaded, and it must appear in the Program Library directory.

39.3.1.5 Memory Map Operation

The T field is 3 if the Loader is to produce a memory map. The LU field is ignored and the Q register is ignored. This operation may take place subsequent to each subroutine load. This type of operation consists of the listing of the names in the entry point table together with their respective addresses.

The first word addresses of common and data storage reservations appear in the map as entry point addresses.

If common storage had been declared during a previous load operation, the name ▽⋈⋈⋈COM▽ together with the common storage relocation base would appear ahead of the entry point table on the list output. If data storage had been declared during a previous operation, the name ▽⋈⋈⋈DAT▽ together with the data storage relocation base would appear ahead of the entry point table on the list output.

39.3.1.6 Entry Point Lookup

The T field is 4 for the Loader to look up an entry point. The Q register specifies the location of the name as follows:

Bit 15 = 1: core location of ASCII name

Bit 0 = 0: name starts in left character of word

Bit 0 = 1: name starts in right character of word

The Loader tries to find the specified name in its symbol table. If the name appears, the A register contains the core address of the name on exit. If the name is not present, the Loader types on E03 message on the output comment device and waits for input.

39.3.1.7 Subroutine Loading

If T = 5 on input, the Loader does the same as if T were 1, but no memory map is produced.

39.3.1.8 Patch to Core Resident

If T = 6 on input, the Loader searches the core resident directory for entry points to match any undefined externals in its table. If any are found, a dummy program is loaded which contains the absolute addresses of core resident entry points.

This dummy program and its directory are written when the system is initialized. The directory is in the same format as the Program Library directory, with all entry points pointing to the same program. This program consists of a dummy NAM block, as many ENT blocks as necessary to accommodate all core resident entry points and a dummy XFR block.

The Loader treats this as a normal program, loads it in and patches any entry points referenced by other programs.

39.3.1.9 Set Data Base

If T = 7 on input, the Q register specifies the core location to which subsequent data blocks should be relocated.

39.3.2 Allocation of Core for Programs Loaded by the Loader

The limits of unprotected core are defined as follows:

1. The location $F6 a number equal to the address of the highest {toward $7FFF} unprotected location +1.

2. The location $F7 contains a number equal to the address of the lowest unprotected location -1.

The Loader occupies the upper part of this block of unprotected core. {Refer to item 28.2.} The lowest {toward 0} core location to occupied by the Loader has the address equal to -

{$F6} - the length of the Loader.

This address will hereafter be referred to as the ▽base address of the Loader.▽

39.3.2.1 Loader Table

The Loader Table generated by the Loader is situated immediately below the Loader in unprotected core. The Loader Table consists of entries of 5 words each in length. The first entry generated by the Loader is recorded at the five addresses immediately preceding the base address of the Loader. As additional entries are made to the Loader Table, the table is expanded in size downward through memory to the lower limit of unprotected core. The address of the lowest {toward 0} location occupied by the Loader Table will hereafter be referred to as the ▽base address of the Loader Table.▽

397 3.2.2 Available Core

That portion of the block of unprotected core not occupied by the Loader and the Loader Tables is core that is available for storage of the relocatable binary input read and processed by the Loader.

By definition the upper limit of available core is equal to the base address of the Loader Table.

The lower limit of available core is by definition the address in the location $F7 which is also the lower limit of unprotected core. (Refer to item 28.3.2.11 for a definition of the lower limit of available core where there is a Data Storage block reservation.)

Programs are loaded into core starting at the location whose address is ($F7)+1 and proceeding toward the upper limit of available core.

397 3.2.3 Temporary Limits of Unprotected Core

The locations $EC and $ED contain the temporary limits of unprotected core. At the time the Loader is brought into core by the monitor during pre-job initialization, the temporary limits of unprotected core are set to the limits of unprotected core:

($F6) ⟶ $EC where ($EC) = upper temporary limit of core

($F7) ⟶ $ED where ($ED) = lower temporary limit of core

The temporary limits of unprotected core may be altered by issuing a core request in which the A register is set to the upper address and Q the lower. The Loader issues such a core request at the end of each Loader operation (except for Maps). The lower temporary limit is reset to the address of the highest (toward $7FFF) location occupied by the programs loaded during this loader operation.

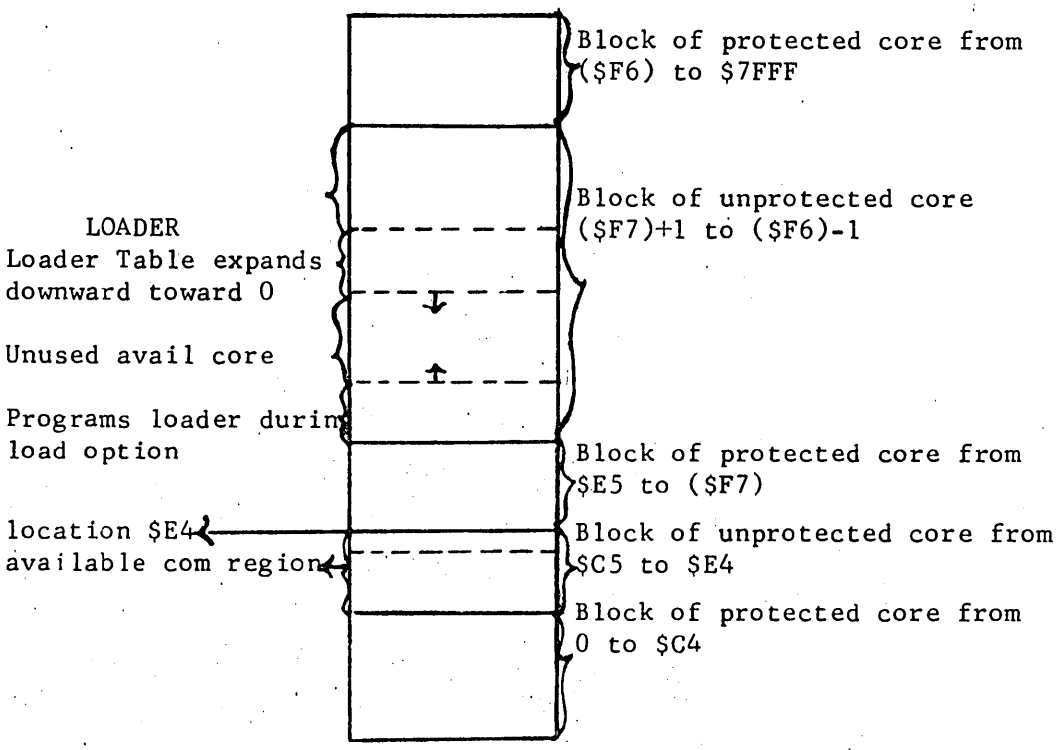
The upper limit = ($EC) is always reset to the upper limit of unprotected core unless a common storage reservation had been made either during the current load operation or a previous one. (Refer to item 28.3.2.6.)

397 3.2.4 Unprotected Core in the Communication Region

In addition to the unprotected core block defined in item 28.3, there is a block of unprotected core in the communication region. The unprotected block begins at the location $C5 and terminates at $E4.

In addition to the limits of available core defined by item 28.3.2.2 the Loader regards the block of unprotected locations in the communications region starting at $C5 and terminating at $E3 as available core. Although $E4 is unprotected, it is not included as part of available core since it contains information used by the Loader for a Load-and-Go operation.

39.3.2.5 Diagram of Unprotected Core and Available Core



39.3.2.6 Allocation of Core for Common Storage Reservation

Locations which are set aside for a Common Storage block reservation during a Loader Operation are situated at the top of unprotected core. Depending on the length of the reservation, it may overlay part or all of the Loader (and/or the Loader Table). The address of the lowest (toward 0) storage cell of this reservation is equal to

($F6) - no. of location of Common Storage.

The starting address of the Common Storage Reservation will hereafter be referred to as the "relocation base for Common Storage".

Item 28.3.2.2 discusses Available Core and defines the limits thereof. When a Common Storage block reservation is set aside during a Loader Operation, this definition remains valid provided the following is true:

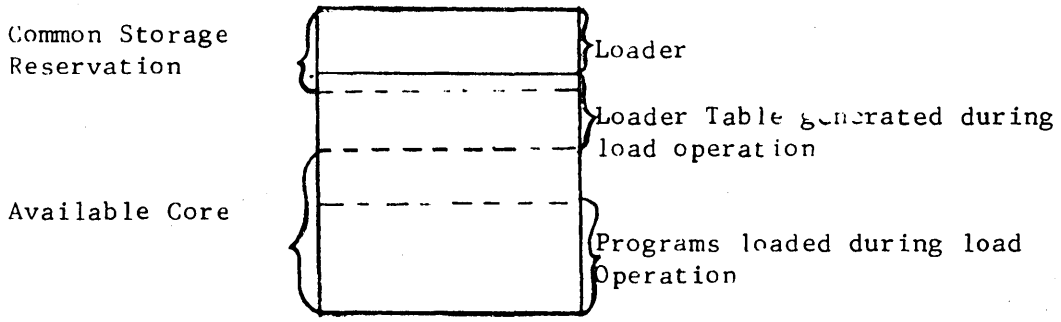
The address equal to the relocation base for Common Storage is numerically greater than or equal to the base address of the Loader Table.

The Loader will not load a program into core if it is to be stored in an area of memory which is set aside for Common Storage. Therefore, if the address equal to the relocation base of Common Storage is numerically less than the base address of the Loader Table, the following is true:

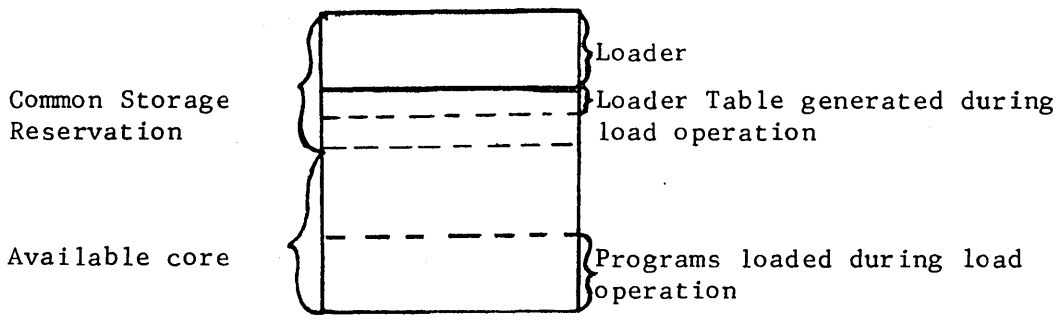
The block of Available Core is truncated at its upper end such that the relocation base of Common Storage replaces the base address of the Loader Table as its upper limit.

Item number 28.3.2.3 discusses the Temporary Limits of Unprotected Core and defines the limits thereof. The upper limit of unprotected core remains as the upper limit address in location $EC until a Common Storage block reservation is made. At this time the relocation base for Common Storage replaces the upper limit of unprotected core as the upper limit address in $EC. It remains constant during all subsequent Loader Operations. (The next time the monitor brings the Loader back into core these temporary limits will be reset to the permanent limits in locations $F6 and $F7.)

39 3.2.7 Diagrams of Unprotected Core with a Common Storage Reservation



In the example above the relocation base of Common Storage is numerically greater than the base address of the Loader Table. Therefore, the base address of the Loader Table is the upper limit of Available Core.



In the above example the relocation base of Common Storage is numerically less than the base address of the Loader Table. Therefore, the relocation base of Common Storage is the upper limit of Available Core.

3913.2.8 Extension of Available Core

When a program is being loaded into core, the first input record of relocatable binary format read by the Loader is the NAM block. From the information in the NAM block, the Loader determines the amount of Data Storage and Common Storage to be reserved, and, in addition, the amount of core necessary to load the program. This amount of core is reserved while the NAM block is being processed. During a Loader Operation, it is possible that the amount of core required for storage of the programs to be loaded exceeds the capacity of available core. If such is the case, the following occurs during the Loading Operation:

The programs are loaded in the manner described in the 1st paragraph of this item. When the NAM block of "PROGRAMX" is read, it is determined that in order to reserve enough core in which to load "PROGRAMX" the capacity of Available Core will be exceeded. The contents of Available Core at this time consists of an absolute record of all those programs loaded during this operation prior to "PROGRAMX". The Loader will write this information onto the mass storage device containing the Scratch area.

Following this "PROGRAMX" and all those programs loaded subsequent to "PROGRAMX" will be loaded into the same block of available core. As the NAM block of each program is read, space is reserved in core for the program to be loaded. If while loading "PROGRAMY", the capacity of Available Core should be exceeded a second time, this procedure is repeated. All those programs loaded prior to "PROGRAMY" and starting with "PROGRAMX" are written as an absolute record onto the mass storage unit containing the Scratch area. This second record is written at the end of the 1st.

The entire procedure repeats itself until the last program is loaded in. When the load of the last program is complete, whatever remains in Available core is written as absolute record onto the mass storage device containing the scratch area at the end of the information already written into this scratch area. The entire information in the scratch area is then read as an absolute record back into core starting at the lowest (toward 0) address in Available core,

($F7)+1.

This information consisting of all the programs loaded during the Loader Operation, when read into core will overlay that area of unprotected core occupied by the Loader Tables and/or the Loader. That area of core occupied by the Loader Table and the Loader is considered to be an "Extension of Available Core" or more simply "Extended Core". Concerning Loader Operations, the block of core referred to as Available Core may also at times be referred to as "Load Time Core". Concerning execution of the programs loaded by the loader, that block of unprotected core from ($F7)+1 to ($F6)-1 may also be referred to as "Execution Time Core".

During a Loader Operation, the Loader maintains 2 sets of address counters. One set is used to reference locations in Load Time Core and the other to reference locations in Execution Time Core. They are respectively referred to as the Load Time and the Execution Time address counters.

If at some point during a Loading Operation, part of the Command Sequences of the program loaded is on mass storage and part is held in Load Time Core, the relocationship between the two sets of address counters is as follows:

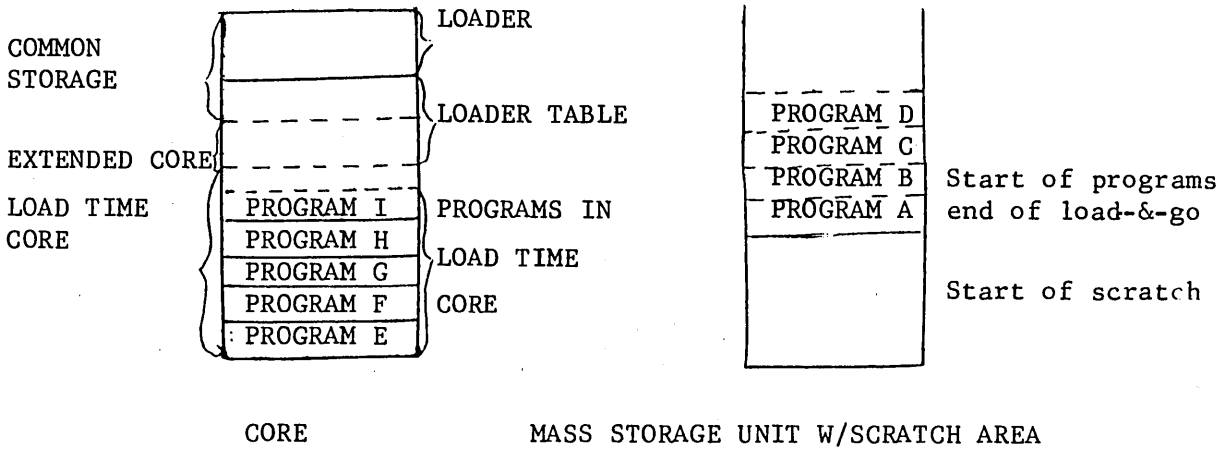
EXECUTION TIME ADDRESS COUNTER - LOAD TIME ADDRESS COUNTER = MASS STORAGE WORD COUNT.

If throughout the entire Loading Operation, the entire Command Sequence of the program loaded is held within Load Time Core, the values for the two sets of address counters are equal. (If a Data Storage block reservation has been made during the Loader Operation the relationship between the two sets of address counters changes. Refer to item 28.3.2.13.

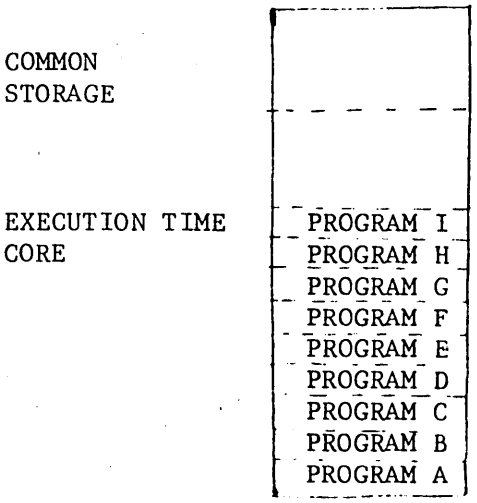
The Loader will not load a program if the core it occupies at Execution Time extends beyond the upper limit of unprotected core. If a Common Storage block reservation has been made during the Loading Operation, the relocation base for Common Storage effectively becomes the "upper limit of Execution Time Core".

The information to be recorded on mass storage is placed in the scratch area. The 1st sector in the scratch area is the beginning of this information unless the scratch area also contains Load-and-Go storage. In this case the location in the communication region $E4 contains the scratch sector number for the end of the load-and-go information. Then the starting sector number for the information placed in the scratch area by the Loader is equal to the 1st sector number of the scratch area + ($E4). The locations $C0 and $C1 contain the starting sector number for the scratch area.

3913.2.9 Diagrams of Unprotected Core Showing Extension of Available Core



The above diagram illustrates allocations of unprotected core and mass storage unit scratch area during loading operation.



The above illustration shows the programs in core following the Loading Operation. That area of occupied by the Loader Table is now overlaid by the last of programs by the Loader.

3913.2.2.10 Odd Sectors

When the Loader transfers information from Load Time Core to the Scratch area on mass storage (in the manner described in item 28.3.2.8), the information is transferred in integer multiples of 96 words, or - the information is transferred in "even sectors". (The word count per mass storage sector is 96.) If the number of words to be transferred from Load Time Core will hereafter be referred to as the "odd sector".

As an illustration, K is the number of words in Load Time Core such that

$$K = N*96 + M$$

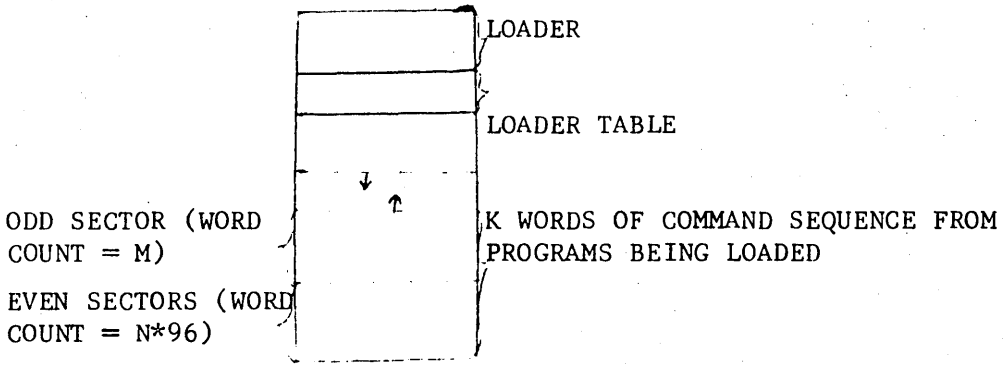
where N is an integer and

$$1 \leq M \leq 95.$$

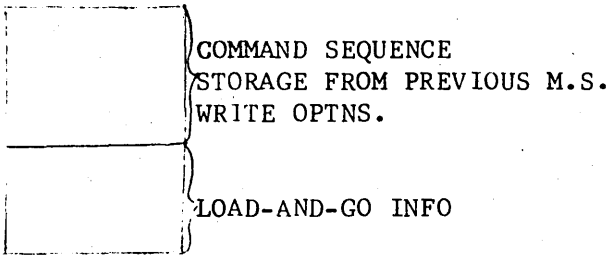
When the information transfer occurs, the first N*96 words in Load Time Core are written in the scratch area on mass storage. The starting sector for this write operation is the one after the last sector containing information from this Load Time Core as a result of a previous write operation. If this is the 1st such mass storage write operation to occur as a result of loading a program, the starting sector is the 1st sector in the scratch area. (In the event there is Load-and-Go information in the scratch area, the starting sector is immediately after the last sector containing Load-and-Go information.)

The last M words are moved from their storage positions to locations in the lower end of Load Time Core. The next program to be loaded is stored immediately above the area in Load Time Core containing the odd sector. The procedure occurs as follows:

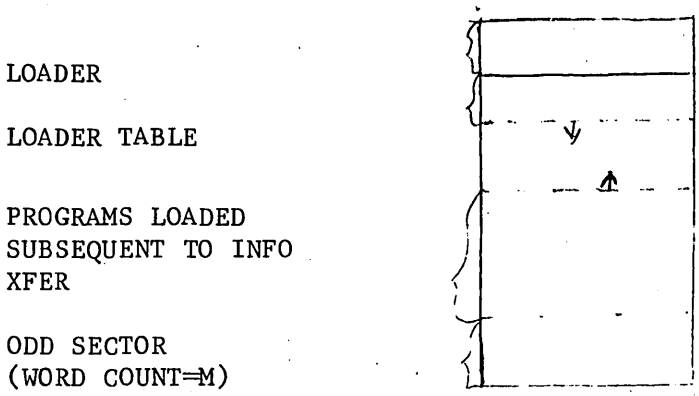
A. Unprotected Core Prior to Info XFER



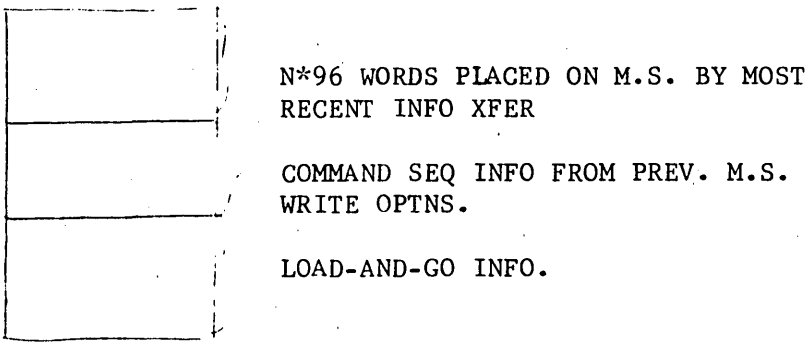
B. Scratch Area of Mass Storage prior to Info XFER



C. Unprotected Core Subsequent to Info XFER



D. Scratch Area of Mass Storage Subsequent to Info XFER



The next time a transfer of information to mass storage takes place, the information in the odd sector will appear on mass storage immediately after the last sector of information to be transferred to the scratch area.

If there is no odd sector, such that

$$K = N*96 \text{ and } M = 0.$$

the starting location for the next program to be loaded following the information transfer is the 1st location in Load Time Core. If the word count K in Load Time Core is less than 96, no information transfer will occur.

3.2.11 Allocation of Core for Data Storage Reservation

A Data Storage block reservation is made during a Loader Operation when the Loader begins to load a program which declares data and no previous Data Storage block reservation was made. Two areas of storage are set aside when a Data Storage block reservation is made. A Data Storage block reservation is made in the following way:

1. The information in Load Time Core is transferred to the scratch area on mass storage in the manner described by item 28.3.2.10.

2. A block of core is set aside at the lower end of Load Time Core. It is equal in length to the amount of Data Storage to be reserved. This area of core will hereafter be referred to as the Load Time Data Storage Buffer.

3. The information transfer in step 1 may have resulted in an "odd sector" remaining in Load Time Core. If so, the odd sector now occupies core locations in the area assigned as the Load Time Data Storage Buffer. The odd sector is to be moved to core locations immediately above the area of core reserved as the Load Time Data Storage Buffer.

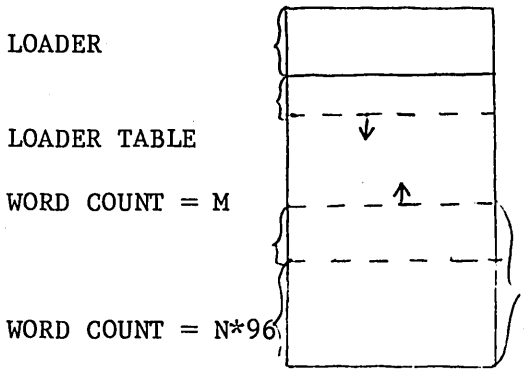
4. An area of core equal in length to the Load Time Data Storage Buffer is set aside in Load Time Core. This second block is situated immediately above the area of core now containing the odd sector. This block of core will hereafter be referred to as the "Execution Time Data Storage Block Reservation."

5. The area of core referred to as the Load Time Data Storage Buffer is no longer included as part of Available Core as defined by item 28.3.2.2. The new lower limit of Available Core (hence, the new lower limit of Load Time Core) becomes the last storage address of Load Time Core.

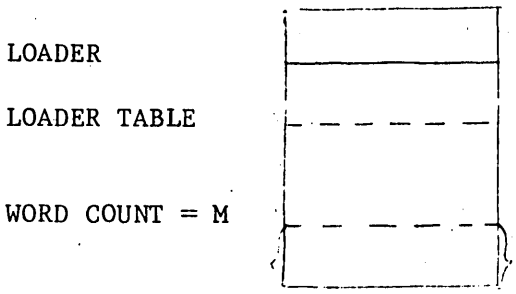
6. The 1st program to be loaded after a Data Storage Block Reservation is made is read into core starting immediately above the Execution Time Data Storage Relocation Base.

The area of core set aside for Data Storage may be preset by the information the Loader reads into core. Such information is placed by the Loader in the Load Time Data Storage Buffer. At the end of either a subroutine Load or a program load operation, the information in the Load Time Data Storage Buffer is transferred to the Execution Time Data Storage Block Reservation. The Load Time Data Storage Buffer is held in core throughout the Loader Operation. The Execution Time Data Storage Block Reservation is considered to be part of the command sequence of the programs being loaded. This block reservation may or may not be transferred to the scratch area on mass storage during a loading operation. Therefore the above mentioned transfer of information may be a core to mass storage transfer. However, if the Execution Time Data Storage Block Reservation remains in core at the end of program loading, the information transfer is core-to-core.

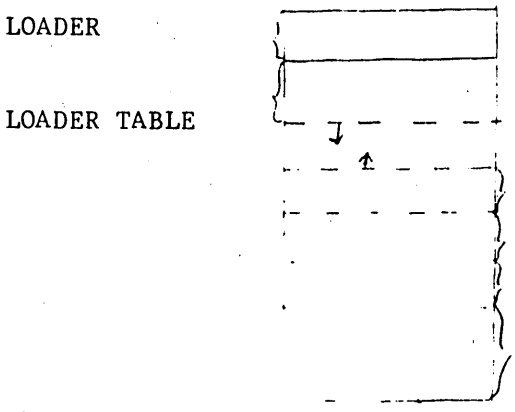
39 3.2.12 Diagrams of Unprotected Core with a Data Storage Block Reservation



The above diagram is unprotected core prior to the Data Storage Block Reservation.



The above diagram is unprotected core after N*96 words have been transferred to mass storage. The odd sector is moved into the bottom end of load time core.



The above diagram is unprotected core subsequent to the Data Storage Reservation.

39.3.2.13 Load Time and Execution Time Addresses

Item 28.3.2.8 showed the relationship between the load time and execution time address counters. This is true so long as there is no later storage block reservation. In general, the relationship of a load time storage address to an execution time storage address for a program is concerned, the following is true:

load time address = execution time address - word count on mass storage + length of Load Time Data Storage Buffer.

39.3.3 Input to the Loader

The Loader issues I/O requests to read formatted records in the Binary Mode. The records are not to exceed 120 characters of data in length. The binary records processed by the Loader will hereafter be referred to as relocatable binary input records. These are the type of records generated as binary output by the assembler. There are six types of relocatable binary input records (hereafter referred to as blocks.) Each block is identified by the 1st word (1st 2 data characters) of the record as follows:

NAME OF BLOCK	CONTENTS OF 1st WORD (HEX VALUES)
NAM	$2050
RBD	$4050
BZS	$6050
ENT	$8050
EXT	$A050
XFR	$C050

Binary records not recognizable to the Loader will be regarded as illegal input and will cause the Loader Operation to be terminated. In addition to the relocatable binary input records, the Loader will process ASCII input. Since the Loader reads only in binary mode, the input records must have "*" as the 1st character and must terminate with a carriage return. A space is accepted in place of a carriage return in the event the input device is a card reader. The two ASCII input blocks acceptable to the Loader are the HEX block which contains hex correction constants and the EOL block which indicates the end of Loader operation. They are identified by the 1st 3 characters in the block:

NAME OF BLOCK	IDENTIFICATION
HEX	*H
EOL	*T carriage return
EOL	*T space

Those ASCII records not recognizable to the Loader are regarded as Monitor Control Statements. Such a record will cause the Loading Operation to be interrupted. An exit is made to the monitor in order to process this statement.

3918 3.4 Exit from the Loader

Exit from the Loader to the Monitor is with an indirect return jump to the address in $EE. Immediately prior to exit from the Loader, the exit parameters are placed in the A, Q & I registers as illustrated below:

EXIT FROM LOADER

A	Q	I	Reason for Termination	Type of Loader Request
XFRADR	0	MSWDCT	EOL statement in input buffer	RBLOAD SUBRLD PROGLD
-0	0	MSWDCN	EOL statement in input buffer & no legal transfer address	RBLOAD
XFRADR	INPUT	MSWDCT	Control statement for monitor in input buffer	RBLOAD SUBRLD PROGLD
-0	INPUT	MSWDCT	Control statement for monitor in input buffer & no legal transfer address	RBLOAD
0	XFRADR	MSWDCT	Load operation terminated due to error which is irrecoverable	RBLOAD SUBRLD PROGLD
0	-0	MSWDCT	Load operation terminated due to irrecoverable error & there is no legal XFER address	RBLOAD SUBRLD PROGLD
-0	0	-0	Entry point table list	MAPS

MSWDCT means word count on mass storage*

XFRADR means transfer address

INPUT is name of 1st word address of relocatable binary input buffer

RBLOAD means relocatable binary load request

SUBRLD means library subroutine load request

~~RBLOAD~~ means program library load request

PRG-1 D

MAPS Means request for entry point table list out

*If MSWDCT = mass storage word count is zero, the entire program is contained within available core as defined in item 28.3.2.2.

39.4 OPERATION OF THE LOADER

The Loader is divided into 24 separate subprograms. The subprograms of the Loader are all non-optional. Each of the 24 subprograms is required for the Loader to operate. The Loader is divided into subprograms for no reason other than to facilitate the eaxe of handling at assembly time. Each subprogram may be assembled independently of the others. Within the source language, no subprogram has either a COM pseudo or a DAT pseudo, therefore the Loader requires neither common nor data storage reservations. The 24 relocatable binary programs produced by the assembler are linked together in the following way:

Entrance to one subprogram from another is made with either a 2 word jump instruction or a 2 word return jump instruction, either using the relative mode of addressing. Many subprograms of the Loader are coded as closed subroutines, each to be entered at a unique entry point location with a return jump instruction. Exit from a closed subroutine is made with a jump to the address placed in its entry point location by execution of the return jump instruction at the time it was entered. Other subprograms are coded as open ended routines. Many of these open ended routines have more than one entry point. Entrance to such an open ended routine at one of its declared entry point locations is made with a jump instruction. Exit from an open ended routine is made with a jump to a location external to another subprogram using a two word jump instruction and the relative mode of addressing. Within the source language of the Loader, the entry point names declared by the ENT pseudo in subprogram X are declared as external names by the EXT* pseudos in any of the other subprograms which reference this name. Because of the EXT* pseudo, the relocatable binary code generated by the assembler for instruction with external names as addresses provides for the relative mode of addressing. As the 24 subprograms are linked together into an absolute record at load time, the relative addressing feature is a major factor in providing for a "run anywhere" capability for operation of the Loader. (The "run anywhere" feature of the Loader is more meaningfully discussed in item 28.5.2.)

The order in which the subprograms of the Loader occur with respect to each other is optional with one exception. The subprogram whose name is LOAD (see item 28.5) must occur at the beginning of the Loader. This is the initialization module. Since entrance to the Loader from the monitor is made with a jump to its lowest (toward 0) location, the initialization module must occur at the bottom end of the block of core containing the Loader.

39.5 LOAD ROUTINE - INITIALIZATION

The Load routine is the subprogram concerned with initialization. Initialization is performed at the beginning of every Loader Operation. Entry to the Loader from the monitor is made by a jump to its lowest (toward 0) core address. There are three sections of this subprogram concerned with initialization named PART1, PART2, and PART3.

39.5.1 PART1

The 1st instruction to be executed upon entry to the Loader is a return jump to the location whose label is PART1. Prior to entering the Loader, the entrance parameters (see item 28.3.1) have been placed in the A&Q registers.

The only function of PART1 is to record the entrance parameters in temporary storage locations:

(A register) ⟶ AINP
(Q register) ⟶ QINP

An exit from PART1 is made by using the 2 word jump instruction stored at the location PARTSW and PARTSW+1. The jump is made to the part of the initialization procedure.

39.5.2 PART2

PART2 of initialization is performed only if the Loader is being used for the first time after it has been placed in core by the Job Processor. PART2 is entered from PART1 by the jump instruction at PARTSW and PARTSW+1. This jump instruction uses the relative mode of addressing. The 16 bit delta value coded into the second word of this instruction is equal to the value of the address expression

PART2-PARTSW-1.

At the completion of PART2, this 16 bit delta value will be replaced by the value for the expression

PART3-PARTSW-1.

In this manner, PART2 of initialization will be performed only during the first Loader Operation after the Loader is placed in core. For subsequent Loader Operations, control passes immediately from PART1 to PART3.

The functions of PART2 are as follows:

1. To establish the relocation base for the Constant Table.
2. To obtain and record the limits of unprotected core.
3. To enter the entry point name from the Table of Presets into the Loader Table.
4. To reset the address of the jump instruction at PARTSW such that PART2 will never be entered for execution during subsequent Loader Operations.

The Loader is a program with a "run anywhere" option. In other words, this program has the capability of being read as an absolute record, placed anywhere in memory and still being executable with little or no modification to the machine language of the Loader as it exists in the system library.

In "run anywhere" programs of primary concern are those instructions which reference memory. There is no problem in the case of either a one word instruction or a two word instruction which uses the relative mode of addressing. There are no two word instructions using the absolute mode of addressing unless the address is set at some time during program execution. If it is necessary to use absolute addressing, the absolute value of a core location may be obtained at any time by use of a return jump instruction:

```
RTJ*          BUFADR
BZS           DIRBUF (96)
LIBSEC NUM    $FFFF,$FFFF
BUFADR NUM    $FFFF
```

In the above sequence of code, the absolute value for the address "DIRBUF" will be placed in the location BUFADR upon execution of the return jump instruction.

39.5.2.1 Constant Table

During Loader Operations, the memory index or I register contains the relocation base for the Constant Table. The label assigned to the 1st location to be occupied by the constant table is CONTAB. The relocation base for the constant table is equal to the value of the address expression

"CONTAB-1".

The manner in which this relocation base is obtained is similar to the procedure described in the last paragraph of item 28.5.2. During the PART2 procedure of initialization it is necessary to compute the absolute values of other address constants. Referring

to the index of the Constant Table in item 28.5.2.1.2, the 29th storage location is the starting location of the input area for storage of relocatable binary input blocks. The execution time value of this address is 29+(I). This address is placed in the location 117+(I). The value for the starting address of this input area +1 = 30+(I) is placed in the location 11 & +(I). Also the value for the starting address -3 = 28 +(I) is placed in the location 122 + (I).

In addition to the above, the 23rd location of the constant table is set to the value of the starting address for the Loader. Upon entry to PART1 with a return jump (see item 28.5.1), the value = "LOAD+1" is placed in the location PART1. Therefore, the location 23+(I) is set to the value = (PART1)-1.

The 91st location of the Constant Table is reserved for the storage of the relocation base of the Constant Table. The name of this location is ISAV and it is declared as an entry point name by the LOAD subprogram. If at any time during a Loader Operation, the contents of the I register should be destroyed (as in the case of a status request), the original value of "CONTAB-1" may be restored to it. In order to do this in a subprogram other than LOAD, it is necessary to declare ISAV as an external name in the other module.

3945.2.1.1 Accessing Locations in the Constant Table

Whenever some routine addresses a location in the constant table, it does so with the following type of instruction:

OPC- n,I

where:

1. OPC represents a mnemonic for a memory reference instruction,
2. n represents an ordinal position in the constant table (position 1 up to position 255), and
3. I represents the memory index.

It is necessary that many locations in the Constant Table must be preset with specific values at the time the Loader is placed in core. Those locations in the Constant Table for which this does not apply are preset to a value of zero.

External pseudos must be included in the source language for LOAD for those locations in the constant table preset to address constants where these address constants refer to locations in other routines. An example is as follows:

SCAN is a closed subroutine but the entry to scan is in the constant table. The 145th to the 148th word in the constant table are set to the following code:

word 145	1	4	00
word 146	$FFFF		
word 147	1	8	00
word 147	hhhh		

where hhhh is an increment for a two word jump instruction using the relative mode of addressing. The source language is as follows:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP+	($7FFF)	RETURN FROM SCAN
	JMP	SCAN	JUMP TO SCAN
	EXT*	SCAN	

where the name SCAN is an entry point name in some other routine.

3.5.2.1.2 Table of Contents of Constant Table

ORDINAL	NAME	USE:
1	COMBAS	relocation base for common storage at execution time
2	DATBAS	relocation base for data storage at execution time
3	PROBAS	relocation base for program currently being loaded at execution time
4	COMLIM	highest address of common storage +1 at execution time
5	DATLIM	highest address of data storage +1 at execution time
6	CSQLIM	highest address of command sequence storage +1 at execution time
7	TABLIM	lowest address (toward 0) of loader table
8	ENDSW	=1 if last relocation byte in RBD or BZS block

ORDINAL	NAME	USE:
9	NGRLSW	negative address relocation switch
10	INPWRD	contains end of command sequence storage
11	INPREL	contains rel. flag for word of sequence storage in RBD and BZS block
12	XFRADR	contains transfer address of name from an XFR block
13	ENTPNT	contains address associated with name in EXT or ENT block
14	LINK	contains address associated with name in loader table
15	INPCTR	used to address core location of command sequence storage at load time
16	TABSCTR	tally of loader table entries
17	ENDINP	highest address in load time core for command sequence storage
18	BLANKS	ASCII code for spaces = $2020
19-21	SYMSTR	used by SCAN to record characters extracted from field containing ASCII input
22	SCANSW	used by SCAN to determine type of ASCII field
23	BASE	relocation base on which it is to operate of loader
24	WRDCNT	address or character reference counter
25	COUNT1	counter
26	BZSSW	used by subroutines common to RBDPRO & BZSPRO; =1 for BZS block; =0 for RBD block
27	BLKCNT	block counter
28	SW6	index counter
29-88	INPUT	input buffer
89	ASAV	temporary storage for A register
90	QSAV	temporary storage for Q register
91	ISAV	storage location for constant table relocation base

92-94	XFRNAM	storage of 6 character XFR name
95-98	NAME	secondary storage for field after SCAN operation
99-102	TABSCH	Loader Table search routine entrance
103-106	TABSTR	Loader Table storage routine entrance
107-110	CHPU	character pickup routine
111-112	BINASC	Storage of ASCII code for number conversion
113-116	PRINT3	error output routine (resume operation)
117	INPXCO	contains address constant = "INPUT" or 29+(I)
118	INPXC1	contains address constant = "INPUT" or 30+(I)
119-121	PRINT2	error output routine (Stop Operation)
122	INDXC3	contains address constant = "INPUT-3" or 26+(I)
123-124	NXTINP	jump instruction to read next input block
125-140	HEX CODES	ASCII codes for hex digits
141-144	ADJOVF	routine to perform address arithmetic (15 bits)
145-148	SCAN	SCAN routine entrance
149-152	CONVRT	binary to ASCII conversion routine entrance
153-156	PRINT4	print out routine (prints characters name and 4 digit hex address)
157-160	PRINT5	print out routine (prints 6 character names)
161	AINPUT	contains A register upon entry to loader
162	QINPUT	contains Q register upon entry to loader
163-166	LINK1	entrance to routine which patches in entry to loader
167-170	LINK2	entrance to routine which ties together 2 strings of link addresses
171	PRODIF	size of absolute record placed by loader on mass storage; the value in PRODIF is equal to the difference of (PROBAS)-(PROSTR)

172	START	start address for Mass Storage I/O
173-174	SECTNO	starting sector for mass storage I/O requests
175-176	INPLUN	logical unit number for input (L & A parameters for calling sequence)
177	DATLEN	length of Data Storage block reservation
178	DATRES	contains starting address in Load Time Core for Data Storabe block reservation
179	LOWCOR	Contains the starting address for Load Time Core (LOWCOR) = ($F7)+1 if (DATBAS) =0 (LOWCOR) = (DATCTR) if (DATBAS) ≠0
180	DATSTR	contains the starting address for the Load Time Data Storage Buffer
181	PROLIM	contains the address of the last word in the odd sector which occupies the lower 1-95 words of Load Time Core
182	DATCTR	contains the highest (toward $7FFF) address reserved for the Load Time Data Storage Buffer +1
183	CSQCTR	contains the highest (toward $7FFF) address reserved for command sequence storage at load time +1
184	DATDIF	contains a value equal to the difference: (DATBAS) - (DATSTR)
185	PROSTR	contains the relocation base for storage of a relocatable binary program in Load Time Core
186	DIFCON	temporary storage
187-190	COREXT	routine which writes absolute records produced from rel. binary input onto mass storage
191	LWRLIM	lowest unprotected location -1
192	UPRLIM	highest unprotected location +1
193	"INPUT"	start address for input device I/O operation
194-195	PROSEC	starting sector number if input device is mass storage

196	AHOLD	temporary storage for A register
197	QHOLD	temporary storage for Q register
198	SECTOR	"SECTOR" = 96_{10}
199-202	DPRADD	entrance to routine to compute absolute value of starting sector for mass storage I/O requests
203	DSECNO	relative value for starting number for storage of Load Time Data Storage Buffer in the scratch area of mass storage
204	TABSNO	relative value for starting sector number for storage of Loader Table in the scratch area of mass storage
205	E4SAVE	contains the relative value of the last sector number for Load-and-Go input.

Many of the constants in the Constant Table may be used by a subprogram in a manner other than that indicated by the Table of Contents. If so, an indication will be made in the maintenance documentation of the subprogram.

39 5.2.2 Core Request for Limits of Unprotected Core

The A register and the Q register are set to zero, and a core request is made in order to obtain the limits of unprotected core. Upon return to the Loader following completion of the request, the A register contains the upper limit and the Q the lower limit of unprotected core.

The upper limit contained in A is recorded at the locations in the Constant Table UPRLIM and COMLIM. The lower limit is read at LWRLIM AND LOWCOR.

39 5.2.3 Processing Table of Presets

The entry point names and addresses in the Table-of-Presets are entered into the Loader Table as entry point names. Item 28.5.2.3.1 is a description of the Loader Table while 28.5.2.3.2 is a description of the Table of Presets.

39 5.2.3.1 Loader Table

The Loader Table comprises a list of entry points and externals processed during a sequence of Loader Requests prior to job execution. The Loader Table begins immediately below the base address of the Loader and is expanded backward (toward 0) in memory during a loading operation.

A single entry in the Loader consists of 5 words. Words 1, 2 and 3 consist of the ASCII code for a name of up to 6 alphanumeric characters. If fewer than six characters are used, the unused character positions are filled with the ASCII code for spaces. If the entry contains an entry point name, word 4 holds the associated entry point address. (Word 4 is positive in value.)

If the entry contains an external name, word 4 contains the one's complement for the associated link address.

Word 5 of the Loader Table entry serves as a pointer. If a name appears in more than one entry in the Loader Table, word 5 of the most recent entry contains the address which points to word 4 of the previous entry. The 1st entry to be made in the Loader Talbe for this name contains a "-0" in word 5. If a name appears in only one entry of the Loader Table, word 5 of this entry is set to a "-0".

The TABSTR routine (see item 28.8.6) is used for the purpose of making Loader Table entries.

39.5.2.3.2 Table of Presets

The Table-of-Presets is located in protected core. The starting address for the Table-of-Presets is recorded at the communication region address of $F2 while $F1 contains the table length.

The format of an entry in the Table of Presets contains the entry point name and address for a program in the core resident portion of the operation system.

When an entry point name from the Table of Presets is entered into the Loader Table, -

1. WORD5 of the Loader Table entry is set to -0,
2. The sign bit of WORD2 of the entry is set to a 1.

If a program is loaded that uses an external name to match the name from the Table-of-Presets, the sign bit of WORD2 for the matching Loader Table entry is reset to zero. If the entry point name from the Table-of-Presets is not referenced as an external by any of the programs loaded, the sign bit remains set at the completion of loading. This name will not appear on the list entry points produced during a memory map operation. (See item 28.6.10.)

39.5.3 PART3

PART3 consists of all initialization required for each Loader request made prior to ⁻ob execution. One such operation will be to transfer (AINP) and (QINP) to locations in the respective constant table AINPUT and QINPUT. Once placed in the constant table, then this information will be more easily accessed by other routines comprising the loader. Another is a CORE REQUEST in which the Loader

asks for the limits of core available to it for storage of relocatable binary inputs. The lower address in the Q register will be recorded as the execution time relocation base for the next program to be loaded in the location PROBAS & CSQLIM. The constants PROSTR and CSQCTR are set to the value for the load time relocation base of the next program to be loaded. The value of this relocation base is equal to -

execution time relocation base + length of Data Block Reservation - word count on mass storage

or

(PROBAS) + (DATLEN) - (PRODIF) PROSTR

The contents of the location $E4 is the relative value for the last sector number in the scratch area containing the load-and-go input. PART3 records this number at the location $E4SAVE. If there is no load-and-go in the scratch area, ($E4) =1. In this case PART3 will set $E4 to zero. (Refer to item 28.8.10 and 28.8.11 for the significance of the number in $E4 and relative values for sector numbers.)

39.5.4 Subroutines Used by and External to Load

TABSTR is used for making Loader Table Entries. (Refer to item 28.8.6.)

39.5.6 Exit from LOAD

Exit from LOAD is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT	BRANCH	
	JMP	BRANCH	

39.6 BRANCH

The bits 0-3 of the location AINPUT determines the branching which occurs. According to the type of operation requested by the program calling the Loader, branching will occur to one of the four locations labeled as follows:

RBLOAD	(See item 28.6.4)
SBLOAD	(See item 28.6.6)
PROGLD	(See item 28.6.8)
MAPS	(See item 28.6.10)
ADRPRO	(See item 28.9.8)

3936.1 Constant Table Storage Referenced by BRANCH

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
AINPUT	161
ASAV	89
BINASC	111
BLANKS	18
BLKCTR	27
COMBAS	1
COMLIM	4
CONVRT	150
COUNT1	25
CSQLIM	6
DATBAS	2
DATCTR	182
DATLEN	177
DATLIM	5
DATRES	178
DATSTR	180
DPRADD	200
E4SAVE	205
ENDINP	17
ENTPNT	13
INPCTR	15
INPREL	11
INPUT	29
INPXCO	117
INPXC1	118
INPLUN	175
LINBUF	29
- same storage position as INPUT	
LINK	14
LOWCOR	179
LWRLIM	191
PRINT3	114
PRINT4	154
PRINT5	158
PRODIF	171
PROSEC	194
QINPUT	162
QSAV	90
SCAN	146
SCANSW	22
SECTNO	173
SECTOR	198
SW6	28
SYMSTR	19
TABCTR	16
TABLIM	7
TABSCH	100
TABSTR	104
WRDCNT	24
XFRNAM	92
XFRADR	12

3961.2 Communication Region Constants Used by BRANCH

	CONSTANT	LOCATION
1.	$7FFF	MASK1 = $42
2.	ordinal for mass storage device containing scratch area	$B3
3.	number of 1st sector in scratch area on mass storage	$C0 and $C1
4.	ordinal for mass storage device containing Program Library and Program Library Directory	$C2
5.	sector number for beginning of Program Library Directory on mass storage unit containing Program Library	$C3 and $C4
6.	Sector number for end of load-and-go in scratch area on mass storage	$E4
7.	address for return to monitor	$EE
8.	entry to operating system for making monitor requests	$F4
9.	ordinal for standard input device for system	$F9

3961.3 Entrance to BRANCH

The BRANCH routine is entered directly from PART3 of the LOAD routine. The BRANCH is entered by the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	BRANCH	

where the name BRANCH is declared as an entry point in the BRANCH routine and as an external in the LOAD routine.

396.4 RBLOAD - Relocatable Binary Loading

The sequence of code for this branch begins at the location whose label is RBLOAD. The Loader selects the input device for the Relocatable Binary Load Operation according to the information in bits 4-15 of the location in the constant table. The name of this location is AINPUT. The Loader will do one of the three steps below in order to select an input device:

1. If bit 15 of AINPUT is a 1, bits 4-14 of AINPUT are ignored and -

   a) the address "$F9" is placed in INPLUN, and
   b) the location INPLUN+1 is set to a 2.

   The input device selected in this case is the standard input medium for the system. The location $F9 contains the ordinal for this device.

2. If bit 15 of AINPUT is a 0 and bits 4-14 = ($B3) then -

   a) the address "$B3" is placed in INPLUN, and
   b) the location INPLUN+1 is set to a 2.

   The input device selected in this case is the scratch unit for the system. The location $B3 contains the ordinal for this device.

3. If bit 15 of AINPUT is a 0 and ($B3) ≠ bits 4-14 of AINPUT, bits 4-14 of AINPUT contain the ordinal for the input device such that -

   a) bits 4-14 of AINPUT are recorded in bits 0-10 of INPLUN, and
   b) the location INPLUN+1 is set to 0.

The locations INPLUN and INPLUN+1 are located in the constant table. The contents of INPLUN and INPLUN+1 become the "L" and "A" parameters to be inserted into a parameter list for the read requests made by the IDRIV subroutine. (Refer to item 28.7.1.) For information concerning parameter lists in I/O request, refer to item 6.4 in the Operating System ERS.

If the input device selected by RBLOAD is the mass storage device containing the scratch area, the locations PROSEC and PROSEC+1 are set to the sector number for the 1st sector in the scratch area. If the input device is a mass storage unit, it is because the Loader Operation involves load-and-go input and the 1st sector in the scratch area is the start of the programs to be loaded. The locations PROSEC and PROSEC+1 are the 193rd and 194th locations of the constant table. Bits 0-14 of PROSEC contain the most significant half and bits 0-14 of PROSEC+1 the least significant half of the starting sector number. If the input device is not a mass storage unit these two locations contain zeros.

Next a return jump is made to the beginning of the routine which performs the actual loading operation. The name of this routine is LOADER. (See item 29.1.) The name LOADER is declared as an entry point within the LOADER routine and as an external in the BRANCH routine. A return to the BRANCH routine from LOADER will be made when one of the following occurs:

1. An EOL input block had been read resulting in the termination of the Loading Operation. Upon the return to BRANCH -

   (A reg.) = -0
   (Q reg.) = 0

2. An operating system control statement has been read resulting in the termination of the Loading Operation. Upon return to BRANCH -

   (A reg.) = -0
   (Q reg.) = 1st word address for storage of the control statement.

3. An unrecoverable error condition has occurred. Upon return to BRANCH -

   (A reg.) = 0
   (Q reg.) = -0

   Upon return to BRANCH, these values are recorded in the constant table as follows:

   (A) ⟶ AINPUT
   (Q) ⟶ QINPUT

3966.5 Exit from RBLOAD

A return jump is made to the LIMSET routine to record the new temporary limits of core. (See item 28.6.12.1.) Another return jump is made to the GETADR subroutine to obtain the transfer address resulting from this loading operation. (See item 28.6.12.2.)

In addition, the following occurs upon exit from RBLOAD:

1. The location $EE contains the address for returning from the Loader to the monitor. This address is recorded in the second word of a 2 word return jump instruction using absolute addressing. The label assigned to the 1st of the 2 locations containing this instruction is LDXIT1.

2. If the Loading Operation did not involve Load-and-Go input, the location $E4 contains a zero. If ($E4) = 0, its original value contained in E4SAVE is restored to it.

3. If (AINPUT) ≠ 0, the Loader Operation was not terminated due to unrecoverable errors. In this case the following values are placed in A, Q & I:

(XFRADR) ⟶ A
(QINPUT) ⟶ Q
(PRODIF) ⟷ I

where

a) (XFRADR) = core location for legal transfer address of program; = -0 if no legal transfer address is indicated.

b) (QINPUT) = 0 if Loader Operation terminated due to EOL input block or = core address for storage of monitor control statement if operation terminated by the occurrance of such a statement in the Loader input.

c) (PRODIF) = word length of command sequence placed on mass storage during Loading Operation. This word count is 0 if the space required for the programs read in by the Loader does not exceed the capacity of Load Time Core.

In the event the Loader Operation was terminated due to an unrecoverable error, (AINPUT) = 0. The A, Q & I registers will be set to the following values:

0 ⟶ A
(XFRADR) ⟶ Q
(PRODIF) ⟶ I

A return jump is made to the monitor from LDXIT1.

396.6 SBLOAD - Subroutine Loading from Program Library

The sequence of code for this branch begins at the location whose name is SBLOAD. The input device for this operation is the mass storage unit containing the Program Library. The sequence of events are as follows:

1. Make initial check of Loader Table for entries containing unpatched externals. (An unpatched external is an external name for which no program has been read on containing an entry point name to match.)

2. A search is made of the Program Library Directory for a name to match each unpatched external in the Loader Table. For each matching name encountered in the Program Library Directory, the corresponding program is loaded from the Program Library.

3. At the completion of step 2, a second search is made of the Loader Table for entries with unpatched externals. An error

condition exists if any unprotected externals remain in the Loader Table. The appropriate error indication is printed followed by a list of the unpatched externals. This information is printed on the list output medium.

4. At the completion of Step 3, the operator has the option to intervene at the console. He may instruct the Loader to complete the operation and prepare for job execution, to look up any unpatched externals in the core resident directory, or to terminate the load and suppress job execution.

   If the Loader is to look up in the core resident directory, it goes back to step 2 with this exception: instead of searching the Program Library Directory, the Loader searches the core resident directory for entry points.

   If the Loader is to complete the operation, proceed to step 5.

5. The Loader sets the new limits of core as a result of the Loader Operation.

6. If no data has been declared at any time during a Loader Operation, proceed to step 6. If data had been declared a transfer of information occurs as follows: The contents of the Load Time Data Storage Buffer is transferred to the area set aside as the Execution Time Data Storage Block Reservation.

7. The Loader obtains the transfer aedress with which to enter the program for execution, and returns to the monitor.

39.6.6.1 Checking Loader Table for Unpatched Externals

Checking Loader Table entries for unpatched externals is accomplished with a program loop. The sequence of code for this loop begins at the address assigned the label TABCHK. Within this loop there are two word jump instructions using the relative mode of addressing. The delta values for the second word of either jump instruction are placed there during program execution.

Located at the addresses SW5 and SW5+1 is one of the two jump instructions. The execution of this instruction is the manner in which an exit is made from this loop. With respect to step 3 in item 28.6.6, this instruction operates as follows:

1. A jump will be made to SW5A, if no unpatched externals occur during a second check of the Loader Table. This may be regarded as the "normal exit" from the loop which checks for unpatched externals in the Loader Table.

2. A jump will be made to SW5B if at least one unpatched external occurs during a second Loader Table check. This may be regarded as the error exit from the loop.

Located at the addresses SW4 and SW4+1 is the other of the two jump instructions. If there are no unpatched externals in the Loader Table, this instruction is never executed.

The execution of the jump instruction at SW4 and SW4+1 occurs only when an unpatched external is encountered during a search of the Loader Table:

1. With respect to step 1 of item 28.6.6, a jump will be made to the location SUBRLD upon the 1st occurrance of an unpatched external during an initial Loader Table check.

2. With respect to step 3 of item 28.6.6, a jump will be made to the location SW4A upon the 1st occurrance of an unpatched external during the second Loader Table check.

3. With respect to step 3 of item 28.6.6, a jump will be made to the location SW4B for each occurrance of an unpatched external during a Loader Table check subsequent to the 1st occurrance.

In the case of the jump instruction at SW4 and SW4+1, a jump will be made to the location SW4A if the location SW4+1 contains a 16 bit delta equal to the value of the address expression:

SW4A - SW4 -1.

Henceforth, this will be referred to as "setting the switch SW4 to the SW4A position" or "setting SW4 to SW4A". This will be indicated symbolically as -

"SW4A ⟶ SW4

In general -

"X" ⟶ Y

means set the location Y to the value "X" or set the switch Y to the X position.

When making the Loader Table check, the entries are examined beginning with the 1st entry and working downward through core. (The 1st entry in the Loader Table occupies the five words immediately preceding the base address of the Loader.)

The location TABCTR contains the number of Loader Table entries. The location SW6 is set to the word length of the Loader Table:

(TABCTR) * 5 ⟶ SW6

The location TABLIM contains the base address of the Loader Table. Each time the TABCHK loop looks at a Loader Table entry, it reduces the index counter SW6 by 5. After the TABCHK loop checks the last entry in the Loader Table, SW6 is reduced to a negative value. Therefore, the test for completion is -

"if (SW6) < 0, execute jump at SW5".

The TABCHK loop looks at the fourth word of the Loader Table entry to determine if it is an unpatched external. The test for an unpatched external is:

"if ((TABLIM) + (SW6) + 3) <0, execute jump at SW4".

3?6.6.2 Initial Loader Table Check

Prior to starting the initial Loader Table check for unpatched externals, the following occurs:

"SUBRLD" ⟶ SW4

"SW5A" ⟶ SW5

Upon the 1st occurrence of an unpatched external, a jump is made at SW4 to the address SUBRLD. Otherwise a jump is made at SW5 to SW5A. (See item 28.6.6.6.)

3?6.6.3 Searching Program Library Directory for Unpatched Externals

The Program Library Directory is recorded on the mass storage unit containing the Program Library. The Program Library is divided into 96 word segments (96 words per sector) as follows:

1. The last 2 words (95th and 96th words) of each sector in the directory is a link which points to the next sector in the directory. (The directory does not necessarily occupy consecutive sectors on mass storage.)
2. The 94th word of a sector in the directory indicates the number of unused words in the 96 word segment if this is the last 96 word segment in the directory. This number will be an integer multiple of 5 (5 words in a directory entry) such that -

   $0 \leq 5n \leq 85$,

   where n = 0, 1, 2, ... 17.
3. The 91st, 92nd and 93rd locations in the sector are not used.
4. A 5 word directory entry contains the following information:

   a. WORD1 to WORD3.

      The six character name for this entry is stored 2 ASCII characters to a word. If (WORD1) = 0, the entry has been deleted.

   b. WORD4 & WORD5

      If the entry contains an entry point name for a program in the library, the last two words of the entry contain the starting sector number in the library for

this program. (This is the sector number where the NAM block for the relocatable binary input is stored.) If the entry contains a file name, the 4th word of the entry contains the one's complement of the file in sectors, and the 5th word contains the starting sector for the file in the library. This sector number is 15 bits in length whereas, sector numbers for programs are 30 bits in length, 15 bits per word of storage.

A search is made starting at the beginning of the program library directory progressing toward the end. The locations in the communication region $C3 and $C4 contain the starting sector for the Program Library Directory.

There is a maximum of 18 entries per sector.

The number in locations $C3 and $C4 is recorded in SECTNO and SECTNO+1. A return jump is made to the MDRIV routine (see item 28.7.3) to read the 1st 96 word segment on the Program Library Directory. One sector is read into the block of locations starting DIRBUF and terminating at DIRBUF+95. The location INPCTR is set to the 1st word address of the 96 word input area:

"DIRBUF" ⟶ INPCTR

Immediately upon return from MDRIV, the link to the next 96 word segment of the directory is recorded:

(DIRBUF+94) ⟶ LIBSEC
(DIRBUF+95) ⟶ LIBSEC+1

The Loader searches forward through the sector looking for an entry such that

1) ((INPCTR)) ≠ 0

meaning that the Directory entry is not deleted, and

2) ((INPCTR)+3) ≥ 0,

meaning that the directory entry is not a file name. Each time one of these conditions is not set, the address counter for the input area is increased:

(INPCTR) + 5 ⟶ INPCTR

The last entry in the sector has been encountered if -

"DIRBUF+90" - (INPCTR)=(DIRBUF+93)

where:

1. "DIRBUF+90" = address following last entry in sector.
2. (INPCTR) = 1st word address of next sector entry.
3. (DIRBUF+93) = number of unused core locations in sector.

When the last entry in DIRBUF has been encountered, a test is made to determine if this is not the last 96 word segment in the directory. It is not the last segment if either

$$(LIBSEC) \neq 0$$

or

$$(LIBSEC+1) \neq 0$$

The next segment in the directory is read and the entire procedure repeated by recording the sector number for the next 96 word directory segment -

$$(LIBSEC) \longrightarrow SECTNO$$
$$(LIBSEC+1) \longrightarrow SECTNO+1$$

and reading the next directory segment using MDRIV.

If the last segment of the directory has been encountered such that

$$(LIBSEC) = (LIBSEC+1) = 0,$$

the end of the directory has been reached. Once the end of the directory has been reached, a jump is made to TABCHK for a second check of the Loader Table. (See item 28.6.6.4.)

If in the process of searching through the Program Library Directory, an entry is encountered such that -

$$((INPCTR)) \neq 0$$

and

$$((INPCTR)+3) \geq 0,$$

a search is made of the Loader Table using the TABSCH routine. (See item 28.8.5.) If upon return from TABSCH, $(SW6) < 0$, it means no entry with a name to match has been found in the Loader Table. If, upon return from TABSCH, $((TABLIM)+(SW6)+3) \geq 0$, it means that an entry with a matching name has been found in the Loader Table, but this entry has already been defined as an entry point name. In either case the address counter INPCTR is increased to look at the next entry in the directory -

$$(INPCTR)+5 \longrightarrow INPCTR.$$

If, as a result of a Loader Table search, an entry has been found such that -

$$\{SW6\} \geq 0$$

and

$$\{\{TABLIM\}+\{SW6\}+3\} < 0,$$

it means that there exists an external name in the Loader Table which references some entry point name contained in a library routine.

This routine must be loaded from the Program Library in order to satisfy this external name. {In the process of loading a routine to satisfy one external, it is possible that many others will have been satisfied as well.} In order to load this routine from the Program Library, the starting sector number is taken from the directory entry and recorded:

$$\{\{INPCTR\}+3\} \longrightarrow PROSEC$$

$$\{\{INPCTR\}+4\} \longrightarrow PROSEC+1$$

A return jump is made to LOADER {see item 29.1} in order to load this program.

Upon return from Loader, the exit parameters in the A and Q registers are recorded in AINPUT and QINPUT respectively. Each time a program is loaded from the Program Library, it is necessary to repeat the entire procedure of searching the Program Library Directory trying to match entry point names in the directory with any external names that may yet remain in the Loader Table.

This procedure is not complete until a complete pass has been made to the end of the Program Library Directory, and, in the process of so doing, no external names have been encountered in the Loader to match any entry point name in the directory. At this time any external names remaining in the Loader Table are regarded as unpatched externals. They will be detected during a second search of the Loader Table. {see item 28.6.6.5.}

39.6.6.4 Searching Core Resident Directory for Unpatched Externals

The Core Resident Entry Point {CREP} Directory is recorded on the mass storage unit when the system is initialized, in the same format as the Program Library Directory. The CREP directory has an entry for each

symbol declared an entry point when the core resident load is completed during system initialization. The last two words of each entry in the directory point to the same mass storage sector number. The CREP program begins at this sector.

The CREP program consists of a dummy NAM block, as many ENT blocks as necessary to accommodate all core resident entry points, and a dummy XFR block. Each ENT block contains up to 14 entry point names, and the absolute address of each name.

Neither the CREP directory nor CREP program are modified at any time, unless, of course, the system is rebuilt with the Initializer. The starting sector address of the CREP directory is saved in the two words just preceeding the System Library Directory. When the CREP directory is to be searched, this initial directory address is used instead of the first sector address of the Program Library Directory; but it is then processed as though it were a search of the Program Library Directory. In subroutine BRANCH this is accomplished by:

{{$EB}-3}+1 ⟶ LIBSEC

{{$EB}-3}+2 ⟶ LIBSEC+1

2 ⟶ AINPUT

and processing the routine SUBRLD normally.

39.6.6.5 Loader Table Check Subsequent to Loading Subroutines

A second check is made on the Loader Table to look for any remaining unpatched externals. Prior to beginning the second Loader Table check the switches SW4 and SW5 are set as follows:

°SW4A° ⟶ SW4

°SW5A° ⟶ SW5

If no unpatched externals occur during the second Loader Table check, a jump is at SW5 to SW5A. (See item 28.6.7.) However, upon the occurrence of the first unpatched external, a jump is made at SW4 to SW4A. At SW4A, the switches SW4 and SW5 are reset as follows:

SW4B ⟶ SW4

SW5B ⟶ SW5

Next, the error indication "E10" is printed using the PRINT3 subroutine. (See item 28.8.7.1.)

The next instruction to be executed is located at the address SW4B. This is a return jump to the PRINT5 subroutine (see item 28.7.7.4) to print the 6 character name of the unpatched external. Prior to entry to PRINT5, the location INPCTR must be set to the 1st word address of the Loader Table entry containing the unpatched external:

(INPCTR) = (TABLIM) + (SW6)

The location INPCTR is set to the 1st word address of each Loader Table entry as this entry is being checked for an unpatched external.

Following the printout of this name, a jump is made to the location TCHKJ1 where the check for additional unpatched externals continues. Upon the occurrence of all subsequent unpatched externals a jump is made at SW4 to the location SW4B where only the external names (and no error indication) is printed. Upon the completion of the 2nd Loader Table check, a jump is made at SW5 to the location SW5B. At SW5B an indication of "E" is printed on the typewriter using the CDRIV subroutine. (See item 28.7.2.)

When the break light lights up on the typewriter keyboard, the operator may enter one of two statements at the typewriter:

1. The statement "*T carriage return" instructs the Loader to terminate the Loader operation and abandon job execution. A zero is placed in the location AINPUT, and a jump is made to RBLXIT. At the location RBLXIT is the beginning of the process for exit from the Loader for a relocatable binary load operation. (See item 28.6.5.)
2. The statement "* carriage return" instructs the Loader to continue with the Loader Operation and prepare for job execution. A jump is made to location SW5A. (See item 28.6.7.)

39.6.7 Exit from SBLOAD

A jump is made to the LIMSET subroutine (see item 28.6.12.1) to record the new temporary limits of core.

Next a test is made to see if data has been declared by some program read in during the Loader Operation. If (DATBAS) ≠ 0, there exists a data block within the command sequence. The contents of the Load Time Data Storage buffer must be transferred to the space in the command sequence of the programs read in reserved as the Execution Time Data Storage Block Reservation. This is done in one of two ways:

1. If (PRODIF) = 0, the entire command sequence is contained within the limits of Load Time Core as follows:
   a. (DATBAS) = Beginning of Data during execution of Programs being loaded.
   b. (DATRES) = Beginning of Execution Time Data Storage Block Reservation in Load Time Core.
   c. (DATSTR) = Beginning of Load Time Data Storage Buffer.
   d. (DATLEN) = Length of Data Block.
   e. (LOWCOR)+1=Beginning of Command Sequence in Load Time Core.
   f. (CSQCTR) = Last address +1 of Command Sequence in Load Time Core.

The contents of the Load Time Data Storabe Buffer is moved (last word first) to the Execution Time Data Storage Block Reservation as follows:

((DATSTR)+(DATLEN)) ⟶ (DATRES)+(DATLEN)
((DATSTR)+(DATLEN)-1) ⟶ (DATRES)+(DATLEN)-1
⋮
((DATSTR)+1) ⟶ (DATRES)+1
((DATSTR)) ⟶ (DATRES)

Following this information transfer, each word in the command sequence is moved from its Load Time core location to its execution time core location:

((LOWCOR)+1) ⟶ (LWRLIM)+1
((LOWCOR)+2) ⟶ (LWRLIM)+2
⋮
((ENDINP)-1) ⟶ (ENDINP)-(DATLEN)-1

where (LOWCOR) = (LWRLIM)+(DATLEN) as of the time a data storage reservation is set aside by NAMPRO. (See item 28.9.2.) ENDINP contains the highest address in Load Time Core used for storage of command sequence input. (See item 28.9.5.4.)

2. If (PRODIF) ≠ 0, the command sequence of the programs read in by the Loader is contained in the scratch area on mass storage. It is necessary to (1) locate the sectors containing the Execution Time Data Storage Block reservation, (2) read them into core, (3) transfer into them the contents of the Load Time Data Storage Buffer, and (4) write them back out onto mass storage.

   In order to perform this operation, it is necessary to compute the value for the starting sector number using the DPRADD routine. (See item 28.8.11.) In order to read and write using the mass storage unit, it is necessary to use the MDRIV routine. (See item 28.7.3.)

At this time, it is necessary to obtain the transfer address to be used for entry into the program for execution. A return jump is made to the routine GETADR (see item 28.6.12.2) to pick up the transfer address. Upon return from GETADR, the location INPCTR contains the address "XFRNAM" which is the 1st of 3 locations containing the ASCII code for the 6 character name. The location XFRADR is the location containing the transfer address. If upon return from GETADR, (XFRADR) = -0, it is because no transfer name had occurred as a result of a Loader Operation or the transfer name in the last XFR block read by the Loader is an illegal one. A zero is stored in AINPUT. An error indication of "E13" will be printed using the PRINT3 subroutine. The transfer name will be printed following this error indication using the PRINT5 routine. In the event no transfer name was recorded at XFRNAM, 6 spaces will be printed on the line following the error indication.

The remainder of the exit procedure is as follows:

1. The location $EE contains the address for returning from the Loader to the monitor. This address is recorded in the second word of a 2 word jump instruction using the absolute addressing. The label assigned to the 1st of the 2 locations containing this instruction is LDXIT2.
2. If the Loading Operation did not involve Load-and-Go input, the location $E4 contains a zero. If ($E4) = 0, its original value contained in E4SAVE is restored to it.
3. If (AINPUT) ≠ 0, the Loader Operation was not terminated due to unrecoverable errors. In this case, the following values are placed in the A, Q & I registers:

   (XFRADR) ⟶ A
   (QINPUT) ⟶ Q
   (PRODIF) ⟶ I

where

a) (XFRADR) = core location for legal transfer address of programs or = -0 if no legal transfer address is indicated.

b) (QINPUT) = 0 if loader Operation terminated due to EOL input block or = core address for monitor control statement if operation terminated by the occurrence of such a statement in the Loader input.

c) (PRODIF) = word length of command sequence placed on mass storage during Loading Operation. This word count is zero if the space required for the programs read in by the Loader does not exceed the capacity of Load Time Core.

In the event the Loader Operation was not terminated due to an unrecoverable error, (AINPUT) = 0. The A, Q & I registers will be set to the following values:

0 ⟶ A
(XFRADR) ⟶ Q
(PRODIF) ⟶ I

4. A return jump is made to the monitor from LDXIT2.

39.6.8 PROGLD

The sequence of code for this branch begins at the location whose name is progld. Upon entry at PROGLD, the location QINPUT contains an address. Recorded at this address, the Loader expects to find the ASCII code for an "*" followed by up to 7 characters, the last of which must be a space or a carriage return.

PROGLD uses the SCAN routine (see item 28.8.1) to extract the program name from the storage area. In order to do this the location WRDCNT is set up as a character reference counter which means that -

1. Bits 1-15 of WRDCNT reference the word containing a character, and
2. Bit 0 of WRDCNT is to reference the right character of the right character (lists 0-7).

Therefore, prior to entering SCAN, WRDCNT is set to reference the 2nd character in the storage area whose starting address is contained in QINPUT:

(QINPUT)*2+1 ⟶ WRDCNT

also prior to entering SCAN, -

0 ⟶ SCANSW &
0 ⟶ A register.

Upon return from SCAN, the 3 locations SYMSTR, SYMSTR+1, & SYMSTR+2 contain the ASCII code for up to 6 characters, and the location (INPREL) contains a terminating character for the name. If the name has fewer than 6 characters, the unused character positions at the right end of the storage area SYMSTR to SYMSTR+3 are space filled.

Upon return from SCAN a search is made of the Loader Table for this name using the TABSCH routine. (See item 28.8.5.) If this name had been found in the Loader Table a jump is made to the TABCHK loop in the code concerned with loading subroutines from the library. If this name is in the Loader Table as an external, it will automatically be loaded from the Library as a subroutine. If this name appears in the Loader Table as an entry point, it is assumed that the appropriate routine containing this entry point has been read in. On the other hand, if upon return from TABSCH, the name has not been found in the Loader Table, a Loader Table entry is made using the TABSTR routine (see item 28.8.6) in which this name is entered as an external. A jump is made to SUBRLD and the process continues as if it were a subroutine load operation. (See item 28.6.6.)

In order to search the loader table for this name, prior to entering the TABSTR routine, the location INPCTR must be set such that -

"SYMSTR" $\longrightarrow$ INPCTR

If, upon return from TABSCH, the name has not been found -

(SW6) $<$ 0,

otherwise the address of the entry containing the name -

(TABLIM) + (SW6)

When entering this name in the Loader Table as an external, the following locations must be set:

"SYMSTR" $\longrightarrow$ INPCTR

-0 $\longrightarrow$ LINK

$7FFF $\longrightarrow$ ENTPNT

-1 $\longrightarrow$ SW6

If upon return from SCAN, the location INPREL does not contain a legal terminator such that -

(INPREL) $\neq$ ASCII code for space, or

(INPREL) $\neq$ $FF indicating carriage return,

an error condition exists. This is regarded by the Loader to be the same as an unpatched external. An error indication of "E10" is given using the PRINT3 set to the address in QINPUT, and a cancel character = $7F is inserted into bits 8-15 of ((QINPUT)). Using the LDRIV routine (see item 28.7.4), the 6 character name and the illegal terminator are printed on the list device. The locations SW4 and SW5 are set -

"SW4B" ⟶ SW4 & "SW5B" ⟶ SW5.

A jump is made to a location in the TABCHK loop to further check the Loader Table for unpatched externals. This process has already been described in item 28.6.6.5.

396.9 Exit from PROGLD

Once the legality of illegality of the input to PROGLD have been established, the Loader Operation continues as if it were a subroutine load operation. This was already indicated by item 28.6.8. Therefore, the exit procedure by which a return to the monitor is made, is that described by item 28.6.7. Exit from SBLOAD.

396.10 MAPS

The sequence of code for this branch begins at the location whose name is MAPS. A memory map consists of a listing of the contents of each loader table entry which contains an entry point name and an entry point address. It will be necessary to transfer an entry point name to the line-of-print buffer. The entry point address which consists of a 15 bit number must be converted to the ASCII code for 4 hex digits before it can be stored in the line-of-print buffer. The routine to cause this conversion has as its entry point a location named CONVRT.

The format for a line of print is as follows:

ENTRY POINT TABLE:

SSSS***COMSSHHHH

SSSS***DATSSHHHH

SSSSXXXXXXSShhhhSSSSXXXXXXSShhhhSSSSXXXXXXSShhhhSSSSXXXXXXSShhhhSSSS

etc.

where:

each "X" represents an alphanumeric character,

each "h" represents a hexadecimal digit,

each "S" represents a space, and

the address for an entry point name appears immediately to the right of the name.

The line "SSSS***COMSShhhh" is printed to indicate the Common Storage Relocation Base, and is omitted if there is no common storage block reservation. The line "SSSS***DATSShhhh" is printed to indicate the Data Storage Relocation Base and is omitted if there is no data storage block reservation. An entry point name in the Loader Table will not be printed if either -

(1) It appears in a duplicate Loader Table entry, or

(2) It is an entry point name from the Table of Presets and no program had been loaded containing an external which references this name.

If the above condition 1 exists, it is detected by the fact that WORD3 from a duplicate Loader Table entry for an entry point name contains a 0. If the above condition 2 exists, it is detected by the fact that there is a sign bit of 1 in WORD2 of a Loader Table entry meaning that this is an entry point name from the Table of Presets. This sign bit will be made 0 when a program is loaded which contains an external which references this entry point name.

With respect to vertical spacing, 4 rows preceed and follow the heading. The lines of print are single spaced. A line of print contains up to 4 entry point names and addresses. If fewer than 4 appear on a line, the right end of the line-of-print buffer is filled with spaces. The output device for producing memory maps is the standard list device.

39.6.11 Exit from MAPS

Exit from MAPS is made as follows:

1. The location $EE contains the address for returning from the Loader to the monitor. This address is stored in the second word of a 2 word jump instruction using absolute addressing. The label assigned to the 1st of the 2 locations containing this instruction is LDXIT3.
2. If the loading operation did not involve Load-and-Go input, the location $E4 contains a zero. If ($E4) ≠ 0, its original value contained in E4SAVE is restored to it.
3. The exit parameters are placed in the A, Q & I register as follows:

   -0 ⟶ A

   0 ⟶ Q

   -0 ⟶ I
4. A return jump to the monitor is made from LDXIT3.

3950.6.12 Subroutines Used by and Internal to BRANCH

There are two subroutines used by and internal to the BRANCH subprogram:

1. LIMSET
2. GETADR

3950.6.12.1 LIMSET

The purpose of the LIMSET subroutine is to set the new temporary limits of unprotected core. Within the LIMSET routine, the A & Q registers are set to the limit addresses, and a core request is issued in order to record these addresses in the locations $EC and $ED. The limit address supplied for this core request are as follows:

1. The lower limit which is placed in the Q register is equal to the last execution time storage address for the command sequence read in during this Loader operation.

   (CSQLIM) - 1 $\longrightarrow$ Q reg.

2. The upper limit address is either the upper limit of unprotected core or the relocation base for common storage -

   a. The upper limit address is the top of unprotected core if there is no Common Storage Block Reservation.

      If (COMBAS) = 0

      (COMLIM) $\longrightarrow$ A reg.

   b. If there is a Common Storage Block Reservation, the upper limit address for the Core Request is the relocation base for common storage -

      If (COMBAS) $\neq$ 0

      (COMBAS) $\longrightarrow$ A reg.

The LIMSET routine is entered with a return jump to the location whose label is LIMSET. Exit from the LIMSET routine is with a jump to the address placed in the location LIMSET upon entry to the routine.

3950.6.12.2 GETADR

The GETADR subroutine is entered by a return jump to the location whose label is GETADR. The GETADR subroutine will search the Loader Table for a name to match the transfer name which appeared in the last XFR block read by the Loader. The transfer name contains up to 6 characters and is stored at XFRNAM, XFRNAM+1 & XFRNAM+2. The transfer name is legal if -

a. A Loader Table entry contains a name to match this transfer name, and -

b. the address in WORD4 of this Loader Table entry is

an entry point address.

If the transfer name is legal, the entry point address from the Loader Table entry is placed in the location XFRADR. The transfer name is illegal if either -

a. There exists no Loader Table entry which contains a name to match this transfer name, or -

b. the address in WORD4 of this Loader Table entry is not an entry point name.

If the transfer name is illegal, a value = -0 is placed in the location XFRADR. An exit is made from the GETADR subroutine by a jump to the address placed in the location GETADR upon entry.

39.6.13 Subroutines Used by and External to BRANCH

Subroutines used by and external to the BRANCH subprogram are as follows:

PRINT3
PRINT4
PRINT5
TABSCH
TABSTR
SCAN
CONVRT
CDRIV
MDRIV
LDRIV
LOADER
DPRADD

39.6.13.1 PRINT3 (See item 28.8.7.1.)

PRINT3 is used for printing error messages on the list output device. Prior to entering PRINT3, the ASCII code for the error message is placed on the A register. The information placed in the A register consists of either 2 characters of 1 character followed by a space. Entrance is made to the PRINT3 subroutine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	PRINT3(114)	
	RTJ-	PRINT3,I	

The output device used by the PRINT3 subroutine is the standard list device for the System. This is the device whose equipment table ordinal appears in the communication region address $FD.

39.6.13.2 PRINT4 (See item 28.7.3.)

The PRINT4 routine is used by MAPS to print the names and relocation

bases for common and/or data storage reservation while listing the entry point table. Prior to entering the PRINT4 subroutine, the A register is set to the binary value for the address to be printed and the Q register is set to the 1st word address of the output area. (The PRINT4 subroutine will cause the value in the A register to be converted to the ASCII code for 4 hex digit and recorded in the output area.) The PRINT4 subroutine is entered by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	PRINT4(154)	
	RTJ-	PRINT4,I	

The output device used by the PRINT4 subroutine is the same as that used by PRINT3 (item 28.10.12.1).

39.6.13.4 TABSCH (See item 28.8.5)

Given the storage area for a 6 character name, the TABSCH subroutine is used to search for this name in the Loader Table. Prior to entering the TABSCH subroutine, the location INPCTR is set to the 1st word address of storage for the given 6 character name. Upon exit from TABSCH -

a. if no Loader Table entry had been encountered with a name to match the given name, then -

(SW6) _ 0, or

b. If a Loader Table entry with a matching name had been located, then -

(SW6) _ 0, and

(TABLIM)-(SW6) = 1st word address of this loader table entry

Entry to TABSCH is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	TABSCH(100)	
	RTJ-	TABSCH,I	

39.6.13.5 TABSTR (See item 28.8.6)

The TABSTR routine is used to enter a name and address into the Loader Table. Prior to entering the TABSTR routine -

a. the location INPCTR is set to the 1st word storage address of a given 6 character name,

b. the location ENTPNT contains the address for this name,

c. the location LINK is set to a -0 if this is the 1st Loader Table entry for this name (or, if not, LINK is set to the address of word 3 of a previous Loader Table entry containing this name), and

d. the location SW6 is either -

1. set to zero if the given name is an entry point, or
2. set to a -0 if the given name is an external.

3956.13.6 SCAN (See item 28.8.1)

The SCAN subroutine is used by the BRANCH subprogram during a program load operation. SCAN is used to extract a name from the storage area. The 1st character of the storage area occurs in the left half of the 1st word and is assumed to be the character *. The second character of the storage area is in the right half of the 1st word and is assumed to be the 1st character of the name. Therefore, the location used as the character reference counter, WRDCNT, is set as follows -

1. bits 1-15 contains the 1st word address of the input area, and
2. bit 0 is set to a 1 to reference the right half of the 1st word.

Upon return from SCAN, the locations SYMSTR, SYMSTR+1, and SYMSTR+2 contain up to 6 characters for the name. The terminal character is stored in the right half of the location INPREL. If the name contains fewer than 6 characters, the unused character positions in SYMSTR to SYMSTR+2 are filled with spaces. If the name contains 6 characters or more, the location INPREL contains the 7th character of the input area as the termination character.

The SCAN subroutine is entered by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	SCAN(146)	
	RTJ-	SCAN,I	

3956.13.7 CONVRT (See item 28.8.4)

The CONVRT subroutine is used by MAPS to convert a 16 bit binary number to the ASCII code for hex digits. The 16 bit number to be converted is the entry point address from a Loader Table entry. Prior to entering CONVRT, the binary number is placed in the A register. Upon return from SCAN, the locations BINASC and BINASC+1 contain the ASCII code for the 4 hexadecimal digits.

CONVRT is entered by execution of the following instruction.

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	CONVRT(150)	
	RTJ-	CONVRT,I	

NOTE: This subroutine is also used by PRINT4.

39.6.13.8 CDRIV (See item 28.7.2)

CDRIV is the subroutine used for I/O operations on the typewriter. Prior to entering the CDRIV routine, the A register is at the 1st word address of the storage area, and either -

1. the Q register is set to the number of words for a write operation, or -
2. the Q register is set to zero for a read operation.

Entrance to the CDRIV subroutine is made by execution of the following instruction -

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	MDRIV	
	RTJ	MDRIV	

39.6.13.9 MDRIV (See item 28.7.3)

MDRIV is the subroutine used for I/O operations on a mass storage device. Prior to entering MDRIV, the A register is set as follows:

1. (A) = starting address if the device to be used is the mass storage unit containing the scratch area.
2. (A) = one's complement of the starting address if the device to be used is the mass storage unit containing the Program Library.

Also, prior to entering MDRIV, the Q register is set as follows:

1. (Q) = number of words if the operation is to read.
2. (Q) = one's complement of the number of words if the operation is to write.

Entry to the MDRIV routine is by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	MDRIV	
	RTJ	MDRIV	

39.6.13.10 LDRIV

LDRIV is the subroutine which is used for output on the standard list device for the system. Prior to entry to LDRIV, the A register is set to the starting address and Q to the number of words for the output operation. Entry to LDRIV is made by execution of the following subroutine:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	LDRIV	
	RTJ	LDRIV	

39.6.13.11 LOADER (See item 29.1)

LOADER is the common subroutine used by RBLOAD, SBLOAD, and PROGLD to perform the functions of reading and processing relocatable binary input blocks. Entry to LOADER is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	LOADER	
	RTJ	LOADER	

39.6.13.12 DPRADD (See item 28.8.11)

DPRADD is the subroutine which computes the absolute value of the starting sector number for an I/O operation using a mass storage device. Prior to entering the DPRADD routine -

1. Bits 0-14 of A contain a 15 bit number which is the relative of the starting sector number.
2. Bits 0-14 of $E4 contain the relative value of the sector number (in the scratch area) which is the end of the load-and-go input.
3. The locations $C0 and $C1 contain the absolute value (30 bits) of the 1st sector number in the scratch area as follows:
   a. The most significant half of the number is in bits 0-14 of $C0, and
   b. The least significant half is in bits 0-14 of $C1.

Upon return from DRPADD, the absolute value of the starting sector number for the I/O operation is recorded in bits 0-14 of SECTNO (most significant half) and in bits 0-14 of SECTNO+1 (least significant half). This number is obtained by the following addition:

(A register) + ($E4) + ($C0,$C1)

Entry to DPRADD is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	DPRADD(200)	
	RTJ-	DPRADD,I	

39.7 I/O FOR LOADER OPERATIONS

There are four routines used by the Loader for I/O operations. Each of these routines is entered with a return jump to its 1st word address. The name given to the 1st word address of each routine is declared as an entry point name for that routine. Also, the entry point name for the routine is referenced as an external within the subprogram elsewhere in the Loader requiring the use of this routine for I/O.

Routines used for I/O operations expect in input parameters such information as the starting address, number of words, starting sector if the device is mass storage, etc. These input parameters are passed to each of these routines through the A register, the Q register, the I register and certain prescribed locations in the constant table. A routine used by the Loader for I/O will insert each of these parameters into the appropriate spots within a parameter list. The format of this parameter list is that used for a formal I/O request, and it is headed by a return jump instruction:

```
RTJ-    (IO)
EQU     IO($F4)
```

There are four routines used by the Loader for I/O operations. These entry point names are as follows:

IDRIV
CDRIV
MDRIV
LDRIV

39.7.1 IDRIV

The routine whose name is IDRIV is used to read either ASCII or binary formatted records from the input device. The binary records are those of the relocatable binary format produced by the 1700 Assembler.

39.7.1.1 Constant Table Storage Referenced by IDRIV

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
BININP	193
INPLUN	175
PROSEC	194

The constants are used as follows:

1. BININP — contains the starting address for a read operation using the input device.
2. INPLUN — is the name given to the 1st of 2 sequential locations which contain information pertaining to the logical unit number of the input device.

3. PROSEC — is the name given to the 1st of two locations which contain a sector number for the read operation. This is ignored if the input medium is not the mass storage device.

39 7.1.2 Communication Region Constants Used by IDRIV

CONSTANT	LOCATION
$7FFF	$42 = MASK1
$8000	$21 = MASK2

39.7.1.3 Entry to IDRIV

The input parameters to IDRIV are as follows:

1. If (A) is positive, the mode is ASCII, and (A) represents the starting address. If (A) is negative, the mode is binary, and (A) represents the one's complement of the starting address.
2. The information pertaining to the logical unit number for the input device is stored at the 175th and 176th locations in the constant table. The names for these locations are INPLUN and INPLUN+1.
3. If the input medium is the mass storage device, PROSEC and PROSEC+1 contain the sector number for the record to be read.

Entry is made to IDRIV with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	IDRIV	
	RTJ	IDRIV	

39.7.1.4 IDRIV I/O Request

The input parameters for the I/O requests are handled in the following way:

1. Starting Address

   If (A) is positive upon entry to IDRIV, the contents of the A register is recorded as the starting address. If (A) is negative upon entry to IDRIV, the one's complement of the A register is recorded as the starting address. The starting address is recorded in the location specified by the S parameter in the calling sequence.

2. Word Count

   Since IDRIV is used primarily to read relocatable binary records produced by the 1700 Assembler, the value 60 will be placed in the parameter list for the I/O request as the N parameter. The value 60 represents the maximum size record that can be read from the input device.

3. Mode

If {A} is positive upon entry to IDRIV, the M parameter in the calling sequence is set for the ASCII mode of operation. If {A} is negative upon entry to IDRIV, the M parameter is set for the binary mode of operation.

4. Logical Unit Number

The information for the logical unit number by which the input device is referenced is taken from the constant table and inserted as parameters into the calling sequence. The contents of INPLUN is inserted as the L parameter and the contents of INPLUN+1 is inserted as the A parameter.

Prior to issuing the I/O request, the completion flag IFLAG is initially set to a 3 bit value of 001. The location in the parameter list reserved for the thread {label = THREAD} contains a zero. The I/O request is issued by execution of the instruction preceding the parameter list -

RTJ- {IO}

Once the operation is initiated, the location THREAD is set non-zero. The IDRIV routine will wait in a loop until completion occurs. While the input operation is in progress, the location THREAD remains at a non-zero value. Upon completion of the operation, THREAD is reset to a zero value. Also, IFLAG is set to a 3 bit value which reflects the state of the input operation upon its completion.

Upon completion of the input operation, a status request is made. As a result of this status request, bits 4-10 of the Q register will indicate whether the input device was or was not a mass storage unit. If the device had been a mass storage unit, the 30 bit sector number stored in PROSEC and PROSEC+1 would have been increased by 1.

If the Loader is loading from the Program Library, then INPUT+58 and INPUT+59 contain the next sector address of the present program. Both LIBEDT and the SYSTEM INITIALIZER insert a thread into each block on the Program Library to allow available sectors to be used as efficiently as possible. Thus, a program need not be on contiguous sectors in the Program Library. The next sector address is saved in PROSEC and PROSEC+1, then the last three words of the input block are cleared to zero before returning to the caller.

Also, as a result of this status request, the A register will contain the information pertaining to the hardware status. This information will be recorded at a location STATUS+1. If, upon completion of the input operation, bit 2 of the location IFLAG is a 0, the operation is regarded as being error free.

The A register is set to a minus zero value, and a jump is made to the exit from IDRIV.

If, upon completion of an input operation, bit 2 of the location IFLAG is a 1, the input operation was terminated due to error. In this case, IDRIV looks at bit 9 of the hardware status information stored at STATUS+1. If bit 9=0, it is assumed that an unrecoverable error condition such as parity exists as a result of the read operation. The A register is set to a 0, and a jump is made to exit from IDRIV. If bit 9 ≠ 0, an error condition due to motion failure has occurred. The A register is set to +1 to indicate the alarm condition and a jump is made to the exit from IDRIV.

3960 7.1.4 Exit from IDRIV

Exit from IDRIV is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	(IDRIV)	

3960 7.2 CDRIV

The routine whose name is SCRIV is used for either input or output operations on the communication device.

3960 7.2.1 Entry to CDRIV

The input parameters to CDRIV are as follows:

1. The A register contains the starting address. The A register is always assumed to be positive by the CDRIV routine.
2. The word count is in the Q register if CDRIV is to perform an output operation. The Q register is otherwise zero.

Entry to CDRIV is made with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	CDRIV	
	RTJ	CDRIV	

3960 7.2.2 CDRIV I/O Request

The input parameters for the CDRIV requests are handled in the following way:

1. Starting Address

   The starting address in the A register is recorded in the location specified by the S parameter of the calling sequence.

2. Request Code

   If the Q register is now zero, the request code is in the calling sequence is set for an output operation. If the Q register is zero, the request code is set for an input operation.

3. Word Count

   If the Q register is non zero, it is assumed by CDRIV to be positive. The value in the Q register is then inserted into the parameter list in the calling sequence as the N parameter. If the Q register is zero, a value of 60 is inserted into the parameter list as the N parameter.

Prior to issuing the I/O request, the completion flag CFLAG is set to a 3 bit value of 001. The location in the parameter list reserved for the thread (label = THREAD) contains a 0. The I/O request is issued by execution of the instruction which preceeds the parameter list -

RTJ- (IO)

Once the operation begins, the location THREAD is set to a non zero value. The CDRIV routine will wait on a loop until completion occurs. While the I/O operation is in progress, the location THREAD will remain at a non zero value. Upon completion of the operation, the location THREAD will be reset to zero. Also, upon completion of the I/O operation, the location CFLAG is set to a 3 bit value which reflects the state of the I/O operation upon its completion.

If the operation was for output, the status is ignored. The A register is set to a minus zero value, and a jump is made to the exit from CDRIV. If the operation was for output, bit 2 of the location CFLAG will be zero if the operation is error free. The A register will be set to a minus zero, and a jump will be made to the exit from CDRIV. If bit 2 of the location CFLAG is a 1, theoperation was terminated due to an unrecoverable I/O error. The A register will be set to a zero, and a jump will be made to the exit from CDRIV.

39.7.2.3 Exit from CDRIV

Exit from CDRIV is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	(CDRIV)	

39.7.3 MDRIV

The routine whose name is MDRIV is used for either input or output operations on the mass storage device.

39.7.3.1 NAME USED IN DOCUMENTATION STORAGE POSITION IN CONSTANT TABLE

START 172

START is the location which contains the starting address for the mass storage I/O.

39.7.3.2 Communication Region Constants Used by the MDRIV Routine

CONSTANT	LOCATION
$8000	$21 = MASK2

39.7.3.3 Entry to MDRIV

The input parameter to MDRIV are as follows:

1. The A register contains the starting address. The A

register is positive if the device used is the mass storage unit containing the scratch area. The A register is negative if the device is the mass storage unit containing the Program Library. If (A) is negative, the one's complement of (A) is the starting address.

2. The Q register is positive if the I/O operation is for reading from mass storage, and it is negative if the operation is for writing. The absolute value for the Q register represents the word count.

3. The locations SECTNO & SECTNO+1 contain the double precision number of 30 bits for the starting sector. The least significant half is stored in SECTNO+1, and the most significant half in SECTNO.

Entry to MDRIV is made with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	MDRIV	
	RTJ	MDRIV	

39.7.3.4 MDRIV I/O Requests

The input parameters for the I/O requests are handled in the following way:

1. Starting Address

   The starting address in the A register is recorded in the location specified by the S parameter in the calling sequence. If (A) is negative, the one's complement of A is recorded.

2. Request Code

   If Q is positive, the request code is set for a read operation. If Q is negative, the request code is set for a write operation.

3. Word Count

   If Q is positive, the value in Q is stored in the parameter list of the calling sequence as the N parameter. If Q is negative, the one's complement of the value in Q is recorded as the N parameter.

Prior to issuing the I/O request, the location in the parameter list reserved for use as a thread (label = THREAD) contains a zero. The I/O request is issued by execution of the instruction preceding the parameter list -

RTJ- (IO)

Once the operation begins, the location THREAD is set to a non-zero value. The MDRIV routine will wait in a loop until completion occurs.

While the I/O operation is in progress, the location THREAD remains at a non zero value. Upon completion of the operation, the location THREAD is reset to 0. Also, upon completion of the operation, a jump is made to the exit from MDRIV.

39.7.3.5 Exit from MDRIV

Exit from MDRIV is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	(MDRIV)	

39.7.4 LDRIV

The routine whose name is LDRIV is used for output on the list device.

39.7.4.1 Entry to LDRIV

The input parameters to LDRIV are as follows:

1. The A register contains the starting address. The A register is assumed to be positive by the LDRIV routine.
2. The Q register contains the word count. The Q register is assumed to be positive by the LDRIV routine.

Entry is made to LDRIV with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	LDRIV	
	RTJ	LDRIV	

39.7.4.2 LDRIV I/O Requests

The input parameters for the I/O requests are handled in the following way:

1. Starting Address

   The starting address in the A register is inserted into the address specified by the S parameter in the calling sequence.

2. Word Count

   The word count in the Q register is inserted into the parameter list of the calling sequence as the N parameter.

Prior to issuing an I/O request, the contents of the location in the parameter list word reserved for use as a thread (label = THREAD) contains a zero. The I/O request is issued by execution of the instruction which preceeds the parameter list -

```
RTJ     ($F4)
```

Once the operation is initiated, the location THREAD is set to a non zero value. The LDRIV routine will wait in a loop until completion occurs. While the I/O operation is in progress, location THREAD reamins at a non zero value. Upon completion of the operation the location THREAD is reset to zero. Also, upon comple tion of the operation, an exit is made from the LDRIV routine.

39.8.4.3 Exit from LDRIV

Exit is made from LDRIV by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	(LDRIV)	

39.8 SUBROUTINES ACCESSED VIA CONSTANT TABLE

The last paragraph of item 28.5.2.1.1 descriges how a subroutine may be accessed from any other routine in the loader with a return jump to a location in the constant table. The subroutines accessed in this manner are as follows:

SCAN
CHPU
ADJOVF
CONVRT
TABSCH
TABSTR
PRINT3
PRINT2 - STOP
PRINT4
PRINT5
LINK1
LINK2
COREXT
DPRADD

39.8.1 SCAN

The SCAN subroutine is used to extract a single element from an input block containing ASCII data. There are four types of elements contained in an input block.

1. Alphanumeric name,
2. Hexadecimal numbers,
3. Decimal numbers,
4. Null elements.

39.8.1.1 Constant Table Storage Referenced by SCAN

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
WRDCNT	24
COUNT1	25
SCANSW	22 (bits 0-3)
SYMSTR	19-21 & bits 4-15 of 22
BLKCNT	27
INPWRD	10
INPREL	11
BLANKS	18
NEGSW	9

Constant Storage is used as follows:

1. WRDCNT is used as a character reference counter. Prior to entering the SCAN subroutine, bits 1-15 of this location must be set to the storage address for the 1st character of a field to be extracted. Bit 0 is set to a zero if the 1st character occurs in the left half of the word and to a 1 if the 1st character is in the right half.

2. COUTN1 is used as a character counter. It is set to the one's complement of the maximum number of characters in a particular type of field may have.

3. SCANSW bit settings are as follows:

   a. If bit 0 of SCANSW is set to a 0 and the field being processed by SCAN contains a numeric operand, the number will be processed as a decimal number unless it is preceded by an "$". The number will then be processed as a hexadecimal number. If bit 0 of SCANSW is set to a 1 and the field being processed contains a numeric operand, the operand is processed as a hexadecimal number whether or not it is preceded by a "$". Bit 0 of SCANSW may be referred to elsewhere in the documentation by the name HEXSW. However, this name is not used in the coding for SCAN.

b. The field being processed by SCAN may have a leading algebraic sign of "+" only if bit 1 of SCANSW is set to a 1.

c. The field being processed by SCAN may have a leading algebraic sign of "-" only if bit 2 of SCANSW is set to a 1.

d. If the field being processed contains a numeric operand, SCAN will convert the ASCII code to a binary number. In addition, if bit 3 of SCANSW is set to a 1, the ASCII code for the digits of the numeric operand will be extracted from the input block.

4. SYMSTR, SYMSTR+1, SYMSTR+2 — is set to the ASCII code for the characters in the field being processed by SCAN. If the field contains a numeric operand, the location SYMSTR is set to zero. If bit 3 of SCANSW=1, the locations SYMSTR+1, SYMSTR+2 and bits 8-15 of SCANSW contain the ASCII code for the digits of the numeric operand.

5. BLKCNT — is set by SCAN to reference character positions in the block of words reserved for SYMSTR. Bits 1-15 are set to the address at which a character is to be stored. The character is stored in the left half of the word if bit 0 is set to a 0. The character is stored in the right half of the word if bit 0 is set to a 1.

6. INPWRD — is set by SCAN to the binary value of a numeric operand after its conversion from the ASCII code. (If the operand is not numeric; i.e., (SYMSTR) $\neq$ 0 the contents of INPWRD will be set to zero.)

7. INPREL — is set by SCAN to the ASCII code for the character terminating the field being processed. The character is stored in bits 0-7 while bits 8-15 are set to zeros.

8. BLANKS — contains the ASCII code for 2 spaces = $2020.

9. NEGSW — is set to a -1 if the operand in the field being processed is preceded by a legal leading algebraic sign of "-". It is set to a +1 if the operand is preceded by a legal algebraic sign of "+". It is set to 0 if the operand contains no legal leading algebraic sign. (A leading algebraic sign of "-" is legal if bit 2 of SCANSW=1, and that of "+" is legal if bit 1 of SCANSW=1.

39.8.1.2 Entry to SCAN

Prior to entry to SCAN, the 24th location of the constant table should be set to reference the field to storage position for the 1st character in storage of the field to be processed. The entire A register should be set to zero, or bits 8-15 set to 0 while bit 0-7 contain the ASCII code for a character. Entry is made to SCAN by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	SCAN	
	EQU	SCAN(144)	

The location in the constant table, NEGSW is set to zero. The locations SYMSTR, SYMSTR+1 and SYMSTR+2 are backrounded to the ASCII code for spaces. The constant $2020 is added to the contents of the location SCANSW and the sum replaces the contents.

If A is zero upon entry, the SCAN routine references storage for the 1st character in the field according to the contents of the location WRDCNT. If A is non zero (as described above), the SCAN routine will process bits 0-7 of the A register as the leading character of the field and reference storage for the second character.

39.8.1.3 Processing a Field Containing a Name

An alphanumeric name may contain a maximum of 6 characters. The characters from which a name may be composed are the alphabetics A-Z and the numerics 0-9. The leading character of a name must be one of the alphabetics A-Z. The last character of a name must be followed by one of the characters used as a terminator. This is any non-alphabetic character.

Upon exit from the SCAN subroutine thename is placed in the locations SYMSTR to SYMSTR+2. These locations are ordinals 19-21 within the constant table. If fewer than 6 characters are contained in the name, each of the unused character positions at the right end of the name is filled with the ASCII code for a space. The terminating character is stored in bits 0-7 of the location INPREL.

If there is excess number of characters, the locations SYMSTR to SYMSTR+2 contain the first 6 characters. Bits 0-7 of the location INPREL contain the 7th character. A transfer is made to the exit from SCAN.

39.8.1.4 Processing a Field Containing a Hexadecimal Number

A hexadecimal number contains a maximum of 4 hexadecimal digits. The characters from which a hexadecimal number may be composed consists of the number 0-9 and/or the letters A-F. Either a "$" must precede the hexadecimal number, or bit 0 of the location SCANSW must be a 1

upon entry to the SCAN subroutine. If neither of these conditions exists, then the following occurs:

1. If the first character of the element is one of the alphabetics A-F, the element will be processed as if it were an alphanumeric name. The terminating character will be either a non alphanumeric character or the 7th character of the name, whichever occurs sooner. (Refer to item 28.8.1.3.)
2. If the first character of the element is one of the numerics 0-9, the element will be processed as if it were a decimal number. The terminating character will be either a non decimal digit or the 6th digit of the number. (Refer to item 28.8.1.5.)

Upon exit from the SCAN subroutine, the location SYMSTR is set to zero, the location INPWRD contains the binary value for the hexadecimal number and the bits 0-7 of the location INPREL are set to the ASCII code for the terminating character. The terminating character is either the 1st non hexadecimal character of the field or the 5th hexadecimal digit of the numeric operand. If the number is preceded by a "-", the one's complement of this binary number is placed in the location INPWRD. If bit 3 of SCANSW=1, the locations SYMSTR+1, SYMSTR+2 and SCANSW are filled with the ASCII code for a dollar sign and followed by up to 4 hex digits for the numeric operand.

The terminating character for a hexadecimal number is either a character which is not a hexadecimal digit of the 5th hexadecimal digit in the number, which ever is first to occur.

If there is an excess number of digits, the location SYMSTR is set to zero, the location INPWRD will be set to the binary value of the hexadecimal number and the last character processed will be in bits 0-7 of INPREL. A transfer will be made to the exit from SCAN.

3968.1.5 Processing a Field Containing a Decimal Number

A decimal number contains a maximum of 5 decimal digits. The characters from which a decimal number may be composed consists of the number 0-9. The terminating character for a decimal number may be non decimal digit. Upon exit from the SCAN subroutine the location SYMSTR is set to zero, the location INPWRD contains the binary value for the decimal number and bits 0-7 contain the ASCII code for the terminating character. If the number is preceded by a "-", the one's complement of the binary number is placed in INPWRD. The locations SYMSTR+1, SYMSTR_2 and the bits 8-15 of SCANSW will contain the ASCII code for up to 5 decimal digits if bit 3 of SCANSW=1.

The value 65535 should be the largest decimal number used as a numeric operand. The SCAN routine will convert this decimal number to a 16 bit binary number of all ones (= hexadecimal number of $FFFF). A decimal number greater than 65535 will be truncated at its most significant

end when it is converted to binary. Only the 16 least significant bits will be recorded by the SCAN routine.

Decimal numbers with leading algebraic signs are processed by SCAN in the same manner as hexadecimal numbers with leading algebraic signs.

39.8.1.6 Processing a Field Containing a Null Element

If the leading character of an element is neither an alphabetic character nor a leading algebraic sign, the element is a null element.

A transfer is made immediately to the eixt from SCAN with -

1. The locations SYMSTR, SYMSTR+1 & SYMSTR+2 set to a value of $2020,
2. The location INPWRD set to a value of 0, and
3. Bits 0-7 of the location INPREL set to the ASCII code for non alphanumeric character (null element).

39.8.1.7 Processing Fields With Leading Algebraic Signs

Upon entry to SCAN, a leading algebraic sign in a field may be either the first character in storage (if A = 0), or it may be the character in bits 0-7 of the A register.

An algebraic sign is a terminator if it is not at the beginning of a field. A leading algebraic sign of "+" is legal only if bit 2 of SCANSW=1. A leading algebraic sign of "=" is legal only if bit 1 of SCANSW=1. An illegal leading algebraic sign acts as a terminator for a field which has no characters. An exit is made from SCAN with SYMSTR, SYMSTR+1 & SYMSTR+2 filled with spaces, and with NGRLSW and INPWRD set to 0, The terminal character is in bits 0-7 of INPREL.

39.8.1.8 Exit from SCAN

An exit from SCAN is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	SCNXIT,I	
	EQU	SCNXIT(143)	

39.8.1.9 Subroutines Used by and External to SCAN

CHPU extracts 1 character from the input and stores it in INPREL. The character reference counter, WRDCNT, is increased by 1 prior to each return from CHPU. Entry is made to CHPU. Entry is made to CHPU with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	CHPU,I	
	EQU	CHPU(108)	

39.8.1.10 Subroutines Used by and Internal to SCAN

GETBIN receives the ASCII code for a digit and returns the 4 bit binary equivalent in the accumulator. The

NUMBER routine examines a character from the field. If the character is a digit between 0 and 9, the ASCII code is placed in A and an exit is made. If HEXSW is set to 1 and the character is between A and F, the ASCII code is placed in A and an exit is made. If the character is not a digit, A is set to -0 and a transfer is made to the exit.

LETTER examines character from the field. If the character is alphanumeric, the ASCII is placed in A and a transfer is made to the exit. If the character is not alphanumeric, A is made =0, and a transfer is made to the exit.

STRCHR stores a character from a name or a number in the block of locations beginning with SYMSTR. BLKCNT, the character storage reference counter is increased by 1 after the storage of each character.

Each of the above subroutines is entered with a return transfer to the location whose name is the same as the subroutine name. Exit is made with a jump to the address in the entry point of the subroutine.

39.8.2 CHPU

The CHPU subroutine is used to extract a single character from the input block containing the ASCII data.

39.8.2.1 Constant Table Storage Referenced by CHPU

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
WRDCNT	24
INPREL	11

Constant storage is used as follows:

1. WRDCNT is used as a character reference counter.
2. INPREL is where the character is stored when it is extracted from the input information block by the CHPU routine.

3978.2.3 Entry to CHPU

Entry is made to the CHPU routine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	CHPU,I	
	EQU	CHPU(108)	

Upon entry to CHPU, bits (1-15) of the location WRDCNT reference a core location. If bit 0 of the WRDCNT is a zero, the left character (bits 8-15) in this core location is extracted and placed in bits (0-7) of the location INPREL. If bit 0 of the WRDCNT is a one, the right character (bits 0-7) are extracted and placed in bits (0-7) of the location INPREL. The contents of WRDCNT is increased by 1 and a transfer is made to the exit.

3978.2.4 Exit from CHPU

Exit from CHPU is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CHPXIT,I	
	EQU	CHPXIT(107)	

3978.3 ADJOVF

The ADJOVF is a subroutine which performs 15 bit address arithmetic.

3978.3.1 Constant Table Storage Referenced by ADJOVF

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
AHOLD	196
QHOLD	197

The locations ASAV and QSAV are used for storage of the A and Q registers respectively upon entry to the ADJOVF routine.

3978.3.2 Communication Region Constants Used by ADJOVF

CONSTANT	LOCATION
$7FFF	$42 = MASK1
$8000	$21 = MASK2

3978.3.3 Entry to ADJOVF

Prior to entry to the ADJOVF subroutine, the two operands for the address arithmetic are placed in the A and Q register. Entry to the ADJOVF subroutine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	ADJOVF,I	
	EQU	ADJOVF(142)	

39.8.3.4 Address Arithmetic

This subroutine is used primarily to add the relative value for a relocatable address to a relocation base. Upon entry to ADJOVF, the A register contains the relocatable address in bits 0-14, and the Q register contains the relocation base. Bit 15 of A is not included in the address arithmetic. Upon exit from ADJOVF, bits 0-14 of the A register contain the 15 bit result for the address arithmetic together with the original setting for the sign bit.

39.8.5.3 Address Arithmetic for Positive Address Relocation

The address arithmetic for positive address relocation is:

relative value + relocation base = absolute value

Address arithmetic for positive address relocation is performed if the Q register is positive upon entry to ADJOVF.

The sign bit of A is extracted, recorded and replaced with a 0. The A and Q registers are summed and the result is placed in A. Since this is 15 bit wrap around arithmetic, the sign bit of A is handled as if it were an end around carry in one's complement arithmetic. The result is in bits 0-14 of A. The original sign bit is inserted into bit 15 of A. A transfer is made to the exit from ADJOVF.

39.8.3.6 Address Arithmetic for Negative Address Relocation

The address arithmetic for negative address relocation is:

-(relative value + relocation base) absolute address

Address arithmetic for negative address relocation is performed if the Q register is negative upon entry to ADJOVF. Bits 0-15 of the A register are set to the one's complement of the relocatable address.

The sign bit of A is extracted, recorded and replaced with a 1. The A and Q registers are complemented, and address arithmetic is performed in the manner described in the last paragraph of item 28.8.3.5. The result in bits 0-14 of A is then recomplemented and the original sign bit is inserted into bit 15 of A. A transfer is made to the exit from ADJOVF.

39.8.3.7 Exit from ADJOVF

Exit is made from the ADJOVF subroutine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	ADJXIT,I	
	EQU	ADJXIT(141)	

39.8.4 CONVRT

The CONVRT routine is used to replace a 16 bit binary number with the ASCII code for 4 characters representing the digits of the equivalent hexadecimal number.

39.8.4.1 Constant Table Storage Referenced by CONVRT

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
COUNT1	25
BINASC	111-112
AHOLD	196
HEX CODES	125-140

Constant storage is used as follows:

1. COUNT1 is used as counter for program execution within a loop.
2. BINASC consisting of 2 words, is filled with the ASCII code for exactly 4 characters representing hex digits.
3. AHOLD is used to hold the bits for the binary number during the conversion process.
4. HEXDIG is the beginning of 16 consecutive locations in the constant table. End of these locations contains the ASCII code for a character representing a hexadecimal digit.

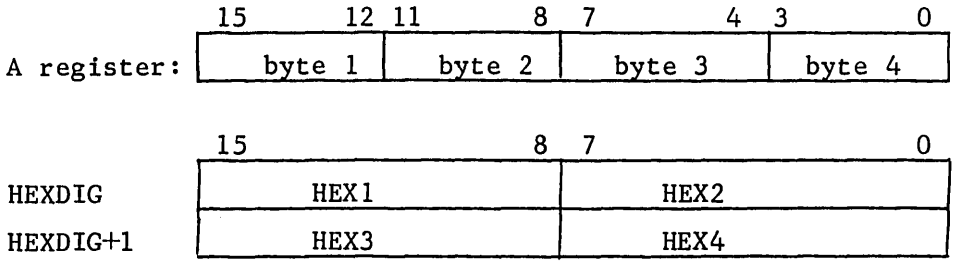
39.8.4.2 Entry to CONVRT

Prior to entry to CONVRT, the A register is set to the binary number for which the ASCII output for the equivalent hexadecimal number is to be generated. Entry is made to CONVRT by executing the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	CONVRT,I	
	EQU	CONVRT(148)	

39.8.4.3 CONVRT Operation

Upon entry to CONVRT, the A register contains a 16 bit binary number. This number is held in AHOLD during the CONVRT operation. The CONVRT eill extract 4 bits at a time from this number and replace them with the ASCII code for one of the hexadecimal digits. The four bit bytes are processed in a left to right order. The 4 ASCII corresponding to these bytes are stored in 2 consecutive words, 2 to a word.



3.8.4.4 Exit from CONVRT

The exit is made from CONVRT by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CNVXIT,I	
	EQU	CNFXIT(147)	

3.8.4.5 Subroutines Used by and Internal to CONVRT

GETHEX is used by CONVRT to replace a 4 bit byte in a binary number with the ASCII code for a hexadecimal digit. The contents of AHOLD are shifted left circular 4 upon entry to GETHEX. Bits 0-3 of AHOLD are placed in the A register. GETHEX will replace this value with the corresponding hex digit and a transfer is made to the exit.

GETHEX is entered with a return jump to the location by that name.

3.8.5 TABSCH

The TABSCH subroutine is used to search the Loader Table for an entry containing a name to match a name supplied by the calling program.

3.8.5.1 Constant Table Storage Referenced by TABSCH

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
COUNT1	25
TABLIM	7
TABCTR	16
SW6	28
AHOLD	196
INPCTR	15

Constant Storage is used as follows:

1. COUNT1 is used as a counter for program execution within a loop.
2. TABLIM contains the lowest (toward 0) core address occupied by the loader table.
3. TABCTR contains the number used as an index. This number is added to the base address of the Loader Table (contained in TABLIM), and the resultant address points to the 1st word of some entry in the Loader Table.
5. AHOLD is used as temporary storage for the A register during execution of the TABSCH subroutine.
6. INPCTR contains the first address of storage of the name being searched for in the Loader Table.

3978.5.2 Communication Region Constants Used by TABSCH

CONSTANT	LOCATION
$7FFF	$42 = MASK1

3978.5.3 Entry to TABSCH

Prior to entering TABSCH, the location INPCTR is set to the 1st word address of the sequential locations containing the name being searched for in the Loader Table. Entry is made to TABSCH with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	TABSCH,I	
	EQU	TABSCH(100)	

3978.5.4 Searching Operation

Upon entry to the TABSCH subroutine -

1. The location INPCTR contains the first word address of some entry in the input block.
2. The location TABCTR contains the number of entries in the loader table, and
3. the location TABLIM contains the first word address of the Loader Table.

The location TABCTR contains the number of Loader Table entries. If (TABCTR) = 0, there are no Loader Table Entries. The location SW6 is set to a -0, and a jump is made to the exit from the TABSCH routine.

The location TABLIM contains the base address of the Loader Table. The location SW6 is to be used as an index counter. When the value is SW6

is added to the address in TABLIM, the resulting address is the 1st word of a Loader Table entry. Initially, upon entry to TABSCH, the location SW6 is cleared to zero. TABSCH will compare bits 0-14 of the first 3 words or each entry of the loader table with bits 0-14 of the first 3 words of the entry from the input block as follows:

((INPCTR)) is compared with ((TABLIM)+(SW6))

((INPCTR)+1) is compared with ((TABLIM)+(SW6)+1)

((INPCTR)+2) is compared with ((TABLIM)+(SW6)+2)

If a comparison is made between the name is a Loader Table entry and the given name, a jump is made to the exit from TABSCH, Upon exit from TABSCH the address obtained by the addition

(TABLIM)+(SW6)

is the 1st core location of the Loader Table entry with the matching name.

Each time a comparison test with a Loader Table entry fails, the value in SW6 is increased in order to access the next Loader Table entry:

(SW6)+5 SW6

If comparison with each Loader Table entry has failed, the end of the Loader Table has been reached. When the end of the Loader Table has been reached, the value in SW6 is as follows:

(SW6) = (TABCTR) * 5

If the name from the input block entry does not match any name in the loader table, the location SW6 is set to -0, and a transfer is made to the exit address.

39.8.5.6 Exit from TABSCH

Exit from the TABSCH subroutine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	SCHXIT,I	
	EQU	SCHXIT(99)	

39.8.6 TABSTR

The TABSTR routine is used to make an entry into the Loader Table.

39.8.6.1 Constant Table Storage Referenced by TABSTR

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
TABLIM	7
TABCTR	16
DATCTR	182
CSQCTR	183

INPCTR	15
COUNT1	25
ENTPNT	13
LINK	14
SW6	28

Constant table storage is used as follows:

1. TABLIM — contains the lowest (toward 0) core address occupied by the Loader Table.
2. TABCTR — contains the number of entries in the Loader Table.
3. DATCTR — contains the upper limit address (last address +1) of the Load Time Data Storage Buffer.
4. CSQCTR — contains the upper limit address in Load Time Core (last address +1) for storage of command sequence read in by the Load.
5. INPCTR — contains the first address of storage of the name being stored in the Loader Table.
6. COUNT1 — is used as a counter for program execution within a loop.
7. ENTPNT — contains the absolute core address associated with the name to be stored in the Loader Table.
8. LINK — contains an address which serves as a pointer. If a name appears at least one other Loader Table entry, LINK contains the address which points to Word 4 of that entry. If it does not appear in some entry in the Loader Table, LINK contains a "-0". (See item 28.5.2.3.2 for Loader Table information.)
9. SW6 — is set to a 0 if the name referenced by INPCTR is to be recorded in the Loader Table as an entry point name. It is set to a -1 if the name referenced by INPCTR is to be recorded as an external name.

3978.6.2 Entry To TABSTR

Prior to entering TABSTR, the location INPCTR is set to the 1st word address of the sequential locations containing the name to be recorded in the Loader Table. Entry is made to TABSTR with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	TABSTR,I	
	EQU	TABSTR(104)	

39.8.6.3 Table Store Operation

Upon entry to TABSTR -

1. The location INPCTR contains the first word address of the name for the entry to be made in the Loader Table.
2. The location TABLIM contains the base address of the Loader Table.
3. The location ENTPNT contains the address for this name.
4. The location SW6 contains either a zero if (ENTPNT) is an entry point address or a -1 if (ENTPNT) is a link address.
5. The location LINK contains a -0 if the name referenced by INPCTR appears nowhere else in the Loader Table, or the address in LINK points to WORD4 of the most recent entry to the Loader Table containing the name referenced by INPCTR.

The TABSTR subroutine will first make sure that by adding another entry to the Loader Table, there will be no overflow of available core. Overflow of available core will result if -

$$(TABLIM) - 5 \geq (CSQCTR)$$

$$(TABLIM) - 5 \geq (DATCTR)$$

If either of these two conditions should arise, a return jump will be made to the PRINT3 error routine where an error indication is printed. A list will be made of all unpatched externals when a jump is made to TABCHK. The name TABCHK is declared as an external in TABSTR and as an entry point in the routine named BRANCH. (See item 28.6.6.5.) If neither of these conditions arises, the name whose first word address is in INPCTR is placed in the loader table. The entry is made as follows:

$(TABCTR) \cdot 1 \longrightarrow TABCTR$
$(TABLIM) - 5 \longrightarrow (TABLIM)$
$((INPCTR)) \longrightarrow (TABLIM)$
$((INPCTR)+1) \longrightarrow (TABLIM)+1$
$((INPCTR)+2) \longrightarrow (TABLIM)+2$

The address for the name is placed in the fourth word of the entry as follows:

$(ENTPNT) \longrightarrow (TABLIM)+3$ if $(SW6) = 0$, or -
$-(ENTPNT) \longrightarrow (TABLIM)+3$ if $(SW6) = 1$

The "pointer" is placed in the last word of the entry as follows:

$(LINK) \longrightarrow (TABLIM)+4$

3978.6.4 Exit from TABSTR

Following a loader table entry, exit is made from TABSTR with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	STRXIT,I	
	EQU	STRXIT(103)	

If overflow of core would have occurred as a result of the Loader Table entry, exit is made from the TABSTR routine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	TABCHK	

Where the name TABCHK is declared as an external in TABSTR and as an entry point name in the routine named BRANCH.

3978.7 LSTOUT

There are four routines used by other subprograms of the Loader as standard routines for output operations on the standard list device for the system. The four subroutines are:

PRINT3
PRINT2
PRINT4
PRINT5

3978.7.1 PRINT3

PRINT3 is a subroutine used for printing error messages. The error messages consist of the letter F followed by 1 or 2 hexadecimal digits. Operation within the Loading Procedure may be resumed immediately following the error message output.

3978.7.1.1 Entry to PRINT3

Prior to entering PRINT3, the A register is set to the ASCII code for two hexadecimal digits. (If a number for an error message contains only 1 digit, the digit will be in bits 8-15 of A. Bits 0-7 of A contain the ASCII code for a space.) PRINT3 is entered with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	PRINT3,I	
	EQU	PRINT3(114)	

3980.7.1.2 Error Message Output

Upon entry to PRINT3 the A register is stored at COMSGE+1 while the ASCII code for a space and the letter E is stored in COMSGE. The 1st word address of the error message output = "COMSGE" is placed in the A register. The length of the message is placed in the Q register. A return transfer is made to the routine CDRIV where the address CDRIV is declared as an external. CDRIV is the entry point for the I/O routine for the comment medium. (See item 28.7.2.)

3980.7.1.3 Exit from PRINT3

Exit from the PRINT3 routine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PR3XIT,I	
	EQU	PR3XIT(113)	

3980.8.7.1.4 Subroutines Used by and External to PRINT3

CDRIV is used for output on the comment medium. This routine is entered with a return jump to its entry point whose name is CDRIV.

3980.8.7.2 PRINT2

PRINT2 is a subroutine used for printing error messages. The error message consists of the letter E followed by 2 hexadecimal digits. The Loading Operation is terminated by entrance to PRINT2.

3980.8.7.2.1 Entry to PRINT2

The A register is set prior to entering PRINT2 in the same manner as described for PRINT3 (see item 28.8.7.1.). Entry to PRINT2 is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PRINT2,I	
	EQU	PRINT2(119)	

3980.8.7.2.2 Error Message Output

Immediately upon entry a return transfer is made to PRINT3 to print the error message. Upon return from PRINT3, a jump is made to the address whose name is STOP. At STOP, the A register is made 0, and the Q register is set to -0.

39.8.7.2.3 Exit from PRINT2

Exit is made from STOP with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	(LOADER)	

where the name LOADER is declared as an entry point in the LOADER routine and as an external in the PRINT2 routine.

39.8.7.2.4 Subroutines Used by and Internal to PRINT2

PRINT3 which is entered for error message output with a return jump to the address = 114 + (I).

39.8.7.3 PRINT4

The PRINT4 subroutine is used to print a message consisting of a 6 character name and a 4 digit hexadecimal address. The format is as follows:

SSSSXXXXXXSShhhh

where -

S - space
X - alphanumeric character, and
h - a hexadecimal digit.

39.8.7.3.1 Constant Table Storage Referenced by PRINT4

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
BINASC	111-112

The constants are used as follows:

1. BINASC locations are used for storage of hex digits after number conversions have binary to ASCII.

39.8.7.3.3 Entry to PRINT4

Prior to entering PRINT4, the Q register is set to the ASCII starting address for the message. The A register contains the binary number to be converted to ASCII code for the 4 remaining charactersof the output. PRINT4 is entered with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	PRINT4,I	
	EQU	PRINT4(154)	

39.8.7.3.4 Output Message Generation

The address in Q is placed in temporary storage. The binary number in A is converted to the ASCII code for 4 hex digits by a return jump to CONVRT routine. Upon return from the CONVRT routine, the locations BINASC and BINASC+1 contain the ASCII code for the number.

The hex digits in BINASC & BINASC+1 are placed in the 7th and 8th words of the output area. The first word address of the output area is placed in the A register. The number of words to be printed (8), is placed in the Q register. A return jump is made to the LDRIV routine for message output. Upon return from LDRIV a jump is made to the exit from the PRINT4 routine.

39.8.7.3.5 Exit from PRINT4

Exit from PRINT4 is with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PR4XIT,I	
	EQU	PR4XIT(153)	

39.8.7.3.6 Subroutines Used by and External to PRINT4

CONVRT — is used to convert a 16 bit binary number to the ASCII code for 4 hex digits. This routine is entered with a return jump to the address = 148(I). (See item 28.8.4.)

LDRIV — is used to print a message on the list device. This routine is entered with a return jump to LDRIV where LDRIV is declared as an external. (See item 28.7.4.)

39.8.7.4 PRINT5

The PRINT5 routine is used to print a six character name.

39.8.7.4.1 Constant Table Storage Referenced by PRINT5

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
INPCTR	15

INPCTR — contains the 1st word address for storage of the 6 character name to be printed.

39.8.7.4.2 Entry to PRINT5

Prior to entering PRINT5, the location INPCTR is set to the 1st word address for storage of the 6 character name to be printed.

Entry is made to PRINT5 with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	PRINT5,I	
	EQU	PRINT5(158)	

39.8.7.4.3 Output Message Generation

The address in INPCTR is placed in the A register. The length of the output is placed in the Q register (3 words). A return transfer is made to CDRIV where this name is referred to as an external address. A transfer to the exit from PRINT5 is made upon return from CDRIV.

39.8.7.4.4 Exit from PRINT5

Exit is made from PRINT5 with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PR5XIT,I	
	EQU	PR5XIT(157)	

39.8.7.4.5 Subroutines Used by and External to PRINT5

CDRIV is used to print the 6 character name. (See item 28.7.2.)

39.8.8 LINK1

Instructions which reference the same external name are tied together by a string of link addresses. The assembler generates two words for each instruction that references an external name. Bits 0-14 of the second word of each instruction contains a link address which points to the location containing the next link address in the string. The last location in the string contains a link address of $7FFF in bits 0-14.

The LINK1 subroutine will replace each of the link addresses in a string with the entry point address for the particular name.

39.8.8.1 Constant Table Storage Referenced by LINK1

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
DATLEN	177
ENTPNT	13
LINK	14
INPWRD	10
NGRLSW	9
PRODIF	171

Constant Table storage is used as follows:

1. DATLEN — contains the length of the data block.
2. ENTPNT — contains the absolute value of the entry point address in bits 0-14.
3. LINK — contains the absolute value of the 1st link address in a string.
4. INPWRD — is used as temporary storage during program execution.
5. NGRLSW — is set positive or negative prior to entry to LINK1. Determines whether patching is for absolute or relative addressing.
6. PRODIF — contains the mass storage word count.

39.8.8.2 Communication Region Constants Used by LINK1

CONSTANT	LOCATION
$7FFF	$42 = MASK1

39.8.8.3 Entrance to LINK1

Both LINK and ENTPNT are set prior to entering LINK1. Prior to entry to LINK1 the location NGRLSW is set either positive or negative. The location NGRLSW is set with ((INPCTR)) = 1st 2 characters of the name. LINK1 is then entered with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	LINK1,I	
	EQU	LINK1(164)	

39.8.8.4 Patching

Upon entry to the subroutine LINK1, the location LINK contains a link address and the location ENTPNT contains an entry point address. The address in LINK points to the 1st location in a link address string. Each location in the link address string contains an address which points to the next location in the string. Bits 0-14 of the last location in the string are set to the value $7FFF.

The address in LINK and ENTPNT are execution time addresses - they refer to locations in execution time core. Each of the addresses in a LINK address string is an execution time address. The command sequence containing the link address string occupies load time core. Therefore, the load time core address for the 1st location in the

string is obtained by the following arithmetic:

(LINK) + (DATLEN) - (PRODIF) = Load time core address for 1st location in link address string.

Similarly, the load time core address for successive locations in the string are obtained as follows:

Execution time address + (DATLEN) - (PRODIF) = load time core location

If the contents of NGRLSW is positive, patching for the absolute mode of addressing will occur.

LINK1 will replace the contents of every location in the string of the link addresses by the entry point address in ENTPNT. The following occurs:

(DATLEN) - (PRODIF) + (LINK) LINK

bits 0-14 of ((LINK)) INPWRD

(ENTPNT) (LINK) at bits 0-14, bit 15 of ((LINK)) is not changed

(INPWRD) LINK

If the contents of NGRLSW are ~~positive~~, LINK1 will replace the contents of every location in the string by a relative value. This value is the difference between the address in LINK and the entry point address. The following occurs:

(DATLEN) - (PRODIF) + (LINK) LINK

bits 0-14 of ((LINK)) INPWRD

(ENTPNT) - (LINK) (LINK) at bits 0-15

(INPWRD) LINK

LINK1 will loop to process the next link address in the string. The procedure terminates when the contents of LINK (bits 0-14) is $7FFF. A transfer is made to the address stored in the entry point.

NOTE: The LINK1 operation may be referred to as patching. Patching, as illustrated above, may be either relative or absolute. However, the mode of patching is uniform for each location in the string of link addresses.

3.8.8.5 Exit from LINK1

Exit is made fromthe LINK1 routine using the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	LKIXIT,I	
	EQU	LKIXIT(163)	

39.8.9 LINK2

LINK2 is the subroutine that is used to tie together two strings of link addresses. The last 15 bit address in each string is $7FFF. When two strings are tied together, the $7FFF at the end of string A is replaced by the address of the location containing the $7FFF at the end of string B. The complement of the 1st address of string B is placed in WORD4 of the Loader Table entry for the external name, replacing the 1st address from string A.

39.8.9.1 CONSTANT TABLE STORAGE REFERENCED BY LINK2

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
DATLEN	177
ENTPNT	13
INPWRD	10
LINK	14
NGRLSW	9
PRODIF	171
PROLIM	181
SW6	28
TABLIM	7

Constant table storage is used as follows:

1. DATLEN — contains the length of the data block.
2. ENTPNT — contains the absolute value of the address which points to the beginning of the 1st string.
3. INPWRD — is used as temporary storage during program execution.
4. LINK — contains the absolute value of the address which points to the beginning of the second string.
5. PRODIF — contains the mass storage word count.
6. PROLIM — contains the upper limit address (last address + 1) for the odd sector (1-95 words in length) at the bottom of unprotected core.
7. TABLIM — contains the lowest (toward 0) address occupied by the Loader Table.
8. SW6 — contains an index, a value which when added to the base of the Loader Table, results in an address which points to an entry in the Loader Table.

39.8.9.2 Communication Region Constants Used by LINK2

CONSTANT	LOCATION
$7FFF	$42 = MASK1
$8000	$21 = MASK2

39.8.9.3 Entry to LINK2

Both LINK and ENTPNT are set prior to entering LINK2. The LINK2 subroutine is entered with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	LINK2,I	
	EQU	LINK2(168)	

39.8.9.4 Linking

Upon entry to LINK2, the location ENTPNT contains a link address from an entry in the EXT input block. The location LINK contains the link address for an entry in the Loader Table. These addresses are both execution time addresses - they reference locations in execution time core. The external names from the input block entry and Loader Table entry match.

The address in ENTPNT is temporarily recorded in the location INPWRD. If the address in ENTPNT is $7FFF a transfer is made to the exit from LINK2. An address of $7FFF is ENTPNT means that within the program currently being read in there is an external name which is not referenced by the address protion of any instruction in the program. If the address in ENTPNT is not $7FFF, it is a pointer to the second address in a string of link addresses which occur in the program currently being loaded. The location in the string which contains a $7FFF is the last address in the string. The address in LINK points to the 2nd address in a string of link addresses which occur in the previous program(s) loaded.

In order to combine these two strings, the following is a necessary condition: It is necessary for all programs to be loaded which reference the external name in question, do so in a uniform manner; (i.e., they must all reference this name with either the relative mode of addressing or with the absolute mode of addressing). This condition is met if the sign bits are alike for the 1st word in each of the two entries.

(INPCTR) = 1st word address of the EXT block entry.

(TABLIM)+(SW6) = 1st word address of the Loader Table entry.

It is necessary that

$$(INPCTR)_{15} = ((TABLIM) + (SW6))_{15}$$

Before this link operation occurs a check is made on the sign bits for the first word of both the EXT block entry and the Loader Table entry for this external name. The sign bit settings for these words must be alike. If they differ, the following error procedure results:

1. A return jump is made to the PRINT3 error subroutine to print the error indication of E12.
2. A return jump is made to the PRINT5 subroutine to print the name of the external.
3. The Loading Operation is terminated when a jump is made to the location STOP where "STOP" is declared as an external.

If the sign bits are alike, the operation of LINK2 is allowed to proceed.

In order to combine the string of link addresses corresponding to the Loader Table entry with the string corresponding to the EXT blank entry, LINK2 will do the following:

1. Beginning at the location whose label is SLEW, LINK2 traces through to the end of the link address string beginning with the address in ENTPNT. The address in ENTPNT, together with the other addresses in the Link address string, are execution time addresses. The link address string is contained in load time core. The load time core address of the 1st location in the string is obtained by -

   $$(DATLEN) - (PRODIF) + (ENTPNT) \longrightarrow ENTPNT$$

2. $_{14}((ENTPNT))_0$ = the execution time value of the next address in the link address string. If $_{14}((ENTPNT))_0$ = \$7FFF, the end of the string has been reached and the operation proceeds with step 4.
3. If $_{14}((ENTPNT))_0 \neq 0$, the load time value for the next address in the link address string is obtained by -

   $$(DATLEN) - (PRODIF) + (ENTPNT) \longrightarrow ENTPNT$$

   Step 2 is repeated.
4. (ENTPNT) = the load time location for the last address in the link address string from the EXT block entry. The link address from the Loader Table entry, = (LINK), is recorded in bits 0-14 of (ENTPNT). Bit 15 of (ENTPNT) is not changed:

   $$_{14}(LINK)_0 + ((ENTPNT))_{15} \longrightarrow (ENTPNT)$$

5. (INPWRD) = the execution time location for the 1st address in the link address string from the EXT block entry. The one's complement of this address is recorded in WORD3 of the Loader Table entry:

   -(INPWRD) (TABLIM) + (SW6) + 3

6. An exit is made from the LINK2 routine.

For purpose of illustration:

1. Prior to the return transfer to LINK2

   a. a previous program loaded contains the following command sequence:

LABEL	OPCODE	ADDRESS	REMARKS
	NAM	PROG A	
	EXT	ABC	
	ORG	DEF	
	RAO	ABC	
	INA	5	
	STA	ABC	

and the machine language is organized as follows:

LOCATION	CONTENTS		
DEF	RAO	01XX	00
	0	$7FF	
DEF+2	LDA	01XX	00
	0	DEF+1	
DEF+4	INA	05	
DEF+5	STA	01XX	00
	0	DEF+3	

   b. the loader table contains the following entry:

WORD1	0	A	B
WORD2		C	△
WORD3		△	△
WORD3	one's complement of DEF+5		

c. the current program being loaded contains the following sequence:

LABEL	OPCODE	ADDRESS	REMARKS
	NAM	PROG B	
	EXT	ABC	
	ORG	GHI	
	LDA	ABC	
	INA	-3	
	STA	ABC	

and the machine language is organized as follows:

LOCATION	CONTENTS		
GHI	LDA	01XX	00
	0	$7FF	
GHI+2	INA	-3	
GHI+3	STA	01XX	00
	0	DEF+1	

d. the EXT block contains the following entry:

WORD1	0	A	B
WORD2		C	△
WORD3		△	△
WORD4	one's complement of GHI+4		

2. Upon returning from LINK2 to the calling program

a. the machine language is organized as follows:

LOCATION	CONTENTS		
DEF	RAO	01XX	00
	0	$7FFF	
DEF+2	LDA	01XX	00
	0	DEF+1	
DEF+3	INA	5	
DEF+4	STA	01XX	00
	0	DEF+3	
GHI	LDA	01XX	00
	0	DEF+5	
GHI+2	INA	-3	
GHI+3	STA	01XX	00
	0	GHI+1	

b. the loader table entry in l.b. becomes

WORD 1	0	A	B
WORD 2		C	△
WORD 3		△	△
WORD 4	one's complement of GHI+4		

For this example, when time comes to perform the LINK1 operation, all patching will be carried out for the absolute mode of addressing.

NOTE: Had the exit psuedo in PROG A been

```
EXT*        ABC
```

while

```
EXT         ABC
```

the following error output would have occurred during the load operation LINK2:

```
E14
ABC
```

The entry in the EXT input block in PROG B will appear as follows:

0	A	B
	C	△
	△	△
one's complement of GHI+4		

If, in the first illustration, no references had been made by this command sequence in PROG B to the external name, the fourth word of the EXT input block will contain the one's complement of $7FFF or $8000.

Under such circumstances it will not be necessary to perform the LINK2 operation. Therefore, if upon entry to LINK2, (ENTPNT) = $7FFF, a transfer will be made immediately to the exit address.

398.9.5 Exit from LINK2

Exit from LINK2 is made with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	LK2XIT,I	
	EQU	KL2XIT(167)	

39.8.9.6 Subroutines Used by and External to LINK2

1. PRINT3 is used to print the error indication E12. (See item 28.8.7.1.)
2. PRINT5 is used to print the name of an external. (See item 28.8.7.4.)

Entry to these subroutines is with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	PRINT3,I	ENTRY TO PRINT3
	EQU	PRINT3(114)	
	RTJ-	PRINT5,I	ENTRY TO PRINT5
	EQU	PRINT5(158)	

39.8.10 COREXT

The lower & upper limits of unprotected core as far as the Loader is concerned, are contained in LWRLIM & UPRLIM respectively. The Job Processor places the Relocatable Binary Loader at the high end of unprotected core. The lowest (toward 0) address occupied by the Loader is placed in the 23rd location of the constant table. The Location TABLIM is set initially to this address. The Loader Table extends downward in core. The location LOWCOR is initially set to (LWRLIM). That area of core available for storage of relocatable binary input is referred to etiher as "load time core" or "available core". The limits of available core are always:

(LOWCOR)+1 to (TABLIM)-1.

The length of any relocatable binary program loaded may not exceed the size of unprotected core. However, it may exceed the size of available core. When this occurs, available core no longer is the permanent storage place for the command sequence at execution time. Instead, available core is used as a buffer into which the Loader stores words of the command sequence after it absolutizes the relocatable binary input. When this buffer is full, its contents are recorded as a block on mass storage. The operation is repeated until the loading operations is interrupted. Upon return to the monitor from the Loader, a location in the constant table will have the tally of the number of owrds on mass storage. The monitor will read this information into the low end of unprotected core before entering the program for execution. It is the function of the COREXT routine to empty the block of available core onto mass storage.

39.8.10.1 Constant Table Storage Referenced by COREXT

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
ASAV	89
QSAV	90
PROSTR	1

LOWCOR	179
PRODIF	171
CSQCTR	183
COUNT1	25
PROLIM	181
DATLEN	177
SECTOR	198
SECTNO	173-174

The contents are used as follows:

1. ASAV — is used for temporary storage of the operand in the A register.
2. QSAV — is used for temporary storage of the operand in the Q register.
3. PROSTR — holds a load time storage base for the next program to be loaded.
4. LOWCOR — contains an address which when incremented by 1 is the lower limit address (lower address-1) of load time core.
5. PRODIF — contains the tally for the number of words written onto mass storage by the COREXT routine.
6. CSQCTR — is used as an address counter.
7. COUNT1 — is used as an address counter.
8. PROLIM — is set prior to the exit from COREXT to an execution time address which is the upper limit of the "odd sector". (See item 28.3.2.10.) This address has a value such that

$$(LOWCOR) \cdot PRODIF)-(DATLEN) \leq (PRODIF) \leq$$
$$(LOWCOR) \cdot 96-(PRODIF)-(DATLEN)$$

9. DATLEN — contains the count for the number of words set aside for a data reservation.
10. SECTOR — contains the word count for 1 sector on mass storage = 96.
11. SECTNO — is the name for the 1st of 2 locations which contains the absolute value of the starting sector number when transferring words from core to mass storage. The number is 30 bits in length. The 15 most significant bits are recorded in bits 0-14 of SECTNO. The 15 least significant bits are recorded in bits 0-14 of SECTNO

39.8.10.2 Entry to COREXT

Entry to the COREXT routine is with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	COREXT,I	
	EQU	COREXT(188)	

39.8.10.3 COREXT Operation

Upon entry to COREXT, the number of words held in available core is equal to

(PROSTR) - (LOWCOR)-1.

The COREXT will only write whole sectors onto disk. The number of words in available core is divided by 96.

(PROSTR) - (LOWCOR)-1 = X + Y

where X is the quotient and Y is the remainder. The number of words written onto mass storage is 96*X. The output area begins at (LOWCOR)+1 and terminates with the address (LOWCOR)+1+96*X. The tally, PRODIF, is increased by the value of 96*X. If X is zero, fewer than 96 words are in available core and no mass storage output occurs.

The contents of the Y words at the high end of load time core are moved to the low end of load time core.

COUTN1 and CSQCTR are address counters used for the core to core transfer. COUNT1 is set to the address of (LOWCOR)+1+96*X, and CSQCTR is set to (LOWCOR)+1. These address counters are increased by 1 for each word transferred. When the transfer is complete, the value in CSQLIM is placed in PROSTR. PROLIM is set to the execution time address of (PROSTR) + (PRODIF) - (DATLEN). A jump is made to the exit from COREXT.

If Y is zero, there is no core to core transfer as described above. The value of (LOWCOR)+1 is placed in PROSTR. PROLIM is set to (PROSTR) + (PRODIF) - (DATLEN). A jump is made to the exit. If X is zero, there is no information to be written onto mass storage in the manner described above. The value = (LOWCOR)+1+Y is placed in PROSTR. PROLIM is set to (PROSTR) + (PRODIF) - (DATLEN). A jump is made to the exit.

39.8.10.4 Exit from COREXT

The exit from COREXT is made in the following way:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CXTXIT,I	
	EQU	CXTXIT(187)	

39.8.10.5 Subroutines Used by and External to COREXT

MDRIV is used to write absolute records from available core onto mass storage. MDRIV is entered with a return jump to the address. (See item 28.7.1.)

DPRADD is used to compute the starting sector number for the mass storage operation. (See item 28.8.11.)

39.8.11 DPRADD

This routine is used to compute the number for starting sector for mass storage operations.

39.8.11.1 Constant Table Storage Referenced by the DPRADD Routine

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
SECTNO	173
DPRXIT	199

The constants are used as follows:

1. SECTNO is the name for the 1st of 2 locations at which the double precision sector number (30 bits in length) is stored once it is computed by the DPRADD routine.
2. DPRXIT is the location to which a jump is made in order to exit from the DPRADD routine.

39.8.1..2 Communication Region Constants Used by DPRADD

CONSTANT	LOCATION
$7FFF	$42 = MASK1
last sector of Load-and-Go	$E4

39.8.11.3 Entrance to DPRADD

Entrance to the DPRADD subroutine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	DPRADD,I	
	EQU	DPRADD(200)	

39.8.11.4 DPRADD Function

Upon entry to DPRADD, the A register contains a sign bit of 0 and a 15 bit number in bits 0-14. The number in A will be referred to as a single precision number. The location $E4 also contains a positive 15 bit number which is the relative value for the sector number at the end of Load-and-Go information on mass storage. The locations $C0 and $C1 contain a double precision number for the 1st sector in the scratch area. The most significant half of the number is in bits 0-14 of $C0 and the least significant half in bits 0-14 of $C1.

The single precision number in the A register is added to the number in $E4. The result, in turn, is added to the double precision number in $C0 and $C1 and the result is placed in SECTNO and SECTNO+1.

The sign bits of SECTNO and SECTNO+1 are 0's. The 15 most significant bits are in bits 0-14 of SECTNO while the 15 least significant bits are in bits 0-14 of SECTNO+1.

39.8.11.5 Exit from DPRADD

Exit from the DPRADD subroutine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	DPRXIT,I	
	EQU	DPRXIT(194)	

39.9 MAIN LOOP WITHIN THE LOADING PROCEDURE

The main loop within the Loading Procedure consists of all the routines and subroutines which are concerned with reading and processing Relocatable Binary Loader Input records.

39.9.1 LOADER

The functions of the LOADER routine are as follows:

1. Read an input block.
2. If I/O error occurs during input operation, take appropriate action.
3. Identify an input record, and branch to the appropriate routine to further process input record.
4. If input record is unrecognizable, take appropriate action.

39.9.1.1 Constant Table Storage Referenced by the LOADER Routine

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
CSQCTR	183
DATLEN	177

DSECNO	203
INPXCO	117
INPUT	29-88
LOWCOR	179
PRODIF	171
PROLIM	181
PROSTR	185
SECTOR	198
SECTNO	173-174
TABCTR	16
TABLIM	7
TABSNO	204

The constants are used as follows:

1. CSQCTR — contains the load time limit address of the command sequence of all programs read in prior to the current Loader Operation.
2. DATLEN — contains the size of core set aside for data storage.
3. DSECNO — contains the starting sector number (relative number) for recording the contents of the Load Time Data storage buffer on the mass storage unit containing the scratch area.
4. INPXCO — contains the starting location of the 60 word input area for storage of the relocatable binary input blocks read by the Loader.
5. INPUT — is the name for the first word address of 60 sequential locations used as storage for relocatable binary input records read by LOADER.
6. LOWCOR — contains the lower limit address of load time core.
7. PRODIF — contains length of the absolute record of the command sequence on mass storage.
8. PROLIM — contains the storage limit address (last address+1) for the odd sector (1-95 words in length) located at the bottom of unprotected core. (Refer to item 28.3.2.10.)
9. PROSTR — contains the load time storage base for the next program to be read in by the loader.
10. SECTOR — contains the word count for a sector on mass storage. Word count = 96.
11. SECTNO — the 1st of 2 locations containing a mass storage sector number. The number is 30 bits in length. The 15 most significant bits are in bits 0-14 of SECTNO. The 15 least significant bits are in bits 0-14 of SECTNO+1.

12. TABCTR — contains the number of entries in the Loader Table.
13. TABLIM — contains the address which is the storage base of the Loader Table.
14. TABSNO — contains the starting sector number (relative value) for recording the contents of the Loader Table on the mass storage unit containing the scratch area.

39.9.1.2 Communication Region Constants Used by the Loader

CONSTANT	LOCATION
$7FFF	$42 = MASK1

39.9.1.3 Entry to the LOADER Routine

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ	LOADER	

where -

the name LOADER is referenced as an external within the program from which the return jump is made, and as an entry point name within the LOADER routine.

If, upon entry to LOADER, (PRODIF) = 0, there is no mass storage word count. The command sequence of programs read in on previous Loader Operations, if any, is contained entirely within the space of Load Time Core. A jump is made to the location SW1AXC, and the 1st input block is read for the current Loader Operation.

If, upon entry to LOADER, (PRODIF) ≠ 0, (PRODIF) = the mass storage word count of the entire command sequence from programs read in during previous Loader Operations. While the current Loader Operation is in progress, the mass storage word count in PRODIF must always be an integer multiple of 96. If, upon entry to LOADER, (PRODIF) is an integer multiple of 96, a jump is made to SW1AXC. If not, the last 1-95 words of the command sequence recorded on mass storage is read back into the bottom of Load Time core and (PRODIF) is reduced by this amount as follows:

If (PRODIF) ≠ 0, and -

if $\frac{(PRODIF)}{96} = X + \frac{Y}{96}$ where $Y \neq 0$, then do the following:

1. Use the DPRADD routine (see item 28.8.11) to compute the absolute value of the scratch area sector containing the & words (where $1 \leq Y \leq 95$). Record this sector number in the locations SECTNO & SECTNO+1.

X + ($E4) + ($C0 & $C1) SECTNO & SECTNO+1

The starting address is placed in the A register. The word count is placed in the Q register. A return jump is made to the MDRIV routine (see item 28.7.3) to read the 1-95 words into the bottom

(LOWCOR) + 1 A

Y Q

Upon return from MDRIV a jump is made to SW1AXC to read the first relocatable binary input block for the current Loader Operation.

3991.4 Reading Input Records

At the location SW1ASC, the location SW1+1 is set to the address "SW1A". The locations SW1 and SW1+1 contain a 2 word jump instruction whose address is set during program execution. NOTE: In order to maintain the "run anywhere" characteristics of the Loader, the location SW1 contains a value of $1800, and SW1+1 is set to the following 16 bit value:

SW1A - SW1 - 1

Input records to the Loader may be either binary formatted or ASCII formatted. All input operations are carried out in the binary mode. Therefore, any ASCII formatted record used as Loader input must have an "*" as the first character. The binary formatted records must be one of the 6 relocatable binary formats produced by the 1700 Assembler.

3991.4.1 Reading Relocatable Binary Input Blocks

The LOADER routine begins its input operation at the location NXTBLK. The Loader reads relocatable binary records or input blocks 60 words in length from the input device. The LOADER routine will read a formatted record in binary mode by doing the following:

1. The 60 word buffer for relocatable binary input is backgrounded to all ones.
2. The A register is set to the one's complement of the starting address.
3. A return jump is made to the IDRIV routine (where the address IDRIV is referenced as an external name).

Upon return from the IDRIV routine, the A register is set as follows:

1. (A) = -0 if the read operation is error free. A jump is made using the two word jump instruction (with the variable address) at SW1 and SW1+1. The jump is made to the address at which the processing for the input record begins.
2. (A) = 0 if the read operation were terminated due to an unrecoverable error encountered while reading. A jump is

12. TABCTR contains the number of entries in the Loader Table.

13. TABLIM contains the address which is the storage base of the Loader Table.

14. TABSNO contains the starting sector number (relative value) for recording the contents of the Loader Table on the mass storage unit containing the scratch area.

39.9.1.2 Communication Region Constants Used by the Loader

CONSTANT	LOCATION
$7FFF	$42 = MASK1

39.9.1.3 Entry to the LOADER Routine

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ	LOADER	

where -

the name LOADER is referenced as an external within the program from which the return jump is made, and as an entry point name within the LOADER routine.

If, upon entry to LOADER, (PRODIF) = 0, there is no mass storage word count. The command sequence of programs read in on previous Loader Operations, if any, is contained entirely within the space of Load Time Core. A jump is made to the location SW1AXC, and the 1st input block is read for the current Loader Operation.

If, upon entry to LOADER, (PRODIF) $\neq$ 0, (PRODIF) = the mass storage word count of the entire command sequence from programs read in during previous Loader Operations. While the current Loader Operation is in progress, the mass storage word count in PRODIF must always be an integer multiple of 96. If, upon entry to LOADER, (PRODIF) is an integer multiple of 96, a jump is made to SW1AXC. If not, the last 1-95 words of the command sequence recorded on mass storage is read back into the bottom of Load Time core and (PRODIF) is reduced by this amount as follows:

If (PRODIF) $\neq$ 0, and -

if $\frac{(PRODIF)}{96} = X + \frac{Y}{96}$ where Y $\neq$ 0, then do the following:

1. Use the DPRADD routine (see item 28.8.11) to compute the absolute value of the scratch area sector containing the & words (where $1 \leq Y \leq 95$). Record this sector number in the locations SECTNO & SECTNO+1.

X + ($E4) + ($C0 & $C1) SECTNO & SECTNO+1

The starting address is placed in the A register. The word count is placed in the Q register. A return jump is made to the MDRIV routine (see item 28.7.3) to read the 1-95 words into the bottom

(LOWCOR) + 1 A

Y Q

Upon return from MDRIV a jump is made to SW1AXC to read the first relocatable binary input block for the current Loader Operation.

3999.1.4 Reading Input Records

At the location SW1ASC, the location SW1+1 is set to the address "SW1A". The locations SW1 and SW1+1 contain a 2 word jump instruction whose address is set during program execution. NOTE: In order to maintain the "run anywhere" characteristics of the Loader, the location SW1 contains a value of $1800, and SW1+1 is set to the following 16 bit value:

SW1A - SW1 - 1

Input records to the Loader may be either binary formatted or ASCII formatted. All input operations are carried out in the binary mode. Therefore, any ASCII formatted record used as Loader input must have an "*" as the first character. The binary formatted records must be one of the 6 relocatable binary formats produced by the 1700 Assembler.

3991.4.1 Reading Relocatable Binary Input Blocks

The LOADER routine begins its input operation at the location NXTBLK. The Loader reads relocatable binary records or input blocks 60 words in length from the input device. The LOADER routine will read a formatted record in binary mode by doing the following:

1. The 60 word buffer for relocatable binary input is backgrounded to all ones.
2. The A register is set to the one's complement of the starting address.
3. A return jump is made to the IDRIV routine (where the address IDRIV is referenced as an external name).

Upon return from the IDRIV routine, the A register is set as follows:

1. (A) = -0 if the read operation is error free. A jump is made using the two word jump instruction (with the variable address) at SW1 and SW1+1. The jump is made to the address at which the processing for the input record begins.
2. (A) = 0 if the read operation were terminated due to an unrecoverable error encountered while reading. A jump is

made to the PRINT2 error exit where the error indication "E1" is printed and the Loading Operation terminates.

3. (A) =+1 if the read operation were terminated because of the alarm condition. The alarm condition arises as a result of motion failure on the part of the input device. A cause of this, to cite an example, is trying to read paper tape when there is no tape in the reader. The significance of the alarm error is discussed in the next four paragraphs.

A relocatable binary program consists of a sequence of relocatable binary records beginning with a NAM block and terminating with an XFR block. These sequential records must be stored on a continuous external medium; i.e., not split between 2 paper tapes. Therefore, once the loading of a relocatable binary program has begun, an XFR block must be read before an alarm condition is sensed.

The alarm condition is acceptable to the Loader if the loading of a relocatable binary program has been completed by reading and processing its XFR block. The alarm condition is also acceptable after reading an ASCII input block.

Prior to reading the lst relocatable binary input block, and subsequent to reading each XFR block, the address for the jump instruction at SW1 is set to SW1A. Also, subsequent to reading the first HEX block, the address for the jump instruction at SW1 is set to SW1E. It remains set to SW1E for the duration of the Loader Operation.

Therefore, an alarm condition is acceptable to the Loader if the delta in the second word of the jump instruction at SW1 is set as follows:

(SW1+1) = SW1A - SW1 - 1

or

(SW1+1) = SW1E - SW1 - 1

If either of these conditions is met a jump is made to the location whose label is ALARMOK. At ALARMOK, the ASCII code for "*T" is placed in the location INPUT, and the location INPUT+1 is set to a -0. A jump is made to SW1F.

If neither of the above conditions is met, a jump is made to the PRINT2 error exit where the error indicator "E14" is printed and loading terminates.

39.9.1.4.2 Reading ASCII Records from the Input Device

Since ASCII records are read from the input device in the binary mode, they will not be processed as ASCII records unless the first character is an "*". The ASCII records should not exceed 120 characters in length. For a maximum length record, the 120th char-

acter is expected to be a carriage return.

The three steps involved in reading an ASCII record are the same as those in reading a binary record (as indicated in the 1st paragraph of item 28.9.1.4.1). Sequential ASCII records need not occur on a continuous external medium. The records are read and processed until the End of Load statement is read ("*T carriage return") or until the alarm condition arises.

Refer to item 28.9.1.4.1 for the procedure to be followed should an alarm condition arise.

39.9.1.4.3 Types of ASCII Records

1. *H, hex correction information and carriage return comprises a statement containing hexadecimal correction constants.
2. *T carriage return is an End of Load Statement.

39.9.1.5 Branching to Process an Input Block

The contents of the first word in the input buffer determines branching. If the bits $_{15}(\text{INPUT})_{8}$ set to the ASCII code for *, branching goes to either HEXPRO or EOLPRO. Otherwise branching is to process relocatable binary input and it is determined by the bits $_{15}(\text{INPUT})_{13}$.

All branching is with the 2 word jump instructions at SW1 using relative addressing. The particular branch chosen to process an input block depends on the address position of this jump instruction.

39.9.1.5.1 SW1A

The address in the switch 1 jump instruction is set to SW1A by Program initialization. It is reset to SW1A each time an XFR block is to be processed. While switch 1 is set to SW1A only, the NAM blocks and ASCII input statements will be processed. The loader will transfer to the error exit PRINT2 for all other blocks received at this time. If the NAM block is received, the loader will reset the jump address in SW1 to SW1B and then jump to NAMPRO. If the HEX or EOL block is received, the loader will set the jump address in SW1 to SW1F and then jump either to HEXPRO or EOLPRO.

The address NAMPRO, HEXPRO and EOLPRO are referenced as external names by jump instructions within the LOADER Routine.

39.9.1.5.2 SW1B

The address in the switch 1 jump instruction is set to SW1B whenever a NAM block is processed.

While jump switch 1 is set to SW1B the loader will process only the following block: RBD, BZS, ENT, EXT and XFR. The loader will branch to RBDPRO, BZSPRO, ENTPRO, EXTPRO or XFRPRO accordingly. The Loader will jump to the PRINT2 error exit each time a NAM, HEX or EOL block is received while the SW1 jump instruction has SW1B for a jump address.

The addresses RBDPRO, BZSPRO, ENTPRO, EXTPRO and XFRPRO are referenced as external names by jump instructions within the LOADER routine.

39.9.1.5.3 SW1C

The address of jump instruction SW1 is set to SW1C whenever an EXT block is received by the loader.

While jump switch 1 is set to SW1C, the loader will process only EXT and XFR blocks. All others will cause the loader to take the PRINT2 error exit.

39.9.1.5.4 SW1F

Subsequent to the most recent return jump to LOADER, a jump will be made to SW1F either when an ASCII input statement is read, or when an alarm condition has arisen which is acceptable to the Loader. If a jump is made to SW1F, one of the following two conditions will exist:

CONDITION 1: The amount of unprotected core occupied by all of the relocatable binary programs has exceeded the size of the block of core from (LOWCOR)+1 to (TABLIM)-1. The location PRODIF contains the size of the absolute record on mass storage. The number of words in PRODIF is always a multiple of 96 (96 words/sector on disk). The number for the sector on disk at which storage of this information begins is a double precision number contained in $C0 and $C1. The portion of the command sequence yet in core is contained in the block of memory from (LOWCOR)+1 to (TABLIM)-1. The exact amount of storage is from (LOWCOR)+1 to (CSQCTR)-1. The portion of the command sequence yet contained in core is written onto the mass storage at the end of portion of the command sequence already on mass storage. In order to dump this information onto mass storage, a return jump is made to the routine COREXT. NOTE: The COREXT routine will only transfer a record length which is an integral multiple of 96. Therefore, with 96 words/sector on mass storage, the odd sector (the last 1-95 words yet in core) will not be transferred. In order to assure that the information in the odd sector will be written onto mass storage, the Q register is set

to the complement of the length of the storage yet held in core which is equal to (CSQCTR) - (LOWCOR)-1. The A register is set to the starting address which is equal to (LOWCOR)-1. The locations SECTNO & SECTNO+1 contain the starting sector number. A return jump is made to MDRIV to dump the last 1-95 words.

If (DATBAS) $\neq$ 0, there exists a data reservation. The relative number for the last sector containing the command sequence is computed and recorded at DSECNO as follows:

$$\frac{(\text{PRODIF})}{96} \longrightarrow A, Q$$

If (Q) $\neq$ 0, (A) +1 $\longrightarrow$ A

(A) $\longrightarrow$ DSECNO

If the contents of the Load Time Data Storage buffer is to be placed on mass storage, the relative value of the starting sector number for the transfer is in DSECNO. If there is no data reservation, DSECNO remains a zero.

If (TABCTR) $\neq$ 0, there is a Loader Table. If the Loader Table is to be recorded on mass storage, it is placed behind the sectors occupied by the Load Time Data Storage buffer. The relative value for the number of the 1st sector to be occupied by the Loader Table is recorded in TABSNO. It is computed as follows:

$$\frac{(\text{DATLEN})}{96} \longrightarrow A, Q$$

If (Q) $\neq$ 0, (A) + 1 $\longrightarrow$ A

(A) + (DSECNO) $\longrightarrow$ TABSNO

If there is no data reservation, the sector number to be recorded in TABSNO is computed as follows:

$$\frac{(\text{PRODIF})}{96} \longrightarrow A, Q$$

If (Q) $\neq$ 0, (A) + 1 $\longrightarrow$ A

(A) $\longrightarrow$ TABSNO

The Load Time Data Storage Buffer and the Loader Table will be transferred temporarily to mass storage if the space they occupy in unprotected core is needed for other uses. (Refer to item 28.9.7 - EOLPRO.)

CONDITION 2: The amount of unprotected core occupied by all of the relocatable binary programs has not exceeded the

size of the block of core from (LOWCOR)+1 to (TABLIM)-1. The entire command sequence storage is held within load time core and the value stored in PRODIF is zero. The address of the jump instruction at SW1 is set to SW1E, and a jump is made to SW1E.

39.9.1.5.5 SW1E

When the address of jump instruction SW1 is set to SW1E, only ASCII input statements will be processed. Binary records will cause the Loader to take the PRINT2 error exit where the error indication "E3" will be printed and loading terminates.

39.9.1.6 Exit from the LOADER Routine

Exit from the LOADER Routine (other than exits due to error) is made whenever an ASCII input statement is read which is assumed by the Loader to be a monitor control statement. In this case, the A register is set to a value of -0 to indicate an error free exit, and the Q register is set to address constant "INPUT" (the beginning of storage for the ASCII control statement). Exit from the LOADER routine is with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	(LOADER)	

In the event it was necessary to take the PRINT2 error exit, the ASCII code for the error number is placed in the A register. A jump is made to the PRINT2 error exit address in the following way:

LABEL	OPCODE	ADDRESS	REMARKS
	~~EQU~~	~~PRINT2(119)~~	
	JMP-	PRINT2,I	

39.9.1.7 Subroutines Used by and Internal to the LOADER Routine

Zero — is used to background the buffer for relocatable binary input to -0 prior to reading input from either the input or the communication devices. ZERO is entered with a return jump instruction.

39.9.1.8 Subroutines Used by and External to the LOADER Routine

1. PRINT2 — is used for error message output. Operation is terminated following printout. (See item 28.8.7.2.)

2. MDRIV — is used to write command sequence storage onto mass storage. Also, it is used to write the Loader Table onto mass storage. (See item 28.7.3.)

3. IDRIV — is used to read input records from the input device. (See item 28.7.1.)

4. COREXT — is used to dump core containing absolute records of command sequence storage onto the mass storage device. COREXT is entered to its entry point location in the constant table. (See item 28.8.10.)

5. DPRADD — is used to compute the absolute value of the starting sector number for all mass storage operations. DPRADD is entered via a location in the constant table. (See item 28.8.11.)

The PRINT2, COREXT & DPRADD routines are entered via the entry point locations in the Constant Table. (See item 28.8.10.1 and 28.8.9.1).

IDRIV and MDRIV are entered with return jump to their respective entry point addresses.

39.9.2 NAMPRO

NAMPRO is the routine which is used to process a NAM block.

39.9.2.1 Constant Table Storage Referenced by the NAMPRO Routine

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
AINPUT	161
BLANKS	18
COMBAS	1
COMLIM	4
CSQCTR	183
CSQLIM	6
DATBAS	2
DATCTR	182
DATDIF	184
DATLEN	177
DATLIM	5
DATRES	178
DATSTR	180
ENTPNT	13
LINK	14
LOWCOR	179
LWRLIM	191
INPCTR	15
INPUT	29-88
PROBAS	3
PRODIF	171
PROLIM	181
PROSTR	185

QSAV	90
SW6	28
TABLIM	7

The constants are used as follows:

1. AINPUT — contains one of the entrance parameters for the current Loader Operation. (Refer to item 28.5.1.)
2. BLANKS — contains a constant used for editing when generating an output message. (BLANKS) = $2020 which is the ASCII code for 2 spaces.
3. COMBAS — contains the Common Storage relocation base.
4. COMLIM — contains the upper limit address of the common storage reservation (equal to last address + 1).
5. CSQCTR — contains the load time core upper limit address for the command sequence of the program whose NAM block is currently being processed. (The limit address is equal to 1 + address of last word of program at load time.)
6. CSQLIM — contains the execution time core upper limit address for the command sequence of the program whose NAM block is currently being processed. (This limit address is equal to 1 + address of last word of program at execution time.)
7. DATBAS — contains the relocation base for the Execution Time Data Storage Block Reservation.
8. DATCTR — contains the upper limit address (equal to last address + 1) of the Load Time Data Storage Buffer.
9. DATDIF — ~~contains the value equal to (DATBAS)-(DATSTR).~~
10. DATLEN — contains the number of words set aside for a data reservation.
11. DATLIM — contains the upper limit address (equal to last address + 1) of the Execution Time Data Storage Block Reservation.
12. DATRES — contains the load time core address of the relocation base for the Execution Time Data Storage Block Reservation.

    (DATRES) = (DATBAS) + (DATLEN)
13. DATSTR — contains the relocation base for the Load Time Data Storage Buffer.
14. ENTPNT — contains the address which is to be inserted into the 4th word of a Loader Table entry.
15. LINK — contains the value which is to be inserted into the 5th word of a Loader Table Entry.

16. LOWCOR — contains the lowest (toward 0) unprotected location available to the loader for storage of relocatable binary input less 1:

    (LOWCOR) = (LWRLIM) if no data storage is declared

    (LOWCOR) = (LWRLIM)+(DATLEN) once a data area is reserved. The contents of LOWCOR becomes the new lower limit of Load Time Core.

17. LWRLIM — contains the lowest unprotected location -1 = lower limit of unprotected core.

18. INPCTR — has 2 uses:

    a) It is used as an address counter during a core-to-core data transfer operation, and

    b) it is set to the 1st storage address for a name to be entered into the Loader Table.

19. INPUT — is the 60 word buffer for storage of relocatable binary input.

20. PROBAS — contains the Execution Time Relocation Base for the program whose NAM block is currently being processed.

21. PRODIF — contains the word count for the amount of command sequence storage placed on the mass storage device during a load operation. (See item 28.8.10.3.) This value is also equal to the difference between the execution time relocation base of a program and its load time storage base providing there is no data block:

    (PROBAS) - (PROSTR) = (PRODIF) if (DATLEN) = 0.

    If there is a data block -

    (PRODIF) = (PROBAS) + (DATLIN) - (PROSTR)

22. PROLIM — contains the execution time value for the upper limit address (last address + 1) of the odd sector (1-95 words in length) stored at bottom of load time core.

23. PROSTR — contains the load time storage base for the program whose NAM block is currently being processed.

24. QSAV — is used for temporary storage of operands in the Q register.

25. SW6 — has two uses:

    a) It is initially cleared to zero. It is set to a -0 once an error condition arises due to overflow of available core.

b) Also, it is used for making Loader Table entries. SW6 is set to a 0 if the Loader Table entry is for an entry point name. It is set to a -0 if the table entry is for an external name.

26. TABLIM contains the base address for storage of the Loader table.

39.9.2.2 Entry to the NAMPRO Routine

The name NAMPRO is declared as an entry point name in the NAMPRO routine and is an external in the LOADER routine. Entry to the NAMPRO routine is with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	NAMPRO	

39.9.2.3 Processing a NAM Block

Common and data storage reservations are set aside when processing a NAM block. The base addresses for common and data storage are printed on the typewriter. In addition, the six character program name in the NAM block is printed on the typewriter together with the base address of the program.

39.9.2.3.1 Reserving Common Storage - NAMPRO

The value equal to the number of words to be set aside for common storage is containeed in the second word of the input buffer, INPUT+1. If (INPUT+1) is zero, NAMPRO does not reserve common storage. Instead, the program transfers immediately to NAMPR1.

If (INPUT+1) is non zero, NAMPRO looks at (COMBAS). Initially (COMBAS) contains zero, but once common storage is declared by a NAM block (COMBAS) contains the relocation base of the common storage. If no common storage reservation had previously been made, NAMPRO will reserve common storage by setting COMBAS to a value equal to (COMLIM) - (INPUT+1). This is otherwise defined as the difference between the upper limit of available core and the number of words in the common storage block. If the Protected Common Flag is set the bounds are left alone.
Once common storage has been reserved, NAMPRO must check for overflow of available core. There will be overflow of available core if the base address of common storage is not greater than the highest address used for command sequence storage:

$$(CSQLIM)-1 \geq (COMBAS)$$

If there is overflow of available core, a -0 will be stored in SW6 and a jump is made to NAMPR1 where the NAM block is processed further. If 2 or more NAM blocks to be processed declare common storage, the largest declaration of common storage should be in the first NAM block to be processed. If (COMBAS) is non zero, the size of the common storage declaration in the current NAM block must not exceed the size of the original block reservation:

$$(COMBAS)+(INPUT+1) - (COMLIM) \leq 0$$

If this condition is met, NAMPRO proceeds to NAMPR1 to further process the NAM block. If not, NAMPRO transfers to the closed error subroutine, PRINT3, to print the error indication "E4". Upon return from PRINT3, NAMPRO transfers to NAMPR1 to further process the NAM block.

39.9.2.3.2 Reserving Data Storage - NAMPR1

Data storage is reserved by the sequence of code beginning at NAMPR1. The value equal to the number of words to be set aside for data storage is contained in the third word of the input buffer, INPUT+2. If (INPUT+2) is zero, NAMPRO does not reserve data storage. Instead, the program transfers immediately to NAMPR2.

If (INPUT+2) is non zero, NAMPRO looks at (DATBAS). Initially DATBAS contains a zero. Once data is declared, (DATBAS) is set to the execution time relocation base of data storage. NAMPRO will reserve data storage by assigning available space in the command sequence storage:

1. (PROBAS) $\longrightarrow$ DATBAS
2. (DATBAS)+(INPUT+2) $\longrightarrow$ DATLIM

In other words:

1. DATBAS is set to the address which would have been the base address of the next program to be loaded had no data storage been declared.
2. The last word address of data storage+1 becomes the base address of the next program to be loaded. It also becomes the upper limit of available core.

Once data storage has been declared, NAMPRO must check for the overflow of available core. If no common storage has been reserved, the address in DATLIM must not exceed the highest address in core:

$$(DATLIM) \leq (COMLIM) \text{ if } (COMBAS) = 0$$

If common storage has been reserved, the address in DATLIM must not exceed the relocation base for common storage:

$$(DATLIM) \leq (COMBAS) \text{ if } (COMBAS) \neq 0$$

If neither of these conditions holds true, an error has occurred due to the overflow of available core. A -0 is stored in SW6, and a transfer is made to NAMPR2 to further process the block.

Once a relocation base for data storage that does not cause overflow of unprotected core has been established, the following occurs:

1. If fewer than 96 words of command sequence storage is contained in unprotected core, no dumping occurs on mass storage. This word count is obtained by

   (PROSTR) - (LOWCOR) -1

   If the word count is 96 or greater, the contents of unprotected core is transferred in integer multiples of 96 words to mass storage. If the number of words to be transferred is not an integer multiple of 96, the last 1-95 words are moved from their current position into the lowest area in unprotected core. This is all accomplished by the COREXT subroutine.

   Upon return, PROSTR contains the storage base for the next load time program. Also, upon return from COREXT, the address in PROLIM is increased by the length of the data storage block:

   (INPUT+2) + (PROLIM) ⟶ PROLIM

2. The number of words to be set aside is recorded in DATLEN:

   (INPUT+2) ⟶ DATLEN

3. A buffer is reserved for data storage (the Load Time Data Storage Buffer). This buffer is used by the Loader for storage of data relocatable command sequence input. The bounds of this buffer are set up by -

   a. (LOWCOR)+1 ⟶ DATSTR

   b. (DATSTR)+(INPUT+2) ⟶ DATCTR, also

   c. (DTACTR) ⟶ CSQCTR

4. Between 0 and 95 words of command sequence (or the "odd sector") remains at the bottom of unprotected core following the dumping operation in step 1. The exact amount is equal to:

   (PROSTR) - (DATSTR)

   upon return from COREXT. Since the Load Time Data Storage Buffer is to occupy Storage at the bottom of unprotected core, the 0-95 words must be moved to storage cells above the Load Time Data Storage Buffer. This means that words

stored in the sequential locations from (DATSTR) to (PROSTR)-1 must be moved to a block or core beginning at (DATCTR) and equal to (PROSTR) - (DATSTR) in length. If this length is zero, there is no core-to-core data transfer, and a jump is made to NAMJ12 (see step number 5).

If (PROSTR) - (DATSTR) $\neq$ 0, the core-to-core transfer is accomplished by the words from the source area to the destination area, the last word first and the 1st word last. In order to implement this data transfer, the location PROSTR is used as the address counter for the source block, and INPCTR as the address counter for the destination block. The address counter INPCTR is set as follows:

(CSQCTR) + (PROSTR) - (DATSTR) $\longrightarrow$ INPCTR

A test is made for overflow of load time core. There is overflow of Load Time core if one of the following conditions exist:

a) Either (INPCTR) > (TABLIM), or -

b) (INPCTR) = (TABLIM) and (INPUT+3) $\neq$ 0, where (INPUT+3) = the amount of Load Time Core to be reserved in addition to the data area for the command sequence of the program whose NAM block is currently being processed.

If a core overflow condition exists, a jump is made to OVFER1 (see step 7). If no ovwrflow condition has occurred, the transfer is made a word at a time until completion when -

(PROSTR) = (DATSTR)

5. The location LOWCOR is set to the new lower limit address for load time core. Prior to setting aside locations for data, the lower limit of load time core coincided with the lower limit of unprotected core:

(LOWCOR) = (LWRLIM) = ($F7)

Subsequent to reserving space at the bottom of unprotected core for the Load Time Data Storage Buffer, the new lower limit of load time core becomes -

(DATCTR) - 1 $\longrightarrow$ LOWCOR.

6. Space is reserved in execution time core for the Execution Time Data Storage Block Reservation (as previously defined in the 1st paragraph of 28.8.2.3.2):

(PROBAS) $\longrightarrow$ DATBAS

(DATBAS) + (INPUT+2) $\longrightarrow$ DATLIM

(DATLIM) $\longrightarrow$ PROBAS & CSQLIM

A check is made for overflow of unprotected core as a result of reserving data. Overflow occurs if:

a) (DATLIM) $>$ (COMBAS) for (COMBAS) $\neq 0$

b) (DATLIM) $>$ (COMLIM) for (COMBAS) $= 0$

c) (INPUT+3) $\neq 0$ and (DATLIM)=(COMBAS) for (COMBAS) $\neq 0$

d) (INPUT+3) $\neq 0$ and (DATLIM)=(COMLIM) for (COMBAS) $= 0$

where (INPUT+3) = length of program relocatable storage of this program. If overflow occurs a jump is made to OVFER1. (See step 7.) Otherwise, the program continues with step 8.

7. At the location whose label is OVFER1, the location SW6 is set to a -0, and a jump is made to NAMPR2 to further process the NAM block.

8. At the location whose name is NAMJ12, the load time core address for the Execution Time Data Storage Block Reservation is recorded in DATRES:

   (CSQCTR) $\longrightarrow$ DATRES

9. Space is reserved in the command sequence in Load Time Core for the Execution Time Data Storage Block Reservation. The difference between this block reservation and the Load Time Data Storage Buffer described in Step 3 is that the latter remains in core until the end of either Subroutine Load or a Program Load Operation.

   The space reserved for the data block in Step 3 will not necessarily remain in core. Along with the rest of the command sequence storage it may be transferred to mass storage. The space reserved at this time will eventually be filled in by the contents of the Load Time Data Storage buffer set aside in Step 3. (Refer to item 28.6.7.)

   Within step 9, the space for the Execution Time Data Storage Block Reservation is set aside by

   (DATRES) + (INPUT+2) $\longrightarrow$ CSQCTR

   (CSQCTR) $\longrightarrow$ PROSTR

10. The difference constant DATDIF is set to the difference between the starting address of the Execution Time Data Storage Block Reservation and the Load Time Data Storage Buffer:

    (DATBAS) - (DATSTR) $\longrightarrow$ DATDIF

11. An additional test is made for overflow of Load Time Core. Overflow of Load Time core occurs if

    (CSQCTR) (TABLIM)

    where (CSQLIM) at the time contains the storage limit for the Execution Time Data Storage Block Reservation in Load Time Core. Also, overflow of Load Time Core occurs if

    (CSQCTR) = (TABLIM)

    and

    (INPUT+3) ≠ 0

    where (INPUT+3) = number of words to be reserved for loading the program whose NAM block is currently being processed.

    If overflow of Load Time Core has not occurred, proceed to step 13. If son, continue at step 12.

12. If the command sequence word count in Load Time Core exceeds 95, a return jump is made to the COREXT routine to transfer an integer multiple of 96 words to mass storage. Upon return from COREXT, the location CSQCTR is set to the limit address in Load Time Core for the Execution Time Data Storage Block Reservation:

    (PROSTR) + (INPUT+2) CSQCTR

    Proceed to step 13.

    If the command sequence word count in Load Time Core is 95 words or less, NAMPRO reacts as if overflow of unprotected core has occurred. This is because the contents of Load Time core must be transferred to mass storage to make room for loading another program. However, in the middle of a Loading Operation, only an integer multiple of 96 words may be transferred to mass storage. Therefore, there isn't enough room in load time core to load another program or to reserve space for data.

    If the overflow condition exists, a jump is made to OVFER1. (See step 7.)

    If, to begin with, there was no overflow of Load Time Core, proceed to step 13.

13. An entry is made in the Loader Table. The name "DATBAS" is entered into the Loader Table as an entry point name. The entry point address is equal to (DATBAS). The sign bit of WORD2 of this entry is set to a 1 as if this Loader Table entry originated in the Table of Presets. The Loader Table entry is made using the TABSTR routine. (Refer to item 28.8.6.)

Prior to entering TABSTR, the following occurs:

a) the location INPCTR is set to the 1st word address for storage of the name "DATBAS"

b) -0 $\longrightarrow$ LINK

c) (DATBAS) $\longrightarrow$ ENTPNT

d) 0 $\longrightarrow$ SW6

Upon return from TABSTR, NAMPRO continues processing the NAM block at NAMPR2.

39.9.2.3.3 Reserving Command Sequence Storage - NAMPR2

Program Relocatable Command Sequence Storage is reserved by the sequence of coding beginning at NAMPR2. The value equal to the number of words to be set aside for program relocatable command sequence storage is contained in the fourth word of the input buffer, INPUT+3. If (INPUT+3) is zero, it is assumed that there is no program relocatable command sequence storage for the relocatable binary program whose NAM block is currently being processed. A transfer is made immediately to NAMPR3.

If (INPUT+3) $\neq$ 0, then the program whose NAM block is currently being processed has an execution time upper limit address equal to

(PROBAS) + (INPUT+3).

If the upper limit for this storage exceeds the upper limit of available core, an overflow error has occurred. The upper limit of core is either the common storage relocation base if common storage had been reserved, or the highest unprotected address in core+1 if no common storage had been reserved. NO overflow has occurred if either

(PROBAS)+(INPUT+3) $\leq$ (COMBAS) if (COMBAS) $\neq$ 0

or

(PROBAS)+(INPUT+3) $\leq$ (COMLIM) if (COMBAS) = 0.

If overflow occurs, the location SW6 is set to a -0, and a jump is made to NAMPR3. If no overflow occurs, the execution time upper limit for program storage is fixed in CSQLIM as follows:

If (CSQLIM) $<$ (PROBAS)+(INPUT+3), (PROBAS)+(INPUT+3) $\longrightarrow$ CSQLIM.

The load time upper limit of program storage is fixed by -

(CSQLIM) - (PRODIF) $\longrightarrow$ CSQCTR

A check is made for overflow into the Loader Table. This occurs when

(CSQCTR) $>$ (TABLIM) -50

Ten Dummy entries are added to prevent a Loader table overflow.

If Loader Table overflow does not occur, a jump is made to NAMPR3. If Loader Table overflow is the case, a check is made to see if the amount of unprotected core available to for Loader input is less than 96 words, the size of 1 sector on mass storage. If

$$(CSQCTR) - (LOWCOR) - 1 \geq 96,$$

the command sequence still in core is entered onto mass storage by a return jump to the COREXT routine. If

$$(CSQCTR) - (LOWCOR) - 1 < 96$$

it is considered to be overflow of unprotected core, and the location SW6 is set to a -0. Processing of the NAM block continues at NAMPR3.

39.9.2.3.4 NAMPR3 - Print Program Name and Execution Time Relocation Base

A test is made to determine if the current Loader Operation is a Program Load Operation. If this is a program Load Operation, bits 0 and 1 of the entrance parameter in AINPUT are set to a binary 10. The program name can execution time relocation base will not be printed, and a jump is made to the location OVFTST to test for core overflow.

If the Loader Operation is not Program Load, the name and execution time relocation base of the program whose NAM block is currently being processed, will be printed. The program name is recorded at the locations INPUT+4, INPUT+5 and INPUT+6. Spaces are recorded at INPUT+2, INPUT+3 and INPUT+7. The address "INPUT+2" is placed in the Q register and the 16 bit value = (PROBAS) is placed in the A register. A return jump is made to the PRINT4 routine (refer to item 28.8.7.3) to list the name and relocation base of the program. During the operation of PRINT4, the 16 bit binary number in A will be converted to the ASCII code for 4 hex digits. The 4 hex digits will be recorded in the output area at locations INPUT+8 and INPUT+9. The program name and execution time relocation base appear on a listing as

SSSSXXXXXSShhhh

where -

1. S indicates a space,
2. X indicates an alphanumeric character (A-Z) or (0-9), and
3. h indicates a hexadecimal digit (0-9) or (A-F).

39.9.2.3.5 OVFTST - Test for Core Overflow Condition

At OVFTST, a check is made for overflow of unprotected core (if SW6 = -0.) If no overflow has occurred, a jump is made to NXTINP to read

and process the next input block. If overflow has occurred, an error indication of "E5" is printed using the PRINT3 subroutine. (Refer to item 28.8.7.1.) No more relocatable binary input blocks are read as Loading terminates. The error exit parameters will be recorded in AINPUT and QINPUT. A jump is made to TABCHK, an entry point in the BRANCH routine where the name TABCHK is referenced in NAMPRO. At TABCHK the Loader Table is examined for unpatched externals. The Procedure has been described previously in item 28.6.6.5.

39.9.2.4 Exit From NAMPRO

If no core overflow had occurred, exit from NAMPRO is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	NXTINP,I	JUMP TO READ NEXT INPUT BLOCK
	EQU	NXTINP(123)	

If overflow of core had occurred, exit is made from NAMPRO with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	TABCHK	

where TABCHK is declared as an external in the NAMPRO routine and as an entry point name in the routine whose name is BRANCH.

39.9.2.5 Subroutines Used by and External to NAMPRO

1. PRINT3 is used to print error indications. (Refer to item 28.8.7.1.)
2. PRINT4 is used to print the program name and execution time relocation base of the program currently being read in. (Refer to item 28.8.7.3.)
3. COREXT is used to dump the command sequence storage place in load time core by the Loader onto the mass storage unit containing the scratch area. The amount of data transferred during any one operation of COREXT is an integer multiple of 96. (Refer to item 28.8.10.)
6. TABSTR is used to make a Loader Table entry when data storage is reserved by the program whose name block is currently being processed. The name "DATBAS" is placed in the Loader Table as an entry point name. The entry point address = (DATBAS). (Refer to item 28.8.6.)

39.9.3 RBDBZS

RBDBZS is the routine which is used to process RBD and BZS blocks.

3?9.3.1 Constant Table Storage Referenced by RBDBZS

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
ASAV	89
BLKCNT	27
BZSSW	26
COMBAS	1
COUNT1	25
CSQLIM	6
DATBAS	2
DATDIF	184
DATLEN	177
DATLIM	5
ENDSW	8
INPCTR	15
INPREL	11
INPWRD	10
INPXCO	117
NGRLSW	9
PROBAS	3
QSAV	90
WRDCNT	24

The constants are used in the following way:

1. ASAV — is used for temporary storage of operands in the A register.
2. BLKCNT — contains the word count for number of sequential locations to be set to zero in a BZS block entry.
3. BZSSW — is set to a 0 if an RBD block is to be processed. Also, it is set to a -1 if a BZS block is to be processed.
4. COMBAS — contains the common storage relocation base.
5. COUNT1 — is used as a loop counter by the NXTWRD subroutine. (Refer to item 28.9.3.6.1.)
6. CSQLIM — contains the execution time core limit address (upper limit) for the command sequence of the program currently being loaded. (This is equal to 1 + address of last word of program at execution time.)
7. DATBAS — contains the relocation base for the Execution Time Data Block Reservation.
8. DATDIF — contains the value equal to (DATBAS) - (DATSTR) where (DATSTR) = the relocation base for the Load Time Data Storage Buffer.

9. DATLEN — contains the number of words set aside for a data reservation.

10. DATLIM — contains the upper limit address (equal to last address + 1) of the Execution Time Data Storage Block Reservation.

11. ENDSW — is the end of block switch. It is set to a 1 when the last entry in the input block (either RBD or BZS) is being processed. Contains a zero at all other times.

12. INPCTR — contains the execution time address assigned to a word from the command sequence input.

13. INPREL — contains in bits 0 and 1, the information which determines which type of address relocation is assigned to a command sequence word from an input block entry.

14. INPWRD — contains a command sequence word from an input block entry.

15. INPXCO — contains the address constant INPUT where INPUT is the 1st location of the area for storage of relocatable binary input blocks read by the Loader.

16. NGRLSW — is the negative relocation switch. It is set to a 1 if negative address relocation is assigned to a word from an input block entry. It is set to a 0 if the address relocation is positive, or if there is no address relocation.

17. PROBAS — contains the Execution Time Relocation Base for the program currently being loaded.

18. QSAV — is used for temporary storage of operands in the Q register.

19. WRDCNT — is used as an address counter to reference words in the input block (either RBD or BZS) entries.

39.9.3.2 Entry to RBDBZS

Both names, RBDPRO and BZSPRO are declared as entry point names in the RBDBZS routine and as externals in the LOADER routine. Entry to either of these routines is as follows:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	RBDPRO	
	JMP	BZSPRO	

39.9.3.3 RBDPRO

Command sequence data from an RBD block is placed in core by RBDPRO. The first entry in the RBD block contains the starting address for loading the command sequence data in the block together with its address relocation byte. Subsequent entries in an RBD block contain

the command sequence words to be recorded at consecutive locations beginning with the starting address. The address relocation byte in an RBD entry is 4 birs in size. The leading bit of all but the relocation byte for the last entry in the block is set to zero. The leading bit of the relocation byte for the last entry in the block is set to a one.

39.9.3.3.1 Initialization for RBD Block Processing at RBDPRO

The jump instructions SW2 and SW3 are two word jump instructions using relative addressing. The address in their respective second words are set during program execution. As part of initialization for RBDPRO, the address of jump instruction SW2 is set to SW2A. The address of jump instruction SW3 will be set to SW3A. Both the loop counter named COUNT1 and the switch named BZSSW are set to zero. The address counter WRDCNT is set to the first word address of the input buffer, "INPUT". Each of these three locations is referenced by a closed subroutine called NXTWRD. This subroutine is used by both RBDPRO and BZSPRO in order to extract all the information pertinent to one entry in the input buffer. The details of the organization of the NXTWRD subroutine are described in item 28.9.3.6.1.

39.9.3.3.2 Starting Address for Command Sequence Storage

A return jump is executed to NXTWRD to extract the starting address for command sequence storage from the input buffer. Upon return from NXTWRD -

1. INPWRD contains the value for the starting address.
2. INPREL contains a 2 bit relocation byte.
3. NGRLSW is set to zero.
4. ENDSW is set to zero.

The starting address may be either program relocatable, data storage relocatable or absolute depending on bits 0 and 1 of INPREL:

1. If (INPREL) = 00, (INPWRD) is an absolute value.
2. If (INPREL) = 01, (INPWRD) is a value relative to the value for the execution time program relocation base of (PROBAS).
3. If (INPREL) = 11, (INPWRD) is a value relative to the value for the relocation base of the Execution Time Data Storage Block Reservation or (DATBAS).

If the starting address is program relocatable, the addresses of the jump instructions SW2 and SW3 contains the values to which they were set as described in item 28.9.3.3.1. If the starting address is data storage relocatable, the addresses for the jump instructions SW2 and SW3 are reset to SW2C and SW3C respectively. The starting address for

relocation by a return jump to the ADJUST subroutine. (Refer to item 28.9.3.6.2.)

If the starting address is either data relocatable or absolute, and-if the address of the jump instructions stored at SW2 and SW3 are to be changed in the manner stated above, then - the addresses are changed prior to entering the ADJUST subroutine. Upon return from the ADJUST subroutine, a jump is made to SW2. Branching occurs according to the address in the jump instruction at SW2 as follows:

1. Branching to SW2A: The execution time limit address for program relocatable input or (CSQLIM) is placed in ASAV. The value = (DATLEN)-(PRODIF) is placed in QSAV, and a jump is made to RBDPR1.
2. Branching to SW2B: The execution time limit address for data storage = (DATLIM) is placed in ASAV. The value in DATDIF is placed in QSAV, and a jump is made to RBDPR1.
3. Branching to SW2C: The limit address communication region storage = "$E4" is placed in ASAV. The location QSAV is cleared to zero, and a jump is made to RBDPR1.

The significance of the values placed in ASAV and QSAV will be described in item 28.9.3.3.3.

39 9.3.3.3 RBDPR1 - Command Sequence Words for RBD Blocks

An entry containing a command sequence word is extracted from the input block by a return jump to the NXTWRD routine. Upon return from NXTWRD -

1. INPWRD contains the word of the command sequence.
2. INPREL contains the 2^0 and 2^1 bits of the 4 bit relocation byte.
3. NGRLSW contains 2^2 bit of the 4 bit relocation byte.
4. ENDSW contains the 2^3 bit of the 4 bit relocation byte.

The location ENDSW will always contain a zero except when the last entry in the block is to be processed. It will then contain a value of 1. The location NGRLSW will contain a zero if the word in INPWRD is either a 16 bit absolute value or bits 0-14 = a relative value with positive address relocation. NGRLSW will contain a value of 1 if the value of bits 0-14 of the location INPREL will be set to one of 4 values:

1. (INPREL) = 00 if (INPWRD) is an absolute value.
2. (INPREL) = 01 if (INPWRD) is a value relative to the execution time program relocation base or (PROBAS).

3. (INPREL) = 10 if (INPWRD) is a value relative to the common storage relocation base of (COMBAS).

4. (INPREL) = 11 if (INPWRD) is a value relative to the relocation base for the Execution Time Data Storage Block Reservation.

Adjusting the contents of INPWRD for address relocation is accomplished in the following way:

TYPE OF RLCTN	NGRLSW	INPREL	VALUE ADJUSTED FOR RLCTN
ABSOLUTE	0	00	(INPWRD)
POS PROGR. RLCTN	0	01	(INPWRD)+(PROBAS)
POS COM STOR RLCTN	0	10	(INPWRD)+(COMBAS)
POS DAT STOR RLCTN	0	11	(INPWRD)+(DATBAS)
NEG PROG RLCTN	1	01	(INPWRD)-(PROBAS)
NEG COM STOR RLCTN	1	10	(INPWRD)-(COMBAS)
NEG DAT STOR RLCTN	1	11	(INPWRD)-(DATBAS)

Where negative relocation is involved, the value in INPWRD is actually the one's complement of a 15 bit number. This means that negative relocation is actually

-(relative value + base address)

or the complement of positive relocation. The address arithmetic necessary for adjusting a relative value for relocation is accomplished by the closed subroutine ADJOVF. The closed subroutine is entered by a return jump to a location in the constant table. (Refer to item 28.8.3.)

The address in INPCTR is the execution time address for the command sequence word stored in INPWRD. Upon return from ADJOVF, storage of the word in INPWRD takes place as follows:

If ((INPCTR))$<$(ASAV), (INPWRD) $\longrightarrow$ (INPCTR) - (QSAV),

Storage occurs if the execution time storage address is less than the limit address. The address in INPCTR is increased by 1. If (ENDSW)=0, a jump is made back to RBDPR1 to process the next word in the RBD block. If (ENDSW)$\neq$0 a jump is made to NXTINP to process the next input block.

If (INPCTR) $\geq$ (ASAV), the input has exceeded the limits of the area reserved for it. Branching occurs according to the address in the jump instruction as SW3 as follows:

1. Branching to SW3A: An error has occurred due to overflowing the amount of space reserved in esecution time core for program relocatable command sequence currently being loaded. An error indication of "E5" is printed using the PRINT3 subroutine. (See item 28.7.1.)

2. Branching to SW3B: An error has occurred due to overflowing the amount of space reserved for the data block. An error indication of "E7" is printed using the PRINT3 subroutine.

3. Branching to SW3C: An error has occurred due to overflowing the amount of space in the unprotected area of the communication region. An error indication of "E6" is printed using the PRINT3 subroutine.

If one of the above error messages is printed, no more input blocks are read and the Loading Operation terminates. The error exit information is recorded and a jump is made to TABCHK. (Refer to item 28.9.3.6.)

39 9.3.5 BZSPRO

Each entry in a BZS block contains the following information:

1. A starting address which is the first word address of a block of core to be cleared to zero by the loader.

2. A relocation byte for the starting address.

3. An absolute number which is the size of the block of core to be cleared to zero by the loader.

The relocation byte for each entry in a BZS block is 4 bits in size. The leading bit of all the relocation byte for the last entry in the block is set to zero. The leading bit of the relocation byte for the last entry in the block is set to a one.

39 9.3.4.1 Initialization for BZS Block Processing

As part of initialization for BZSPRO, the address counter WRDCNT is set to the 1st word address of the storage area for the BZS block, = "INPUT". The switch named BZSSW is set to a -0 in order to process a BZS block.

39 9.3.4.2 BZSPR1 - BZS Entries

The jump instructions SW2 and SW3 are two word jump instructions using relative addressing. The addresses in their respective second words are set during program execution. Each time a jump is made to the location BZSPR1, the address in the jump instruction at SW2 is set for SW2A, and the address in the jump instruction at SW3 is set for SW3A.

The loader executes a return jump to NXTWRD to extract an entry from the BZS block. Upon return from NXTWRD the information from the entry is stored in the following way:

1. INPWRD contains the starting address.
2. INPREL contains the 2^0 and 2^1 bits of the 4 bit relocation byte.
3. NGRLSW contains the 2^2 bit of the 4 bit relocation byte.
4. ENDSW contains the 2^3 bit of the 4 bit relocation byte.
5. BLKCNT contains the size of the block to be cleared to zero by the Loader.

The location ENDSW will always contain a zero except when the last entry in the block is to be processed. It will then contain a value of 1. The location NGRLSW will always contain a zero since the word in INPWRD is either a 16 bit absolute value or a 15 bit relative value with either positive program or data storage relocation. The location INPREL will be set to one of 3 values:

1. (INPREL) = 00 if (INPWRD) is an absolute value.
2. (INPREL) = 01 if (INPWRD) is a value relative to the value for the execution time program relocation base or (PROBAS).
3. (INPREL) = 11 if (INPWRD) is a value relative to the value for the relocation base of the Execution Time Data Storage Block Reservation.

39.9.3.4.3 Relocation for Starting Address

A return jump is executed to the closed subroutine, ADJUST, to obtain the absolute value for a relative address in INPWRD for relocation. (Refer to item 28.9.3.6.2.) Upon return starting from ADJUST -

1. INPCTR contains the starting address for command sequence storage adjusted for relocation.
2. If the starting address for the BZS block is relative to the execution time data storage relocation base, the address in the jump instruction SW2, originally set to SW2D is reset to SW2E, while that in SW3 is set to SW3E.
3. If the starting address for the BZS block is absolute, the address in jump instruction SW2, originally set to SW2D is reset to SW2F, while that in SW3 is set to SW3F.

39.9.3.4.4 Zero Storage in BZS Block

The Loader proceeds to store zero at all execution time locations starting with the address in INPCTR and terminating with the address equal to (INPCTR) + (BLKCNT)-1.

A jump is made to SW2 where branching occurs according to the address in the jump instruction at SW2 as follows:

1. Branching to SW2D: The execution time limit address for program relocatable input or (CSQLIM) is placed in ASAV. The value = (DATLEN)-(PRODIF) is placed in QSAV, and a jump is made to BZSPR2.
2. Branching to SW2E: The execution time limit address for data storage = (DATLIM) is placed in ASAV. The value = (DATDIF) is placed in QSAV, and a jump is made to BZSPR2.
3. Branching to SW2F: The limit address for communication region storage = "$E4" is placed in ASAV. The location QSAV is cleared to zero, and a jump is made to BZSPR2.

   At BZSPR2 a test is made to see if the storage address in INPCTR is greater than or equal to the storage limit address in ASAV. If not, the load time storage location = (INPCTR) - (QSAV) is cleared to zero. The loop for storing zero in a location is repeated, increasing the contents of INPCTR by 1 and decreasing the contents of BLKCTR by 1 until -

   a. Either (BLKCTR)=0 in which the entire block of core is cleared to zero, or
   b. (INPCTR) = (ASAV) in which case the storage limit address has been exceeded.

If b is the case, branching occurs as SW3 as follows:

1. Branching to SW3D will occur if the starting address in the BZS entry is relative to the execution time program relocation base. The address "SW3D" and "SW3A" are identical.
2. Branching to SW3E will occur if the starting address is relative to the relocation base for the data block. The addresses "SW3B" and "SW3E" are identical.
3. Branching to SW3F will occur if the starting address is absolute. The address "SW3F" and "SW3C" are identical.

If a 2 is the case, the overflow condition has not occurred, and a jump is made to BZSPR3. At BZSPR3, a test is made to see if the last entry in the BZS block has been processed. If (ENDSW)=0, a jump is made to BZSPR1 to process the next entry in the block. If (ENDSW)≠0, a jump is made to NXTINP to read the next input block.

39.8.3.5 Exit from RBDBZS

Exit from either routine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP- EQU	NXTINP,I NXTINP(123)	JUMP TO NXTINP TO READ NEXT INPUT BLOCK

If exit from either routine is caused by the occurrence of an overflow condition, exit from the RBDBZS routine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	TABCHK	

where the name TABCHK is referenced as an external within RBDBZS and as an entry point in the BRANCH routine.

39.9.3.6 Subroutines Used by and Internal to RBDPRO and BZSPRO

There are two subroutines assembled into the same program with RBDPRO and BZSPRO. These are NXTWRD and ADJUST.

39.9.3.6.1 NXTWRD

This subroutine is used to extract an entry from either an RBD block or a BZS block. An RBD block entry consists of a command sequence word together with its 4 bit relocation byte. A BZS block entry consists of a starting address for a BZS block reservation together with its 4 bit relocation byte and an absolute number which is the size of the BZS block reservation. If entry to NXTWRD is from RBDPRO, the switch BZSSW has been set to zero. If entry to NXTWRD is from BZSPRO, the switch BZSSW has been made non zero and negative.

If BZSSW is zero, NXTWRD will process an entry of the input block in the following ways:

1. Of a 4 bit relocation byte (bits 0-3) right to left,
   a. bits 0 and 1 are placed in INPREL,
   b. bit 2 is placed in NGRLSW, and
   c. bit 3 is placed in ENDSW.
2. The command sequence word is placed in INPWRD.

If BZSSW is non zero, NXTWRD will process an entry of the input block in the following way:

1. Of a 4 bit relocation byte (bits 0-3 right to left):
   a. bits 0 and 1 are placed in INPREL,
   b. bit 2 is placed in NGRLSW, and
   c. bit 3 is placed in ENDSW.
2. The starting address for the BZS block reservation is placed in INPWRD.

3. The size of the BZS block reservation is placed in BLKCNT.

39.9.3.6.2 ADJUST

The ADJUST subroutine is used by BZSPRO and RBDPRO to

1. Determine the relocation base for the starting address of the BZS reservation in each entry of the command sequence, and
2. If the address is not absolute, to increase the relative value by the appropriate relocation base.

Upon entry to ADJUST, the address of concern is in INPWRD. All 15 bit address arithmetic is carried out of the ADJOVF subroutine. Prior to entering ADJOVF, the A register contains the relative address contained in INPWRD. The relocation base is in the Q register. Upon return from ADJOVF, the result is placed in INPCTR.

Address relocation is handled in one of three ways:

1. Absolute Addresses = No Relocation

   No address relocation is necessary for absolute address. The contents of INPWRD is placed in INPCTR:

   (INPWRD) ⟶ INPCTR

   If the ADJUST routine determines that the value for the absolute starting address is either less than "$C5" or greater than "$E3", a jump is made to SW3 to print the appropriate error indication and then to proceed to the error exit. If ADJUST determines that the absolute starting address was within the bounds of $C5-$E3, the addresses for the jump instructions at SW2 and SW3 are reset as follows:

   a. If entry to ADJUST was from RBDPRO,

      SW2C ⟶ SW2, and

      SW3C ⟶ SW3.

   b. If entry to ADJUST was from BZSPRO,

      SW2F ⟶ SW2, and

      SW3F ⟶ SW3.

   Following this a jump is made from ADJUST.

2. Program Relocatable Addresses

   Following a return transfer to ADJOVF, the relative address is increased by the command sequence relocation base, and

the result is placed in INPCTR:

(INPWRD) + (PROBAS) ⟶ INPCTR

A jump is made to exit from ADJUST.

3. Data Storage Relocatable Addresses

Following a return transfer to ADJOVF, the relative address is increased by the data storage relocation base, and the result is placed in INPCTR:

(INPWRD) + (DATBAS) ⟶ INPCTR

a. If entry to ADJUST was made from RBDPRO,

SW2B ⟶ SW2, and

SW3B ⟶ SW3.

b. If entry to ADJUST was made from BZSPRO,

SW2E ⟶ SW2, and

SW3E ⟶ SW3.

39.9.3.8 Subroutines Used by and External to RBDPRO and BZSPRO

1. PRINT3 is used for printing error indications. (Refer to item 28.8.7.1.)
2. ADJOVF is used by ADJUST for address arithmetic. (Refer to item 28.8.2.)

39.9.4 ENTEXT

The ENTEXT subprogram of the Loader is used to process EXT and ENT blocks.

39.9.4.1 Constant Table Storage Referenced by ENTEXT

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
DATLEN	177
ENTPNT	13
INPCTR	15
INPWRD	10
INPXCC	122
LINK	14
NGRLSW	9
PROBAS	3
PRODIF	171
PROLIM	181
SW6	28
TABLIM	7

The constants are used in the following way:

1. DATLEN — contains the number of words set aside for the data reservation.
2. ENTPNT — contains the address (or the one's complement thereof) which is to be inserted into the 4th word of a Loader Table entry. For a LINK1 operation (patching), it contains an entry point address. For a LINK2 operation (linking), it contains the link address from the EXT block entry.
3. INPCTR — is used as an address counter to reference entries in the input block (either ENT or EXT).
4. INPWRD — is used as a temporary storage location.
5. INPXCC — contains the address constant = "INPUT-3".
6. LINK — contains the address to be inserted into the 5th word of a Loader Table entry. For either a LINK1 operation (patching) or a LINK2 operation (linking) it contains the link address from the Loader Table entry.
7. NGRLSW — is set prior to LINK1 operation to determine mode of addressing for patching (either relative or absolute).
8. PROBAS — contains the execution time relocation base for the program currently being loaded.
9. PRODIF — contains the word count for the amount of command sequence recorded on the mass storage unit containing the scratch area.
10. PROLIM — contains the execution time value for the upper limit (last address + 1) of the odd sector (1-95 words long) stored at the bottom of Load Time Core.
11. SW6 — is set to 0 to enter an entry point into the Loader Table, or it is set to a -1 to enter an external. Also, as a result of a Loader Table Search, it contains a positive value which is added to (TABLIM) in order to locate a Loader Table entry with a name to match an input block entry.
12. TABLIM — contains the base address of the Loader Table.

39.9.4.2 Entry to ENTEXT

The names ENTPRO and EXTPRO are declared as entry point names in the ENTEXT routine and as external names in the LOADER Routine. Entry to either of these routines is as follows:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	ENTPRO	
	JMP	EXTPRO	

3912.4.3 ENTPRO

An entry in an ENT block contains a 6 character entry point name and a 15 bit entry point address. ENTPRO will perform one of three functions based on the follwoing conditions:

CONDITION 1: is that there exists no entry in the Loader Table which contains a 6 character name to match the entry point name from the input block. ENTPRO will enter the entry point name and address into the loader table. WORD5 of the Loader Table entry is set to -0.

CONDITION 2: is that there exists an entry in the Loader Table which contains a 6 character entry point name which matches the 6 character entry point name in the input block. The loader performs a special error operation where there is duplication of entry point names.

CONDITION 3: is that there exists an entry in the Loader Table which contains a 6 character name for an external which matches the entry point name in the input block. The link address in WORD 4 of the Loader Table entry points to the beginning of a string of link addresses. The link addresses in this string are contained in command sequemce storage entirely within the limits of Load Time Core, exclusive of the area at the bottom of Load Time Core containing the 1-95 words of the odd sector. In other words, the execution time value for each link address in the string must be greater than or equal to (PROLIM). ENTPRO using the LINK1 routine, will perform the necessary link operation to patch locations referencing this external name with the entry point address.

CONDITION 4: is that there exists an entry in the Loader Table which contains a 6 character name for an external which matches the entry point name in the input block. The link address in WORD 4 of the Loader Table entry points to the beginning of a string of link addresses. The link addresses in this string are contained in command sequence storage which is recorded either entirely or in part on mass storage. In other words, in the link address string for this external name there is at least one link address whose execution time value is less than (PROLIM).

Using the TABSTR routine, ENTPRO will enter the entry point name and address in the input block into the Loader Table. WORD 5 of this Loader Table entry will contain the address pointing to WORD 4 of the entry containing the matching external name. A linking operation similar to that described for condition 3 will be performed, at the time an EOL record is read by the Loader. (See item 28.9.7.)

39.9.4.3.1 ENTPR1 - ENT Block Entries

ENT block entries are four words each. The first three words contain a 6 character entry point name. (Fewer than 6 characters in a name means that the ASCII code for spaces will be used as fill at the right end of the name.) If the entry point address is program relocatable, the fourth word of the entry will contain a positive number which is the relative 15 bit entry point address. If the entry point address is absolute, the fourth word of the entry will contain a negative number which is the one's complement of the 15 bit entry point address. The end of the input block is marked by a word of either +0 or -0 in place of an entry point name.

INPCTR is used by the closed subroutine NXTNAM (see item 28.9.4.6.1) to extract entries from the ENT block. Initially INPCTR is increased by 4 upon each entry to NXTNAM from ENTPRO. This will set INPCTR to the first word address of the next entry in the ENT block or, as a result of the return jump to NXTNAM either -

1a. -((INPCTR)+3) ENTPNT
if ((INPCTR)+3) 0, or -

1b. ((INPCTR)+3) + (PROBAS) ENTPNT
if ((INPCTR+3) _ 0, and

2. (INPCTR) + 4 INPCTR

There will be no return from NXTNAM if ((INPCTR)) = 0. Instead, there will be a jump from the NXTNAM subroutine to NXTINP to process the next input block.

39.9.4.3.2 Searching the Loader Table for Matching Name

There is a return jump to the closed subroutine TABSCH to find an entry in the Loader Table containing a 6 character name to match the entry point name from the input block. Upon return from TABSCH only one of the following occurs:

1. If a matching name had been found, the first address of the Loader Table entry is (TABLIM)+(SW6) where 0

   a. (TABLIM) is the lowest core address occupied by the Loader Table, and -

   b. (SW6) is an increment equal to the difference between the lowest address of the Loader Table and the first address of the Loader Table entry with the matching name.

2. If a matching name had not been found, (SW6) is negative.

39.9.4.3.3 No Matching Name

If a matching name is not found, the entry point name from the input block must be entered into the Loader Table. The Loader Table entry is made as follows:

1. LINK is set to -0,
2. SW6 is set to 0, and
3. a return jump is made to TABSTR.

Upon return from TABSTR, (ENTPNT) will have been placed in WORD 4 and (LINK) will have been placed in WORD 5 of the Loader Table entry. Following the terutn from TABSTR, there is a jump to ENTPR1 to process the next input block.

39 .9.4.3.4 Matching Name Found

If a matching name had been found in the Loader Table, the 1st word address of the entry is given by

(TABLIM)+(SW6)

The location (TABLIM)+(SW6)+3 contains the address for this name. This address is then pladed in LINK. If this address is an entry point address, then -

(LINK) > 0

If this address is a link address, then

(LINK) < 0.

39 .9.4.3.5 Matching Name is Entry Point

It is an error condition of an entry point name in the ENT block matches an entry point name in the Loader Table. A return jump is made to PRINT3 to print the error message "E8". A return jump is also made to PRINT5 to list the entry point name. A jump is made to the location whose label is STOP, whereupon the Loading operation is terminated.

39.9.4.3.6 Matching Name is External

If the matching name found in the Loader Table entry is an external name, the contents of the location LINK is the one's complement of the 1st address in the link address string. The contents of LINK is therefore complemented in order to obtain the link address. Next, it is necessary to determine if the entire link address string is or is not contained within the limits of Load Time Core (exclusive of the area containing the odd sector). In order to test for this, a return jump is made to a closed subroutine within the ENTEXT subprogram. The location at which this routine is entered has the name EXAMEN. Upon return from the EXAMEN subroutine one of these two conditions exists:

CONDITION 1: (A) = -0 if the link address string is contained entirely within the desired area of Load Time Core (load Time Core exclusive of the odd sector), or -

CONDITION 2: (A) = 0 if the link address string is recorded either entirely or in part on the mass storage unit containing the scratch area.

If Condition no. 1 occurs, then -

1. WORD 4 of the Loader Table entry for the matching name is filled with the entry point address:

   (ENTPNT) $\rightarrow$ (TABLIM)+(SW6)+3

   (The link address for this entry is replaced with an entry point address.)

2. A return jump is made to the LINK1 routine to replace each link address in the string with the entry point address. Upon return from LINK1 a jump is made to ENTPR1 to process the next entry in the ENT block.

If Condition no. 2 occurs, then -

1. The address = "(TABLIM)+(SW6)+3" is placed in the location LINK, while (ENTPNT) = the entry point address.
2. The location SW6 is cleared to zero.
3. A return jump is made to TABSTR.

Upon return from TABSTR, the address in ENTPNT was placed in WORD 4 of the new Loader Table entry, while the address in LINK was placed in WORD 5. A jump is made to ENTPR1 to process the next entry in the block.

39.9.4.4 EXTPRO

An entry in an EXT block contains a 6 charzcter external name and a 15 bit link address. EXTPRO will perform one of three functions based on the following conditions:

CONDITION 1: is that there exists no entry in the Loader Table which contains a 6 character name to match the external name from the input block. EXTPRO will enter the external name and the link address into the Loader Table. WORD 5 of this Loader Table entry is set to a -0.

CONDITION 2: is that there exists an entry in the Loader Table which contains a 6 character entry point name which matches the external name in the input block. The link address for the name in the input block entry is the beginning of a link address string contained entirely within the desired limits of Load Time Core. EXTPRO, using the LINK1 routine, will perform the necessary link operation to patch locations referencing this external name with the entry point address.

CONDITION 3: is that there exists an entry in the Loader Table which contains a 6 character external name which matches the 6 character external name in the input block. The link address in WORD 4 of the Loader Table entry is the beginning of a link address string contained entirely within the desired limits of Load Time Core. The link address from the entry in the input block is also the beginning of a string contained entirely within the desired limits of Load Time Core. The link address string referenced by WORD 4 of the Loader Table entry is "tied" to the end of the link address string referenced by WORD 4 of the input block entry. This is accomplished by using the LINK2 subroutine.

CONDITION 4: is that there exists an entry in the Loader Table which contains a 6 character name for an external which matches the external name in the input block. The link address in WORD 4 of the Loader Table entry points to the beginning of a string of link addresses. However, this link address string is contained in command sequence storage recorded either entirely or in part on the mass storage device containing the scratch area. Using the TABSTR routine, EXTPRO will enter the external name and link address from the input block entry into the Loader Table. WORD 5 of the new Loader Table entry will contain the address pointing to WORD 4 of the previous Loader Table entry containing the matching external name.

39.9.4.4.1 EXTPR1 - EXT Block Entries

EXT block entries are four words each. The first three words contain a 6 character external name. (Fewer than 6 characters in a name means that the ASCII code for spaces will be used as fill at the right end of the name.) If the link address is program relocatable, the

fourth word of the entry will contain a positive number which is the relative 16 bit link address. If the link address is absolute, the fourth word of the entry will contain a negative number which is the one's complement of the 15 bit entry point address, the end of the input block is marked by a word of zeros in place of an entry point name.

INPCTR is used by the closed subroutine NXTNAM to extract entries from the EXT block. Initially INPCTR is set to the address "INPUT-3" by EXTPRO. The contents of INPCTR is increased by 4 upon each entry to NXTNAM from EXTPRO. This will set INPCTR to the first word address of the next entry in the EXT block. As a result of the return jump to NXTNAM either -

1a. -((INPCTR)+3) $\longrightarrow$ ENTPNT

if ((INPCTR)+3) <0, or -

1b. ((INPCTR)+3)+(PROBAS) $\longrightarrow$ ENTPNT

if ((INPCTR)+3) ≥ 0, and -

2. (INPCTR)+4 $\longrightarrow$ INPCTR

There will be no return from NXTNAM if ((INPCTR))=0. Instead there will be a jump from NXTNAM to NXTBLK to process the next input block.

39 .9.4.4.2 Searching Loader Table for Matching Name

There is a return jump to the closed subroutine TABSCH to find an entry in the Loader Table containing a 6 character name to match the external name from the input block. Upon return from TABSCH only one of the following occurs:

1. If a matching name had been found, the first word address of the Loader Table entry is (TABLIM)+(SW6), where -
   a. (TABLIM) is the 1st address of the Loader Table, and -
   b. (SW6) is an increment equal to the difference between the lowest address of the Loader Table and the first address of the Loader Table entry with the matching name.
2. If a matching name had not been found, (SW6) is negative.

39 .9.4.4.3 No Matching Name

If a matching name is not found, the external name from the input block must be entered into the Loader Table. The Loader Table entry is made as follows:

1. LINK is set to -0,
2. SW6 is set to -0, and
3. a return jump is made to TABSTR.

Upon return from TABSTR, (ENTPNT) will have been placed in WORD 4 of the Loader Table entry, and (LINK) will have been placed in WORD 5,. Following the terutn from TABSTR there is a jump to EXTPR1 to process the next input block.

39 .9.4.4.4 Matching Name Found

If a matching name had been found in the Loader Table, the 1st address of the entry is given by -

(TABLIM)+(SW6).

The location (TABLIM)+(SW6)+3 contains the address for the name. This address is then placed in LINK. If this address is an entry point address, then -

(LINK) $\geq$ 0.

If this address is a link address, then -

(LINK) $<$ 0.

39 .9.4.4.5 Matching Name is Entry Point

If the matching name in the Loader Table entry is an entry point name, the contents of the location LINK is the one's complement of the 1st address in a link address string. The contents of LINK are therefore complemented in order to obtain the link address. Next, it is necessary to determine if the entire link address string is or is not contained within the limits of Load Time Core (exclusive of the area containing the odd sector). In order to test for this, a return jump is made to a closed subroutine within the EXTENT subprogram. The location at which this routine is entered has the name EXAMEN. Upon return from the EXAMEN subroutine one of these two conditions exists:

CONDITION 3: (A) = -0 if the link address string is contained entirely within the desired area of Load Time Core (Load Time Core exclusive of the odd sector), or -

CONDITION 4: (A) = 0 if the link address string is recorded either entirely or in part on the mass storage unit containing the scratch area.

If condition 3 occurs, the beginning of the link address string referenced by the Loader Table entry is tied to the end of the link address string referenced by the input block entry. This is accomplished by a return jump to the LINK2 subroutine. (See item 28.8.9.)

If Condition 4 occurs, theprocedure to be followed is that specified by Condition 2 of item 28.9.4.3.6. However, prior to the return jump to TABSTR, the location SW6 is set to -0 instead of 0.

39.9.4.5 Exit from ENTEXT

The exit from either of these routines is with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	NXTINP,I	JUMP TO NXTINP TO READ
	EQU	NXTINP(123)	NEXT INPUT BLOCK

In the event the same name is declared as an entry point name by more than one program being loaded, an exit is made from the EXTENT routine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	STOP	
	JMP	STOP	

where the name STOP is an entry point in the LSTOUT subprogram. (See item 28.8.7.2.)

39.9.4.6 Subroutines Used by and Internal to ENTPRO and EXTPRO

The NXTNAM routine is used by both ENTPRO and EXTPRO to extract the next entry to be processed from the input block. The EXAMEN routine will slew through a string of link addresses to determine if they are entirely in core, or if they are either entirely or in part located on mass storage.

39.9.4.6.1 NXTNAM

This subroutine is used to extract an entry from either an EXT block or an ENT block. A single word entry in either type of block consists of a 6 character name and a 15 bit address as follows:

WORD 1	Char 1	Char 2
WORD 2	Char 3	Char 4
WORD 3	Char 5	Char 6
WORD 4	Address	

Upon entry to NXTNAM, the contents of INPCTR is increased by 4 to set it to the first word address of the next entry to be processed. If ((INPCTR)) is either a +0 or a -0, the last entry of the input block had been processed.

NXTNAM will jump to NXTINP to process the next input block. If ((INPCTR)) is not zero, NXTNAM will look at the 15 bit address in the last word of the entry. If ((INPCTR)+3) is negative, it is the one's complement of a 15 bit address which is absolute. NXTNAM will place the one's complement of the contents of the 4th word of the entry in ENTPNT:

$$-((INPCTR)+3) \longrightarrow ENTPNT$$

If this address is positive, it is a program relocatable address. NXTNAM will add the relative value in word 4 of the entry to the program relocation base and place the result in ENTPNT:

$$(INPCTR)+4 \longrightarrow INPCTR$$

A jump is made to the exit from NXTNAM.

39 .9.4.6.2 EXAMEN

This routine will slew through a link address in the following way:

1. Get the first link address in the string = (LINK). If (LINK) = $7FFF the end of the string has been reached. The entire string sits in Load Time Core. Place a -0 in the A register and go to exit. If (LINK) ≠ $7FFF, record address and go to step 2.
2. If execution time value of current link address in string is less than (PROLIM), the link address is not within the desired limits of Load Time Core, Place a +0 in the A register and go to exit. Otherwise, continue at Step 3.
3. Given the execution time value for the current link address in string, compute the load time value:

   load time address = execution time address + (DATLEN)-(PRODIF).
4. Given the load time value of the current link address, go to that location, pickup the execution time value of the next link address in the string. Replace current link address with next one in string. Go back to step 2 and continue operating in slew loop.

39.9.4.7 Subroutines Used by and External to ENTPRO and EXTPRO

1. TABSCH is used to search the Loader Table for a name to match the name in the input block entry being processed. (See item 28.8.5.)
2. TABSTR is used to enter a name from the input block entry being processed into the Loader Table. (See item 28.8.6.)
3. LINK1 is used to patch each location in a string of link addresses with the entry point address for the name in the input block entry being processed. (See item 28.8.8.)
4. LINK2 is used to tie two strings of link address together. (See item 28.8.9.)
5. PRINT3 is used by ENTPRO to print an error indication when it encounters an entry point name in the input block which matches an entry point name in the Loader Table. (See item 28.8.7.1.)
6. PRINT5 is used by ENTPRO to print the entry point name causing the error described in 5. (See item 28.8.7.4.)

39.9.5 XFRPRO

XFRPRO is the routine which processes an XFR block.

39.9.5.1 Constant Table Storage Referenced by XFRRPO

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
AINPUT	161
BLANKS	18
CSQCTR	183
CSQLIM	6
DATLEN	177
ENDINP	17
INPUT	29-88
PROBAS	3
PROSTR	185
XFRNAM	92-94

The constants are used as follows:

1. AINPUT contains one of the entrance parameters for the Loader Operation.
2. BLANKS a constant = ASCII code for 2 spaces. (BLANKS) = $2020.

3. CSQCTR — contains one plus the highest load time address occupied by the command sequence storage of the program whose XFR block is currently being processed.

4. CSQLIM — contains one plus the highest execution time address occupied by the command sequence storage of the program whose XFR block is currently being processed.

5. DATLEN — contains the number of words set aside for the data block.

6. ENDINP — contains the highest load time address for command sequence storage.

7. INPUT — is the 1st location of a 60 word area reserved for storage of relocatable binary input.

8. PROBAS — is set to the execution time relocation base for the next relocatable binary program to be loaded.

9. PROSTR — is set to the load time storage base for the next relocatable binary program to be loaded.

10. XFRNAM — contains the 6 character transfer name from the last XFR block processed which had a name (see item 28.9.5.3.)

39.9.5.2 Entry to XFRPRO

The name XFRPRO is declared as an entry point name in the XFRPRO routine and as an external in the LOADER routine. Entry to XFRPRO is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	XFRPRO	

39.9.5.3 Recording Transfer Name

When the Loader is brought into core, the locations XFRNAM, NFRNAM+1 & XFRNAM+2 are set to zero. If the XFR block contains a transfer name, it is recorded in the locations INPUT+1, INPUT+2, and INPUT+3. If the XFR block does not contain a transfer name, each of these 3 locations contains the ASCII code for 2 spaces.

If the XFR block contains a name, XFRPRO replaces the current contents of XFRNAM, XFRNAM+1 and XFRNAM+2 with the ASCII code for this 6 character name.

39.9.5.4 Recording Relocation Base for Next Program

Prior to the exit from XFRPRO:

1. (CSQLIM) ⟶ PROBAS where (PROBAS) = exec. time relocation base of next program to be read in.
2. (CSQCTR) ⟶ PROSTR where (PROSTR) = load time storage base of next progran to be read in.
3. (CSQCTR)+(DATLEN) ⟶ ENDINP where (ENDINP) - highest load time core location to be occupied by command sequence storage.

39.9.5.5 Exit from XFRPRO

A test is made to determine the type of Loader Operation is in progress. If bits 0 and 1 of AINPUT are set to 0, the Loader Operation is Relocatable Binary Load. Exit from XFRPRO is then made with the following instruction.

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	NXTINP,I	GO TO NXTINP TO BEGIN
	EQU	NXTINP(123)	LOAD OF NEXT PROGRAM

Upon execution of this instruction the Loader will look for the next input block which must be one of the following:

1. The NAM block of the next program,
2. a hex block
3. An EOL block
4. if no data is read by the Loader, it is the same as if an EOL block had been received. (See item 29.1.4.1.)

If the Loader Operation is either a Subroutine Load or a Program Load, exit from XFRPRO is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	ALRMOK	

where the name ALRMOK is referenced as an external in XFRPRO and as an entry point in the LOADER routine. (See item 28.9.1.4.1.)

39.9.6 HEXPRO

HEXPRO is the routine which is used to process a HEX block.

39 .9.6.1 Constnat Table Storage Referenced By HEXPRO

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
ASAV	89
BLANKS	18
COMBAS	1
CSQCTR	183
CSQLIM	6
DATBAS	5
DATCTR	182
DATDIF	184
DATLEN	177
DATSTR	180
DIFCON	186
INPCTR	15
INPREL	11
INPUT	29-88
INPWRD	10
INPXC1	118
LOWCOR	179
LWRLIM	191
NAME	95-98
NGRLSW	9
PROBAS	3
PRODIF	171
QSAV	90
SCANSW	22
SECTNO	173-174
SECTOR	198
SW6	28
SYMSTR	19
TABLIM	7
WRDCNT	24

The constants are used as follows:

1. ASAV — is used for temporary storage of the operand in the A register.
2. BLANKS — contains the ASCII code for 2 spaces = $2020.
3. COMBAS — contains the common storage relocation base.
4. CSQCTR — contains the limit address of the input area (last address + 1) for storage of hex correction constant.
5. CSQLIM — contains upper limit of command sequence (last address + 1) at execution time.
6. DATBAS — contains the relocation base for the Execution Time Data Storage Block Reservation.

7. DATCTR — contains the limit address of the Load Time Data Storage Buffer.

8. DATDIF — contains the difference equal to -

(DATBAS)-(DATSTR).

9. DATLEN — contains the number of words to be set aside for the data block reservation, = (DATCTR)-(DATSTR).

10. DATSTR — contains the relocation base for the Load Time Data Storage Buffer.

11. DIFCON — is used for temporary storage.

12. INPCTR — is the address counter which is used for storage of hex correction constants in the input area.

13. INPREL — contains the terminating character of a field in the HEX block when the feild is extracted using the SCAN routine.

14. INPUT — is the storage area for input blocks read by the Loader.

15. INPWRD — contains the binary value for numeric fields extracted from the input area by the SCAN routine. (See item 28.8.1.)

16. INPXC1 — contains the address constant "INPUT+1".

17. LOWCOR — contains the lower limit address (starting address -1) for Load Tome Core,

=(LWRLIM)+(DATLEN)

18. LWRLIM — contains the lower limit of unprotected core,

=($F7)

19. NAME — contains the ASCII code for the name used in the starting address field of the HEX block.

20. NGRLSW — contains a 0 if there is no loading algebraic sign for a numeric field in the HEX block. It contains a +1 if the leading algebraic sign is legal and is a "+". It contains a -1 if the leading algebraic sign is legal and is a "-".

21. PROBAS — contains the program relocation base for HEX block correction constants. (For additional information refer to item 28.9.6.4.2 on starting addresses.)

22. PRODIF — contains the word count of the command sequence on mass storage.

23. QSAV — is used for temporary storage of the operand in the Q register.

24. SCANSW — is used as a bank of switches set prior to entry to SCANSW. See item 28.9.6.4.1 for use of SCANSW.

25. SECTNO — is the name for the 1st of 2 locations which contains a 30 bit number which is the absolute value for a starting sector for an I/O operation. The 15 most significant bits are located in bits 0-14 of SECTNO. The 15 least significant bits are located in SECTNO+1.

26. SECTOR — contains the word count for 1 sector on mass storage, = 96.

27. SYMSTR — is used as storage for up to 6 characters of a field extracted from the input block by the SCAN routine.

28. TABLIM — contains the base address of the Loader Table.

29. WRDCNT — is used as the character reference counter in order to extract characters from the input area. Bits 1-15 of WRDCNT are set to the address of the character. Bit 0 = 0 if the character is in the left half or =1 if in the right half of the word.

44.9.6.2 Communication Region Constants Used by HEXPRO

CONSTANT	LOCATION
$7FFF	$42 = MASK1
$8000	$21 = MASK2

44.9.6.3 Entry to HEXPRO

The name HEXPRO is declared as an external in the LOADER routine and as an entry point in the HEXPRO routine. The HEXPRO routine is entered from LOADER with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	HEXPRO	

44.9.6.4 Processing a HEX Block

HEXPRO will extract correction constants from entries in the HEX block and store them at consecutive locations beginning with a starting address for the operation. The starting address is contained in the first entry of the input block. Subsequent entries each contain a correction constant together with its relocation byte. Entries within the HEX block are separated by commas. The last entry

in the block is followed by a carriage return (or a space if the input is from the card reader). A correction constant is missing if two commas occur with no entry data to separate them. The contents of the location which would have received the missing correction constant is not disturbed by HEXPRO. HEXPRO terminates operation when either the last entry in the block is processed or when an error has occurred while processing the HEX block.

A HEX block may have a maximum of 120 characters including the carriage return which is the last character of the block. (A space is accepted in place of a carriage return in case the input is from the card reader.)

Immediately upon entry to HEXPRO, a test is made for the proper terminating character in the 120th character position. If it is neither a space nor a carriage return, an illegal character of $00 will be inserted in its place. Consequently when HEXPRO encounters this illegal character during block processing, an error message will be printed and further processing of the HEX block is terminated.

39.9.6.4.1 Extracting HEX Block Entries

Entries within the HEX block may contain terms which are either names of up to 6 characters or hexadecimal numbers of up to 4 digits. The subroutine, SCAN, is used to extract a term of an entry from the input block. The SCAN sburoutine is entered by a return jump. Upon entry to the SCAN subroutine, the bits 0-2 of the location SCANSW may have the following setting:

1. Bit 0 = 1 if the term is assumed to be a hex number, or -
   = 0 if no such assumption is made by the calling program.
2. Bit 1 = 1 if the term may have no leading algebraic sign of "+", or -
   = 0 if the term may have no leading algebraic sign of "+".
3. Bit 2 = 1 if the term may have no leading algebraic sign of "-", or -
   = 0 if the term may have no leading algebraic sign of "-".

In addition, if the term is numeric, the SCAN subroutine will pick up the ASCII characters for the digits of the number as well as the binary value of the number whenever bit 3 of SCANSW is set to 1. Unless the A register is zero upon entry to SCAN, it is assumed that the leading character of the term about to be processed is in bits 0-7 of the A register. Upon return from SCAN:

1. If the term is a name, the locations SYMSTR to SYMSTR+2 contain the ASCII code dor up to 6 characters of the name. If the name contains fewer than 6 characters, the unused

character positions are spac filled. The terminal character is in bits 0-7 of INPREL. The terminal character may be either the 7th character of the name or the first non alphanumeric character (neither A thru Z nor 0 thru 9) to occur prior to the 7th character position. The location NGRLSW will be set to either -1, 0 or +1 depending on whether the name was preceded by a "-" sign, no sign or a "+" sign.

2. If the term is numeric, the location is always zero. The location INPWRD contains the binary value. If bit 3 of SCANSW was set to a "1" the locations SYMSTR+1, SYMSTR+2 and the left half of SYMSTR+3 contain the ASCII code for either:

   a. Up to 5 decimal digits, or

   b. a "$" followed by up to 4 hexadecimal digits.

39 .9.6.4.2 Starting Address

The starting address may consist of either an entry point name (up to 6 characters) with or without an increment, or an absolute hexadecimal address of up to 4 hex digits and no increment. If the starting address is absolute, it is extremely important that the hexadecimal number be preceeded by "$". It is otherwise assumed to be a decimal number. However, where an increment is used with an entry point name, the increment is assumed to be a hexadecimal number. Also, neither an absolute starting address nor an entry point name may be preceded by an algebraic sign of "+" or "-" although a hexadecimal increment may have such a leading algebraic sign.

39 .9.6.5.3 Entry Point Names and Hexadecimal Increments As a Starting Address

The A register is ste to 0. Then there is a return transfer to SCAN. Upon return from SCAN, a six character name is stored at the 3 locations SYMSTR, SYMSTR+1 and SYMSTR+2. The name must appear in some entry of the Loader Table, and the address associated with the name must be an entry point address. A return jump is made to TABSCH to search for the entry point name in the Loader Table. Upon return from TABSCH one of the following occurs:

1. If (SW6) is negative, the entry point name does not appear in the table. A return jump is made to the error subroutine PRINT3 to print an error indication of "E13". Another return jump is made to the subroutine PRINT5 to print the entry point name. A jump is made to NXTINP to process the next block of input.

2. If ((TABLIM)+(SW6)+3 is negative, there is an entry in the Loader with a name to match the entry point name. However, the address for this name is the one's complement of a 15 bit link address. The error procedure described in the previous paragraph takes place.

3. If ((TABLIM)+(SW6)+3) is positive, it is taken as the 15 bit address associated with the entry point name in the starting address. This address is placed in INPCTR. This address is also placed in PROBAS.

Upon return from SCAN, the character which follows the entry point name is contained in the location (INPREL). If this character is a comma, there is no hexadecimal increment included in the starting address. If this character is a plus or minus, there is either a positive or negative increment included in the starting address.

The A register is set to the ASCII code for the leading algebraic sign -

(INPREL) $\longrightarrow$ A.

A second return jump is made to SCAN to pick up the binary value of this increment. Upon return from SCAN, (SYMSTR) must be zero, (INPWRD) must contain the binary value of the increment (or its one's complement if negative) and (INPREL) must contain a comma. If any one of these conditions is not met, a return jump is made to the PRINT3 error subroutine where the error indication "E13" is printed. A return jump is made to PRINT5 to print the name. A jump is made to NXTINP to read the next input block. If no error had occurred, (INPWRD) is added to (INPCTR) and the result is placed in INPCTR. NOTE: In a HEX block, the hexadecimal increment for the starting address is the only number that may have a leading algebraic sign. Numeric starting addresses may not have a leading algebraic sign. Also, if a numeric starting address is not followed by a comma, the error procedures described above occurs.

39.9.6.4.4 Absolute Number as a Starting Address

Upon the return from the first return transfer to SCAN, an absolute starting address is assumed if (SYMSTR) is zero and INPWRD contains a binary value. This value is placed in INPCTR. The location INPREL must contain a comma. Any other character would cause an error operation similar to that described in item 28.9.6.4.2.

39.9.6.4.5 Preseting Data with HEX Correction Constants

Data storage may be preset during a loading operation using HEX corrections. The starting address consists of either the entry point name "DATBAS" $\pm$ HEX increment or this entry point name with no increment.

39.9.6.4.6 Limits of Core Which may be Preset with HEX Correction Constants

HEX correction constants may not be stored in locations in protected core. The two areas of execution time core which may be preset with HEX correction constants are $C5 - $E3 and (LWRLIM+1) to (CSQLIM).

When the absolute value of the starting address for the HEX block is determined, it is placed in INPCTR. A test is made to see if the address in INPCTR is not for a location in unprotected core. No error indication will be made if either of the following possibilities is the case:

1. $(LWRLIM) < (INPCTR) < (CSQLIM)$
2. $\$C5 \leq (INPCTR) \leq \$E3$

If either -

$(INPCTR) < \$C5$

or -

$\$E4 \leq (INPCTR) \leq (LWRLIM)$

then -

the error indication "E6" is printed to indicate the starting address is for a location in protected core. The error procedure described in item 28.9.6.4.2 then follows.

39.9.6.4.5 Locating Storage Area for HEX Correction Constants

The input area for storage of the HEX correction constants is determined at the location whose label is HEXPR5. This is accomplished as follows:

1. If the starting address is greater than "$C4" but less than "$E4", the input area is in the unprotected area of the communication region. Therefore -
   a. (INPCTR) = starting address for the HEX block,
   b. "$E4" $\longrightarrow$ CSQCTR, where (CSQCTR) = limit address for the input area, and
   c. +1 $\longrightarrow$ SW6.

   A jump is made to HEXPR6 to process the individual hex correction constants.

2. If the starting address field of the HEX block contains an entry point name = "DATBAS", the input area is in the Load Time Data Storage Buffer. Therefore -

   a. (INPCTR) - (DATDIF) ⟶ INPCTR, where (INPCTR) = starting address for the input area,

   b. (DATCTR) ⟶ CSQCTR, where (CSQCTR) = limit address for the input area, and

   c. 0 ⟶ SW6.

   A jump is made to HEXPR6 to process the individual HEX correction constants.

3. If the starting address field of the HEX block contains an entry point name, and if the entire command sequence storage remains in Load Time Core such that (PRODIF) = 0, then -

   a. (INPCTR) + (DATLEN) ⟶ INPCTR, where (INPCTR) = starting address for the input area,

   b. (CSQLIM) + (DATLEN) ⟶ CSQCTR, where (CSQCTR) = limit address for the input area, and

   c. 0 ⟶ SW6.

   A jump is made to HEXPR6 to process the individual HEX correction constants.

4. If the starting address field of the HEX block contains an entry point name, and if the entire command sequence is on mass storage such that (PRODIF) ≠ 0, then the input area is located in 1 or 2 sectors on mass storage. Therefore, 2 sectors will be brought into core as follows:

   a. (LOWCOR)+1 ⟶ INPCTR, where (INPCTR) = starting address of input area,

   b. (LOWCOR)+194 ⟶ CSQCTR, where (CSQCTR) = limit address for the 192 words brought in from mass storage,

   c. -1 ⟶ SW6,

   d. The absolute value of the number for the 1st of 2 sectors is computed using the DPRADD routine. The sector number is recorded in SECTNO and SECTNO+1, and

   e. A return jump is made to MDRIV to read 192 words inot core starting at (LOWCOR)+1.

   A jump is made to HEXPR6 to process individual correction constants.

39.9.6.4.6 Correction Constants - HEXPR6

A return transfer is made to SCAN for each correction constant.

1. A legal correction constant has been extracted.
2. An illegal correction constant has been detected.

3. The entry contains no correction constant.

39.9.6.4.7 Storing a Legal Correction Constant

If upon return from SCAN, an entry contains a legal correction constant (a constant of no more than 4 hexadecimal digits), (SYMSTR) is zero, (INPWRD) is the binary value of the correction constant and (INPREL) contains either a "+" or a "-" as the terminating character for the correction constant. The plus sign indicates positive address relocation for the correction constant while the minus sign indicates negative address relocation. The particular relocation base to be used for address relocation depends on the relocation flag which follows the "+" or "-" sign. The relocation flag is extracted from the input block by a return jump to CHPU. (See item 28.8.2.) Upon return from CHPU, the location INPREL contains in bits 0-7 the ASCII code for a relocation flag where the relocation flags are as follows:

A absolute or no relocation

P program relocation

C common storage relocation

D data relocation

Relocation is handled as follows:

TYPE OR RELOCATION	ACTION TAKEN
absolute	(INPWRD)+0 ⟶ INPWRD
negative absolute	(INPWRD)-0 ⟶ INPWRD
positive program rel.	(INPWRD)+(PROBAS) ⟶ INPWRD
positive common storage rel.	(INPWRD)+(COMBAS) ⟶ INPWRD
positive data storage rel.	(INPWRD)+(DATBAS) ⟶ INPWRD
negative program rel.	(INPWRD)-(PROBAS) ⟶ INPWRD
negative common storage rel.	(INPWRD)-(COMBAS) ⟶ INPWRD
negative data storage rel.	(INPWRD)-(DATBAS) ⟶ INPWRD

The subroutine ADJOVF is used to perform the address arithmetic. (See item 28.8.3.) If the relocation flag in INPREL is not one of the above characters, it is regarded by HEXPRO to be an error. Further processing of the HEX block is terminated, and a jump will be made to the location HEXERX. (See item 28.8.6.4.10.)

If the relocation flag for the correction constant is legal, HEXPRO will attempt to store the constant at the location whose address is in INPCTR. Before this constant is stored, a test must be made for overflow of the input area. Overflow of the input area has occurred if -

$$(\text{INPCTR}) \geq (\text{CSQCTR}).$$

If overflow of the input area has occurred, one of the following will happen:

1. (SW6) = 0 or -1, the error message "E5" is printed to indicate overflow of the input area. A jump is then made to the location HEXERX. (See item 28.9.6.4.10.)
2. If (SW6) = +1, the error indication "E6" is printed to indicate overflow of the unprotected area of the communication region and an attempt has been made to store a correction constant in a location in protected core. A jump is made to HEXERX. (See item 28.9.4.6.10.)

If there is no overflow of available core, the contents of INPWRD is stored at (INPCTR). The contents of INPCTR is increased by 1 to the bext address of storage for correction constants. A return transfer is made to CHPU. Upon return from CHPU, bits 0-7 of the location INPREL contain the character which follows the entry relocation flag for the correction constant. If this character is a comma, HEXPRO will look to process the next entry in the block. If this character is a carriage return (or a space), HEXPRO will transfer to NXTINP to process the next entry in the block. However, if the input area is from mass storage, the 192 words at the bottom of Load Time Core must be written back onto mass stoage before a jump is made to NXTINP. If this character is neither a comma, space or carriage return, a jump is made to the location HEXERX. (See item 28.9.6.4.10.)

Since the driver does not transfer a carriage return to the input buffer, the presence of a carriage return in the input block is indicated with a "$FF". (Prior to reading a record of relocatable binary input, each location in the input buffer is filled with the value $FFF.)

39.9.6.4.8 Illegal Correction Constants

Illegal correction constants are those which have more than 4 digits. Illegal relocation flags or a relocation flag not separated from the constant by a "+" or a "-" sign. If an illegal correction constant should occur, processing of the HEX block is terminated. A jump is made to the location HEXERX. (See item 28.9.6.4.10.)

Binary values for illegal correction constants are not stored in core. However, if a correction constant and its relocation flag are followed by an illegal field terminator, the binary value for the constant is stored in core. Processing of the HEX block from that point on, however, is terminated, and a jump is made to the location HEXERX.

39.9.6.4.6 Missing Correction Constants

A field is considered to have a missing correction constant only if upon return from SCAN the following is true:

1. (INPREL) = ASCII code for comma,
2. (SYMSTR) = ASCII code for blanks,
3. (INPWRD) = is set to zero.

When a field contains a missing correction constant, the contents of the location whose address is in INPCTR is not disturbed. The address in INPCTR is increased by 1, and subsequent correction constants in the input block are stored beginning with the next address.

If upon return from SCAN, (INPREL) contains a value = $FF in place of a comma, the field with the missing correction constant is considered to be the last field in the input block. An exit is made from HEXPRO, the procedure for which is described in the second-from-the-last paragraph in item 28.9.6.4.7.

39.9.6.4.10 HEXERX

When a jump is made to HEXERX, a test is made to determine whether or not the input area consists of 192 words of command sequence read in from mass storage.

If (SW6) 0, the command sequence for the input area did not come from mass storage. The error procedure described in item 28.9.6.4.2 is followed. If (SW6) _ 0, the command sequence of the 192 word input area is written back onto the 2 mass storage sectors from which it originated. (This is followed by the error procedure described in item 28.9.6.4.2.)

39.9.6.5 Exit from HEXPRO

Exit from HEXPRO is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	NXTINP,I	JUMP TO NXTINP TO READ
	EQU	NXTINP(123)	NEXT INPUT BLOCK

39.6.9.6 Subroutines used by and External to HEXPRO

1. MDRIV is used for mass storage I/O operations. MDRIV is entered with a return jump to its entry point name, MDRIV, which is declared as an external in the calling PROGRAM. (See item 28.7.3.)
2. DPRADD is used to compute the starting sector number for all mass storage I/O operations. DPRADD is entered with a return jump to its entry point in the constant table. (See item 28.8.11.)
3. SCAN is used to extract a field from the input block. A field may contain either the starting address, an increment of the starting address or a correction constant. SCAN is entered with a return jump to its entry point in the constant table. (See item 28.8.1.)

4. CHPU — is used to extract a single character from the input block. CHPU is entered with a return jump to its entry point in the constant table. (See item 28.8.2.)

5. PRINT3 — is used to print error messages. PRINT3 is entered with a return jump to its entry point in the constant table. (See item 28.8.7.1.)

6. PRINT5 — is used to print the name of a starting address following the printout of an error indication. PRINT5 is entered with a return jump to its entry point in the constant table. (See item 28.8.7.4.)

39.9.7 EOLPRO

EOLPRO is the routine which is used to process the EOL block.

39.9.7.1 Constant Table Storage Referenced by EOLPRO

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
BASE	23
DATLEN	177
DATSTR	180
DIFCON	186
DSECNO	203
ENTPNT	13
INPCTR	15
INPWRD	10
LINK	14
LOWCOR	179
LWRLIM	191
NGRLSW	9
PRODIF	171
SECTNO	173-174
SECTOR	198
SW6	28
TABCTR	16
TABLIM	7
TABSNO	204

The constants are used as follows:

1. BASE — contains the base address of the Loader.

2. DATLEN — contains the number of words set aside for a data reservation.

3. DATSTR — contains the storage base address for the Load Time Data Storage Buffer.

4. DIFCON — is used for temporary storage.

5. DSECNO — contains the relative value of the starting sector number for recording the Load Time Data Storage Buffer in the scratch area on mass storage. (See item 28.9.1.5.2.)

6. ENTPNT — is set to WORD 4 of a Loader Table when EOLPRO looks at the table entry.

7. INPCTR — is set to the address of WORD 1 of a Loader Table as EOLPRO looks at the table entry.

8. INPWRD — is used for temporary storage.

9. LINK — is set to WORD 5 of a Loader Table entry when EOLPRO looks at the table entry.

10. LOWCOR — contains the lower limit address (lowest address -1) of Load Time core -

= (LWRLIM) + (DATLEN):

11. LWRLIM — contains the lower limit of unprotected core = ($F7).

12. NGRLSW — is the switch which determines the address mode for which patching will occur during a LINK1 (patching) operation. NGRLSW is set positive if patching is for absolute addressing and negative for relative addressing. (See item 28.8.8 - LINK1.)

13. PRODIF — contains the word count of the command sequence recorded on mass storage.

14. SECTNO — is the 1st of 2 locations containing a 30 bit sector number. This is the absolute value for the number of the starting sector for an I/O operation on mass storage. The 15 most significant bits are stored in bits 0-14 of SECTNO, and the 15 least significant bits in bits 0-14 of SECTNO+1.

15. SECTOR — contains the word count for 1 sector on mass storage = 96 words.

16. SW6 — contains some positive value which is an integer multiple of 5 such that

(TABLIM) + (SW6) = 1st word address of some Loader Table entry.

17. TABCTR — contains the number of entries in the Loader Table.

18. TABLIM — contains the base address of the Loader Table.

19. TABSNO — contains the relative value of the starting sector number for recording the Loader Table in the scratch area on mass storage. (See item 28.9.1.5.4.)

39.9.7.2 Entry to EOLPRO

The name EOLPRO is declared as an entry point name in the EOLPRO routine and as an external in the LOADER routine. Entry to the EOLPRO routine is with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	EOLPRO	

39.9.7.3 Processing an EOL Block

If the command sequence storage read in by the Loader has not overflowed the capacity of Load Time core, a jump is made to EOLPRO. This will occur if (PRODIF) = 0. At the location EOLJOO, the A register is set to a -0 and the Q register to a +0. A jump is made to the address in the location LOADER where the address LOADER is external to the EOLPRO routine. (See item 28.9.1.)

If the command sequence storage read in by the Loader has overflowed the capacity of Load Time Core such that (PRODIF) ≠ 0, EOLPRO will perform a check of the Loader Table. The contents of PRODIF is recorded temporarily in DIFCON. The contents of DATLEN is recorded temporarily in DATNUM. EOLPRO checks for duplicate Loader Table entries such that one entry contains an entry point name and address and the other entries by this name contain external names and link addresses. The link addresses point to link address strings which are situated on mass storage. It is up to the EOLPRO routine to locate such a string, bring it into core, patch each location in the string with the entry point address and write it back onto mass storage. Also it is up to the EOLPRO routine to look for other duplicate entries in the Loader Table for this name. EOLPRO will erase the duplicate entry for the Link string it has just patched.

At the end of the EOLPRO operation, the only link address strings on mass storage remaining unpatched will be those for which no entry point names had been encountered during the current Loader Operation.

EOLPRO begins its Loader Table check by setting SW6 to 0. If the Loader Table is emtpy such that

(SW6) = (TABSTR*5 where (TABCTR) = 0,

a jump is made to EOLJOO where the exit procedure described above will take place. If (SW6) ≠ (TABCTR)*5, a jump is made to EOLPR1.

39.9.7.3.1 EOLPR1

The following occurs at EOLPR1:

1. The address equal to (SW6)+(TABLIM) is placed in INPCTR, where (INPCTR) is the 1st word address of a Loader Table entry. WORD 4 of this entry is placed in LINK. WROD 3 of this entry is placed in ENTPNT.

2. If there is no other entry in the Loader Table for this name (LINK) = -0 and a jump is made to EOLJ10. If (LINK) ≠ -0, the address in LINK points to WORD 3 of some other entry in the Loader Table containing this name. With such being the case:

   a. If (ENTPNT) < 0, the entry whose 1st word address is (TABLIM) + (SW6) is not an entry point name. A lump is made t to EOLJ10.

   b. If (ENTPNT) > 0, the entry whose 1st address is (TABLIM) + (SW6) is an entry point name. A jump is made to EOLPR2.

3. At EOLJ10 the index count in SW6 is increased by 5 -

   (SW6) + 5 ⟶ SW6.

   The end of the Loader Table will have been reached when

   (SW6) = (TABCTR)*5

   If this condition is met, a jump is made to EOLJ00 where the exit procedure described in 28.9.7.1 occurs.

   If this condition is not met, the EOLPRO routine will proceed to EOLPR1 to look at another of the Loader Table entries.

39 .9.7.3.2 EOLPR2

At EOLPR2, the next Loader Table entry for this name is located. The last word of the current entry for this name is "plugged" with a "-0", or

-0 ⟶ (INPCTR)+4.

The address of WORD 3 for the next Loader Table entry occurs in LINK. The next Loader Table entry for this name is an external name entry, and WORD 3 contains the one's complement of the link address. The link address is placed in temporary storage:

-((LINK)) ⟶ INPWRD

This entry point address from the current Loader Table entry replaces this link address in the next Loader Table entry for this name.

(ENTPNT) ⟶ (LINK)

WORD 5 of the next Loader Table entry for this name is recorded in LINKSW:

((LINK)-3) → NGRLSW

The link address from the next Loader Table entry for this name is placed in LINK:

(INPWRD) → LINK

A return jump is made to the TABIO (see item 28.9.7.5) routine with (A) = -0. This is to place the Load Time Data Storage Buffer (if there is one) and the Loader Table in temporary storage in the scratch area on mass storage (see item 29.9.7.5.).

Upon return from TABIO, a jump is made to EOLPR4.

39.9.7.3.3 EOLPR4

At EOLPR4, the EOLPRO routine will locate on mass storage the command sequence containing the link address string to be patched. It will bring this information into core, perform the patching operation and write the command sequence back onto mass storage in the sectors from which it came. As a result of the return jump to the TABIO routine, there is a maximum amount of unprotected core in which to place the command sequence as it is brought in from mass storage. The steps necessary to carry out the above operation are as follows:

1. (LINK) = the 1st address in the link address string to be patched. The quotient of the division

$$\frac{(LINK)-(LWRLIM)-1}{96}$$

represents the relative value for the sector number containing (LINK). The A register is set to this value as a result of the division.

2. The number of words to be read from mass storage is computed and recorded in NUMWDS -

(A)+1 → A

(A)*96 → NUMWDS

3. The upper and lower limits of Load Time Core for this operation are respectively (LWRLIM) and (BASE). The location PRODIF is set to the value = difference between execution time upper limit address minus load time upper limit address. The execution time upper limit address for reading from mass storage is -

(NUMWDS) + (LWRLIM)+1.

Therefore -

(NUMWDS)+(LWRLIM)+1-(BASE) →(PRODIF).

4. It is now necessary to compute the word length of unprotected core to the highest possible integer multiple of 96.

$$\frac{(BASE)-(LWRLIM)-1}{96} \rightarrow A, Q$$

(A) * 96 → A Reg.

5. Set the location NUMWDS to the smallest of the two values of (A reg.) and (NUMWDS).

If (A) < (NUMWDS),

(A) → NUMWDS.

6. The location ADDRS is set to the load time core starting address for reading from mass storage:

(BASE)-(NUMWDS) → ADDRS

7. The A register is set to the execution time value of the starting address for reading from mass storage.

(ADDRESS) + (PRODIF) → A

8. The relative value of the starting sector number for reading from mass storage is computed and recorded in SECNUM.

$$\frac{(A)-(LWRLIM)-1}{96} \rightarrow A, Q$$

(A) SECNUM

9. The DPRADD routine (See item 28.8.11) is used to compute the absolute value of the starting sector number for reading from mass storage. Upon return from DPRADD, the number is recorded in SECTNO and SECTNO+1.

10. The A register is set to the load time starting address and the Q register to the word length for reading from mass storage:

(ADDRS) → A

(NUMWDS) → Q

A return jump is made to MDRIV (see item 28.7.3) to read from mass storage.

11. The location DATLEN is set to 0. A return jump is made to the LINK1 routine to perform the patch operation on the command sequence brought in from mass storage.

12. Upon return from LINK1, the command sequence read in from mass storage will be rewritten onto the mass storage device. The DPRADD routine is used to compute the absolute value of the starting sector number. The A register is set to the load time starting address = (ADDRS). The Q register is set to the number of words to be written = (NUMWDS). The Q register is then complemented since the mass storage I/O operation is writing. A return jump is then made to MDRIV.

13. Upon return from MDRIV, PRODIF and DATLEN are reset to their original values:

    (DIFCON) → PRODIF

    (DATNUM) → DATLEN

14. The A register is set to a 0 and a return jump is made to TABIO (see item 28.9.7.5) in order to restore the Load Time Data Storage Buffer and the Loader Table to their former positions in unprotected core.

15. Upon return from TABIO, EOLPRO will look at LINKSW in order to determine if there are any other Loader Table entries by this name. If

    (LINKSW) = $FFF,

    there are no other Loader Table entries by this name. A jump is made to EOLJ10. (Refer to step 3 in item 28.9.7.3.1.) If (LINKSW) ≠ $FFFF, there is at least one other entry in the Loader table for this name. The location INPCTR is set to

    (LINKSW)-3

    and a jump is made to EOLJ11 in order to process that entry. The location EOLJ11 is 2 locations after EOLPR1. The Loader Table entry for this name which is stored at (LINKSW)-3 will be processed in the manner described in item 28.9.7.3.1.

39 .9.7.4 Exit from the EOLPRO Routine

The name LOADER is declared as an entry point name in the LOADER routine, and as an external name in the EOLPRO routine. The exit from EOLPRO is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	(LOADER)	

39 .9.7.5 Subroutines Used by and Internal to EOLPRO

The only subroutine used by and internal to EOLPRO is TABIO. TABIO is used to record the Load Time Data Storage Buffer and the Loader Table onto mass storage so that the locations in core they occupy may be used for other purposes. When the use of this core is no longer required, the same TABIO routine will restore the Load Time Data Storage Buffer and the Loader Table to core. Entry to the TABIO subroutine is with a return jump to the location whose label is TABIO.

In order to dump these segments of core on mass storage, TABIO must be entered with (A) = -0. In order to restore the information to core, TABIO must be entered with (A) = +0. Upon entry to TABIO, the A register is recorded in the location TBIOSW.

if (DSECNO) = 0, the TABIO routine will assume there is no Load Time Data Storage Buffer. If (DSECNO) is not zero, the DPRADD routine is used to compute the absolute value of the starting sector number for transferring the Load Time Data StorageBuffer to or from mass storage. The starting address = (DATSTR) is placed in the A register and the number of words = (DATLEN) is placed in the Q register. If (TBIOSW) is negative, the Q register is complemented for writing. A return jump to MDRIV will cause the I/O to be performed.

If (TABSNO) = 0, the TABIO routine will assume there is no Loader Table. If (TABSNO) is not zero, the DPRADD routine is used to compute the absolute value of the starting secotr number for transferring the Loader Table to or from mass storage. The starting address is equal to (TABLIM) is placed in the A register, and the number of words to be written or read is equal to (TABCTR)*5 is placed in the Q register. If (TABIOSW) is negative, the Q register is complemented for a write operation. A return jump is made to MDRIV in order to perform the I/O operation.

Exit from the TABIO routine is with a jump to the location TABIO where (TABIO) = address placed there at the time of entry.

39.9.7.6 Subroutines Used by and External to EOLPRO

1. LINK1 is used to patch each location in a string of link addresses with an entry point address. LINK1 is entered with a return jump to 164+(I). (See item 28.8.8.)
2. MDRIV is used for I/O on mass storage. MDRIV is entered with a return jump to the location whose name is MDRIV and this name is declared as an external by the EOLPRO routine. (See item 28.7.3.)
3. DPRADD is used to compute the absolute value of the starting sector number for a mass storage I/O operation DPRADD is entered with a return jump to the location 200+(I). (See item 28.8.11.)

39.9.8 ADRPR0

ADRPR0 is the routine used to look up an entry point. Loader tries to find the specified name in its symbol table.

39.9.8.1 Constant Table Storage referenced by ADRPR0

NAME	STORAGE POSITION
TABLIM	7
INPWRD	10
INPREL	11
INPCTR	15
BLANKS	18
SYMSTR	19
SCANSW	22
WRDCNT	24
SW6	28
INPUT	29
TABSCH	100
INPXC0	117
AINPUT	161
@INPUT	162
MS@UAN	171

The Constants are used as follows:

1. AINPUT — contains one of the entrance parameters for the Loader 0peration
2. BLANKS — contains the ASCII code for 2 spaces
3. INPCTR — used as on address counter to reference entries in the input block
4. INPREL — contains the terminating field when the field is extracted using the scan routine
5. INPUT — storage area for input blocks read by the Loader
6. INPWRD — contains tne binary valve for the numeric fields extracted from the input area by the SCAN routine
7. INPXC0 — contains the address constant INPUT where INPUT is the 1st location of the area for storage of relocatable binary input blocks read by the Loader

8.	QINPUT	Q register upon entry to the Loader
9.	SCAN	SCAN routine entrance
10.	SCANSW	used by SCAN to determine the type of ASCII field
11.	SW6	As a result of a Loader Table search, it contains a positive valve which is added to {TABLIM} in order to locate a Loader Table entry with a name to match on input block entry.
12.	SYMSTR	used as storage for up to 6 characters of a field extracted from the input block by the SCAN routine.
13.	TABLIM	base address of the Loader Table
14.	TABSCH	loader table search routine entrance
15.	WRDCNT	address of character reference counter
16.	MSQUAN	size of absolute record placed by the loader on mass storage.

39.9.8.2 Entry to ADRPRO

The name ADRPRO is declared external in the BRANCH routine and as an entry in the ADRPRO routine.

39.9.8.3 Starting Address

The starting address may consist of either an entry point name {up to 6 characters} with or without an increment, or an absolute hexadecimal address of up to 4 hex digits and no increment. If the starting address is absolute, it is extremely important that the hexadecimal number be preceded by ▽$▽. It is otherwise assumed to be a decimal number. However, where an increment is used with an entry point name, the increment is assumed to be a hexadecimal number. Also, neither an absolute starting address nor an entry point name may be preceded by an algebraic sign of ▽+▽ or ▽-▽ although a hexadecimal increment may have such a leading algebraic sign.

39.9.8.4 Entry Point Names and Increments as a Starting Address

The A register is set to 0. Then there is a return transfer to SCAN. Upon return from SCAN, a six character name is stored at the 3 locations SYMSTR, SYMSTR+1 and SYMSTR+2. The name must appear in some entry of the Loader Table, and the address associated with the name must be an entry point address. A return jump is made to TABSCH to search for the entry point name in the Loader Table. Upon return from TABSCH one of the following occurs:

1. If SW6 is negative, an error has occurred.

   A jump is made to the error routine to print out ▽E3▽ and wait for a reply.

2. If no error has occurred {SW6} = an index to the loader loader table entry.

Upon return from SCAN, the character which follows the entry point name is contained in the location {INPREL}. If this character is a comma, there is no hexadecimal increment included in the starting address. If this character is a plus or minus, there is either a possitive or negative increment included in the starting address.

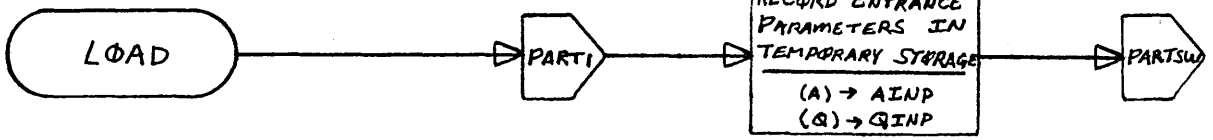
The A register is set to the ASCII code for the leading algebraic sign.

{INPREL} A

A second return jump is made to SCAN to pick up the binary value of this increment.

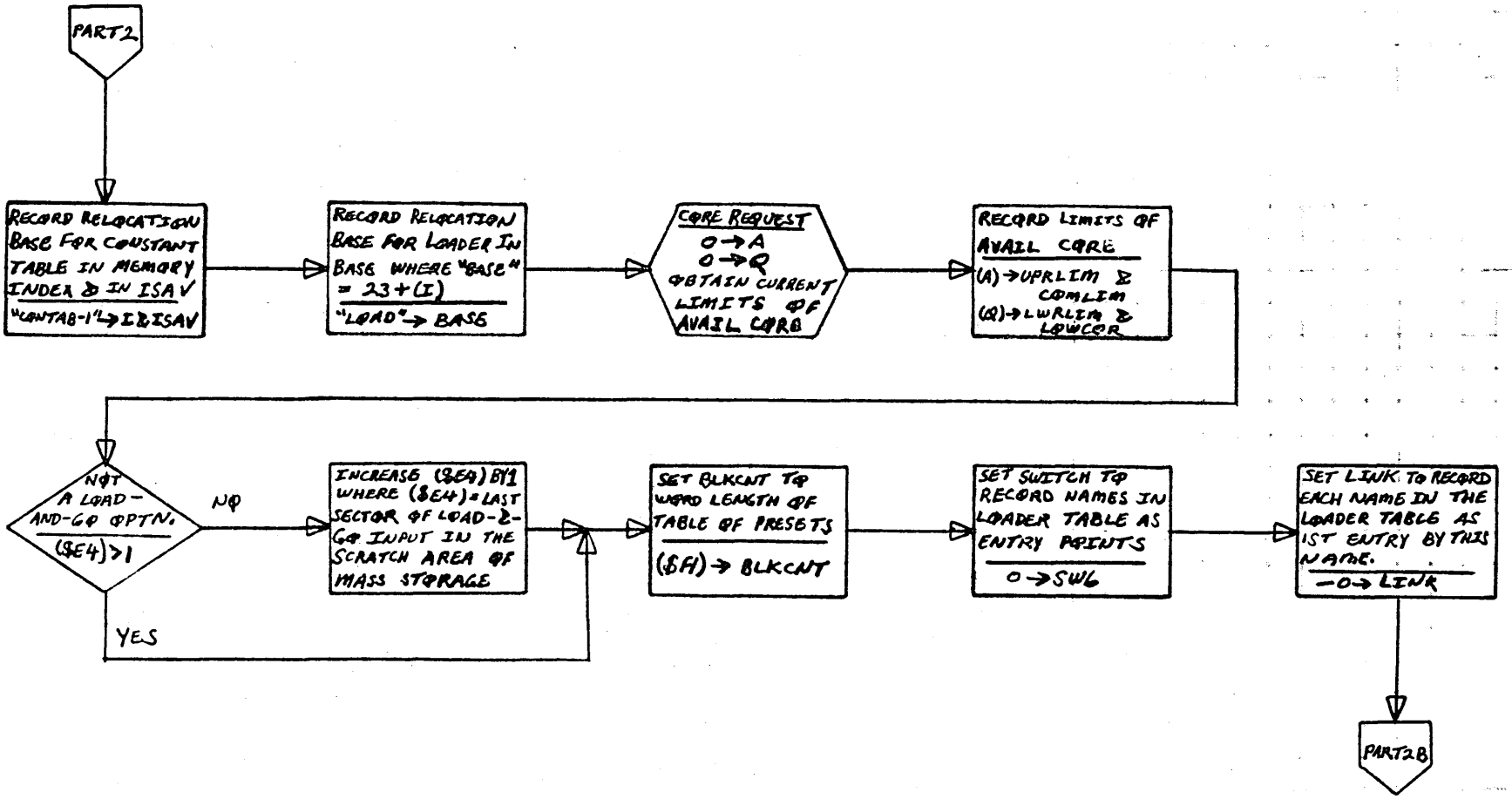
Upon return from SCAN, {SYMSTR} must be zero, {INPWRD} must contain the binary value of the increment {or its complement if negative} and {INPREL} must contain a space or carriage return.

If no error has occurred {INPWRD} is added to AINPUT} and the result is placed in {AINPUT}.



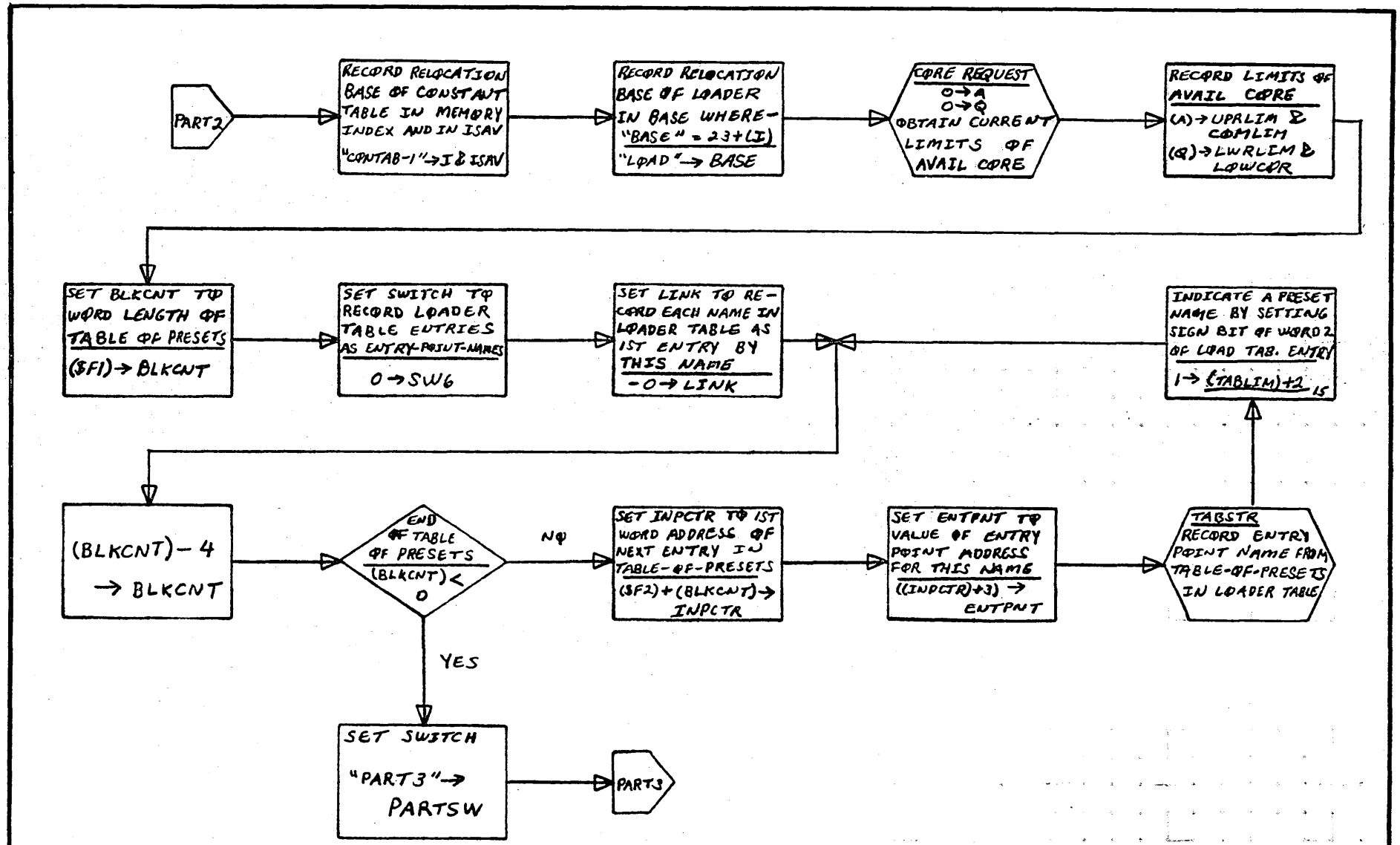
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
FLOWCHART ☐	LOAD Module PAGE 1 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T.G. DATE 2-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 Operating System LOAD Module	PAGE 2 OF 4	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY T. Gordon	DATE 12-6-66	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

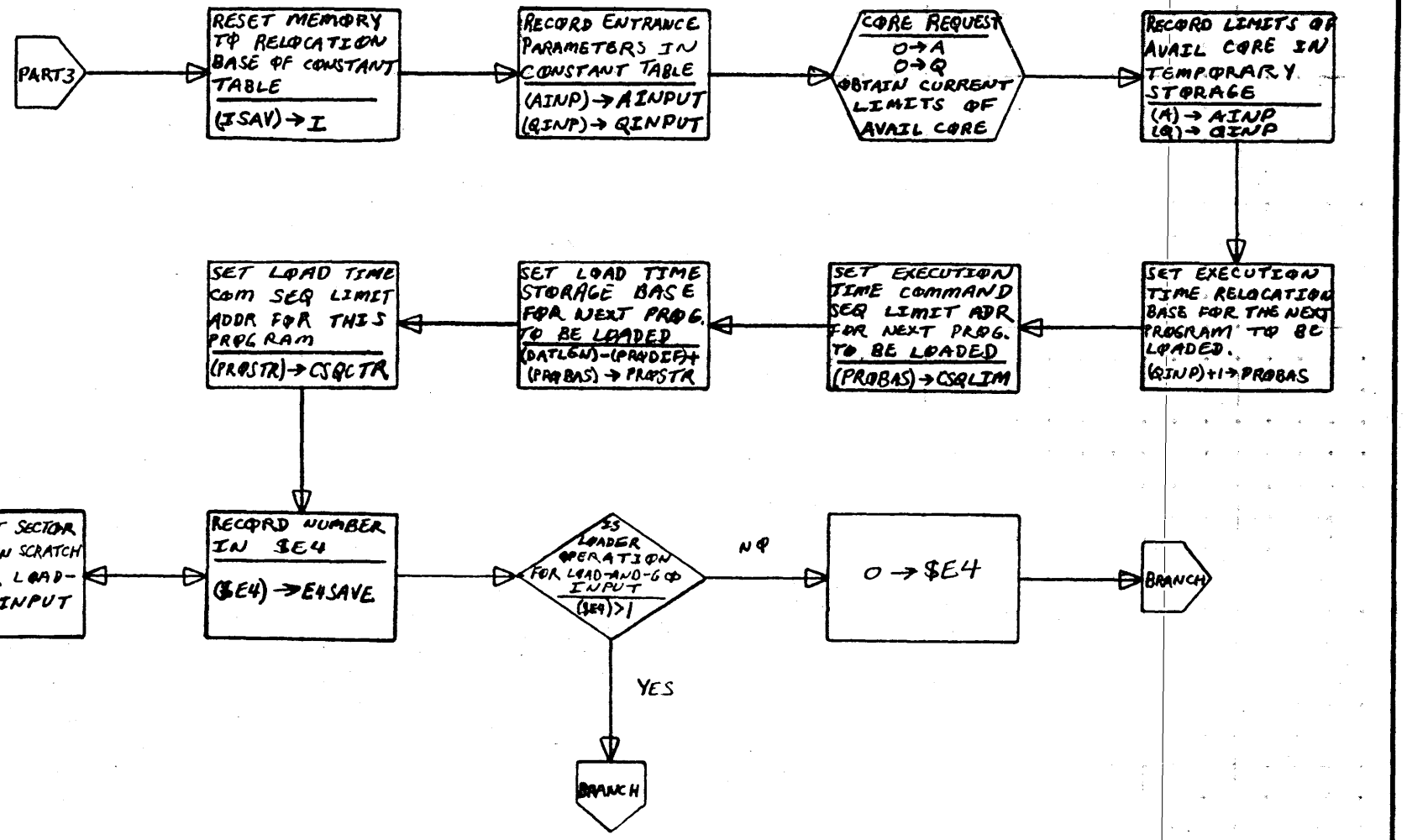


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	LOAD Module	PAGE 3 OF 4	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

39 162

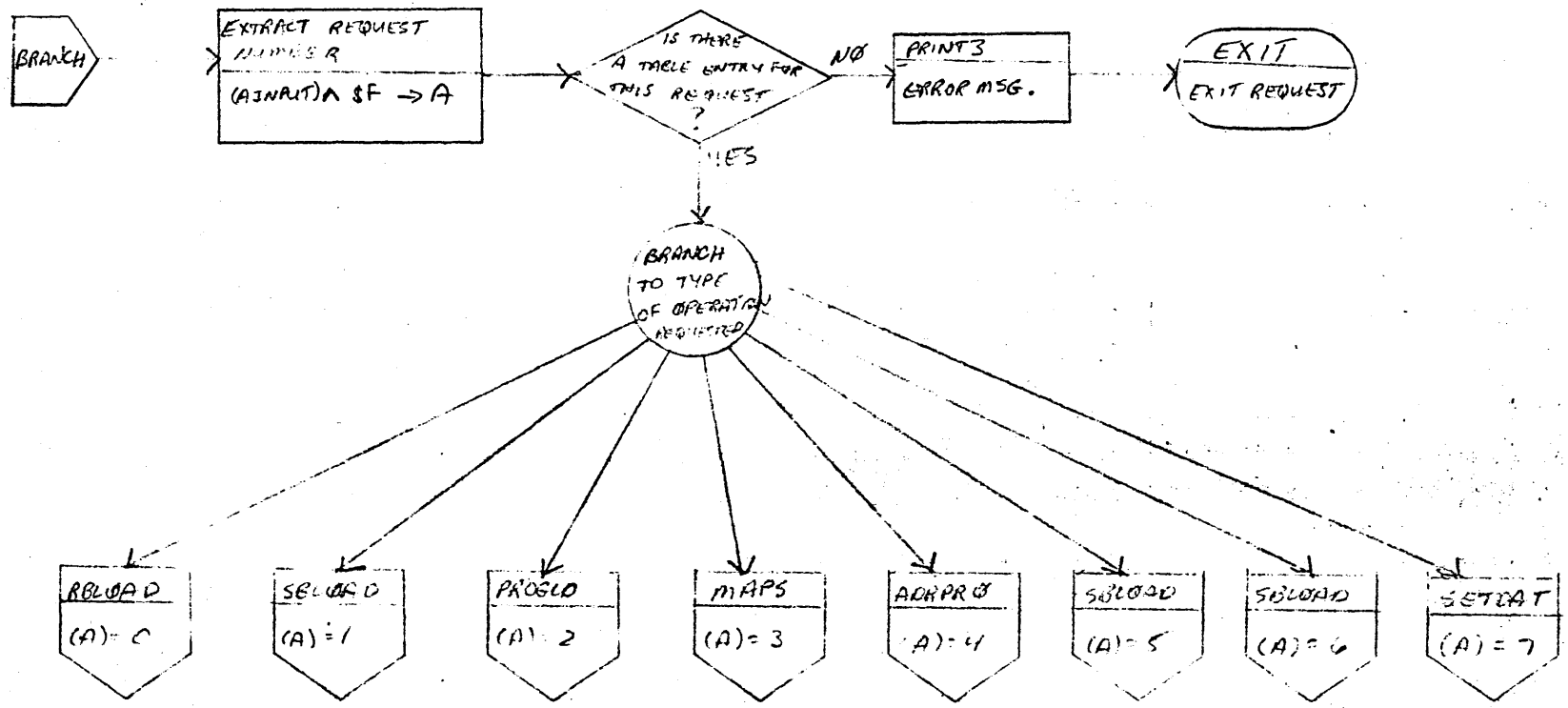


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
FLOWCHART ☐	1004 Module PAGE 4 OF 4	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. Cockson DATE 12-6-6	TASK NAME			

MAR 5 1971

39.163

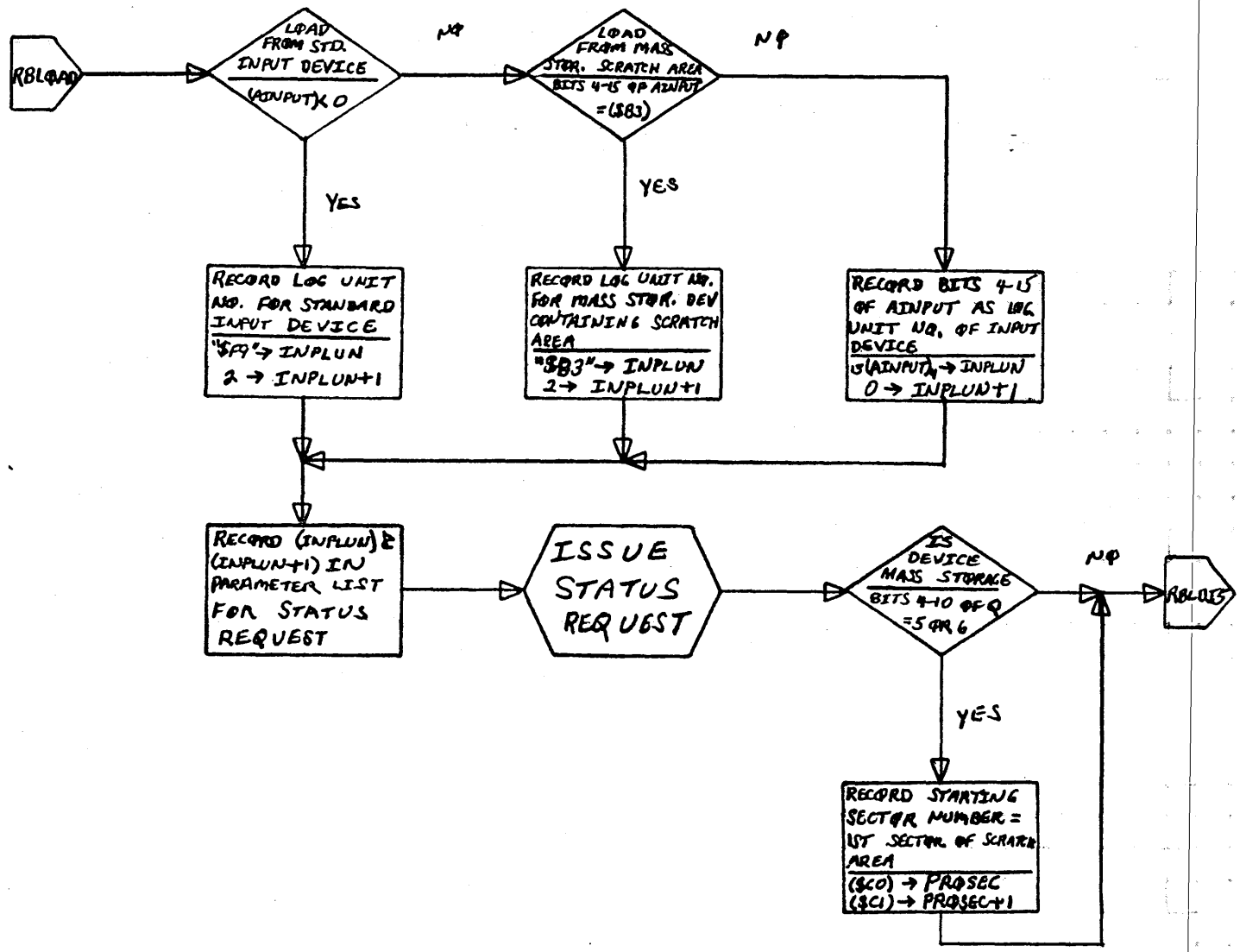
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	BRANCH MODULE PAGE 1 OF 21	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

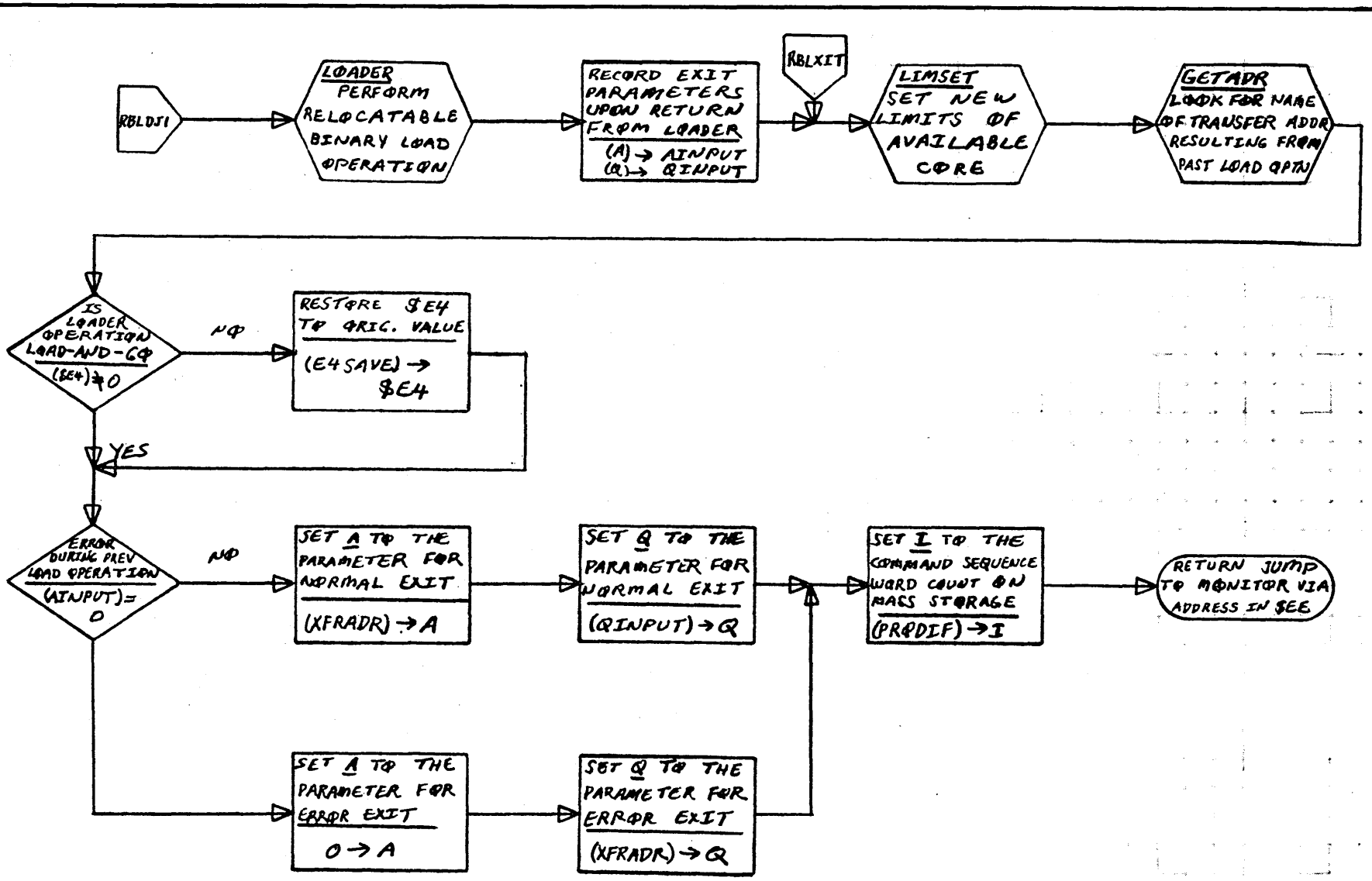
CA 127-1

MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS I.M.S. MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	BRANCH INTAKE PAGE 12 OF 21	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. Hanson DATE 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

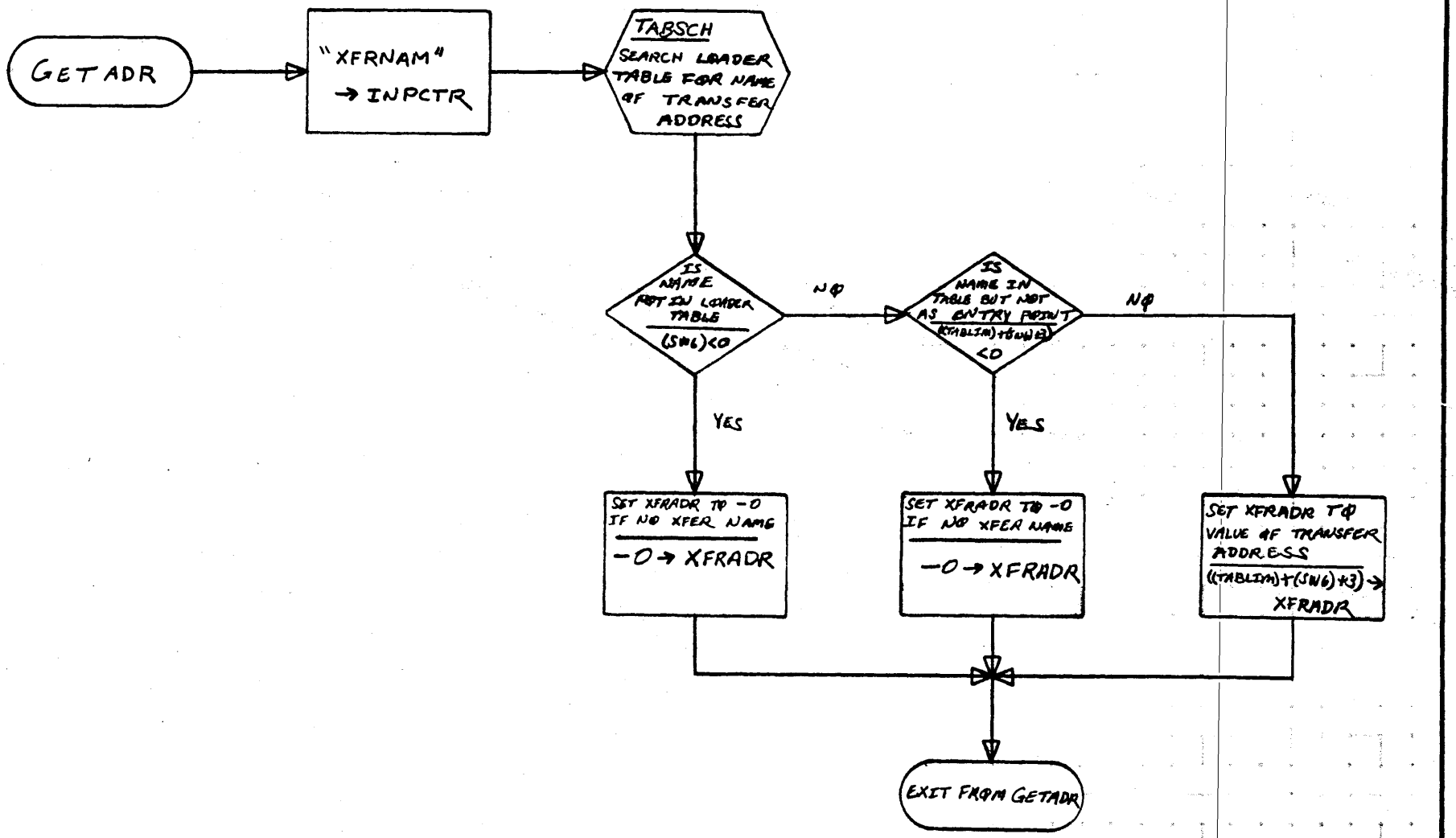


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	BRANCH Module	PAGE 3 OF 21	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. [illegible]	DATE 12-6-66	TASK NAME		

MAR 5 1971

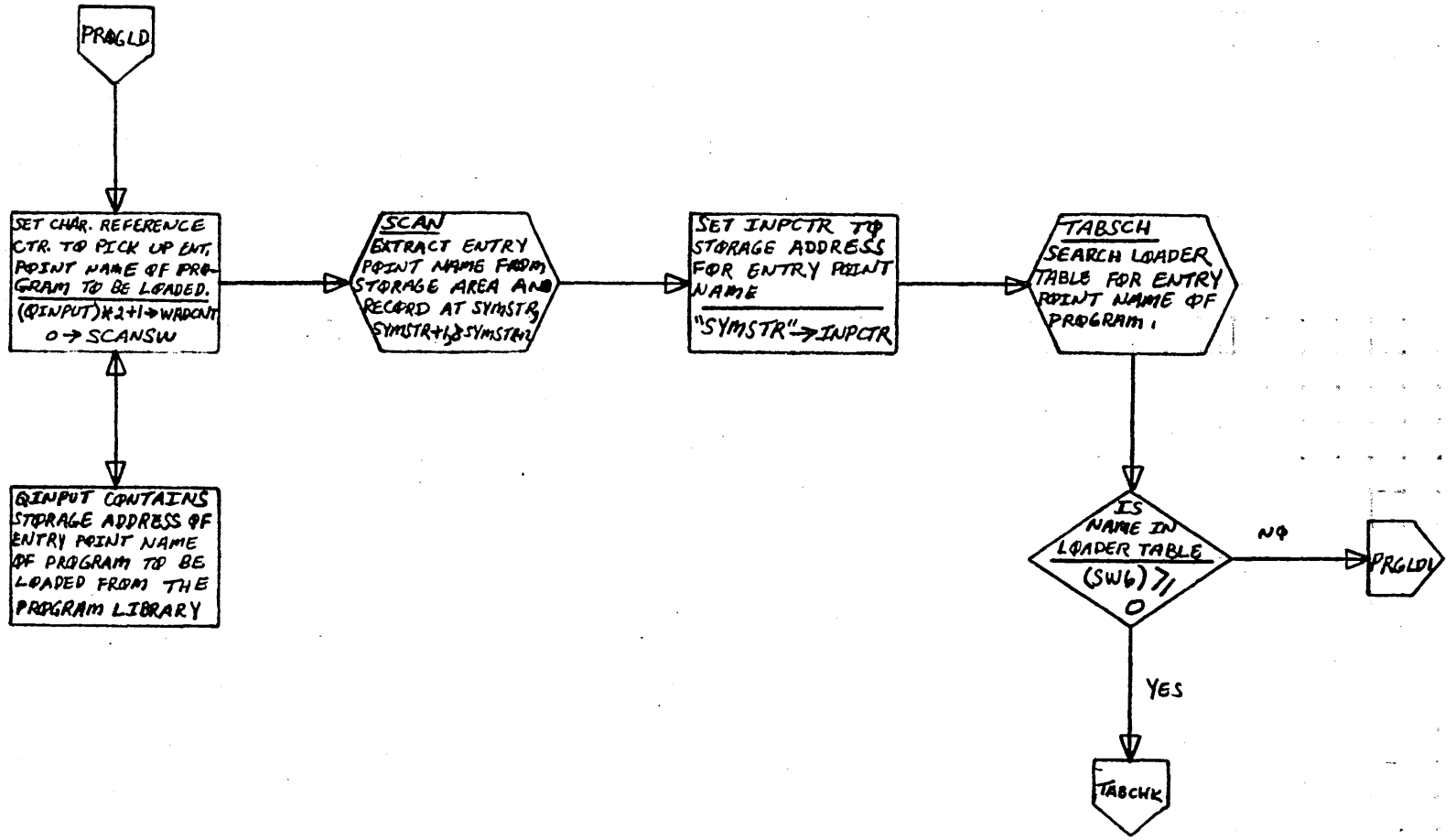
39.166

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 Computing System		PROJECT MGR.		
	Reference Model	PAGE 1 OF 21	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. Jordan	DATE 12-6-66	TASK NAME		

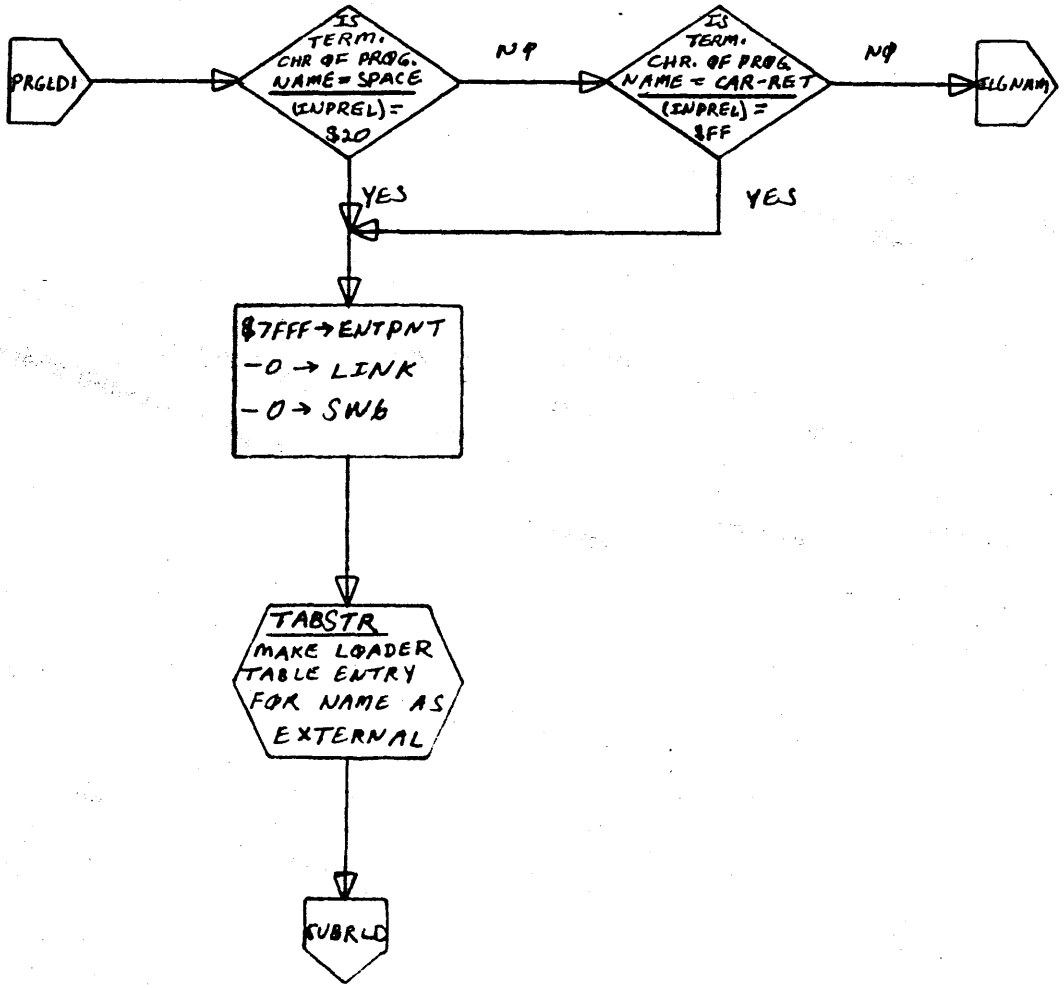
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	BRANCH 1604 PAGE 5 OF 21	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

MAR 5 1971

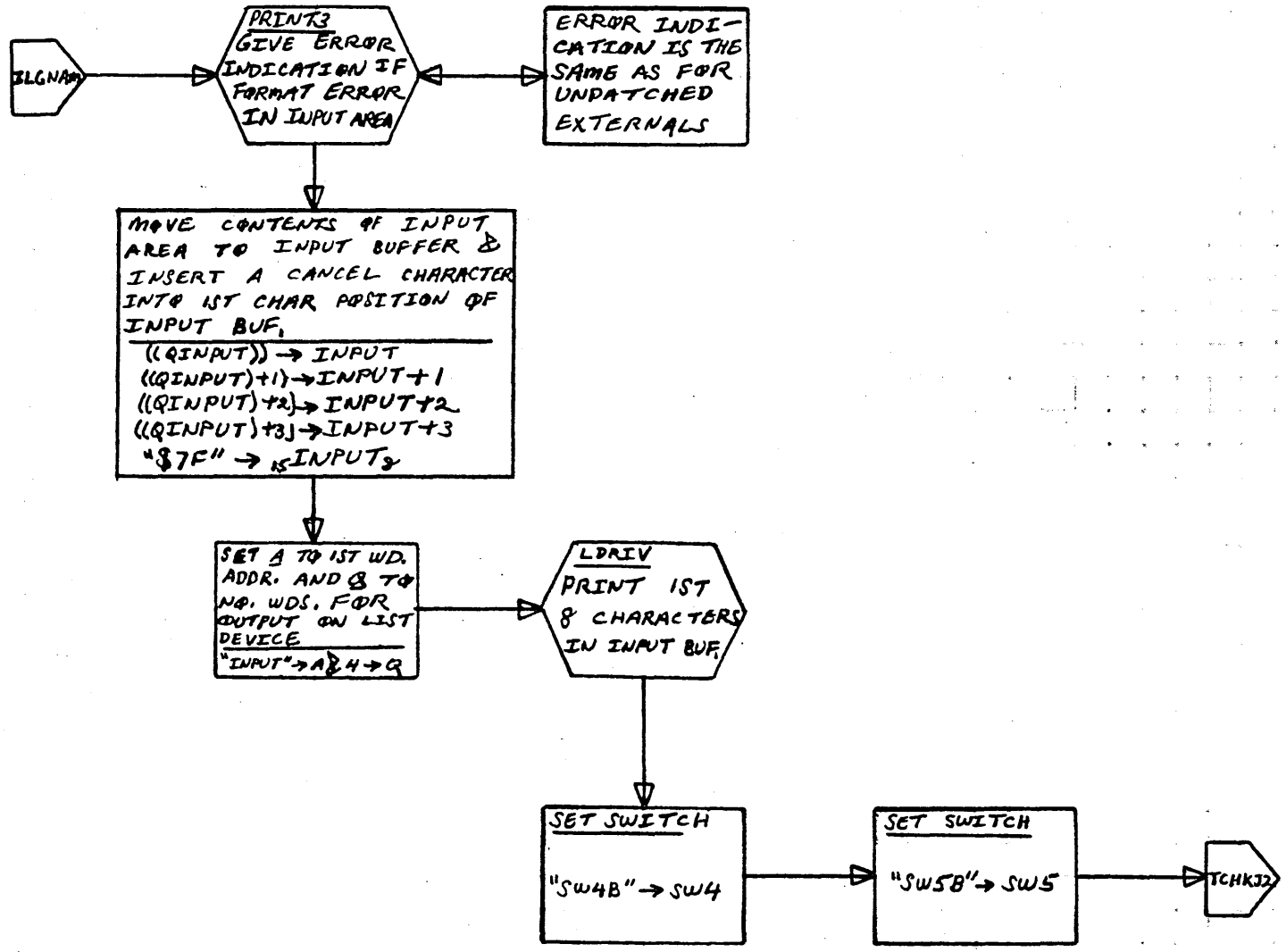
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE IMS Coupling System		PROJECT MGR.		
	BRANCH [illegible]	PAGE 6 OF 21	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY [illegible]	DATE 12-6-66	TASK NAME		

MAR 5 1971

AA1385 (FORMERLY CA127-1)

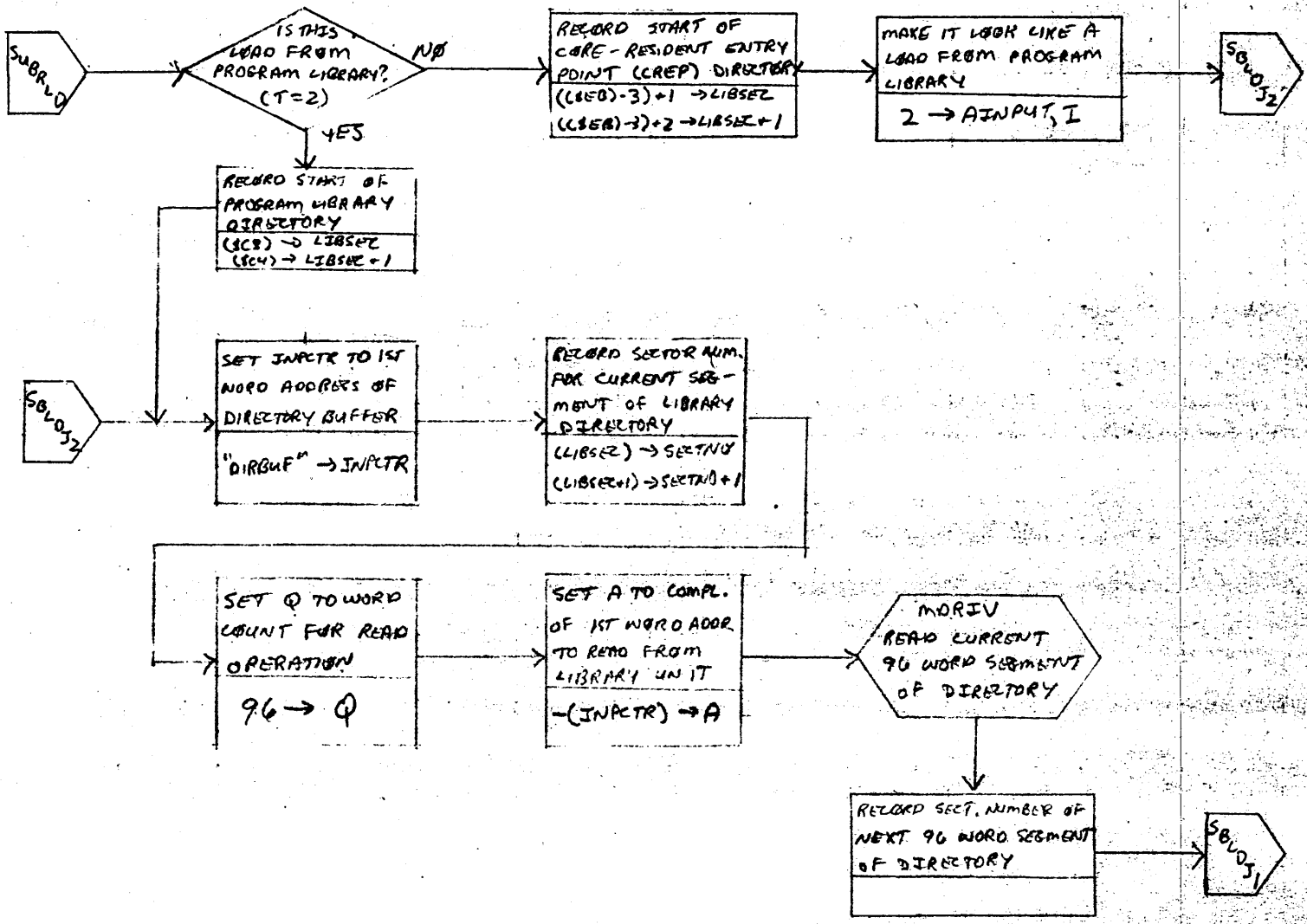


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
	Branch Module	PAGE 7 OF 21	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.170

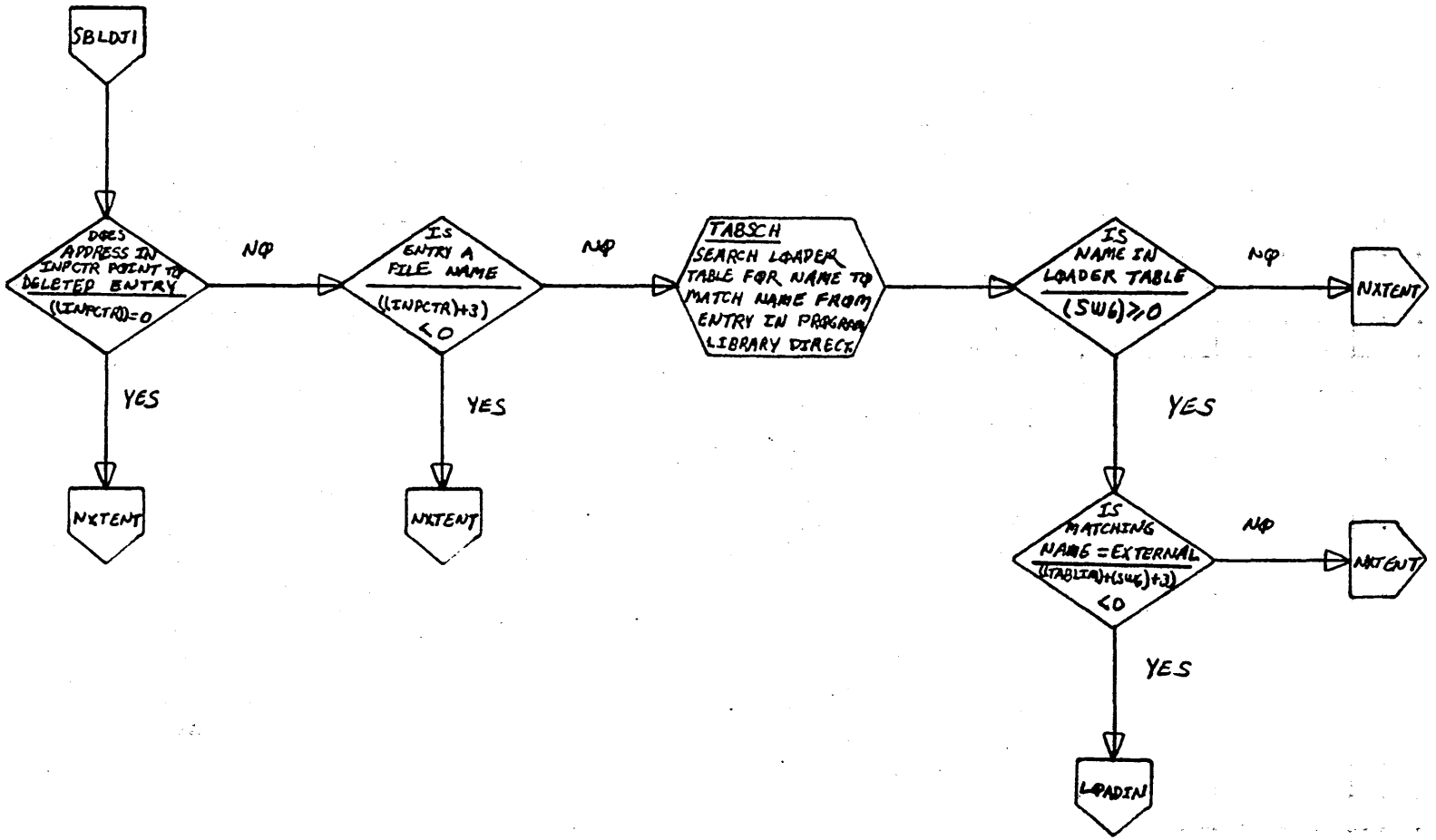
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1706	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700MSOS	PROJECT MGR.			
	BRANCH MODULE PAGE 8 OF 21	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

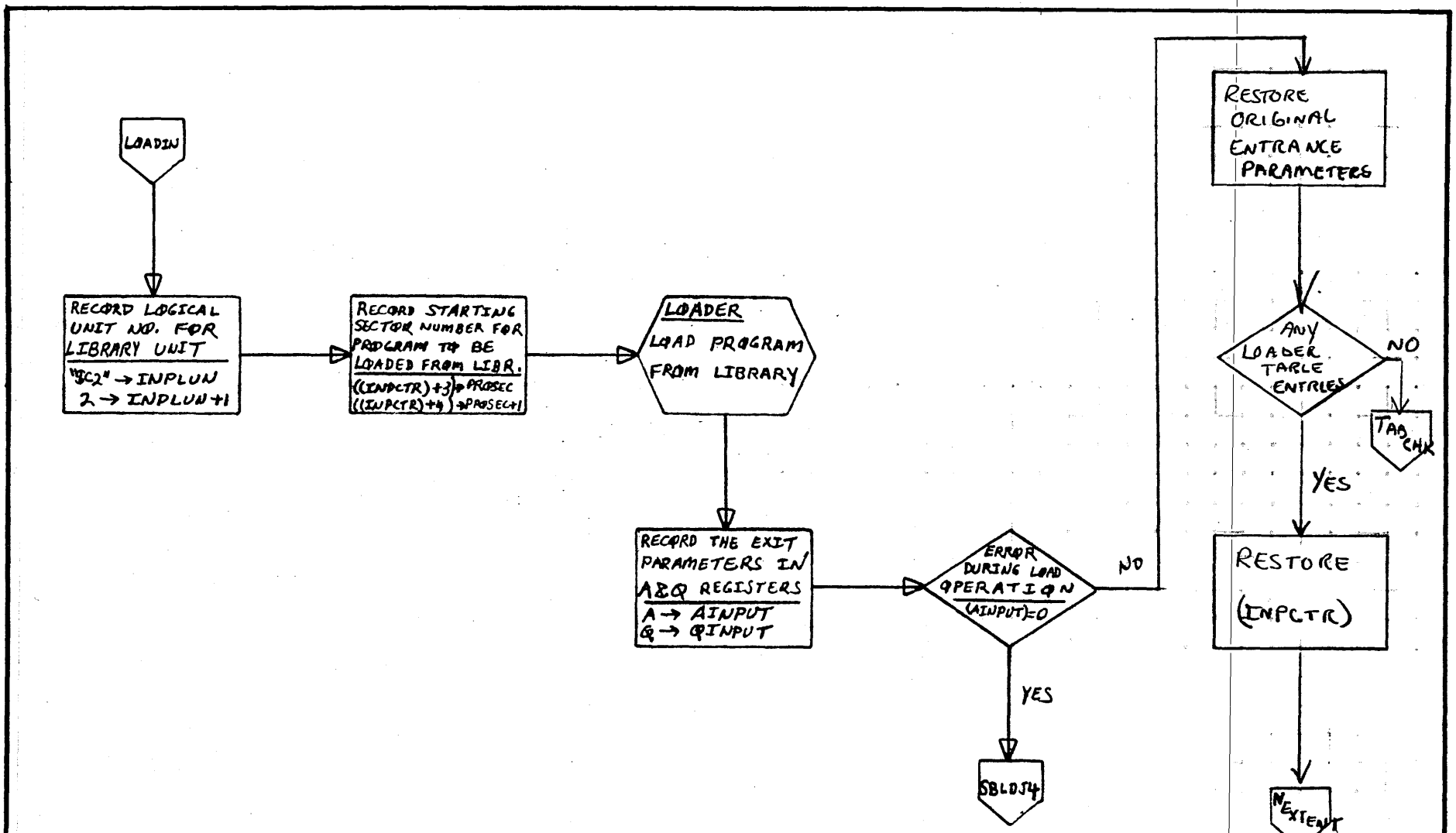
MAR 5 1971

CA 127-1



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IDIS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
FLOWCHART ☐	BRANCH MODULE PAGE 9 OF 21	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

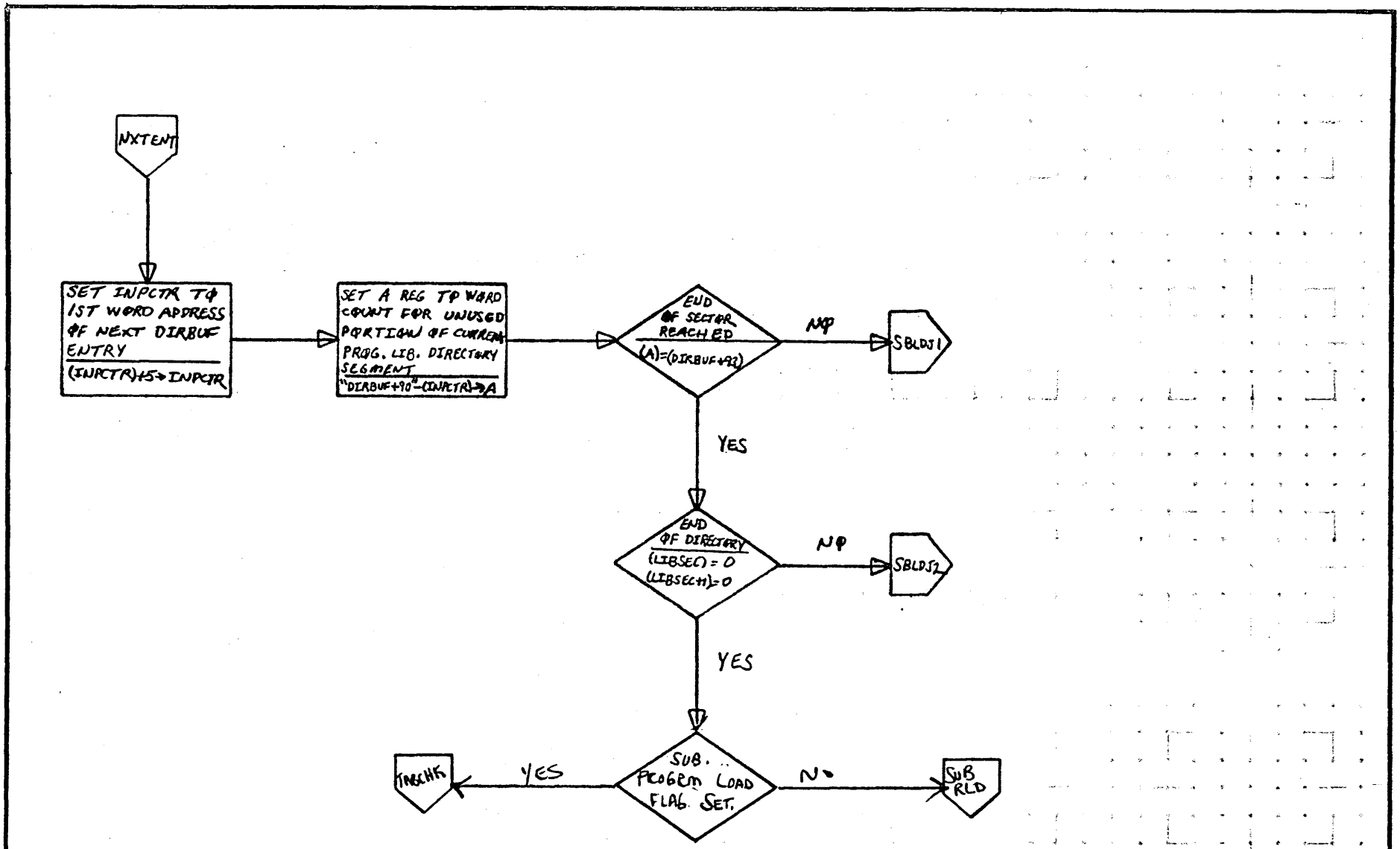
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE 1700 Operating System		PROJECT MGR.			
Branch Module	PAGE 18 OF 21	PROJECT NAME			
NUMBER	ISSUE DATE	TASK NO.			
DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME			

MAR 5 1971

3E 1.73

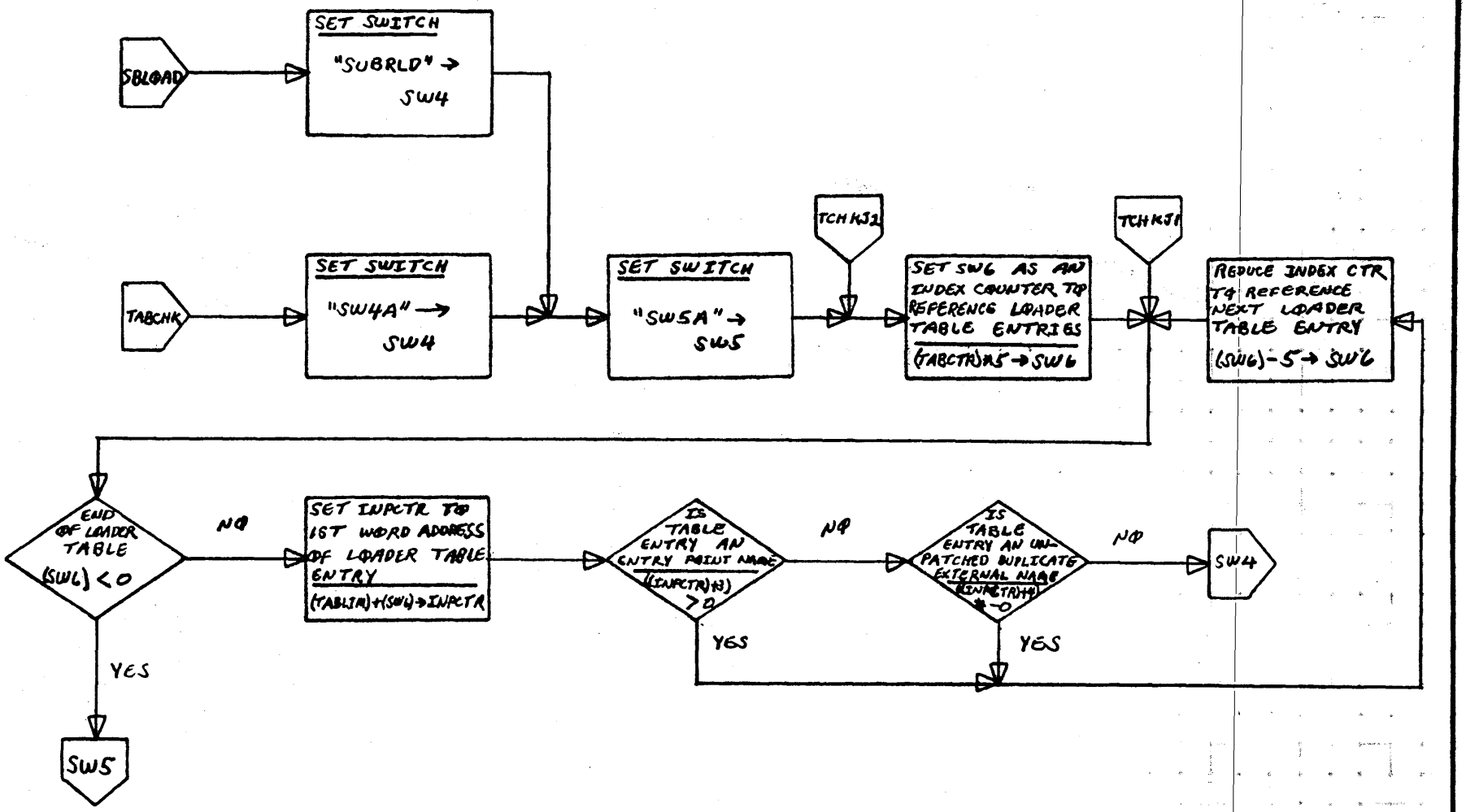


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	Branch Module PAGE 11 OF 21	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T Gordon DATE 12-6-66	TASK NAME			

MAR 5 1971

39.174

AA1385 (FORMERLY CA127-1)

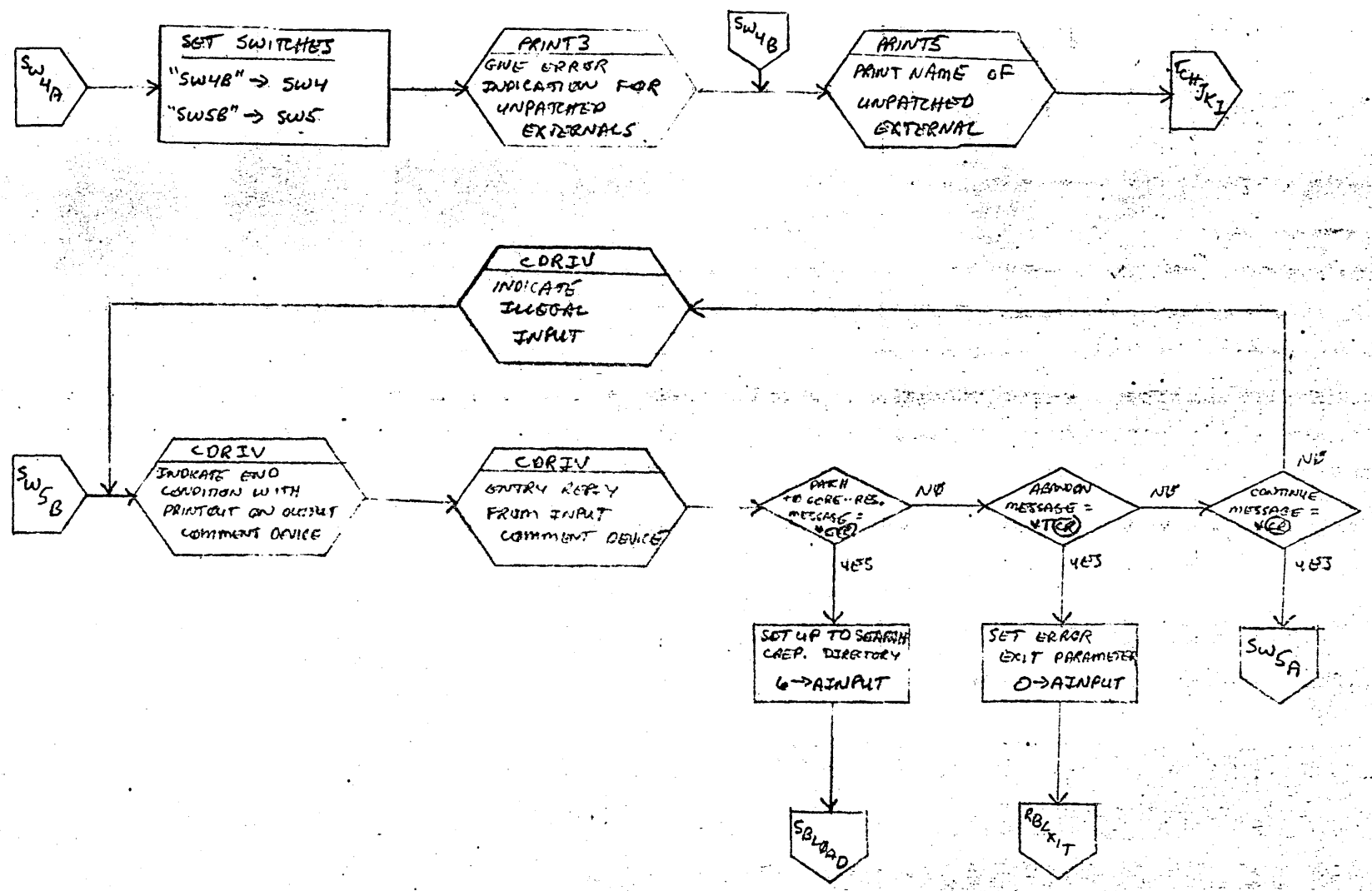


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS I45 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	BRANCH Module PAGE 12 OF 21	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

AA1385 (FORME CA127-1)

MAR 5 1971

39.175

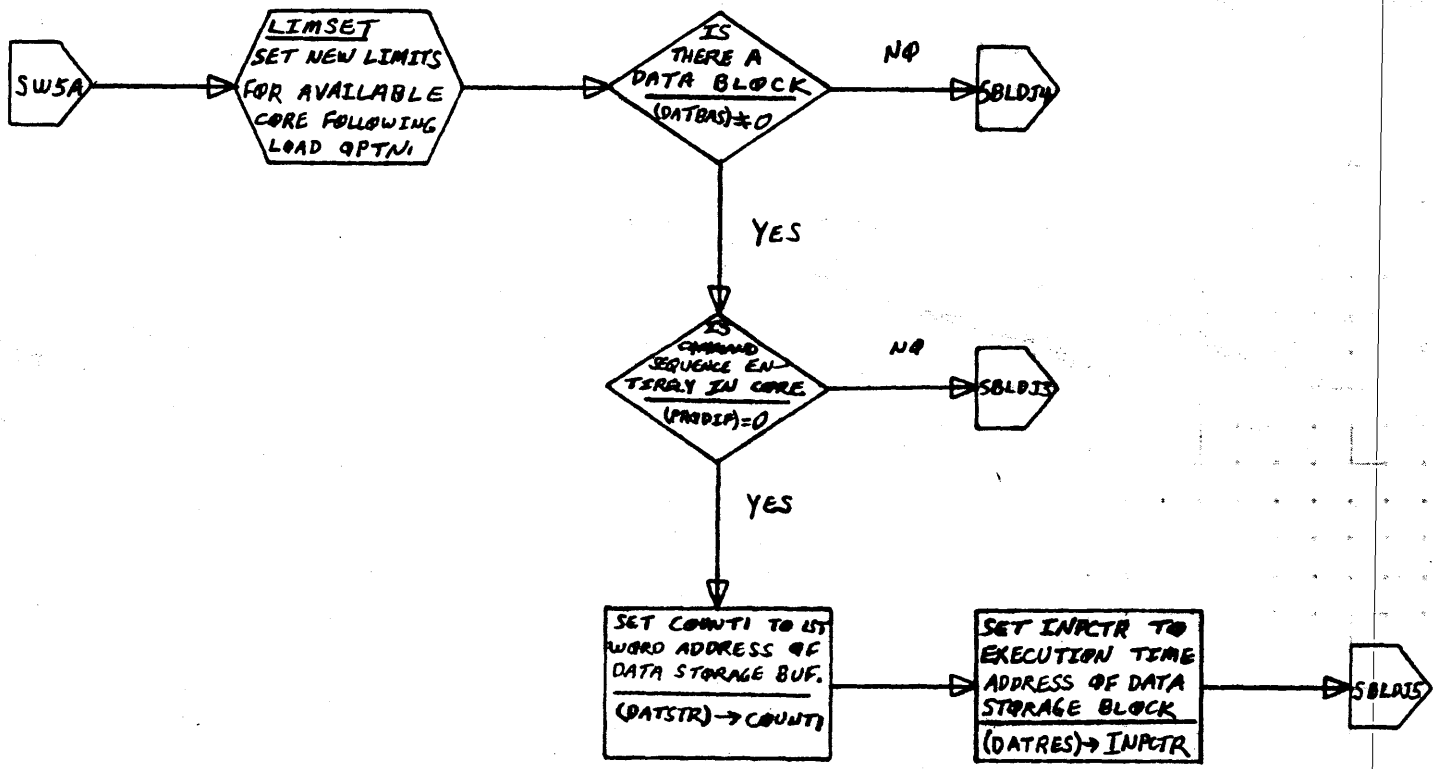


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 MSOS	PROJECT MGR.			
FLOWCHART ☐	BRANCH PAGE 13 OF 21	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

39.176

CA 127-1

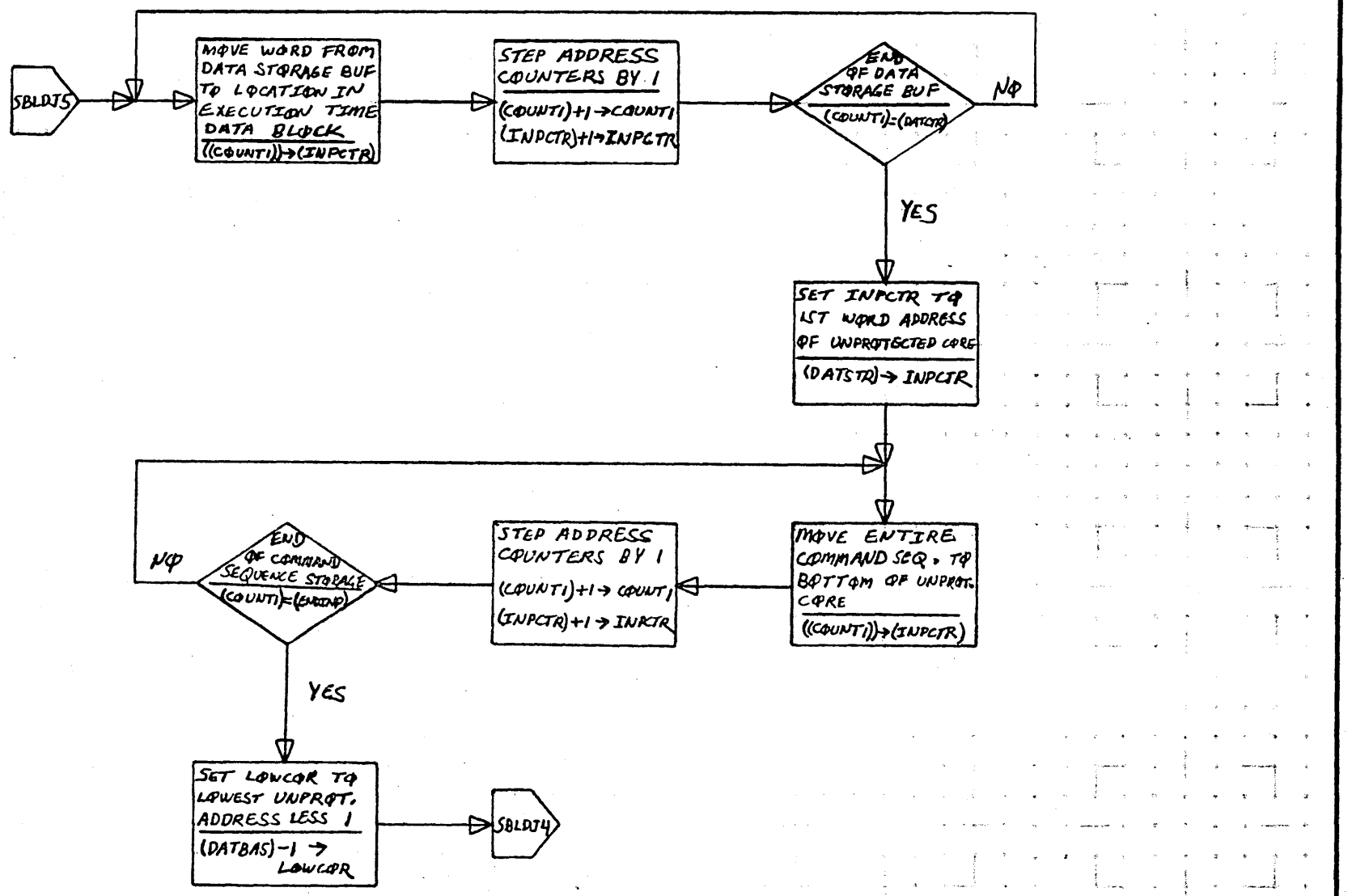


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.			
FLOWCHART ☐	BRANCH	PAGE 14 OF 21	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME			

AA1385 (FORM CA127-1)

MAR 5 1971

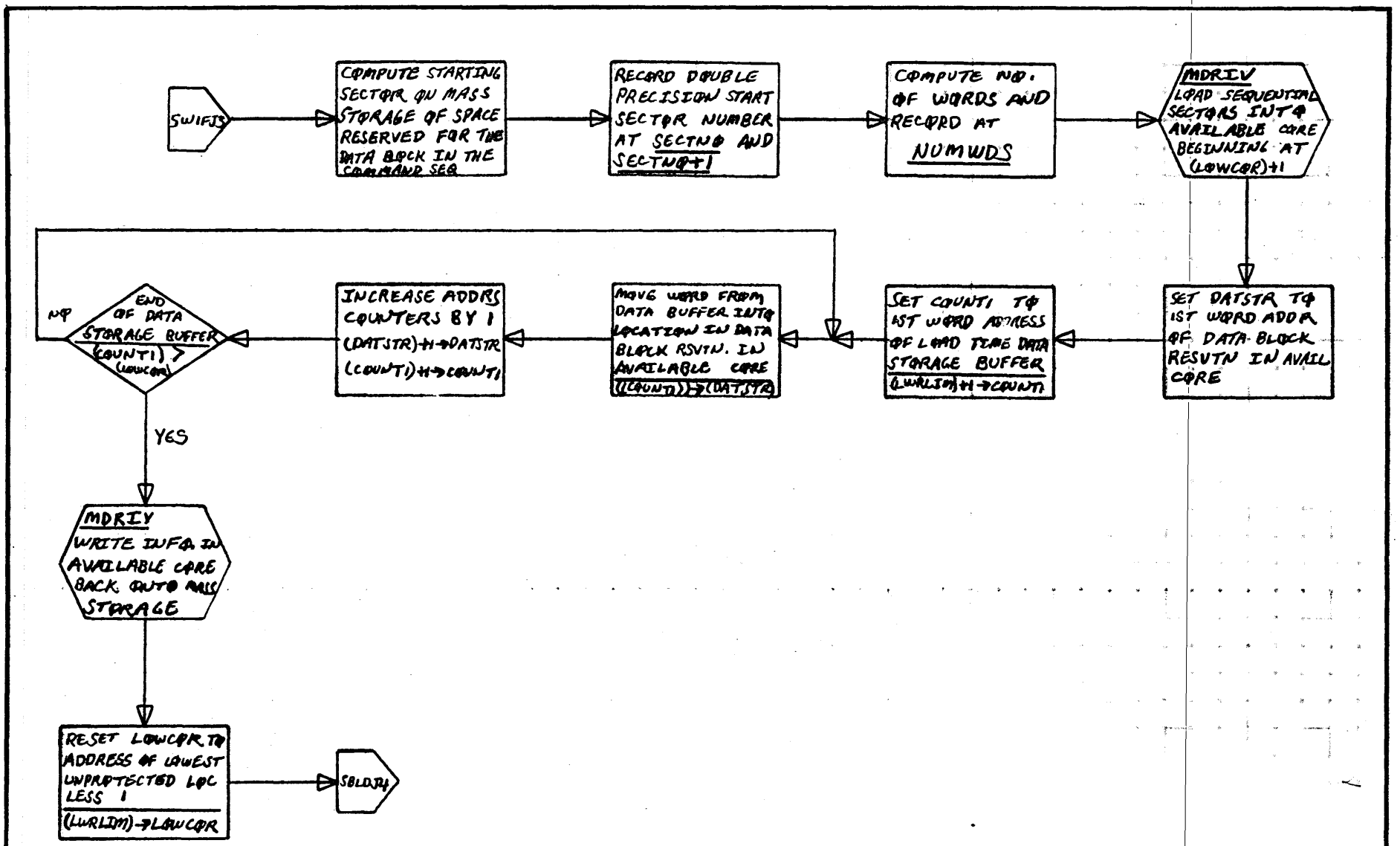
39,177



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.			
FLOWCHART ☐	BRANCH	PAGE 15 OF 21	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME			

MAR 5 1971

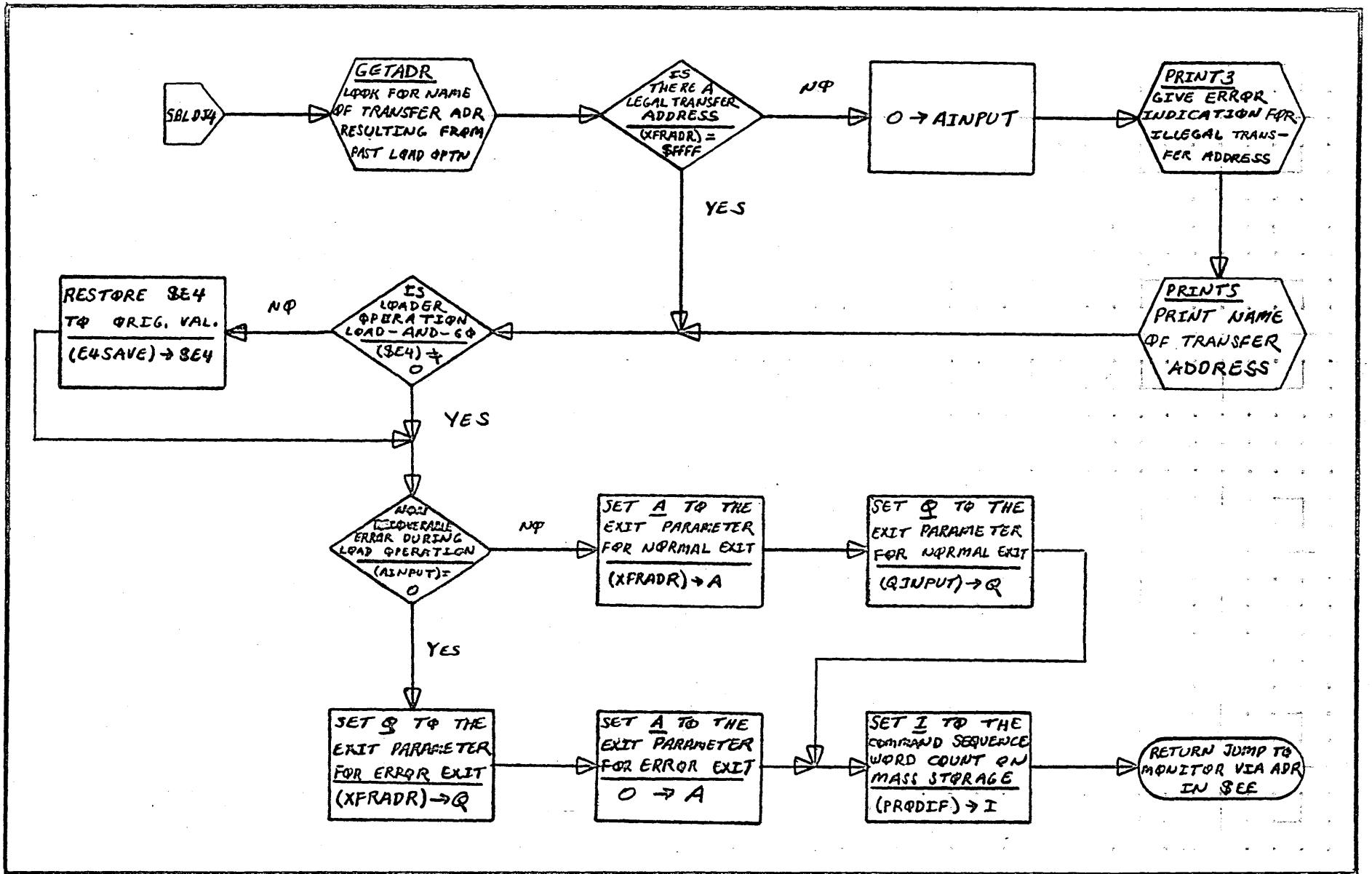
39.178



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.		
FLOWCHART ☐	BRANCH	PAGE 16 OF 21	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

MAR 5 1971

AA1385 (FORME CA127-1)

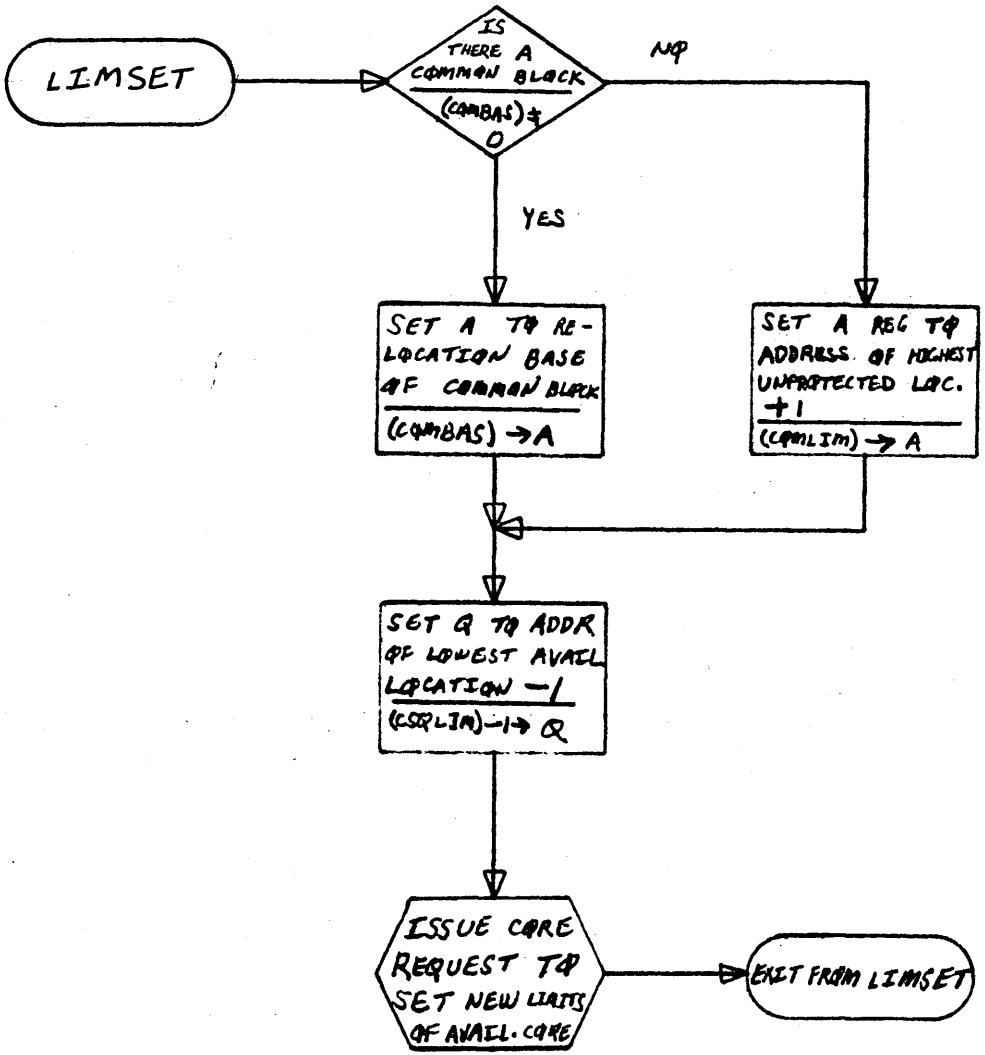


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1100	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1100 OPERATING SYSTEM		PROJECT MGR.		
FLOWCHART ☐	BRANCH	PAGE 19 OF 21	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.180

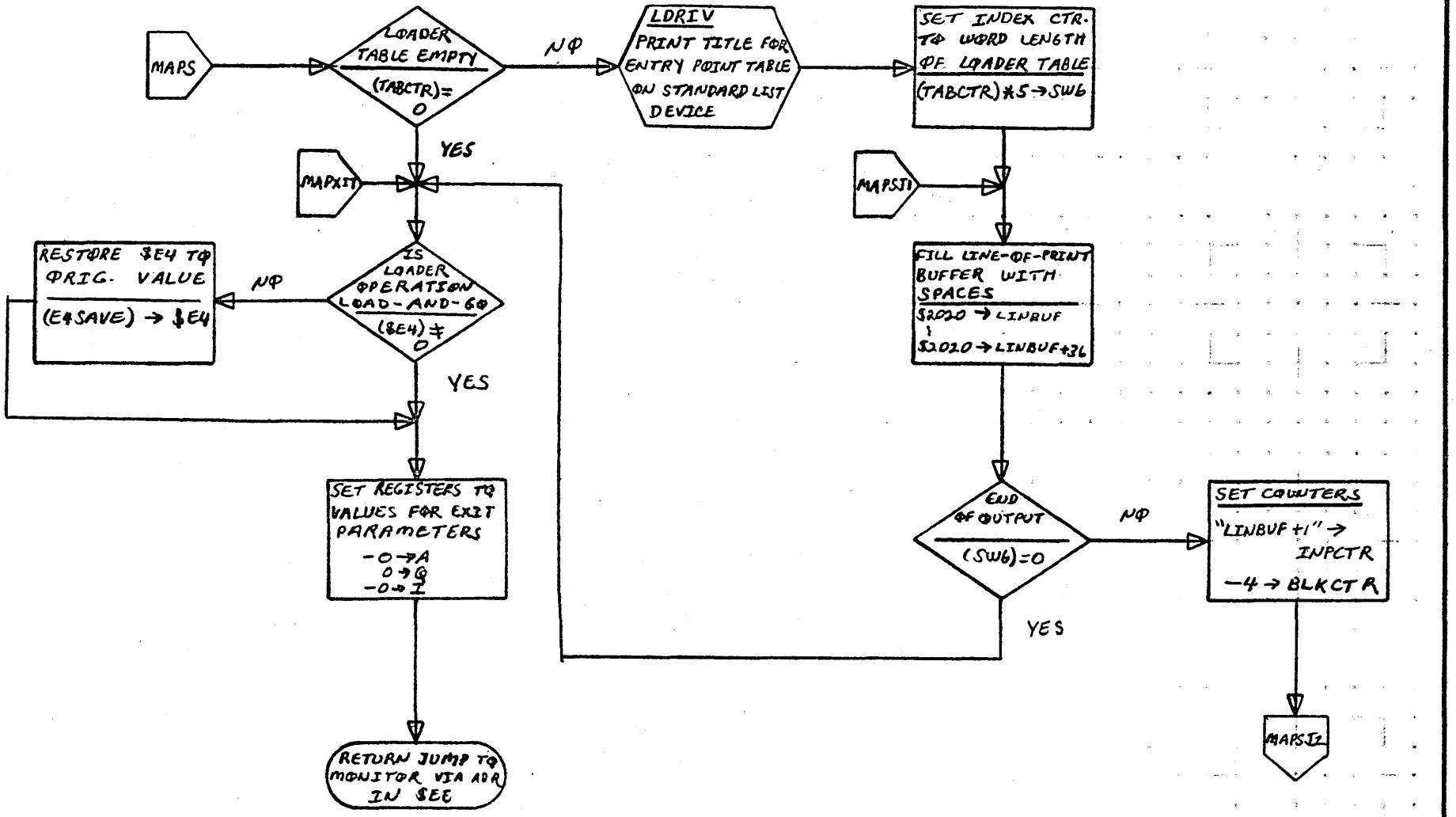
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1100	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1100 OPERATING SYSTEM		PROJECT MGR.			
FLOWCHART ☐	BRANCH	PAGE 18 OF 21	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME			

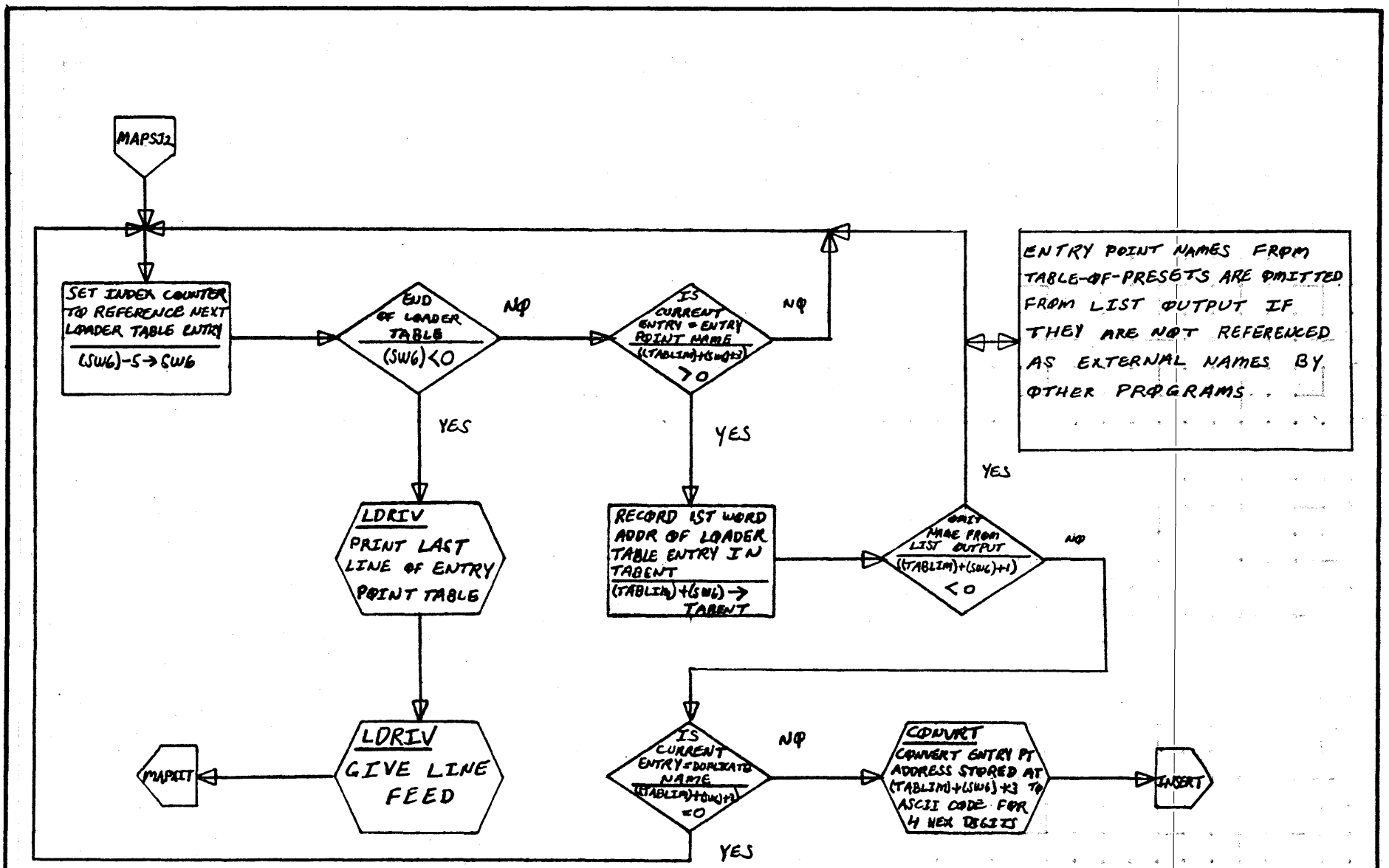
MAR 5 1971

39.181



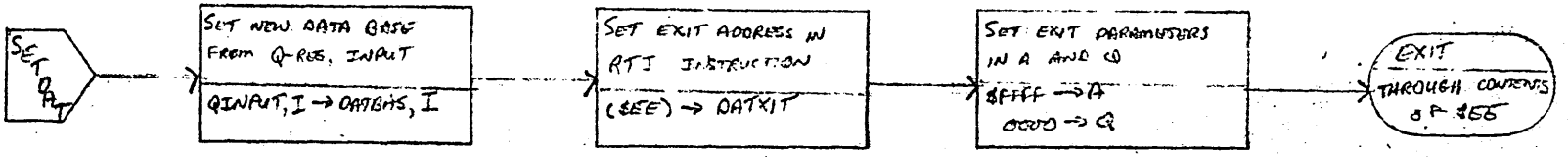
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.		
FLOWCHART ☐	BRANCH	PAGE 19 OF 21	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.			
BRANCH	PAGE 20 OF 21	PROJECT NAME			
NUMBER	ISSUE DATE	TASK NO.			
DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

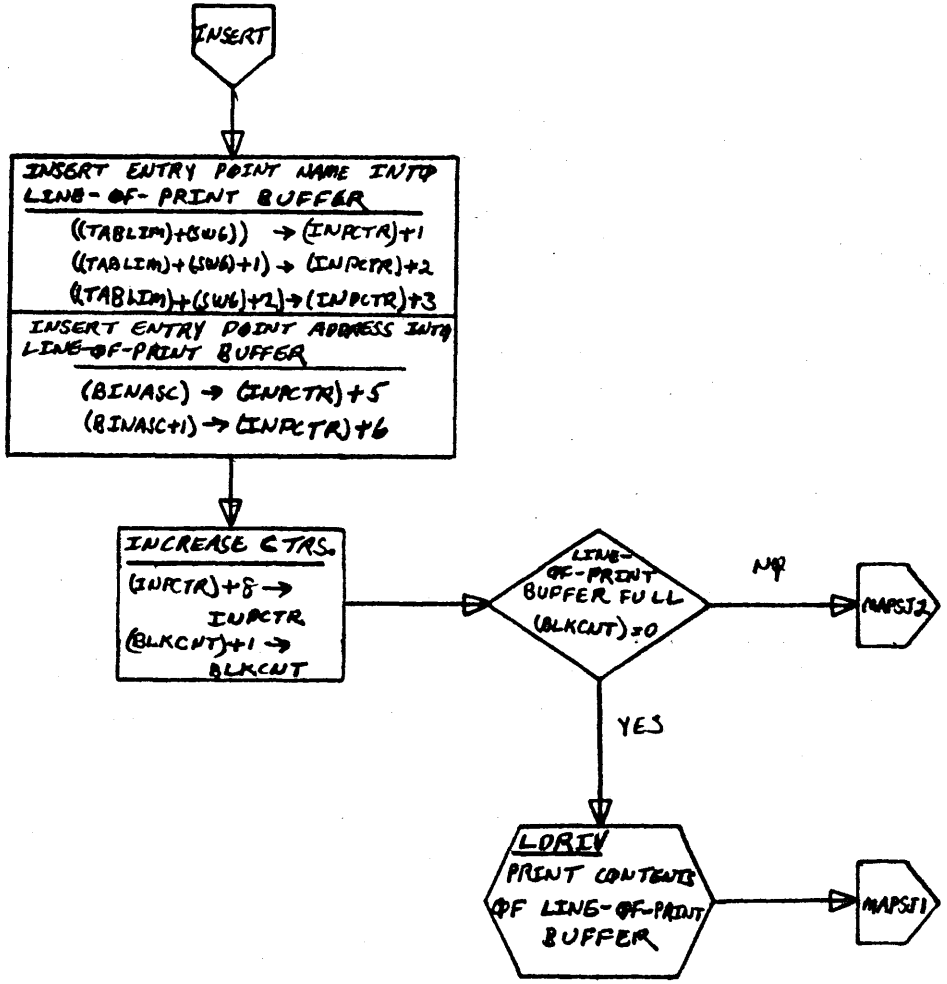


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS UTILS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 DIAGNOSTIC [illegible]	PROJECT MGR.			
	BRANCH PAGE 27 OF 27	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE 4/10/67	TASK NAME			

CA 127-1

MAR 5 1971

59 - 184

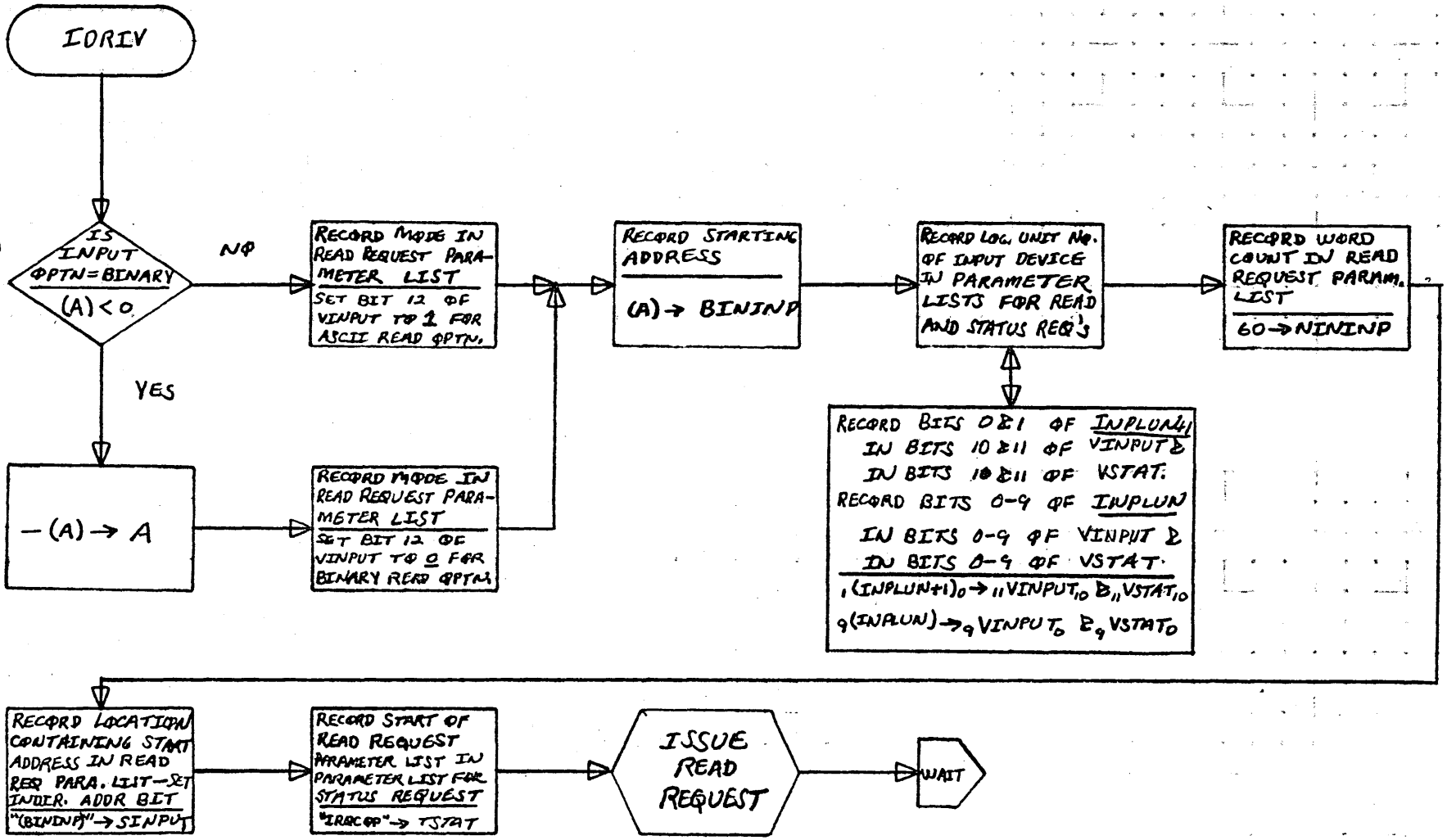


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.			
FLOWCHART ☐	BRANCH	PAGE 28 OF 21	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME			

MAR 5 1971

39.185

AA1385 (FORMERLY CA127-1)

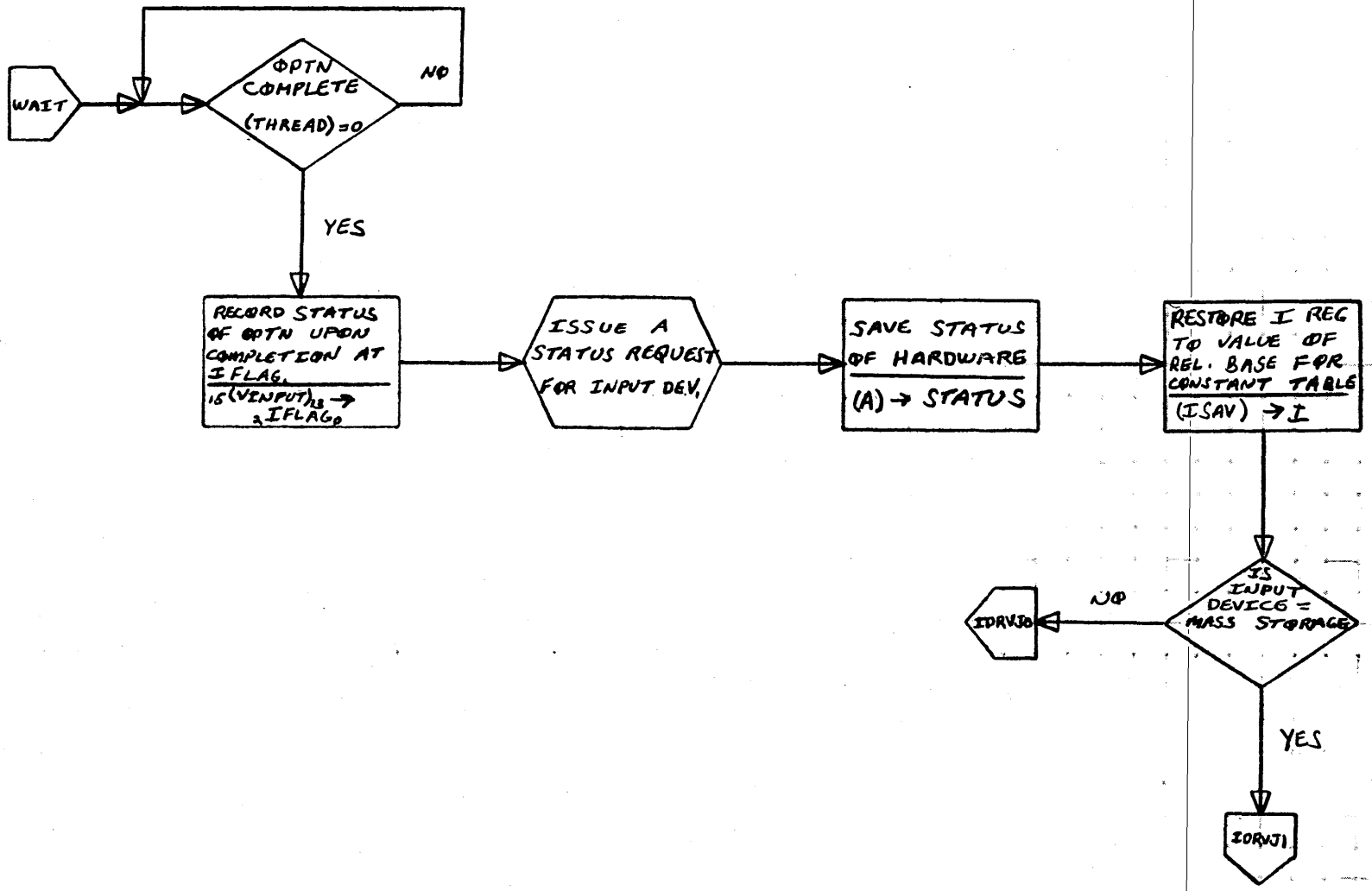


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1100	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1100 Operating System		PROJECT MGR.		
FLOWCHART ☐	IDRIV	PAGE 1 OF 3	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.166

AA1385 (FORMERLY CA127-1)

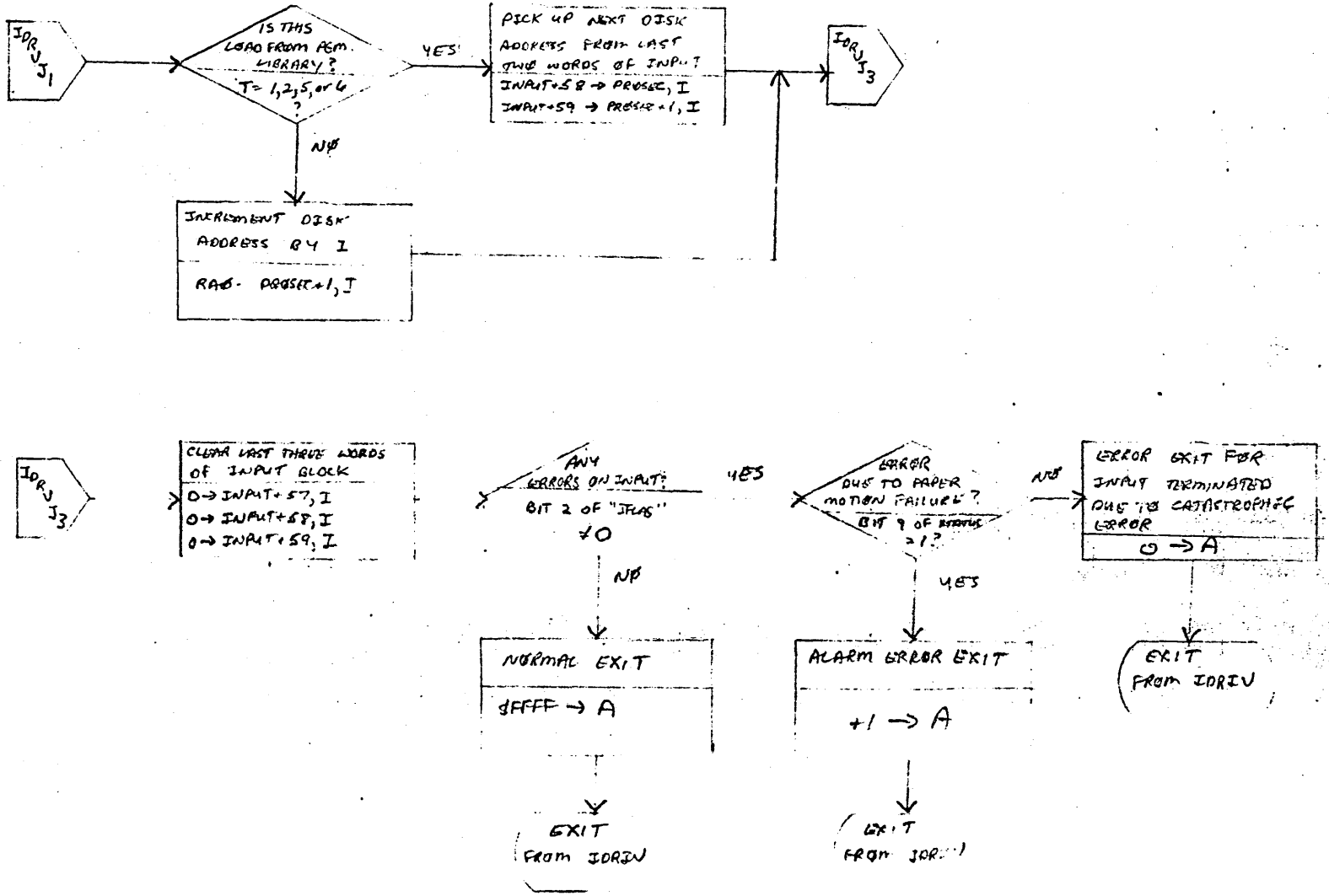


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	IDRIV PAGE 2 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. Gordon DATE 12-6-66	TASK NAME			

AA1385 (FORM CA127-1)

MAR 5 1971

39.187

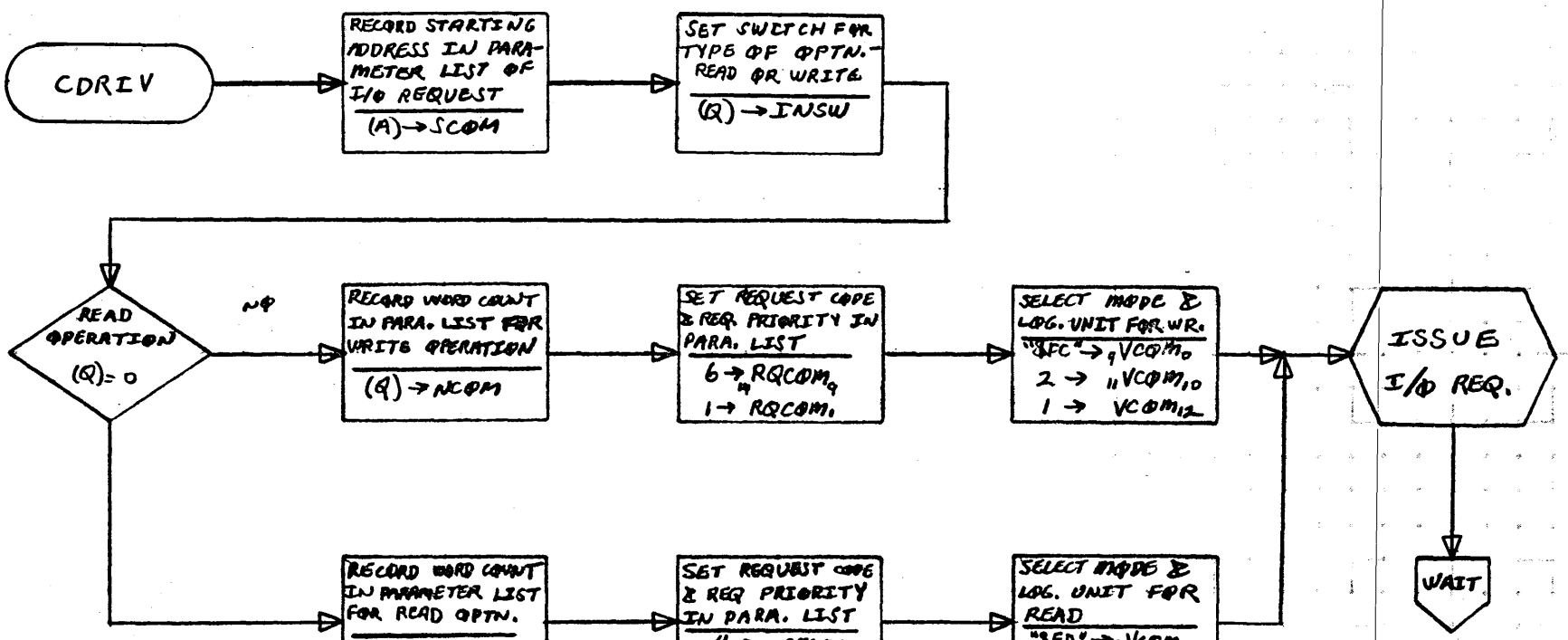


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

60454300

CA 127-1

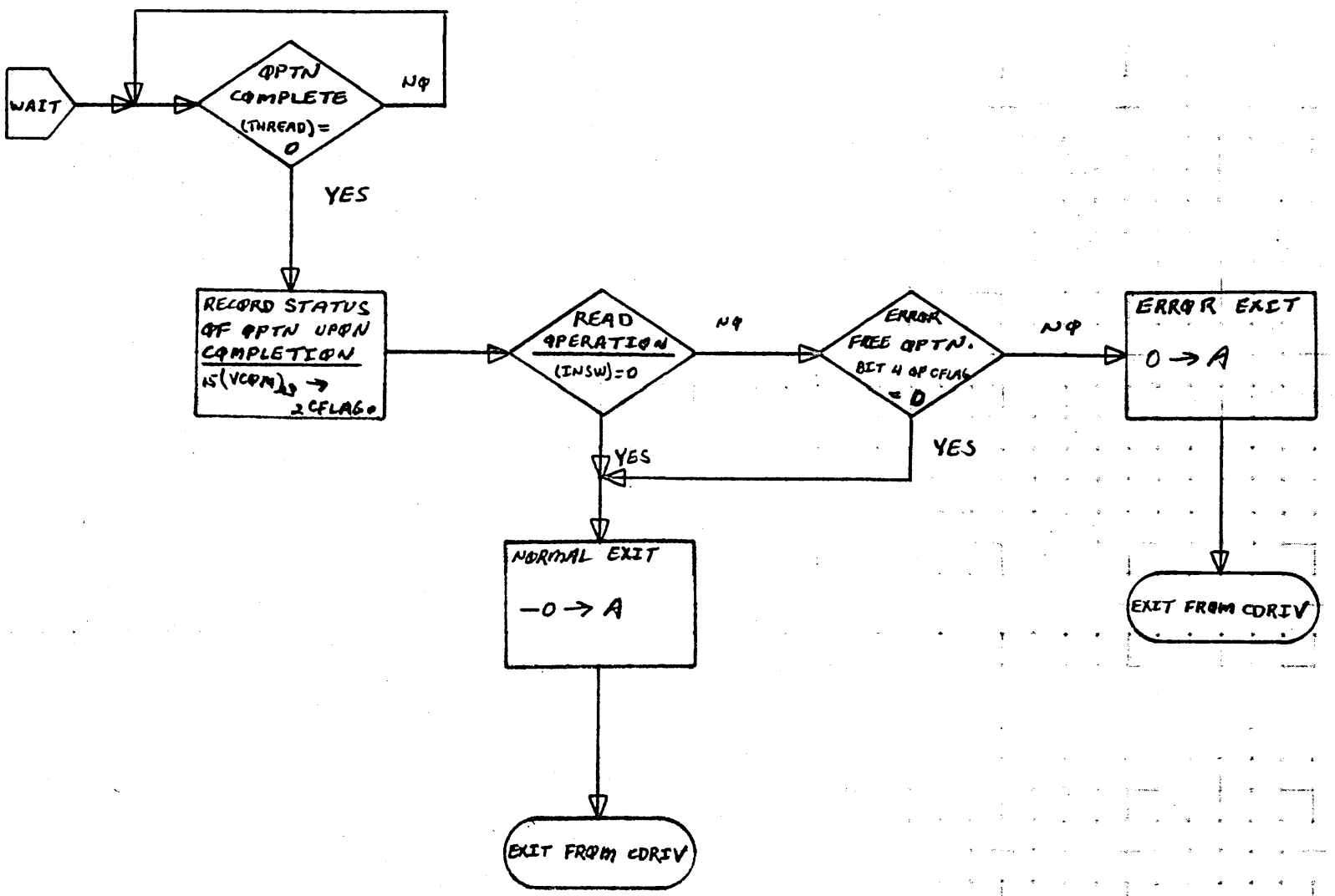


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: 1700 OPERATING SYSTEM		PROJECT MGR.		
	CDRIV	PAGE 1 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY: T. GORDON	DATE: 12-6-66	TASK NAME		

AA1385 (FORM CA127-1)

MAR 5 1971

39.185

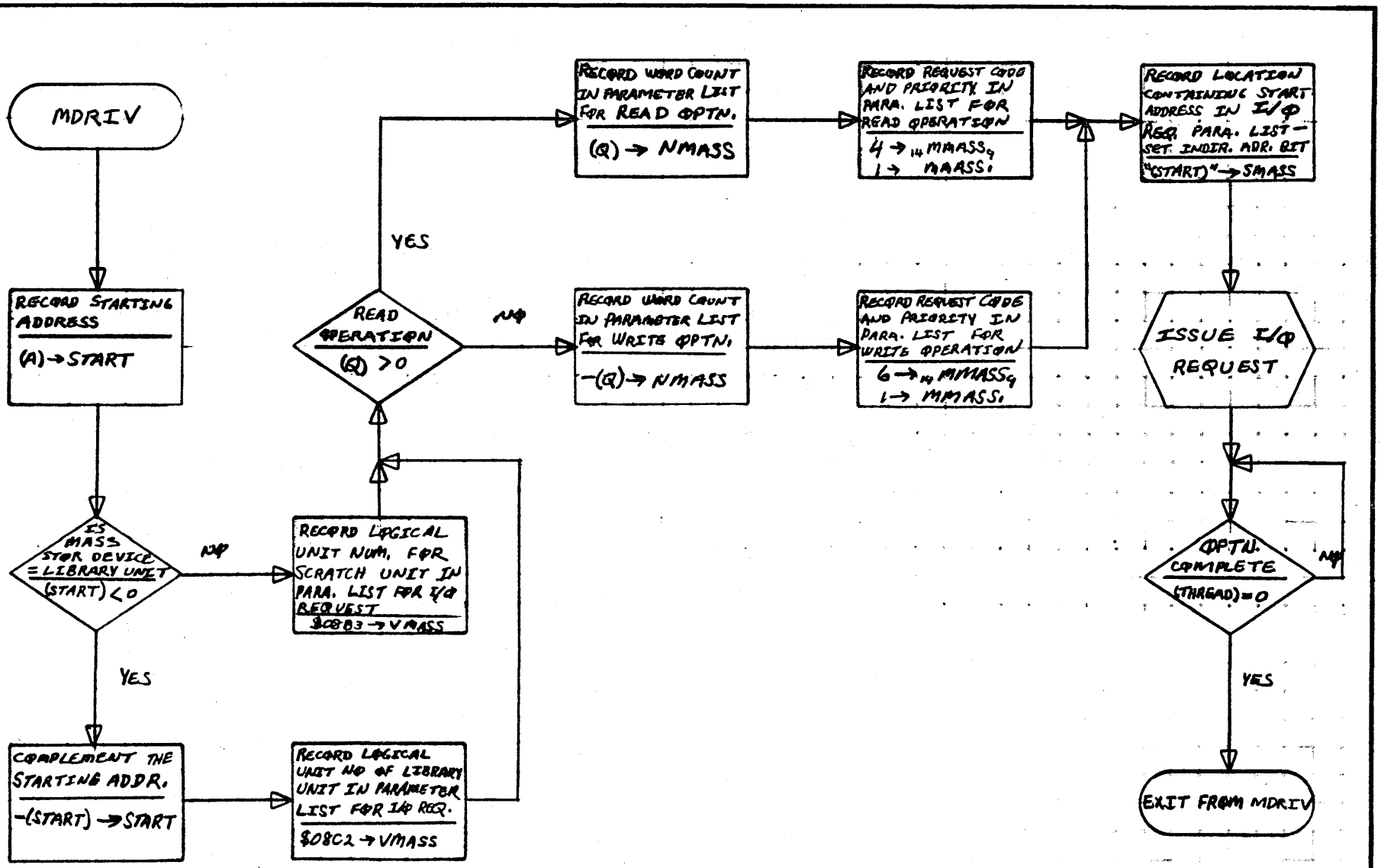


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	CDRIV PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

MAR 5 1971

39.190

AA1385 (FORMERLY CA127-1)

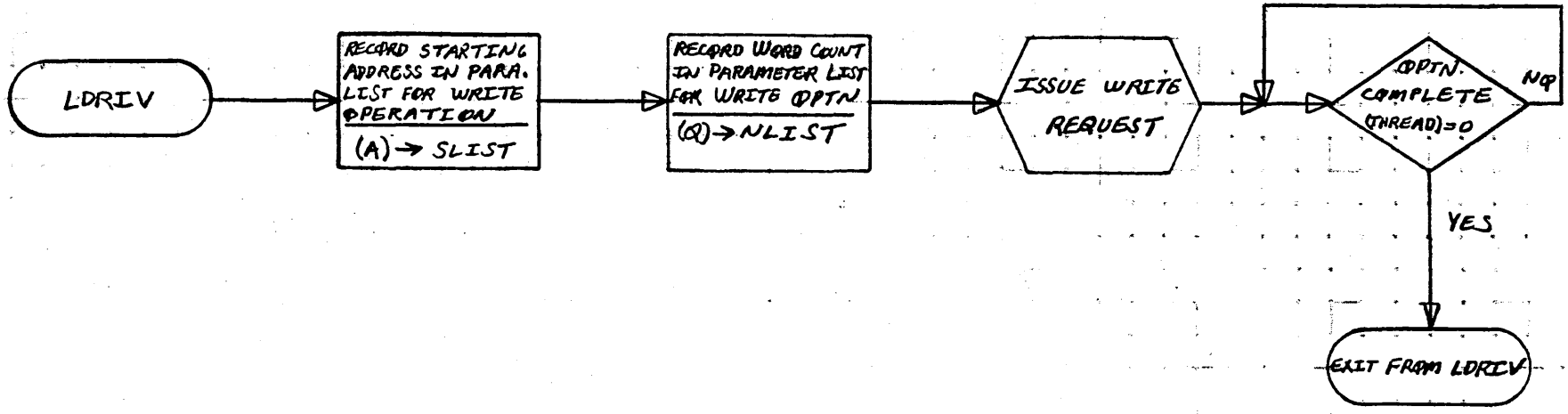


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
	MDRIV	PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.191

AA1385 (FORM CA127-1)

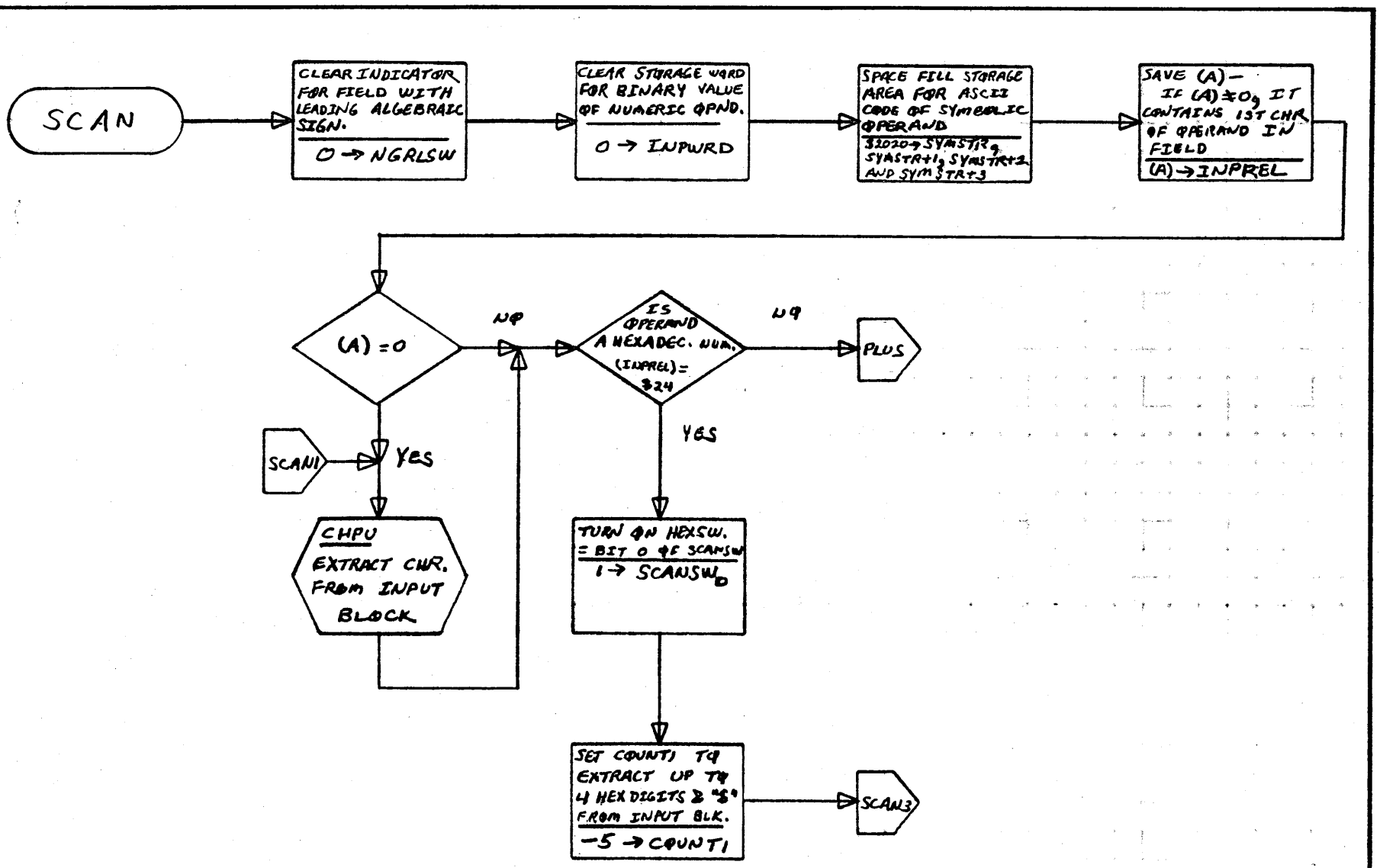


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS THS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	LDRIV PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. Gordon DATE 12-6-66	TASK NAME			

MAR 5 1971

3.9.192

AA1385 (FORMERLY CA127-1)

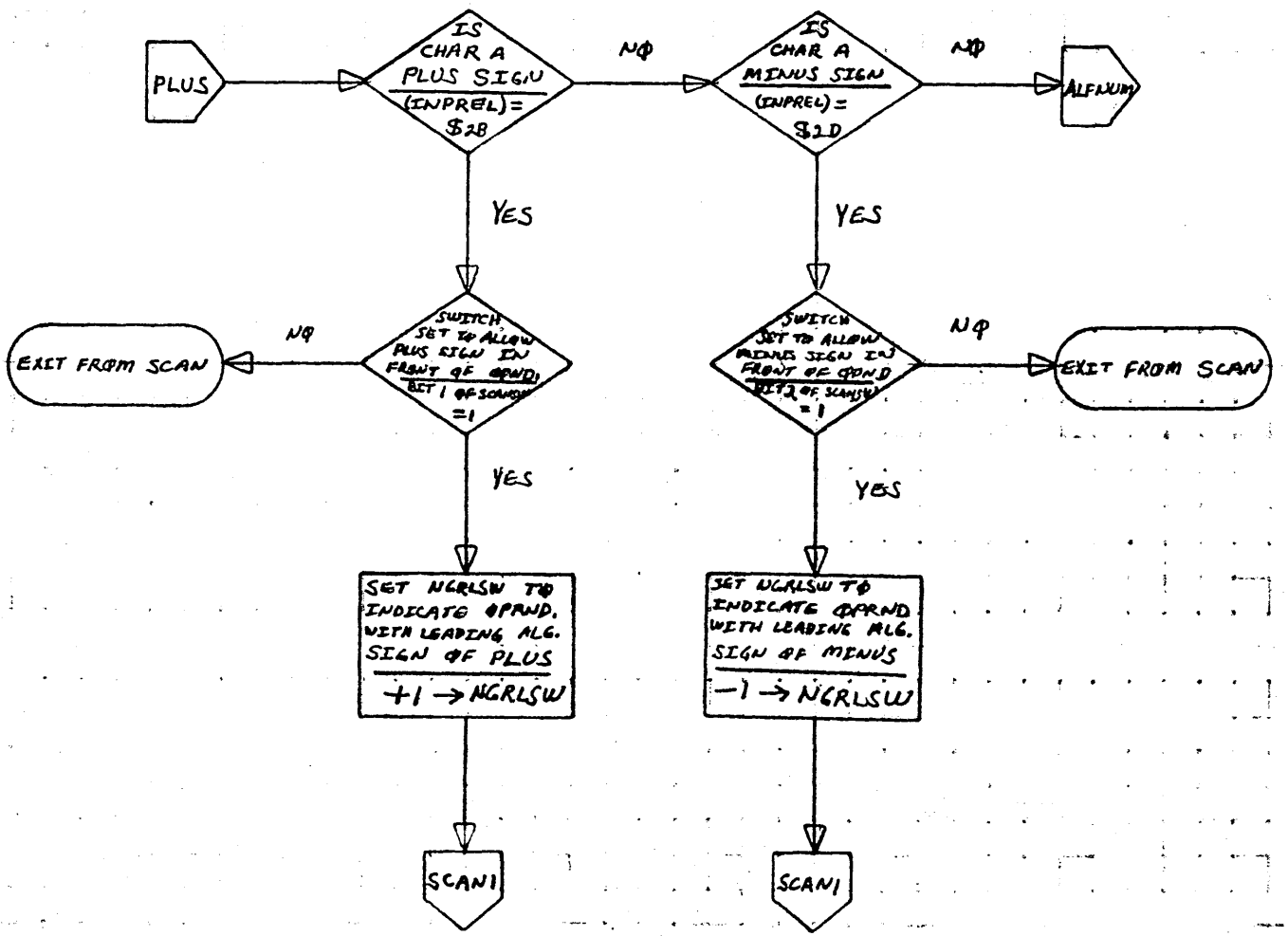


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS THS	MACH. TYPE 1100	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1100 Operating System		PROJECT MGR.		
	SCAN	PAGE 1 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.193

AA1385 (FORME) CA127-1)

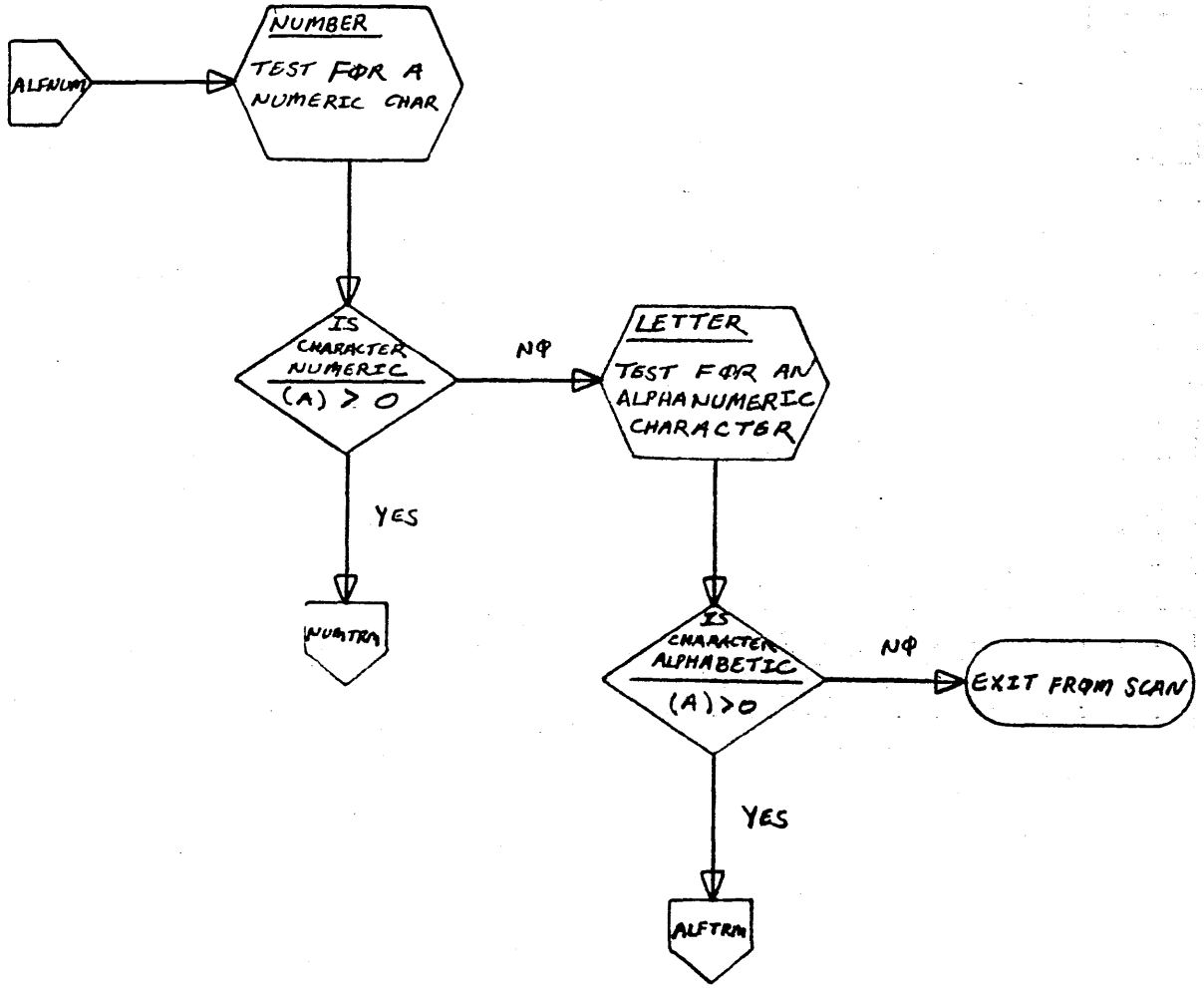


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	SCAN	PAGE 2 OF 10	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

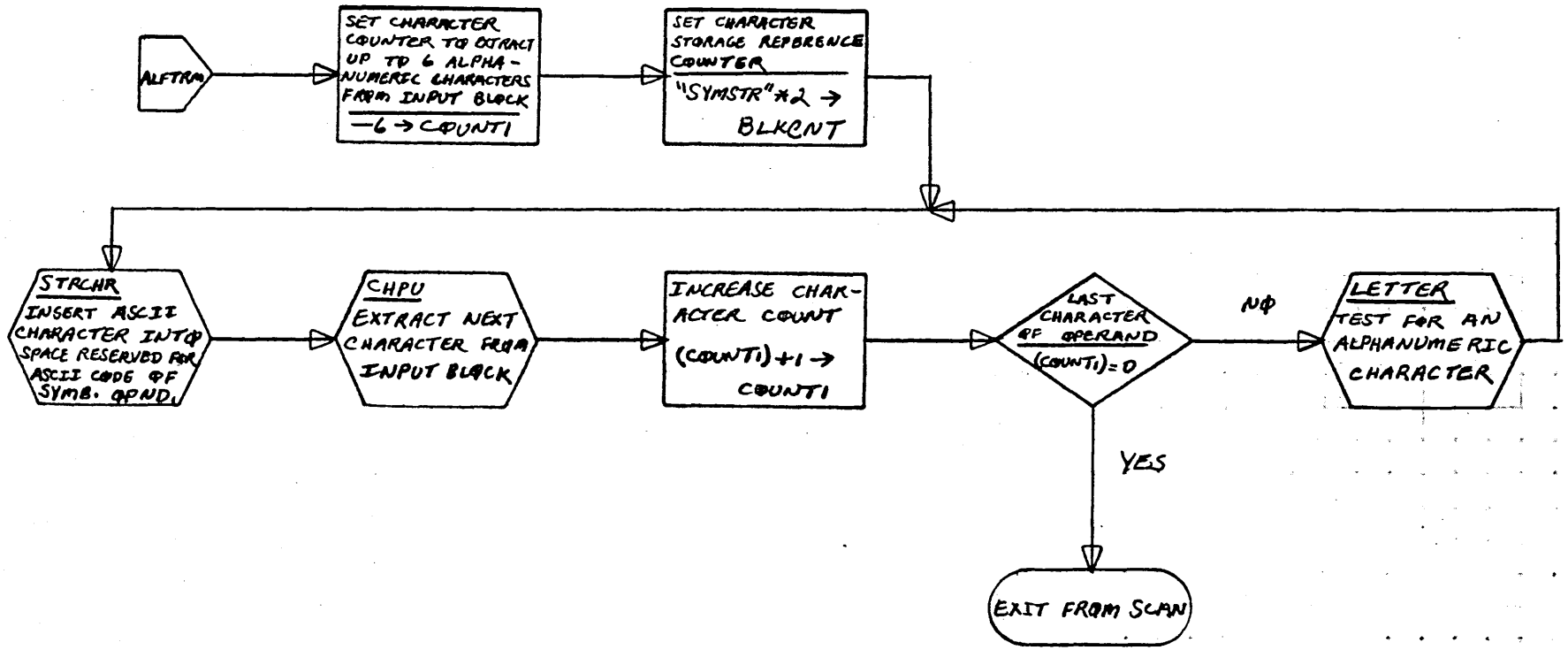
3-9-194

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: I/S MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: 1700 Operating System	PROJECT MGR.			
	SCAN PAGE 3 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY: T. Gleason DATE: 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

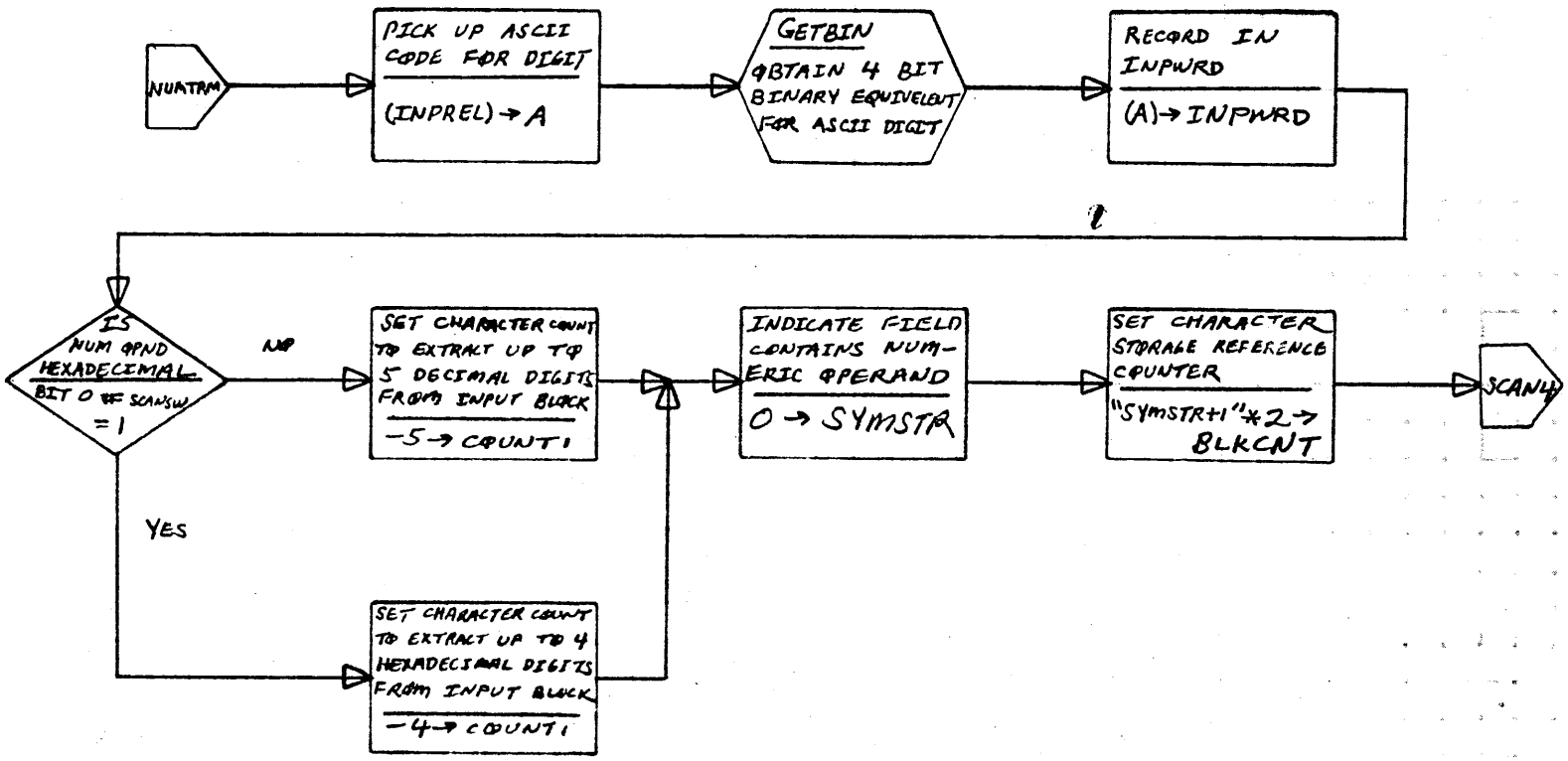


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IHS MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: 1700 OPERATING SYSTEM	PROJECT MGR.			
	SCAN PAGE 4 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY: T. GOLDEN DATE: 12-6-66	TASK NAME			

MAR 5 1971

EL-191

AA1385 (FORMERLY CA127-1)

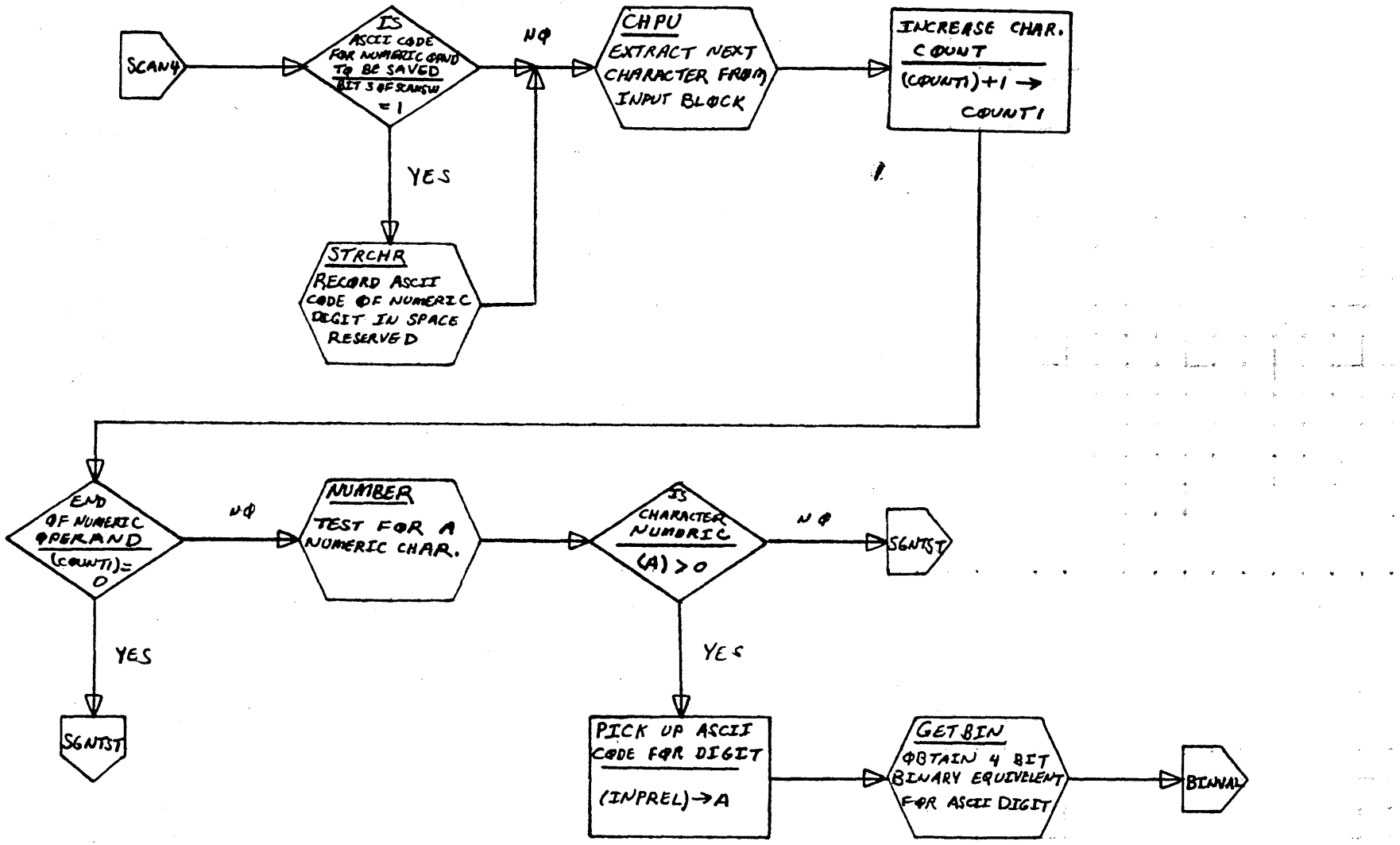


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: 1700 OPERATING SYSTEM		PROJECT MGR.		
	SCAN	PAGE 5 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY: T. GORDON	DATE: 12-6-66	TASK NAME		

MAR 5 1971

339197

FORMER CA127-1)

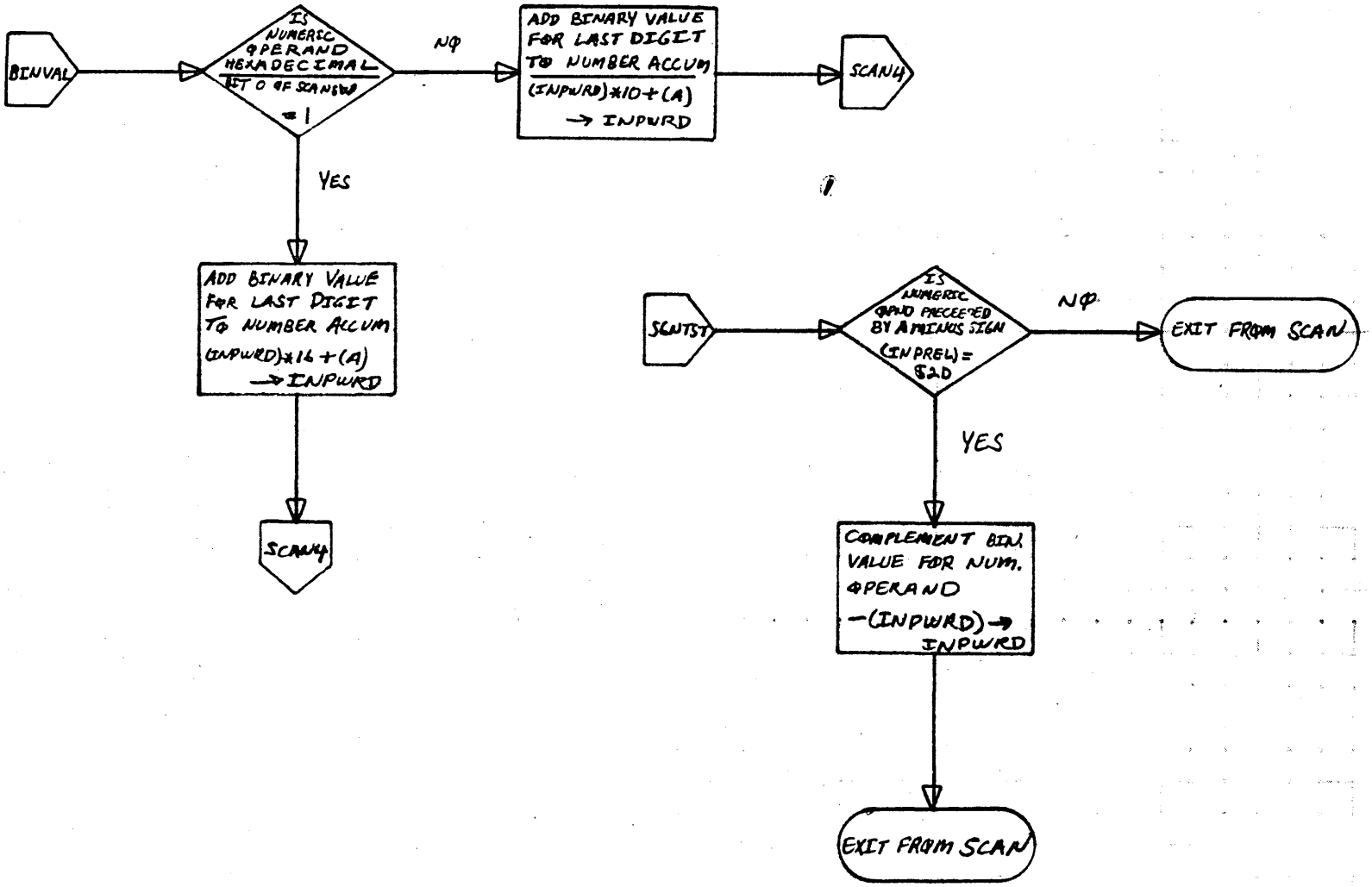


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS JHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.			
FLOWCHART ☐	SCAN	PAGE 6 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME			

MAR 5 1971

39 1.198

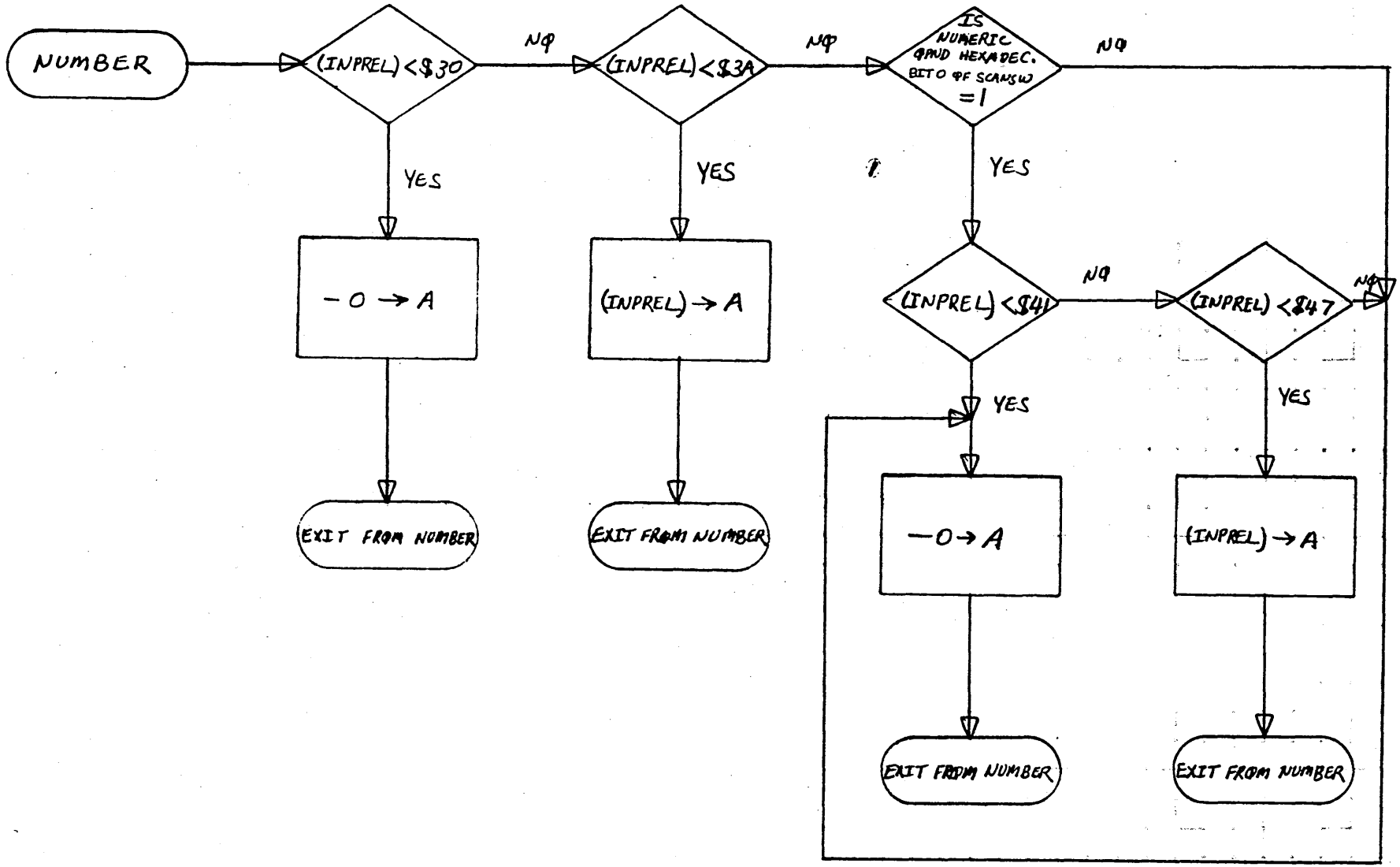
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.		
	SCAN	PAGE 1 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. GORDON	DATE 12 to 66	TASK NAME		

MAR 5 1971

AA1385 (FORM CA127-1)

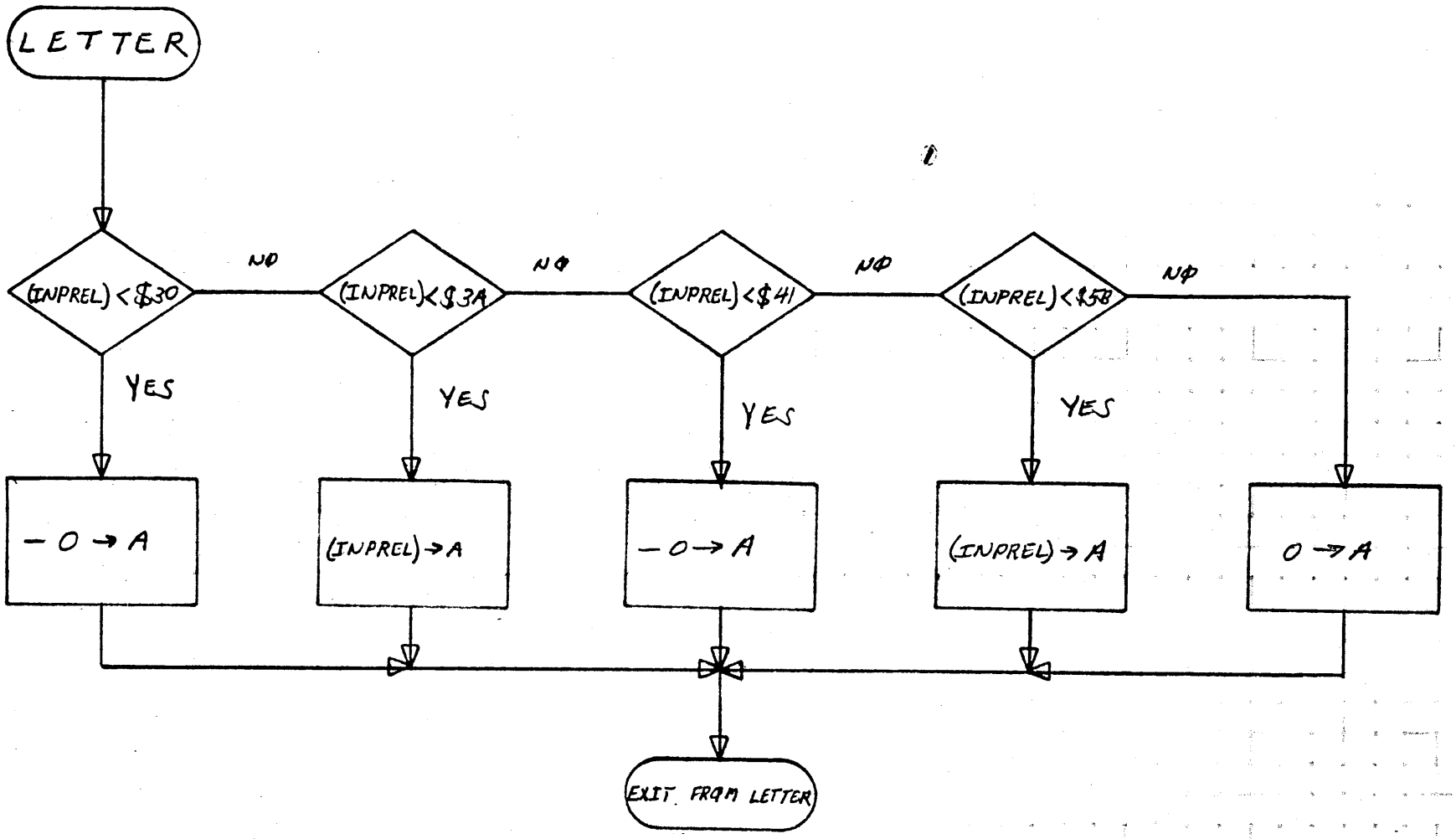


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS THS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.		
	SCAN	PAGE 8 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

MAR 5 1971

39. 200

AA1385 (FORMERLY CA127-1)

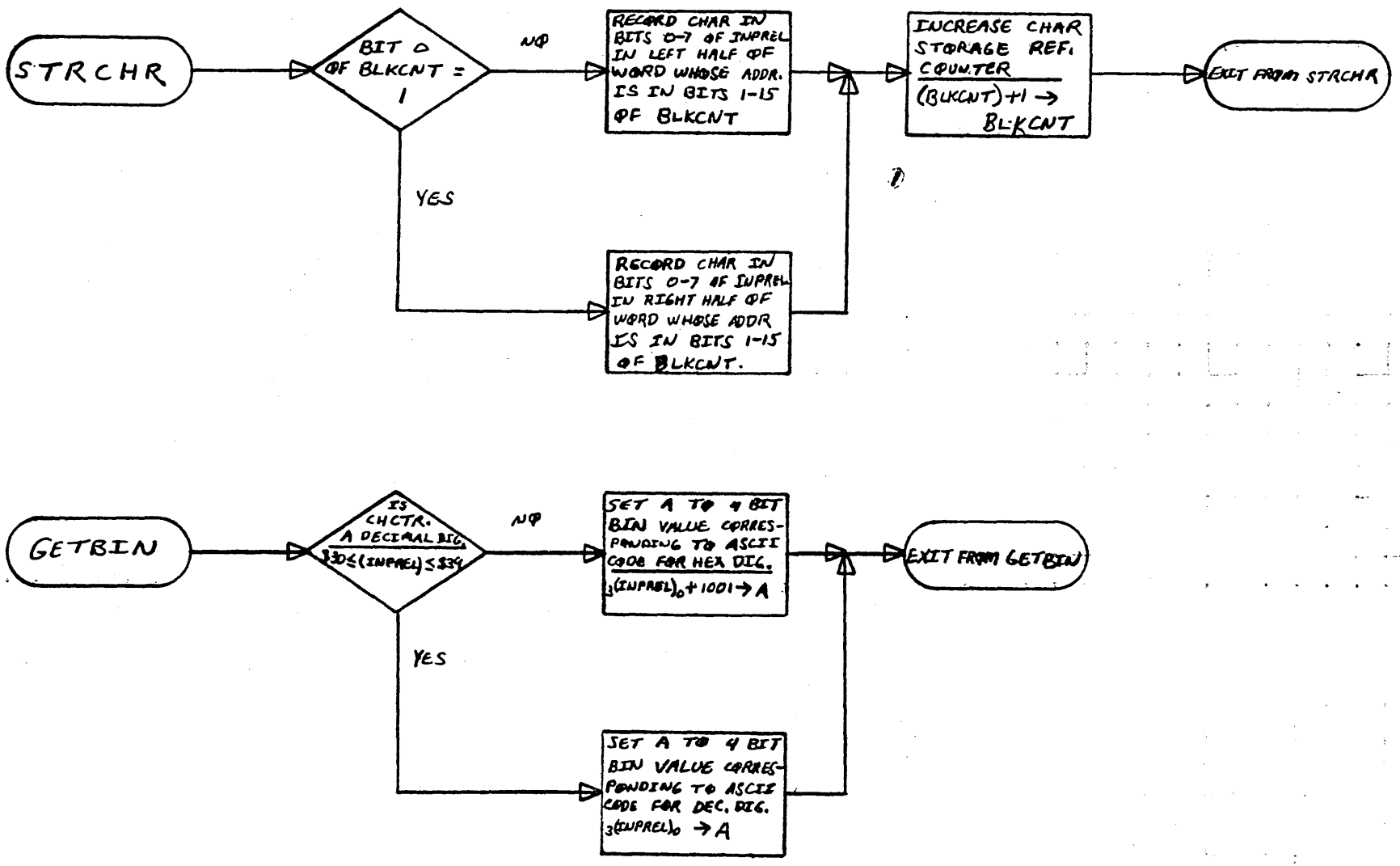


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.			
	SCAN	PAGE 9 OF 10	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME			

AA1385 (FORME CA127-1)

MAR 5 1971

39.201



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS: IMS — MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE: 1700 OPERATING SYSTEM	PROJECT MGR.			
SCAN — PAGE 10 OF 10	PROJECT NAME			
NUMBER — ISSUE DATE	TASK NO.			
DRAWN BY: T. GORDON — DATE: 12-6-66	TASK NAME			

MAR 5 1971

39202

AA1385 (FORMERLY CA127-1)

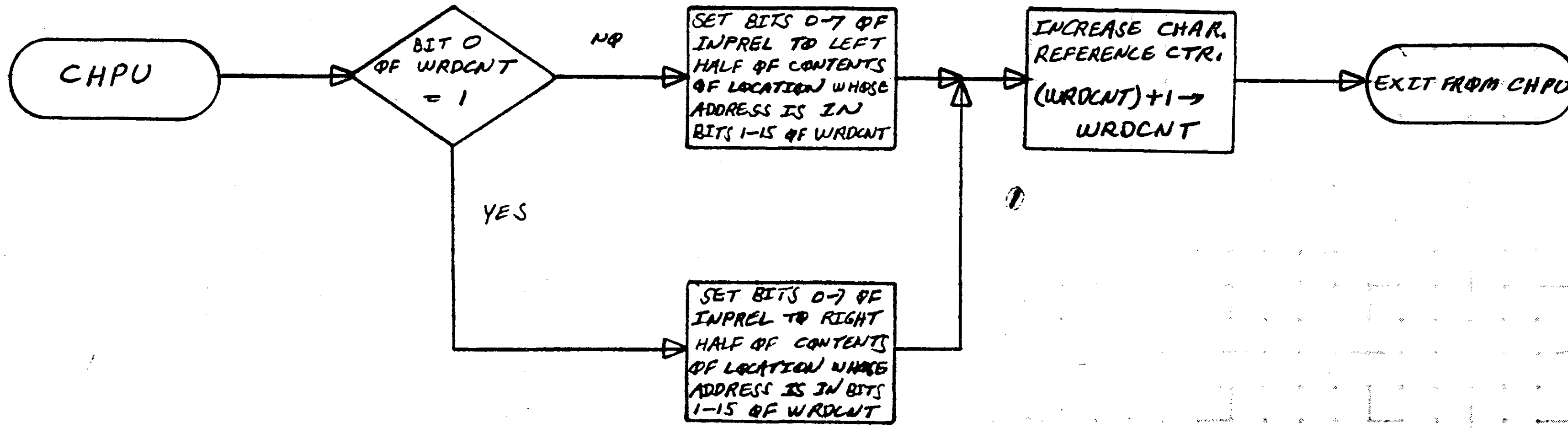
1 2 3 4 5

A

B

C

D



MAR 5 1971

39.203

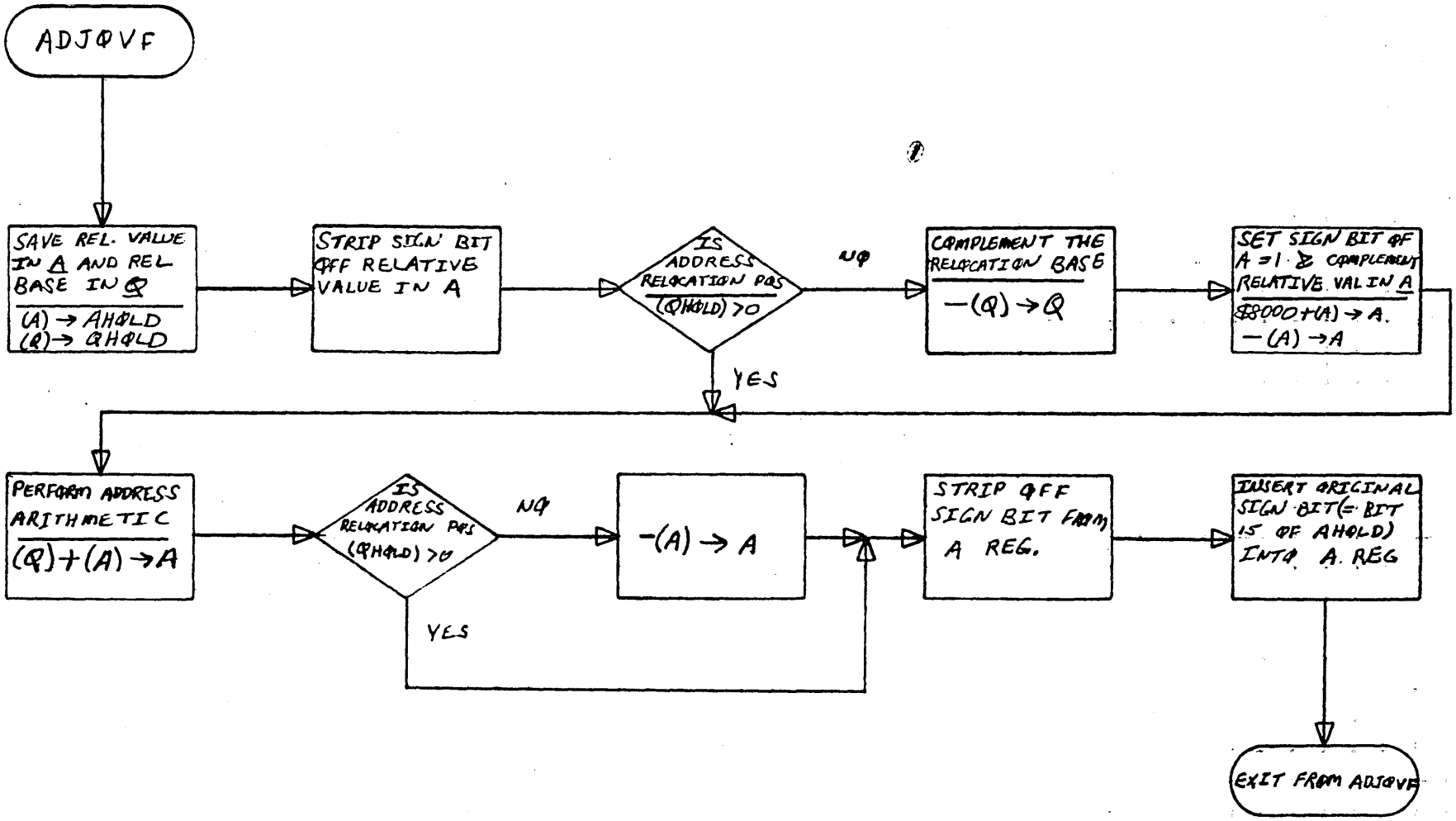
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	
DOCUMENT TITLE 1700 OPERATING System		PROJECT MGR.	
CHPU	PAGE 1 OF 1	PROJECT NAME	
NUMBER	ISSUE DATE	TASK NO.	
DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME	

REV	APPROVED	DATE

AA1385 (FORMERLY CA127-1)

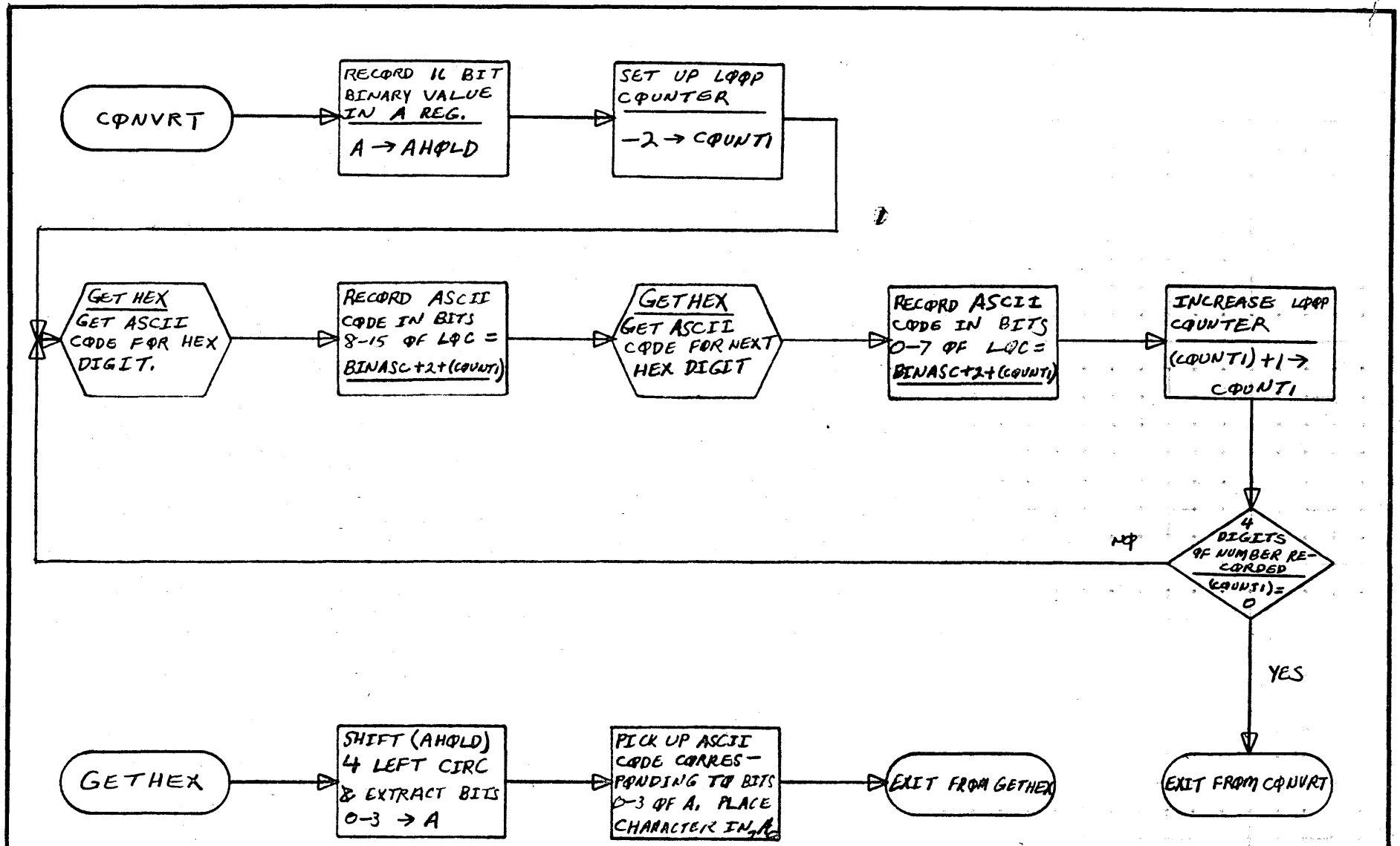


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	ADJØVF PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

MAR 5 1971

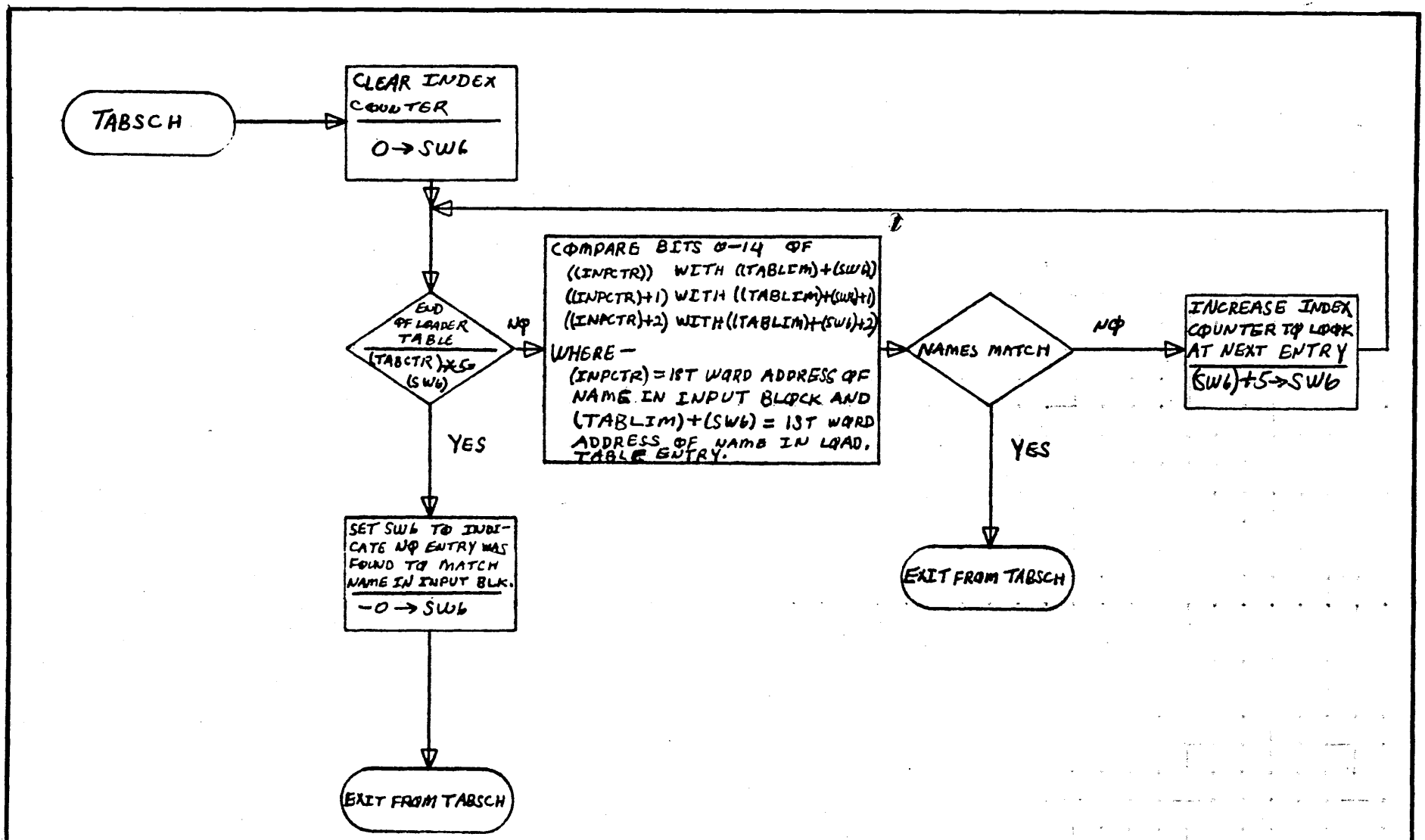
39.204

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	CONVRT	PAGE 1 OF 1	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

AA1385 (FORMERLY CA127-1)

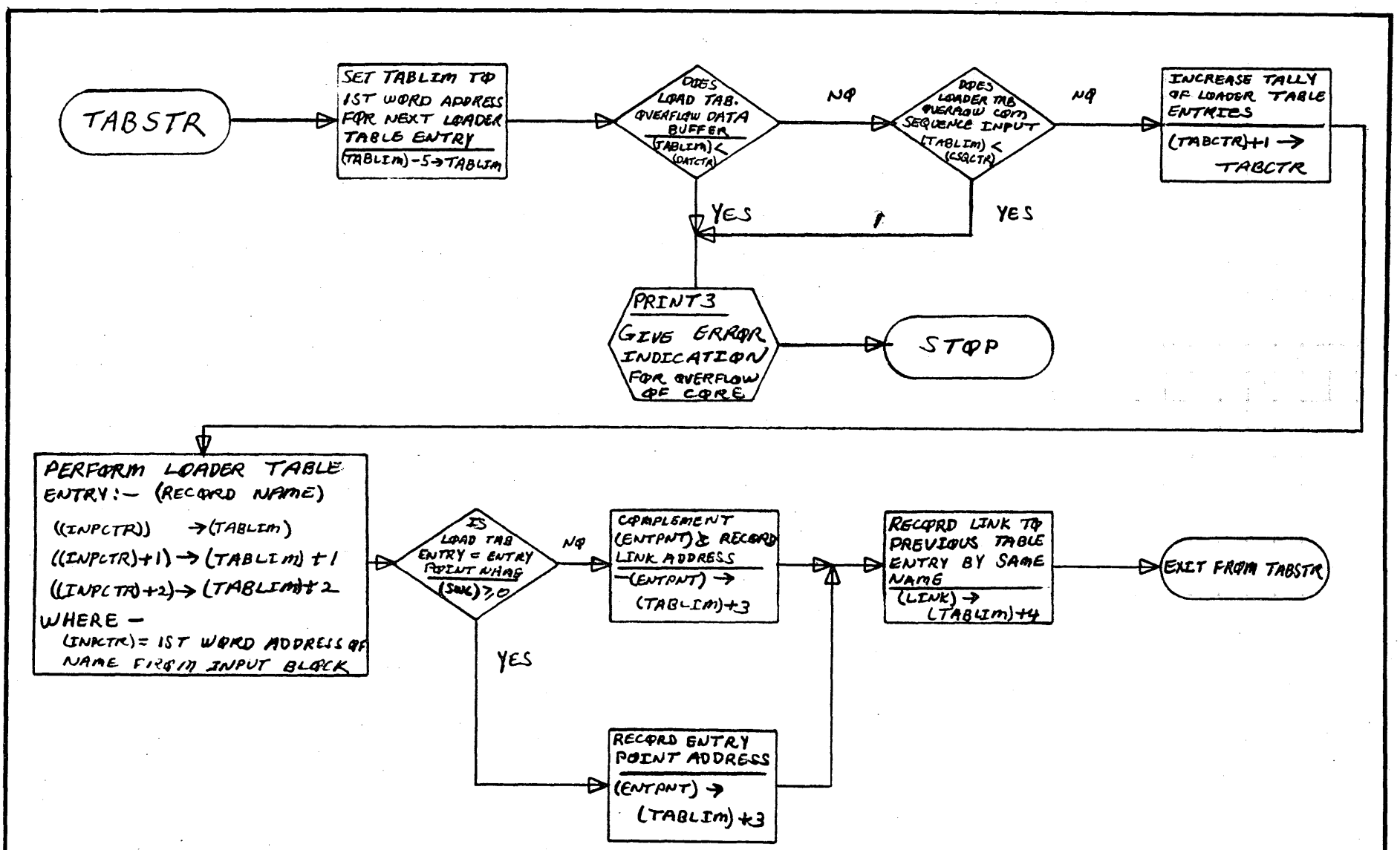


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
	TABSCH	PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39,206

AA1385 (FORMERLY CA127-1)

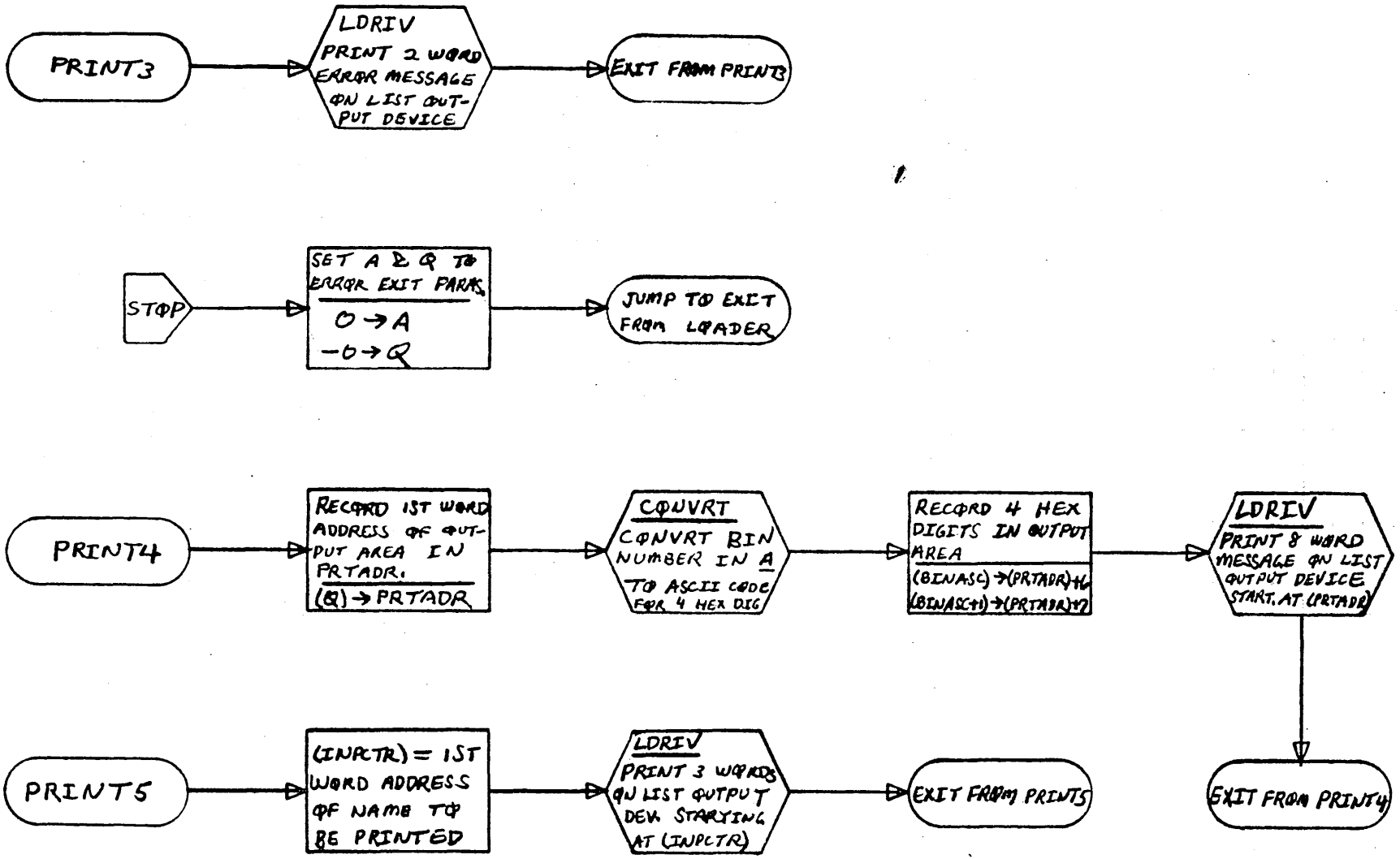


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: EOS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1700 OPERATING SYSTEM		PROJECT MGR.		
FLOWCHART ☐	TABSTR	PAGE 1 OF 1	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY: T. GORDON	DATE: 12-6-66	TASK NAME		

MAR 5 1971

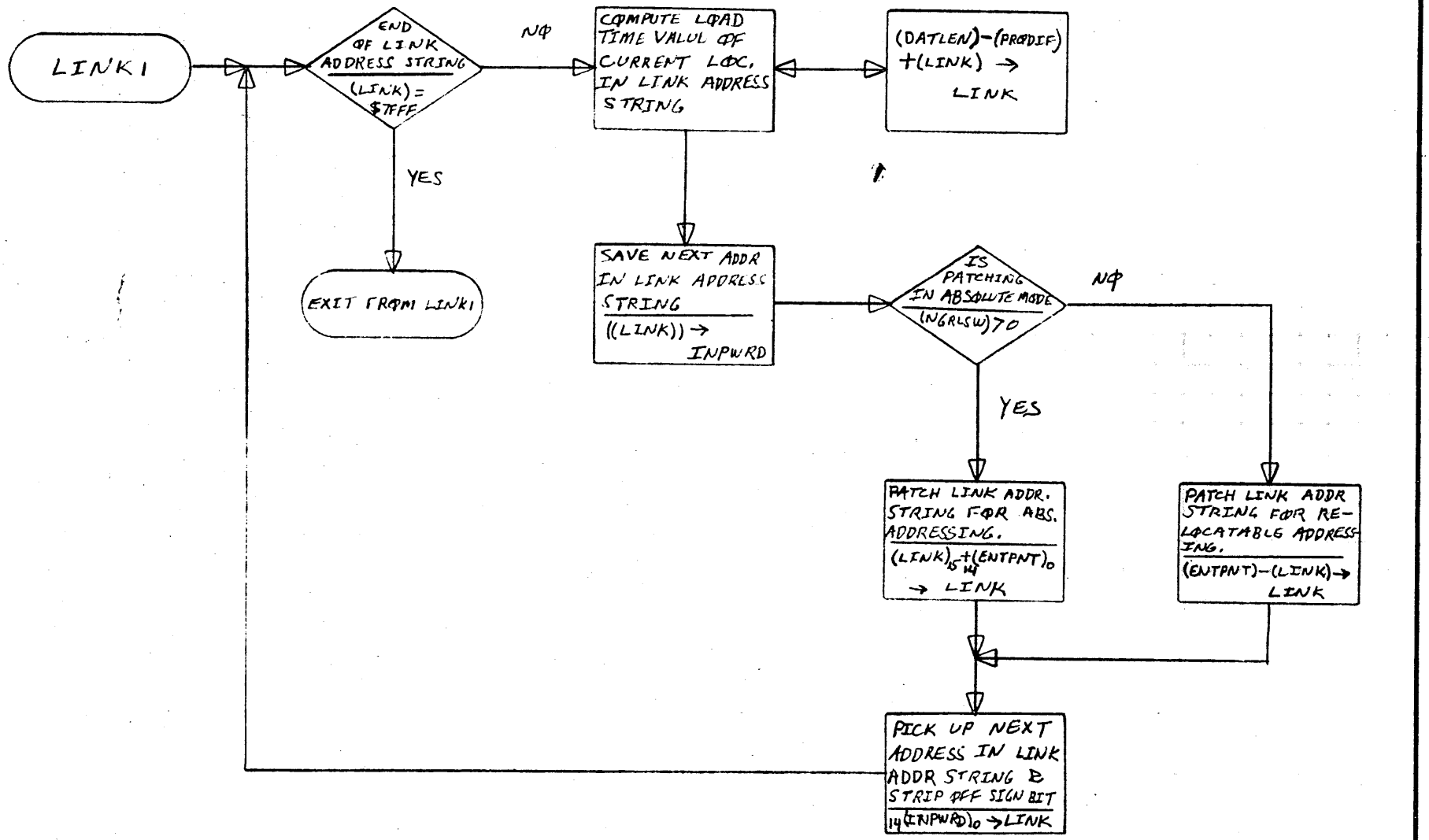
39.207

AA1385 (FORMER CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	LSTOCT	PAGE 1 OF 1	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

AA1385 (FORMERLY CA127-1)

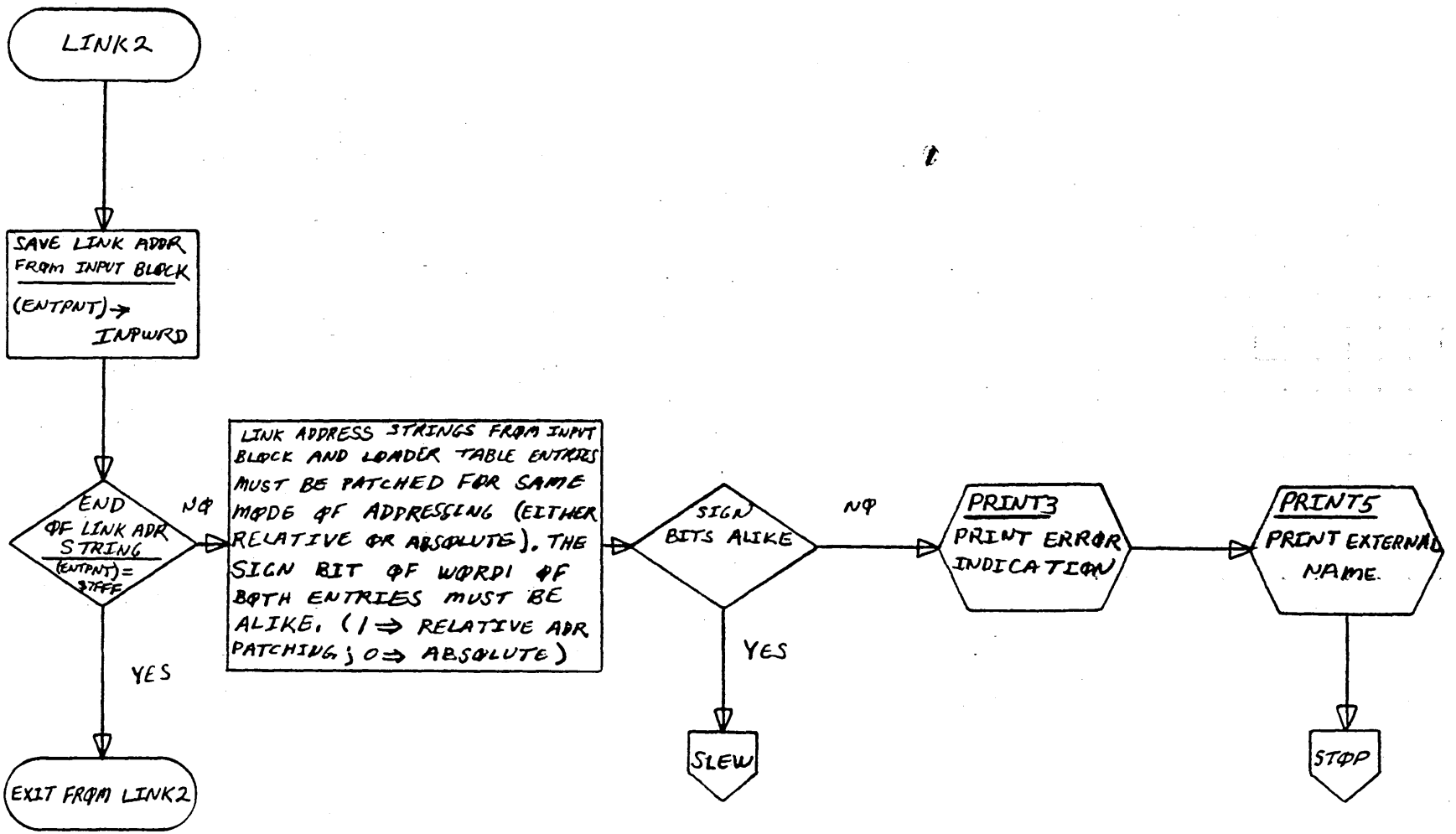


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	LINK1 PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. Gordon DATE 12-6-66	TASK NAME			

MAR 5 1971

39.209

AA1385 (FORMERLY CA127-1)

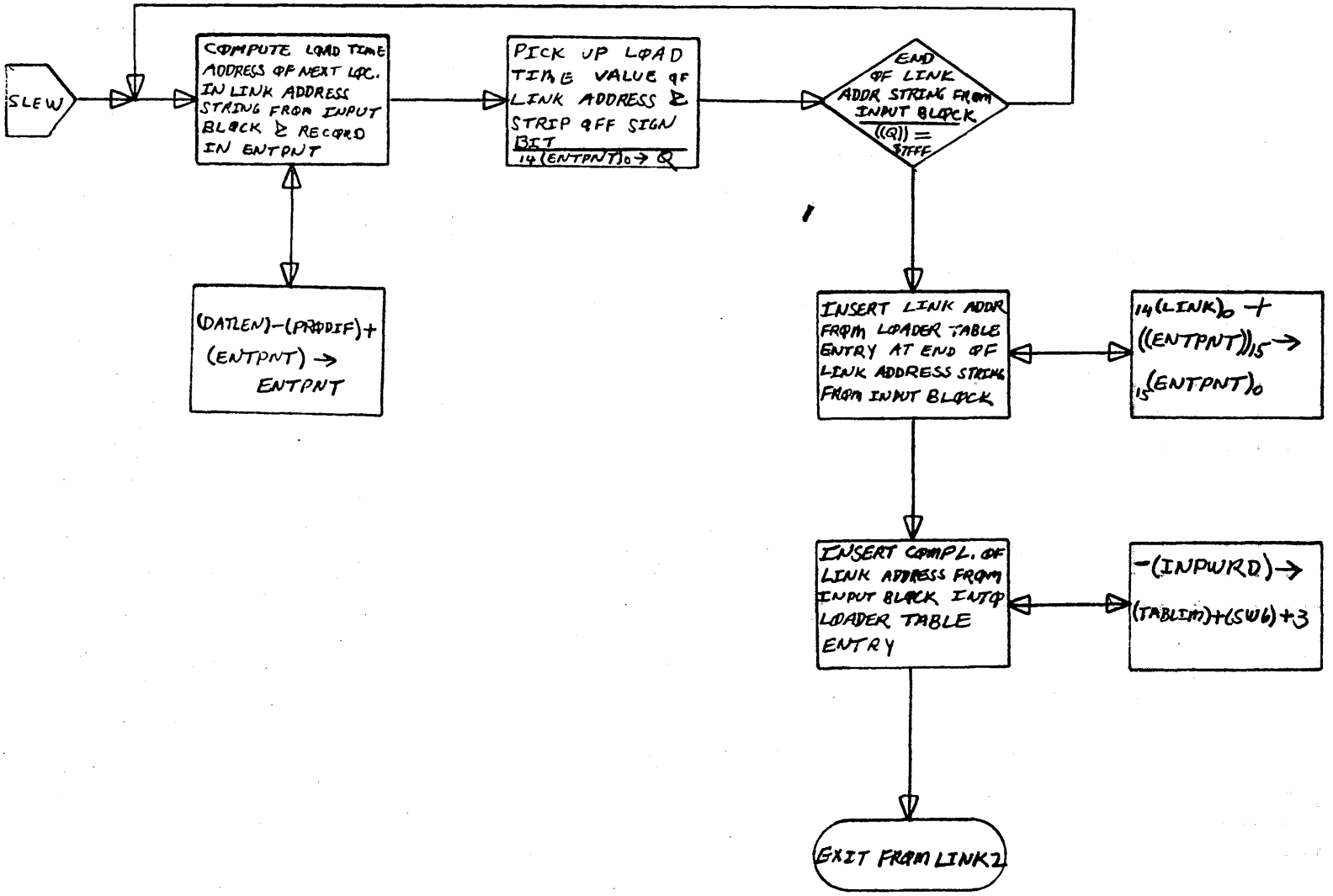


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.		
FLOWCHART ☐	LINK-2	PAGE 1 OF 2	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.210

AA1385 (FORMERLY CA127-1)

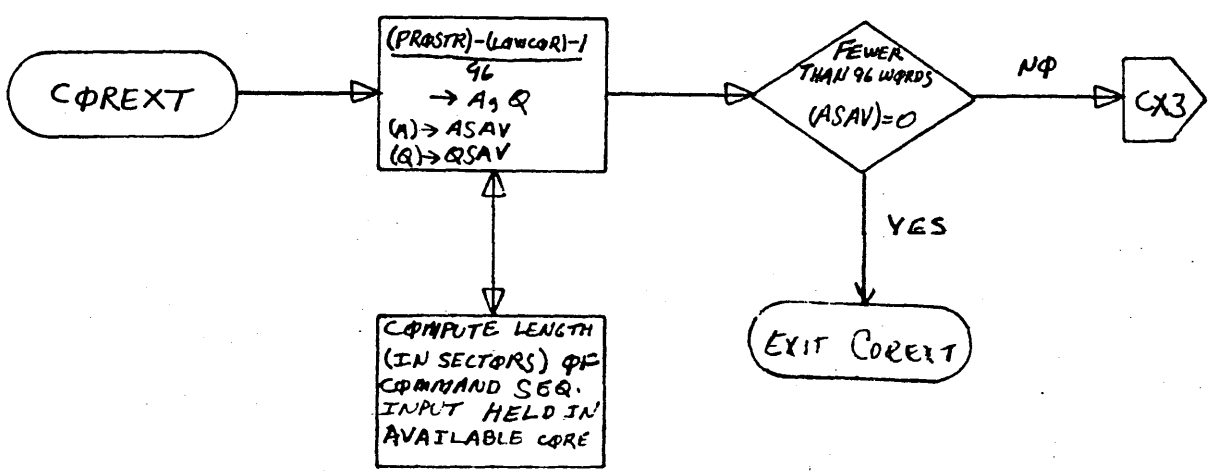


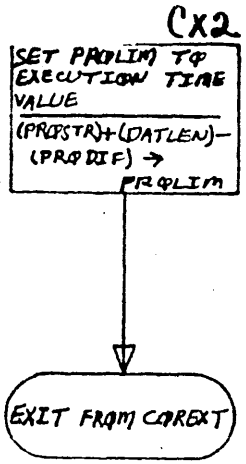
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	LINK2 PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY F. GORDON DATE 12-6-66	TASK NAME			

AA1385 (FORME A127-1)

MAR 5 1971

39.211

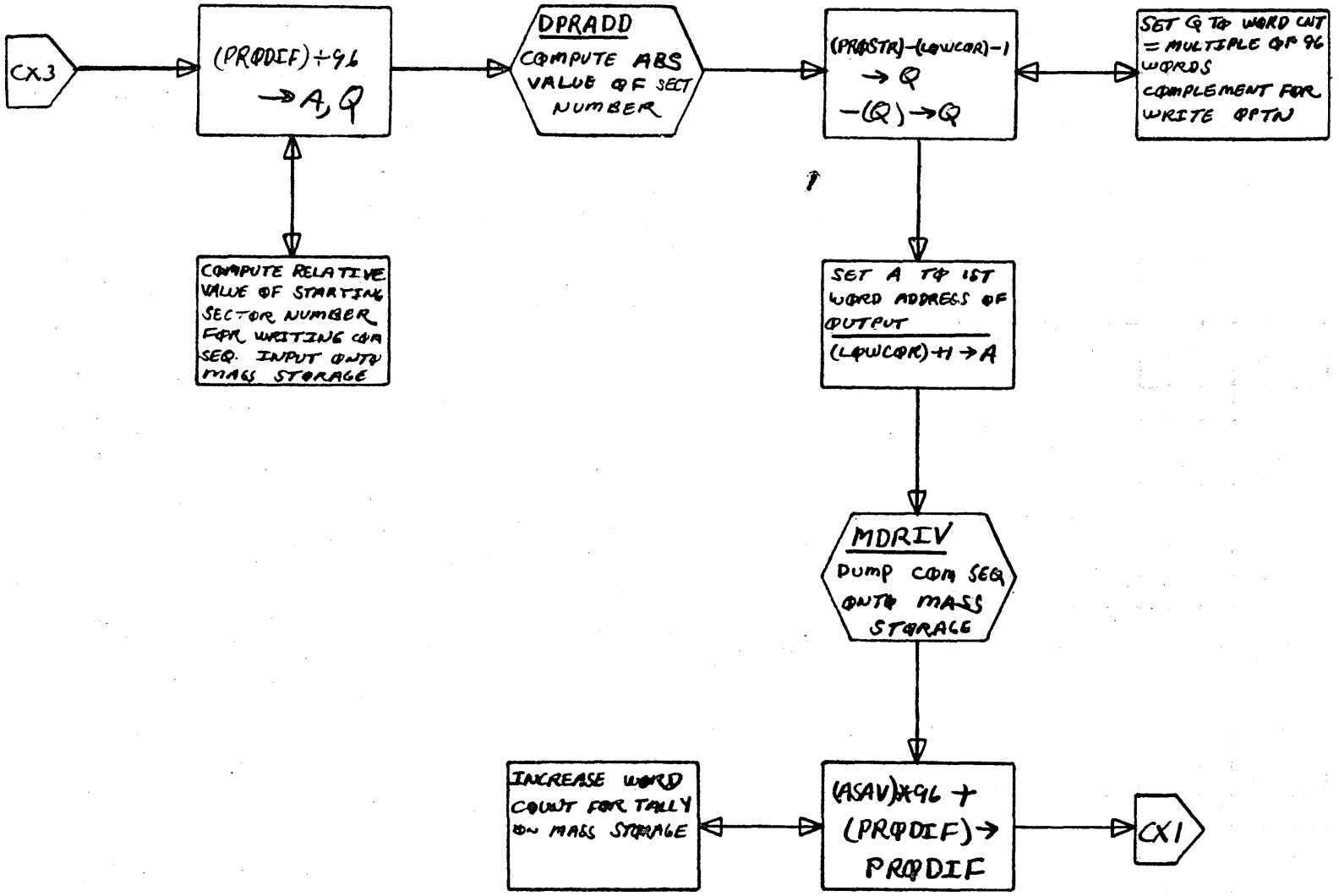




CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.		
	COREXT	PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

MAR 5 1971

39 2 12

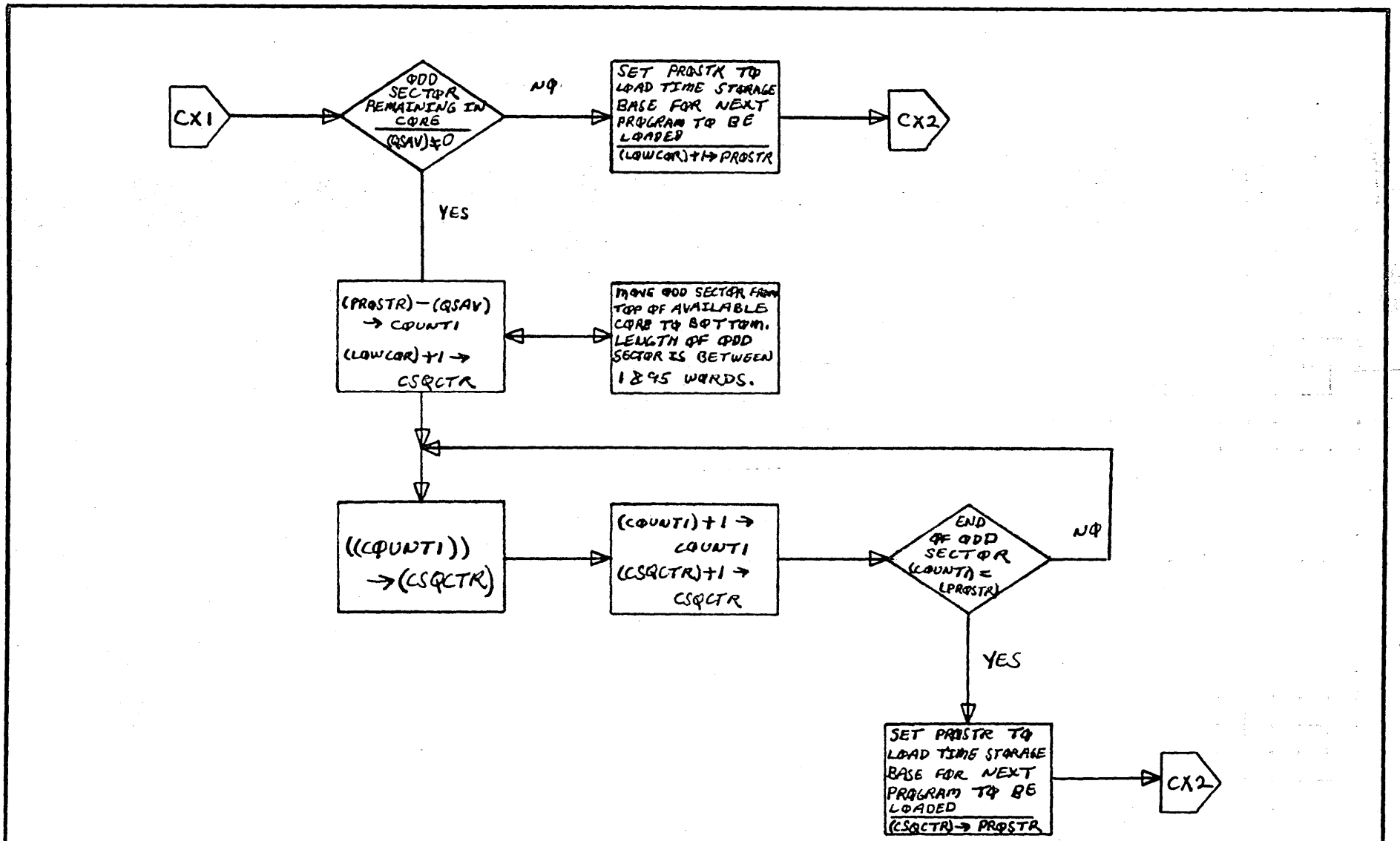


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.			
	CORE+1	PAGE 2 OF 3	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

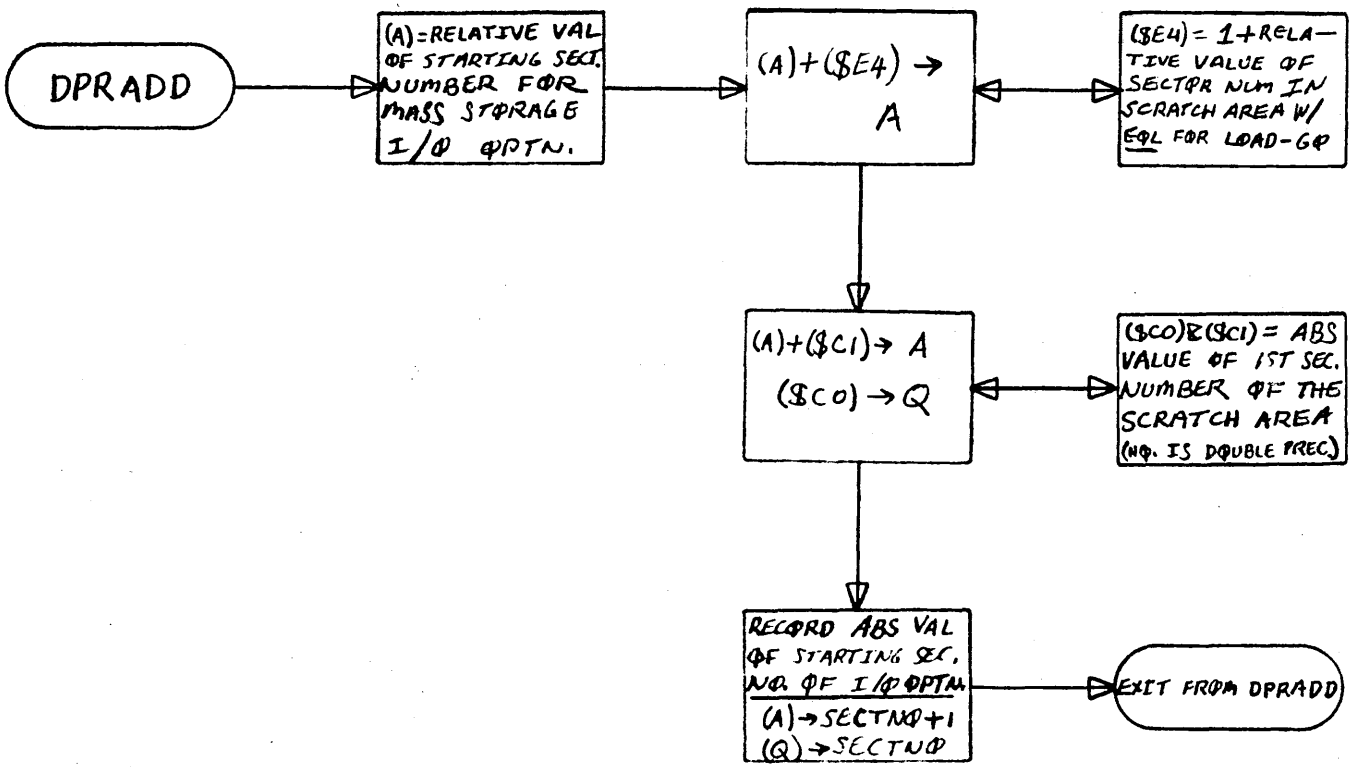
39.213



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: EM8	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	COREXT	PAGE 3 OF 3	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY: T. Gordon	DATE: 12-6-66	TASK NAME		

MAR 5 1971

MAR 5 1971

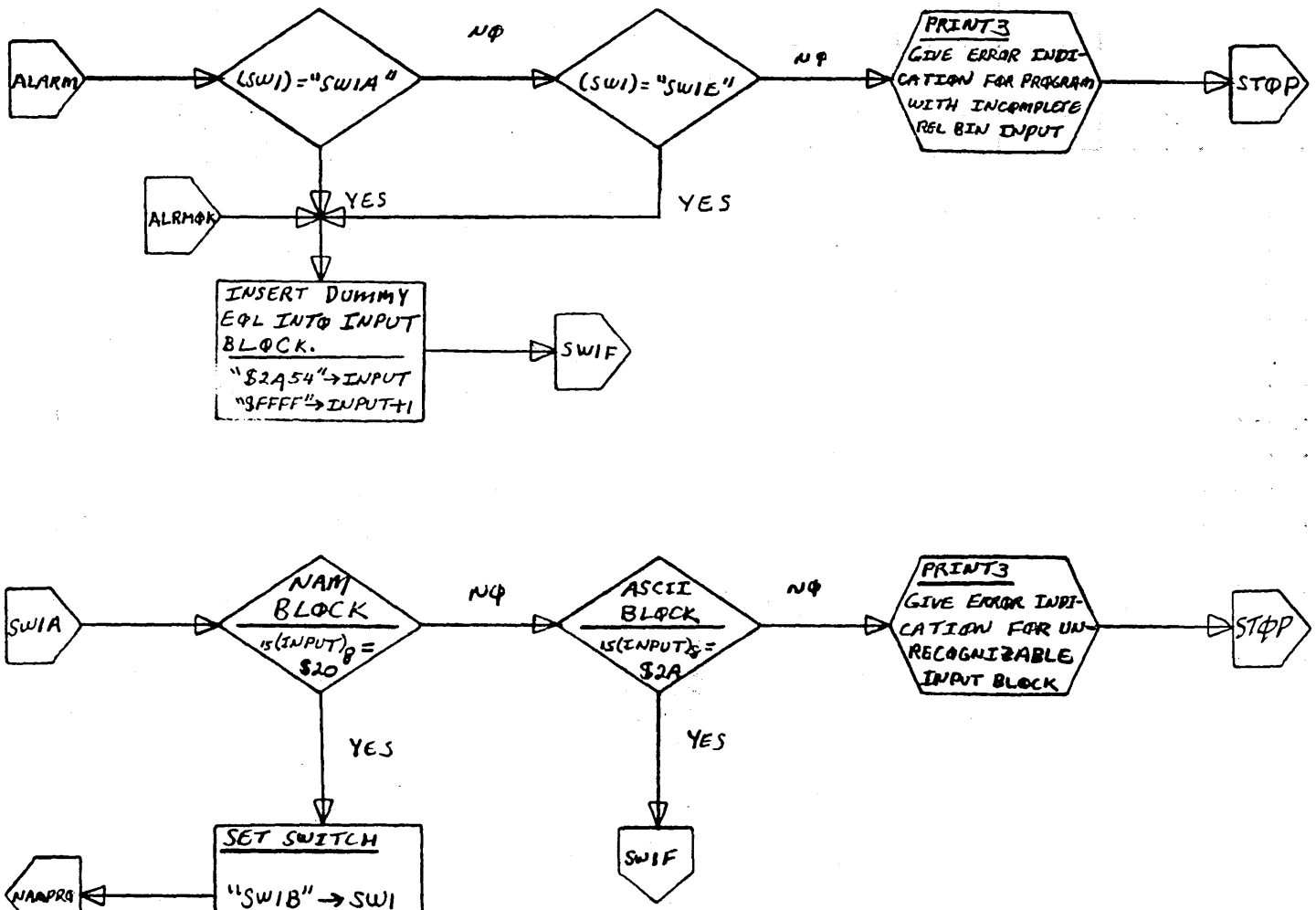


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1145 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	DPRADD PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T Gordon DATE 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

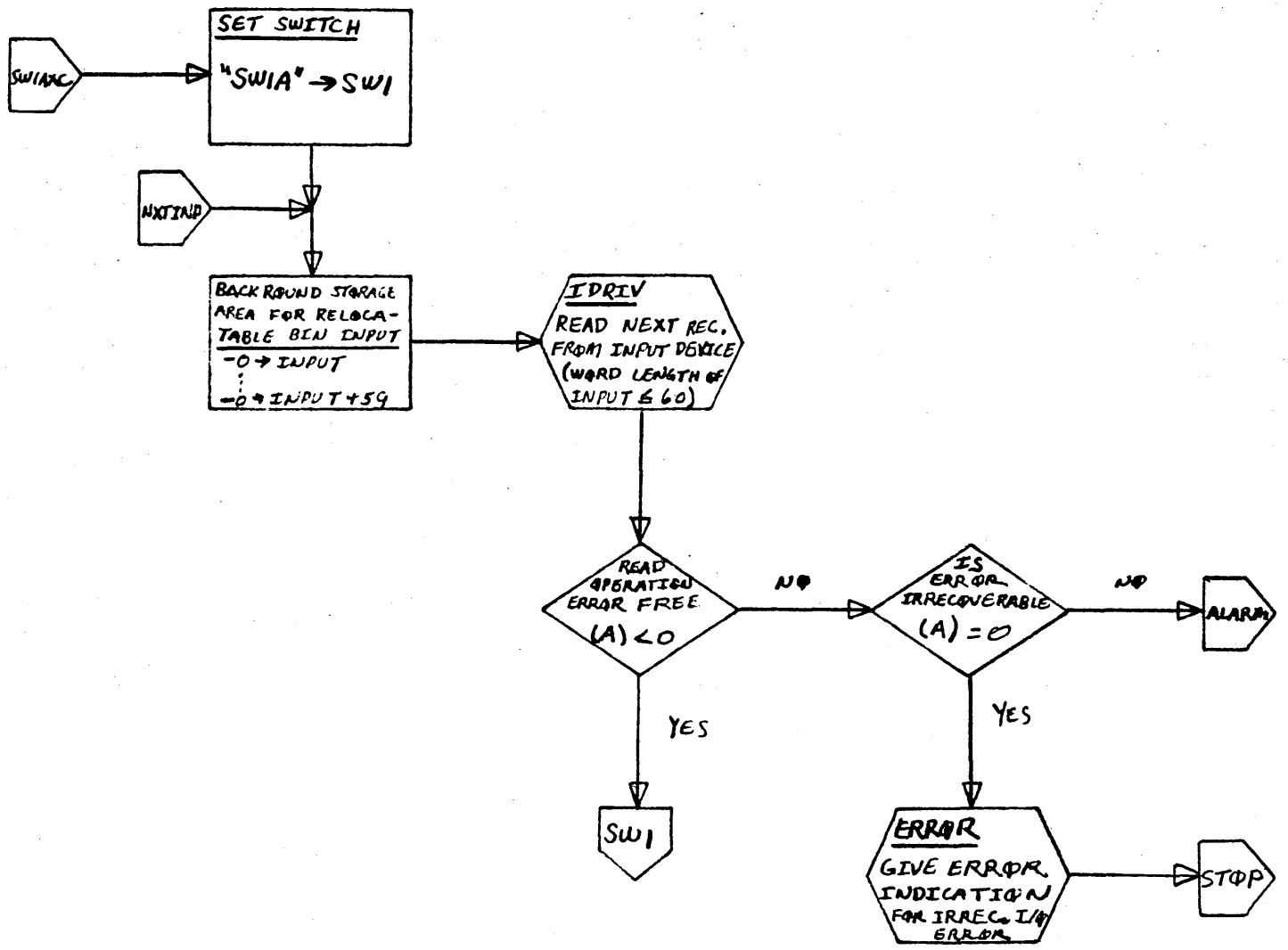
39.215



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: INS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1700 OPERATING SYSTEM		PROJECT MGR.		
FLOWCHART ☐	LOADER	PAGE 3 OF 8	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY: T. GORDON	DATE: 12-6-66	TASK NAME		

MAR 9 1971

39,216

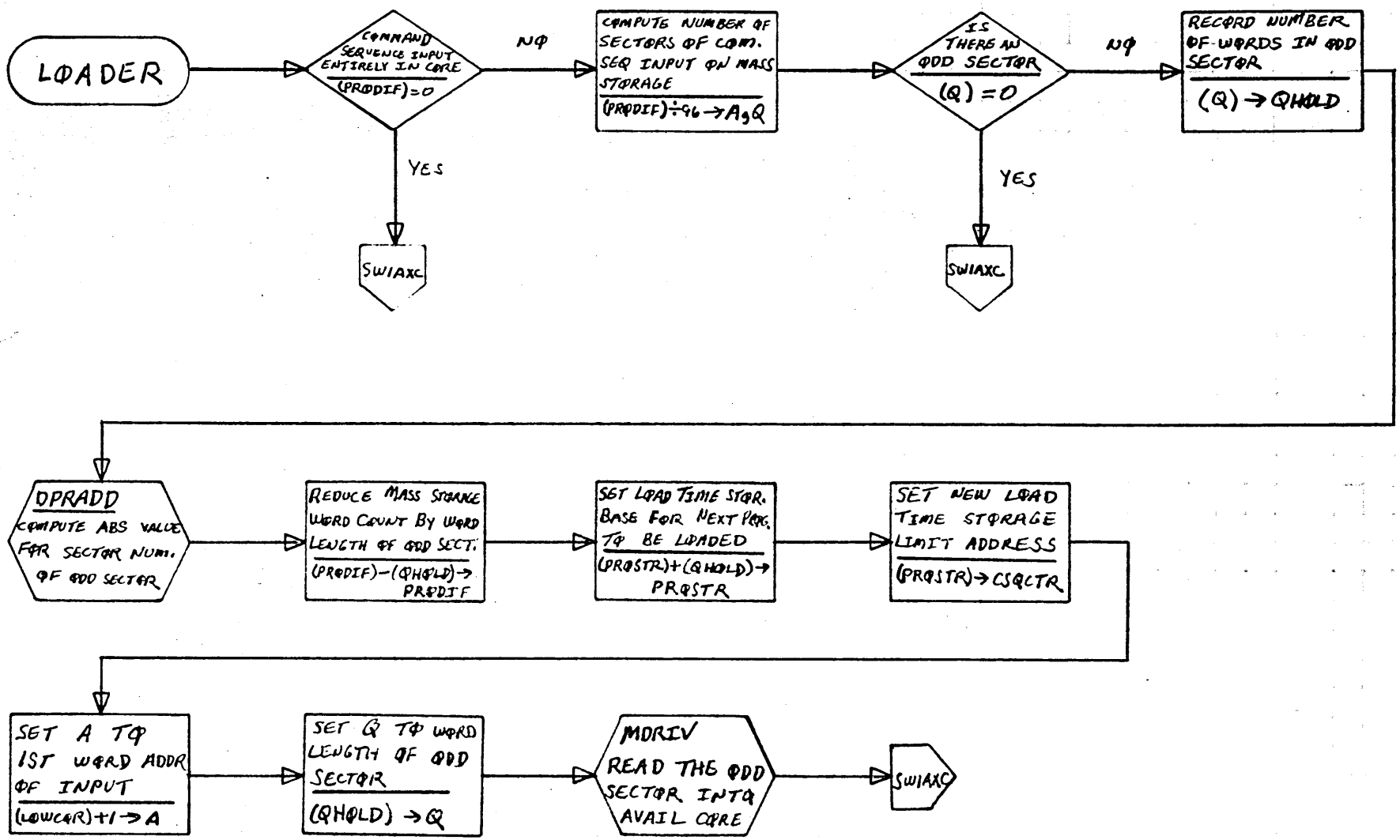


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	LOADER PAGE 2 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6 66	TASK NAME			

AA1385 (FORMERLY A127-1)

MAR 5 1971

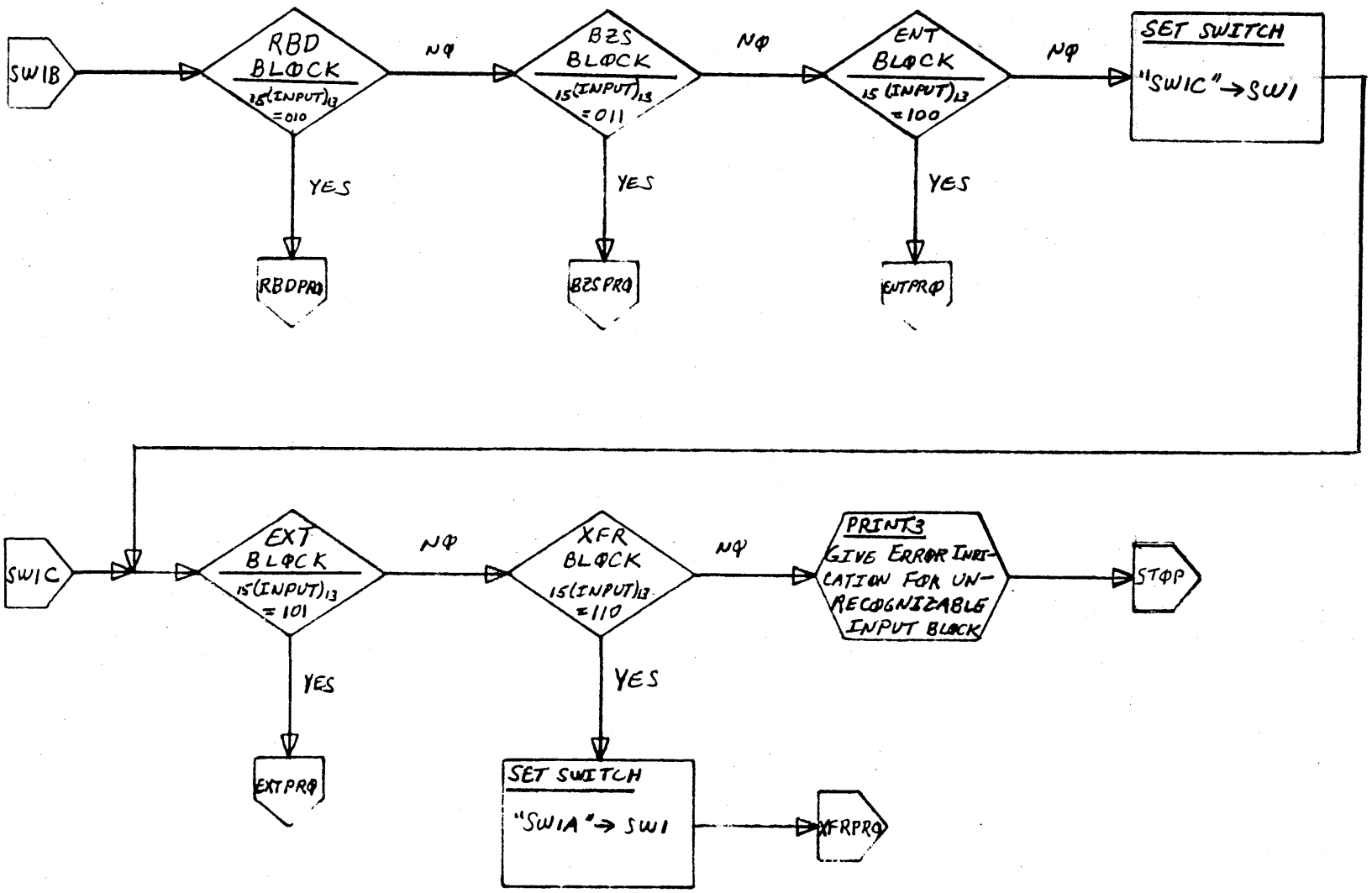
39.217



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM		PROJECT MGR.		
	LOADER	PAGE 1 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. GORDON	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.218

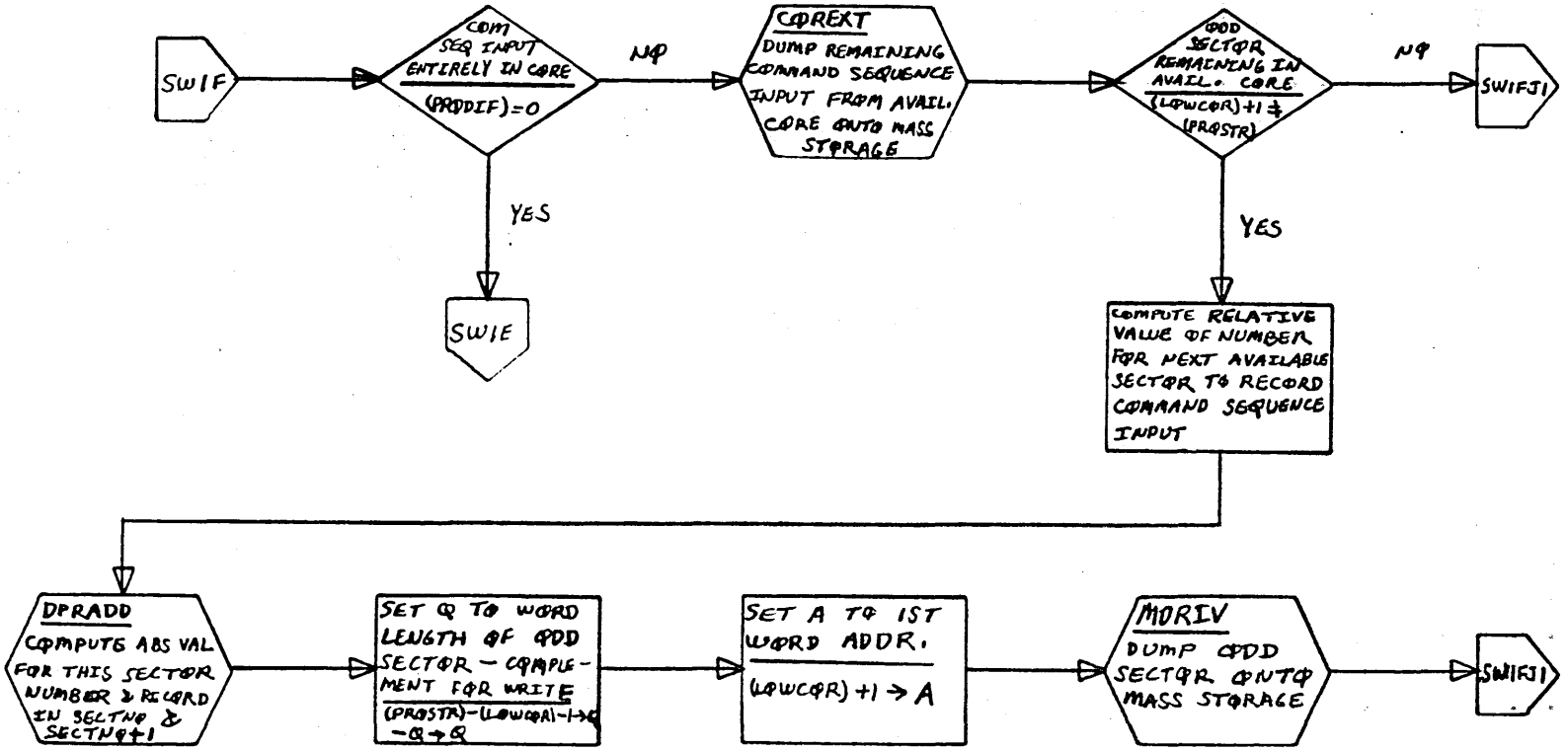


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1700 OPERATING SYSTEM		PROJECT MGR.			
FLOWCHART ☐	LOADER	PAGE 4 OF 8	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY: T. GORDON	DATE: 12-6-66	TASK NAME			

AA1385 (FORME A127-1)

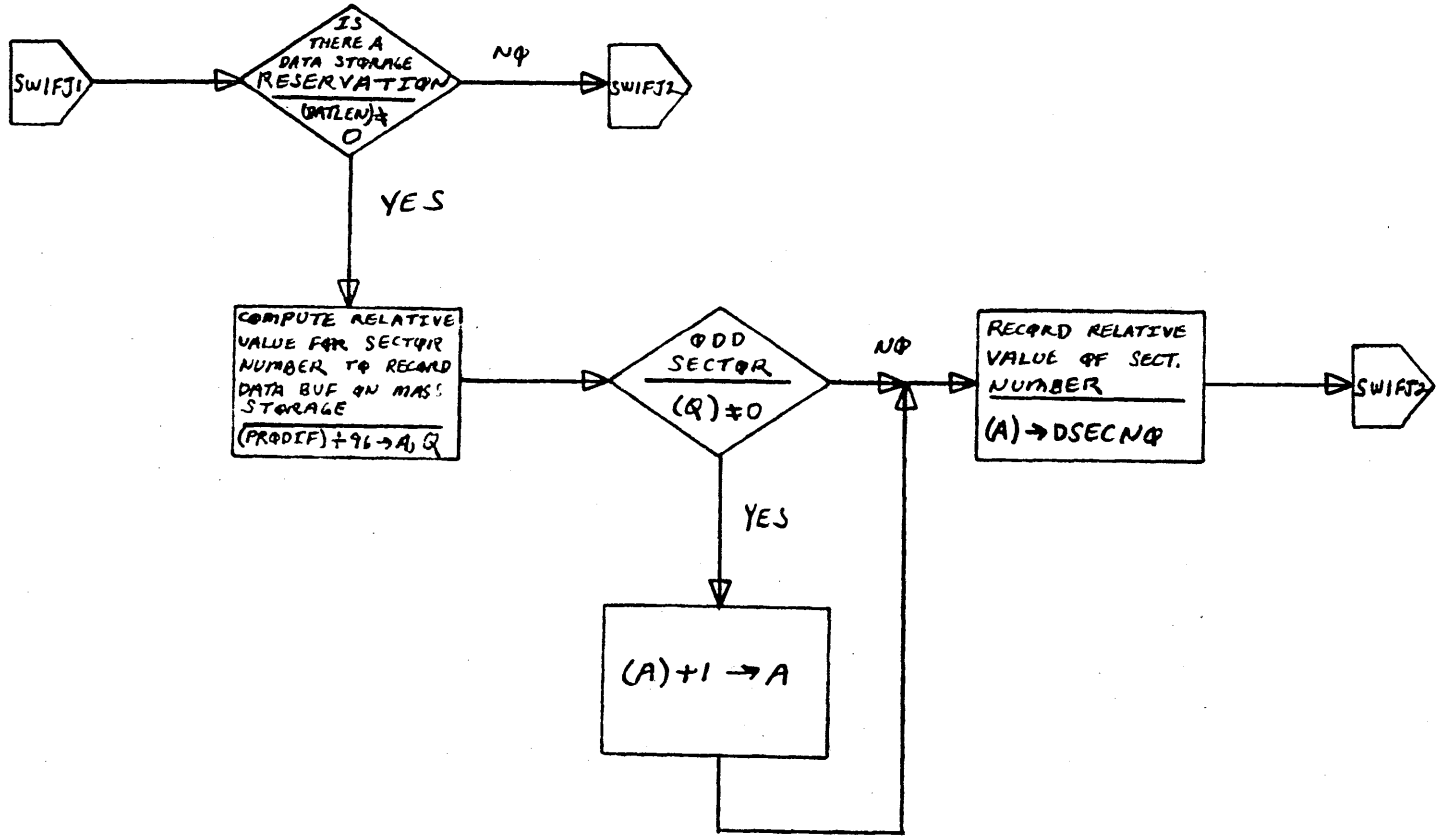
MAR 5 1971

39219



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	LOADER PAGE 5 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

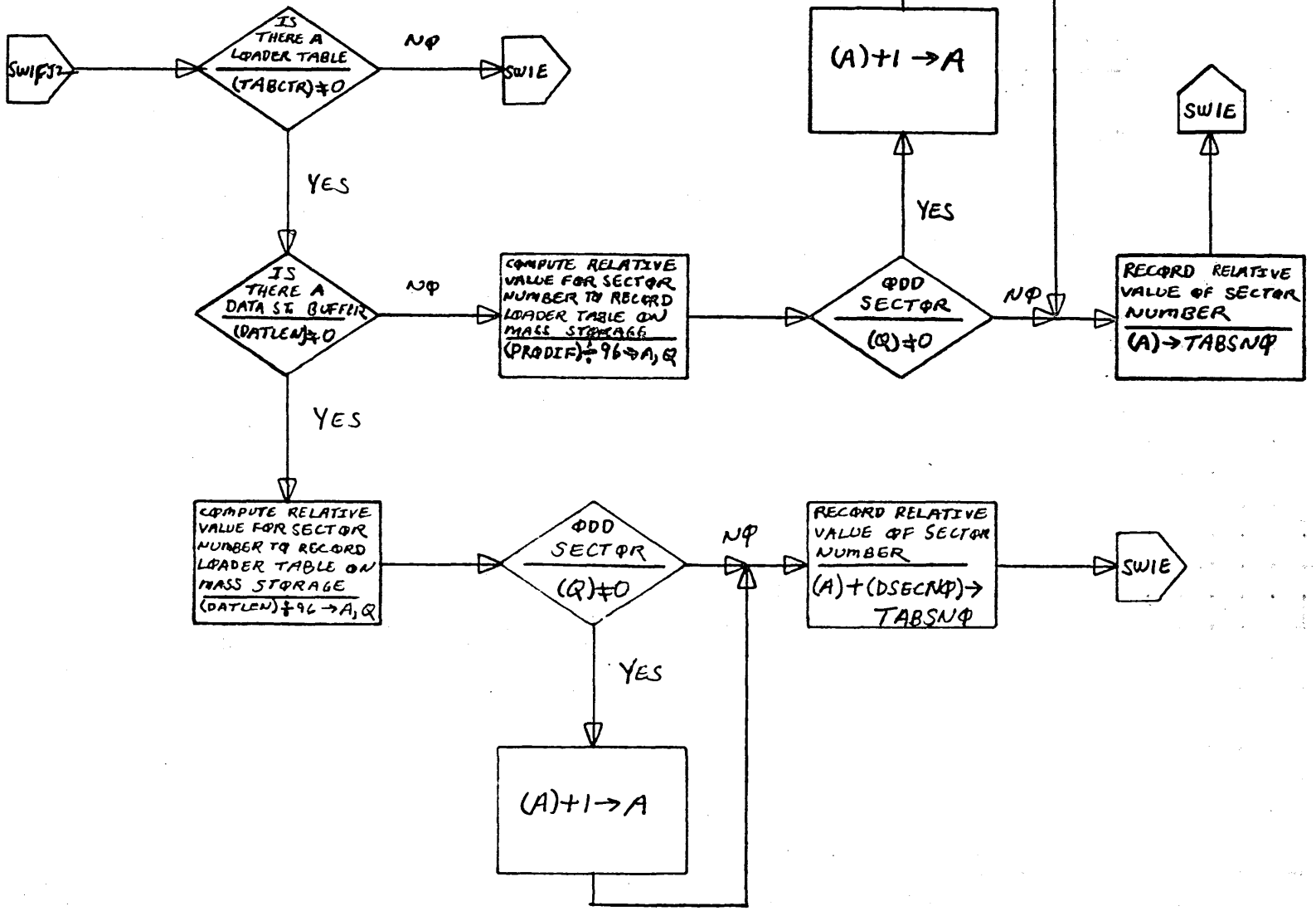
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	LOADER PAGE 6 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

MAR 5 1971

39.221

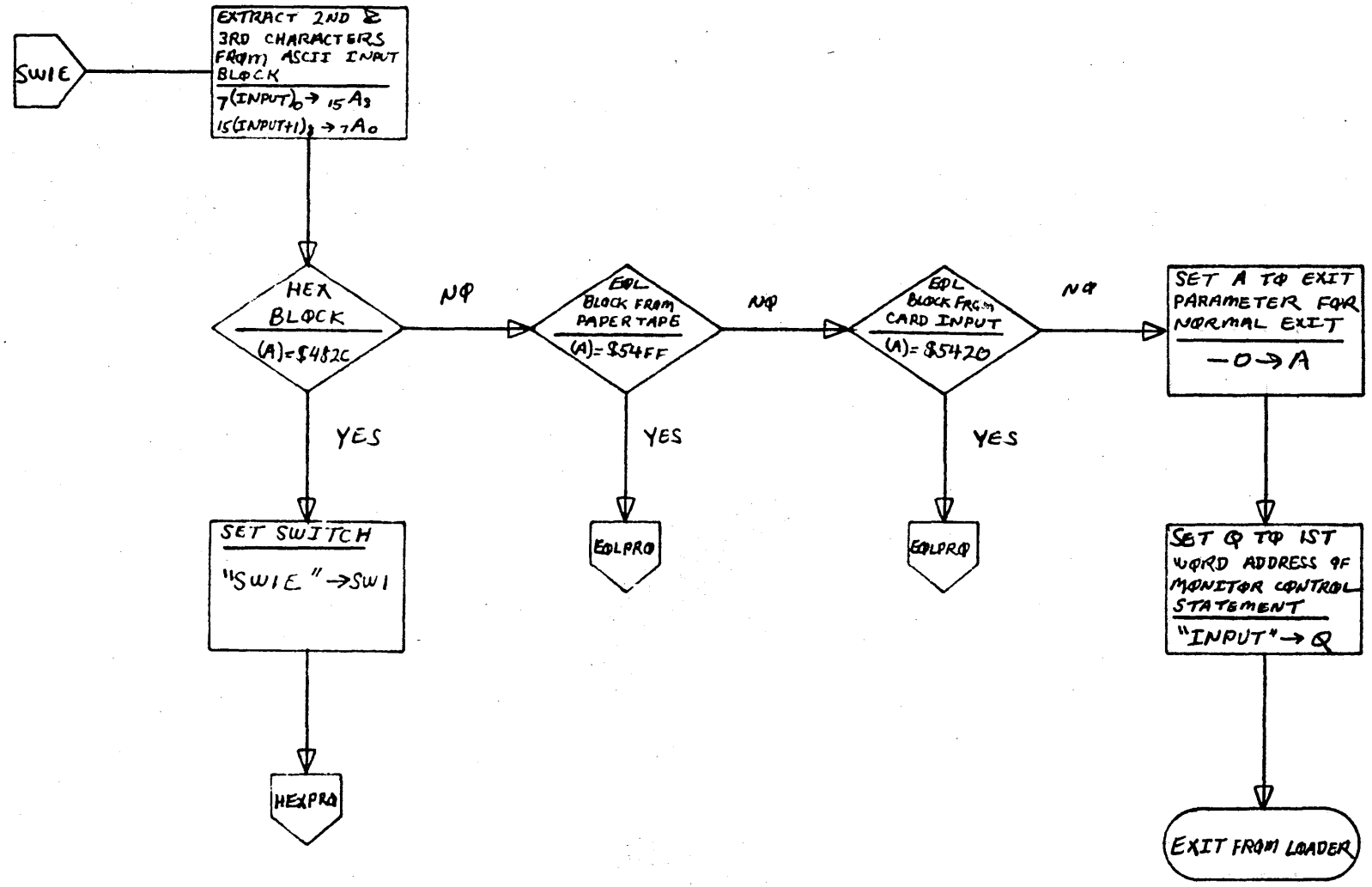


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	LOADER PAGE 4 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

MAR 5 1971

39,222

AA1385 (FORMERLY CA127-1)

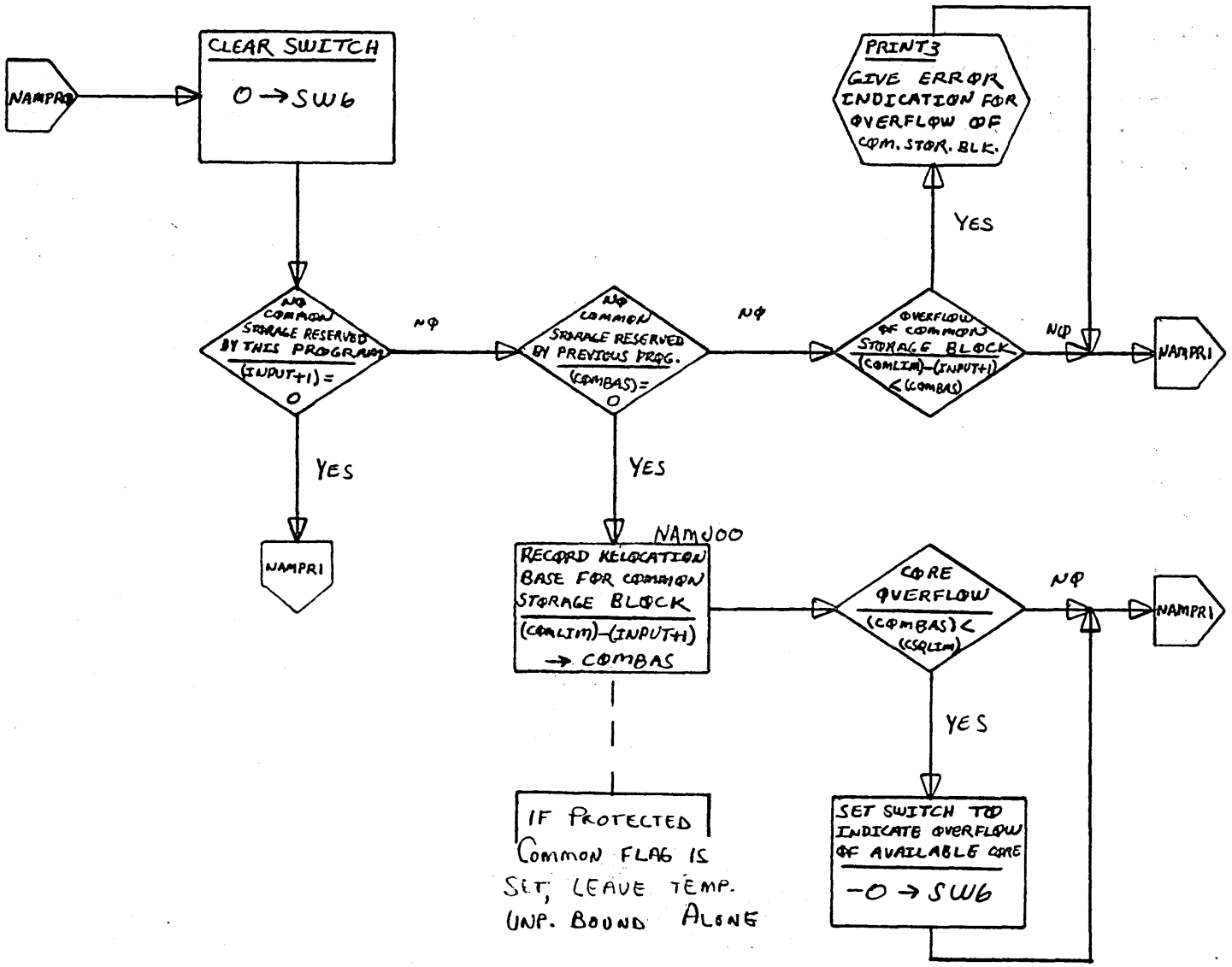


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	LOADER PAGE 8 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

MAR 5 1971

39.223

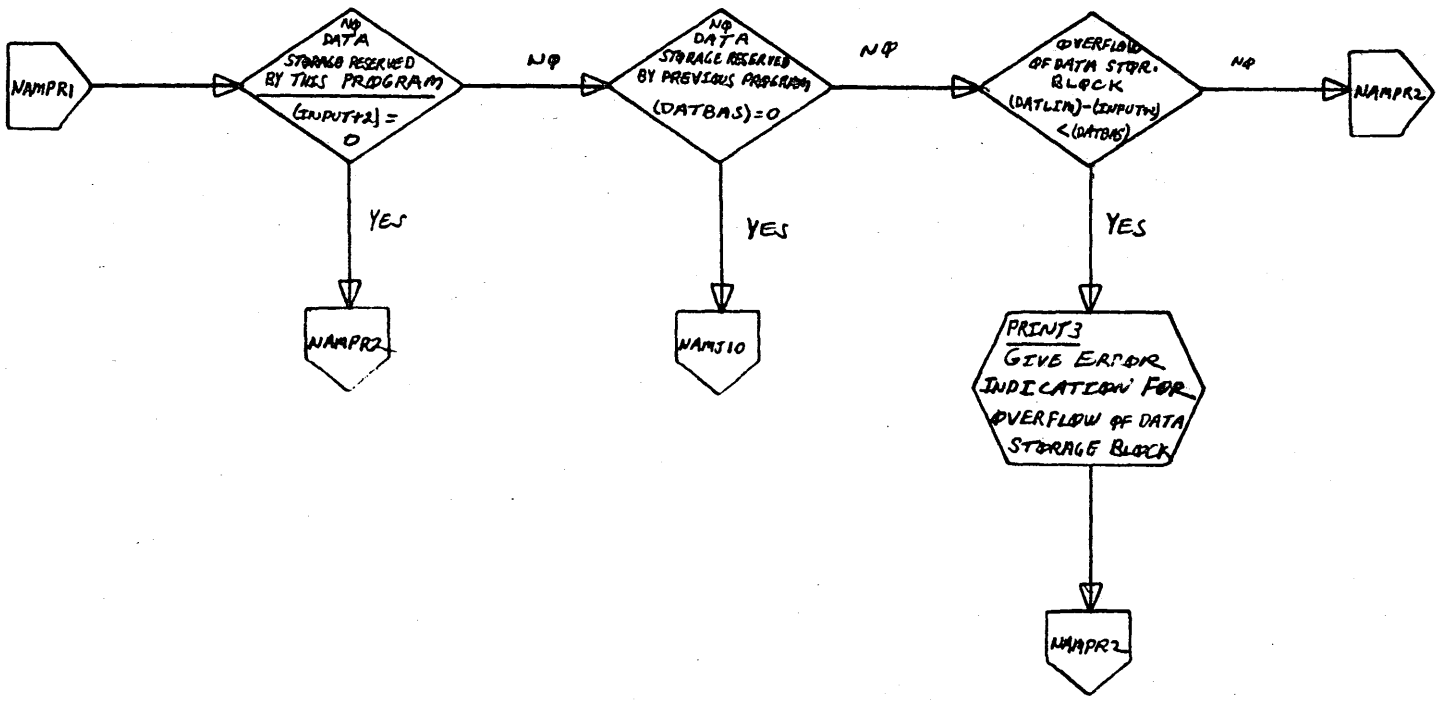
AA1385 (FORME A127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS THS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	NAMPR0	PAGE 1 OF 8	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. Gocon	DATE 12-6-66	TASK NAME		

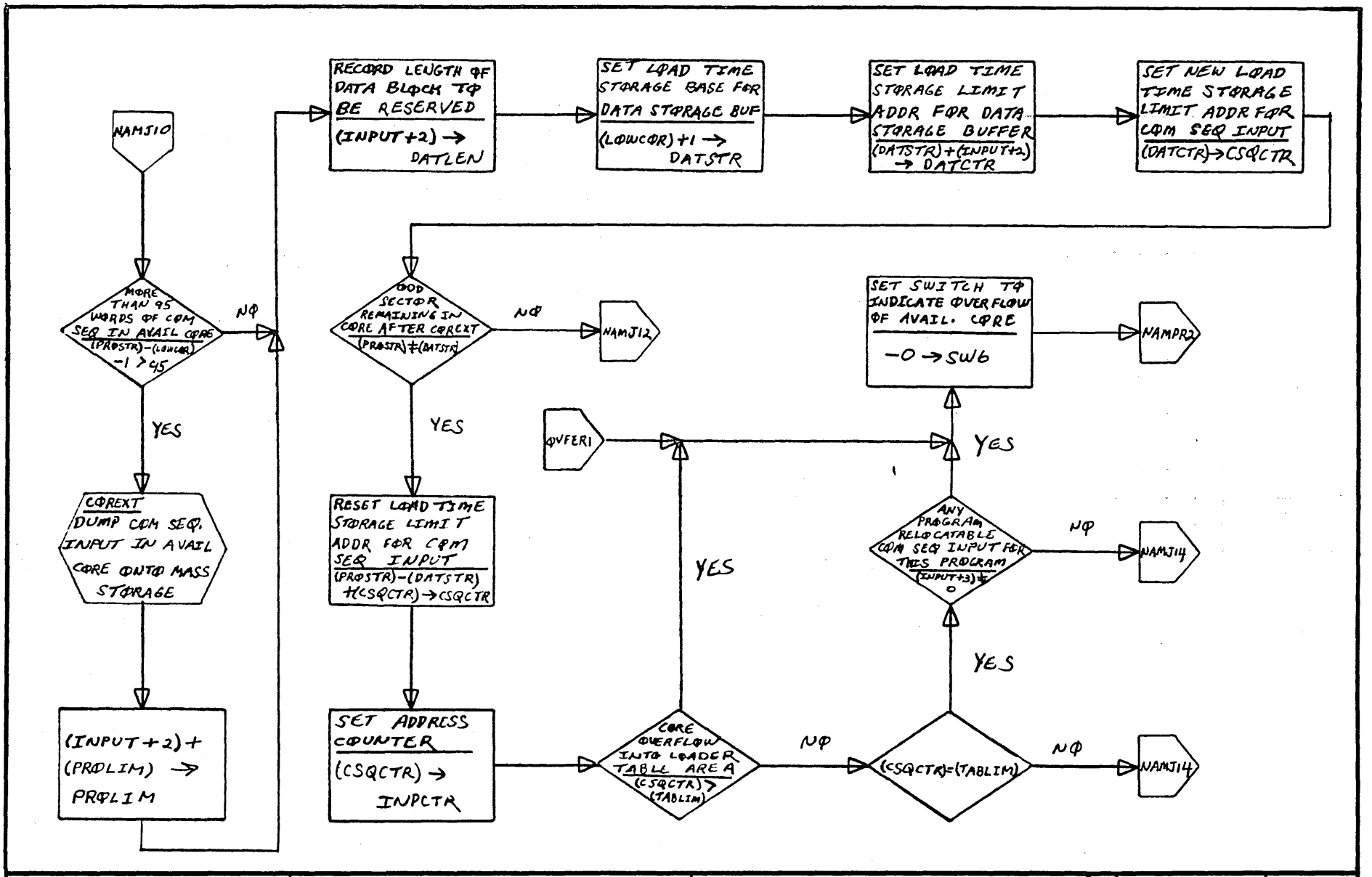
MAR 5 1971

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EHS MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System NAMPR0 PAGE 2 OF 8	PROJECT MGR.			
		PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. Gordon DATE 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

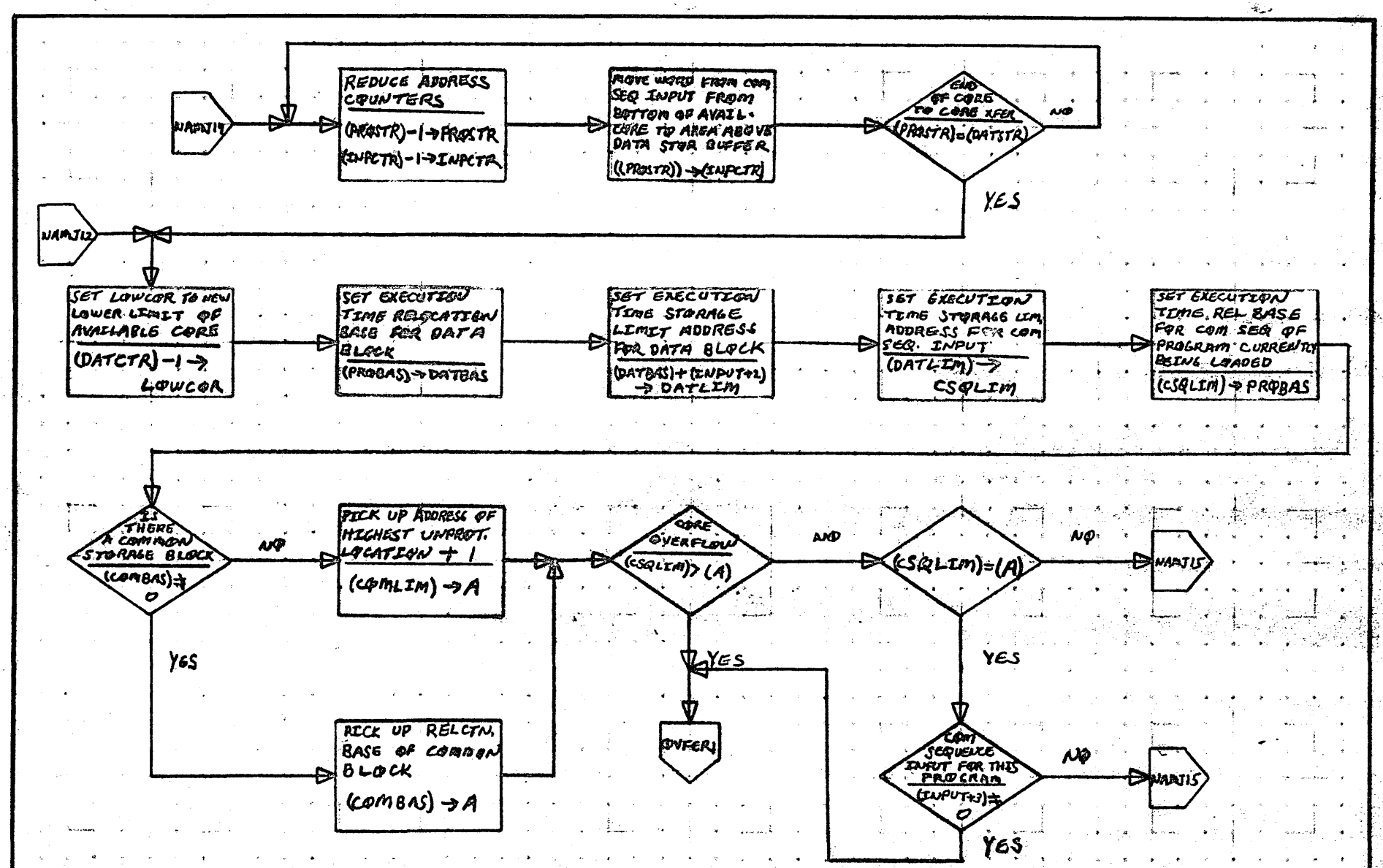


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
	NAMPRO	PAGE 3 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.22.6

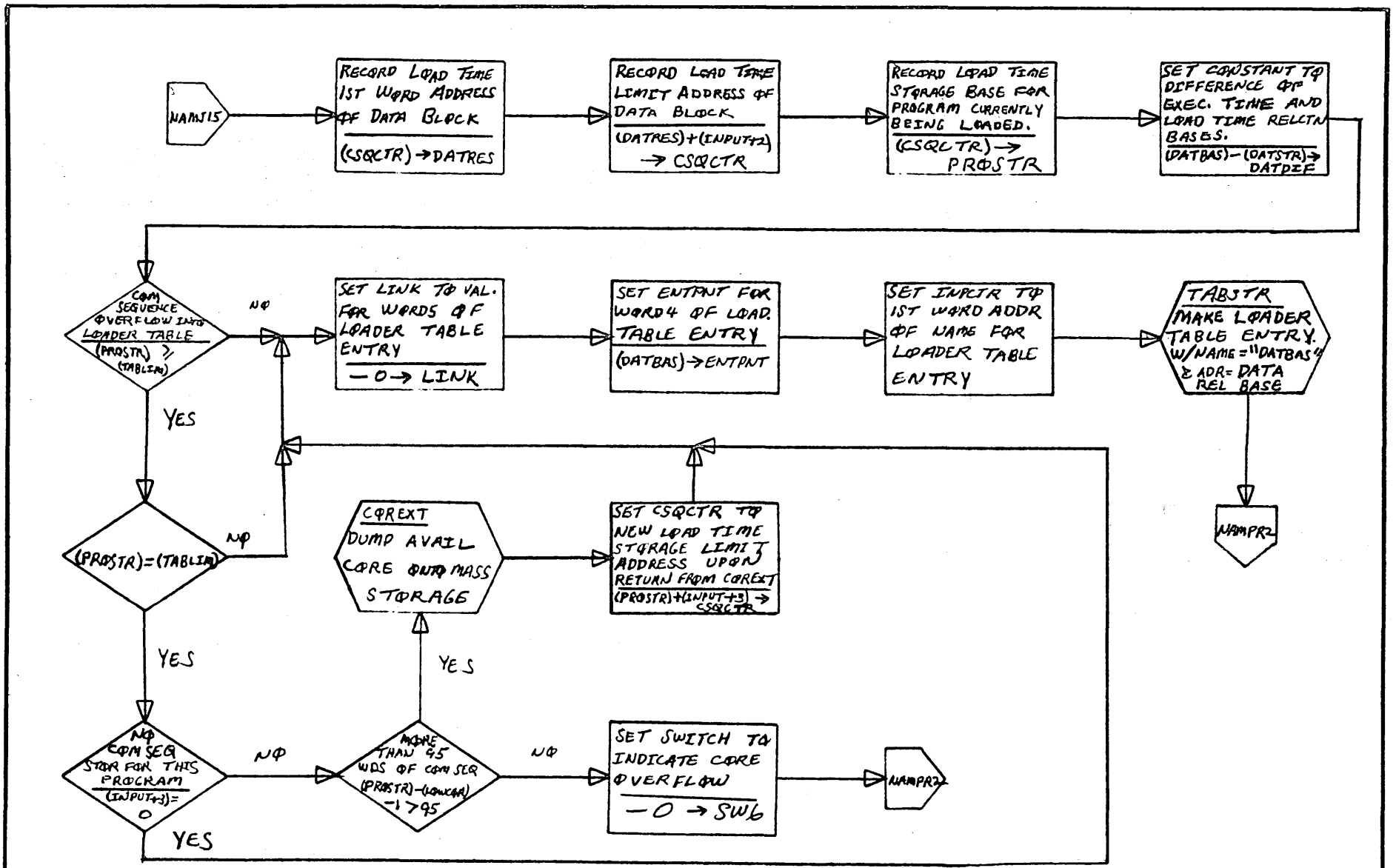
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS THE MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	NAMREQ PAGE 4 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. Gordon DATE 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

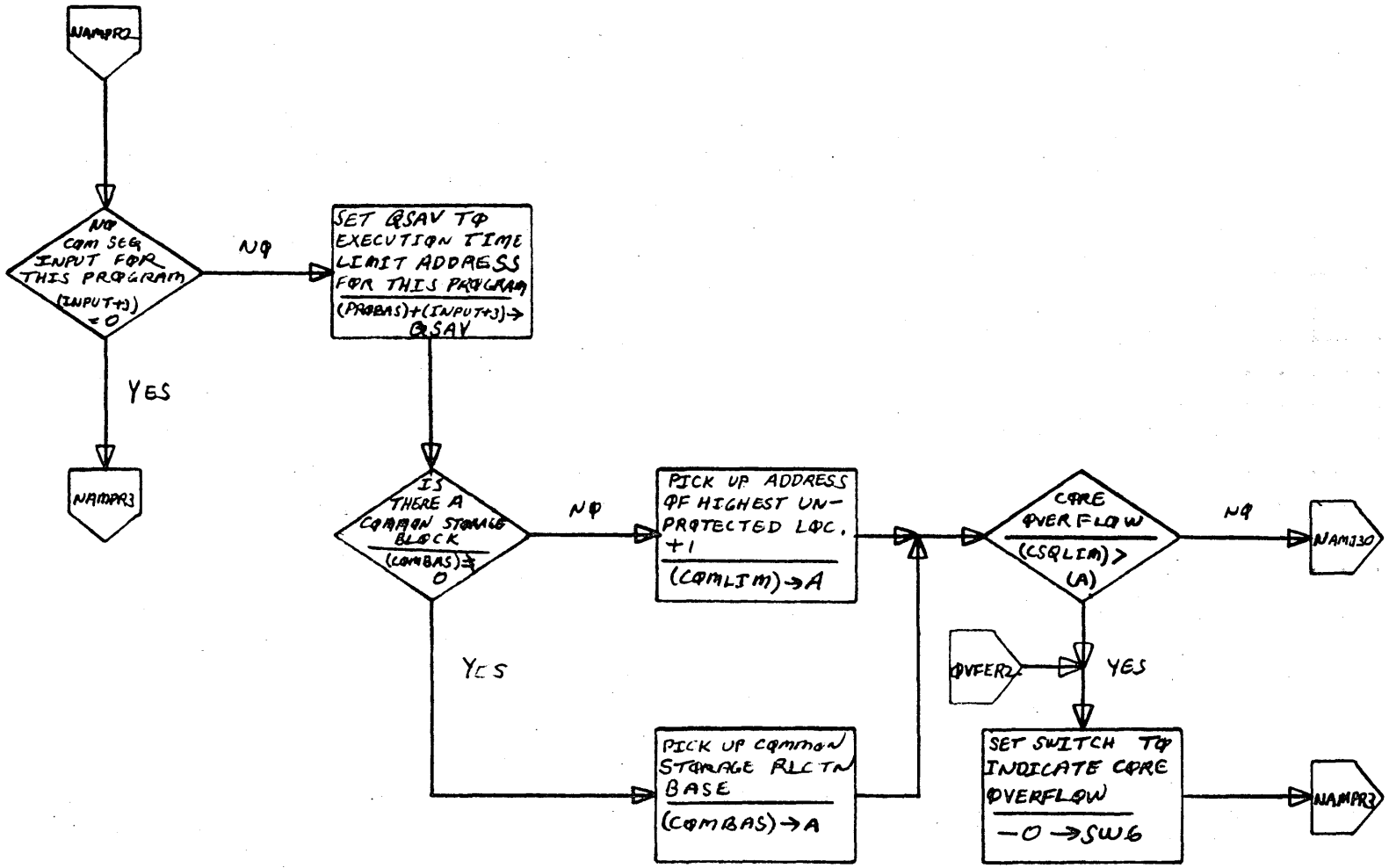


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IHS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
	NAHPR0	PAGE 5 OF 8	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

39.228

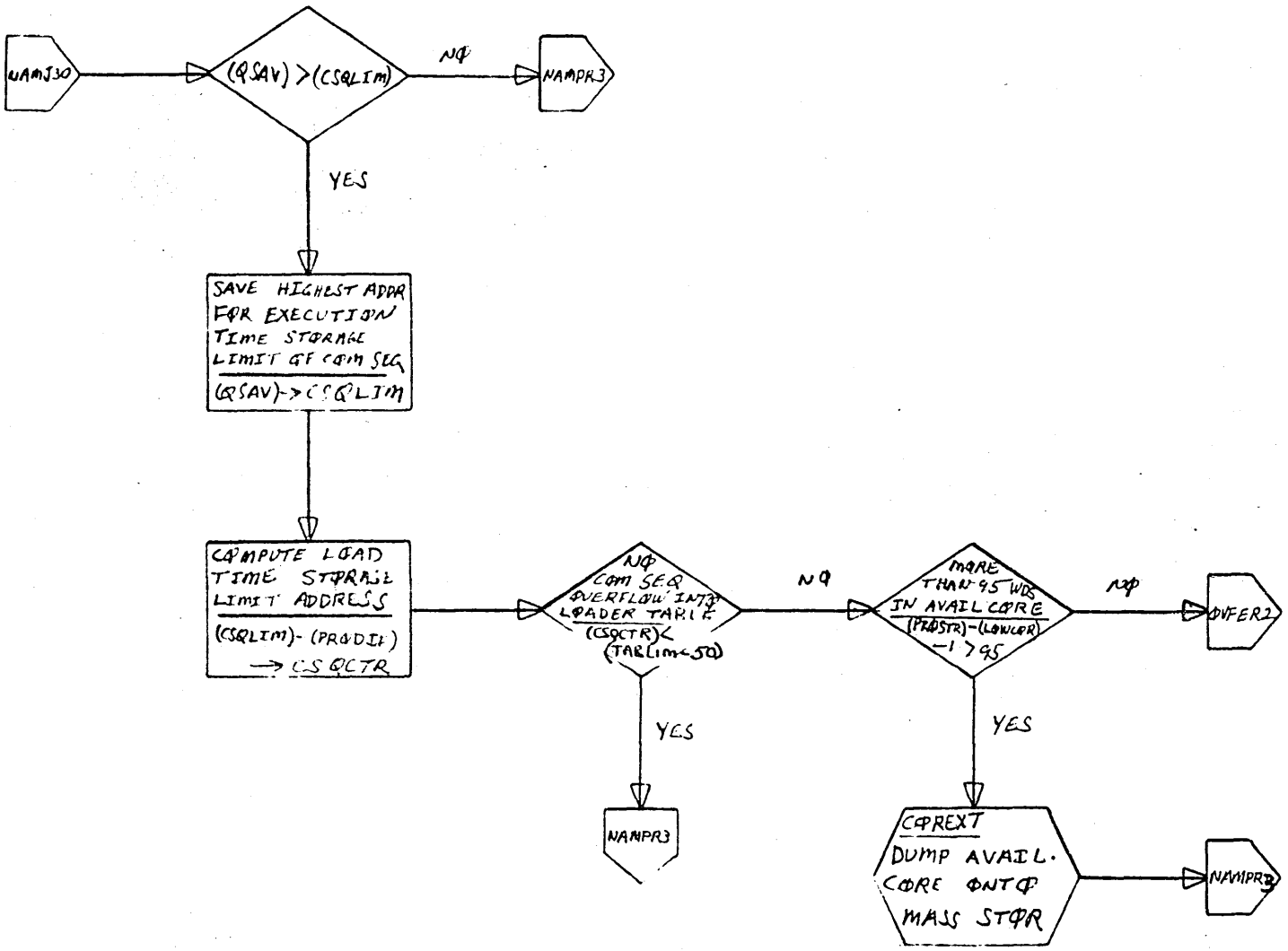


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EHS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
	NAMPR2 PAGE 6 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T Gordon DATE 12-6-66	TASK NAME			

AA1385 (FORME) CA127-1

MAR 5 1971

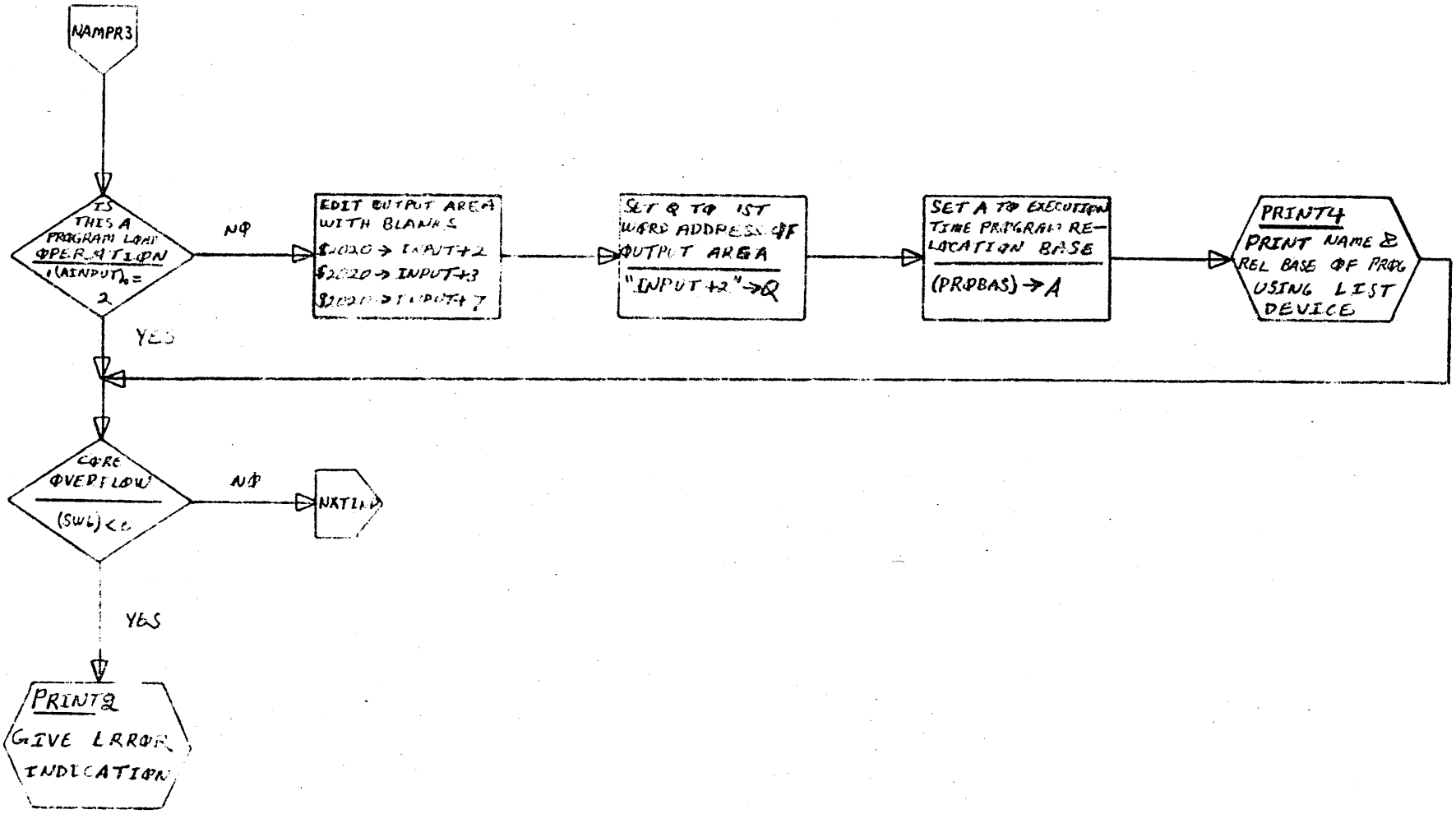
39,229



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS THS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
	NAMPR0 PAGE 7 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 6 1971

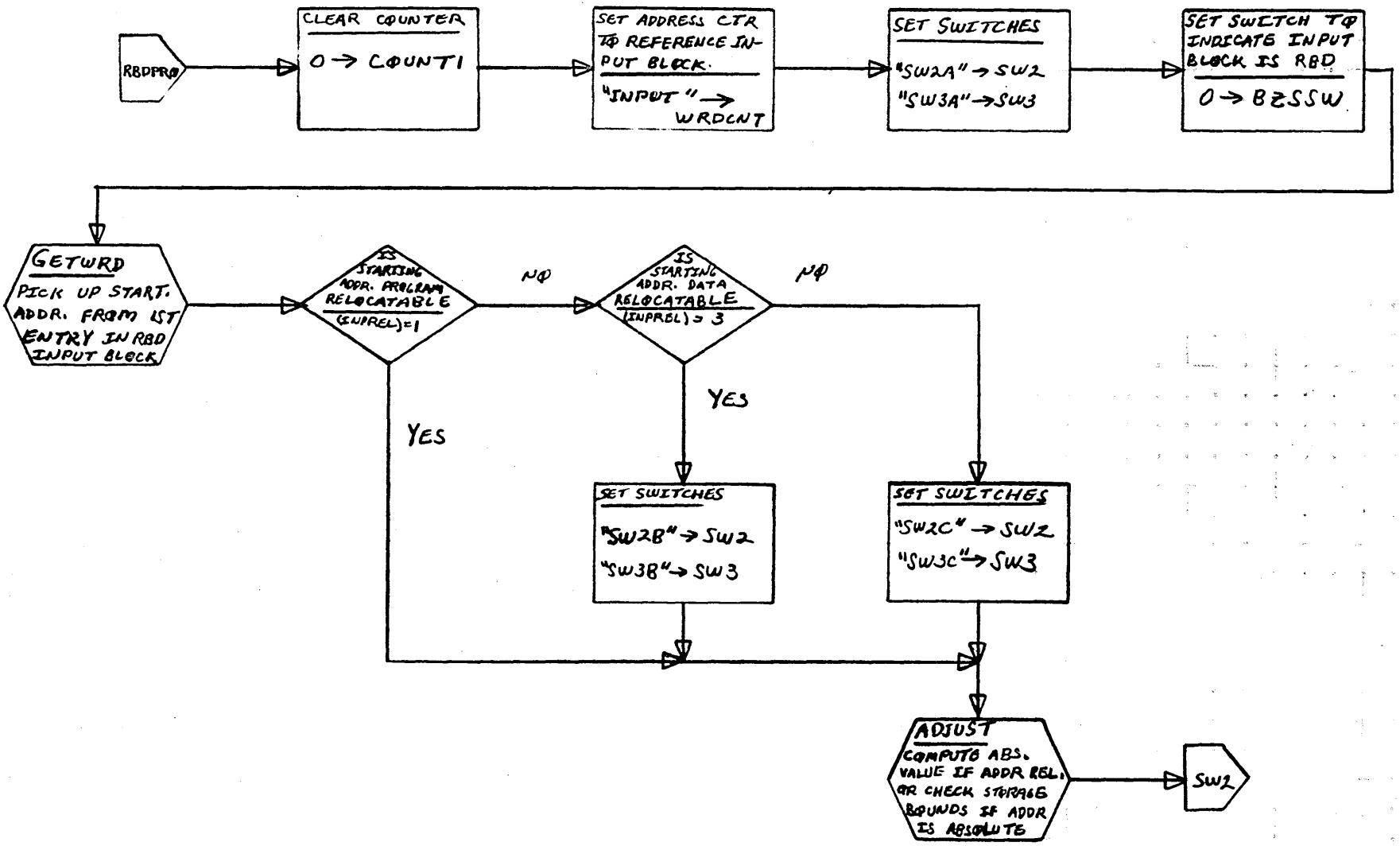
39.230



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH TYPE	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT TITLE 6400 Reading System		PROJECT MGR			
	NUMBER	PAGE 8 OF 8	PROJECT NAME			
		ISSUE DATE	TASK NO			
		DATE	AUTHOR			

MAR 5 1971

39.211

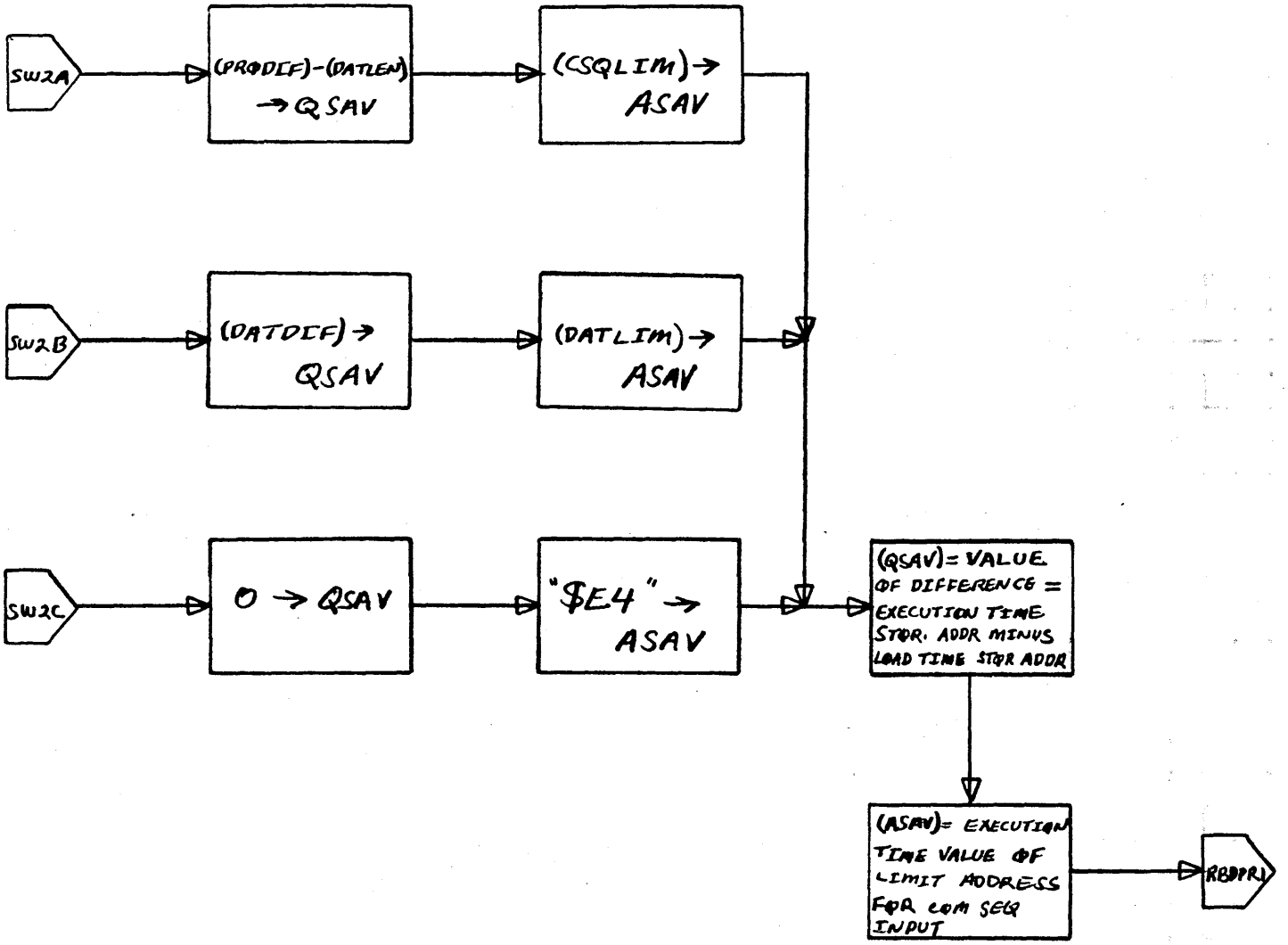


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS THIS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR			
	RBDBZS PAGE 1 OF 12	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY T. GORDON DATE 12-6-66	TASK NAME			

MAR 5 1971

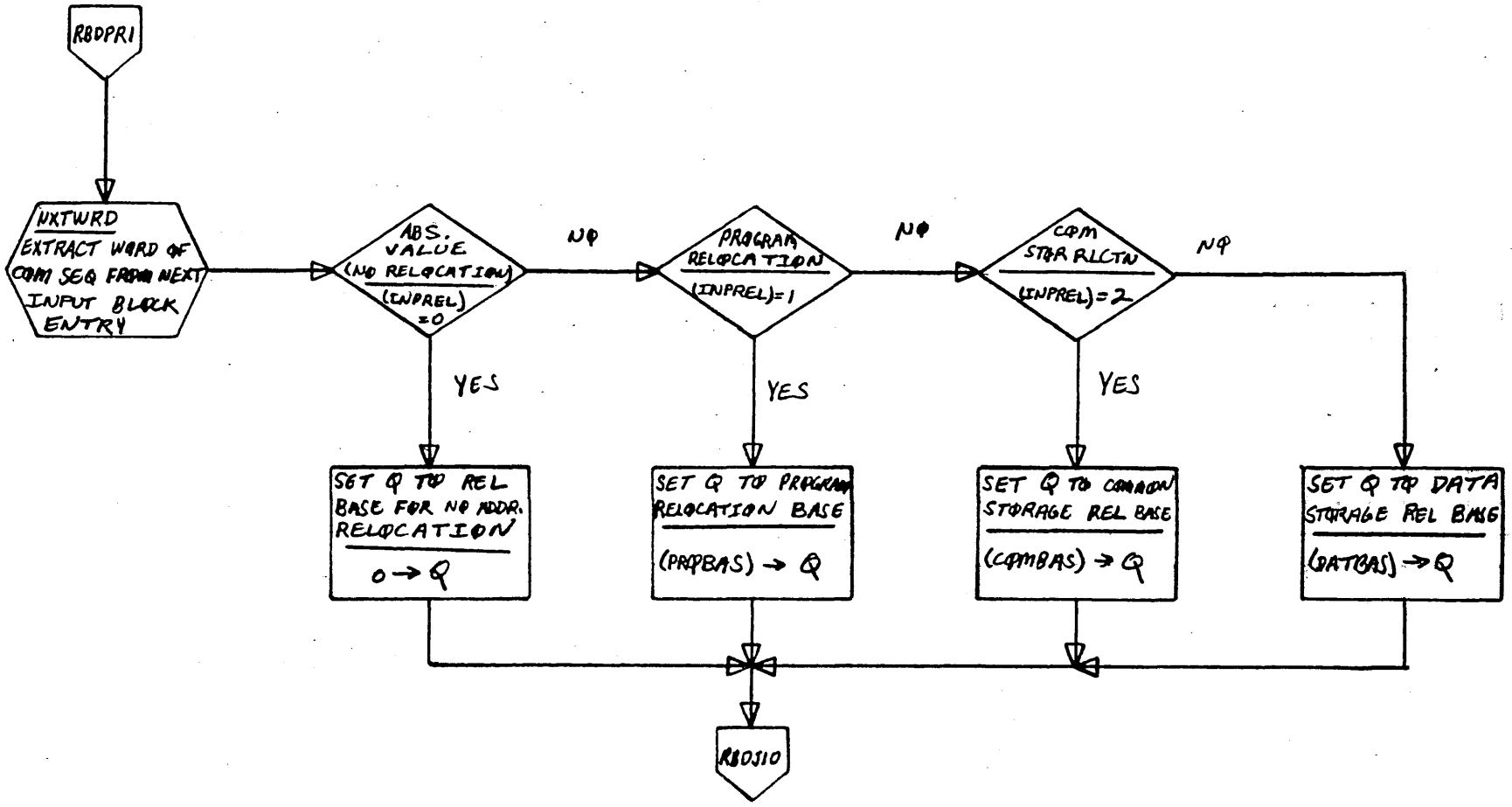
59,232

AA1385 (FORMERLY CA127



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	RBDBZS PAGE 2 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

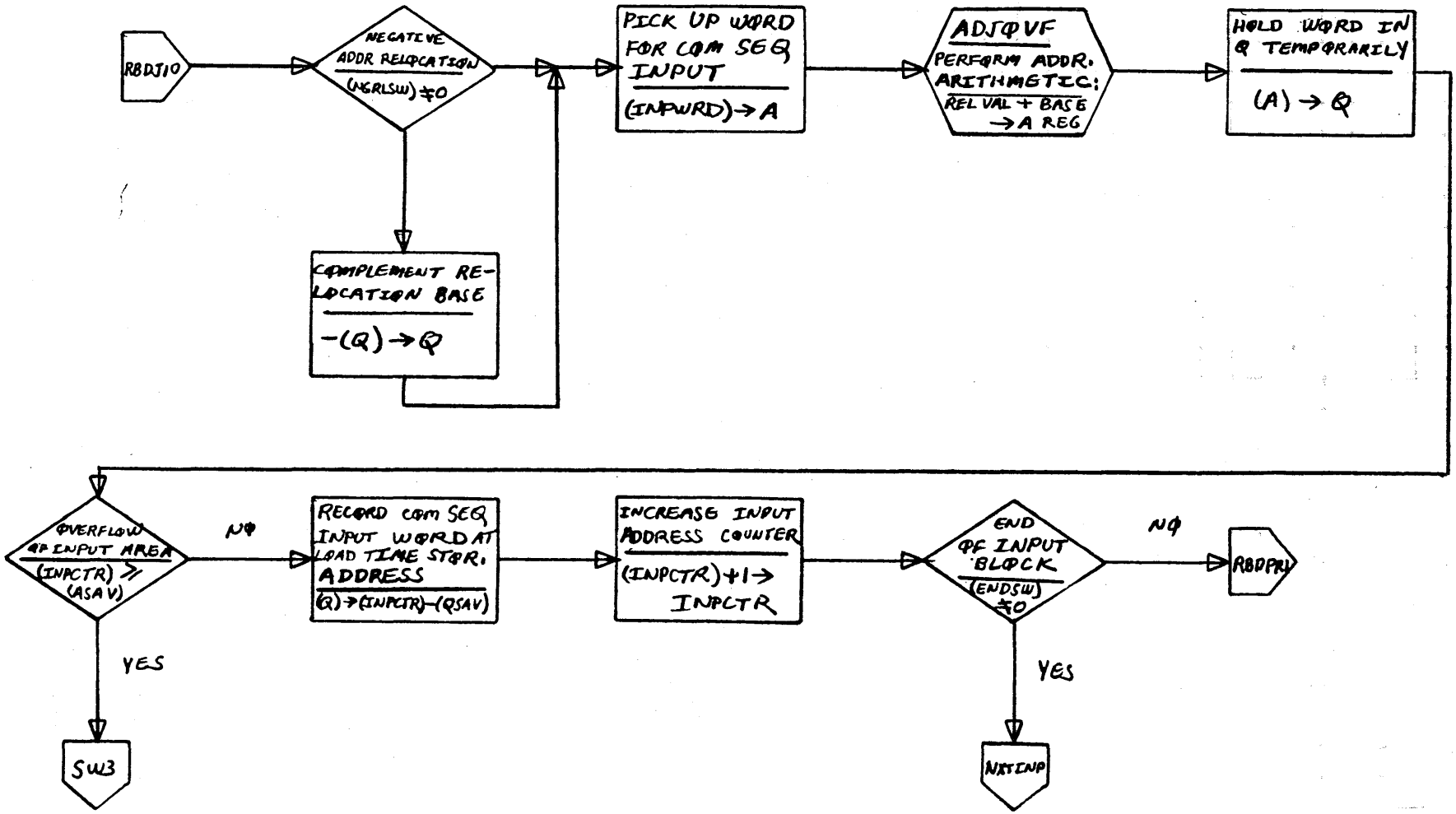
MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	RBDBZS PAGE 3 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

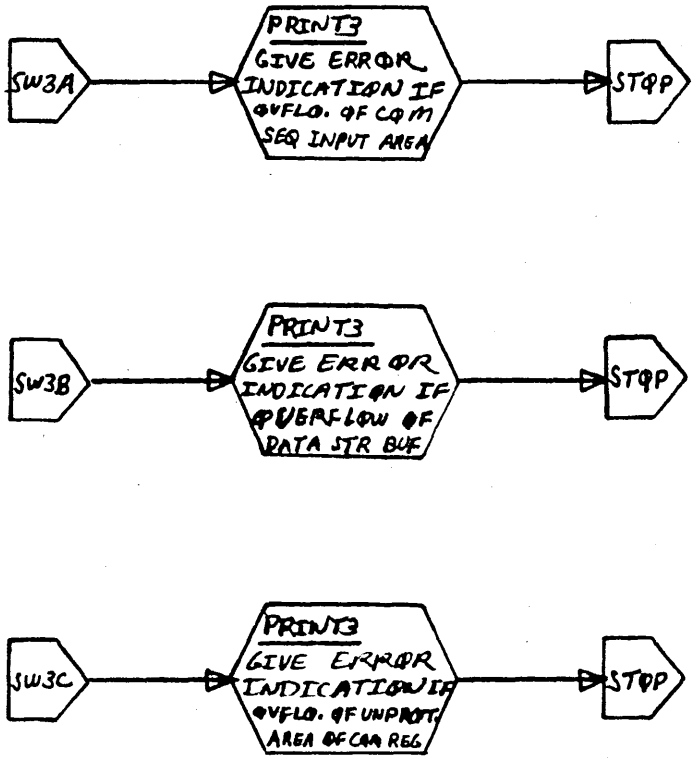
E9.23H



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	RBDBZS PAGE 4 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORME CA127-1)

MAR 5 1971

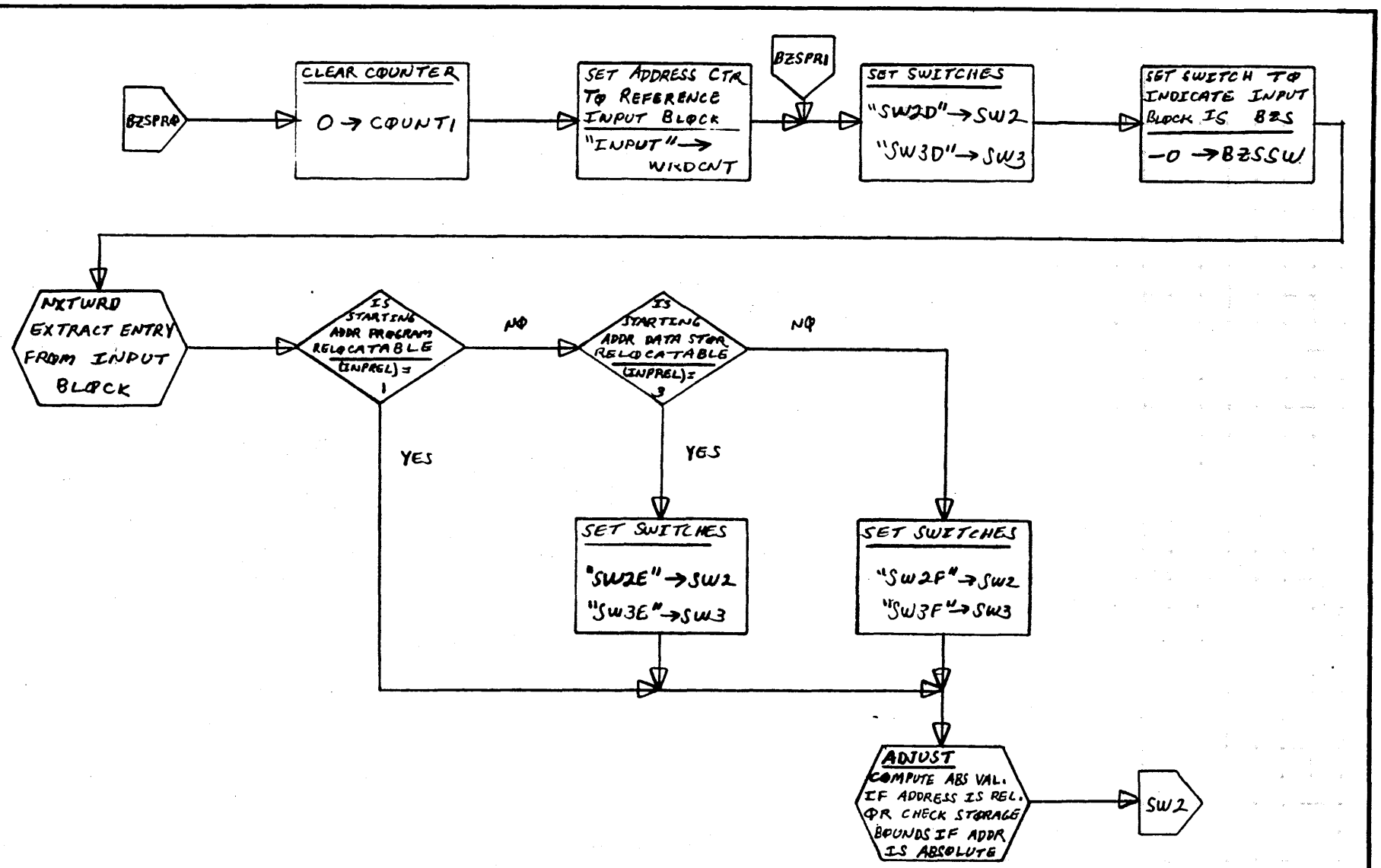


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	RB0825 PAGE 5 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

EB 2.36

AA1385 (FORMERLY CA127-1)

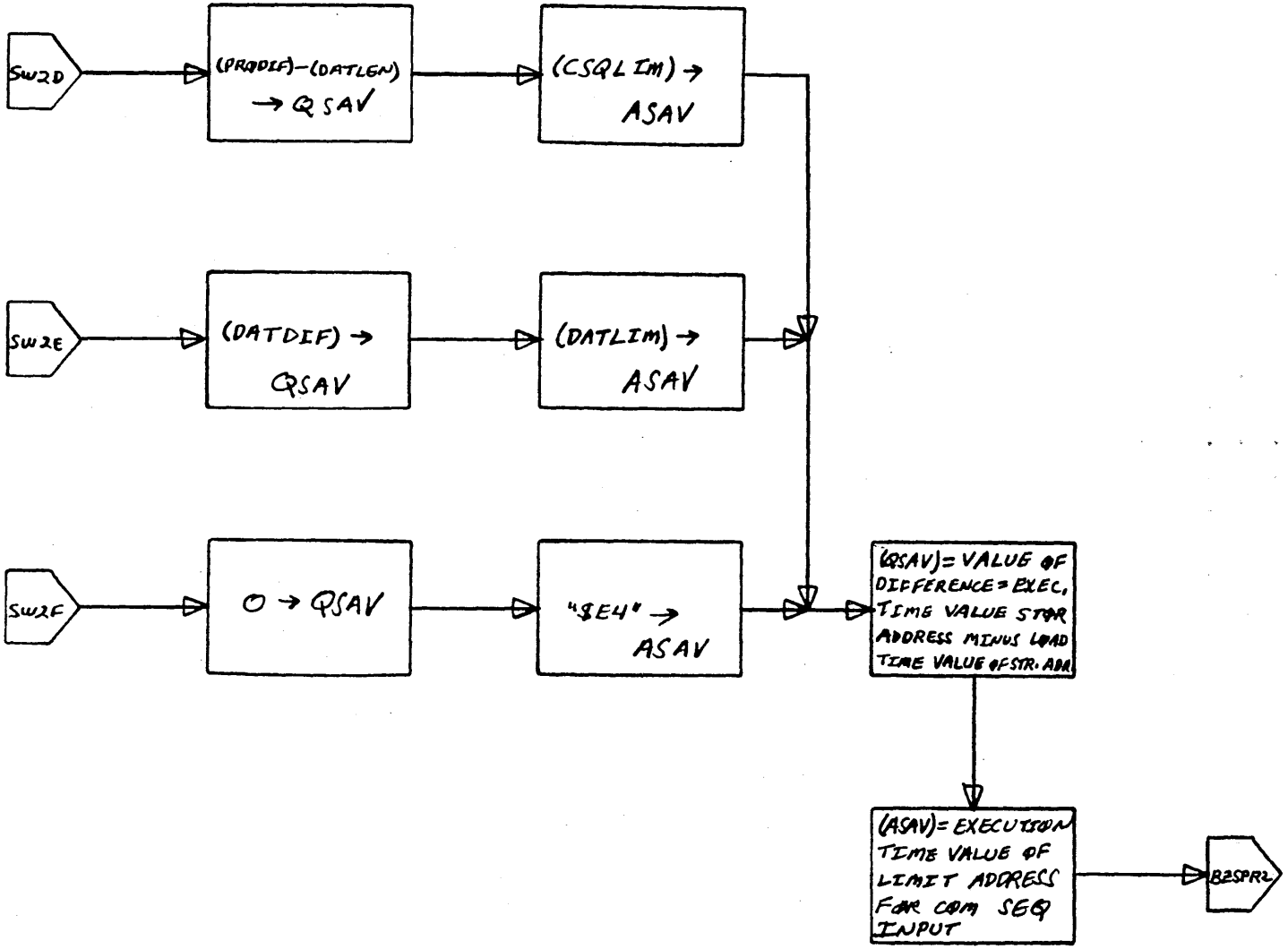


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	RBOBZS PAGE 6 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

59.237

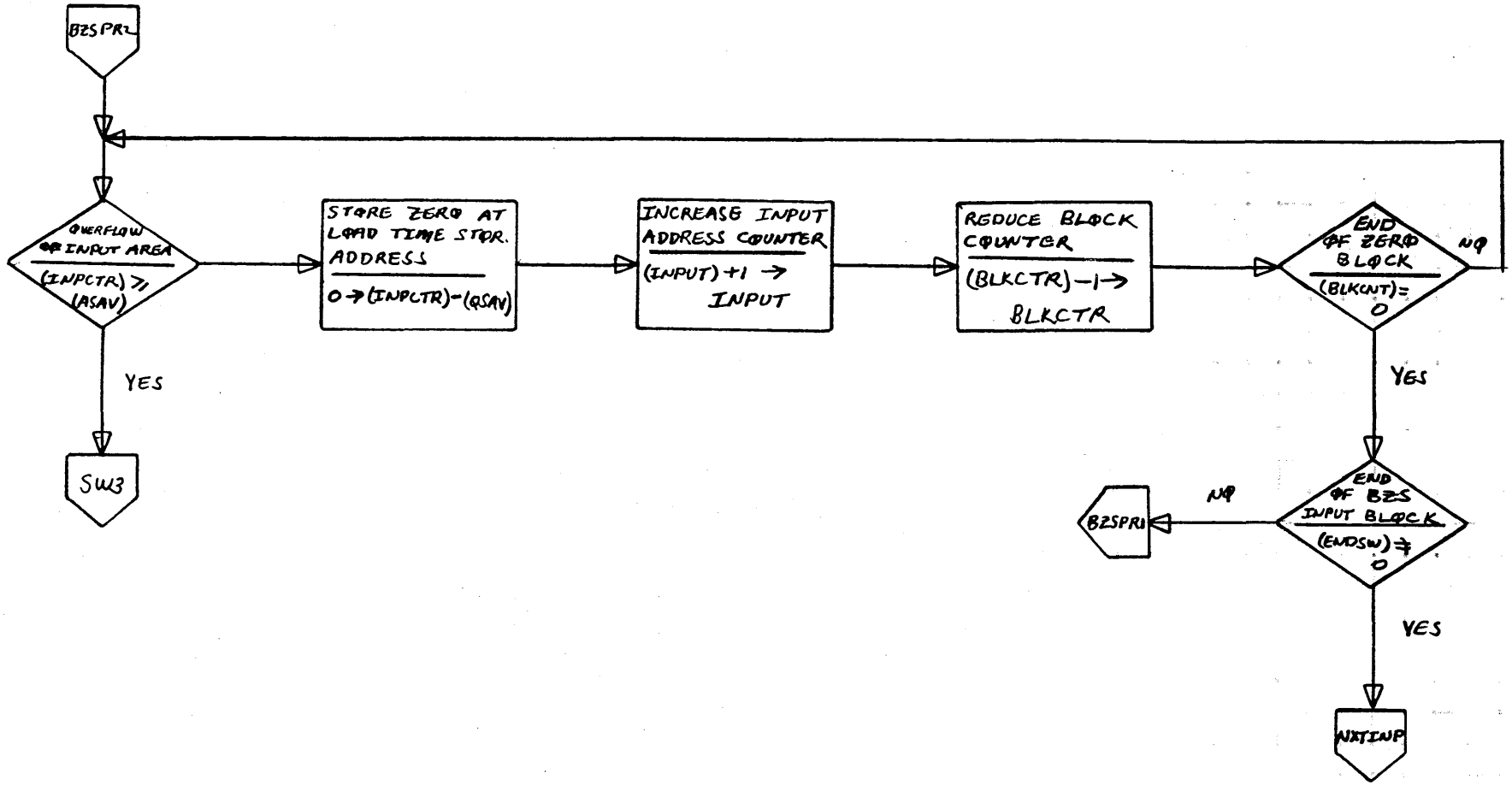
AA1385 (FORMER A127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	RBOBZS PAGE 7 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

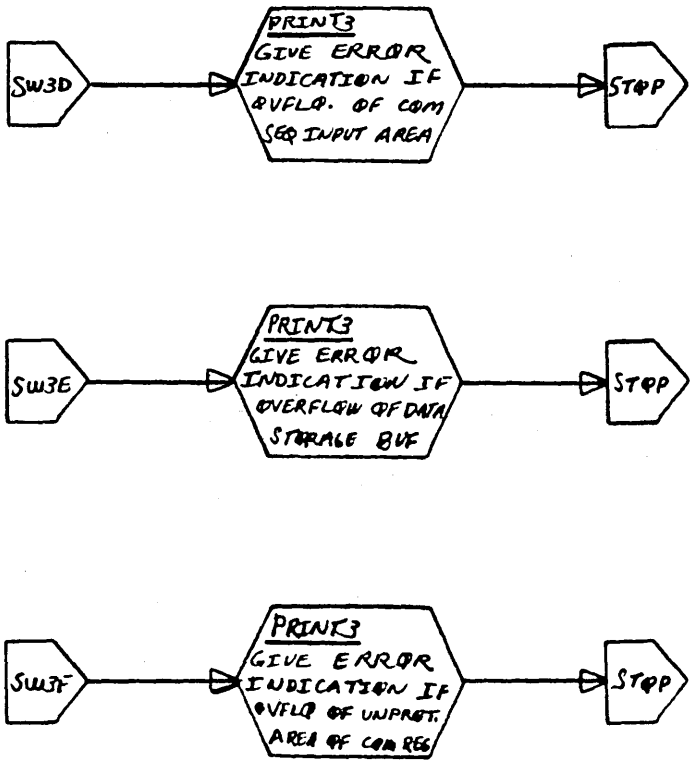
39.238



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR			
FLOWCHART ☐	R80BZS PAGE 8 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.239

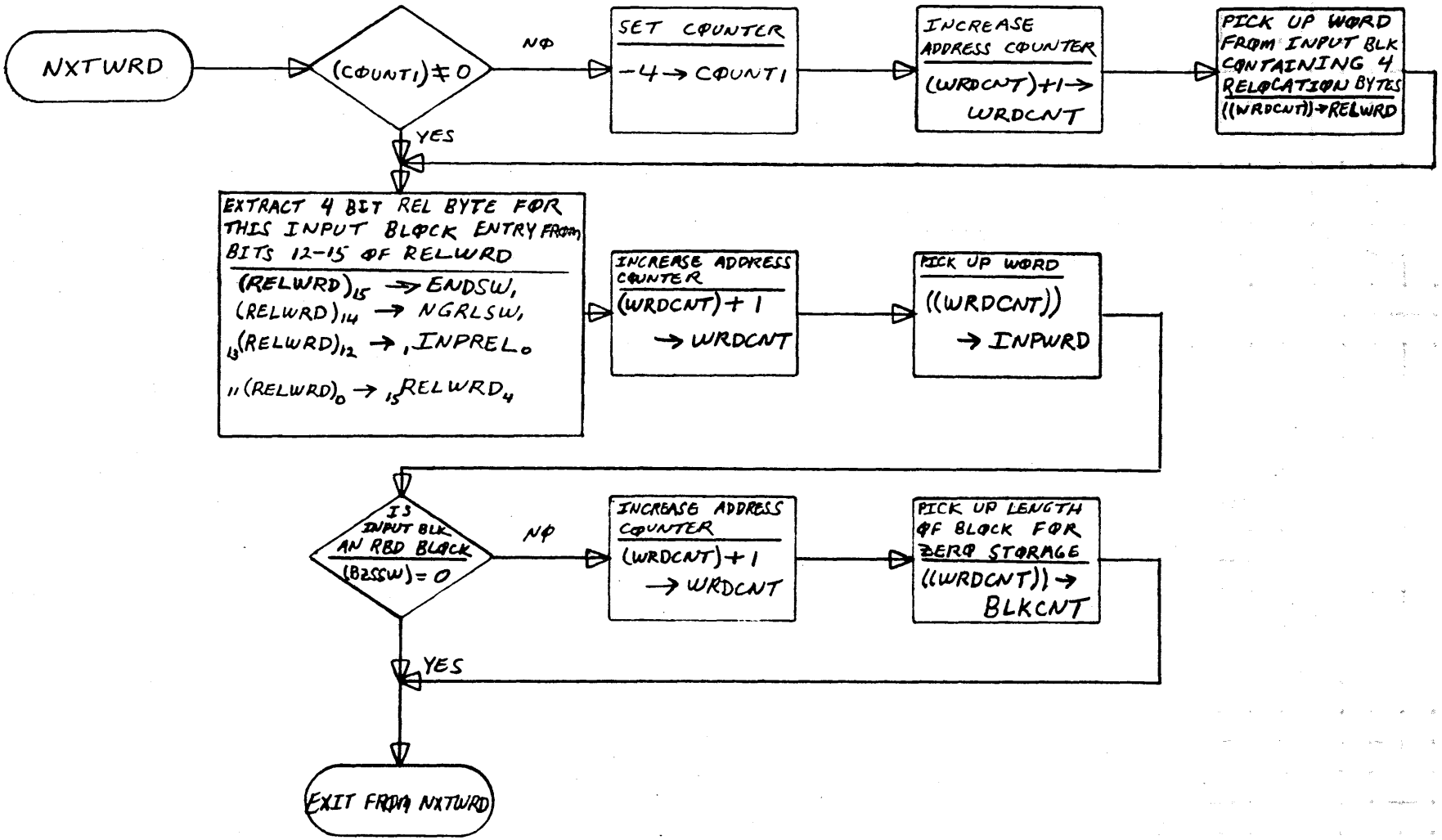


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	RBOAZS PAGE 9 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

39.240

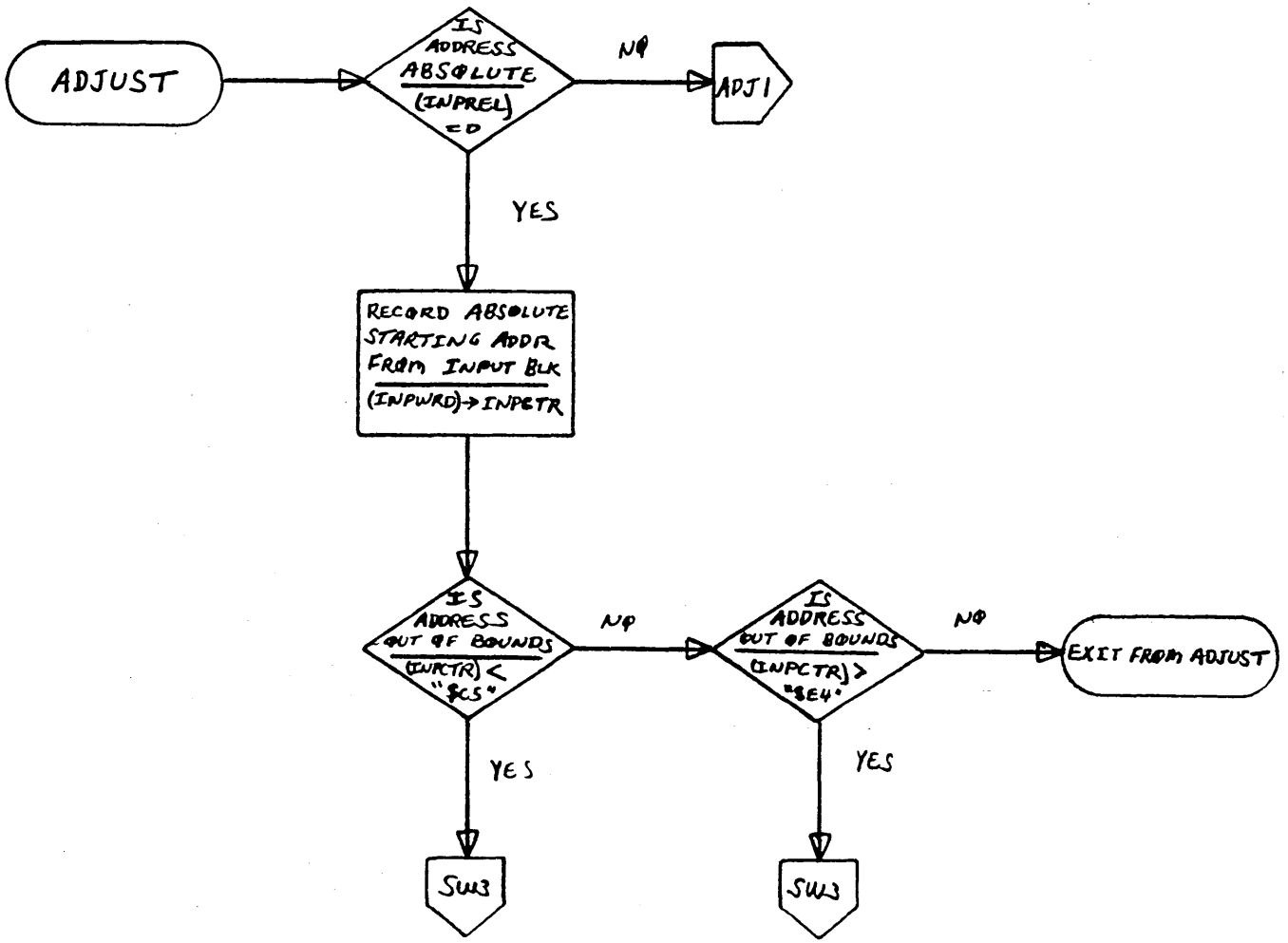


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	RBCB25 PAGE 10 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMER A127-1)

MAR 5 1971

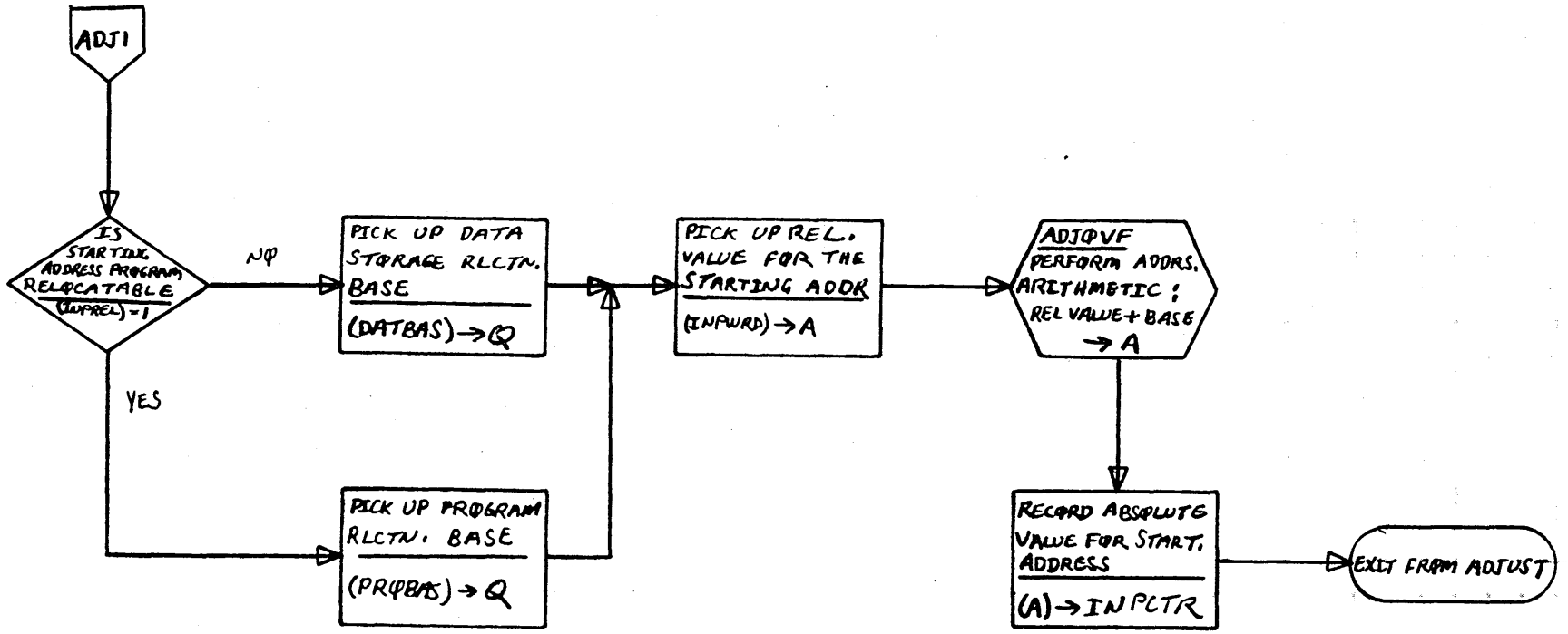
39,241



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	60308500 PAGE 11 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

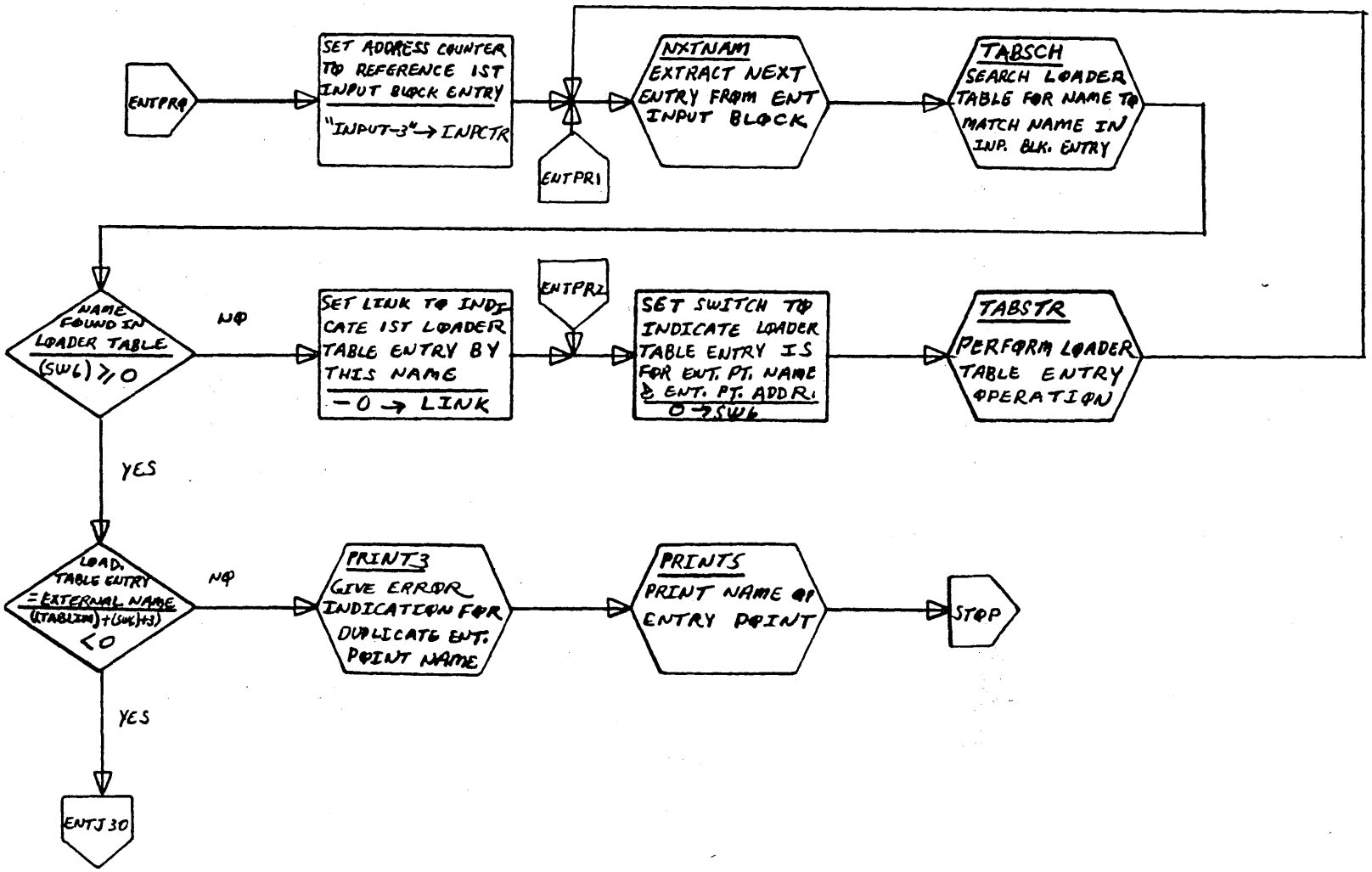
39,242



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	RBOBZS PAGE 12 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

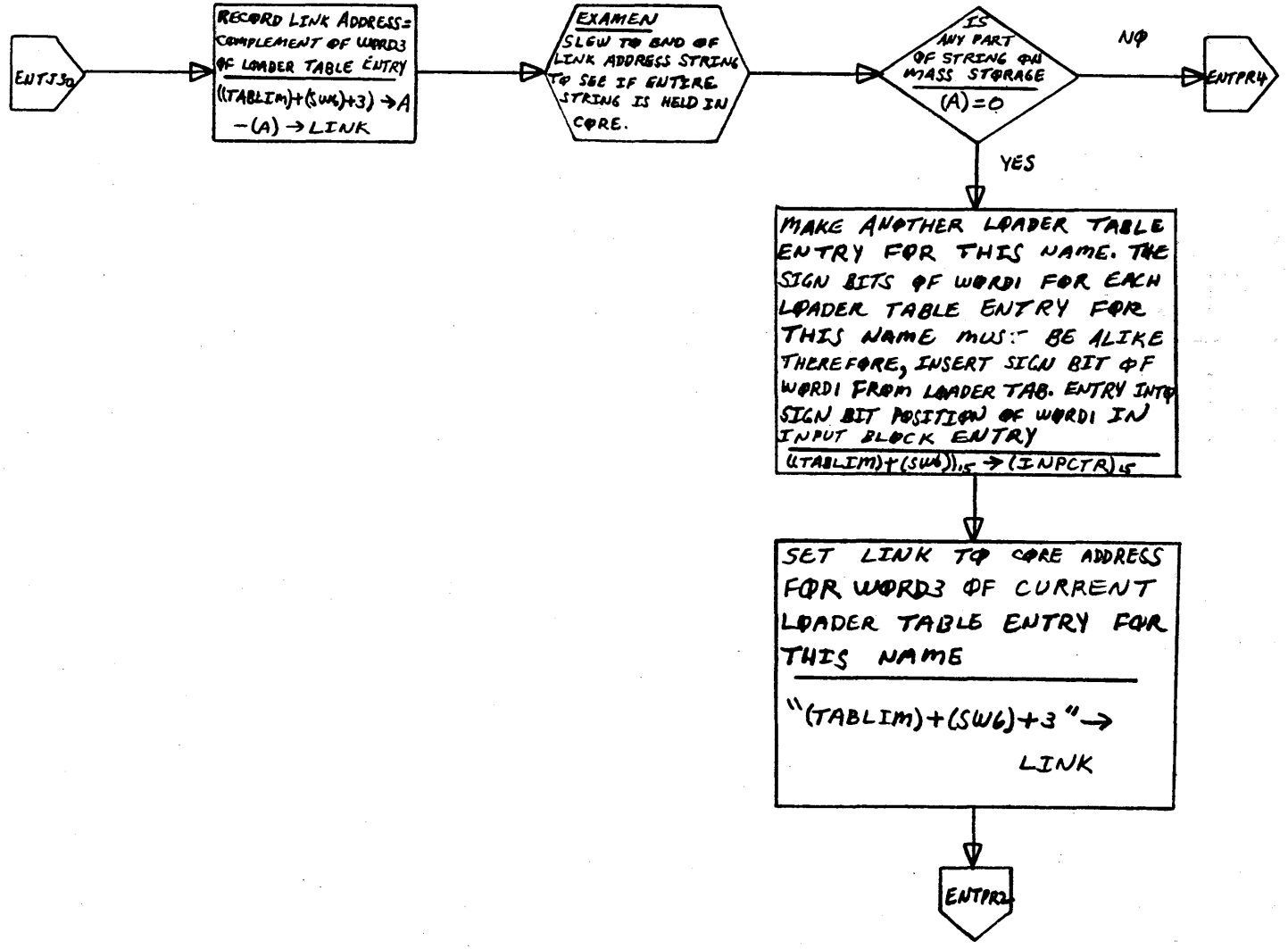
MAR 5 1971



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1100	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1100 Operating System		PROJECT MGR.		
FLOWCHART ☐	ENTEXT	PAGE 1 OF 9	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. Gordon	DATE 12 6 66	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

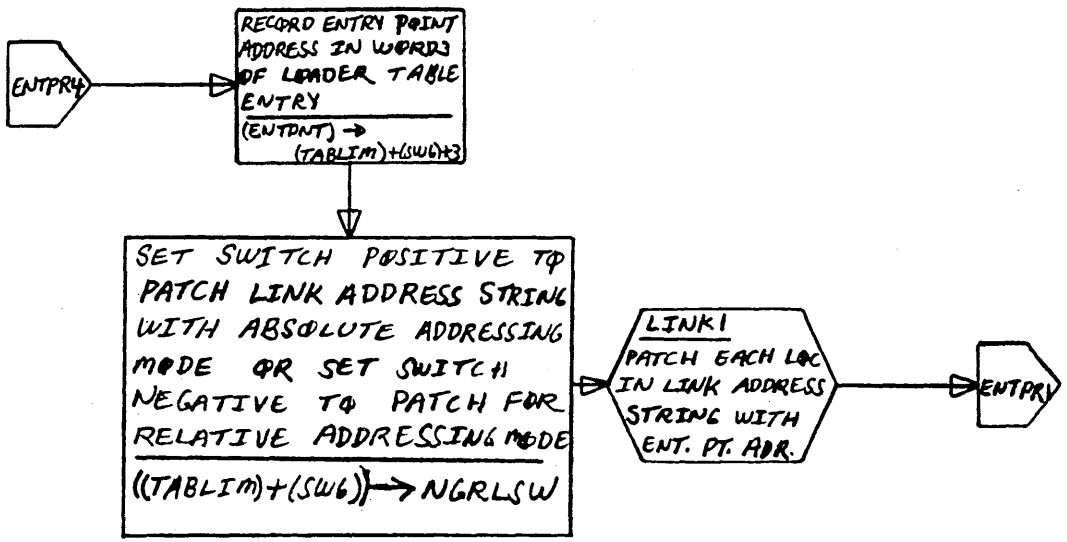


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	ENTEXT PAGE 2 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

59,245

AA1385 (FORMERLY CA127-1)

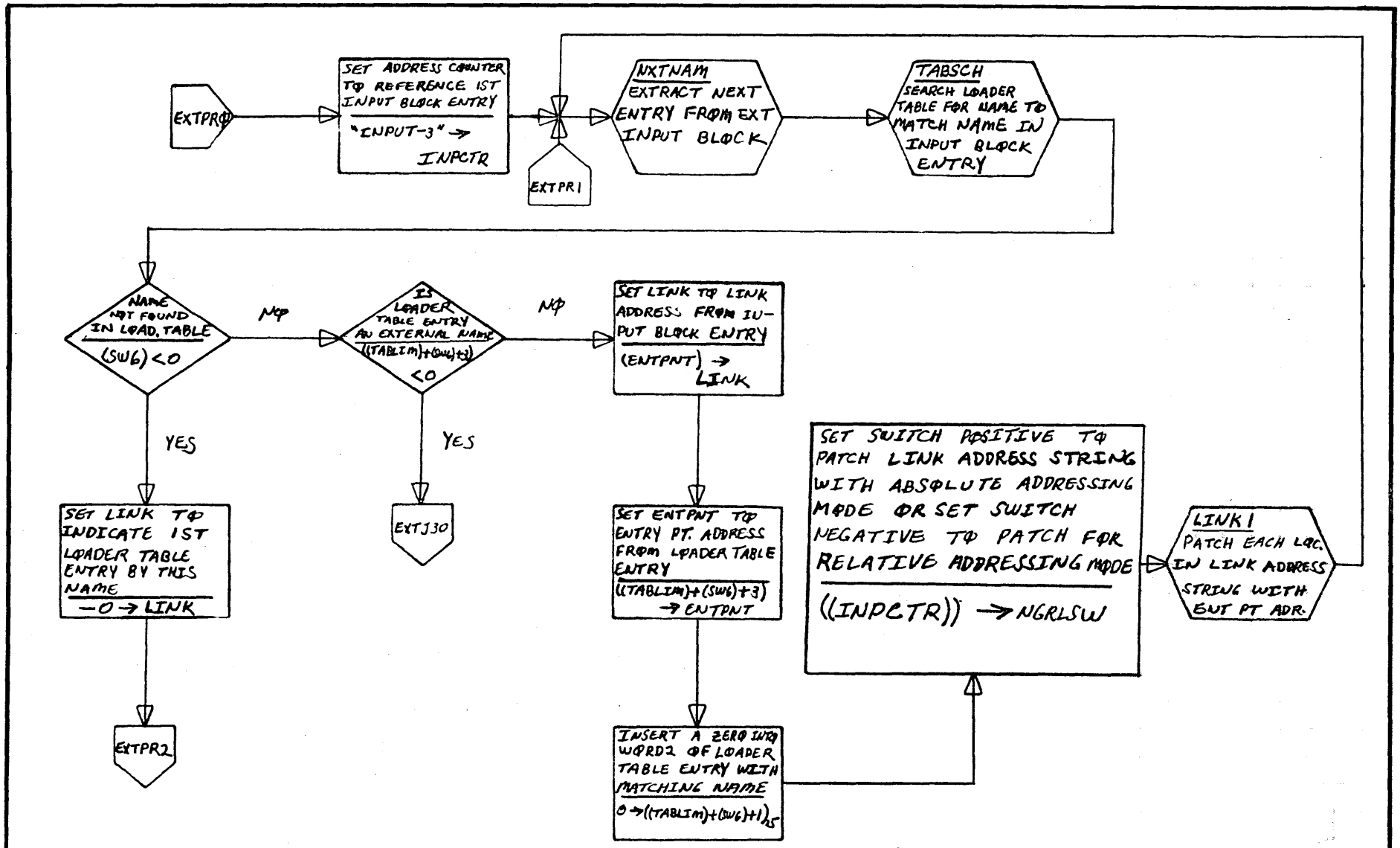


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	ENTEXT PAGE 3 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

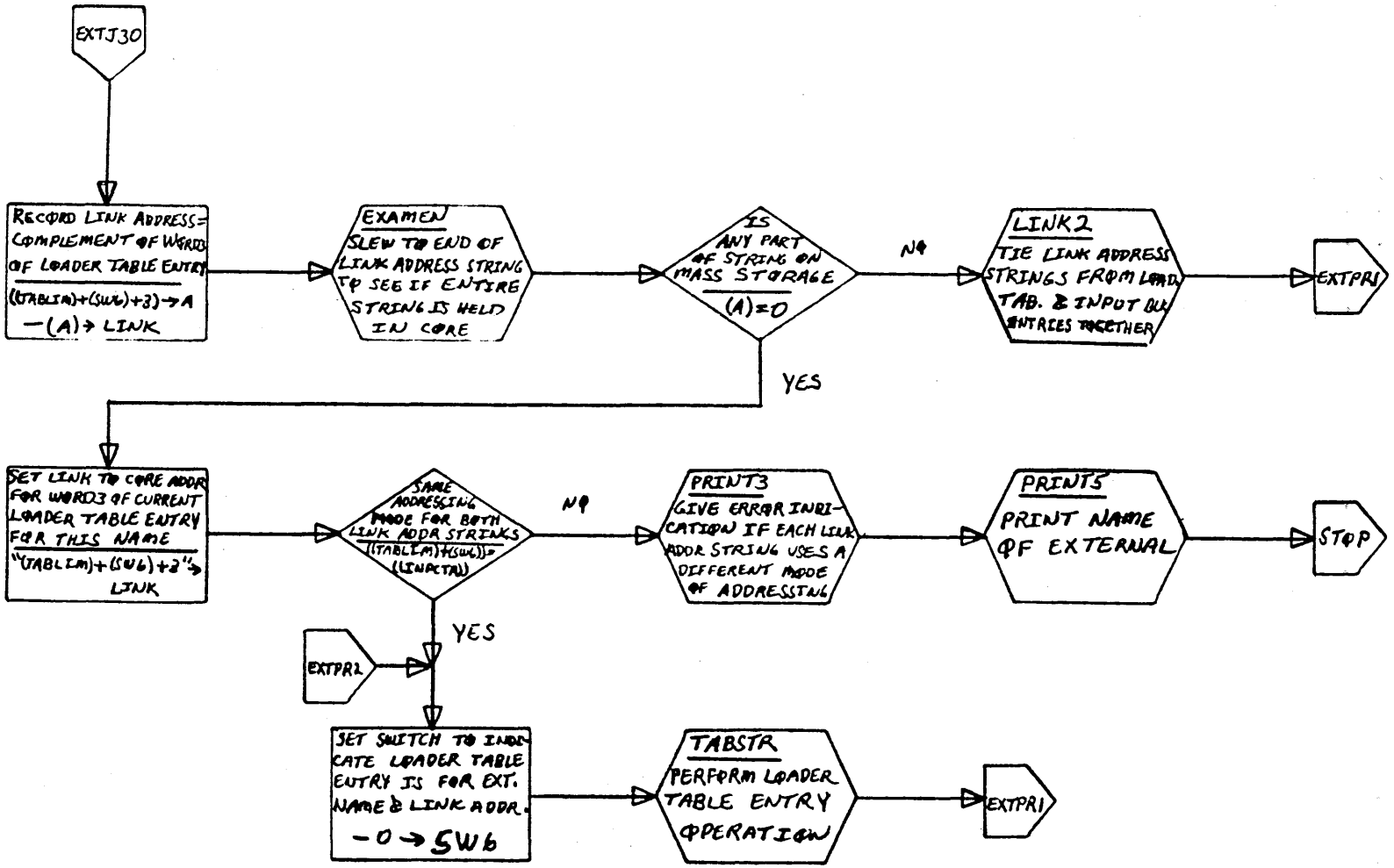
39.24

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	ENTEXT PAGE 4 OF 17	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

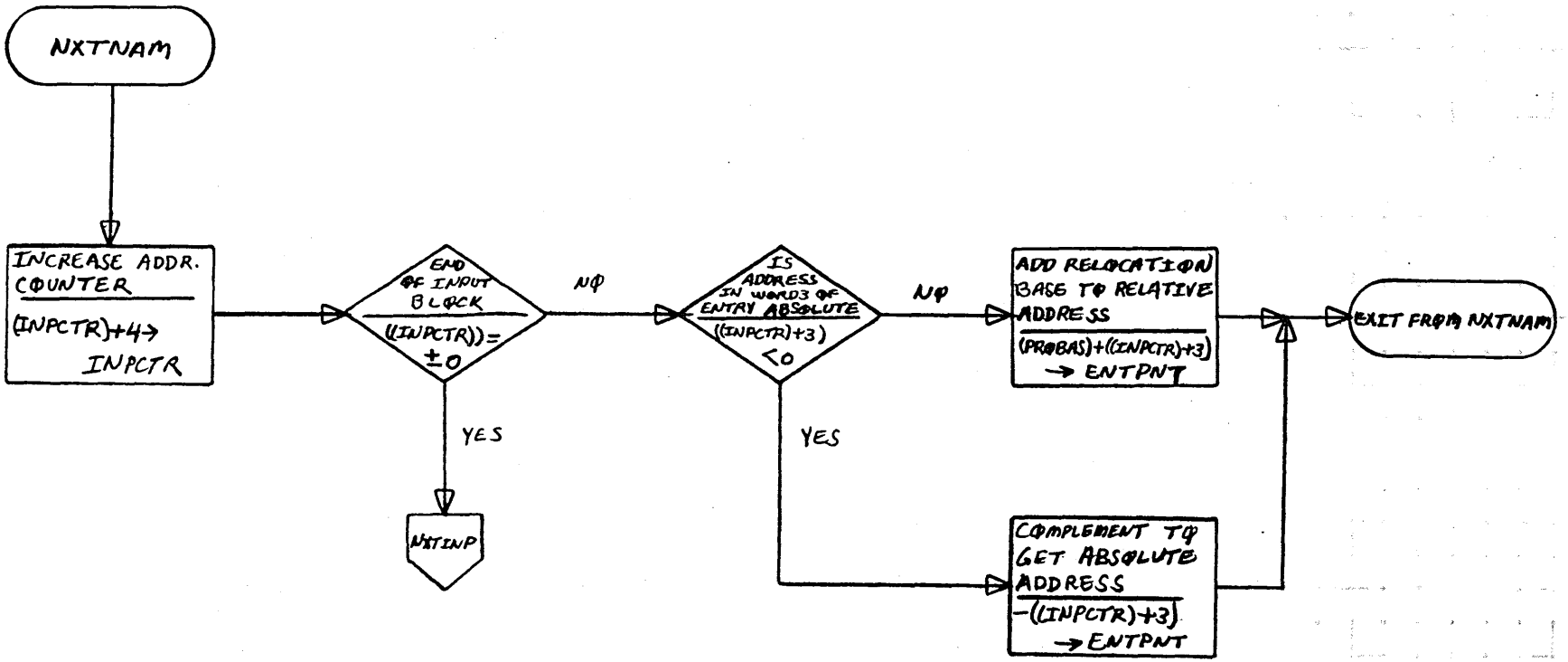


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE	PROJECT MGR.			
	ENTEXT PAGE 5 OF 7	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.248

AA1385 (FORMERLY CA127-1)

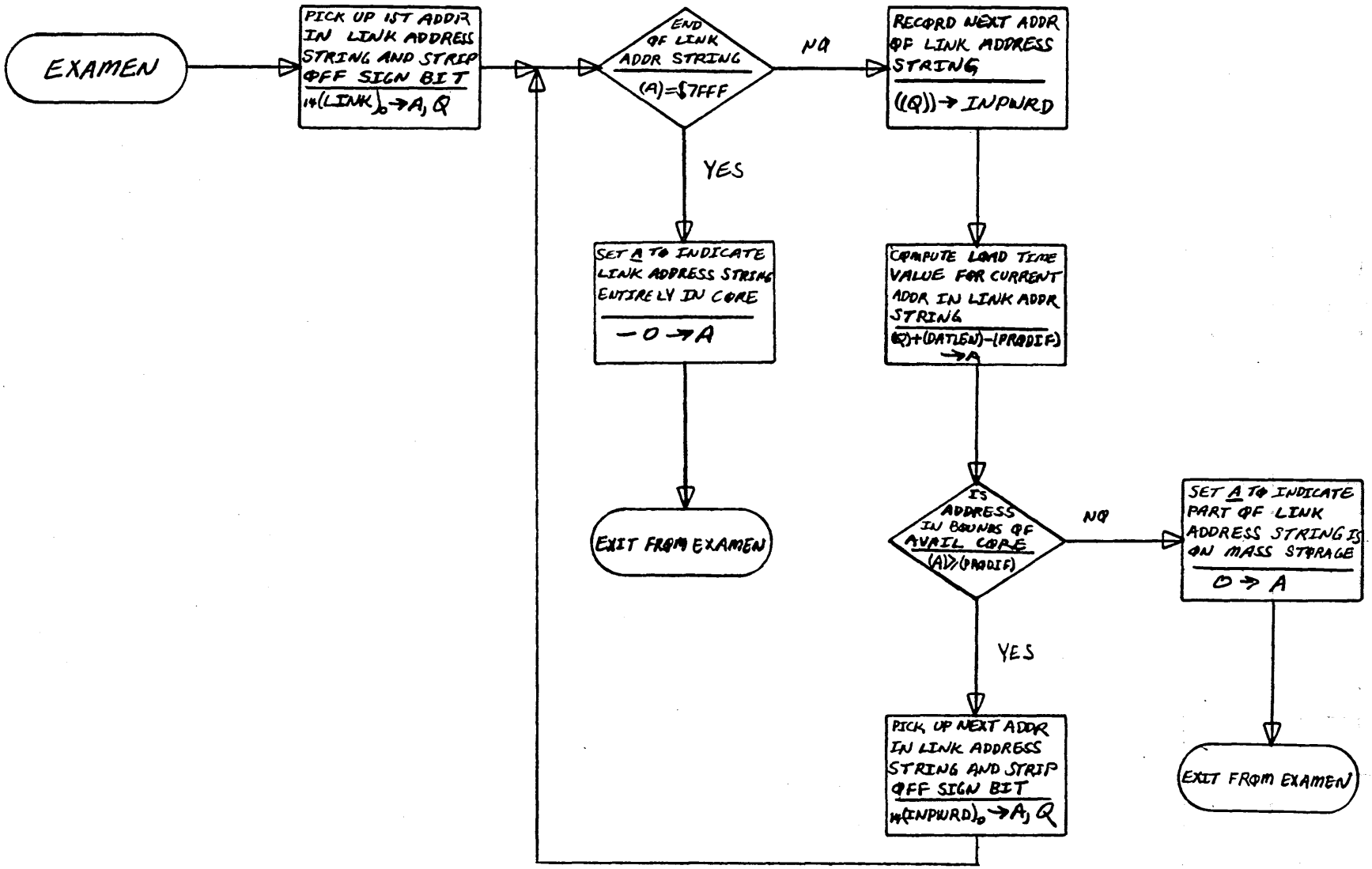


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	ENTEXT PAGE 6 OF 7	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.24.8

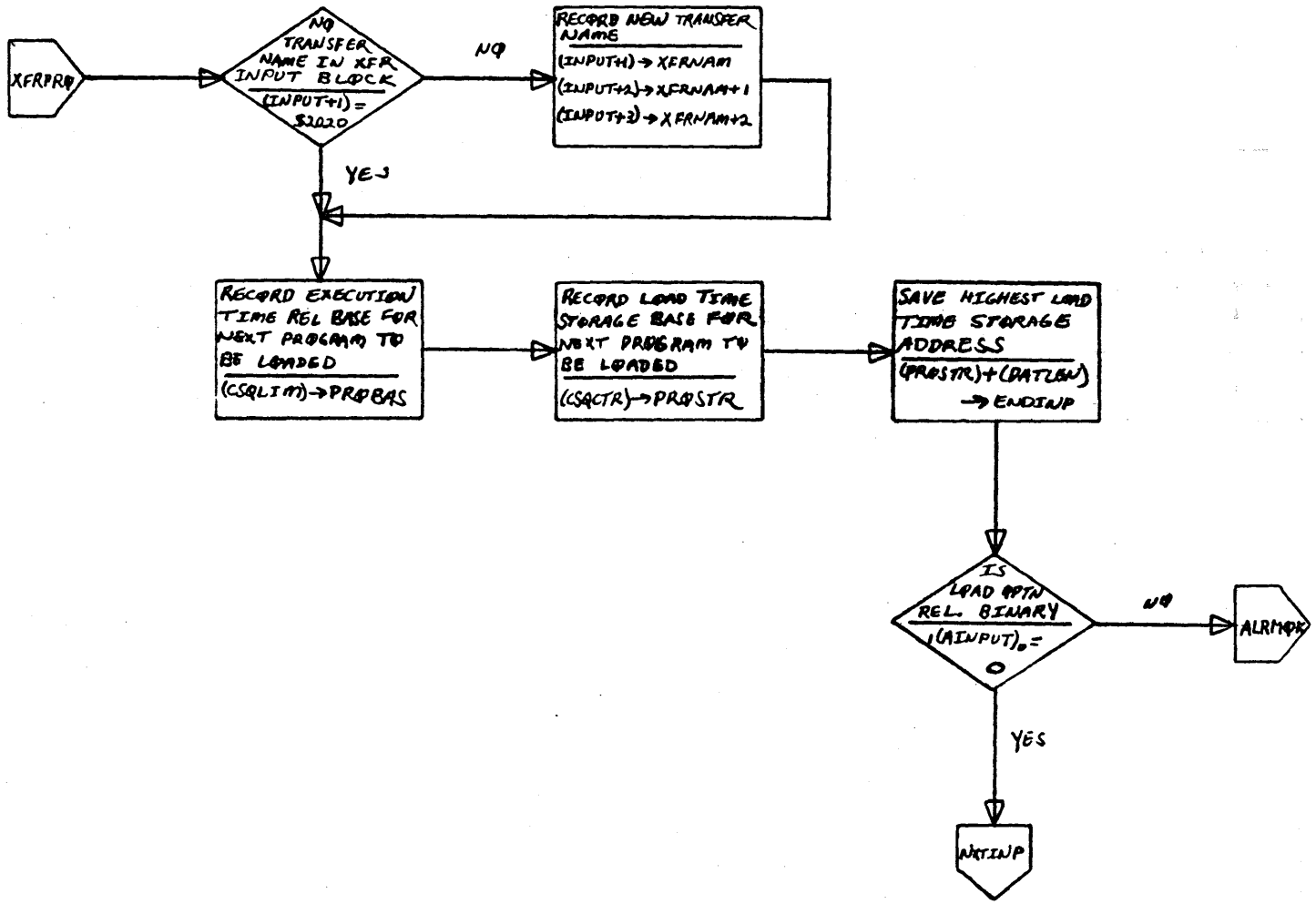
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	ENTEXT PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.250

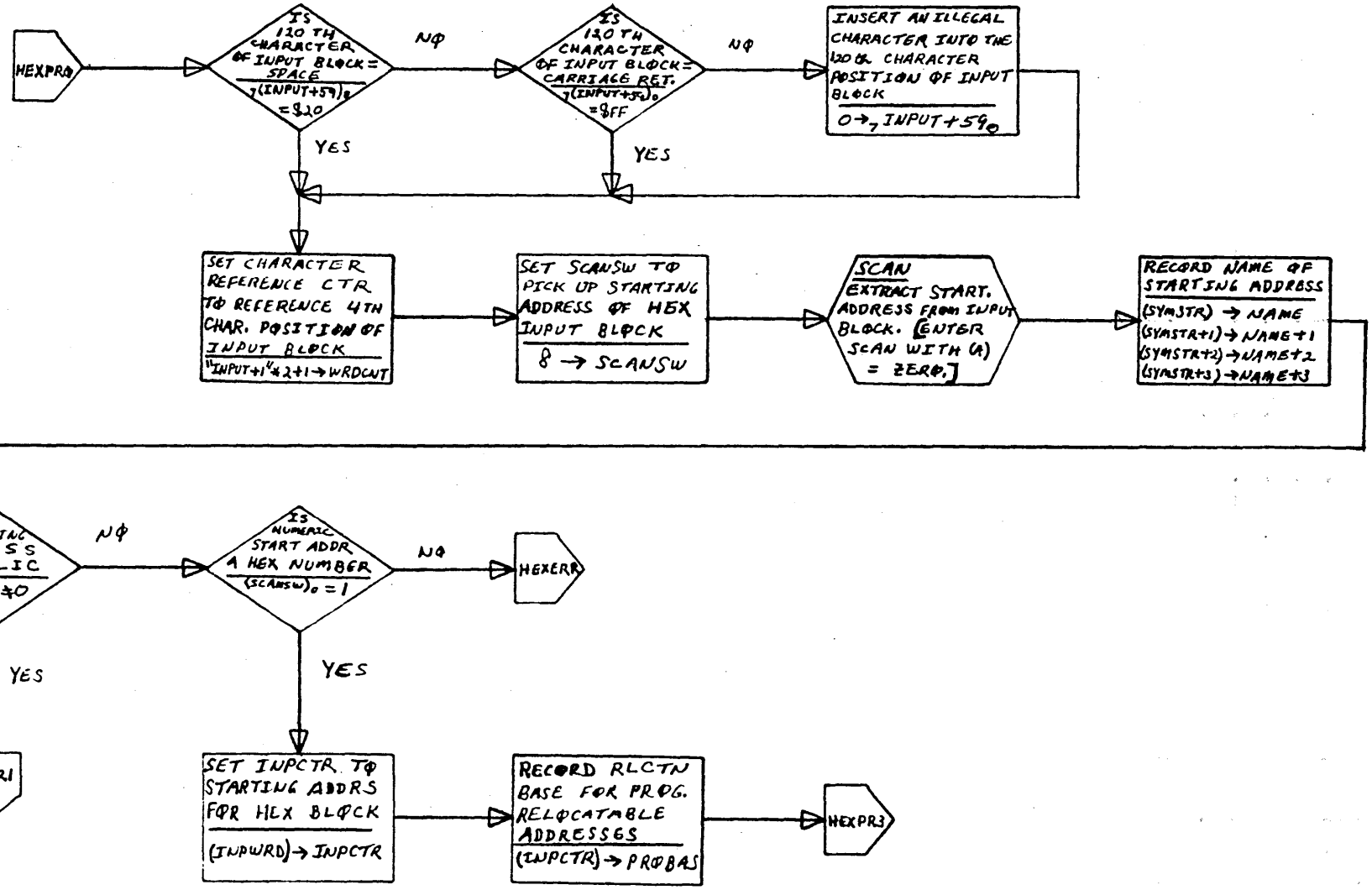


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS THS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
XFRPRP PAGE 1 OF 1	PROJECT NAME			
NUMBER ISSUE DATE	TASK NO.			
DRAWN BY T. Gordon DATE 12-6-66	TASK NAME			

MAR 5 1971

39.251

AA1385 (FORMERLY CA127-1)

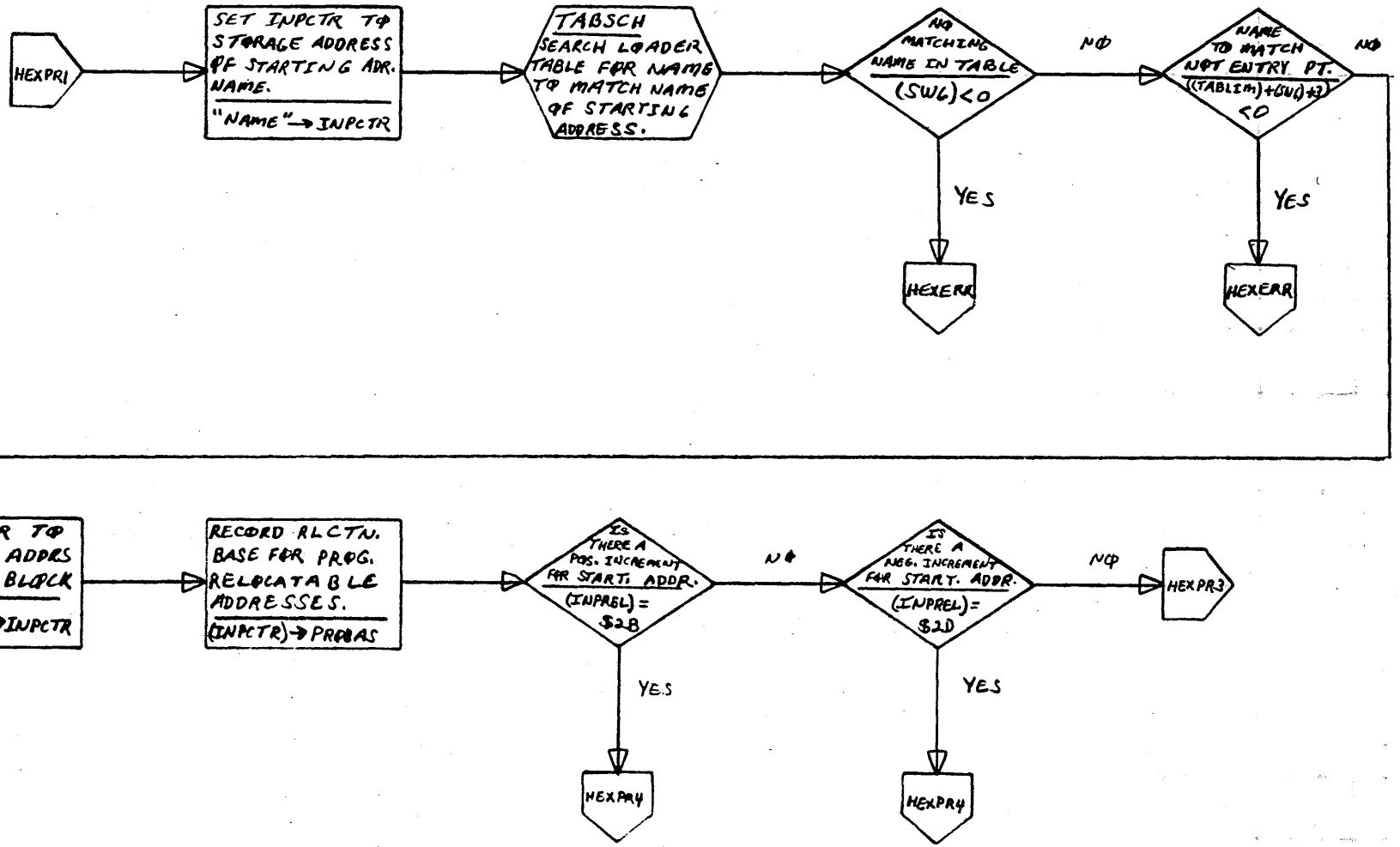


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1.1.5	MACH. TYPE M100	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE M100 Operating System		PROJECT MGR.		
	HEXPRO	PAGE 1 OF 12	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.252

AA1385 (FORMERLY CA127-1)

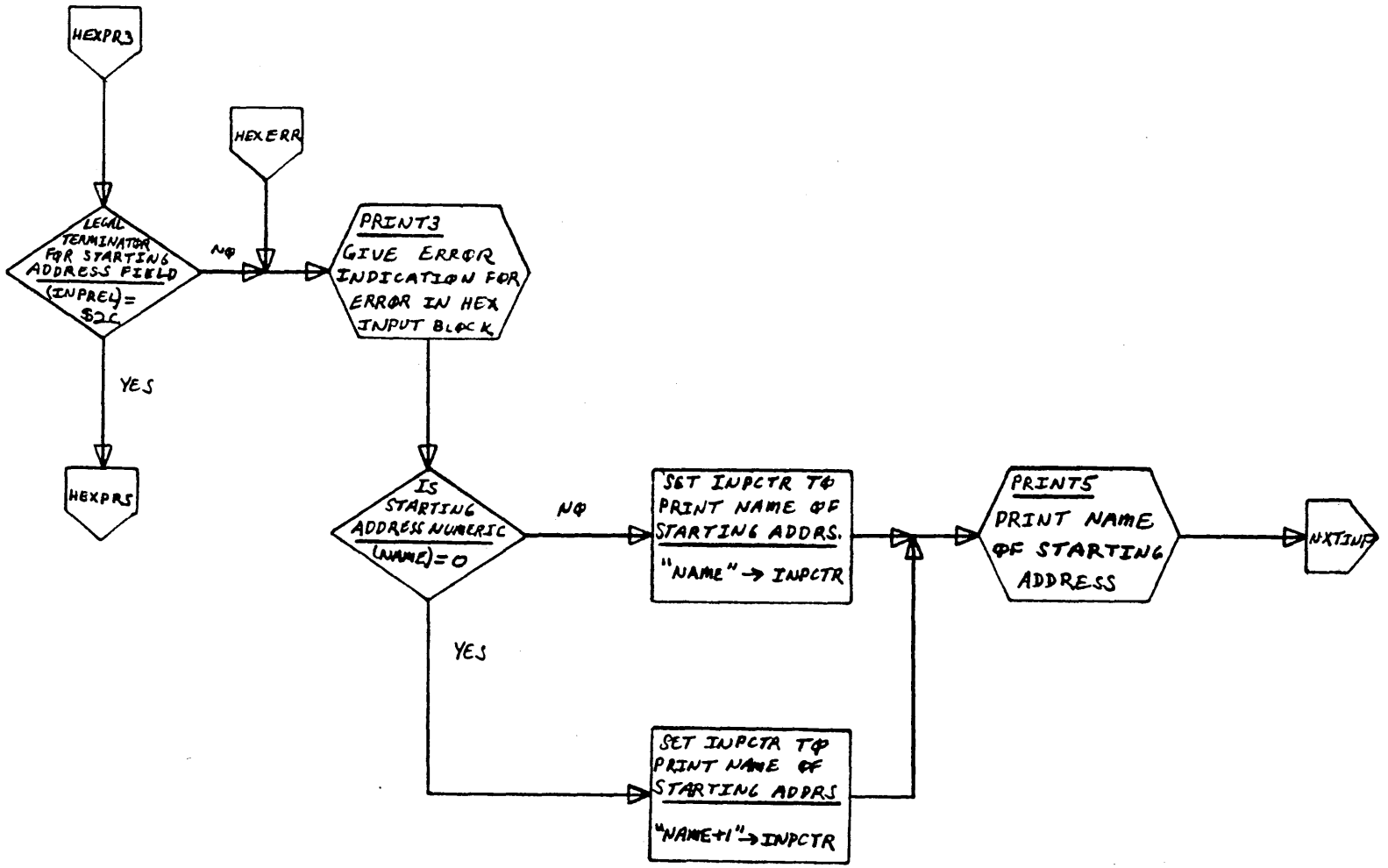


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	NEVPEC PAGE 2 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMER [illegible] A127-1)

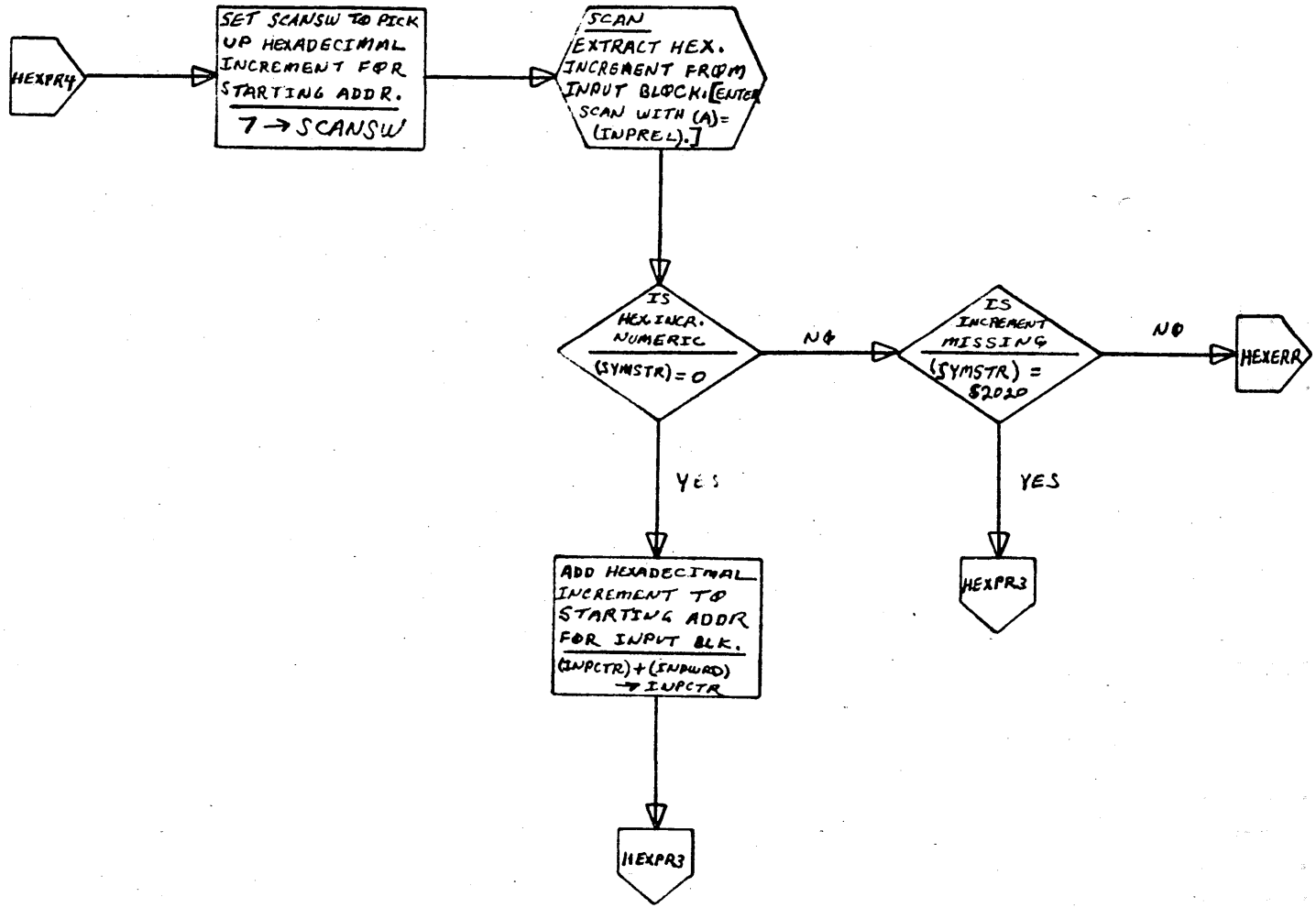
MAR 5 1971

39.253



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	HEXPR0 PAGE 3 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

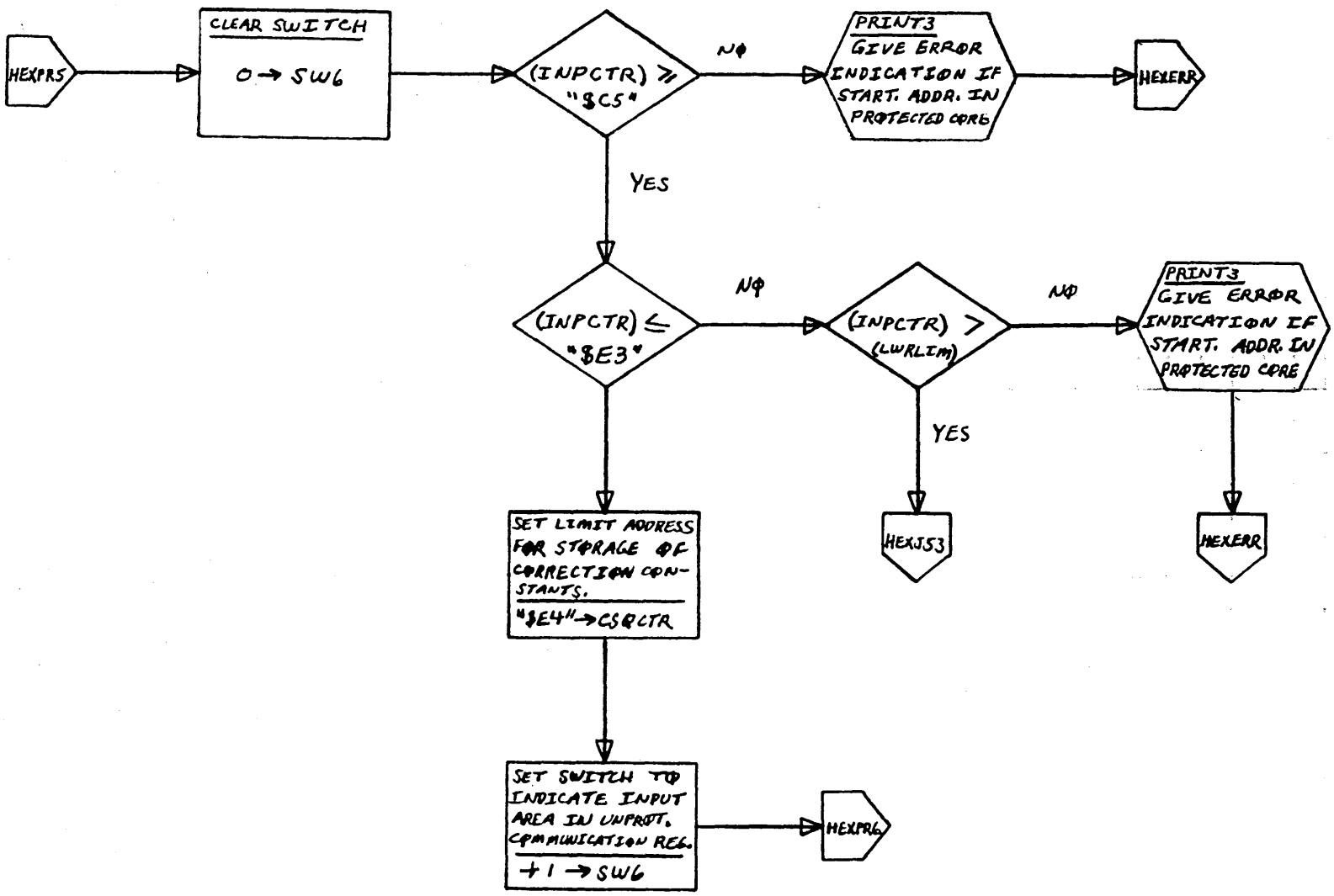


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	HEXPR0 PAGE 4 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.255

AA1385 (FORMERLY CA127-1)

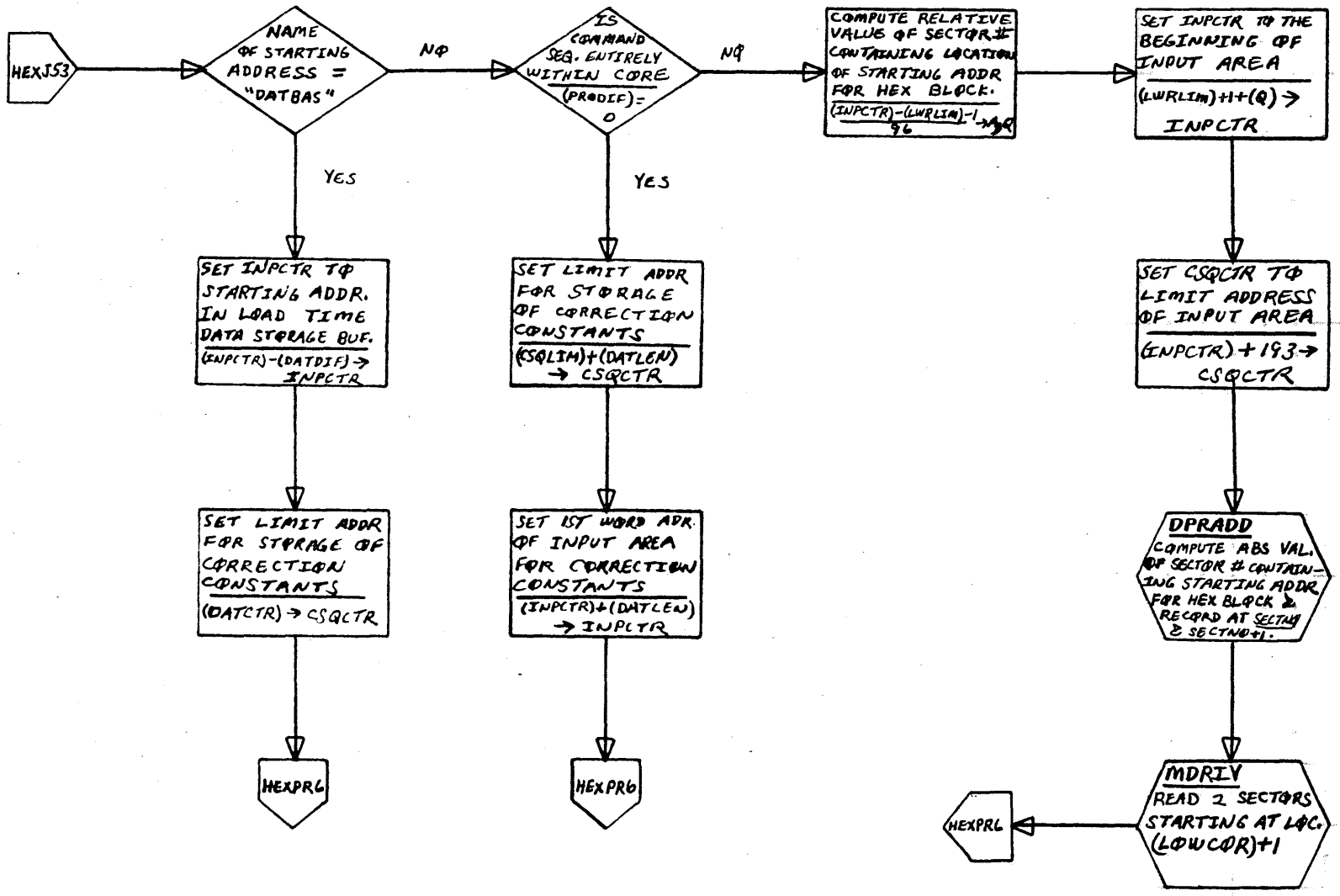


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE	PROJECT MGR.			
	HEXPRØ PAGE 5 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

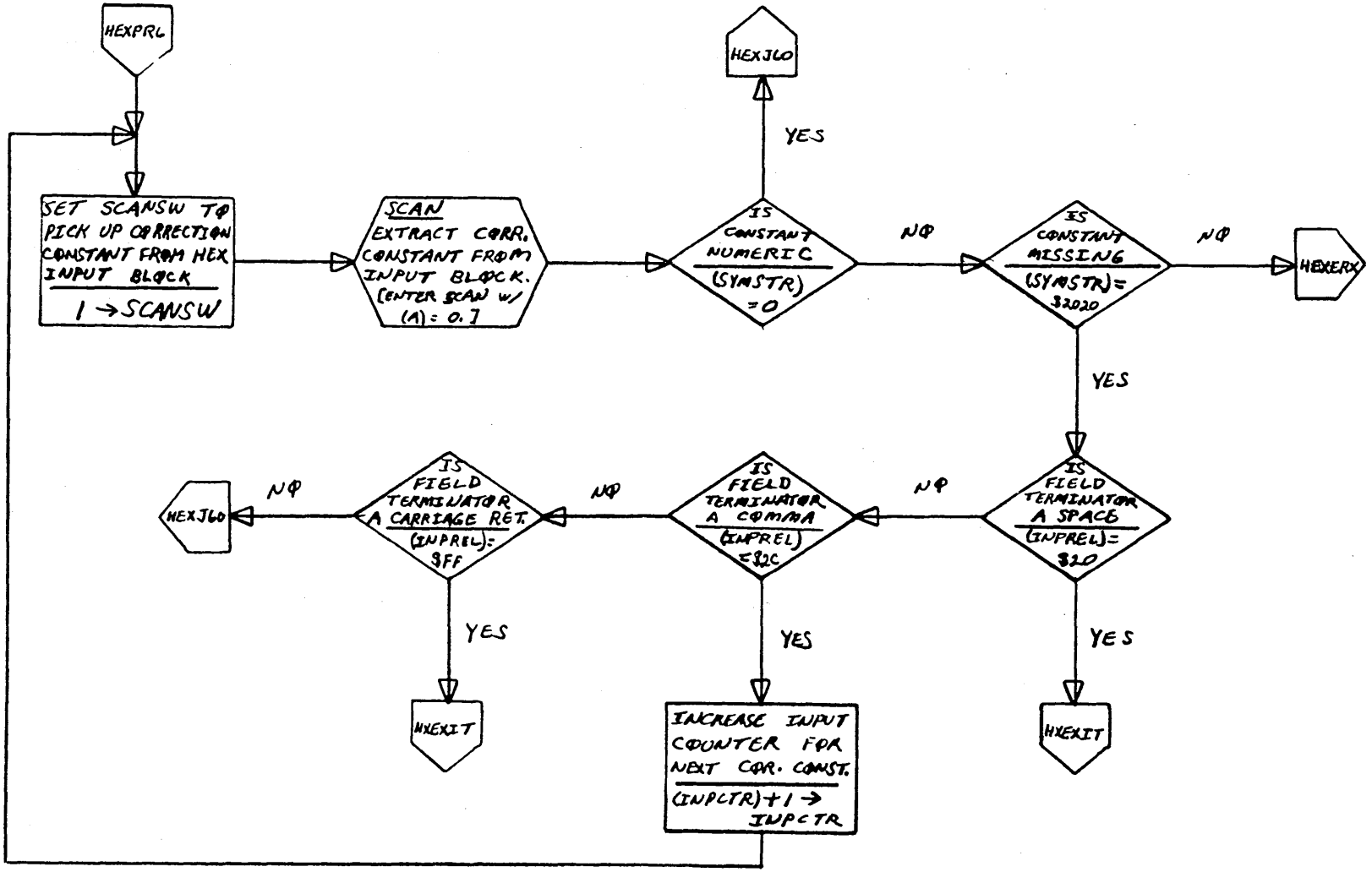
39.256

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	HEXPR6 PAGE 6 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

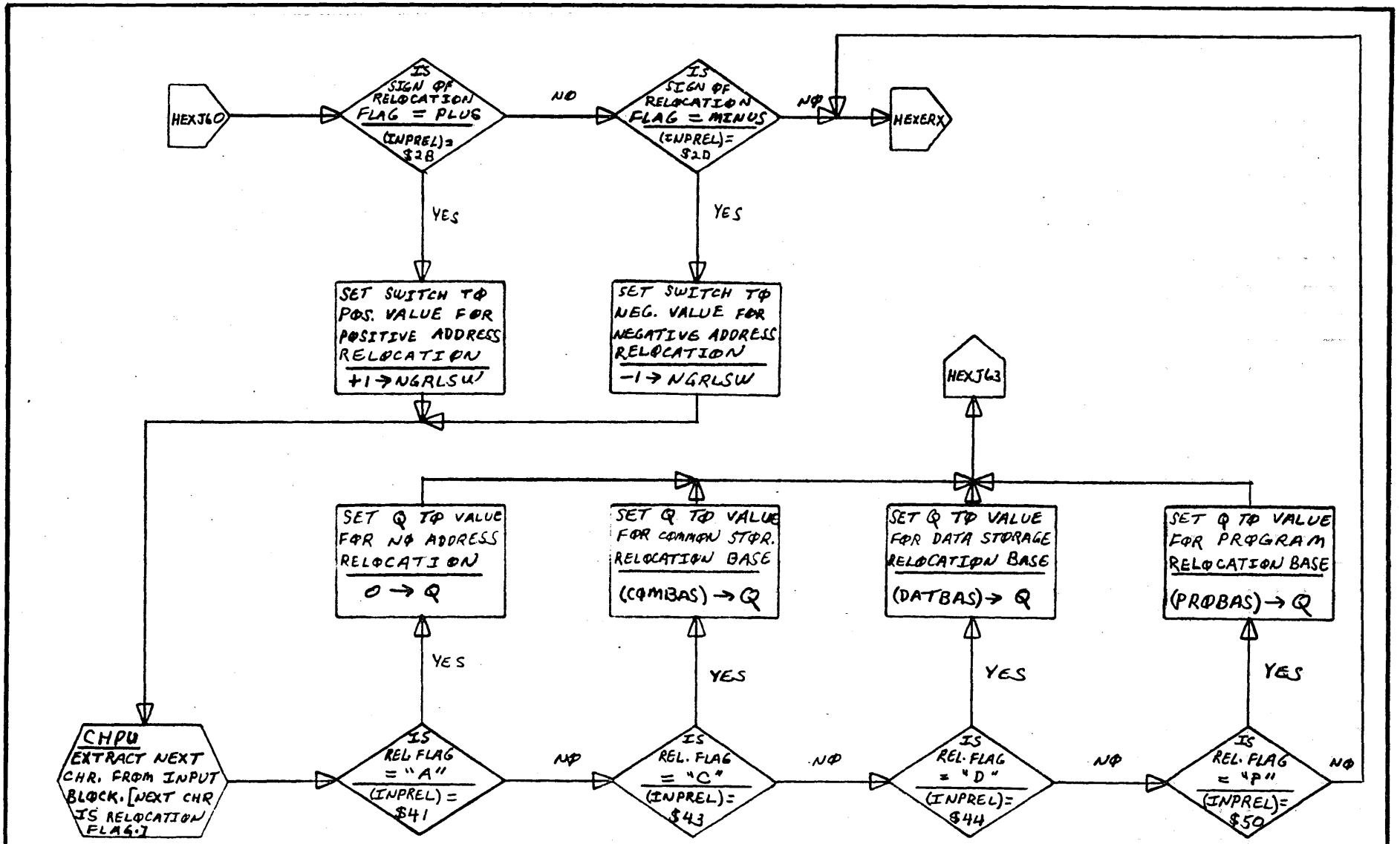


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	NEXERP PAGE 7 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39 .258

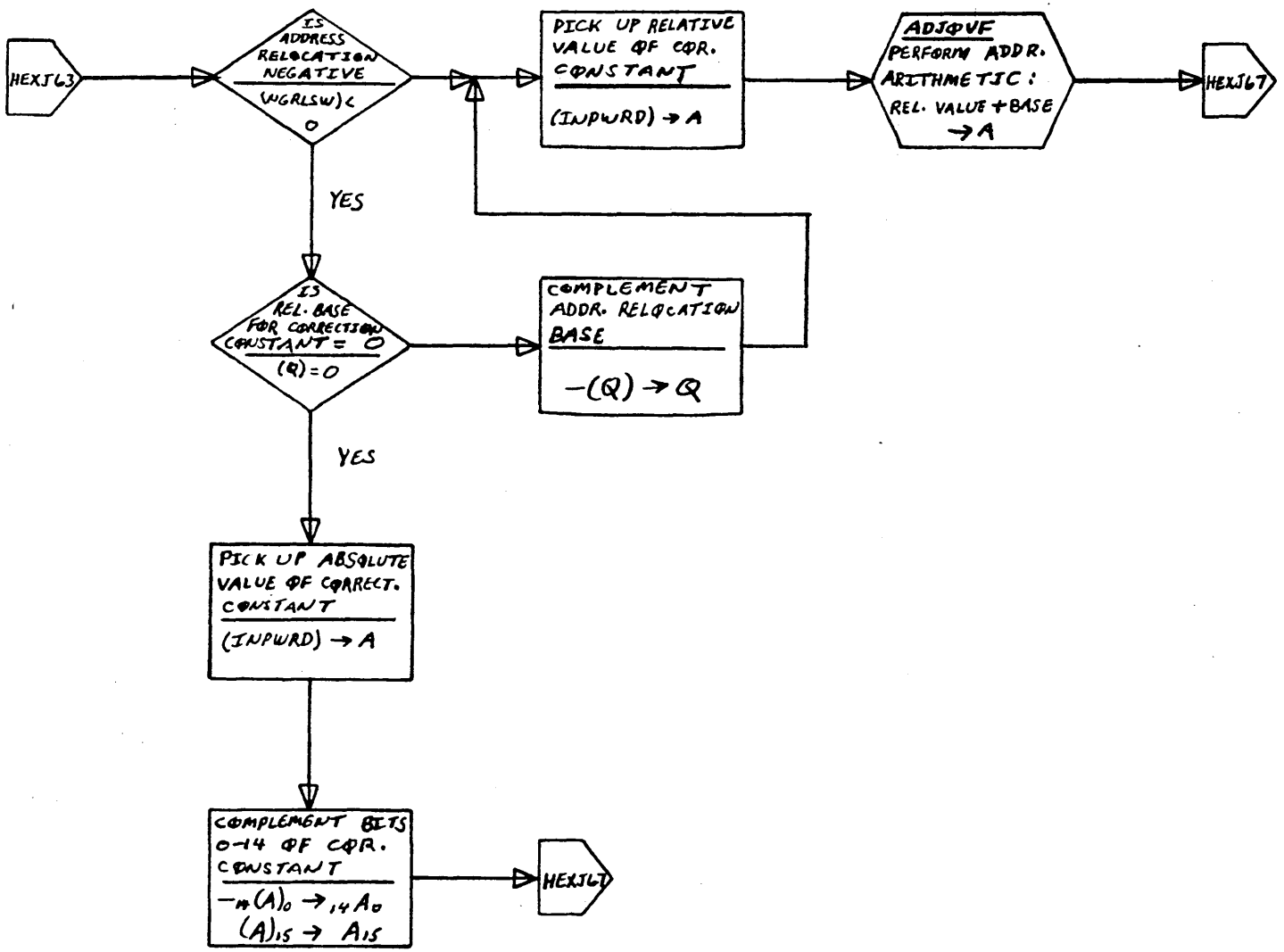
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	HEXJ60 PAGE 8 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

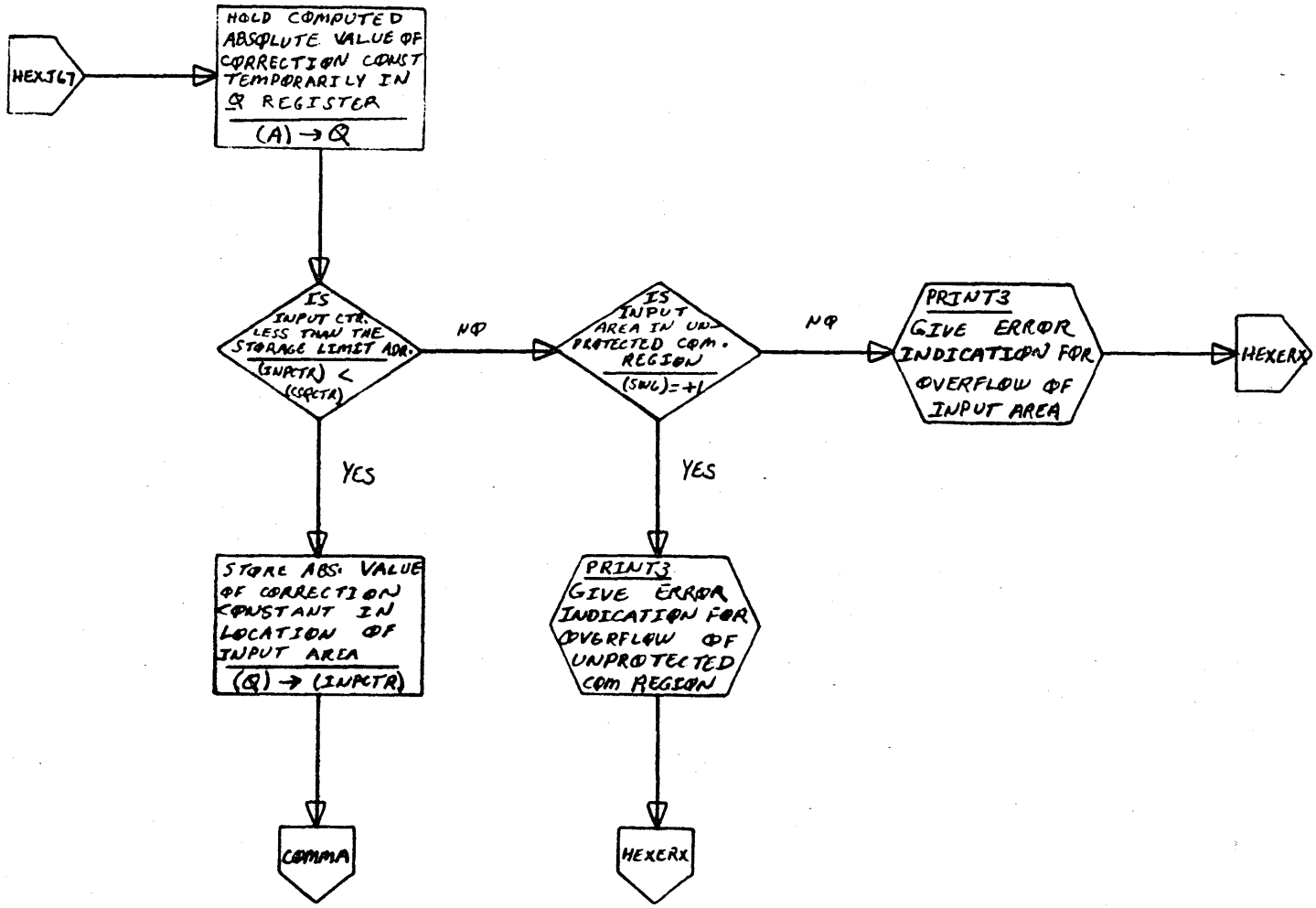


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	HF4FC0 PAGE 9 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.260

AA1385 (FORMERLY CA127-1)

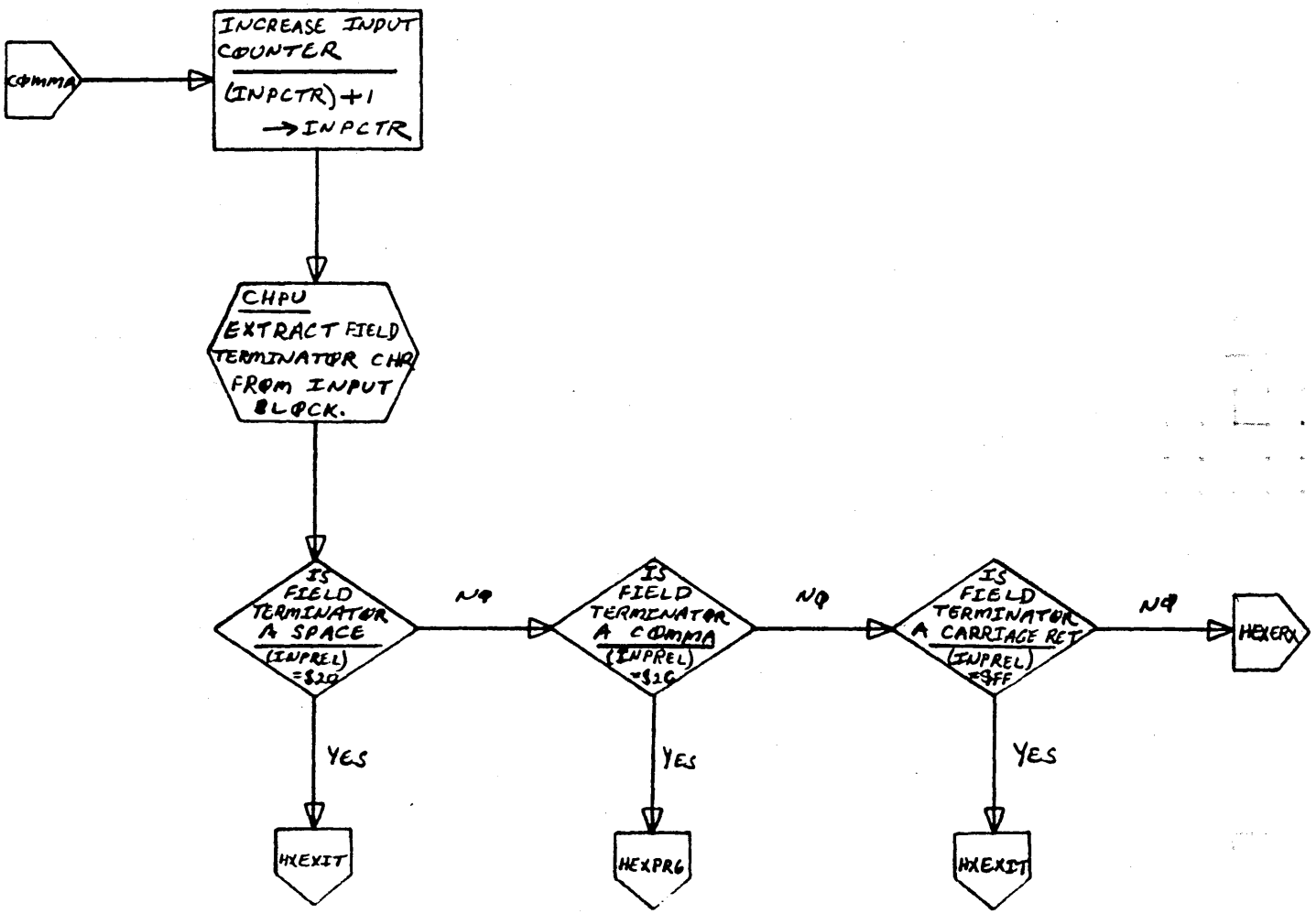


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	HEXPRO PAGE 10 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMER CA127-1)

MAR 5 1971

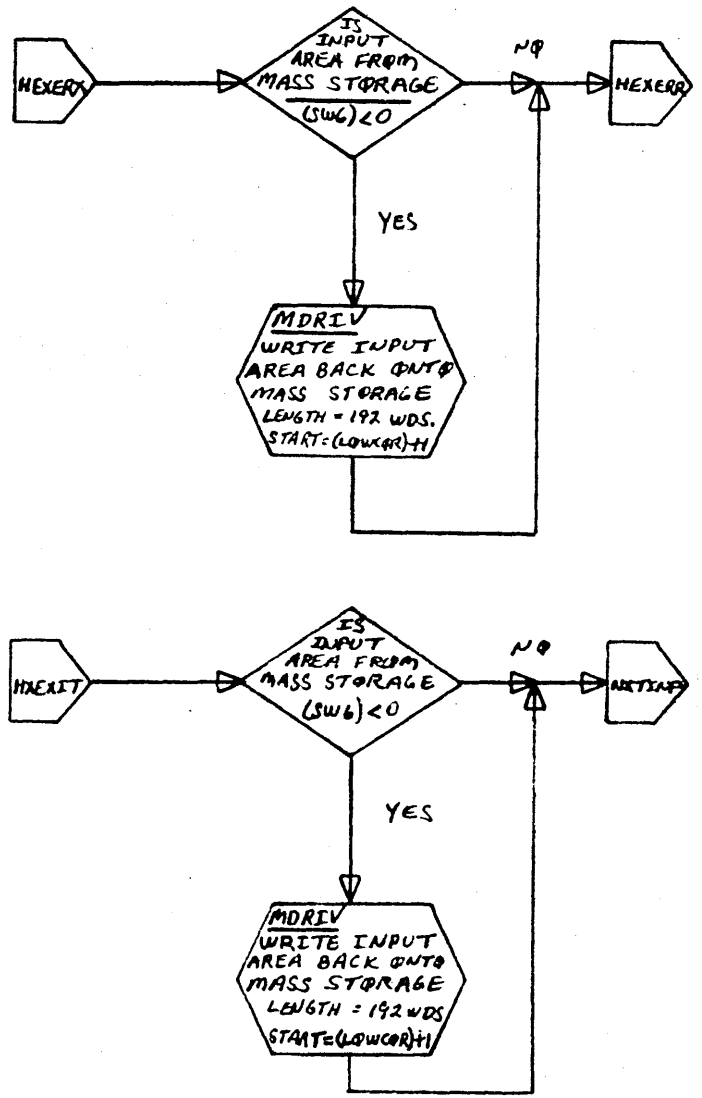
39,261



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐☐	NEXXOP PAGE 11 OF 12	PROJECT NAME			
DECISION TABLE ☐☐	NUMBER / ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

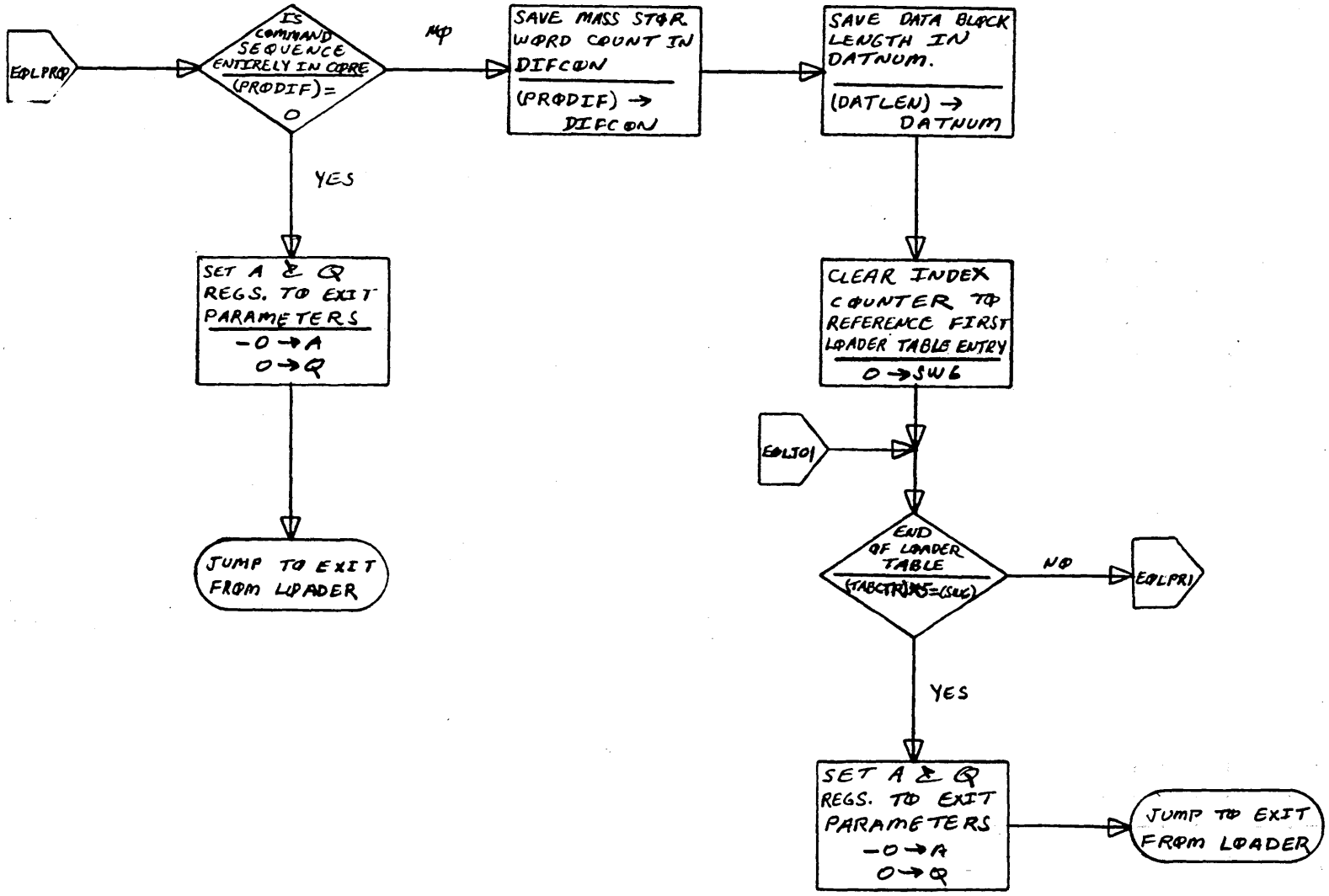


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	NFYPRO PAGE 12 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMER CA127-1)

MAR 5 1971

39263

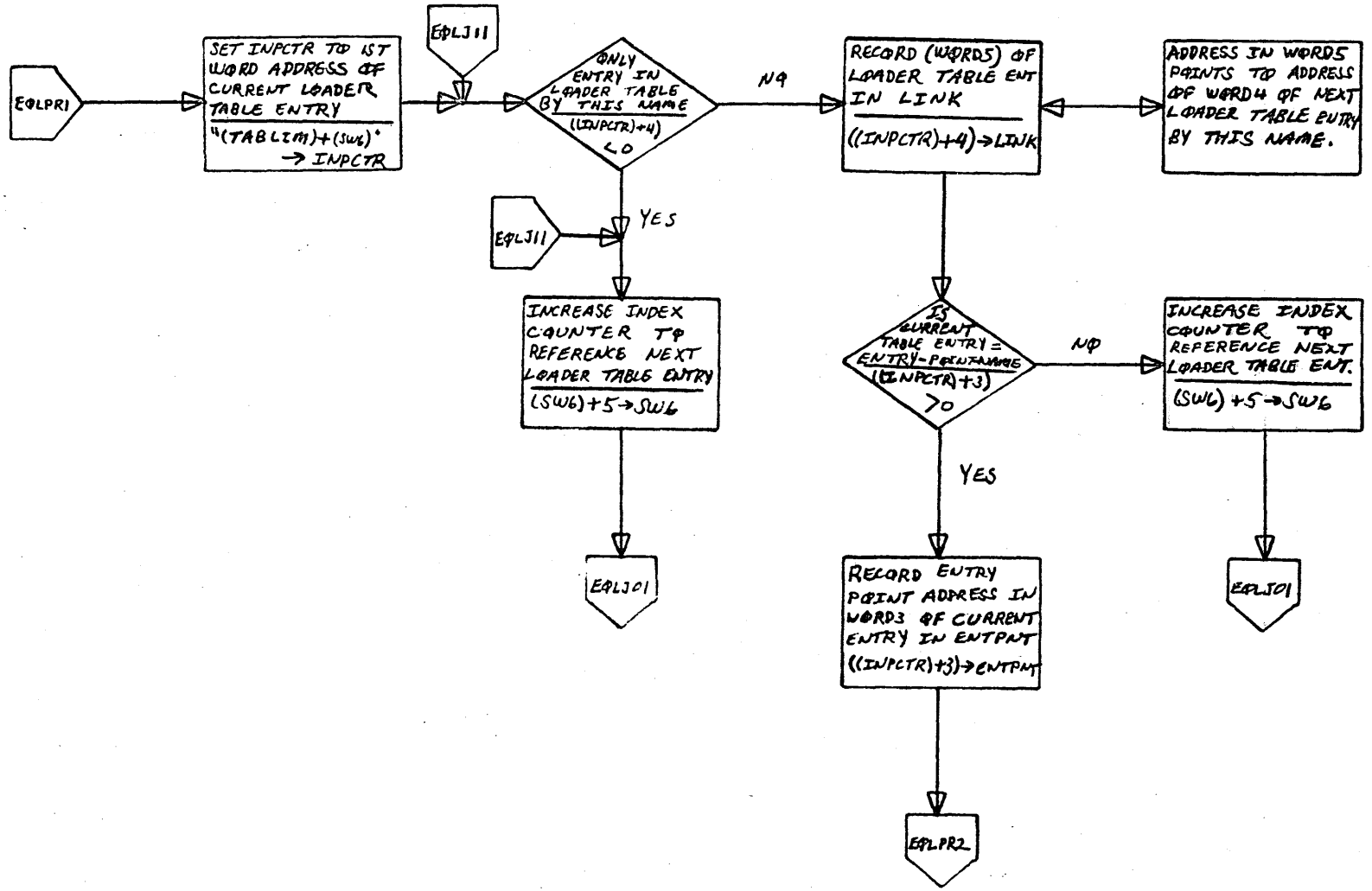


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 3116 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700 OPERATING SYSTEM EQLPRQ PAGE 1 OF 6	PROJECT MGR.			
		PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

39.264

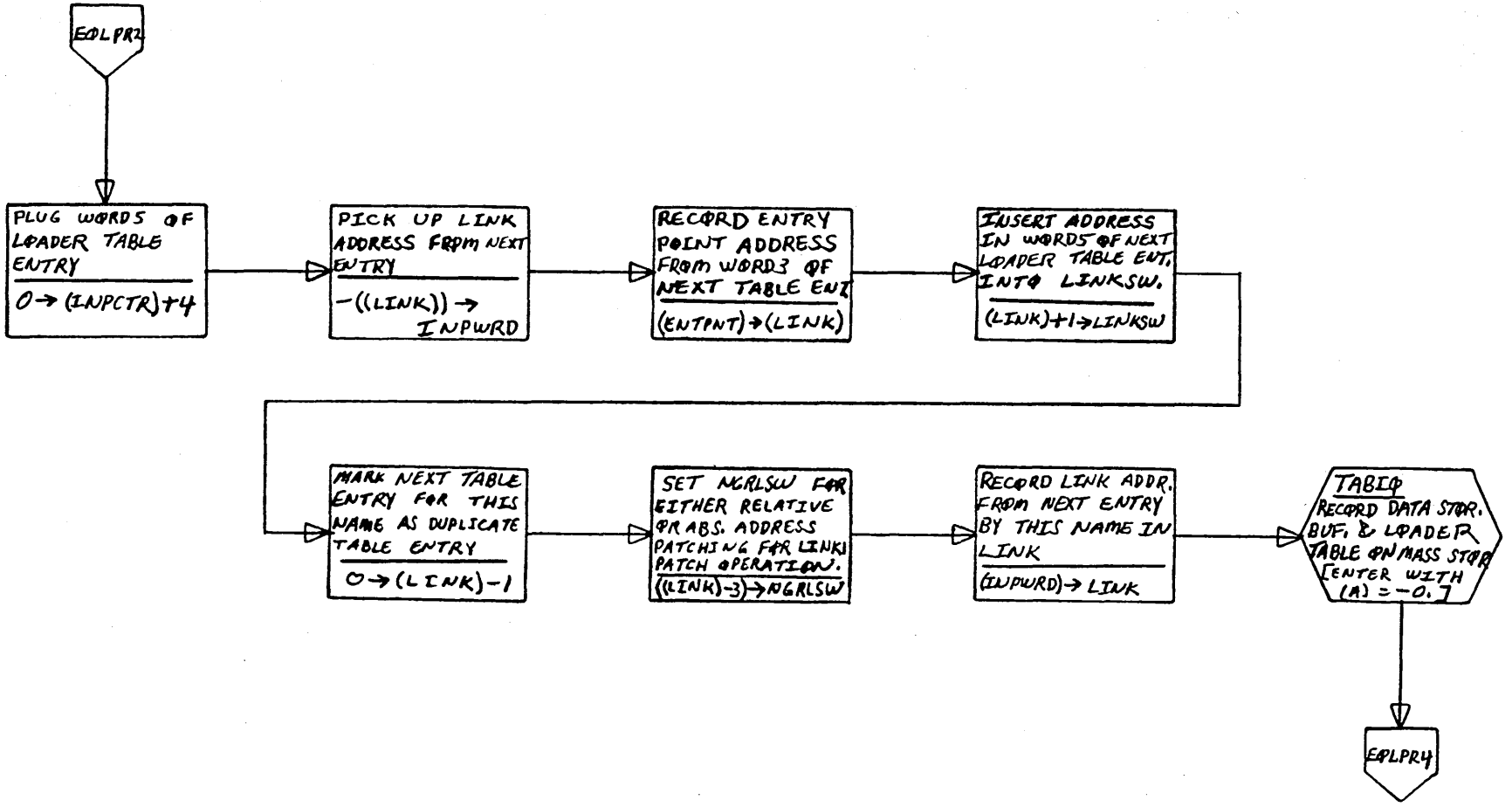
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	EQLPR0 PAGE 2 OF 6	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

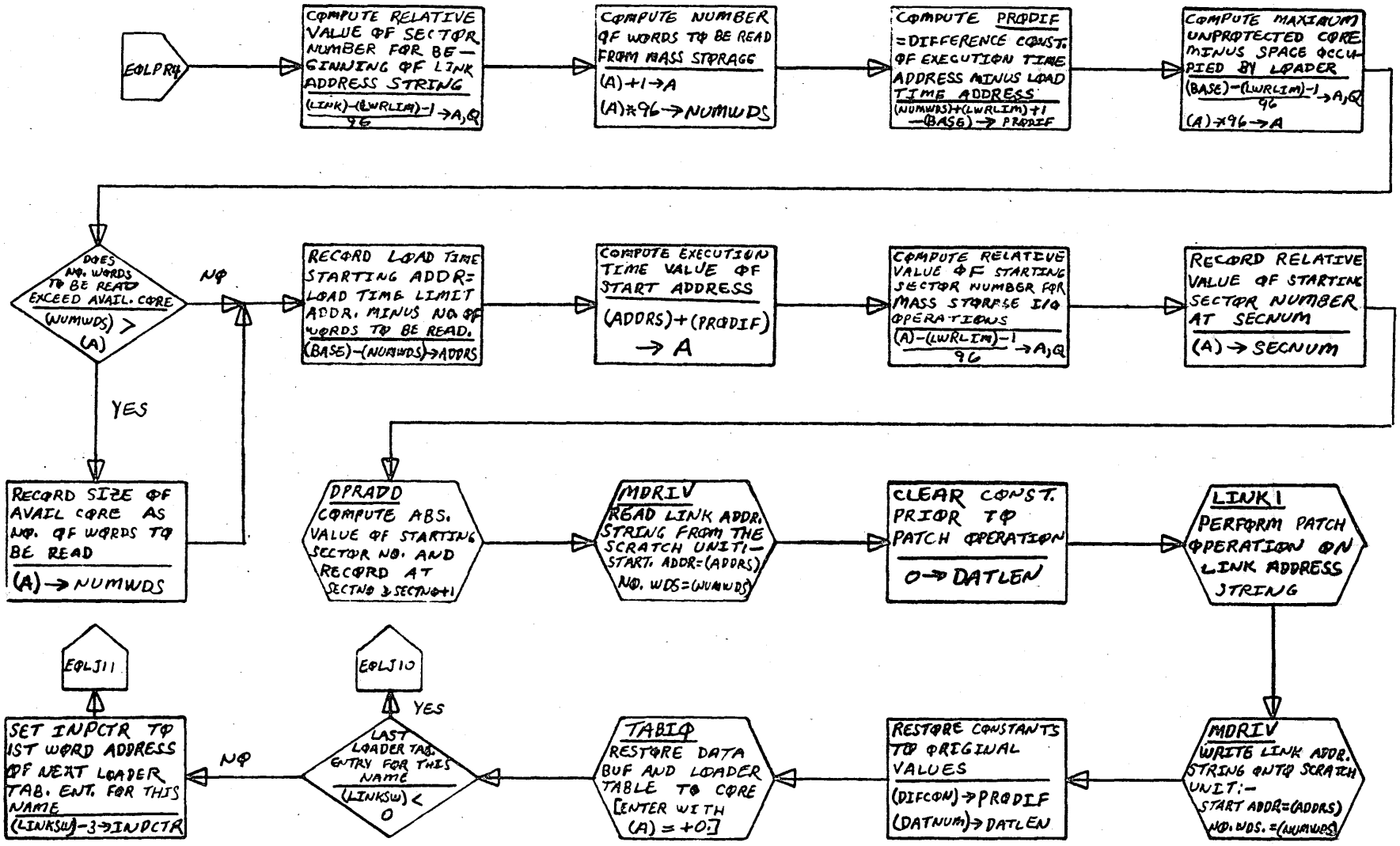


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	EQLPR0 PAGE 3 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

39.26.6

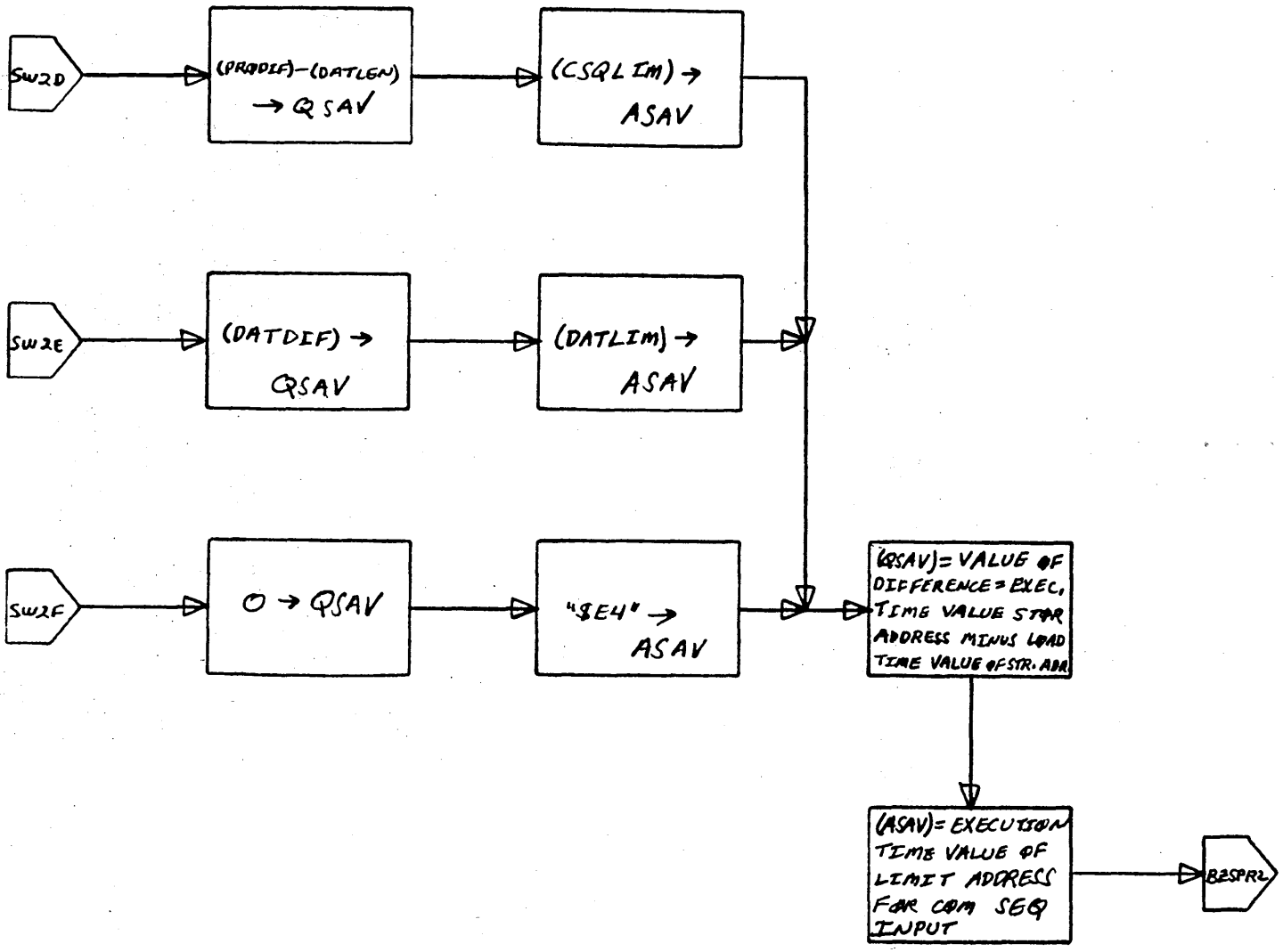


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	EQLPRG PAGE 4 OF 6	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

59.267

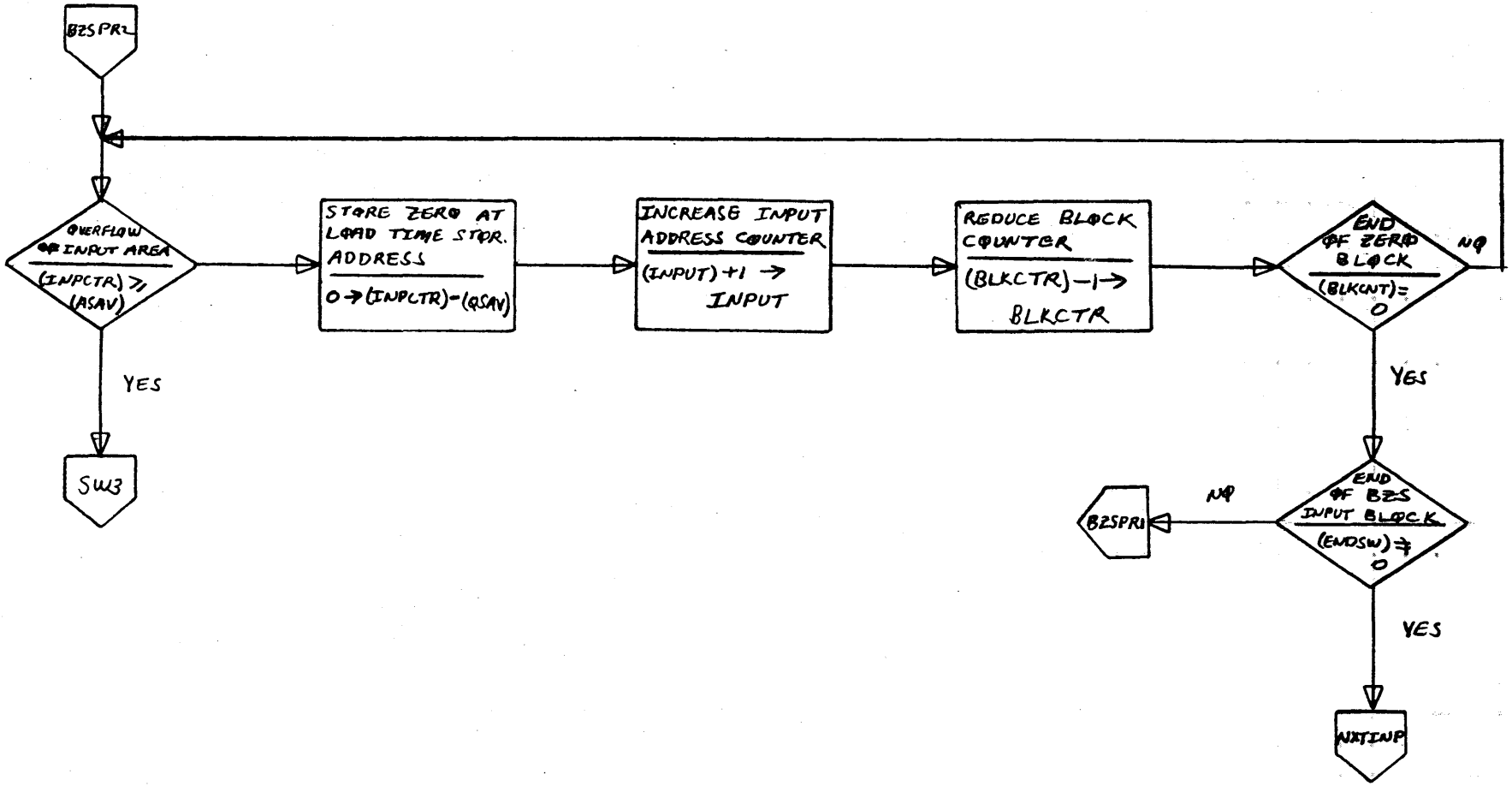


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	R00825 PAGE 4 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

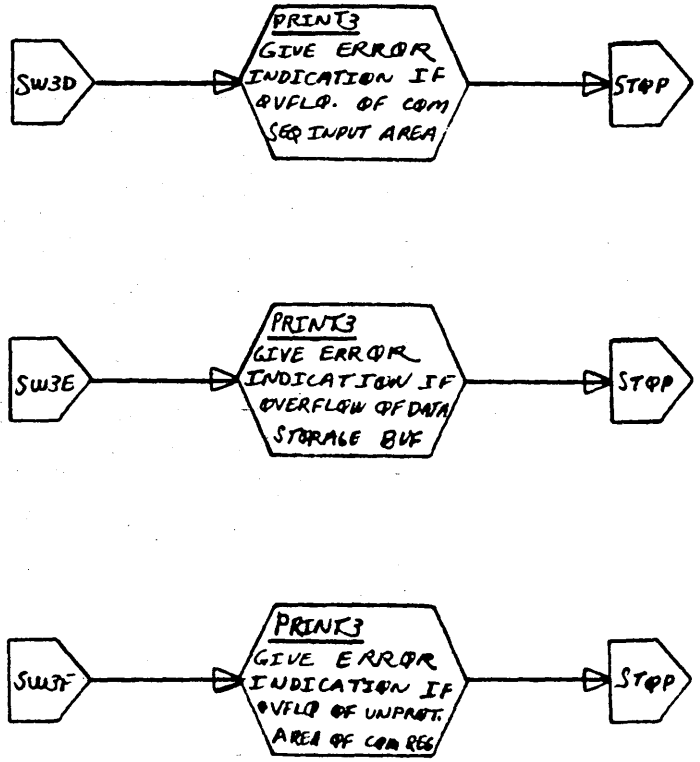
39.238



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR			
	RB0025 PAGE 8 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.239

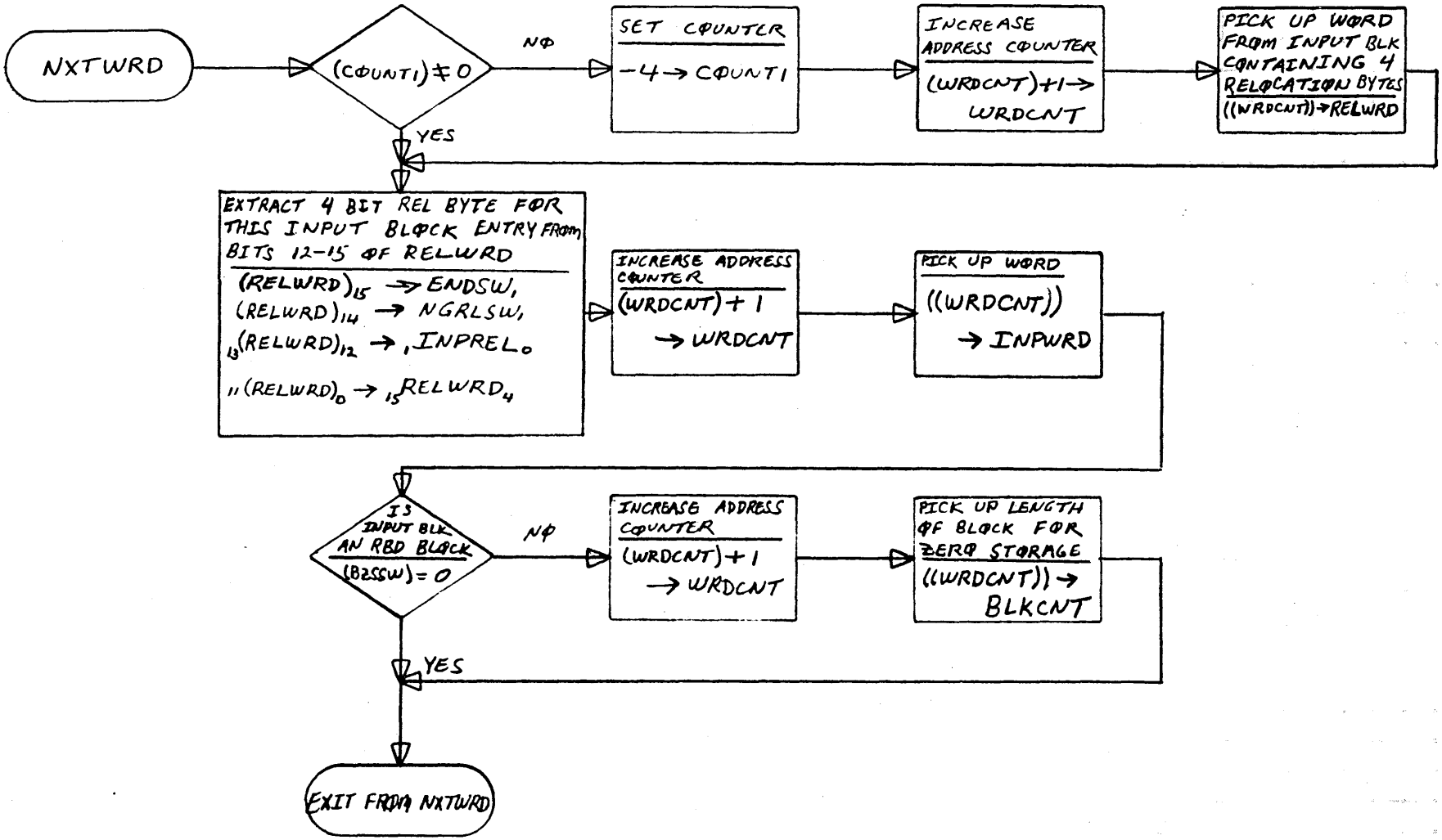


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	RBDB25 PAGE 9 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.240

AA1385 (FORMERLY CA127-1)

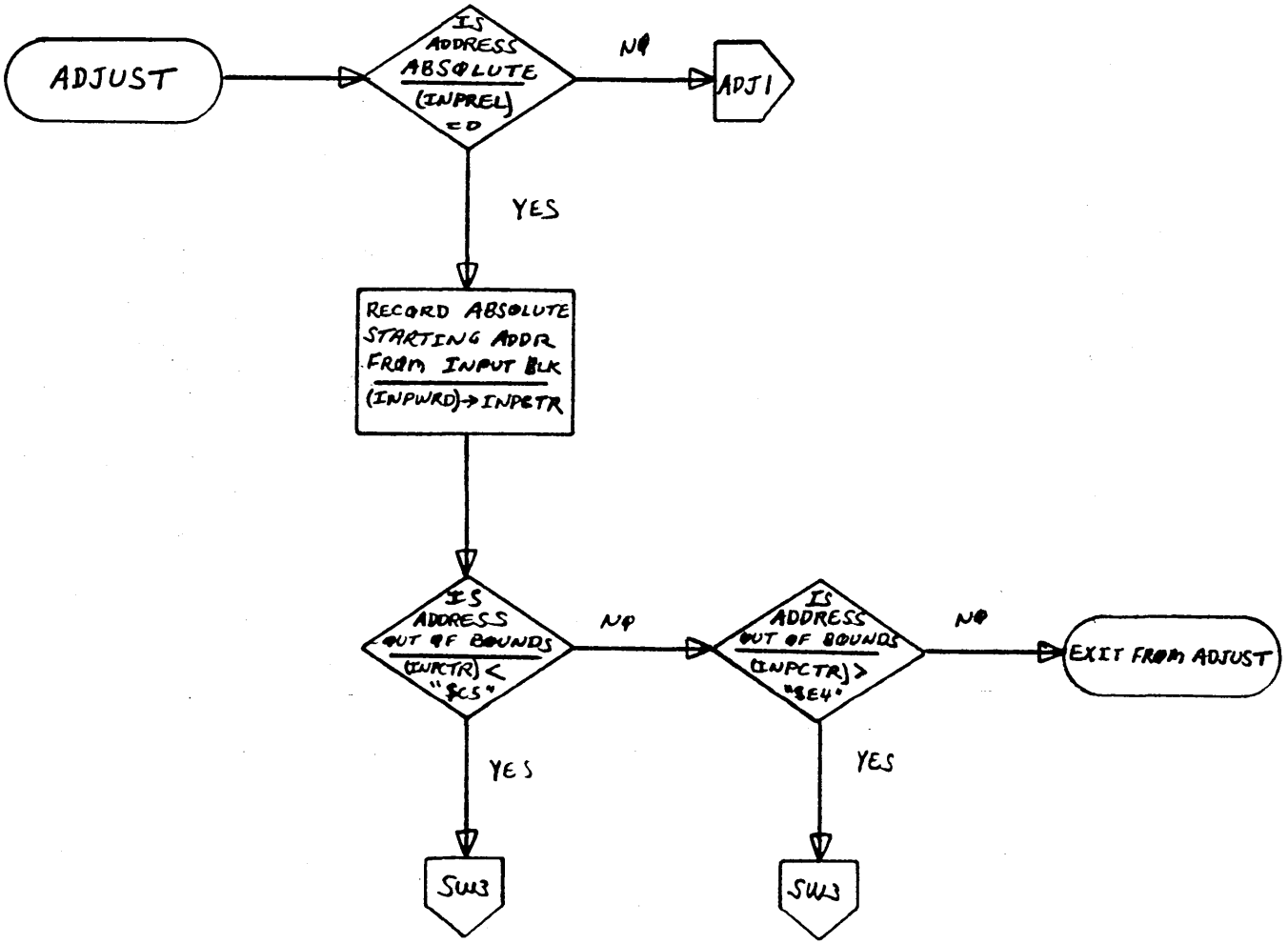


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	60CB25 PAGE 10 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMER A127-1)

MAR 5 1971

39,241

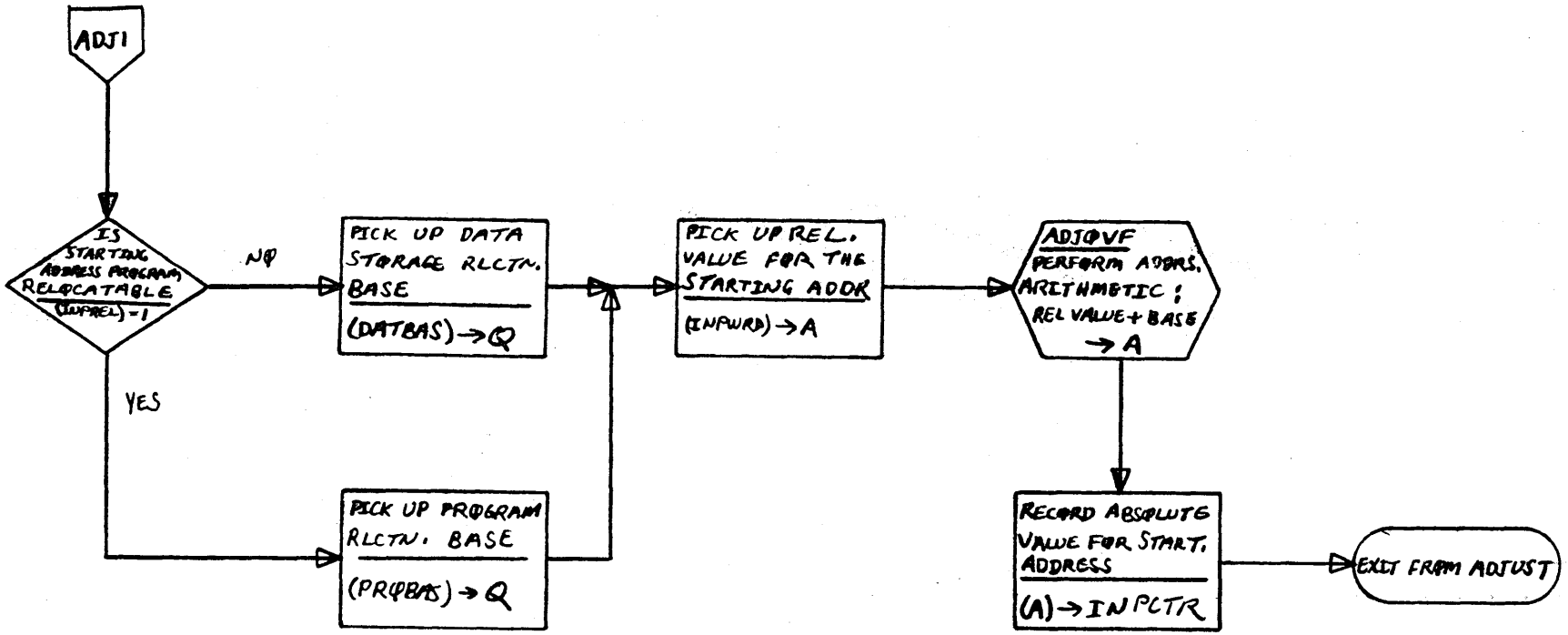


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	RBDBZS PAGE 11 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.242

AA1385 (FORMERLY CA127-1)

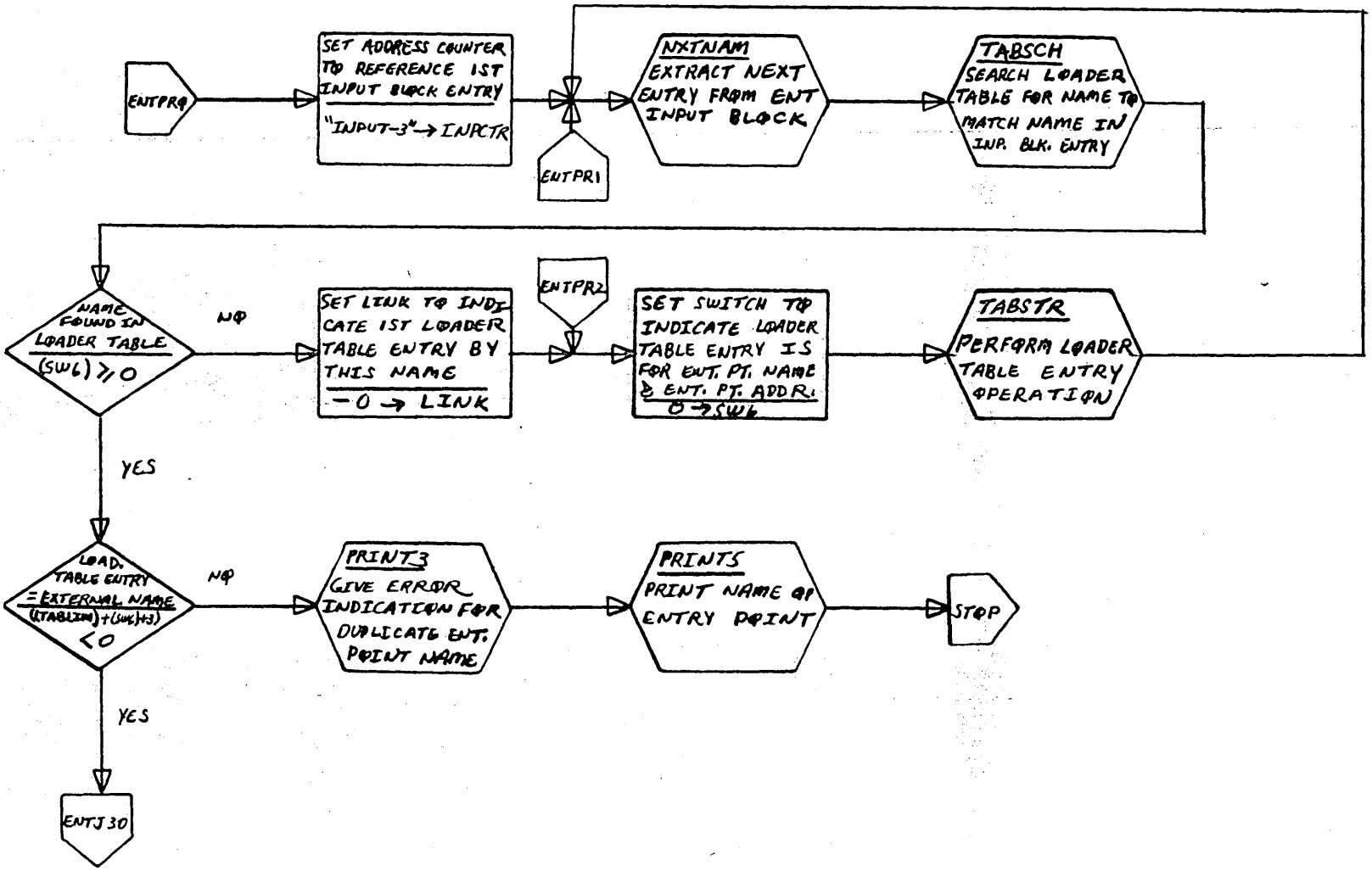


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	RBDB25 PAGE 12 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

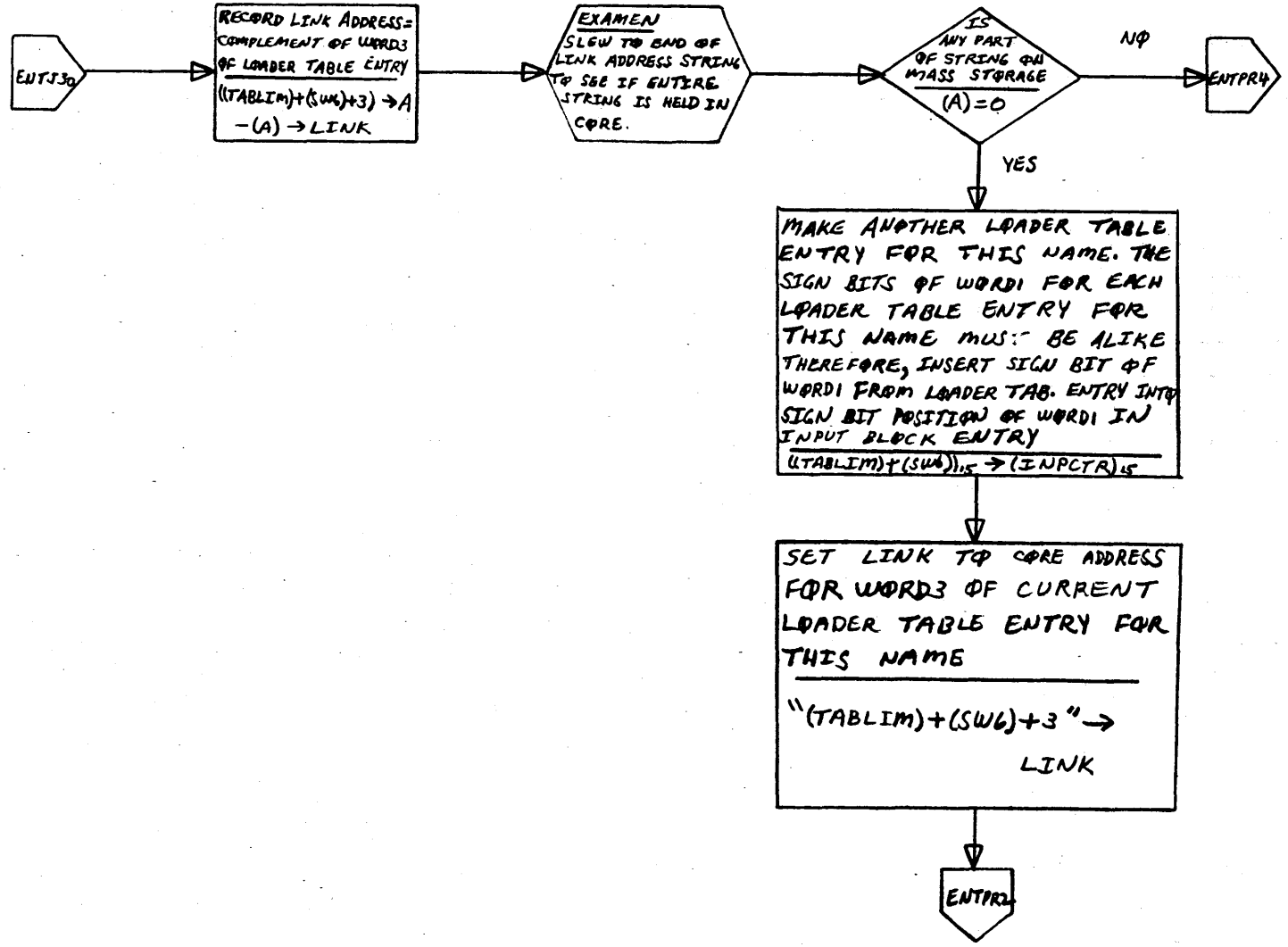
MAR 5 1971

39 .243



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.			
FLOWCHART ☐	ENTEXT	PAGE 1 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. GORDON	DATE 12 6 66	TASK NAME			

AA1385 (FORMERLY CA127-1)

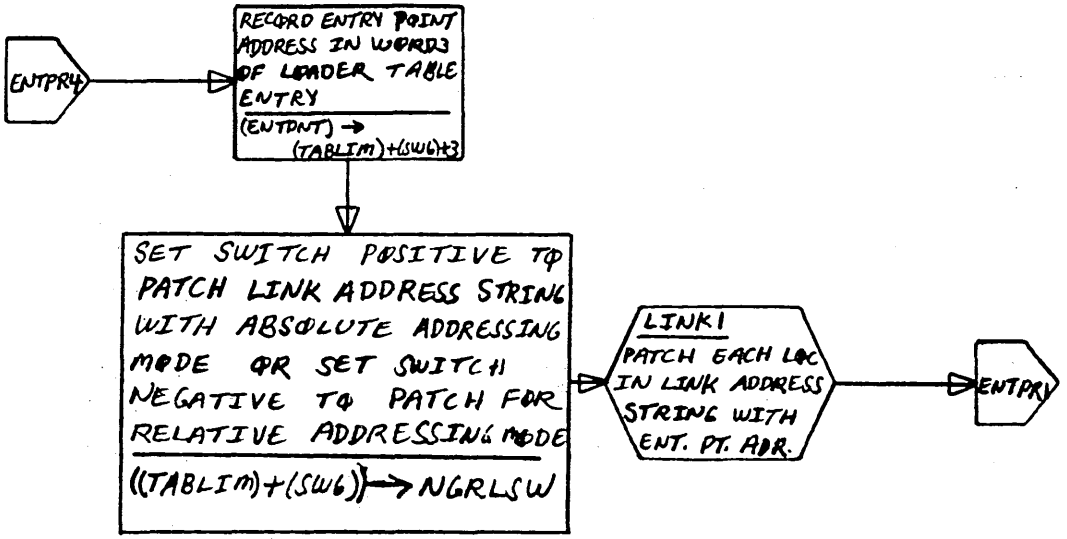


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	ENTEXT PAGE 2 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

59,245

AA1385 (FORMERLY CA127-1)

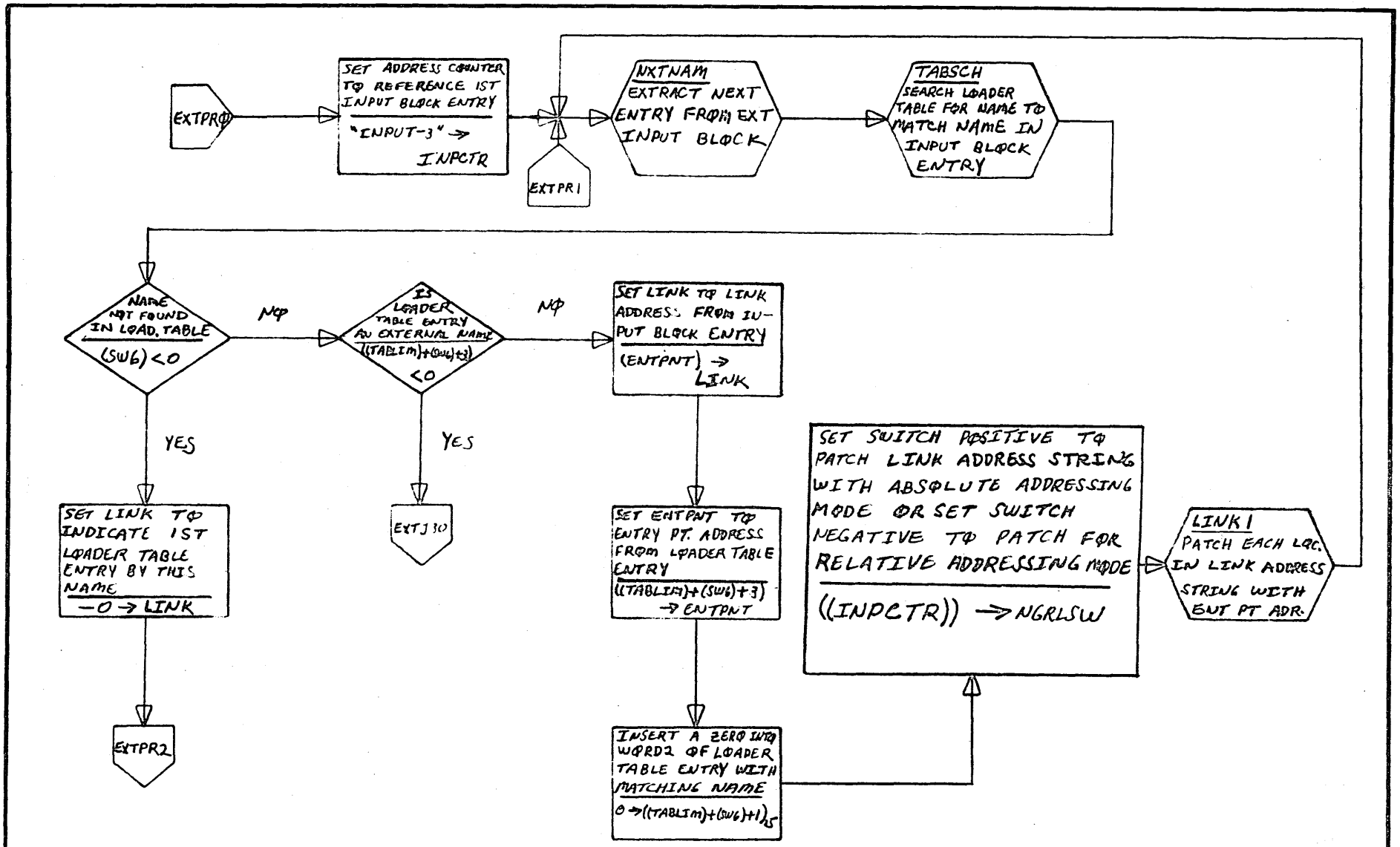


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	ENTEXT PAGE 3 OF 7	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

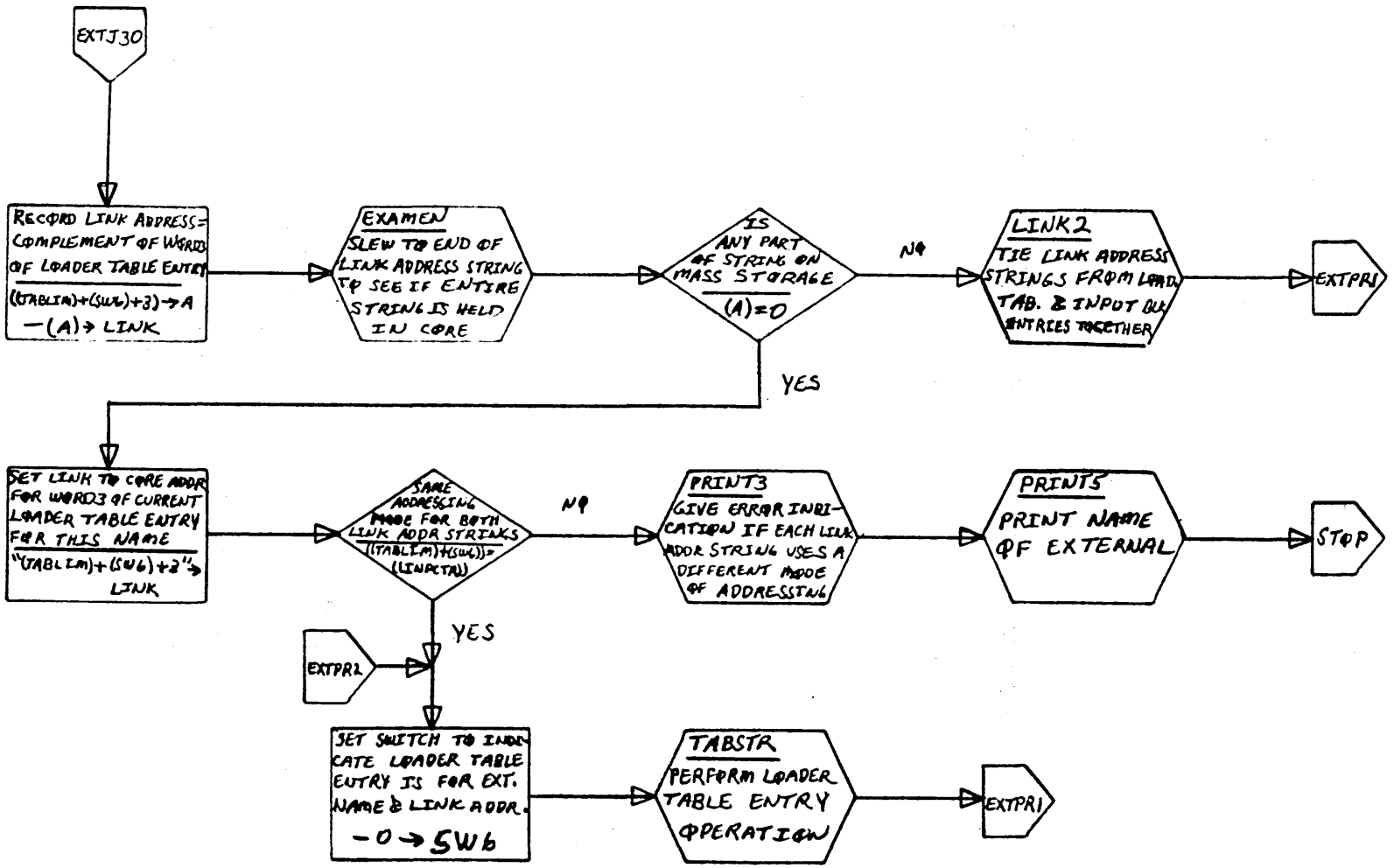
MAR 5 1971

39.24



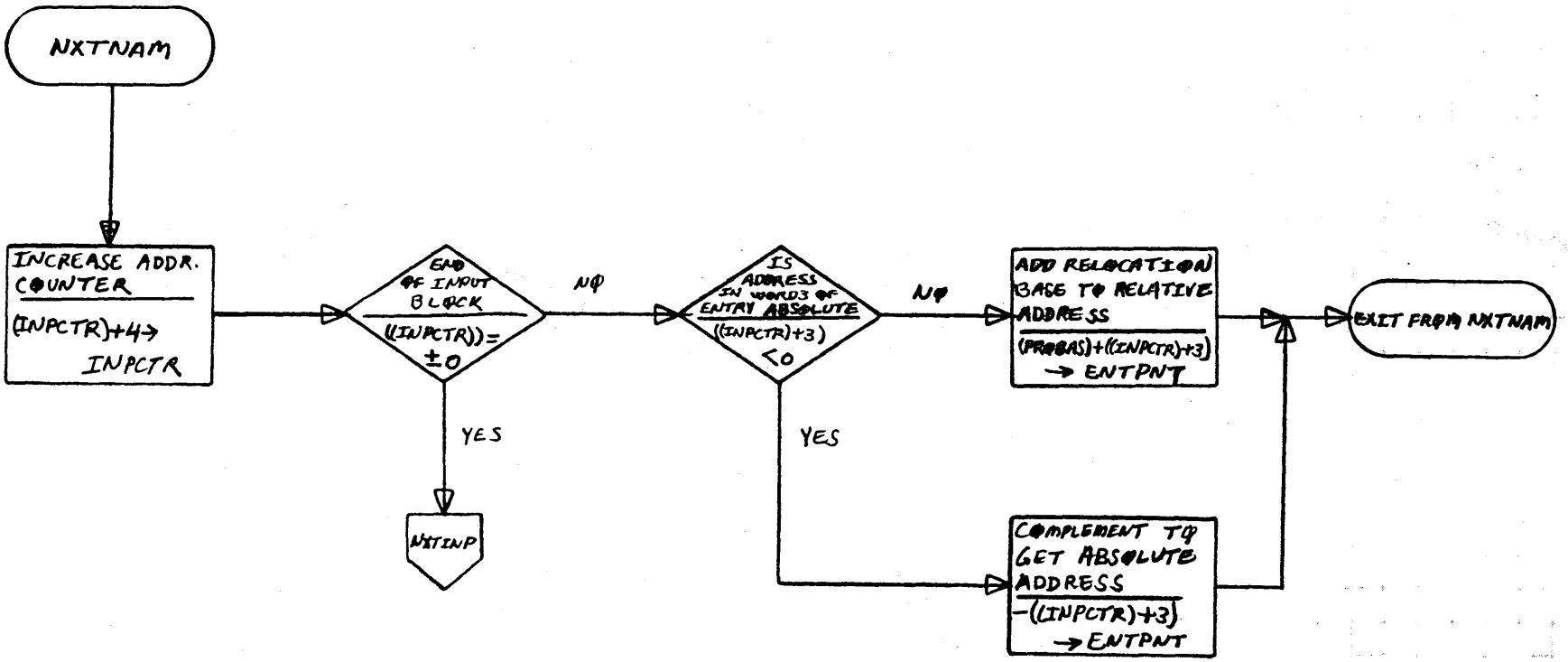
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	ENTEXT PAGE 4 OF 17	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	ENTEXT PAGE 5 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

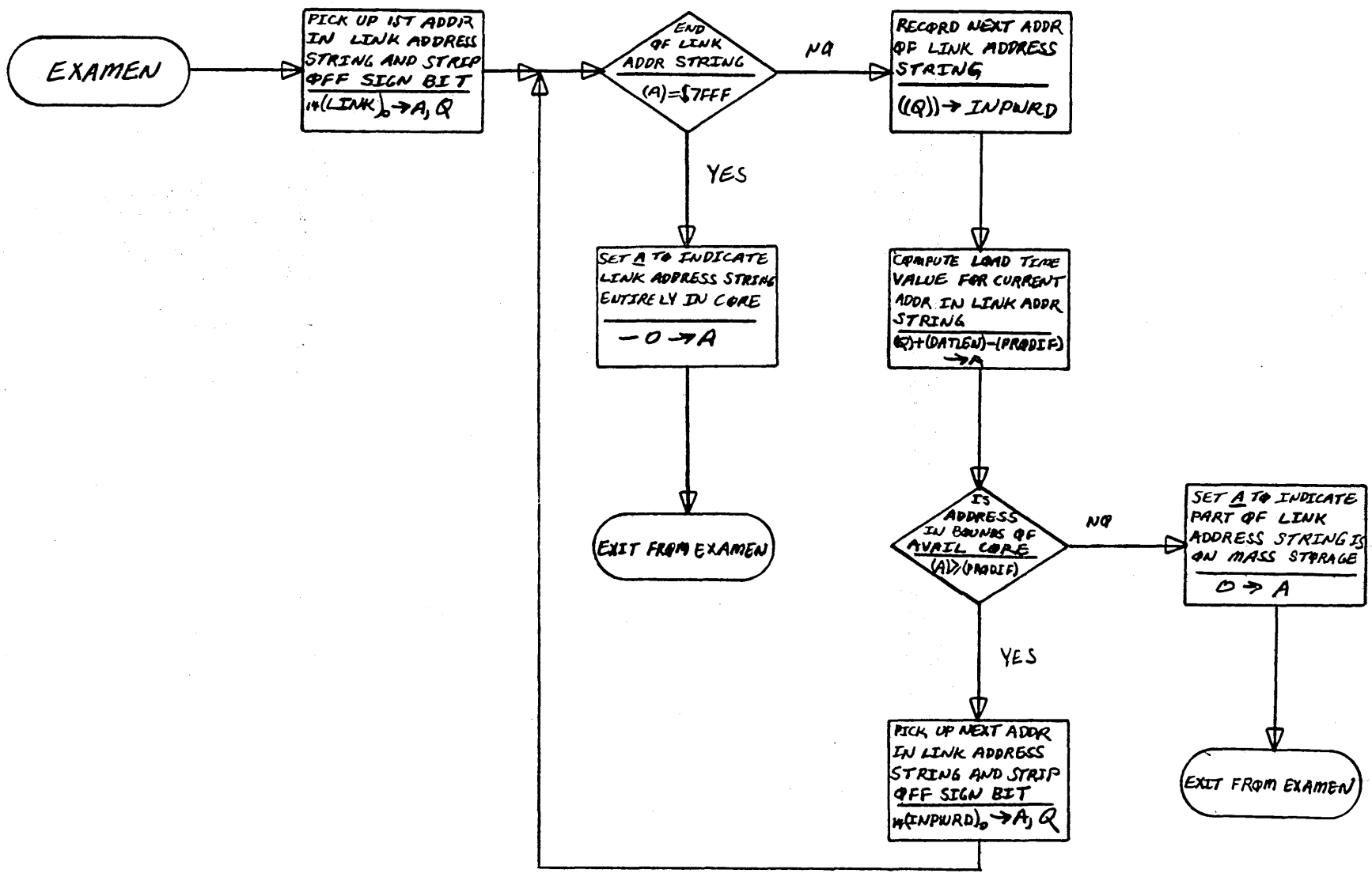


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	ENTEXT PAGE 6 OF 7	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.24.5

AA1385 (FORMERLY CA127-1

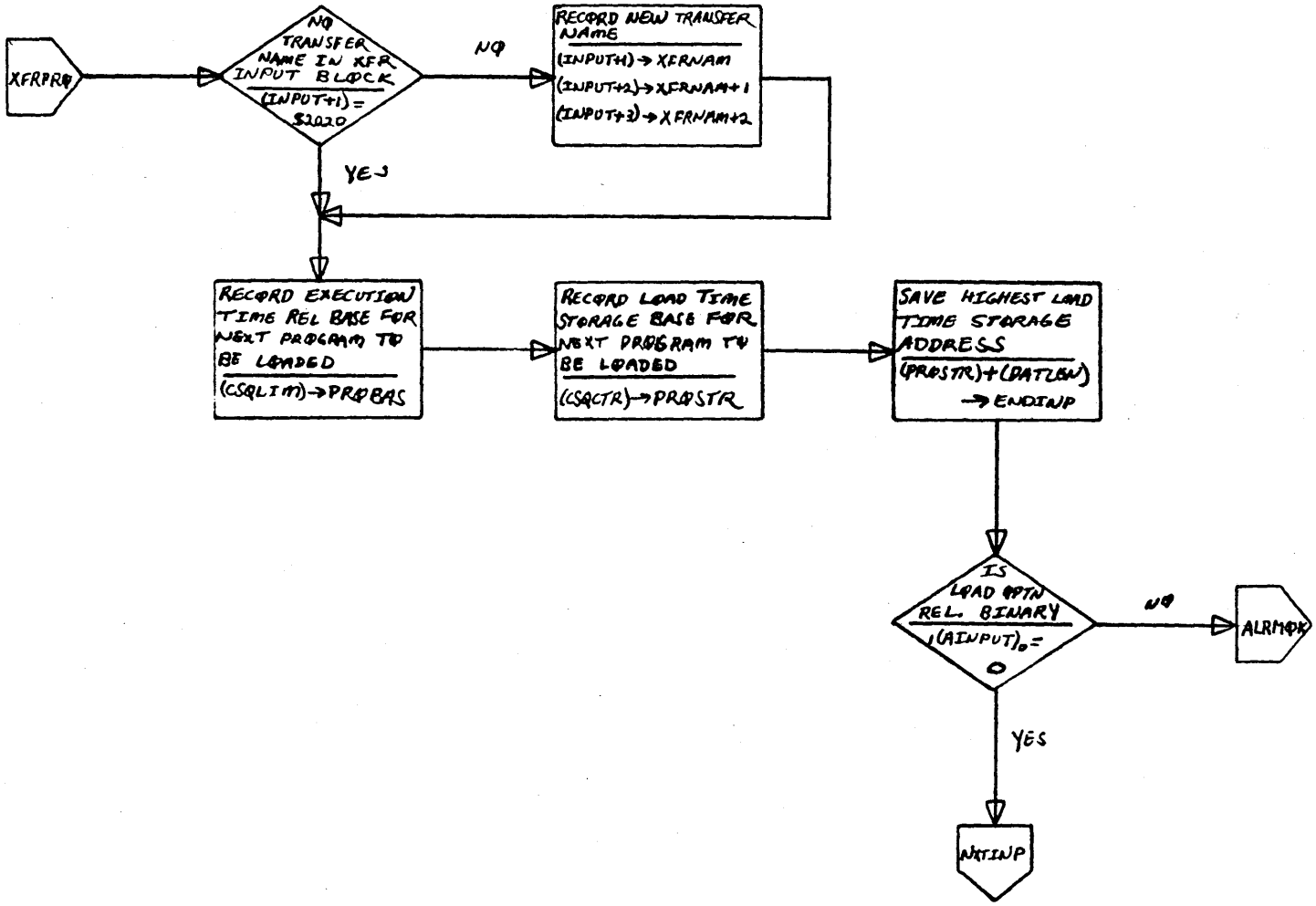


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	ENTEXT PAGE 1 OF 1	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

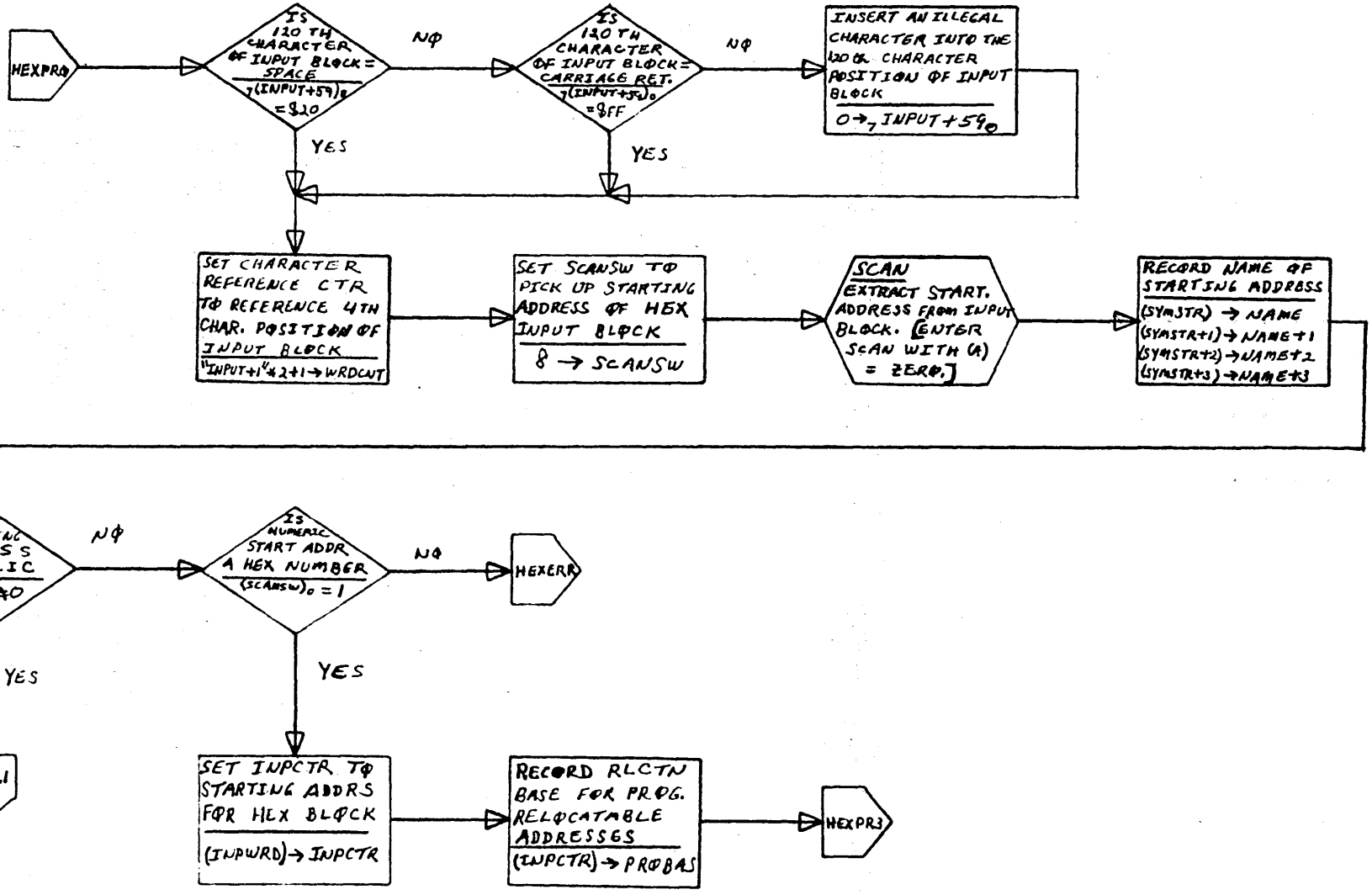
39.250

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS THS MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System	PROJECT MGR.			
FLOWCHART ☐	XFRPRQ PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY T. Gordon DATE 12-6-66	TASK NAME			

AA1385 (FORMERLY CA127-1)

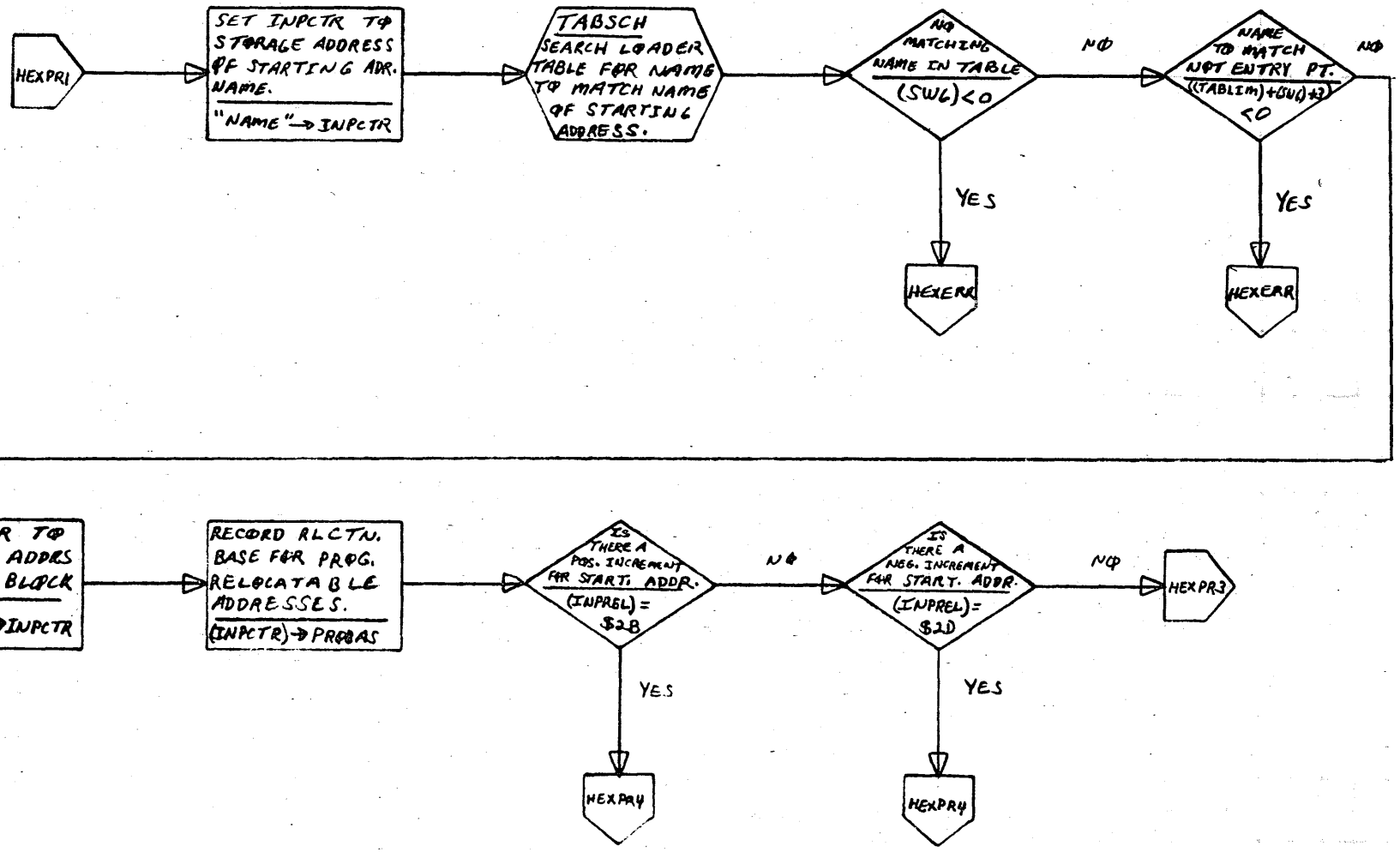


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IIIS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 Operating System		PROJECT MGR.		
FLOWCHART ☐	HEXPR0	PAGE 1 OF 12	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY T. Gordon	DATE 12-6-66	TASK NAME		

MAR 5 1971

39.252

AA1385 (FORMERLY CA127-1)

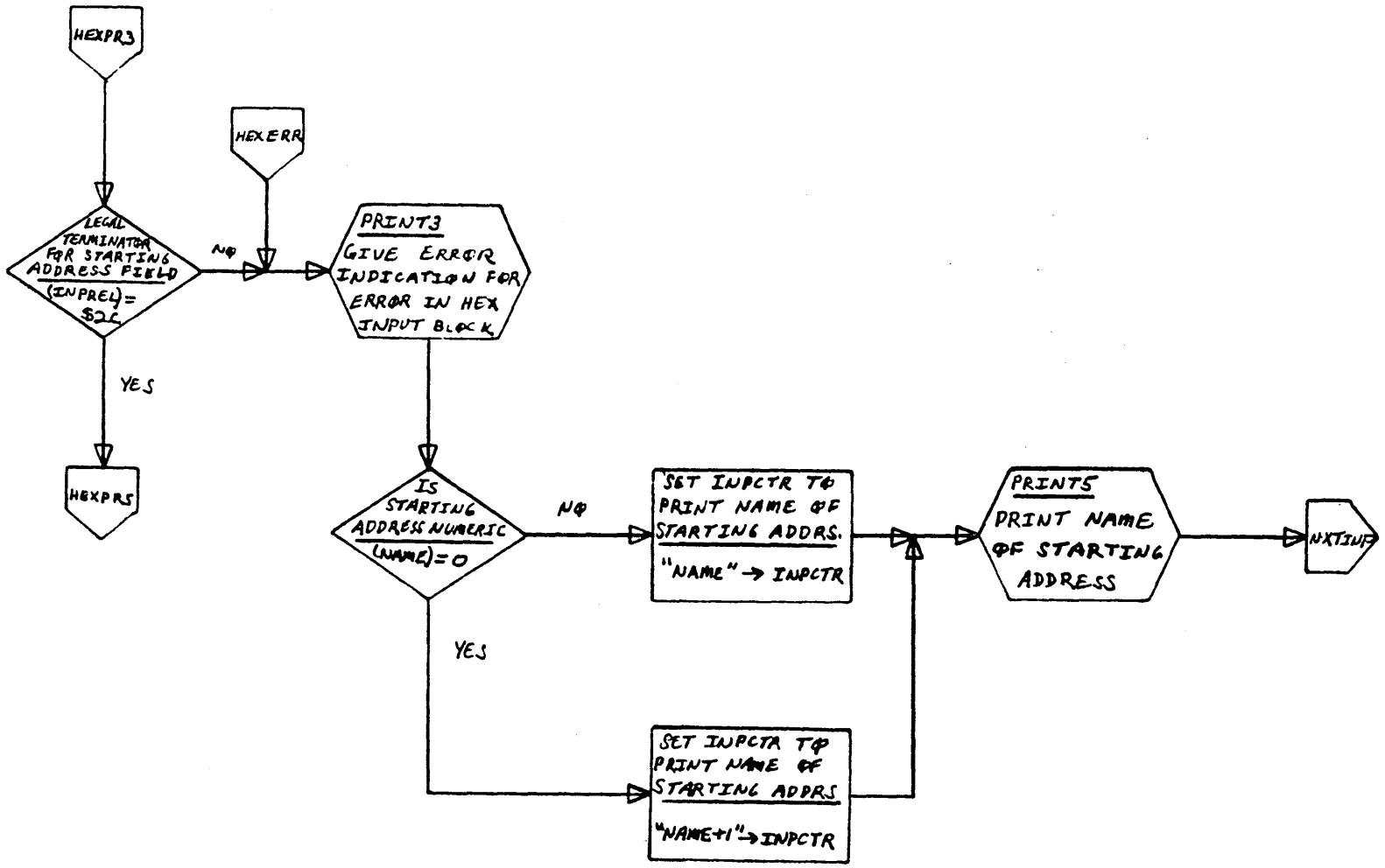


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE		PROJECT MGR.			
NEXPRO	PAGE 2 OF 12	PROJECT NAME			
NUMBER	ISSUE DATE	TASK NO.			
DRAWN BY	DATE	TASK NAME			

AA1385 (FORMER [illegible] A127-1)

MAR 5 1971

39.253

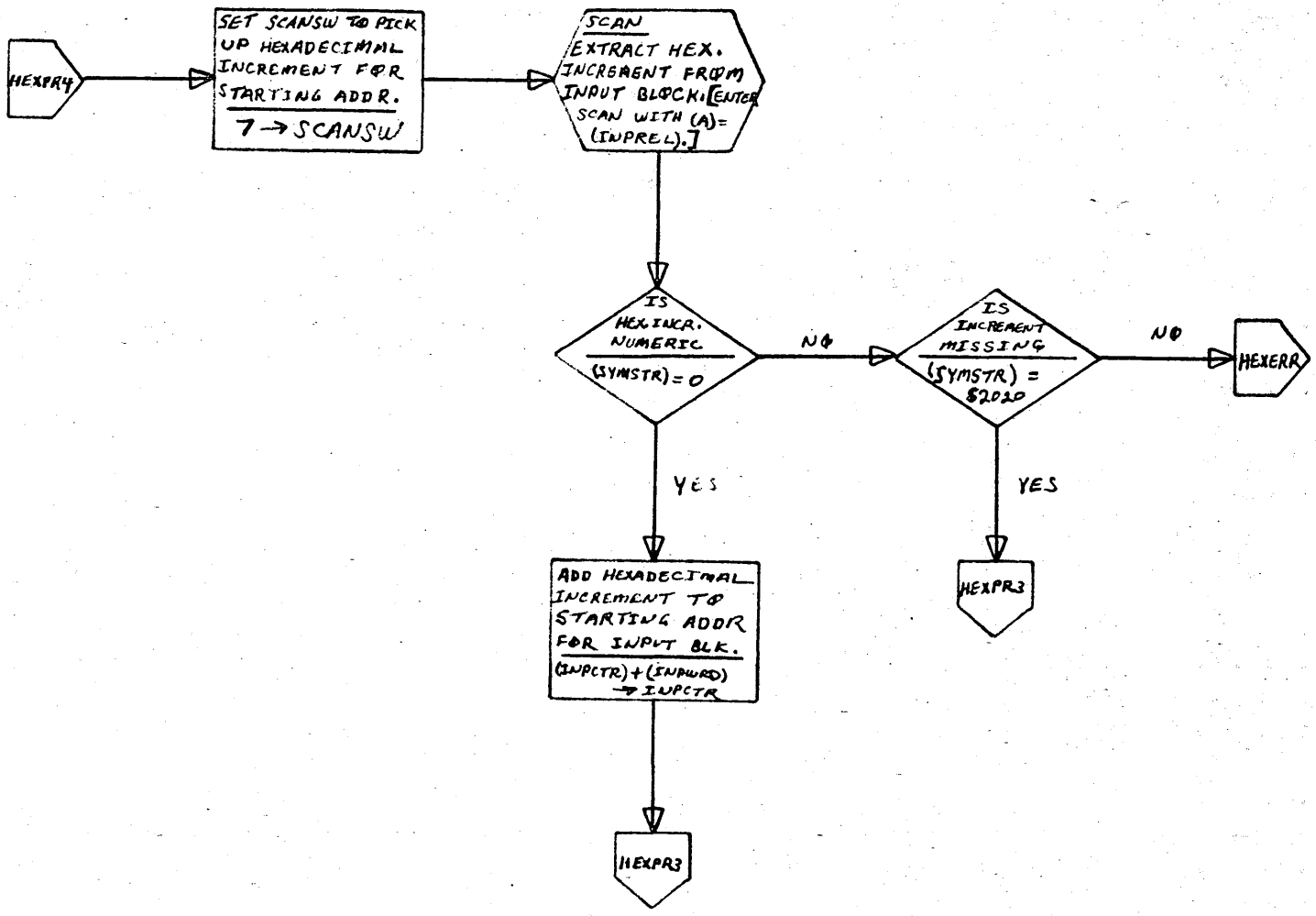


CONTROL DATA CORPORATION	DOCUMENT CLASS / MACH TYPE	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE	PROJECT MGR.			
SAMPLE CODE ☐	HEXPR0 PAGE 3 OF 12	PROJECT NAME			
FLOWCHART ☐	NUMBER / ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY / DATE	TASK NAME			
OTHER ☐					

MAR 5 1971

39,254

AA1385 (FORMERLY CA127-1)

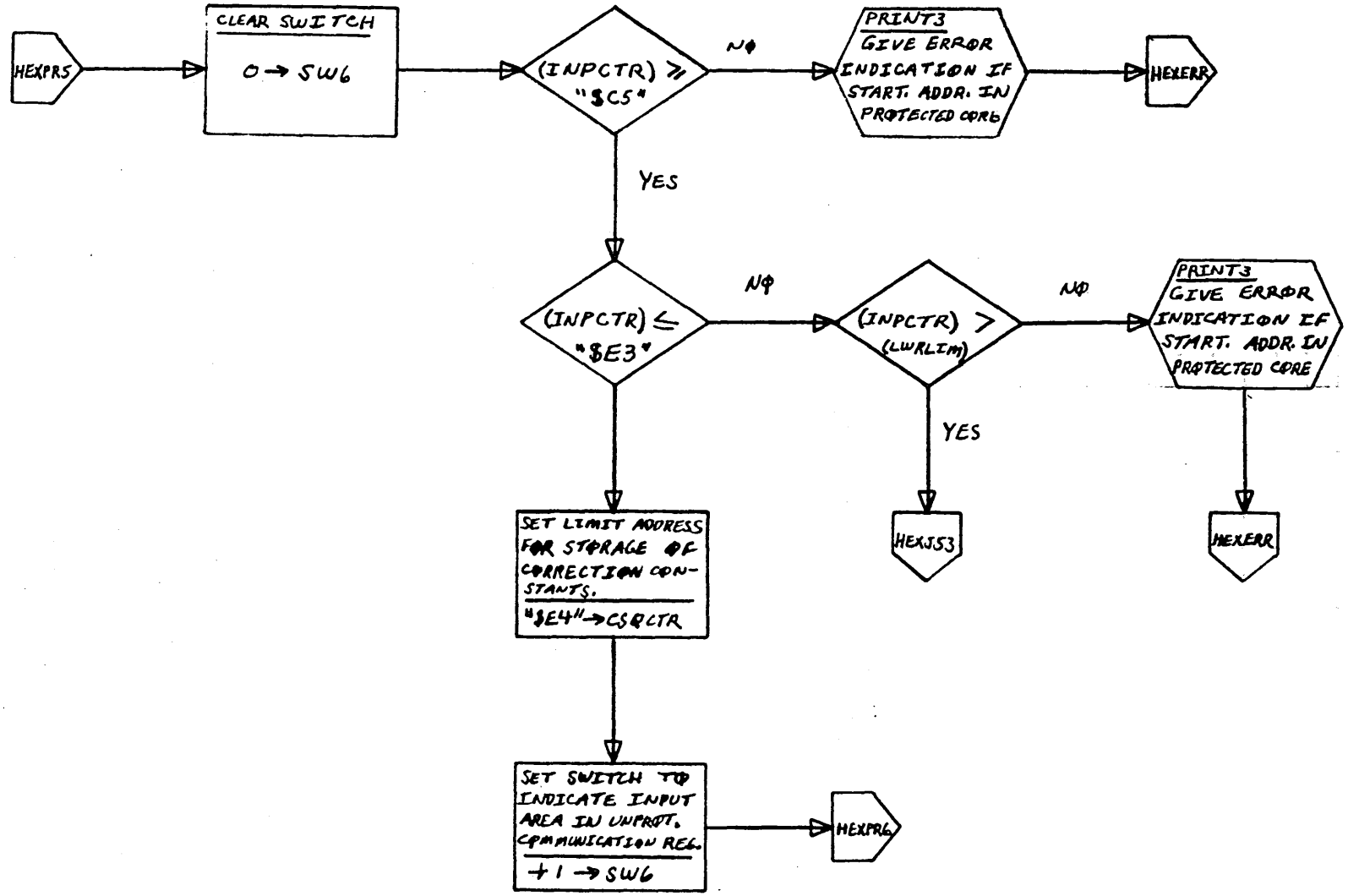


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	HEXPR0 PAGE 4 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.255

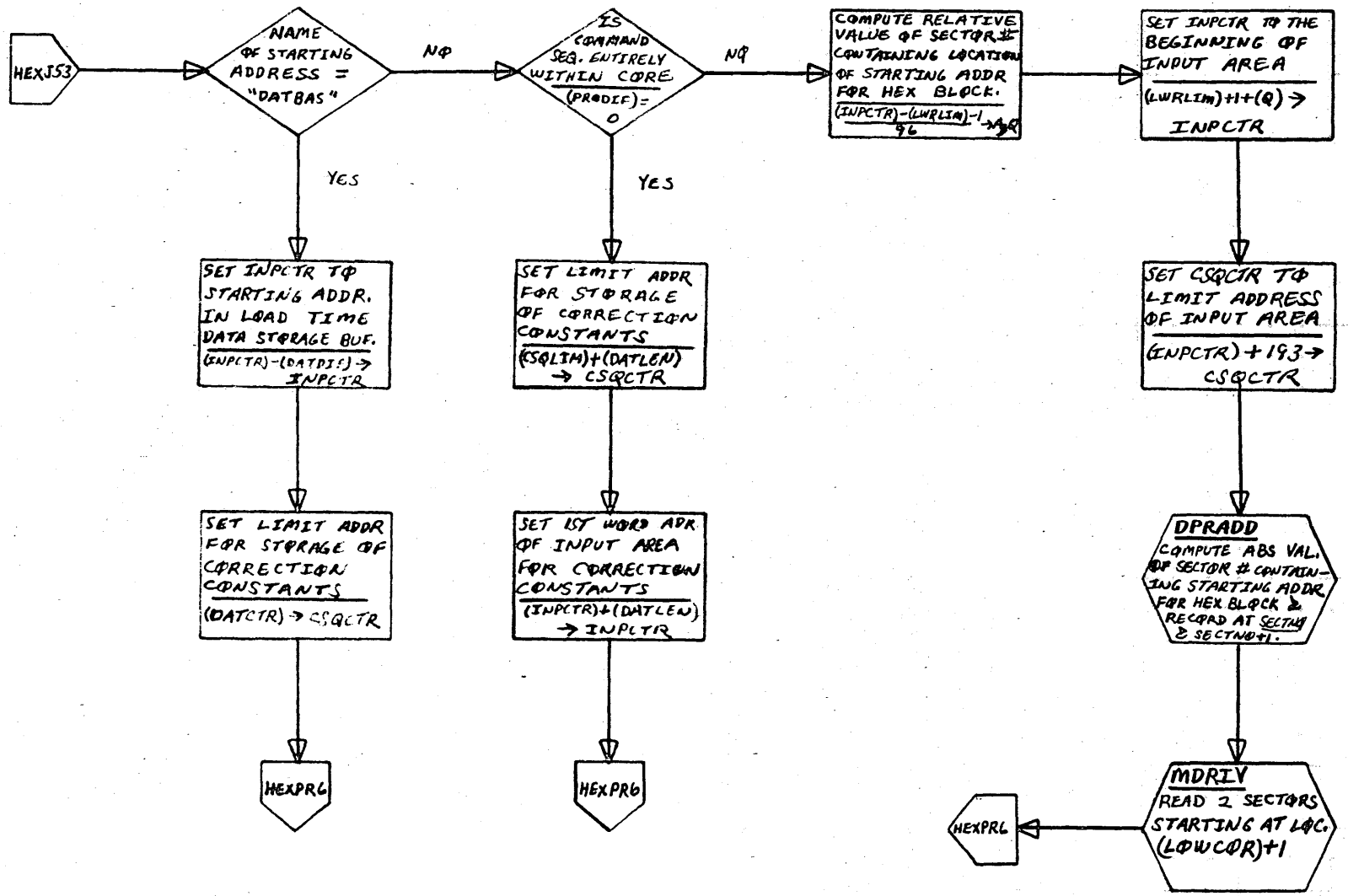
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	HEXPR0 PAGE 5 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.256

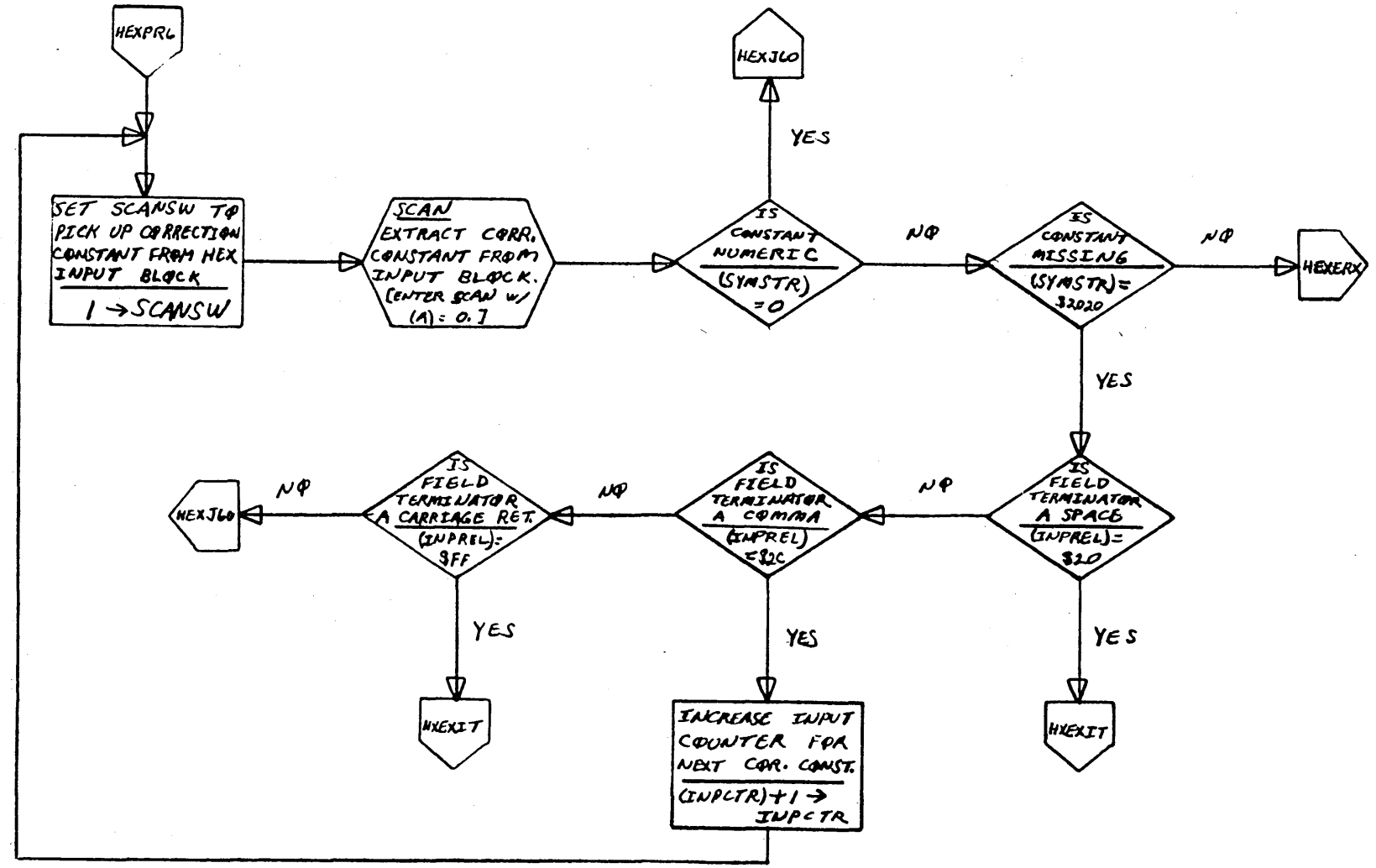


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE		PROJECT MGR.			
HEXPRG	PAGE 6 OF 12	PROJECT NAME			
NUMBER	ISSUE DATE	TASK NO			
DRAWN BY	DATE	TASK NAME			

MAR 5 1971

39.257

AA1385 (FORMERLY CA127-1)

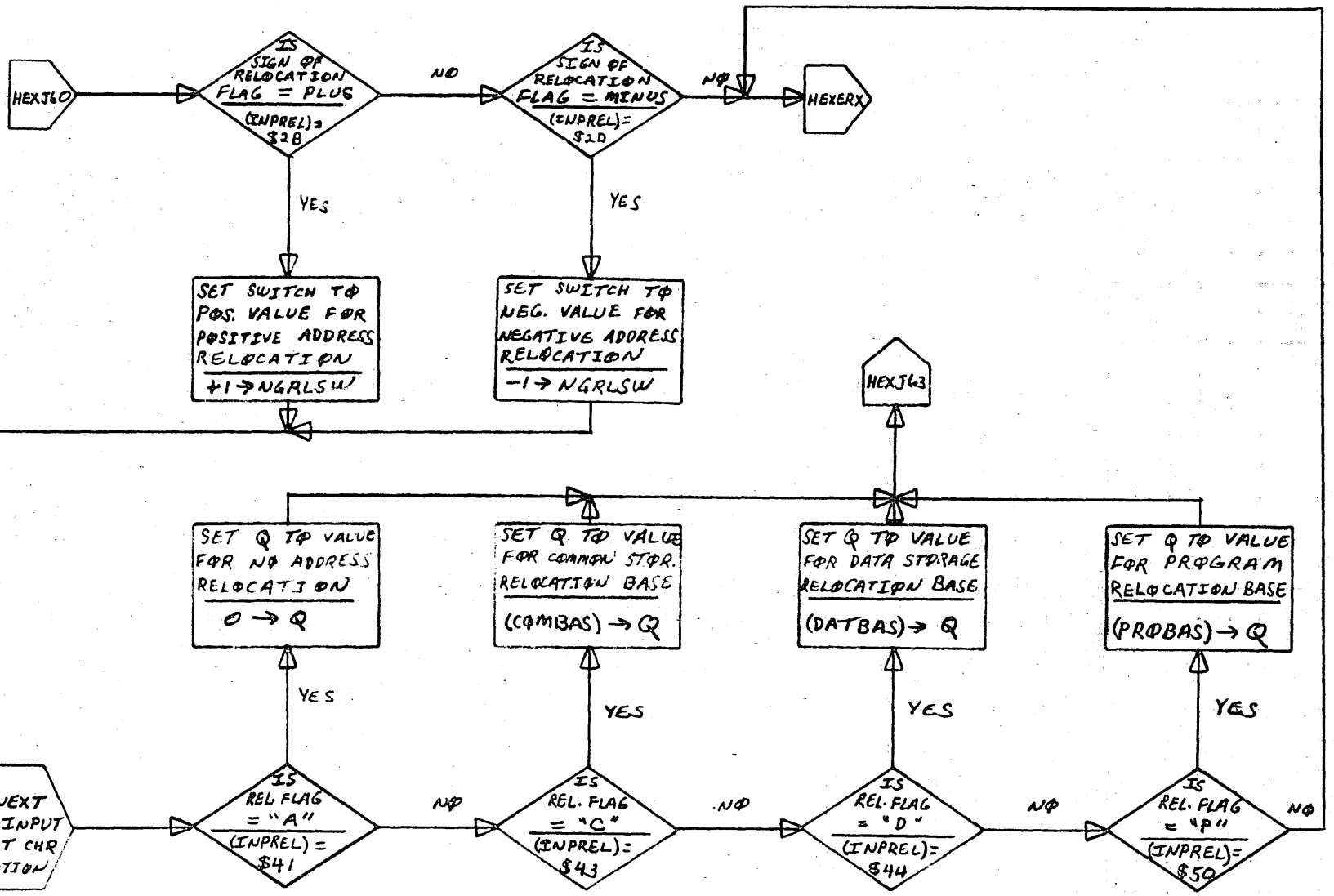


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	REVISED PAGE 7 OF 12	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

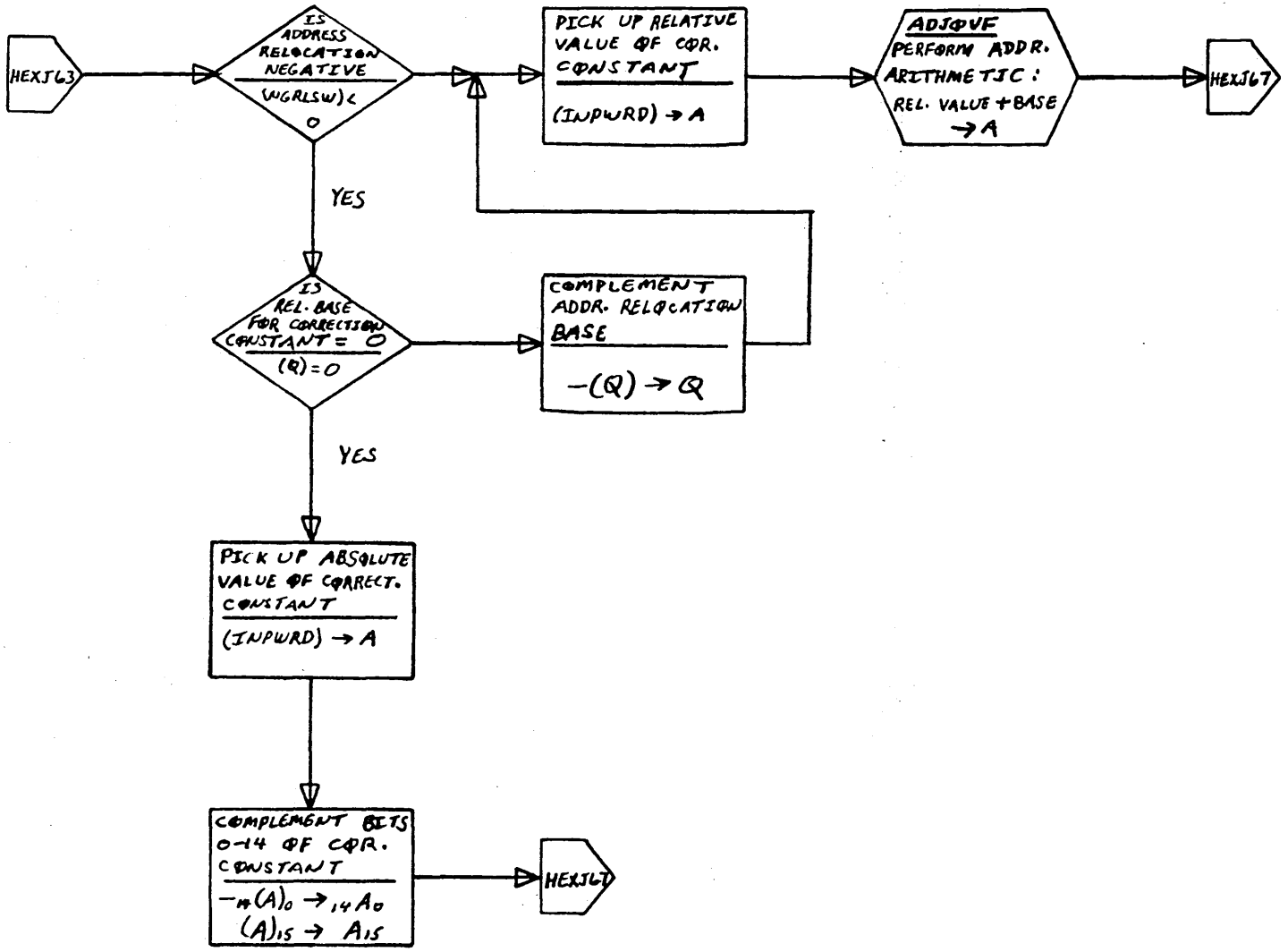
39 .258

AA1385 (FORMERLY CA127-1)



MAR 5 1971

CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	NF4118 PAGE 8 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

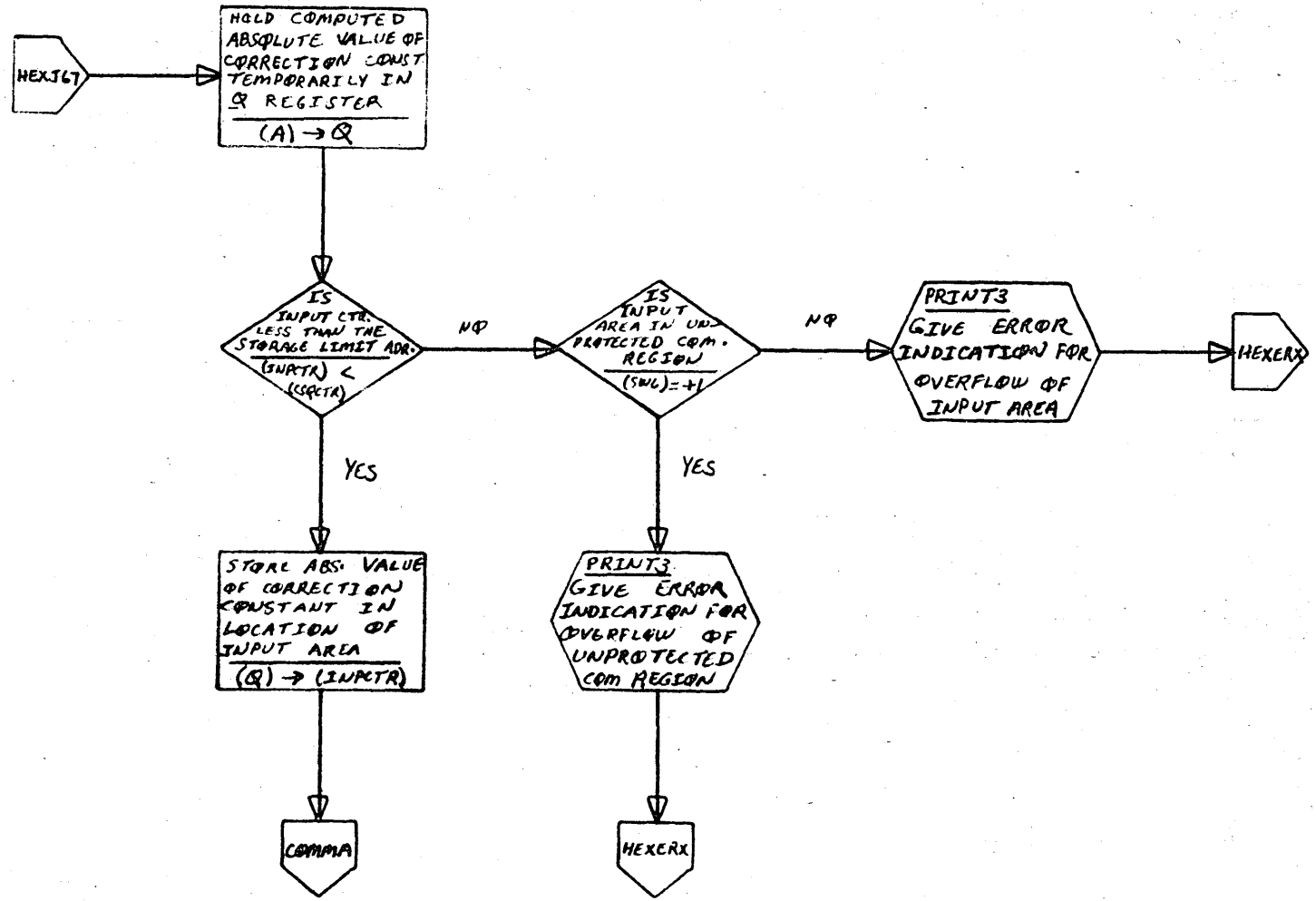


CONTROL DATA CORPORATION	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE		PROJECT MGR.			
SAMPLE CODE ☐		HEXJ64 PAGE 9 OF 12	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

MAR 5 1971

39.260

AA1385 (FORMERLY CA127-1)

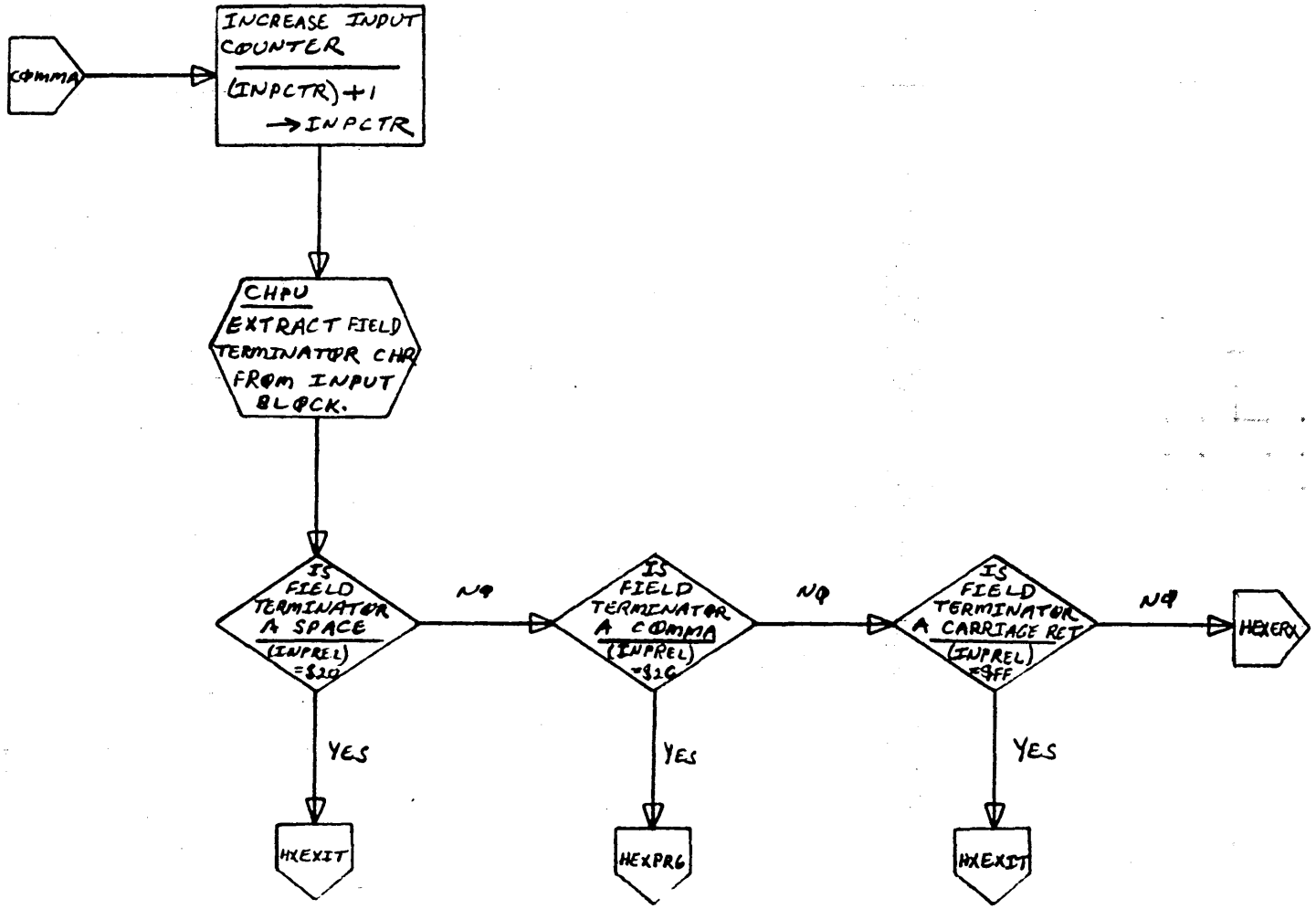


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	NUMBER PAGE 10 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.261

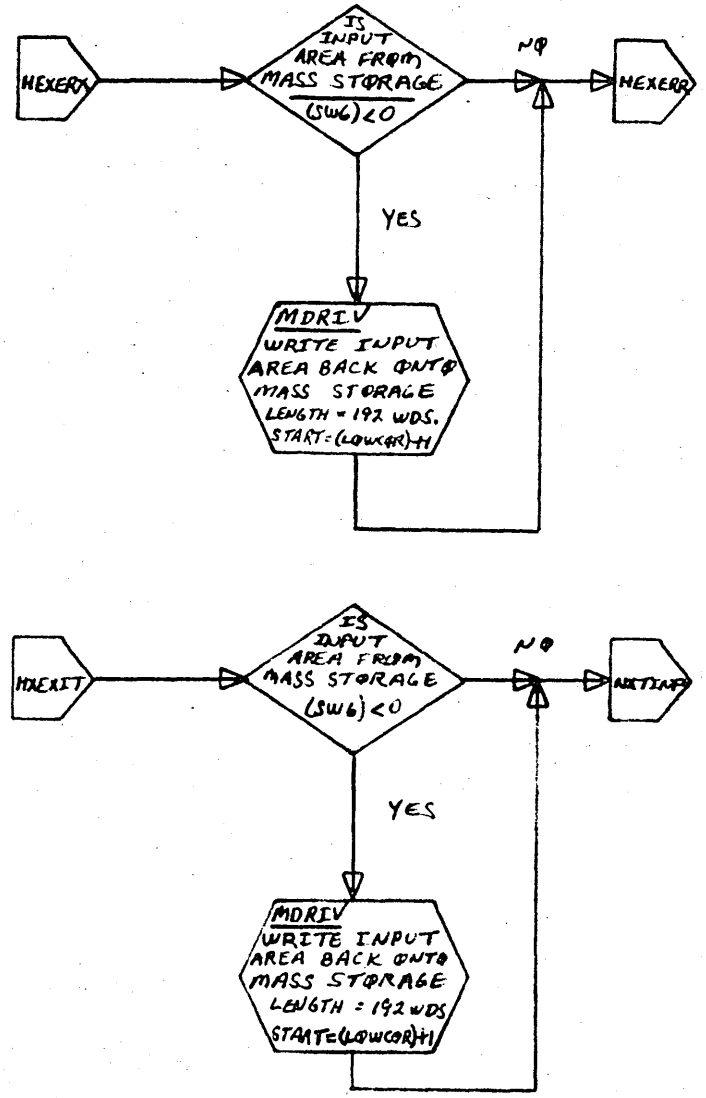
AA1385 (FORMER L A127-1)



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐☐	HF4880 PAGE 11 OF 12	PROJECT NAME			
DECISION TABLE ☐☐	NUMBER / ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

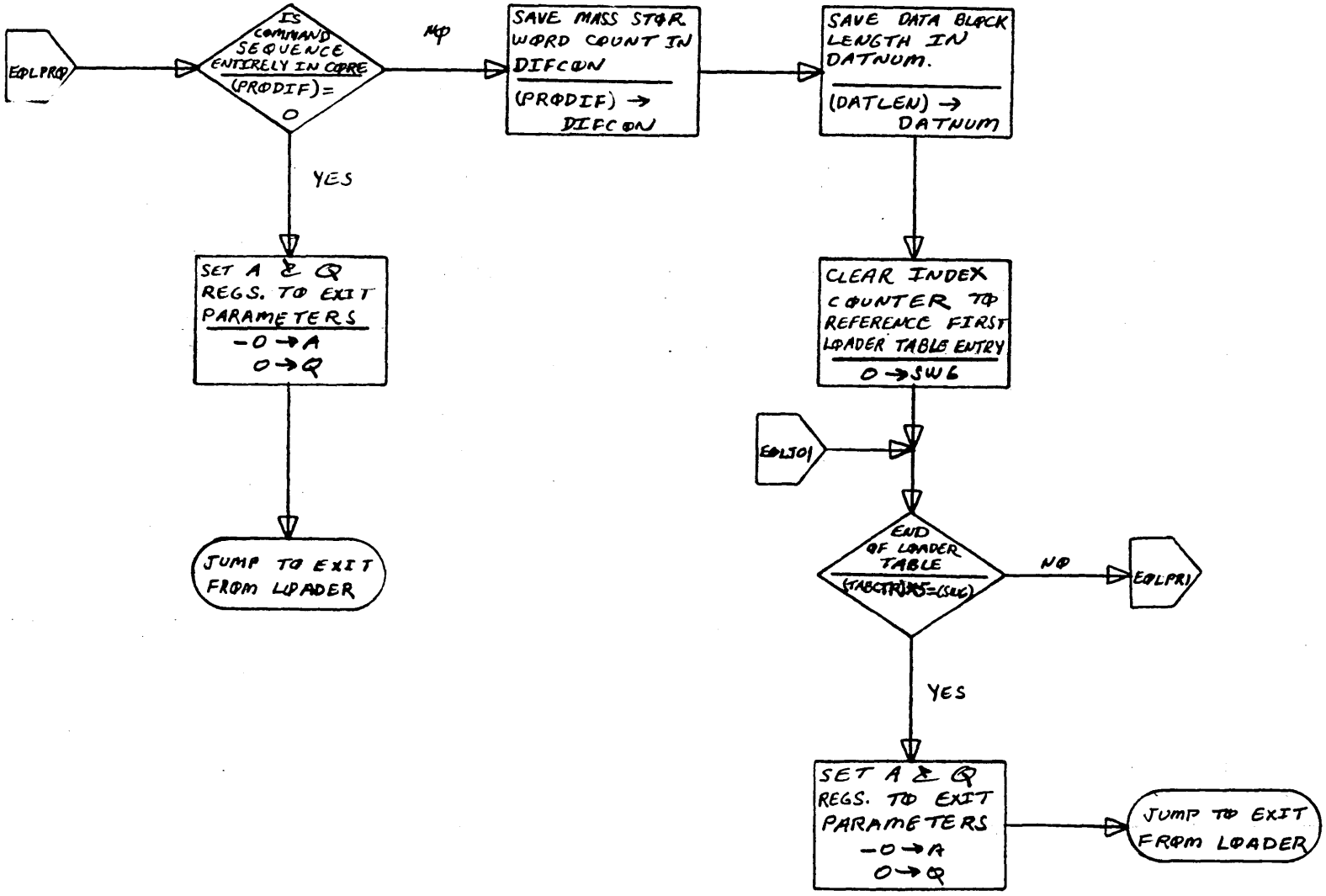
AA1385 (FORMERLY CA127-1)

MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE	PROJECT MGR.			
	NF4PP.0 PAGE 12 OF 12	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

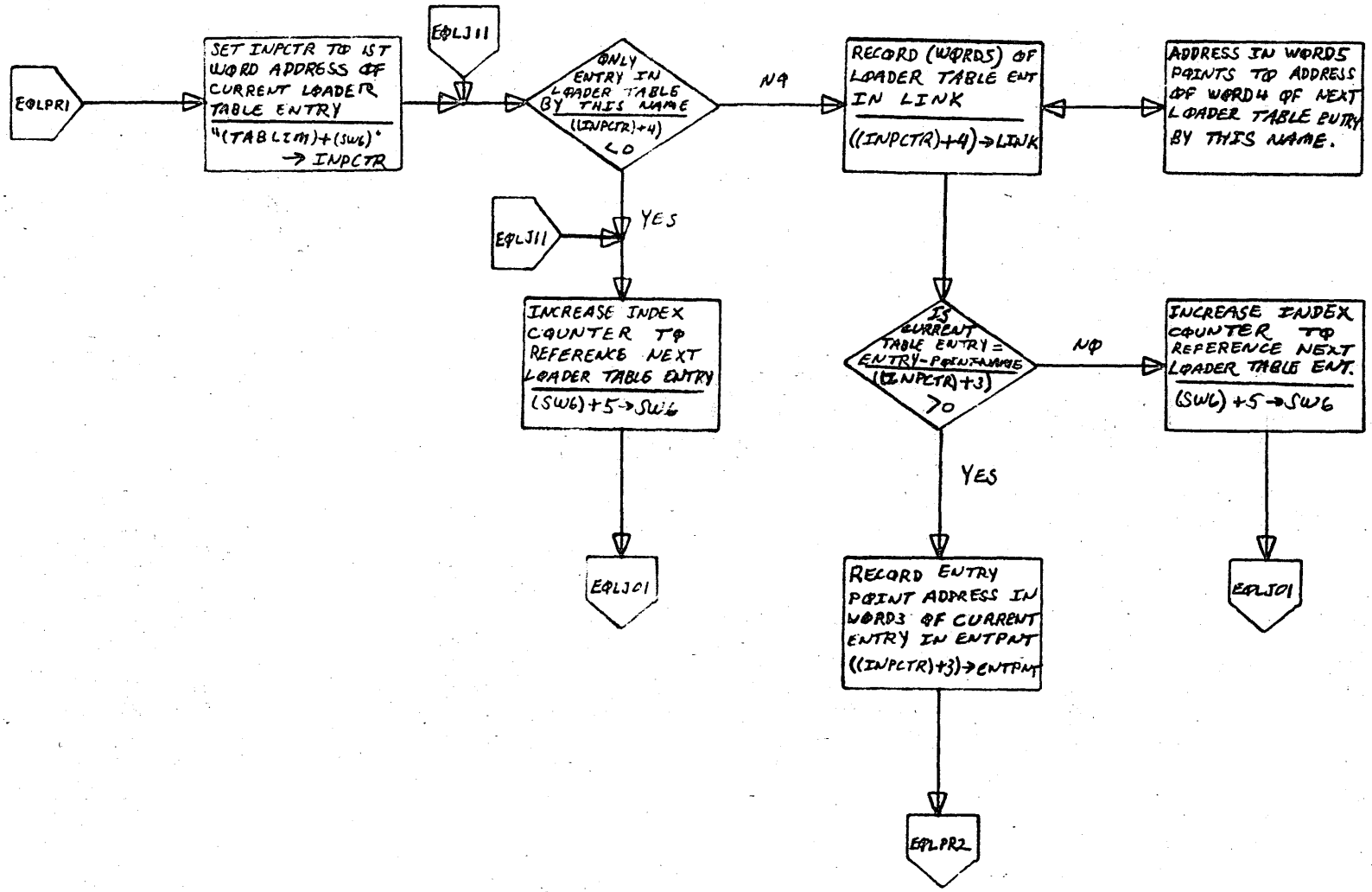


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS 3116 MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1700 OPERATING SYSTEM	PROJECT MGR.			
FLOWCHART ☐	EQLPRQ PAGE 1 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

39.264

AA1385 (FORMERLY CA127-1)

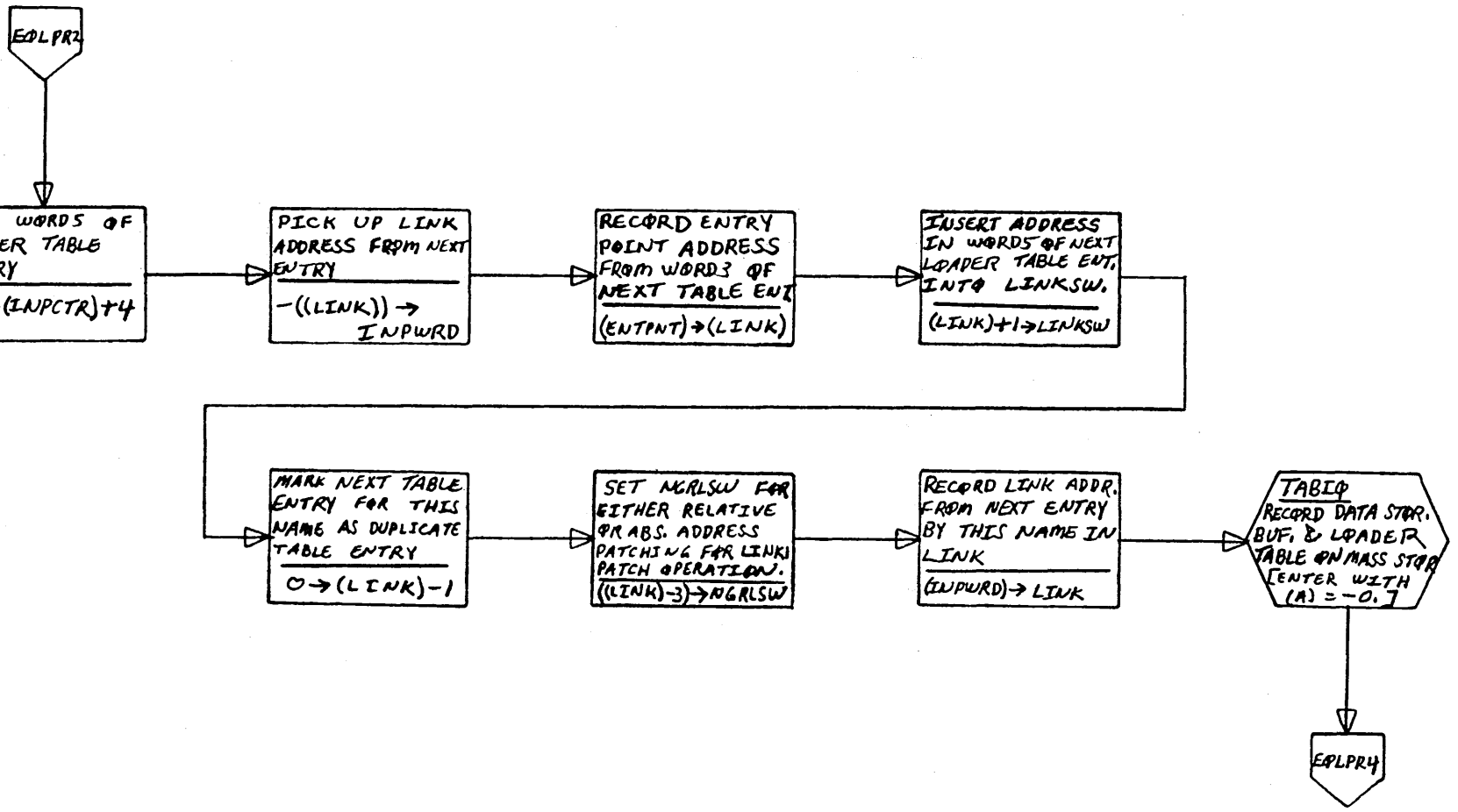


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	EQLPR PAGE 3 OF 6	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

39.265

AA1385 (FORMERLY CA127-1)

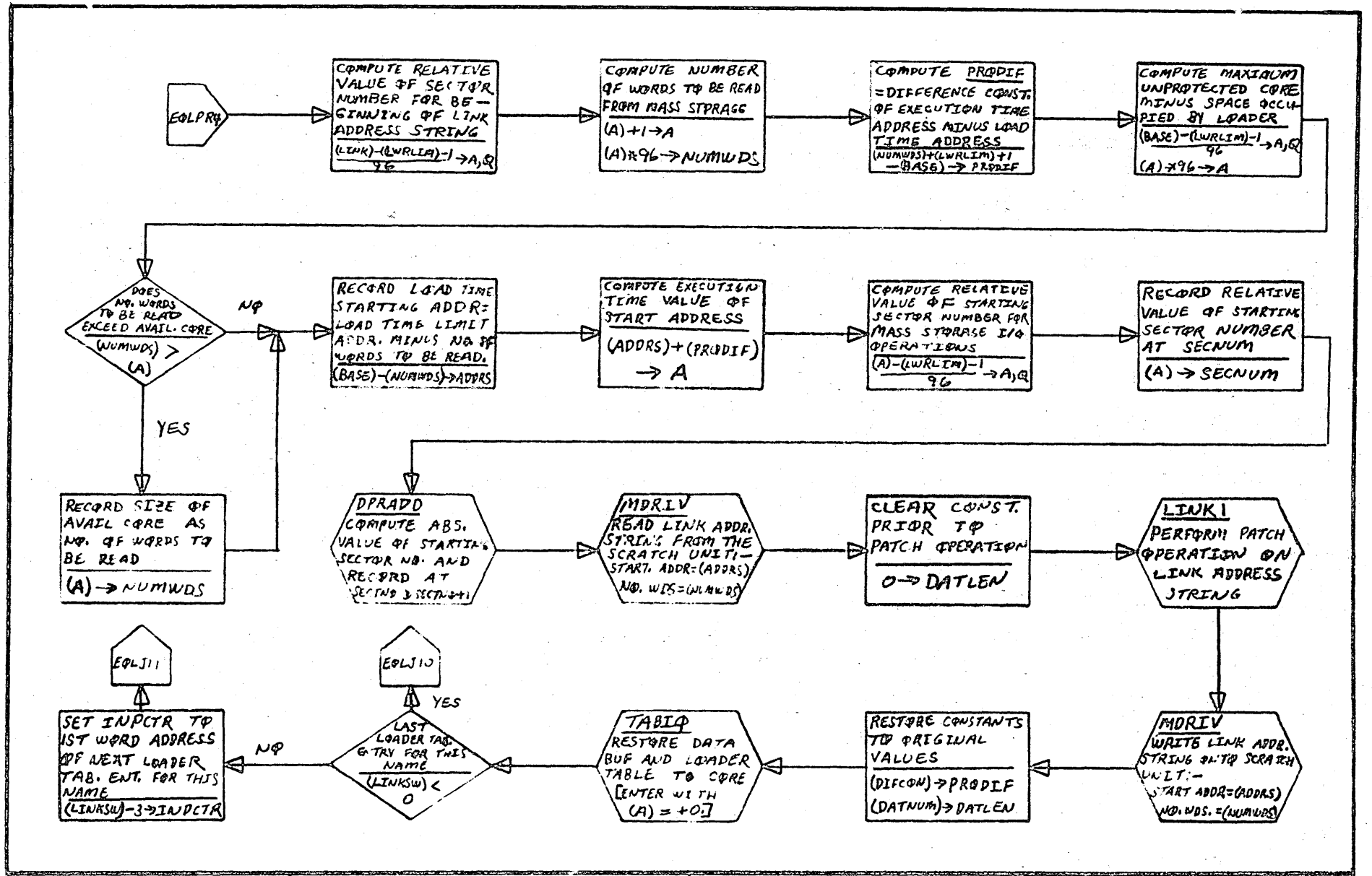


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	EØLAR1Ø PAGE 3 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

39.26

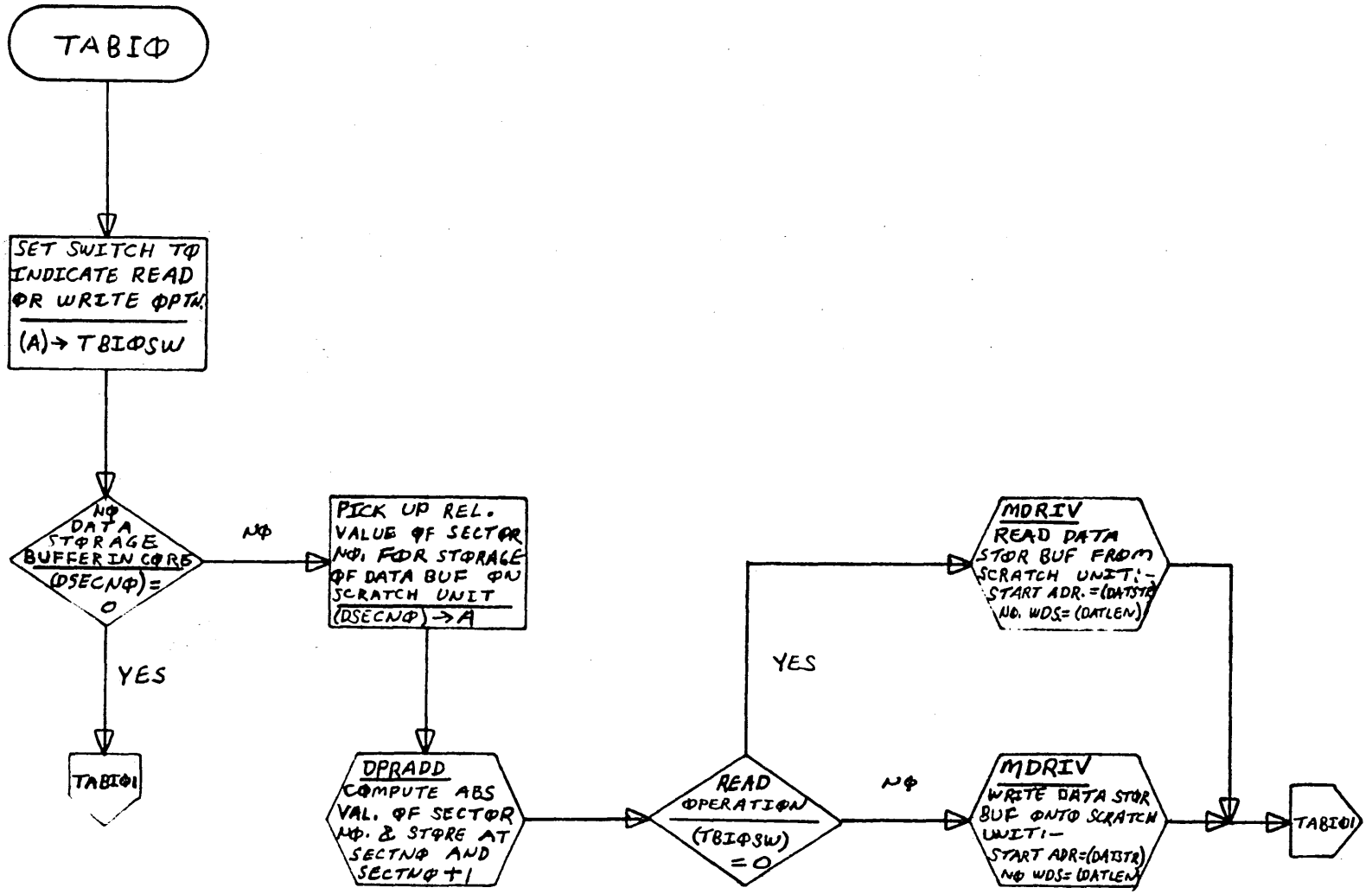


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	EQLPRQ PAGE 4 OF 4	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA 185 (FORMERLY CA 2[illegible]-1)

MAR 5 1971

60 267

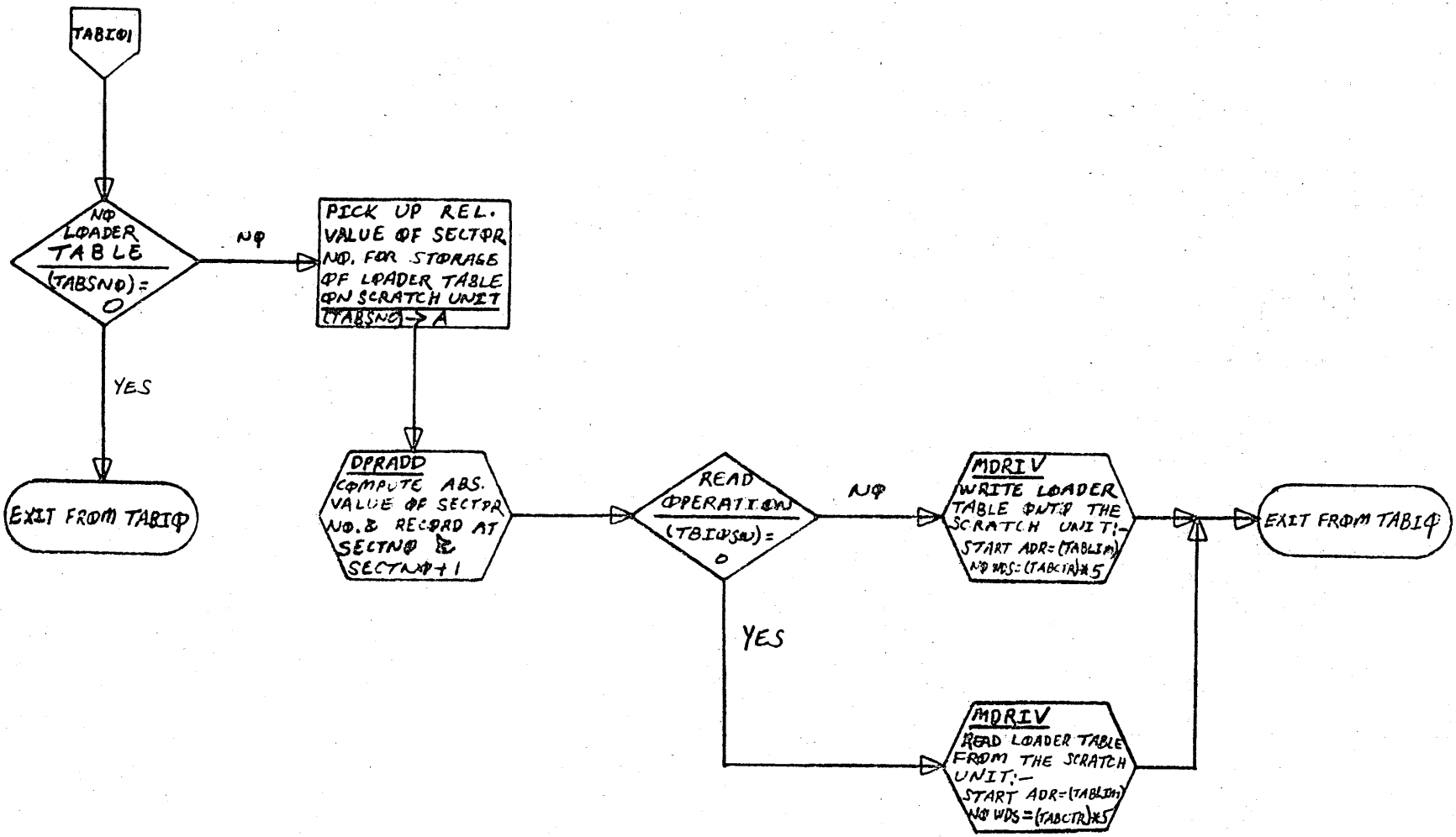


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	EQUFΦ PAGE 5 OF 6	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

E9.26B

AA1385 (FORMERLY CA127-1)

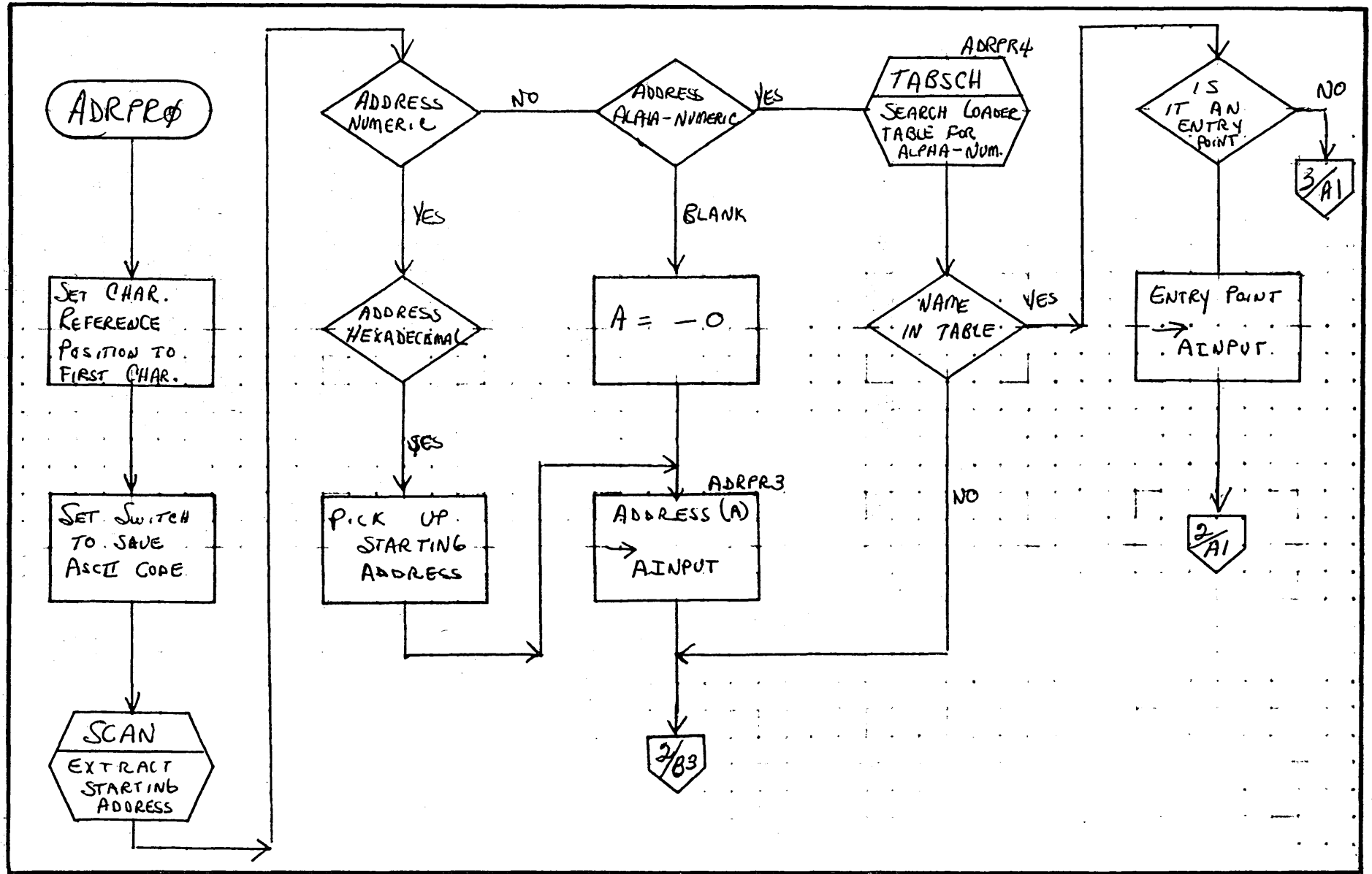


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	Edit Ø PAGE 6 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN B· / DATE	TASK NAME			

MAR 5 1971

39.469

AA1385 (FORMERLY CA127-11)



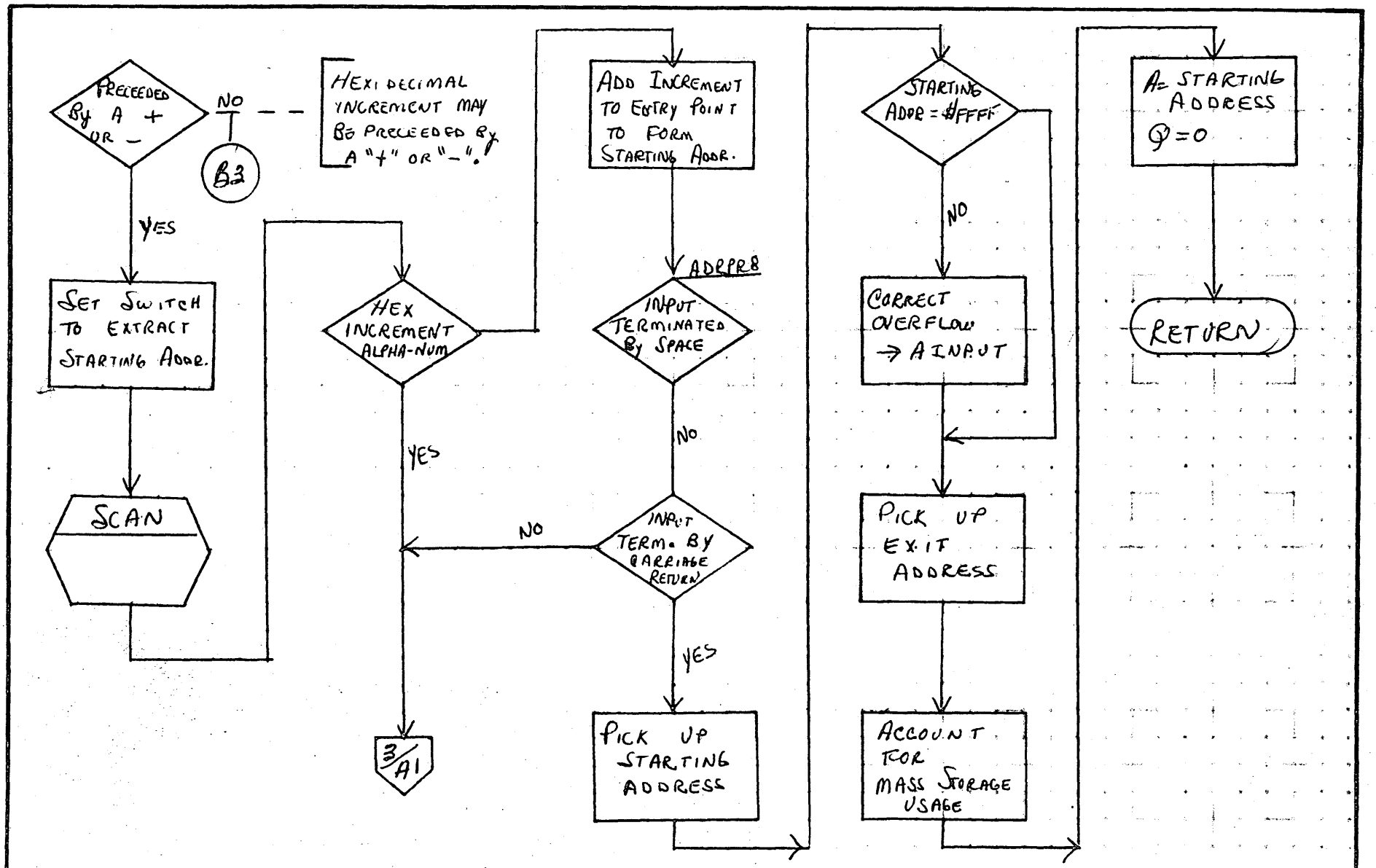
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LOADER	PROJECT MGR.			
	ADRPRO PAGE 1 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

39.270



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LOADER		PROJECT MGR		
	NUMBER AUGPRO	PAGE 2 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

39.271

PRINTED IN U.S.A.

41.0 MASS MEMORY DRIVERS - MASDRV and DBLDRV

41.1 FUNCTION

41.1.1 MASDRV is a control program for the drivers that reside on mass memory.

It reads the driver into the reserved buffer area or queues the driver if a busy driver is in the buffer area. It also queues the 1713 driver if, although the buffer area is available, the 1713 is busy.

41.1.2 DBLDRV is a dual buffer version of MASDRV.

41.2 ENTRY POINTS

MASDRV - Scheduled by the Read/Write request processor as the driver initiator.

MAS300 - Checks if buffer is available for a mass memory driver and takes the driver off the queue. It is scheduled when a driver no longer needs the buffer it resides in {driver has no more requests}.

MASINT - Entry that is stored into the driver's continuator address when driver is not busy.

MASHNG - Entry that is stored into driver error routine address when driver is not busy

MAS400 - Entered from the reader or punch initiators when the 1713 is busy. It will put these drivers on the queue.

MAS500 - Entered from the keyboard/printer driver when the 1713 is busy. It will set the 'keyboard is waiting' flag.

41.3 EXTERNALS AND DESCRIPTION

ALTDEV - System subroutine that handles device errors (Alternate Device Handler)

41.4 GENERAL PROGRAM INFORMATION

41.4.1 EQUATES

CMPRL - Priority level of MASDRV or DBLDRV. No driver residing on mass memory can have an initiator priority below this priority level.

NMASDR - Number of drivers residing on mass memory.

LNGTH - Must be equated to the length of the largest driver in MASDRV or the length of the two largest drivers in DBLDRV.

41.4.2 ASSEMBLY OPTIONS

Each driver that can be mass memory resident has a corresponding equate in MASDRV and DBLDRV. If the driver is selected to reside on mass memory, the corresponding equate should be set to a 1. This will cause the addresses of MASDRV, MASINT, and MASHNG to be inserted in the driver's initiator, continuator, and error routine in the physical device table instead on addresses within the driver (the exceptions are the tape drivers).

If any tape driver is selected to be mass memory resident, the addresses in the physical device table will contain addresses in MASDRV or DBLDRV of the routine that handles the tape drivers feature of multiple logical units on one device.

41 .7 GENERAL DESIGN PECULIARITIES

41 .7.1 MASDRV and DBLDRV are re-entrant only while they are queuing the drivers that have been scheduled. Because both MASDRV and DBLDRV contain a mass storage read request, they cannot be made completely reentrant. Because of this limitation, the main program of MASDRV or DBLDRV is always scheduled to run at a single priority level determined by the equate CMPRL.

41 .8 PROGRAM LOGIC

41 .8.1 MASDRV

MASDRV is entered with the Q register set to the physical device table address of the driver. The driver is then put on the queue. If the current priority level is the same as the priority level MASDRV runs at, MAS300 is entered, otherwise MAS300 is scheduled at the correct priority level. MAS300 checks if any driver is in the buffer and, if not, takes a driver off the queue. If there is no driver on the queue the program will exit to the dispatcher.
If a driver is currently in the buffer, a check of word 5 of its physical device table is made {if the driver is a magnetic tape logical unit, the location ADRINT is checked}. If zero {driver not busy} the PHSYTB of the current driver in the buffer is stuffed with the addresses of MASDRV,MASINT, and MASHNG and the check of the queue is then made. If the current driver is busy, exit is made to dispatcher.

The sector number and length of the new driver is stored in the mass storage read request and the driver is read into the buffer area. At completion of the read, the request is checked for error. If there was an error, exit is made to the Alternate Device Handler. If there was no error, exit is made to the first location of the buffer which will be the driver initiator routine.

41.8.2 DBLDRV

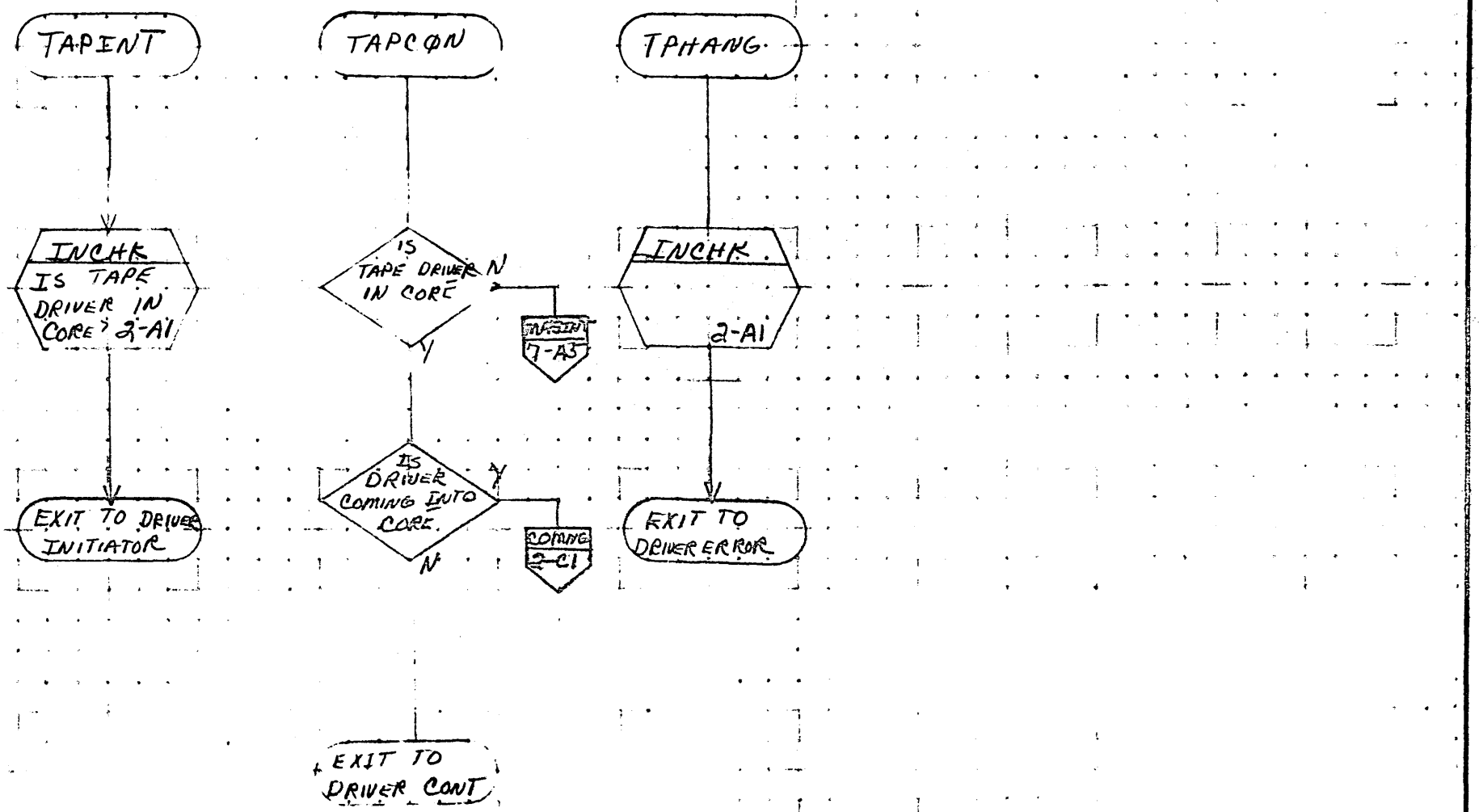
DBLDRV functions the same as MASDRV except that it will check both buffers for availability.

41.8.3 1713 CONSIDERATIONS

When the 1713 driver is present in the system, MAS300 will check if the keyboard is waiting to be executed. The keyboard is given priority in the event the system is waiting to output a message {i.e. from the Alternate Device Handler}.

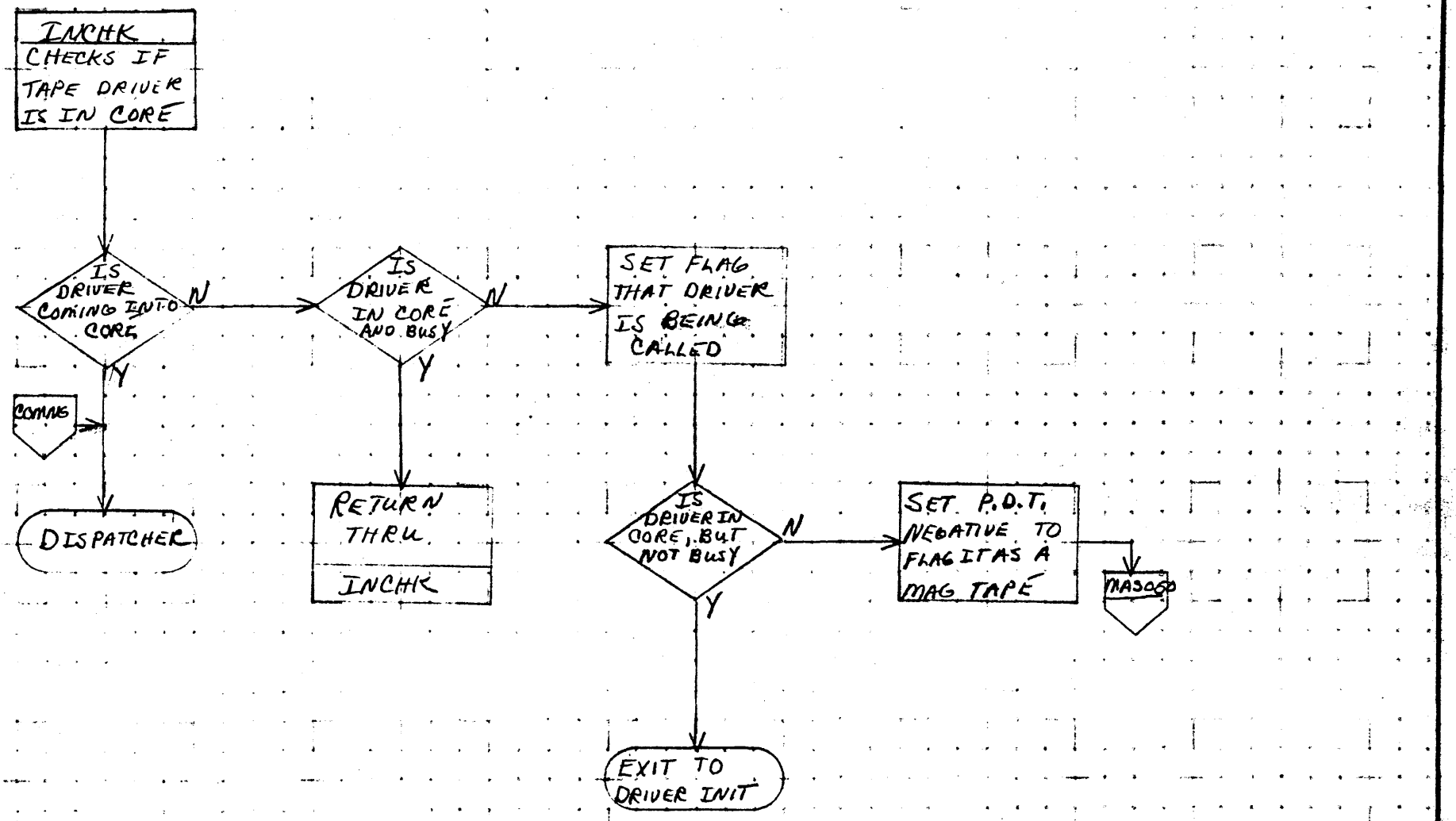
The ▽keyboard wait▽ flag is only set when the keyboard driver has been scheduled and finds the 1713 busy with a read or punch operation. The driver then exits to MAS500 which sets this flag.

If the 1713 reader or punch drivers find the 1713 busy after they have been initiated, they will return to MAS400, which put them on the queue.



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE MASDRV	PROJECT MGR.			
FLOWCHART ☐	PAGE 1 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971 41.5



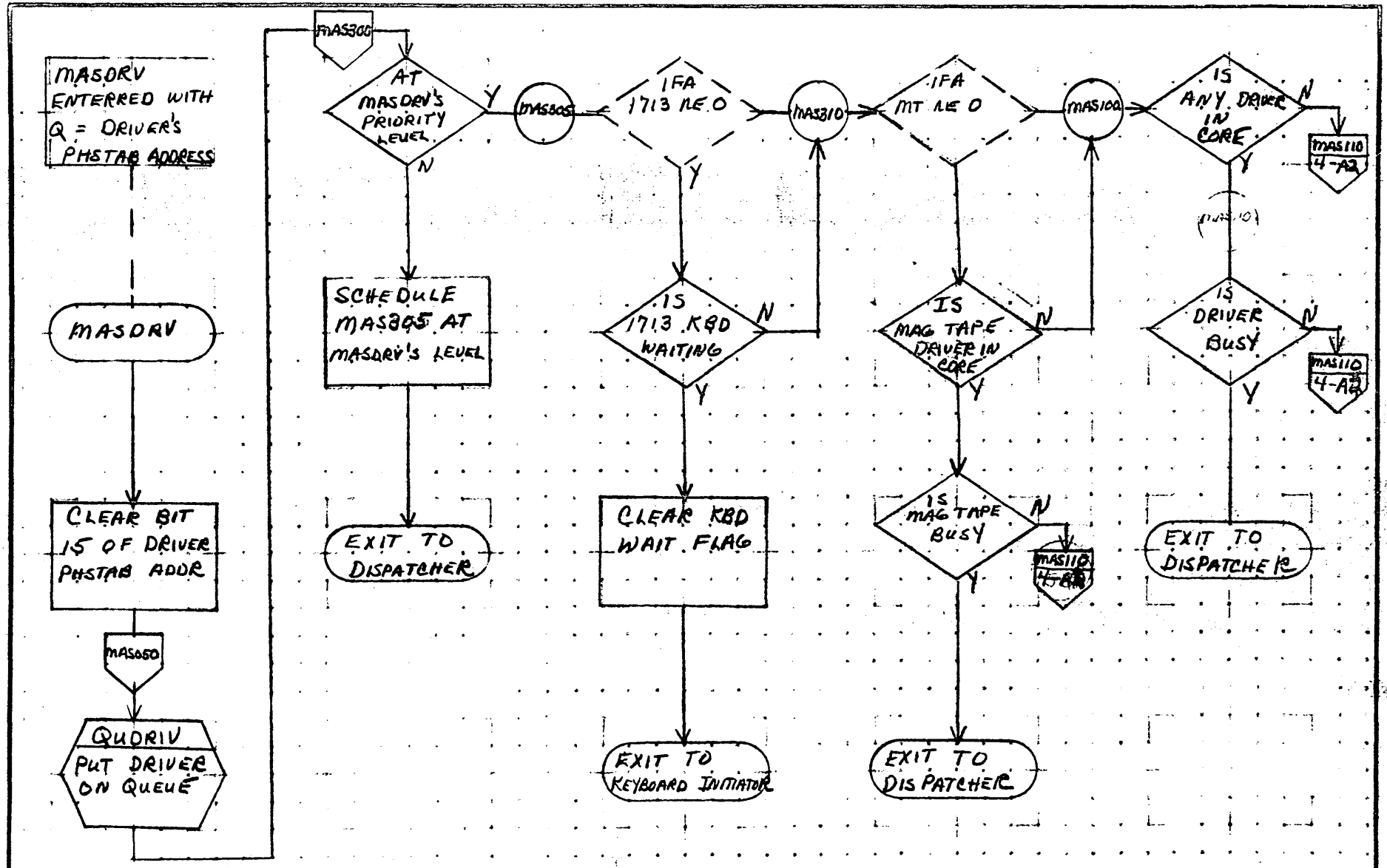
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE MASDRV	PROJECT MGR.			
FLOWCHART ☐	PAGE 2 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAY 3 1971

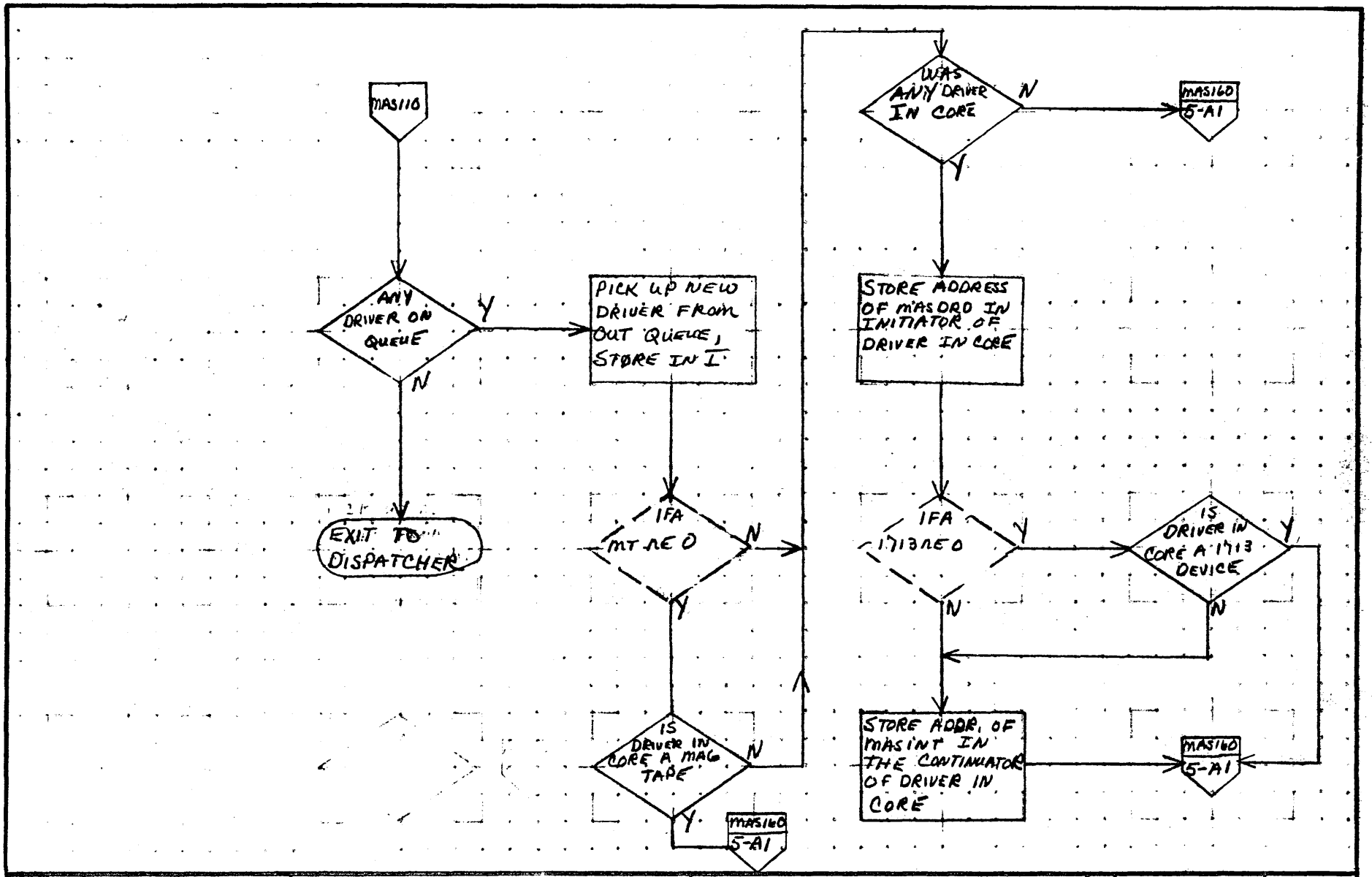
4-4-6



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: MSOS 3.0 MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: MASDRV	PROJECT MGR.			
	PAGE 3 OF 7	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

4.4.2



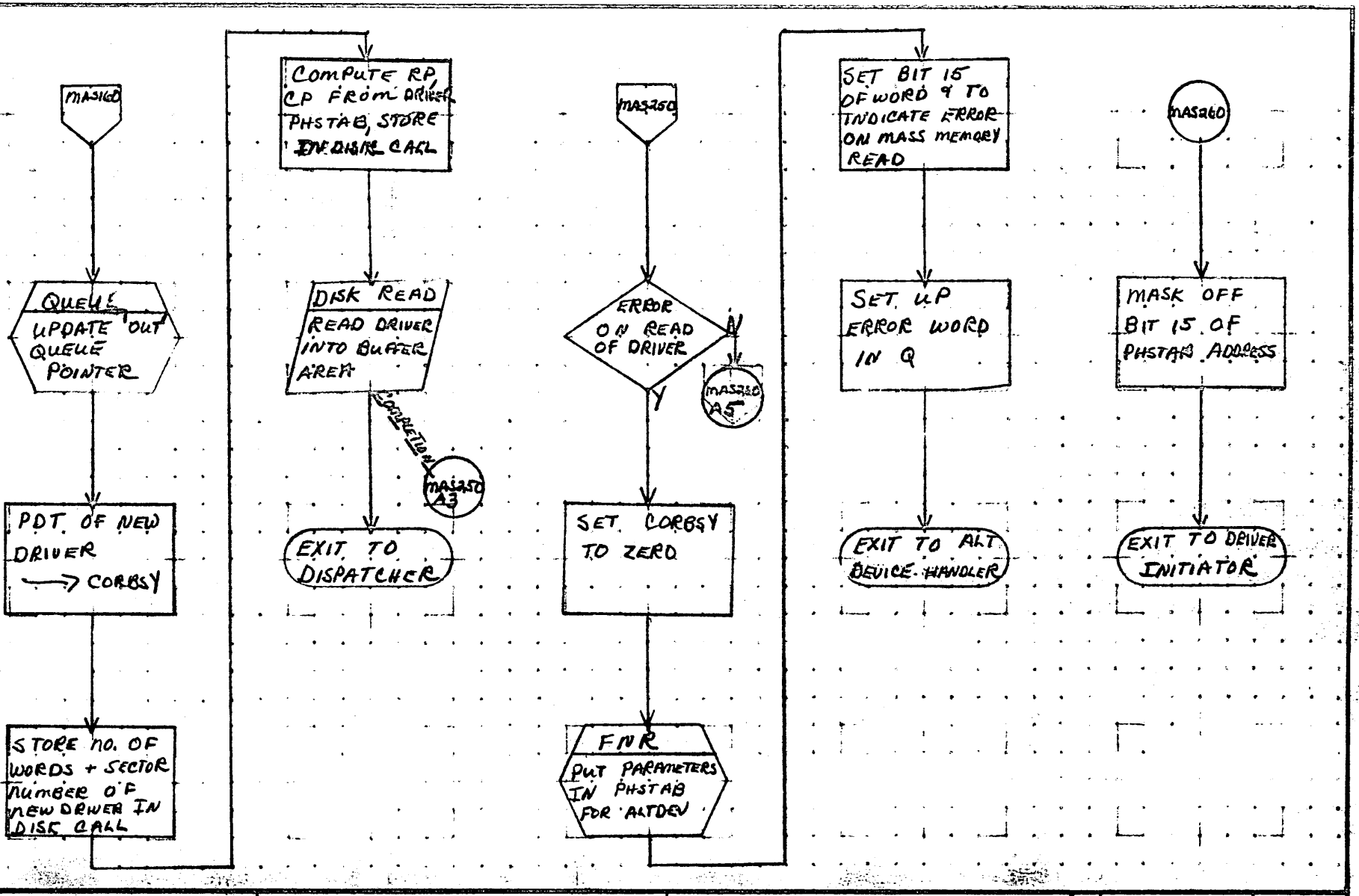
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE MASDRV	PROJECT MGR.			
FLOWCHART ☐	PAGE 4 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

41-B

AA1385 (FORMERLY CA127-1)

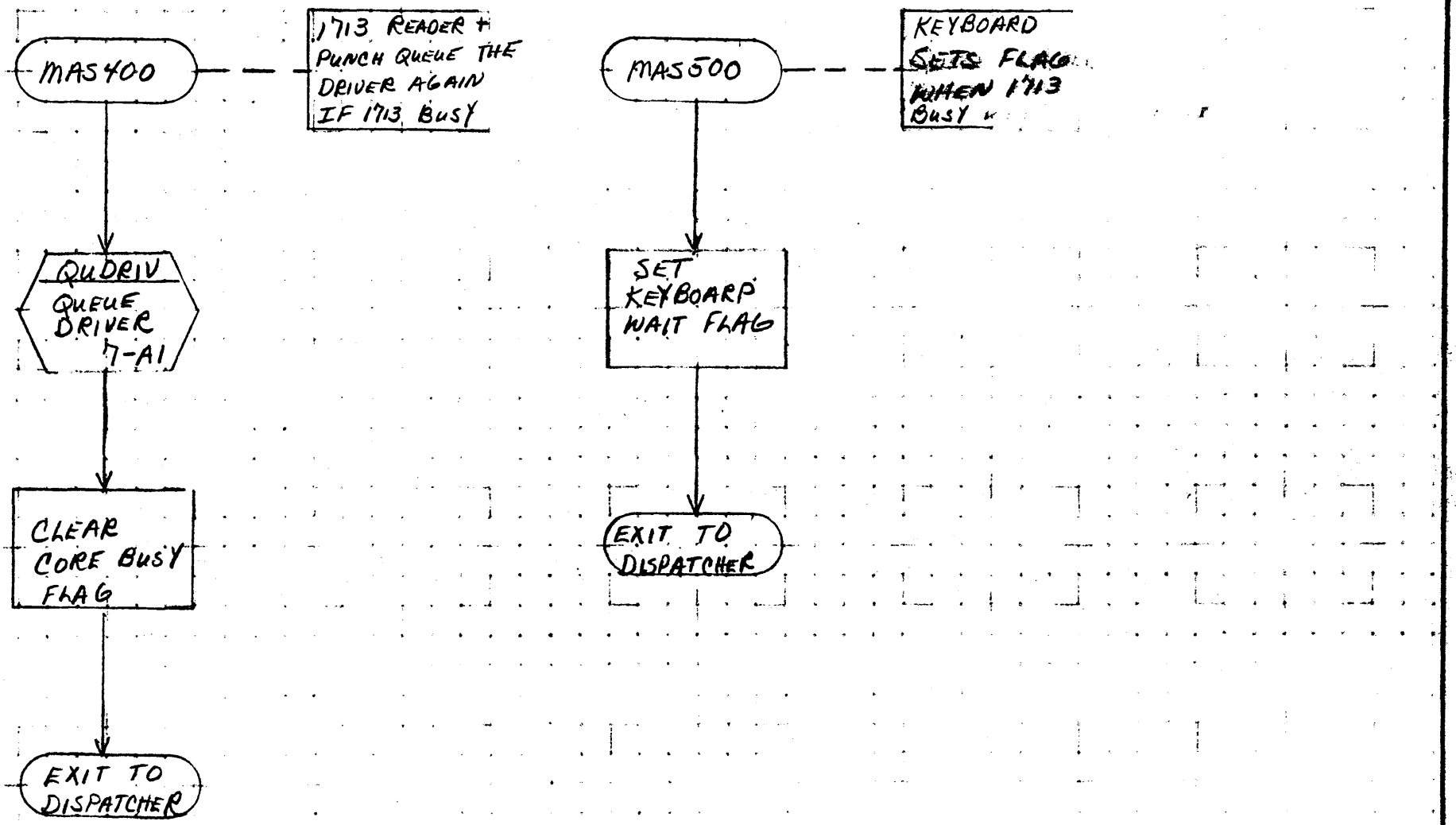
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MASDRV	PROJECT MGR.			
	PAGE 5 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

pl.9



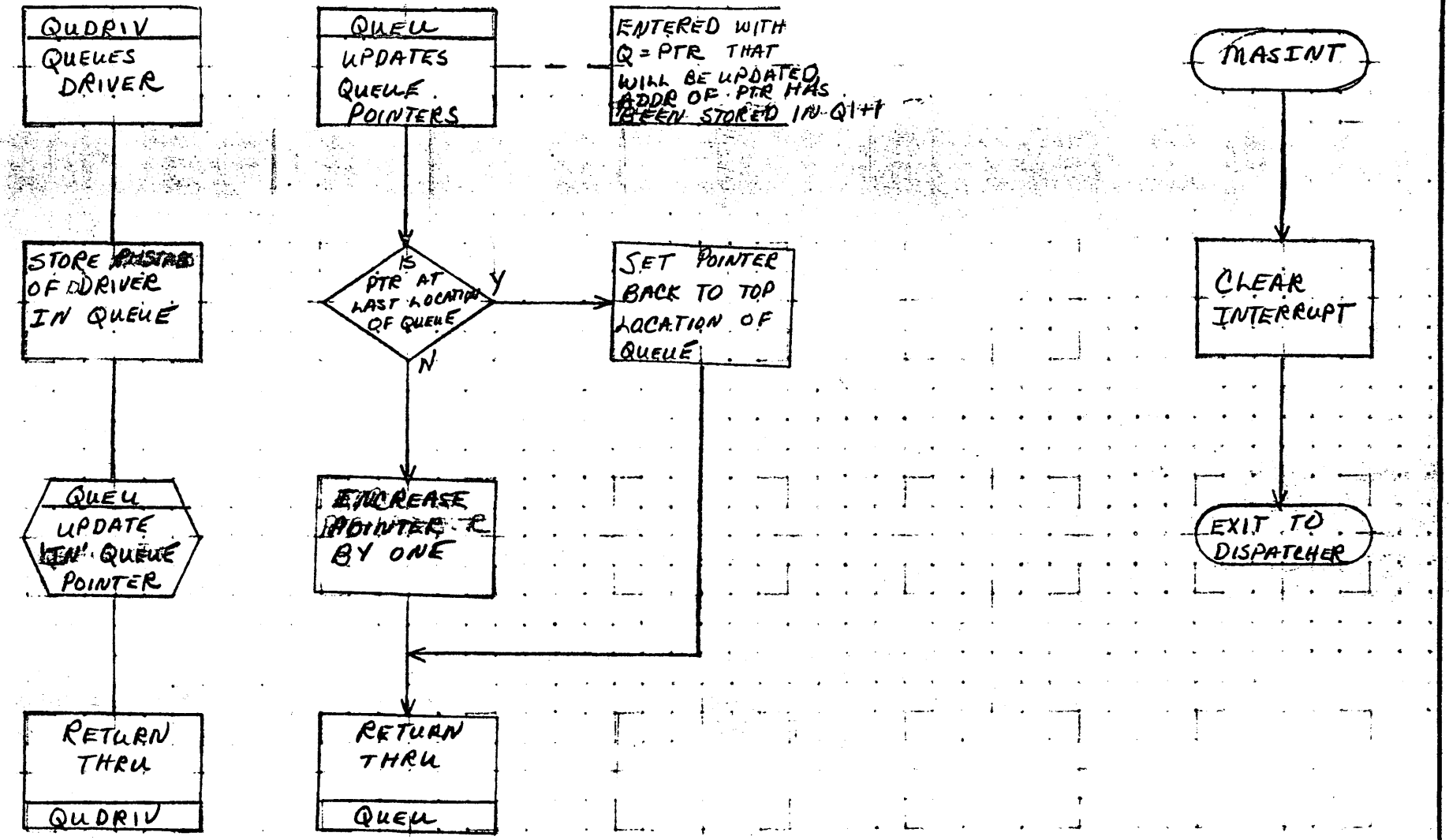
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MASDRV	PROJECT MGR.			
	PAGE 6 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

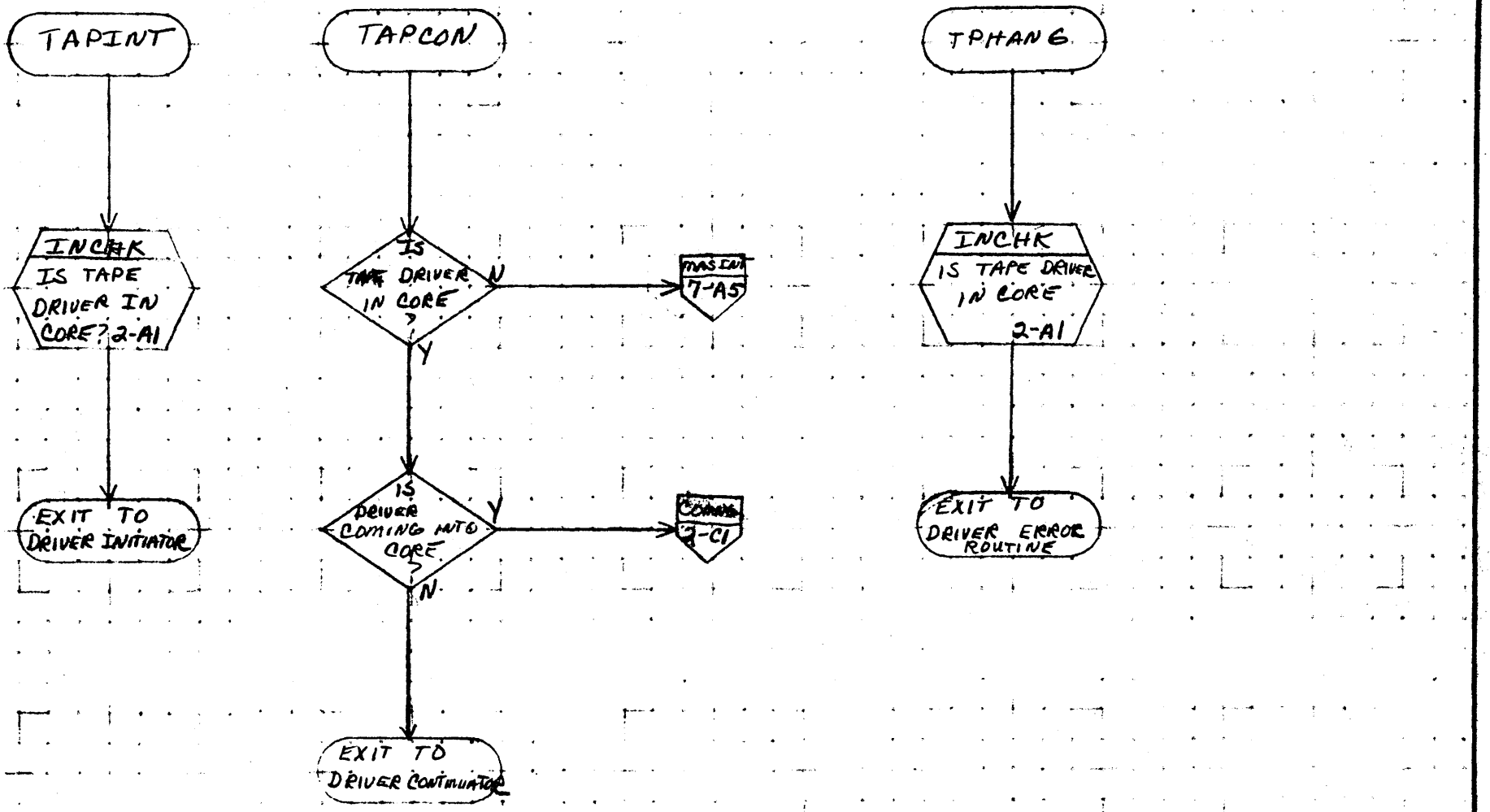
41.10



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MASDRV	PROJECT MGR.			
	PAGE 7 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

41.11



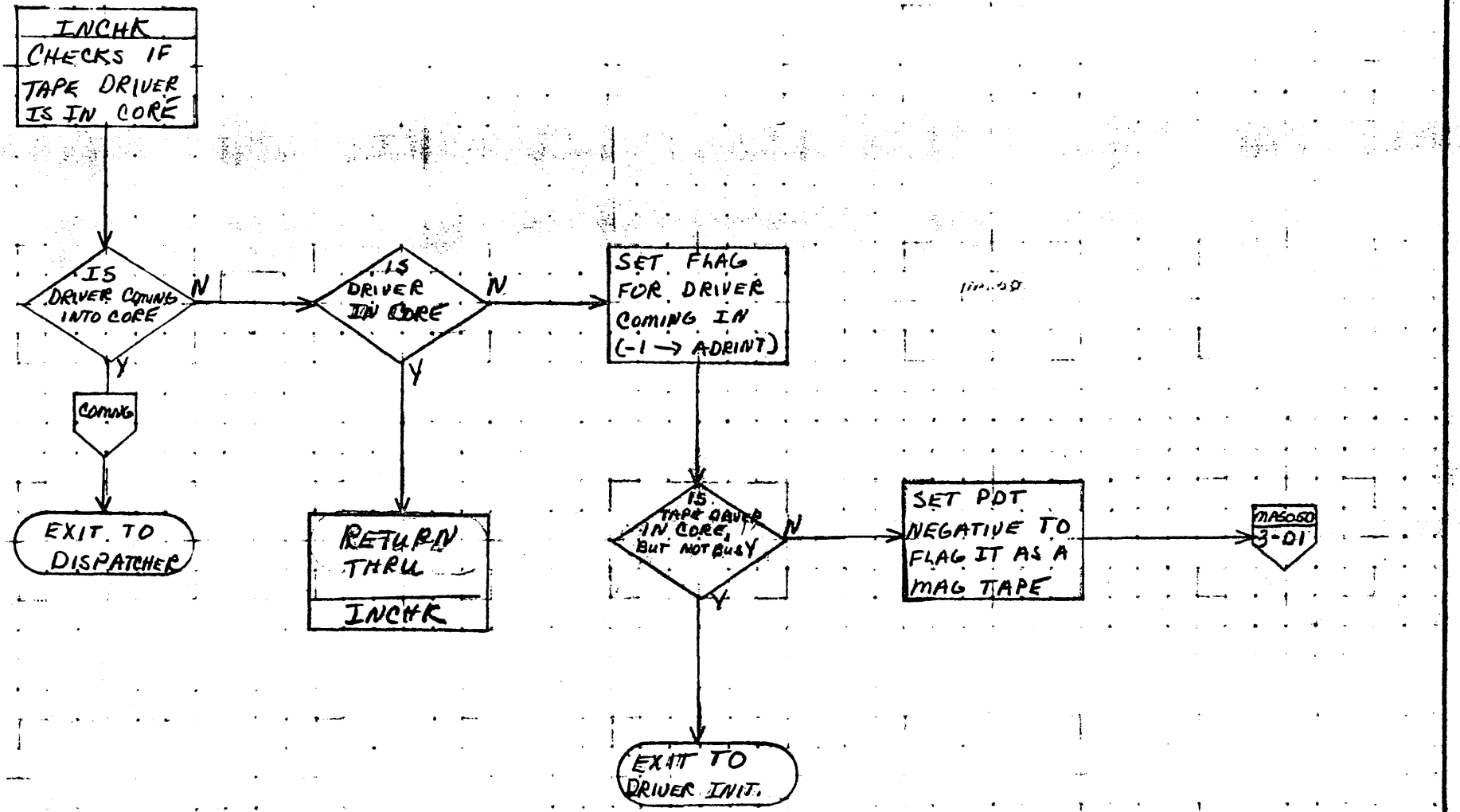
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DR'ORV	PROJECT MGR.			
	PAGE 1 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

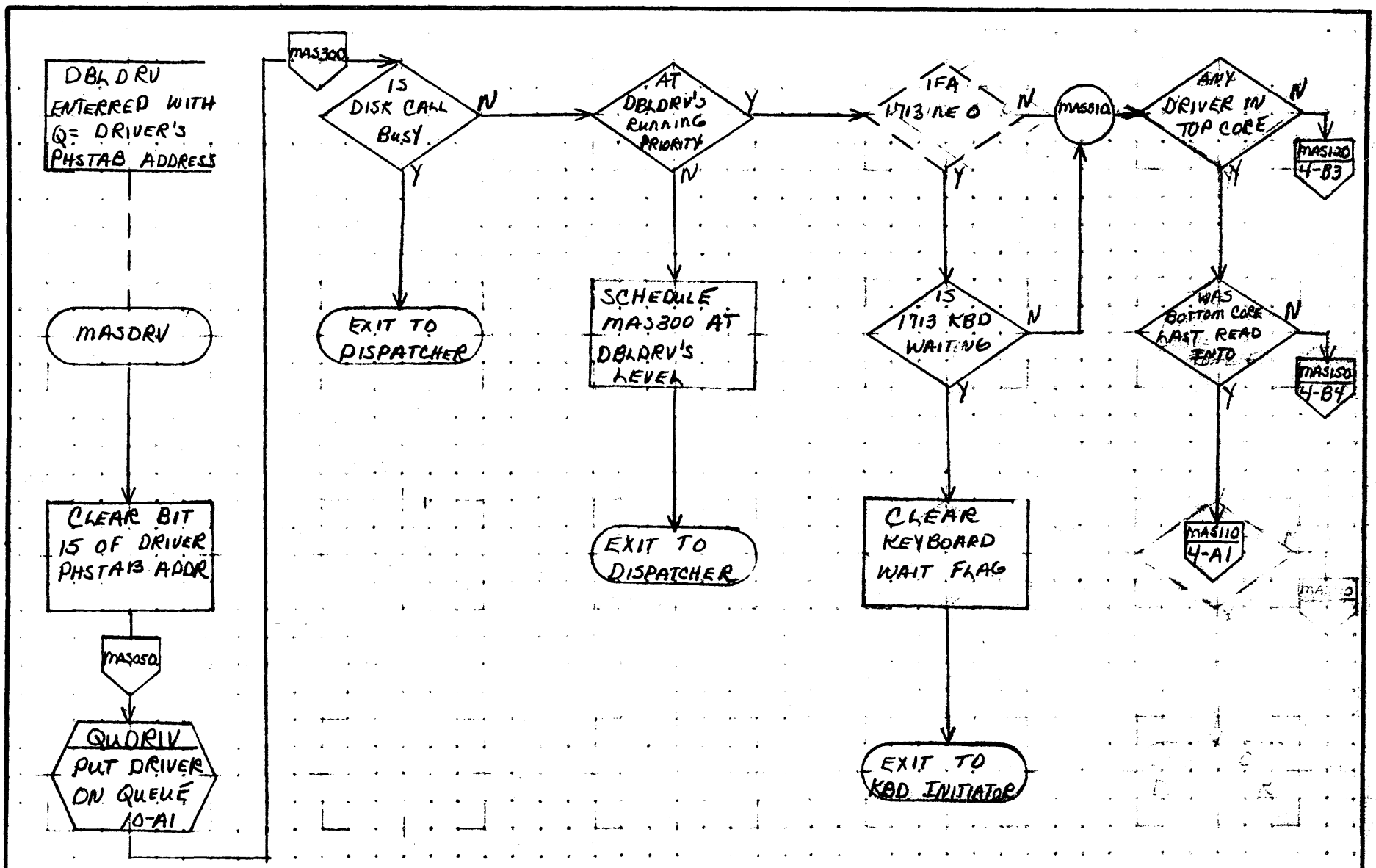
11.12



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS MSOS 3.0 MACH-TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DBLDRV	PROJECT MGR			
FLOWCHART ☐	PAGE 2 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

41.13



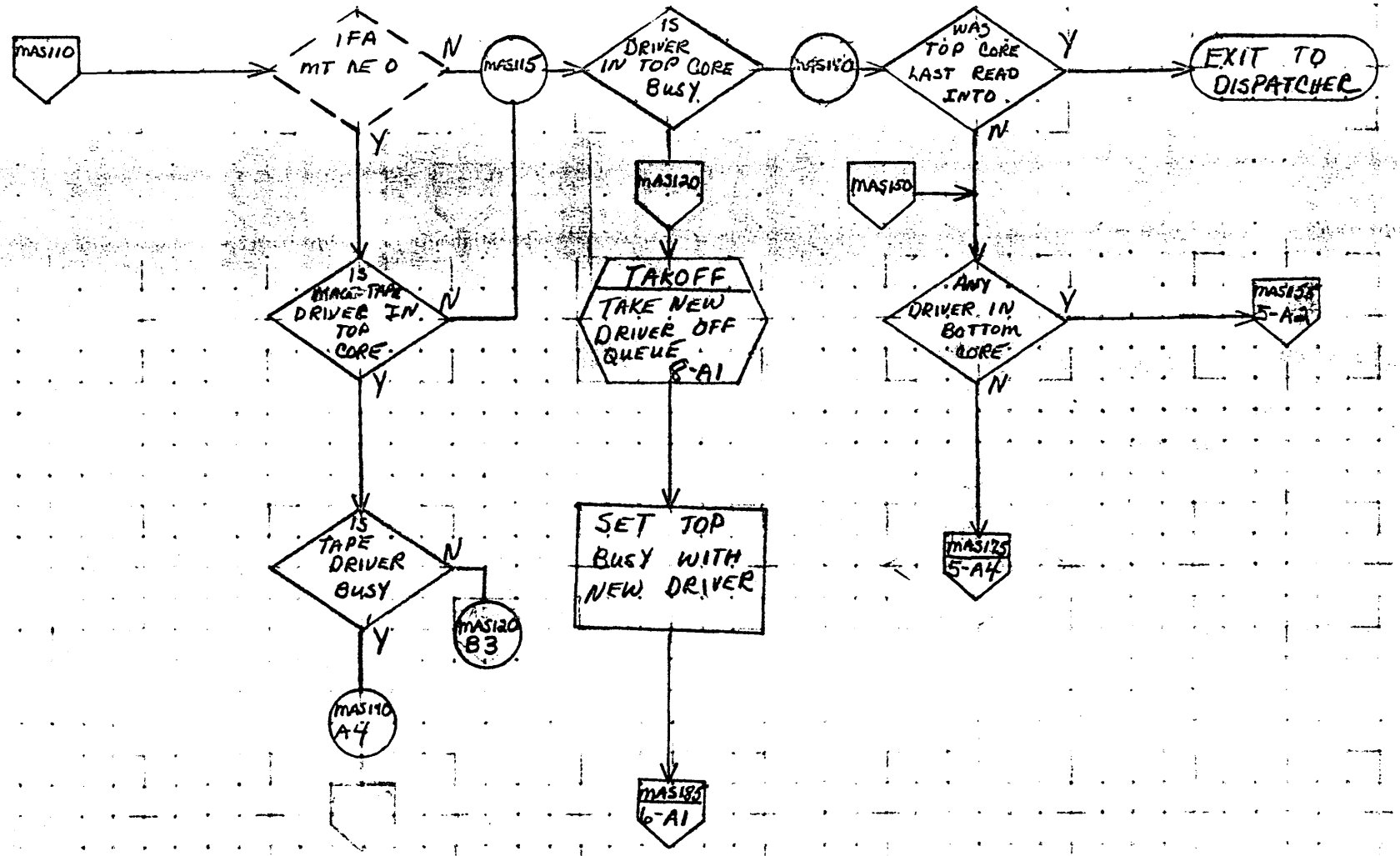
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DBLDRV	PROJECT MGR.			
	PAGE 3 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

41-14

AA1385 (FORMERLY CA127-1)

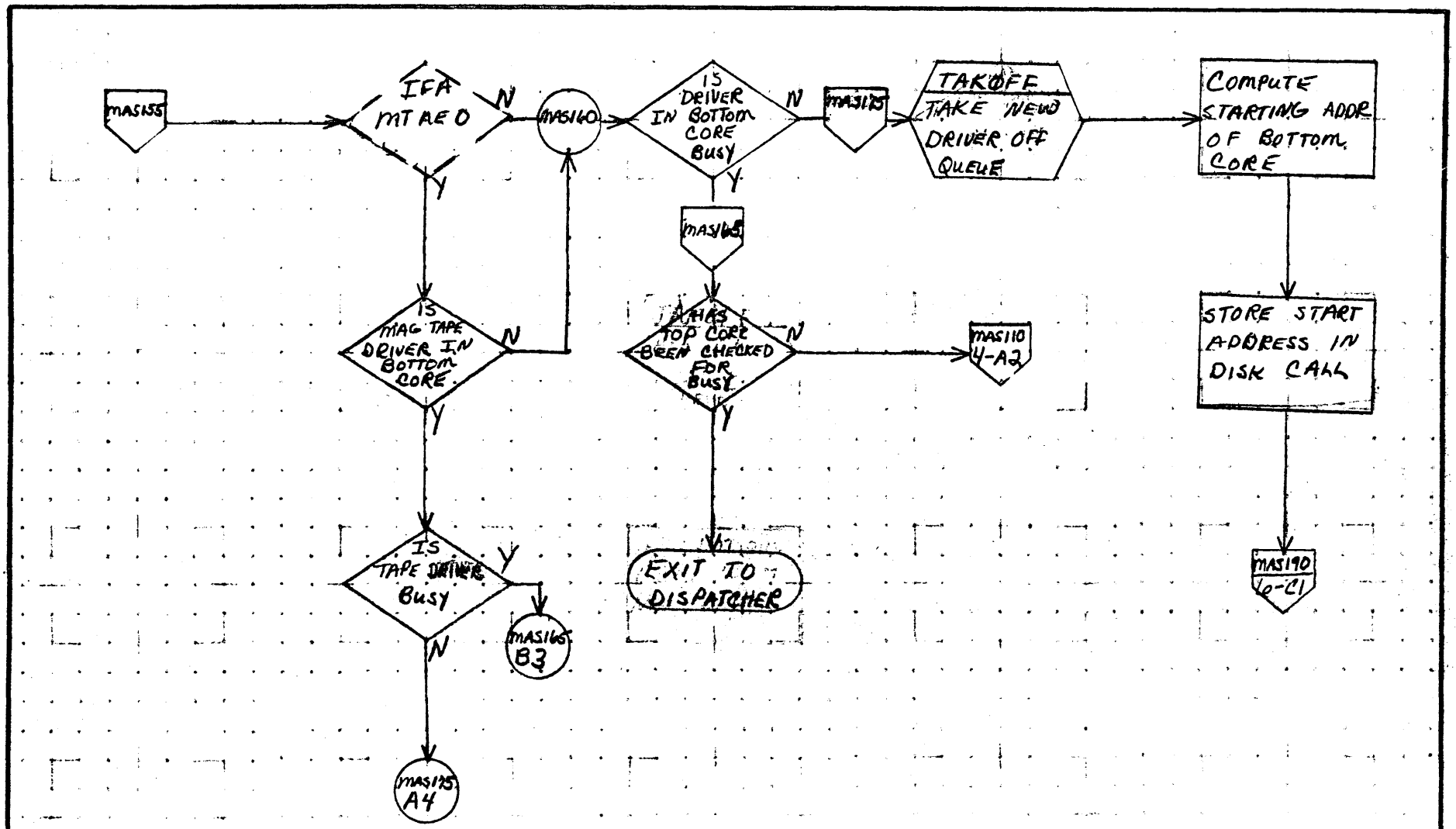
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DBLDRV	PROJECT MGR.			
	PAGE 4 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

41.15



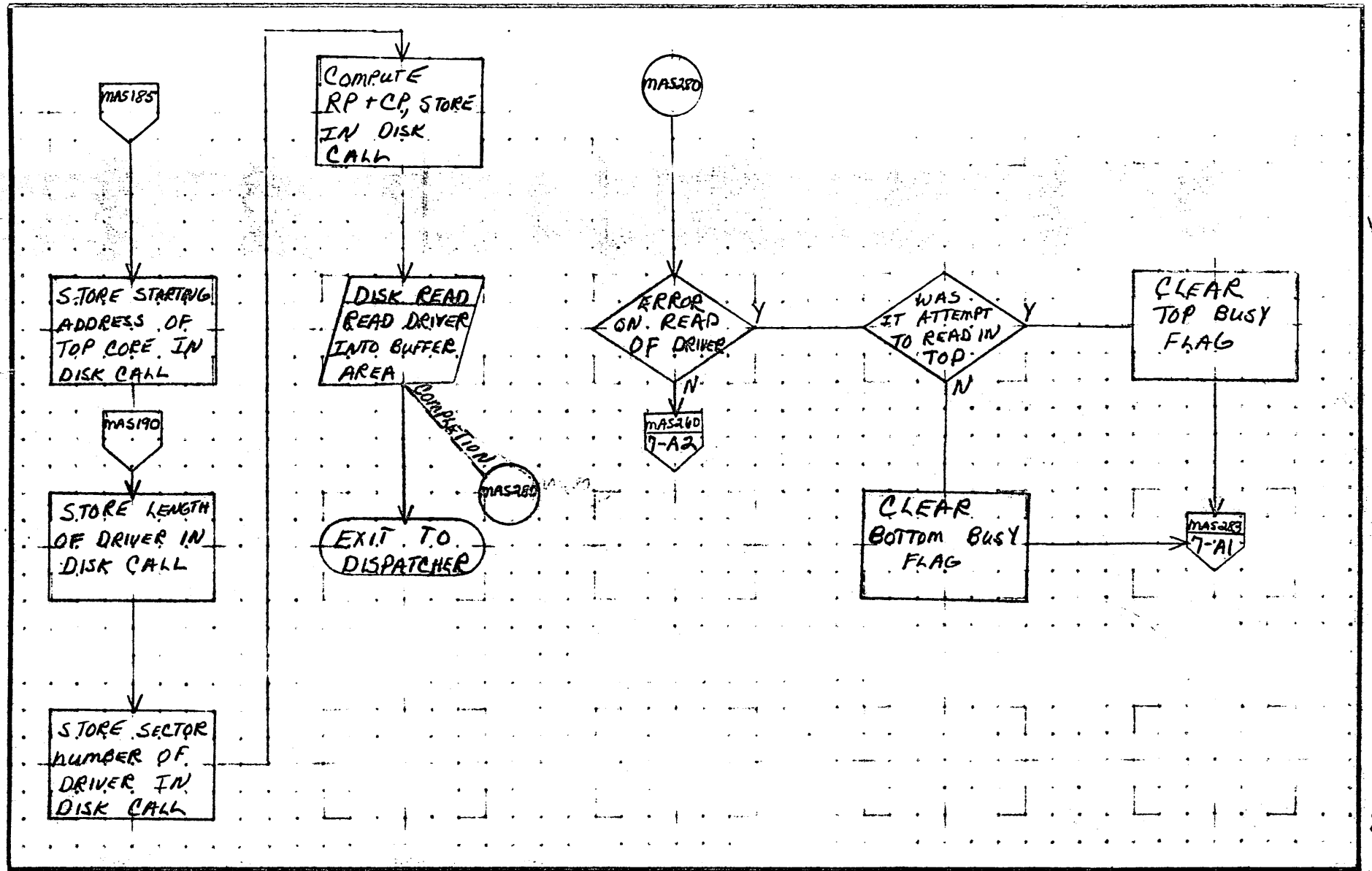
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS MSOS 3.0	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE DBLDRV		PROJECT MGR.		
FLOWCHART ☐		PAGE 5 OF 10	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 19/1

41.16

AA1385 (FORMERLY CA127-1)

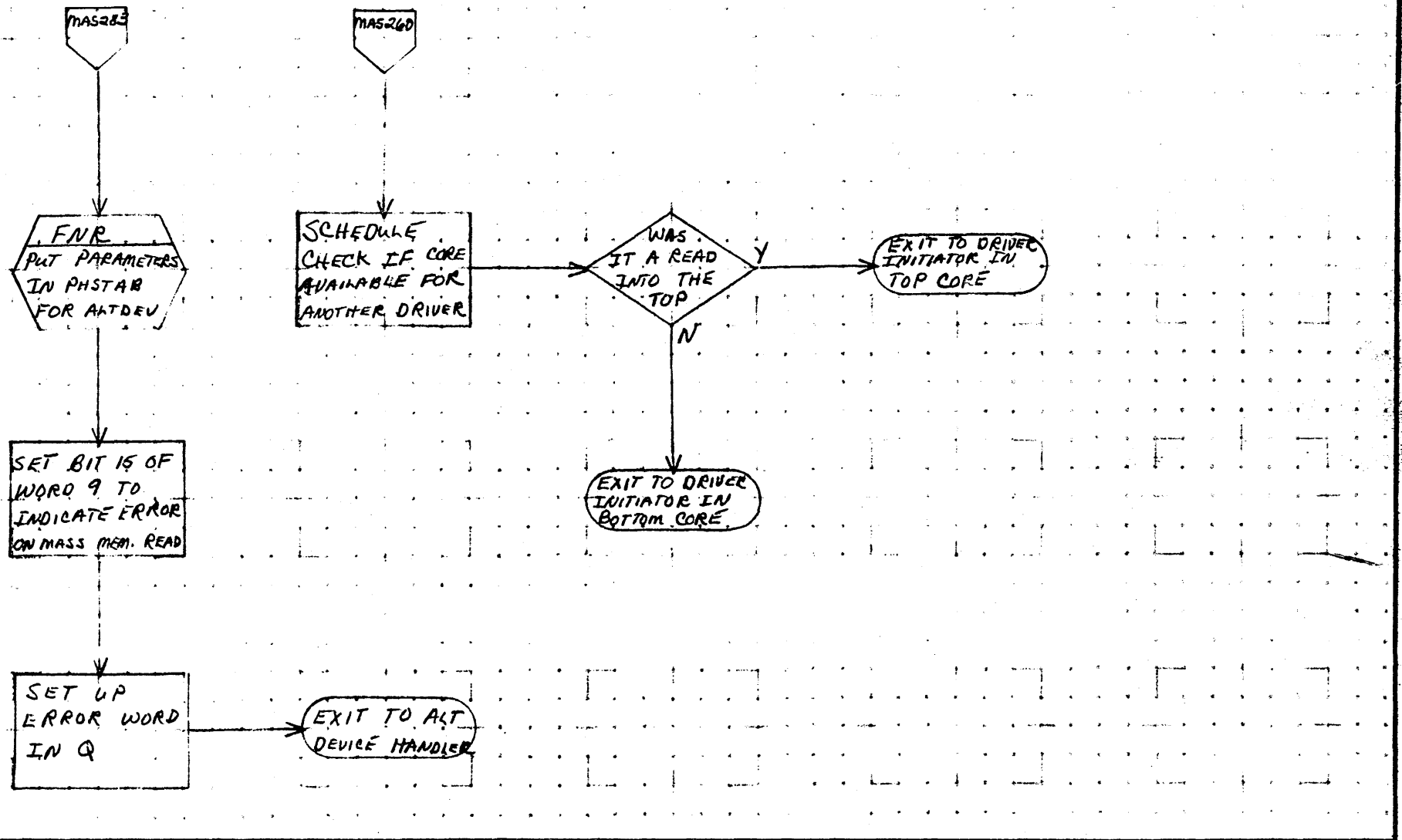
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DBLDRV	PROJECT MGR.			
FLOWCHART ☐	PAGE 6 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

MAR 5 1971

41.17



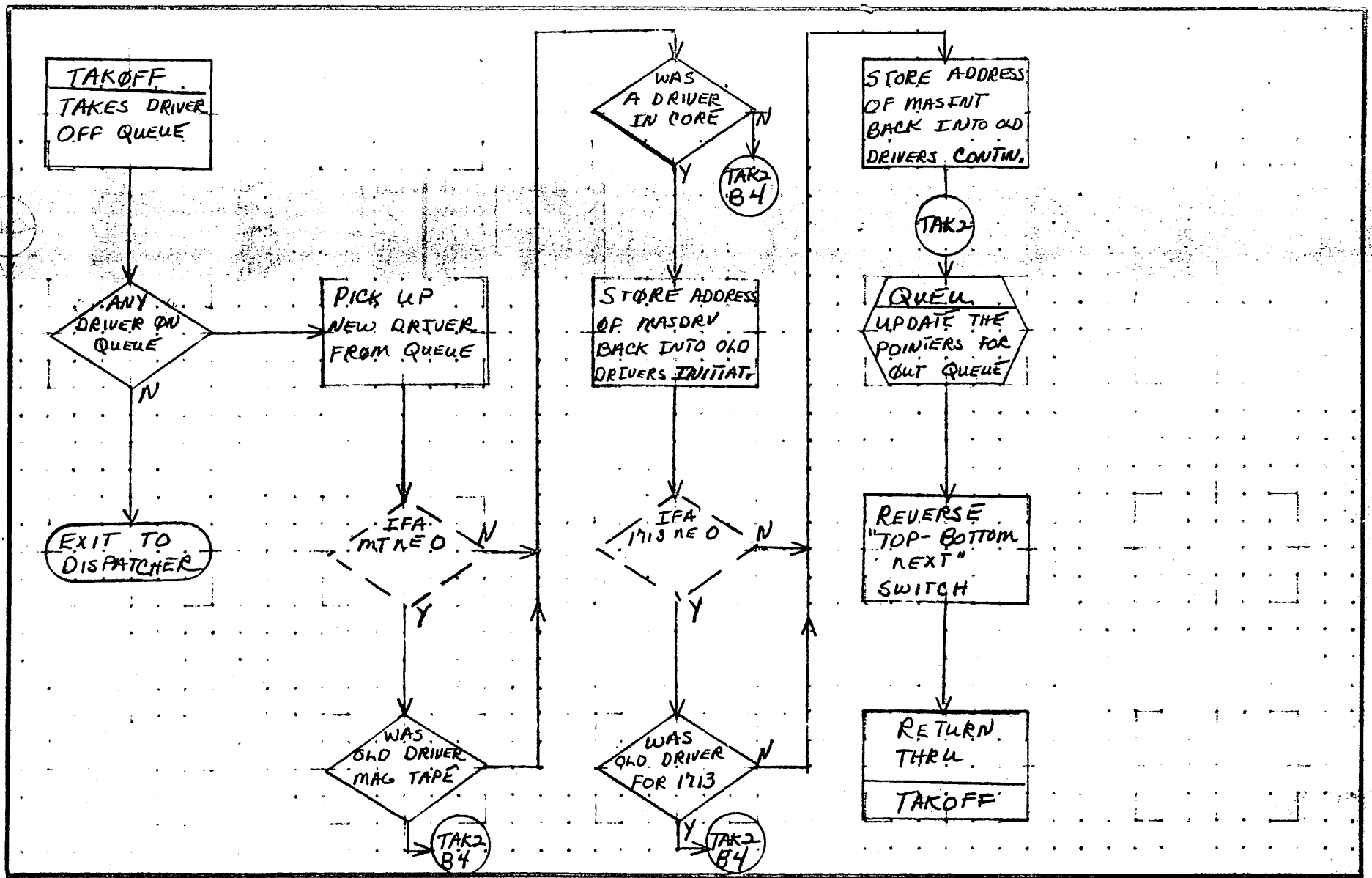
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DBLDRV		PROJECT MGR.		
		PAGE 7 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

41.18

AA1385 (FORMERLY CA127-1)

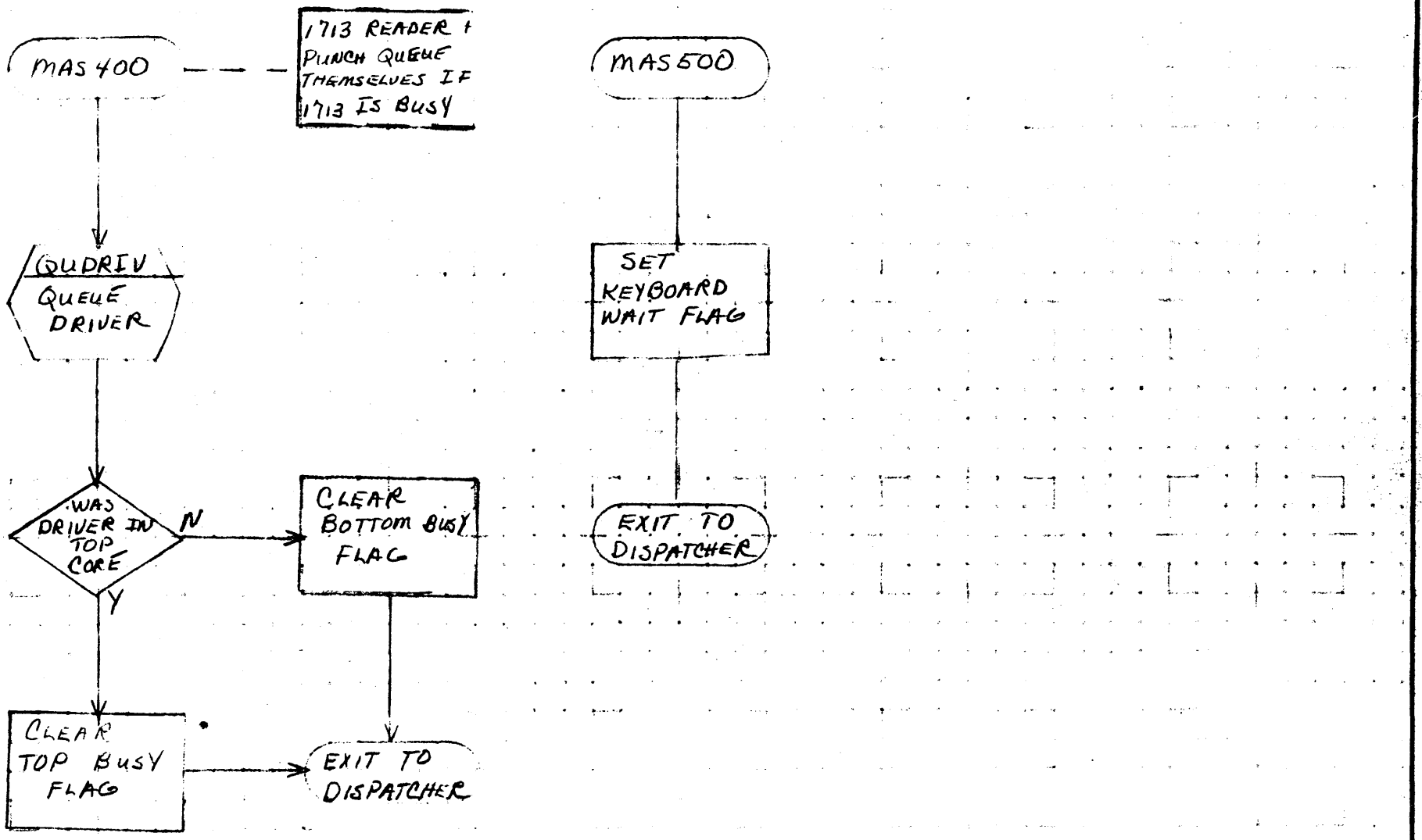
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DBLDRV	PROJECT MGR.			
	PAGE 8 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

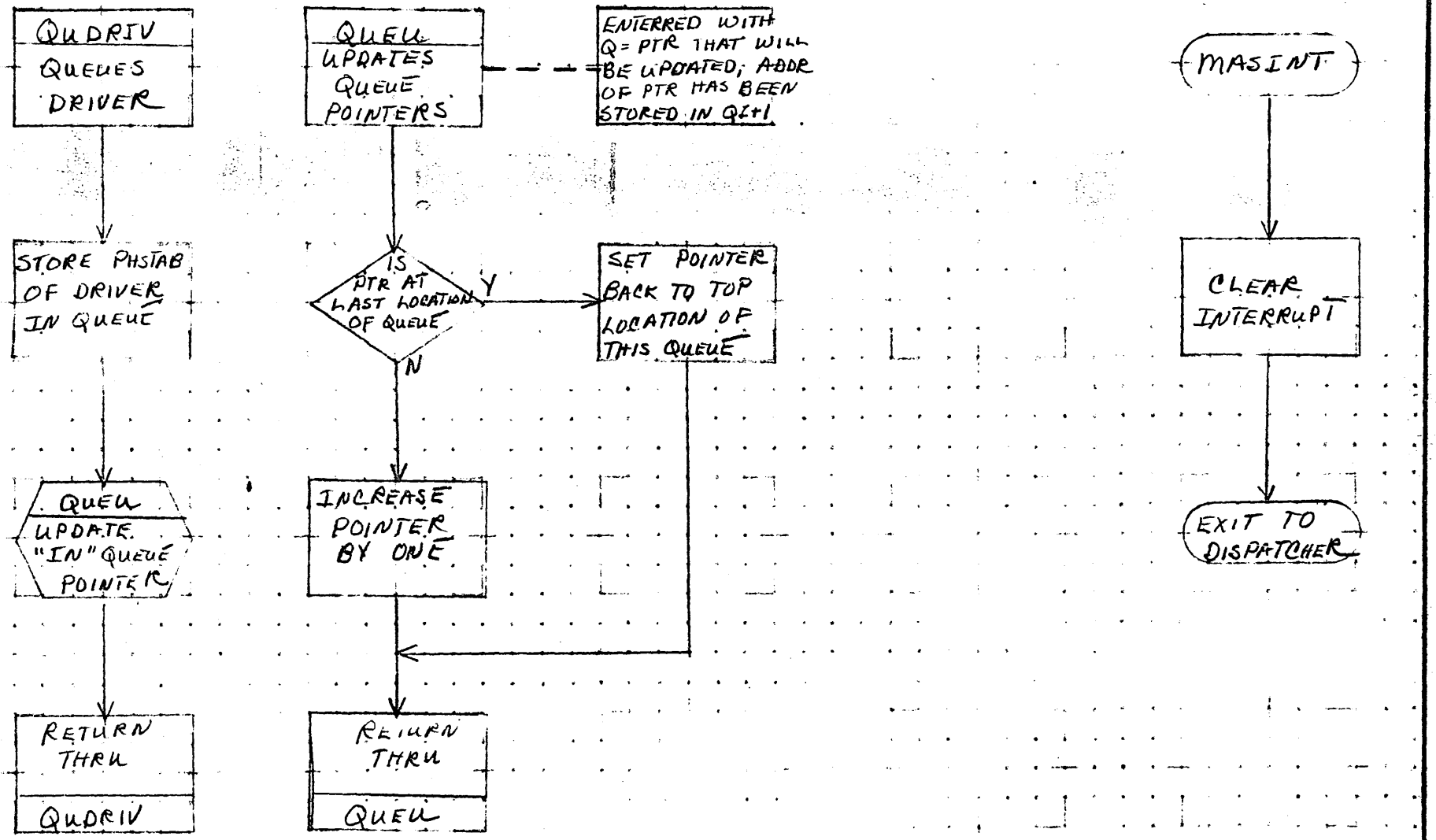
41.19



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE UBLDR V	PROJECT MGR.			
	PAGE 9 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

41.20



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MSOS 3.0 MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DBLDRV	PROJECT MGR			
	PAGE 10 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN B DATE	TASK NAME			

MAR 5 1971

41.2.1

42.0 Buffered Data Channel Allocation - BUFALC

42.1 Program Function

BUFALC makes it possible to have more than one device connected to the 1706. I/O requests are queued on a first-in, first-out basis.

42.2 Exit Interfaces

Return is made to P+3 in the driver with I = Current Phystb address.

42.3 Entry Interfaces

A = 0 if FNR has found no more requests

A ≠ 0 if FNR has found a request

42.4 Internal Description

The following calling sequence is required for entry to BUFALC:

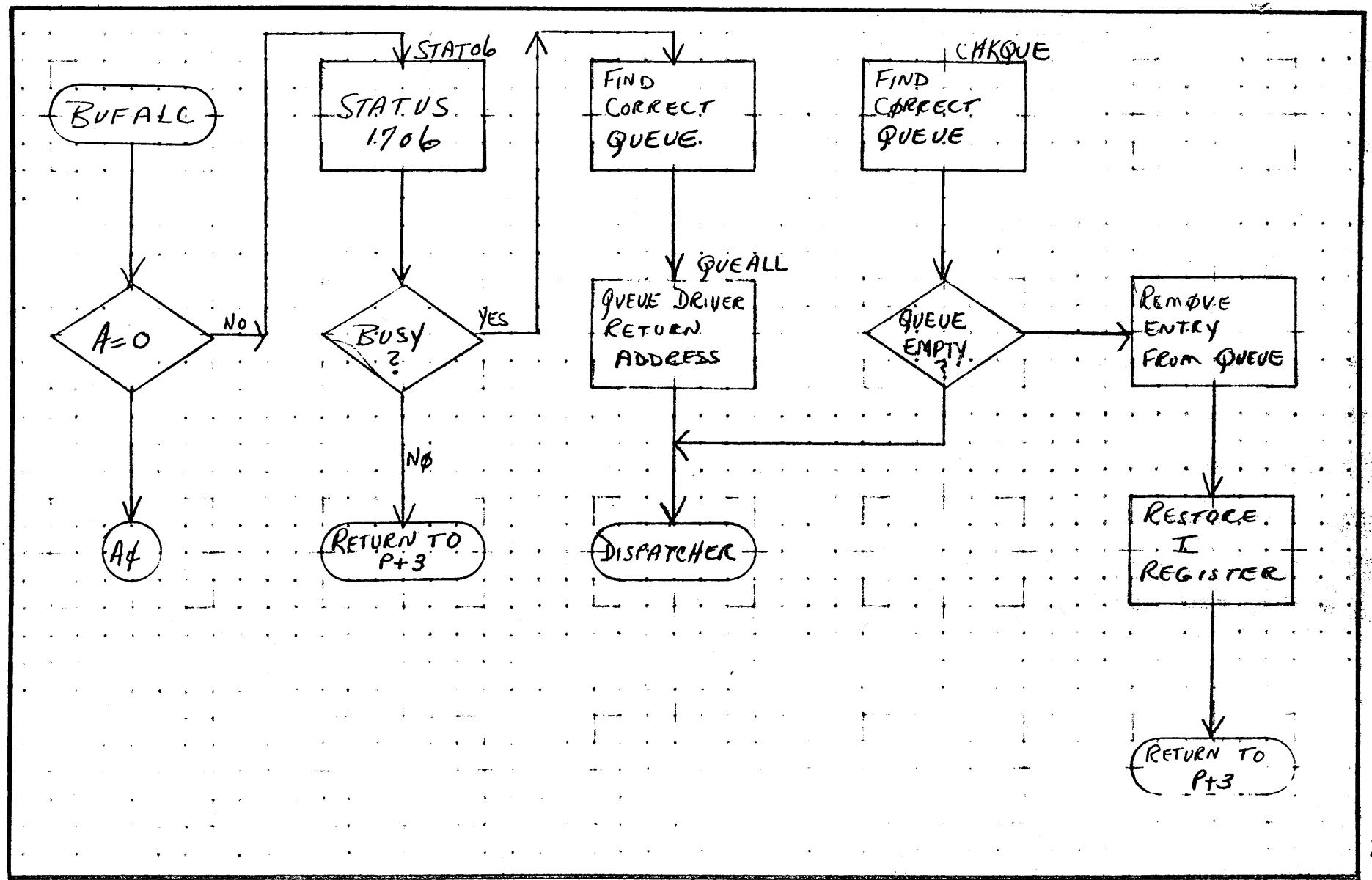
```
EXT  BUFALC
RTJ  BUFALC
ADC  0        1706 reject address
ADC  0        Current Phystb address
```

Entry to BUFALC is made after P+1 and P+2 returns from FNR. The current status of the 1706 is checked. If the 1706 is available, return is made to P+3 in the driver with the I register intact. If the 1706 is busy, the return address is put on the queue and exit made to the dispatcher.

When no more requests are found via FNR the queue is checked for any waiting requests. If any, I is restored and return is made to P+3 in the driver. If no more requests are found, exit is made to the dispatcher.

42.5 Queue

A queue is maintained for storage of the driver return address. One word is reserved for each device on the buffered data channel.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE BUI ALC		PROJECT MGR		
		PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE 10/27	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

42.3

43.0 ENGINEERING FILE

43.1 Program Name: Engineering File Recording Routine L0G

43.1.1 Function:

This program records the error from the failing device in an error table in core.

43.1.2 Entry Point Names:

L0G - entry to program
SCHFLG - switch for the logging overlay
ERRT0P - pointer to current location in error table
MASNM - number of mass memory devices in system
C0RTBL - engineering file for mass memory devices
ETBLE - error table

43.1.3 Externals:

NUMLU - Number of logical units in system
L0GGER - Name of program which moves the error into the proper engineering file. This program is mass memory resident
L0G1A - Physical Device Table Thread

43.1.4 Entry Interfaces

From the Alternate Device Handler

I - Base of the Physical Device table of the failing device

Q - Error word {error code bits 5-0
logical unit of failing device 15-6}

A - Zero

From a driver {to log a recoverable error}

I - Base of the Physical Device Table of the failing device

Q - Error Code {right adjusted}

A - ≠ 0

43.1.5 External Interface

The program exits via an indirect jump through its entry cell

I - Base of the Physical Device Table of the failing device {restored to its value at entry}

Q - Restored to value at entry

A - Restored to value at entry

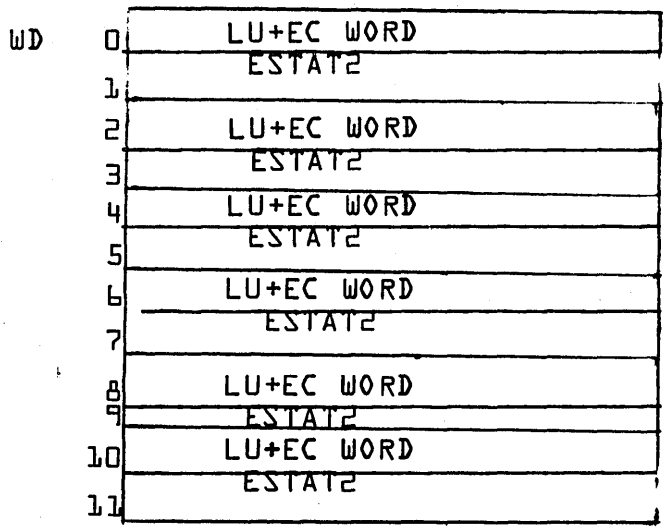
43.1.6 General Program Information

This routine is used to record an error. It is called by the Alternate Device Handler or by a driver when a hardware error occurs on a device. This program logs the error to an error table. It then writes the updated error table out on the system image on mass memory. Finally it schedules the mass memory resident program which will move the error to the engineering file for the logical unit assigned to the failing device.

Low core cell $E9 holds the base address of the system image on mass memory. This program is assembled as a part of the Alternate Device Handler. It is, however, not a part of the Alternate Device Handler and could be built {with minor modifications} as a separate module. This program uses volatile storage.

MASNM - Number of mass memory devices in system
Equate which must be set for each system

ETBLE



The error word and status for an error is stored in the next available 2 word slot in ETBLE. If the table is full the error data will be stored in the first entry (the oldest error being lost in that case). In other words ETBLE is circular table. The logging routine is scheduled whenever an error is recorded. However it is possible to get several errors before the logging routine gets control and logs them to the engineering file.

43.1.7 General Design Peculiarities

1.7.1 LOG is a reentrant program.

1.7.2 ERRTOP, ETBLE table, and CORTBL must be block stored together.

1.7.3 CORTBL is the entry point for the engineering files for the mass memory devices.

1.7.4 The engineering files for mass memory devices are kept in core. They must be block stored as stated in section 1.7.2. Further they must be stored in the same order as they are arranged in the LOGIA table.

1.7.5 Errors are logged from the Alternate Device Handler - ADEV or from the subroutine in which the error occurred.

1.7.6 LOG is a subroutine

43.1.8 Program Logic

The program gets volatile storage. It saves the input parameters and positions the error code. Then it finds the logical unit by comparing the PDT Base in I with the LOGIA table to find a match. The program merges the logical unit and the error code into the error word. The status word ESTAT2 is read from the PDT. Then the error word and the status word are moved into the next 2 available locations in ETBLE. To guard against the possibility of losing these errors in case core should get clobbered ETBLE is written out onto the core image on mass memory. Just before exiting the program schedules the logging routine LOGGER to move the error to the appropriate engineering file.

43.2 Program Name: Engineering File Logging Routine LOGGER

43.2.1 Function:

This routine moves the error data from the Error Table ETBLE to the engineering file for the logical unit assigned to the physical device on which the error occurred.

43.2.2 Entry Point Names:

LOGR - entry to this routine

43.2.3 Externals Description

ETBLE - Error table
CORTBL - Engineering File for mass memory devices
LOGIA - Physical Device Table Thread
SCHFLG - LOGGER switch; = 1 Logger Scheduled;
= 0 Logger not scheduled
ERRTOP - Pointer into ETBLE

43.2.4 Entry Interfaces

No parameters are passed via the register.

43.2.5 External Interfaces

The registers are not restored on exit. There are no parameters passed on exit.

43.2.6 General Program Information

16.1 The program is mass memory resident

16.2 Engineering Files for all physical devices except the mass memory devices are kept on mass memory.

16.3 Engineering Files for mass memory devices are kept in core.

16.4 Errors are stored with the most recent at the top of the Error Word, status word area in the Engineering File. Errors are lost off the bottom.

43.2.7 General Design Pecularities

17.1 This program uses the word addressable disk driver.

43.2.8 Program Logic:

This program moves the errors from the error table to the Engineering File.

The program gets an error from the error table.

It finds the logical unit on which the error occurred.

It reads the engineering file for that logical unit into a buffer in core.

NOTE: {If the error to be logged occurred on a mass memory device the program will use the engineering file for the device that is already in core. It does not need to read one in from mass storage. It uses duplicate logic to that used for the non-mass memory devices. After the file is updated, it updates the portion of the system image on mass memory.}

It moves the error and status words down, moving out the oldest error, to make room for this new error date.

It then moves the new error data into the engineering file in the buffer.

It increments the error counter for the error code.

It writes out the updated file into the engineering file on mass storage.

If there are more errors in the error table it goes back to handle the next error.

If there are no more errors in the error table it clears the error table, resets the error table pointer and exits.

43.3 Program Name: Engineering File Display SELFS

43.3.1 Function:

The purpose of this program is to display the engineering file.

43.3.2 Entry Point Names:

SELFS - Program Entry

43.3.3 Externals Description

CORTBL - Entry address for the core resident engineering files (for the mass memory devices)

MASNM - Number of mass memory devices in system

NUMLU - Number of logical units in system

43.3.4 Entry Interfaces

There are no input parameters on entry.

43.3.5 External Interfaces

There are no output parameters

43.3.6 General Program Information:

This program is mass memory resident

The display of the engineering file is on the standard print output device.

This program can be called by:

1. Manual Interrupt
2. Type: SELFS

The file is printed. On completion hit Manual Interrupt to continue the system running.

43.3.7 General Design Peculiarities

This program uses the word addressable disk driver.

This is a mass memory resident program.

43.3.8 Program Logic

The program stores its own location and initializes counters.

Then it locates the engineering file on mass storage and reads in a file.

If the file is empty it goes to get the next file.

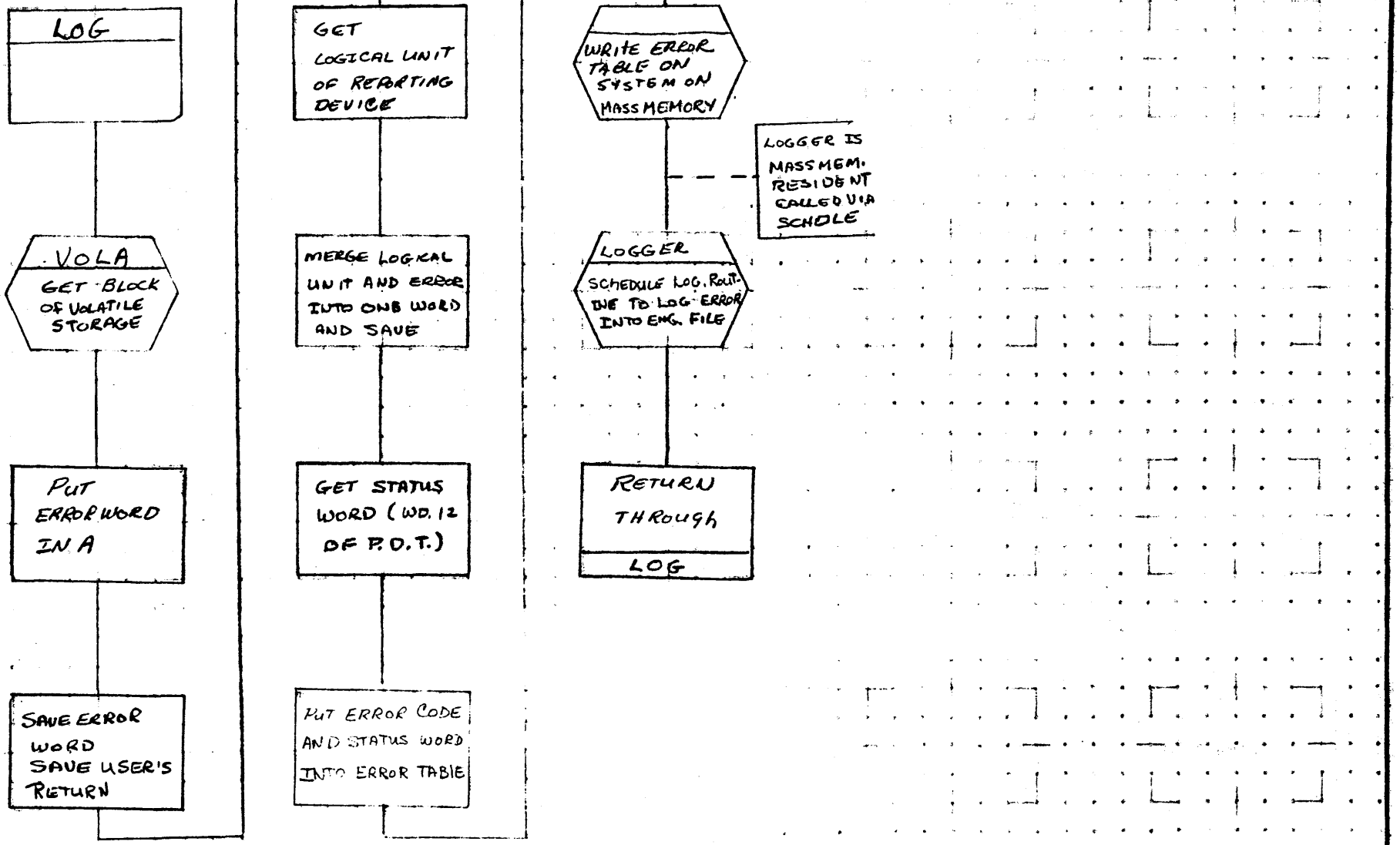
If the file is not empty it moves the file name into the printing buffer and prints it.

It then continues printing the file; converting the status words and error codes attaching labels and inserting spacing until the file is printed.

When the complete file is printed it goes to get the next file.

In getting the file it checks to see if it has processed the last file on mass storage. If it has not it returns to process the next file.

If all files on mass storage are processed it processes the engineering file in core {those kept for the mass memory devices}. When these are processed the program releases itself and exits.



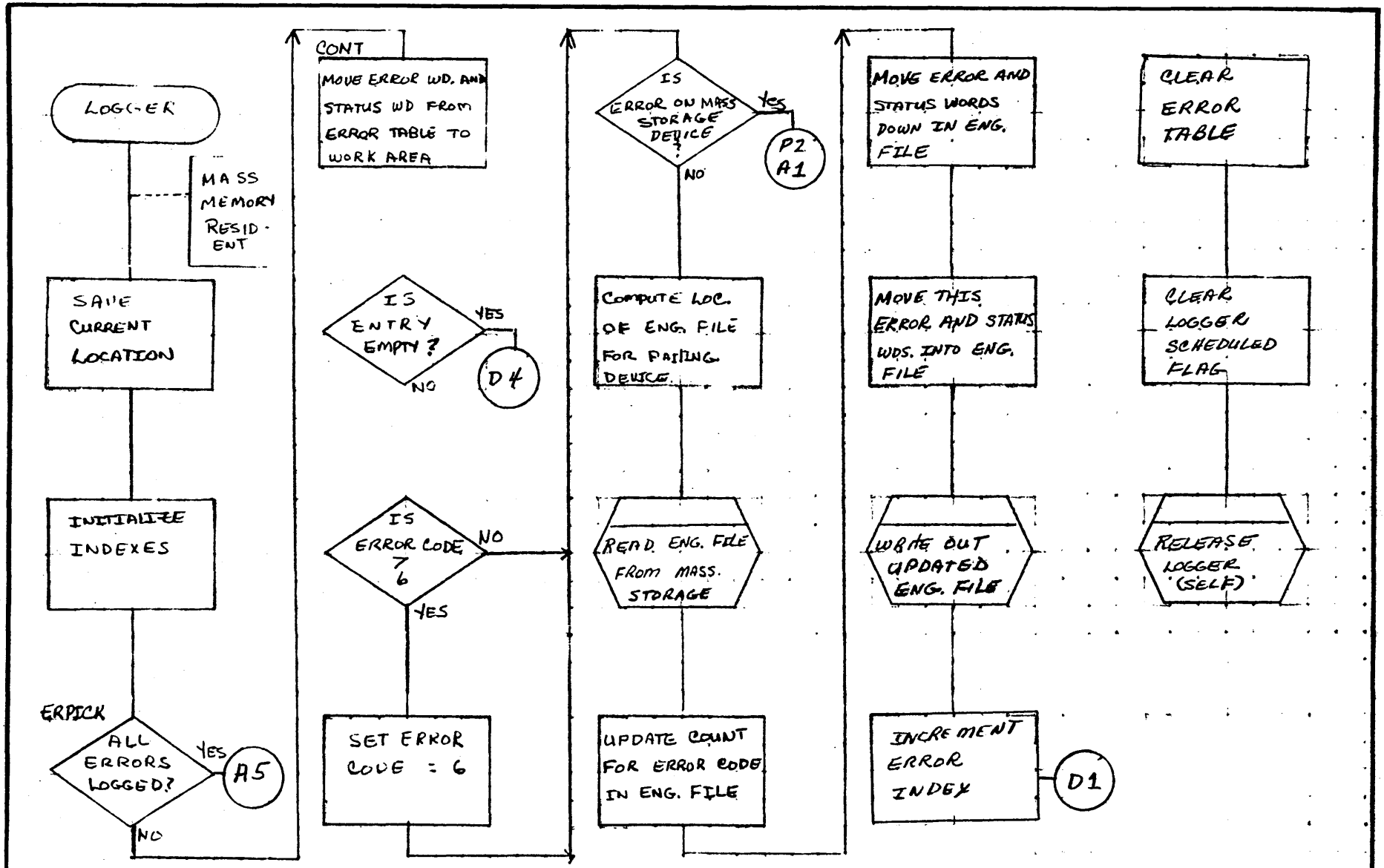
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LOG		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

43.1

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



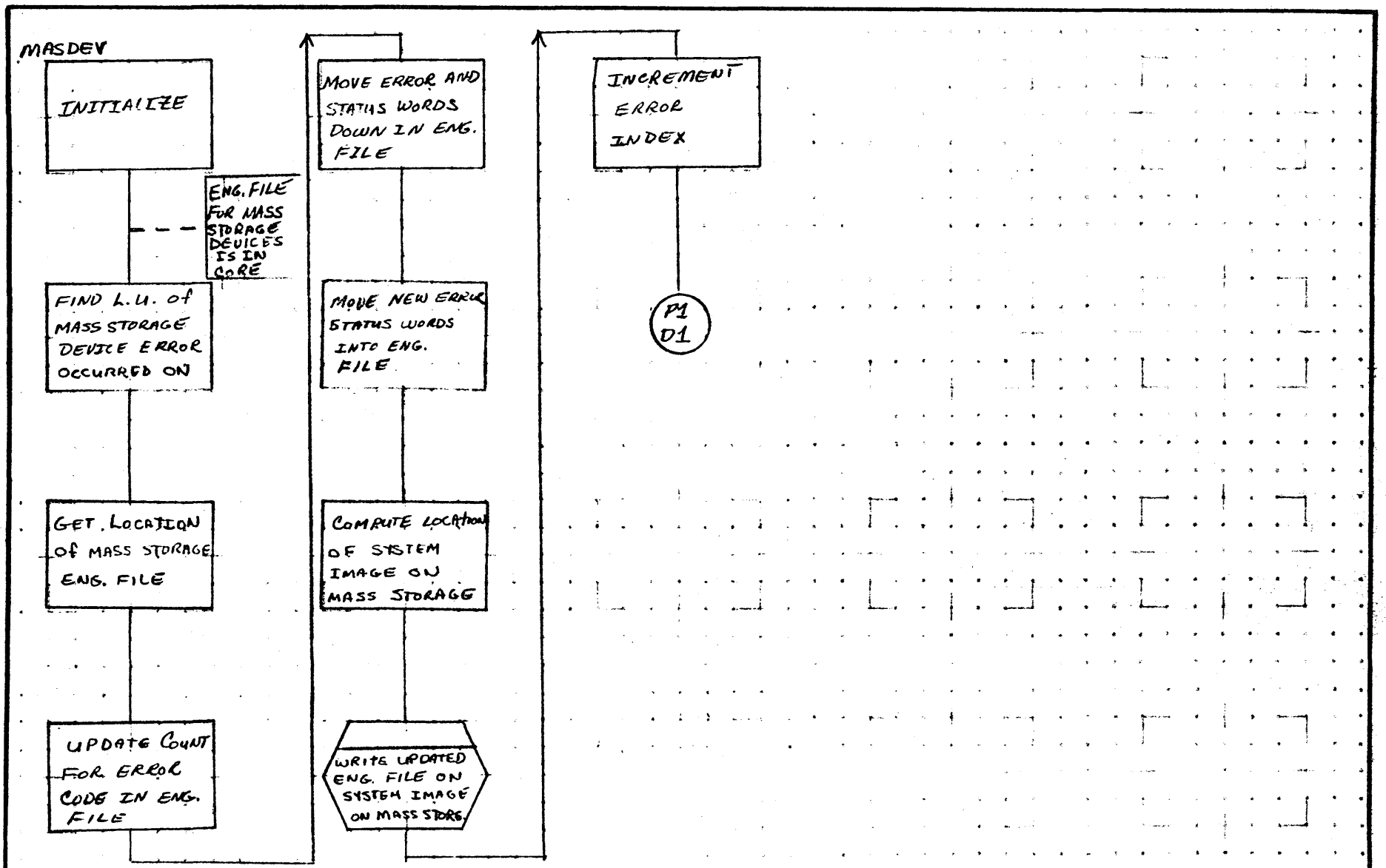
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LOGGER		PROJECT MGR		
	MSOS 3.0	PAGE 1 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA 385 (FORMERLY CA127)

PRINTED IN U.S.A.

MAR 5 1971

4.3.2



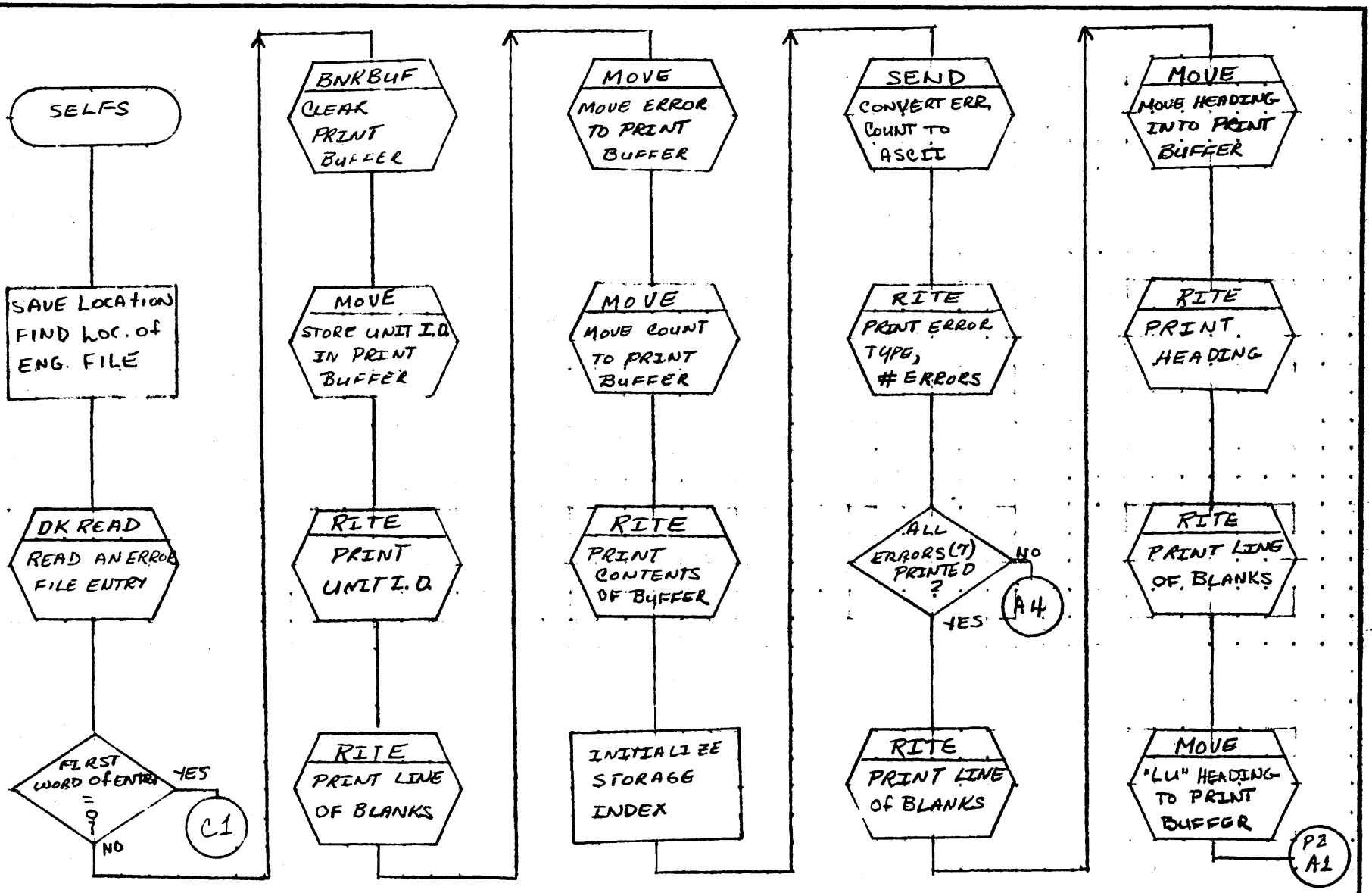
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: LOGGER		PROJECT MGR.		
	MSOS 3.0	PAGE 2 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

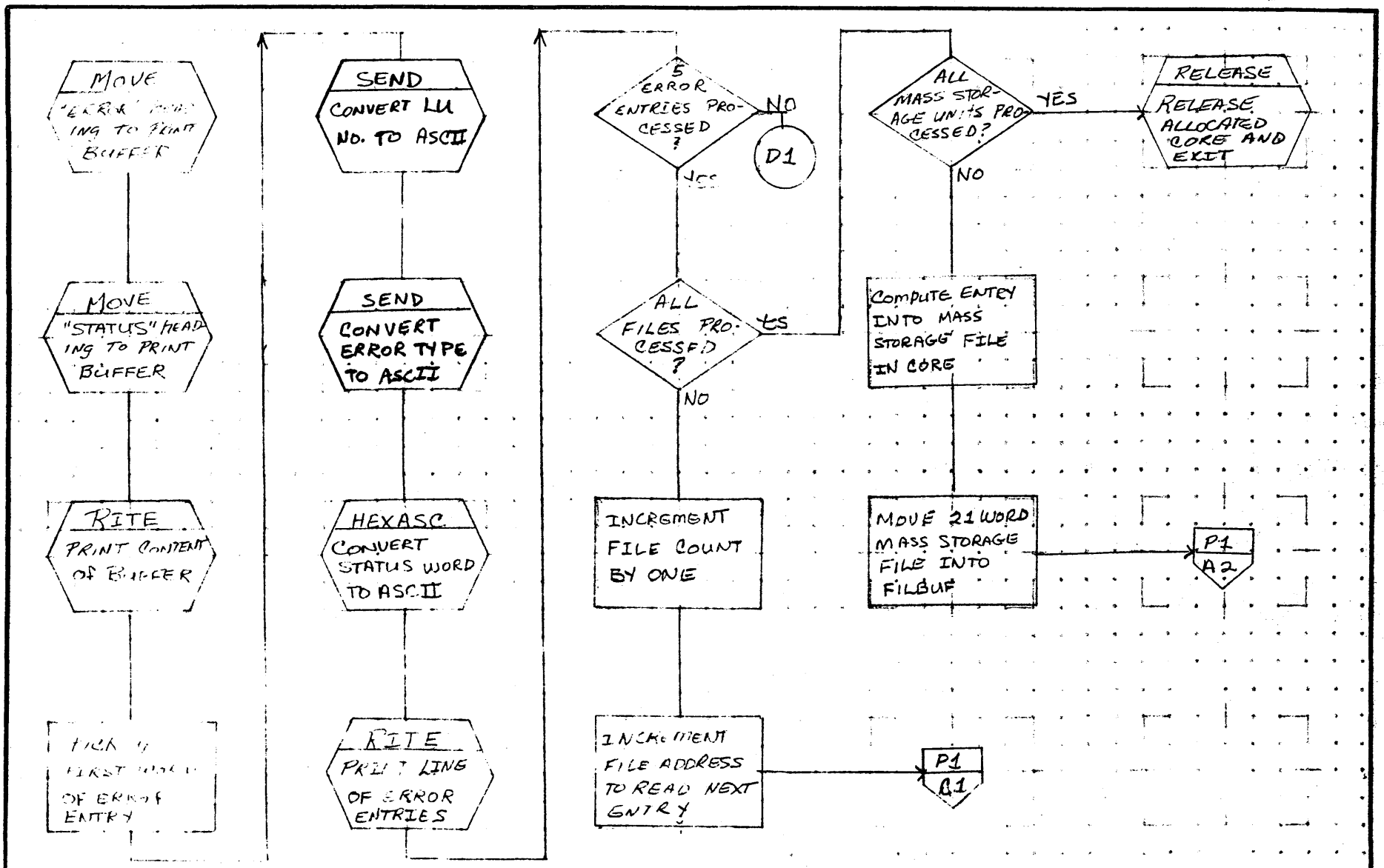
4.3.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SELFS		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

43.4



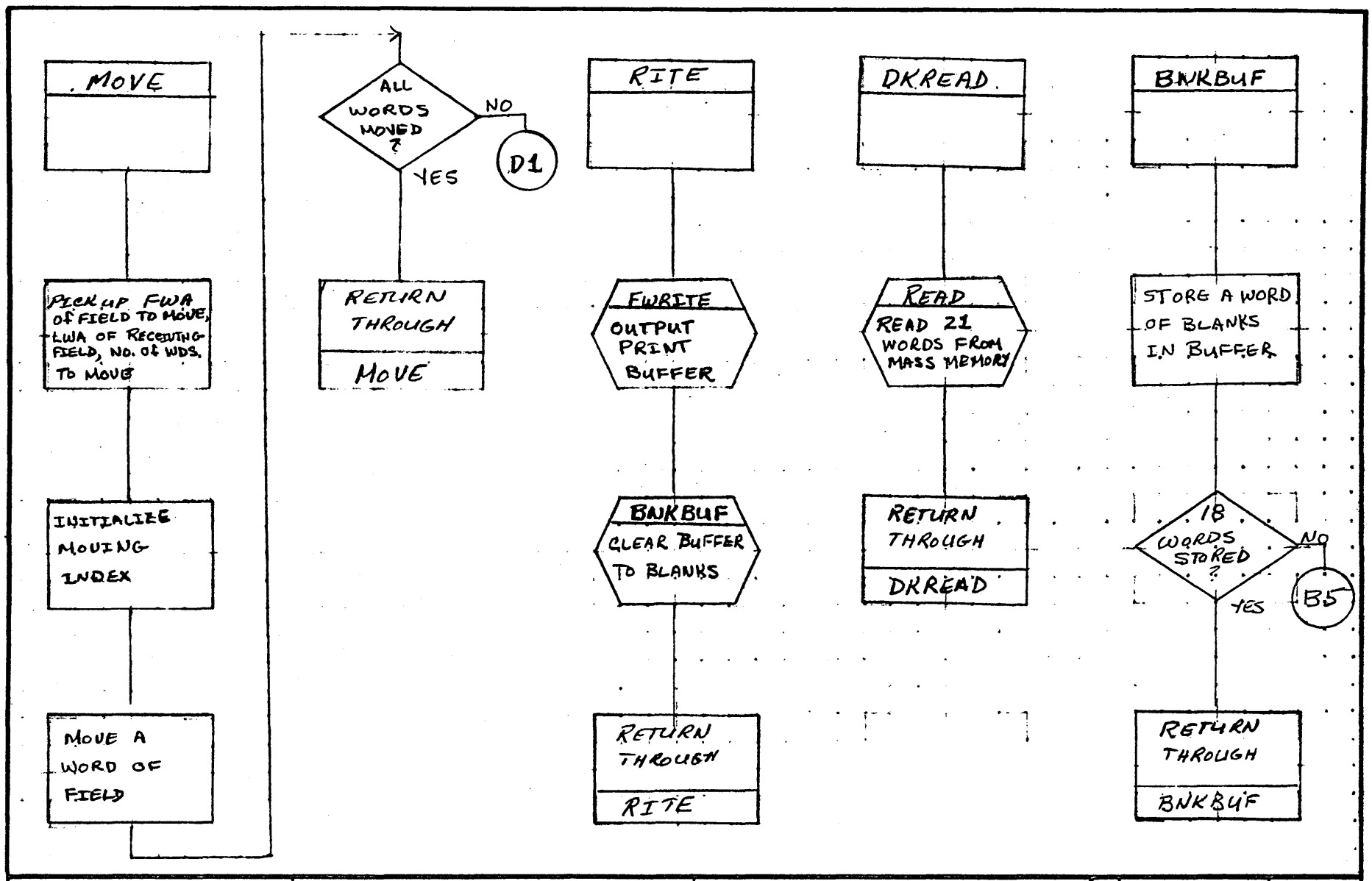
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE SELFS		PROJECT MGR.			
	MSOS 3.0	PAGE 2 OF 5	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

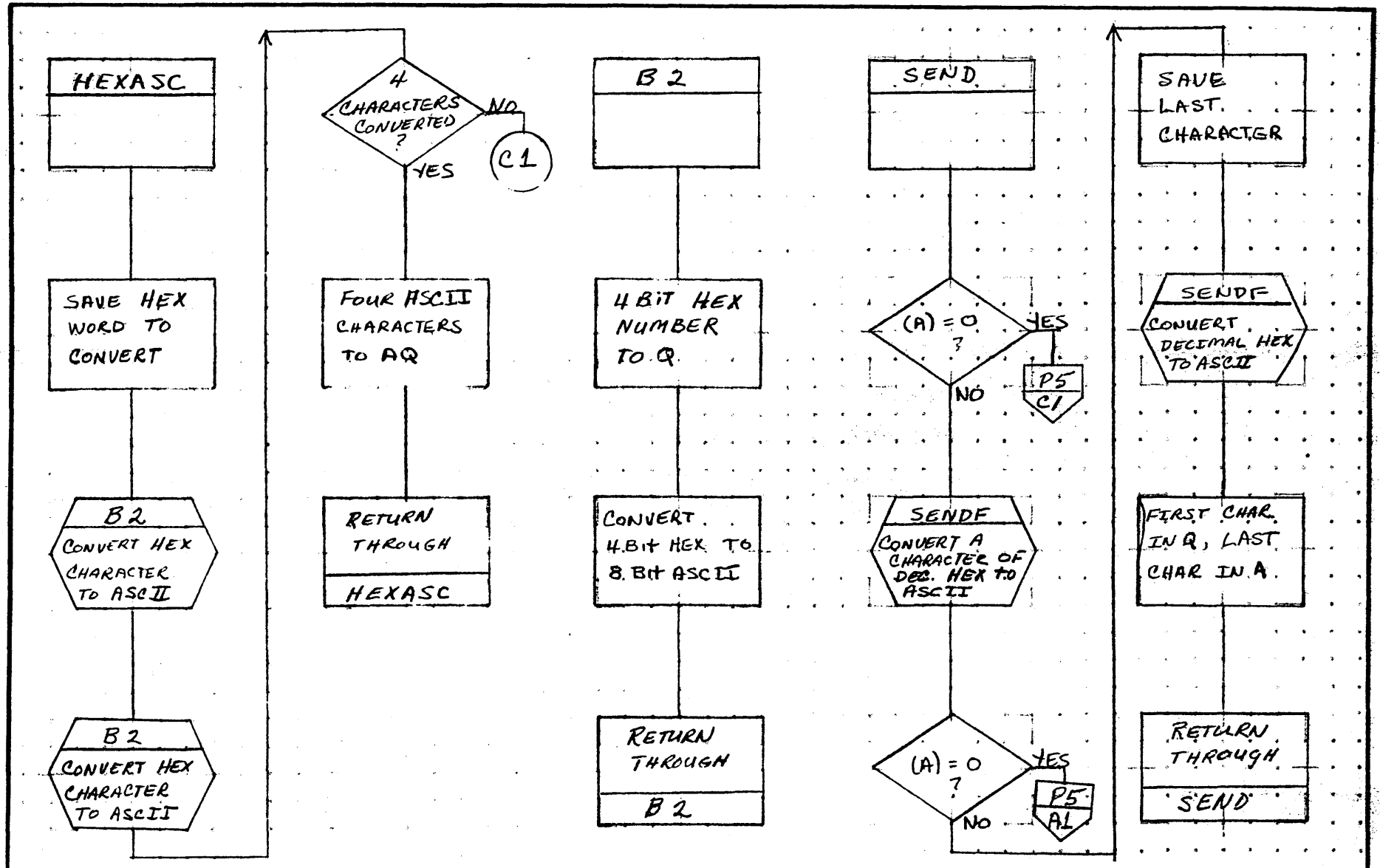
43.5



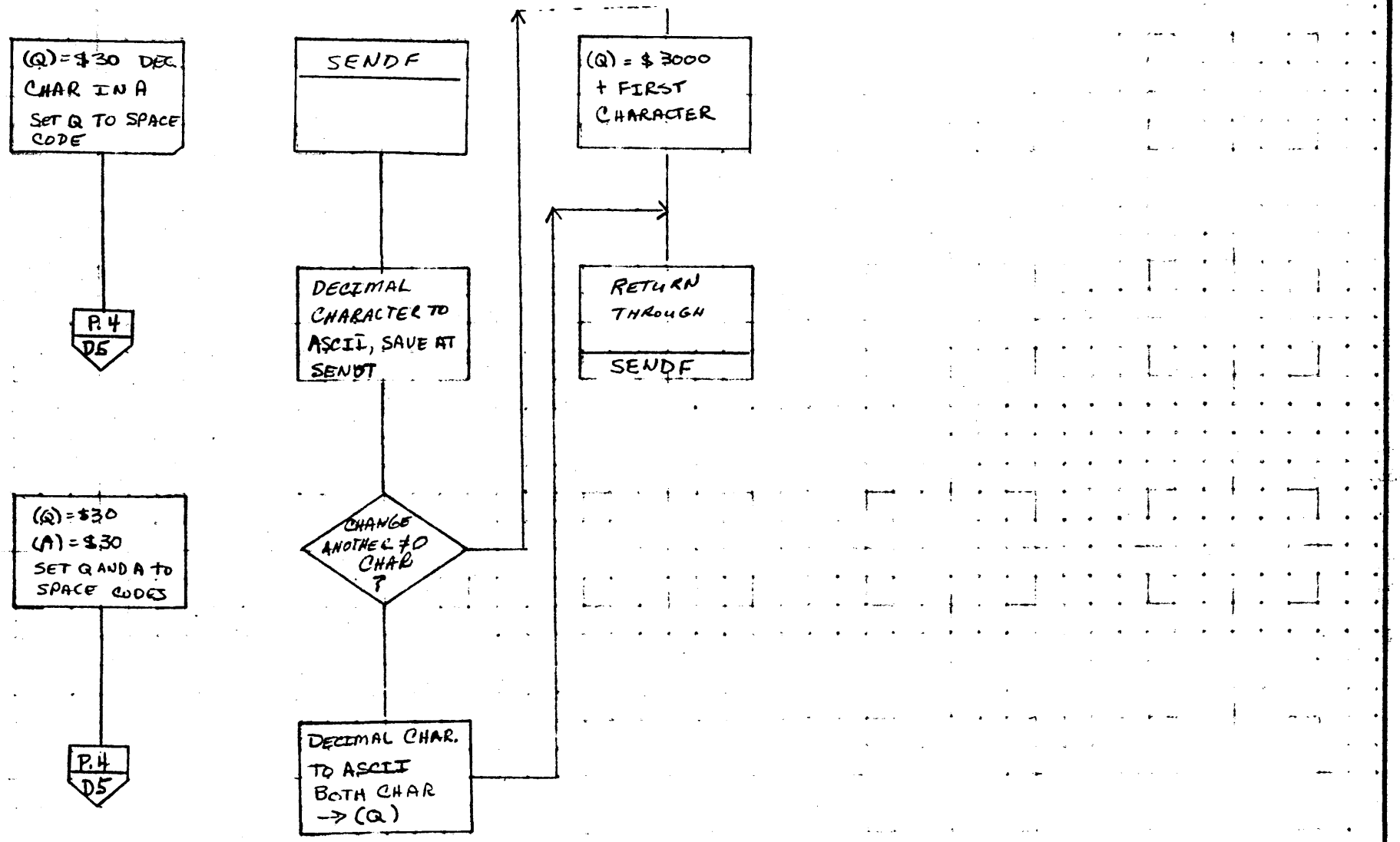
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE SELFS		PROJECT MGR			
	MSOS 3.0	PAGE 3 OF 5	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

43.6



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE SELFS		PROJECT MGR.		
	MSOS 3.0	PAGE 4 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: SELFS		PROJECT MGR			
	MSOS 3.0	PAGE 5 OF 5	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

4.3.8

44.0 1711 TELETYPE DRIVER

44.1 ENTRY POINTS

TYPEI	initiator entry
TYPEDR	continuator entry
TYPERR	error entry

44.2 EXTERNALS

None

44.3 ENTRY INTERFACES

Q = address of the PHYSTB

44.4 EXTERNAL INTERFACES

None

44.5 GENERAL PROGRAM INFORMATION

44.5.1 PHYSICAL DEVICE TABLE

The Teletypewriter driver refers to the physical equipment table entries with the following names:

CALL	= word 7
STATUS	= word 12
SWITCH	= word 9
CORE	= word 10
LASTP1	= word 11
TEMP	= word 15
COREIN	= word 13
ERRTAB	= word 8
TIME	= word 4

CALL:	Q setting to get unit status.
STATUS:	equipment status.
SWITCH:	Switch settings listed below.
CORE:	address into which the next data word is to be stored.
LASTP1:	the last core location plus one to be stored into.
TEMP:	Temporary storage.
COREIN:	Original starting location in case cancel is used.

ERRTAB: request status word.
TIME: Diagnostic clock location.

▽SWITCH▽ bits are used as follows:

Read

bit 0 - zero for read
1 - one if formatted
2 - one if lower
4 - one for pass switch
5 - one for cancel
6&7 - null count for turning on the break light

Write

bit 0 - one for write
1 - one if formatted
2 - one if lower
3 - one if operation complete
4 - zero if 1st character {to send carriage return}
5 - zero for control {to send line feed}
6&7 - cancel count - to send 3 cancel characters after TAB, VTAB & FORM

44.6 DESCRIPTION

44.6.1 INITIATOR

TYPEI is the initiating entry to the driver. At entry, routine FNR is called. If no request is found exit is made to the dispatcher.

If one is found the PHYSTBS are initialized. Write mode if set and interrupt on data, alarm and EOT selected. The diagnostic clock is set, a cancel character is output to cause an interrupt and exit is made to the dispatcher.

44.6.2 CONTINUATOR

TYPEDR is the continuator or interrupt time entry. At entry, if caused by manual interrupt, the pending manual interrupt bit is set, interrupts are cleared and interrupt on data, alarm and EOT selected. If the unit is busy and read mode is set, a character is input. If the unit is not busy, pending manual interrupt is cleared. If the unit is in use, it is marked not in use and the location of the interrupted request put in Q. If not in use, zero is put in Q. Exit is then made to the manual interrupt module MANINT.

44.6.3 ERROR ENTRY

If entry was not due to manual interrupt but was caused by alarm or diagnostic clock procedure ERROR is entered. This procedure sets the error field of PHYSTBQ and enters the completion procedure.

44.6.4 COMPLETION

If the interrupt was EOP or the completion switch is set, the completion routine is entered. This procedure clears interrupts, resets the diagnostic clock, schedules the completion routine and re-enters TYPEI.

44.6.5 DATA TRANSFER OPERATION

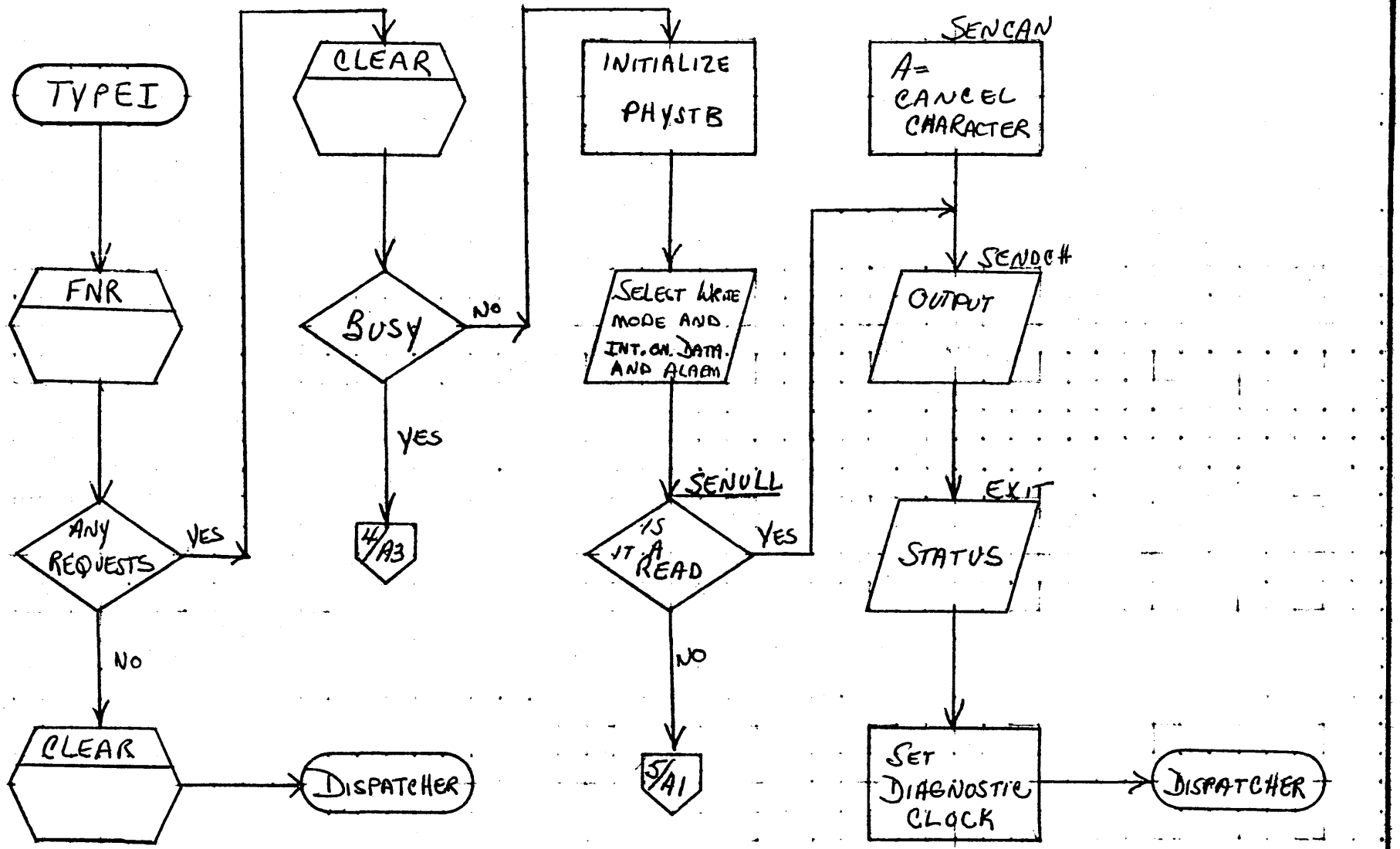
If entry was due to data interrupt, status is checked to see if the teletypewriter motor is on. If not, a cancel character is sent and exit is made to the dispatcher. If the motor is on and this is a write operation, the WRITE procedure is entered. If it is a read operation and the unit is busy (a character has been typed) procedure READ is entered. If it is a formatted read, null count is checked to see if two nulls have been sent out to turn on the break light. If not, a null is sent out and exit is made to the dispatcher. If the nulls have been sent out, interrupts are cleared, read mode is set, interrupt on data, alarm and EOT is selected and exit is made to the dispatcher.

If it is a write operation and the format switch is set, the 1st character switch is checked. If it is set a carriage return is sent, the 1st character switch is not set, the control switch is reset and exit is made to dispatcher. If the control switch was not set the spacer count is checked. If it is non zero, it is decremented, a cancel character is sent out and exit is made to the dispatcher. If the spacer count was zero or the write was not formatted, the data word from core is placed in Q. If the upper character switch is set, Q is shifted right 8 bits. The upper 9 bits of Q are then cleared. If the character is a null (zero), $7F (cancel) is substituted. If the character is a TAB, VTAB or FORM, the spacer count is set to 3. The character is then sent out. If this is the last word (unformatted single character output) the completion switch is set and exit is made to the dispatcher. If it is now set for upper, the core location is incremented. If it is now the last location, the completion switch is set. The dispatcher is then entered.

If the operation was read and the unit is busy, the character is input. If the unit is not marked in use exit is made to the dispatcher. If it is in use the parity of the character just input is checked. If it is incorrect exit is made to procedure ERROR.

If parity is correct the format switch is checked. If set, the character just input is checked. If it is a line feed exit is made to the dispatcher. If it is a carriage return the cancel switch is checked. If it is not set the common completion routine is entered. If it is set the input buffer is backgrounded to all $FFFF. Core location is set back to the original setting effectively repeating the request. If it is a cancel character the cancel and pass switches are set and exit is made to the dispatcher. If it is not a cancel character the pass switch is checked. If it is set exit is made to the dispatcher. If the pass switch is not set or the request is not formatted $FF is put into the upper half of the word containing the character just input. The upper lower switch is then checked. If set for upper the word is shifted left 8 bits and stored into the specified core location. If set for lower the word is added with the specified core location. If this is the last word, this was an unformatted single character input operation and exit is made to the common completion routine. If not,

the upper lower switch is reversed. If now set for lower exit is made to the dispatcher. If now set for upper the core location is incremented. If it is not now the last word exit is made to the dispatcher. If it is the last word and the input is unformatted exit is made to the common completion procedure. If the operation is formatted the pass switch is set and exit is made to the dispatcher.

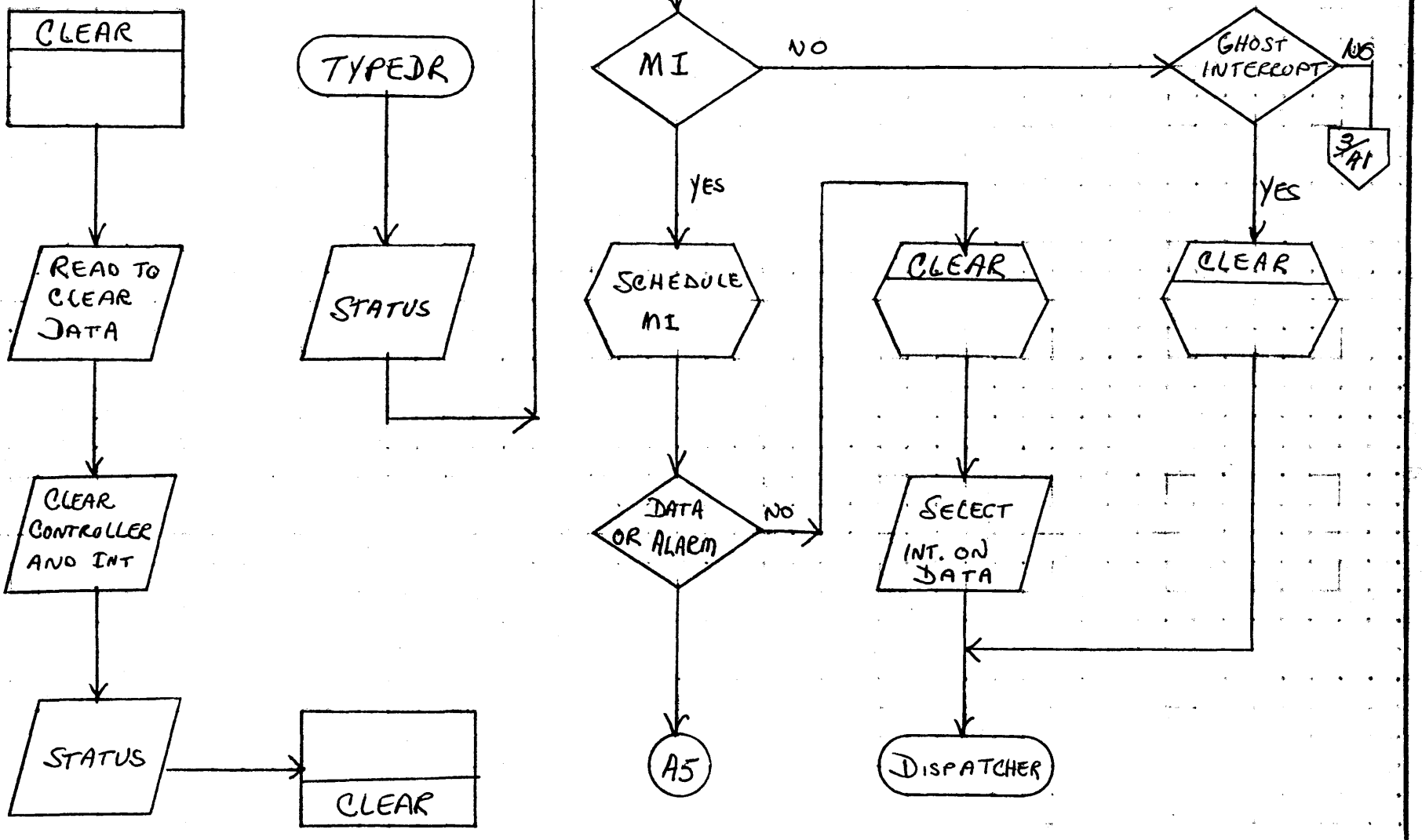


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1711 TELETYPE		PROJECT MGR		
FLOWCHART ☐		PAGE 1 OF 10	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

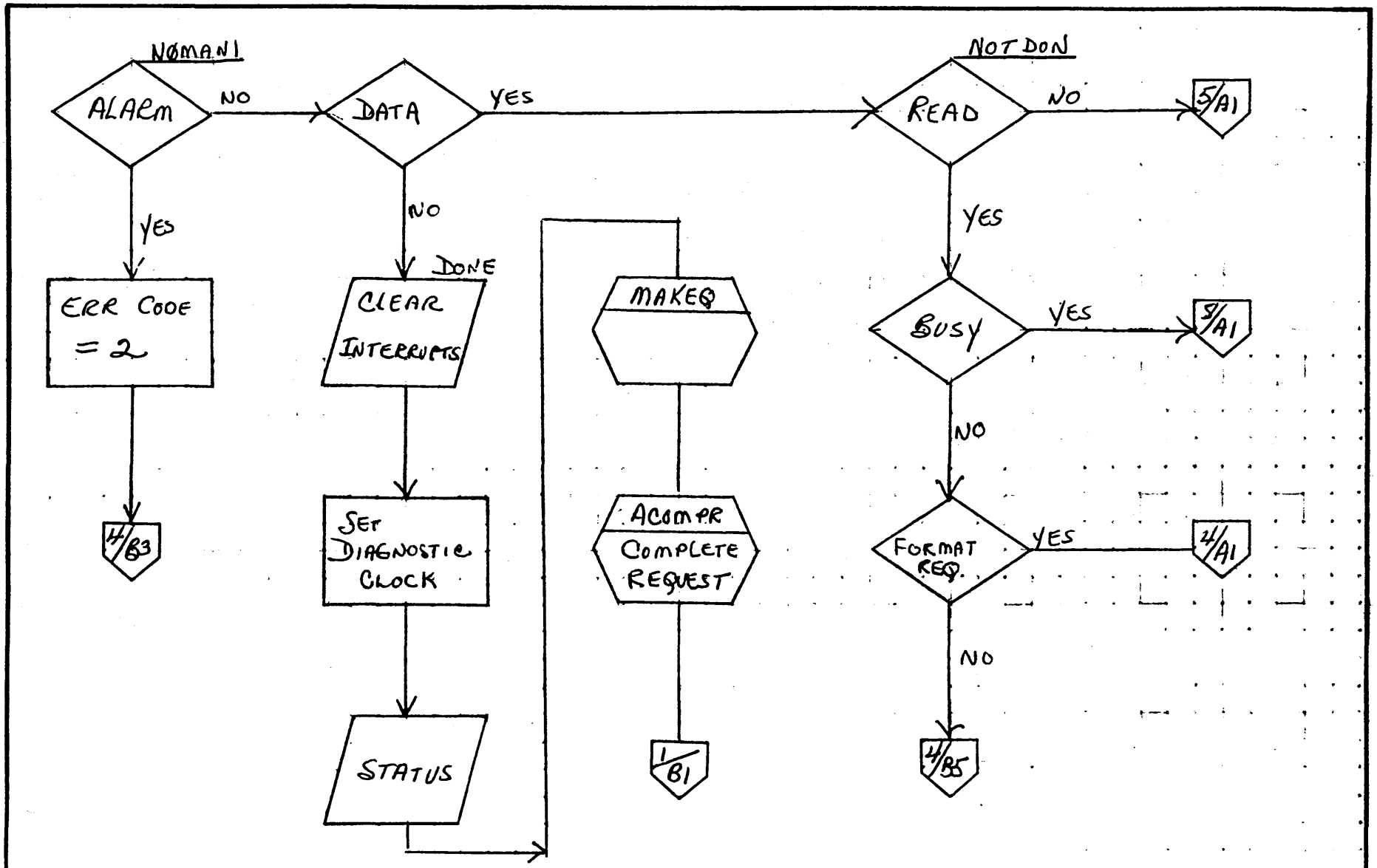
MAR 5 1971

AA1385 (FORMER CA 27

PRINTED IN USA



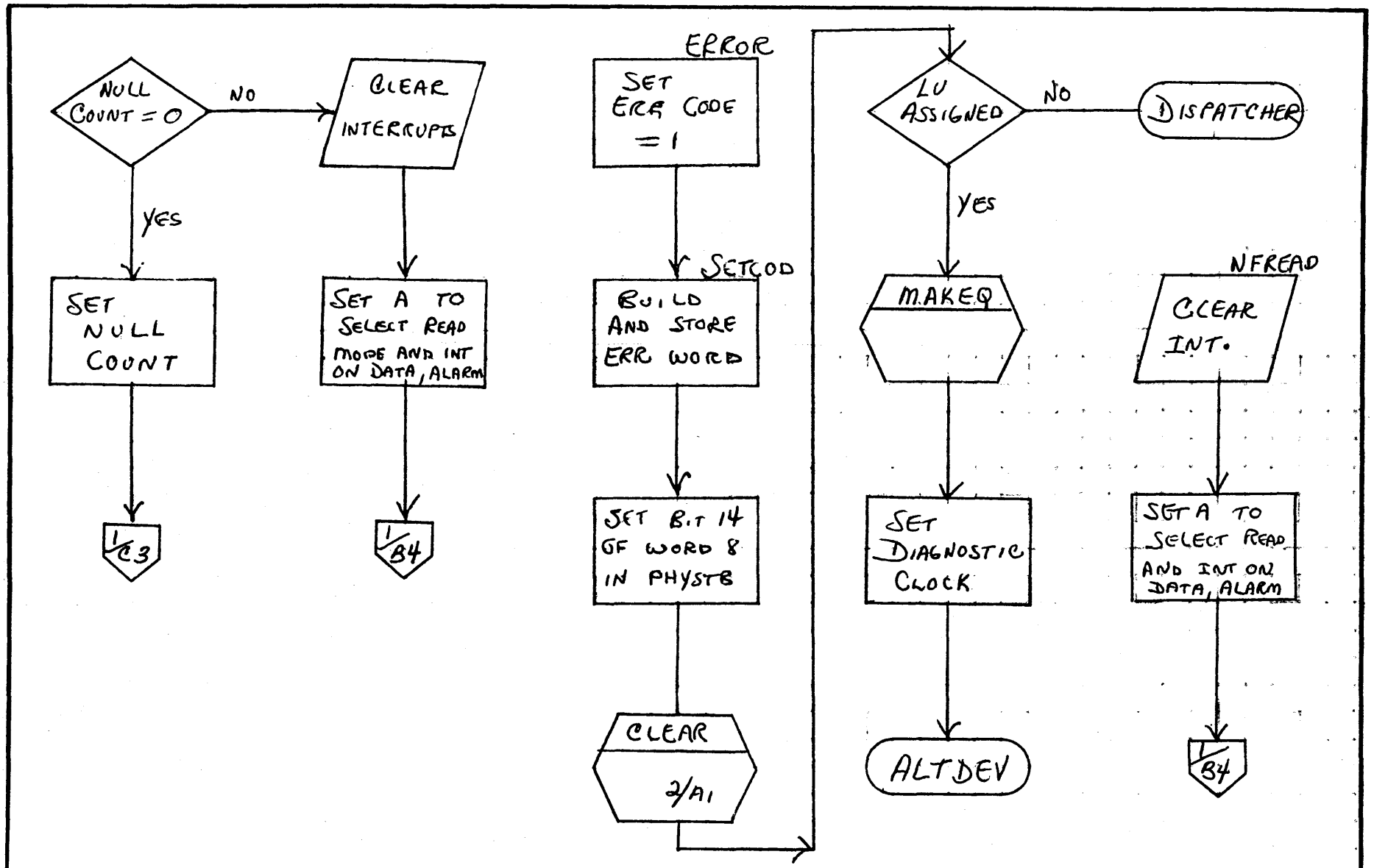
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1711 TELETYPE	PROJECT MGR			
	PAGE 2 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1711 TELETYPE		PROJECT MGR		
FLOWCHART ☐		PAGE 3 OF 10	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

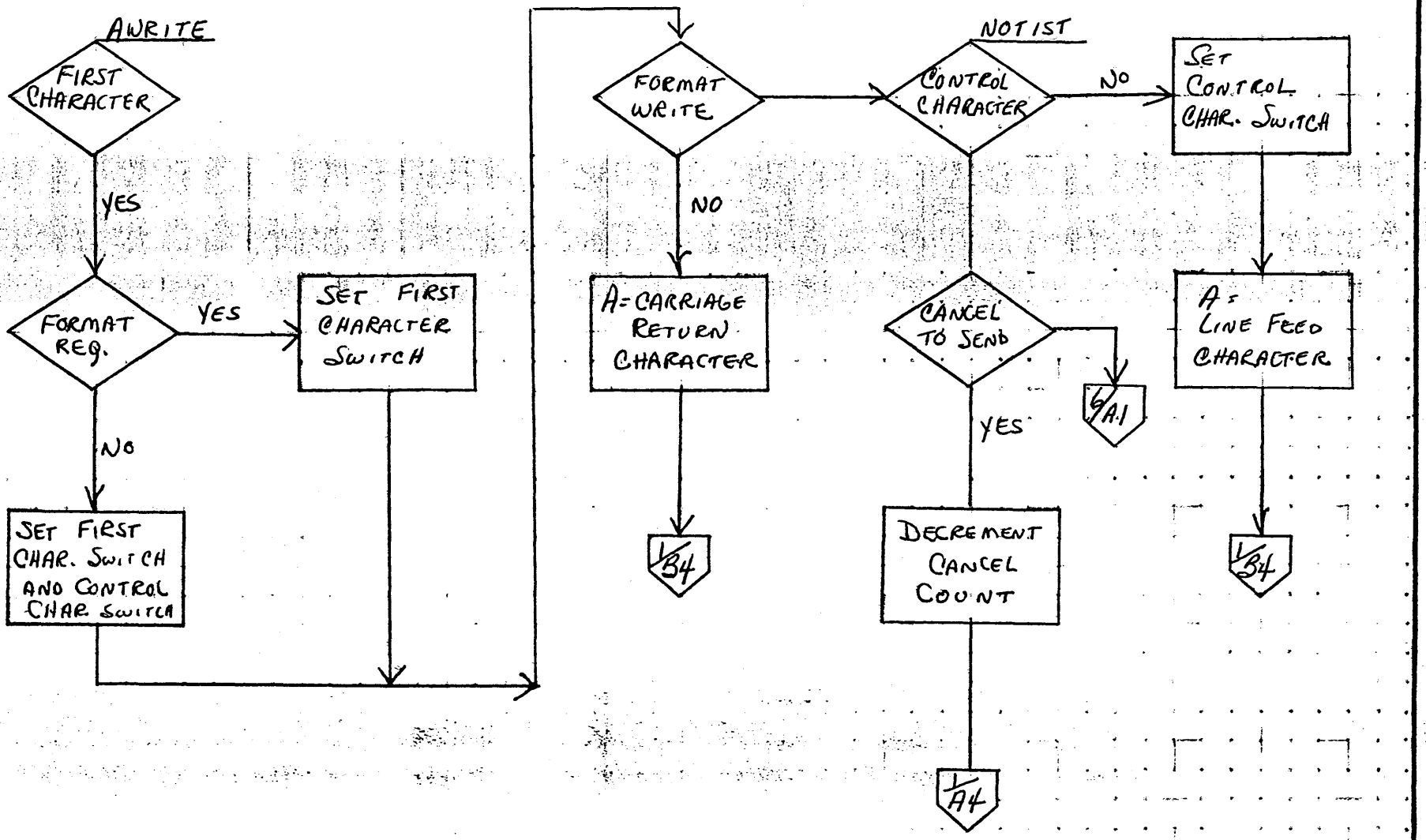
MAR 5 1971

44.8



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700 DOCUMENT TITLE 1711 TELETYPE PAGE 4 OF 10 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO PROJECT MGR PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

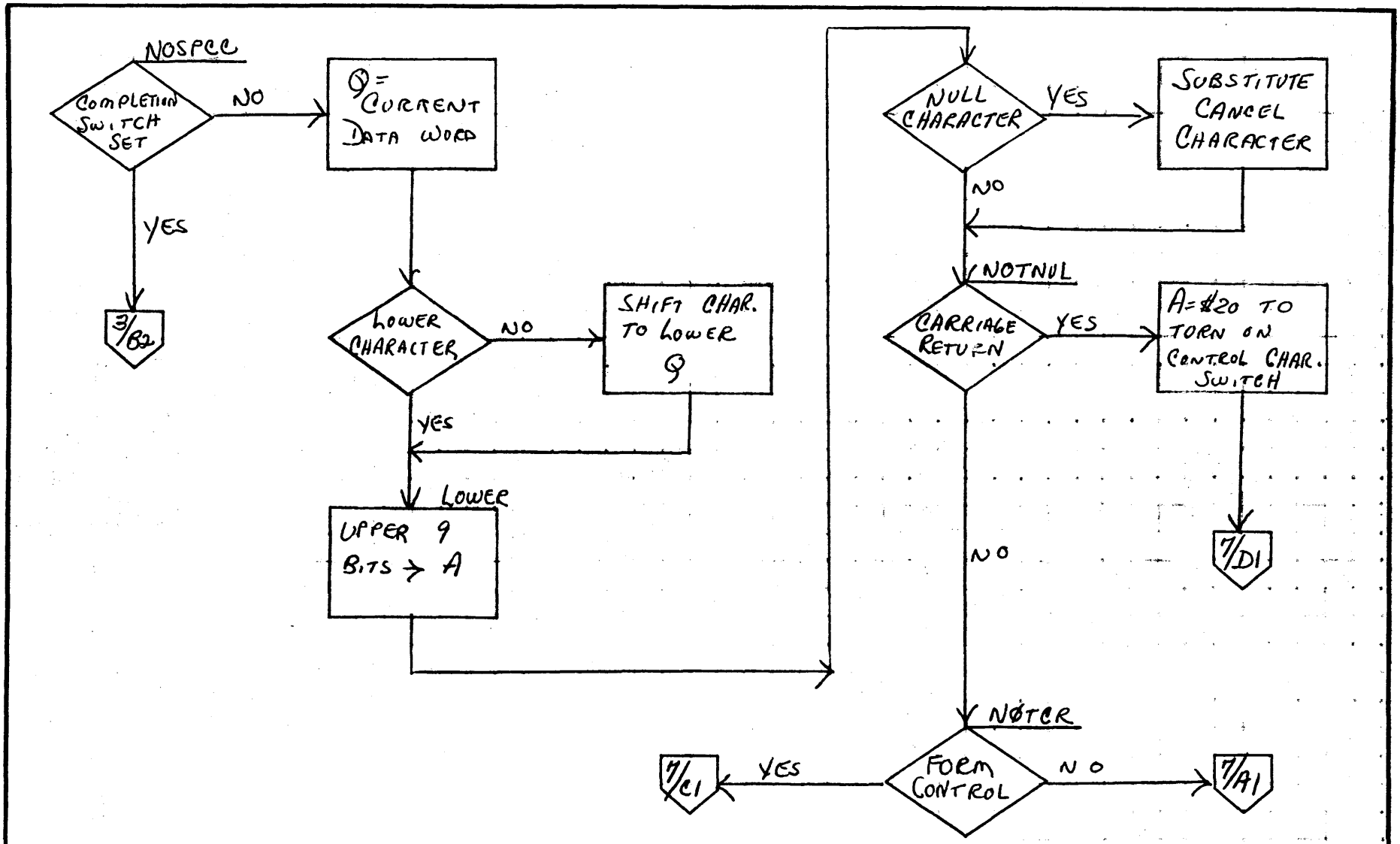
MAR 5 1971



CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE 1711 TELETYPE		PROJECT MGR.			
SAMPLE CODE ☐		PAGE 5 OF 10	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

MAR 5 1971

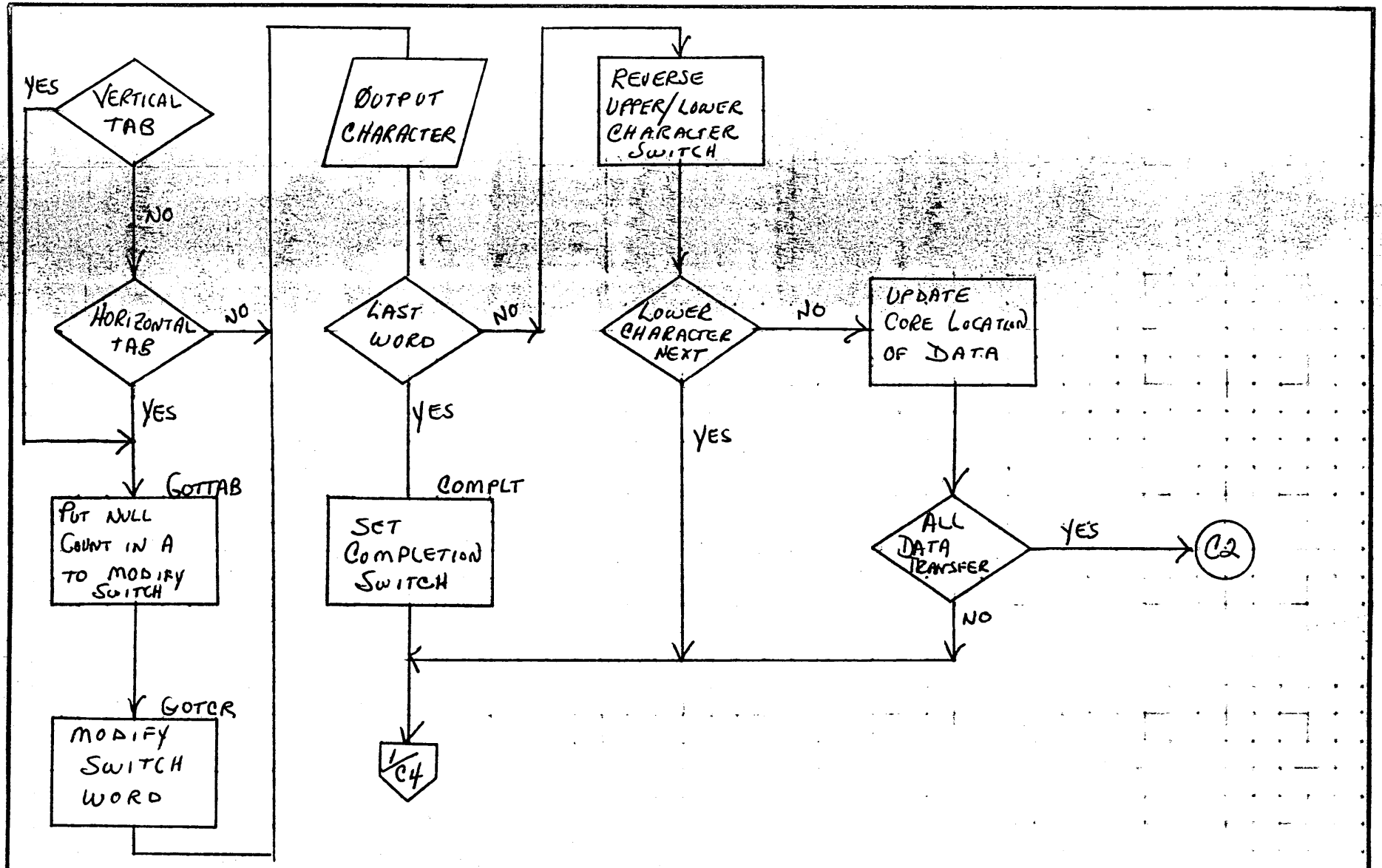
44.10



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1711 TELETYPE	PROJECT MGR.			
	PAGE 6 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

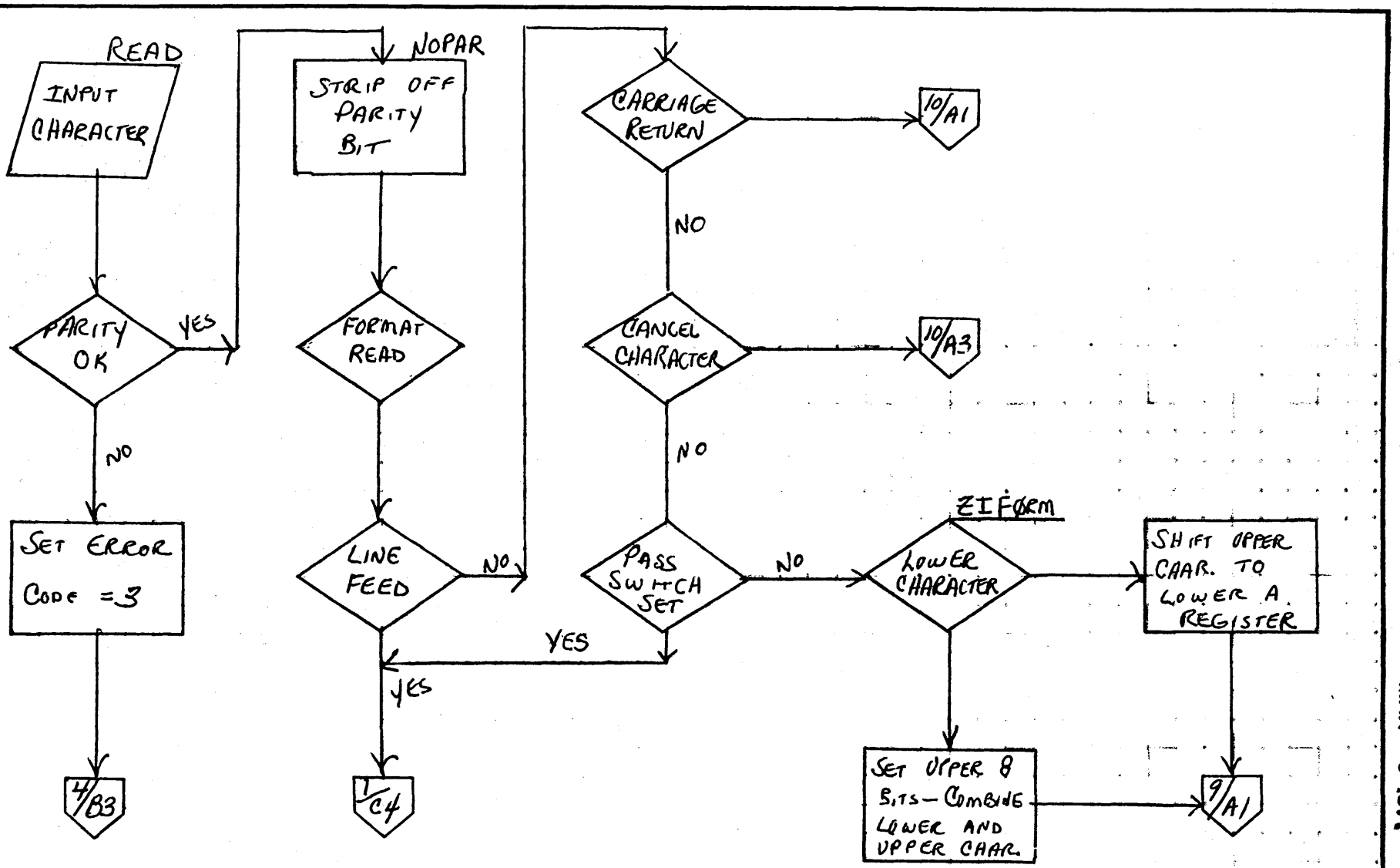
44.7.14



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1711 TELETYPE		PROJECT MGR.			
FLOWCHART ☐		PAGE 7 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

44.12



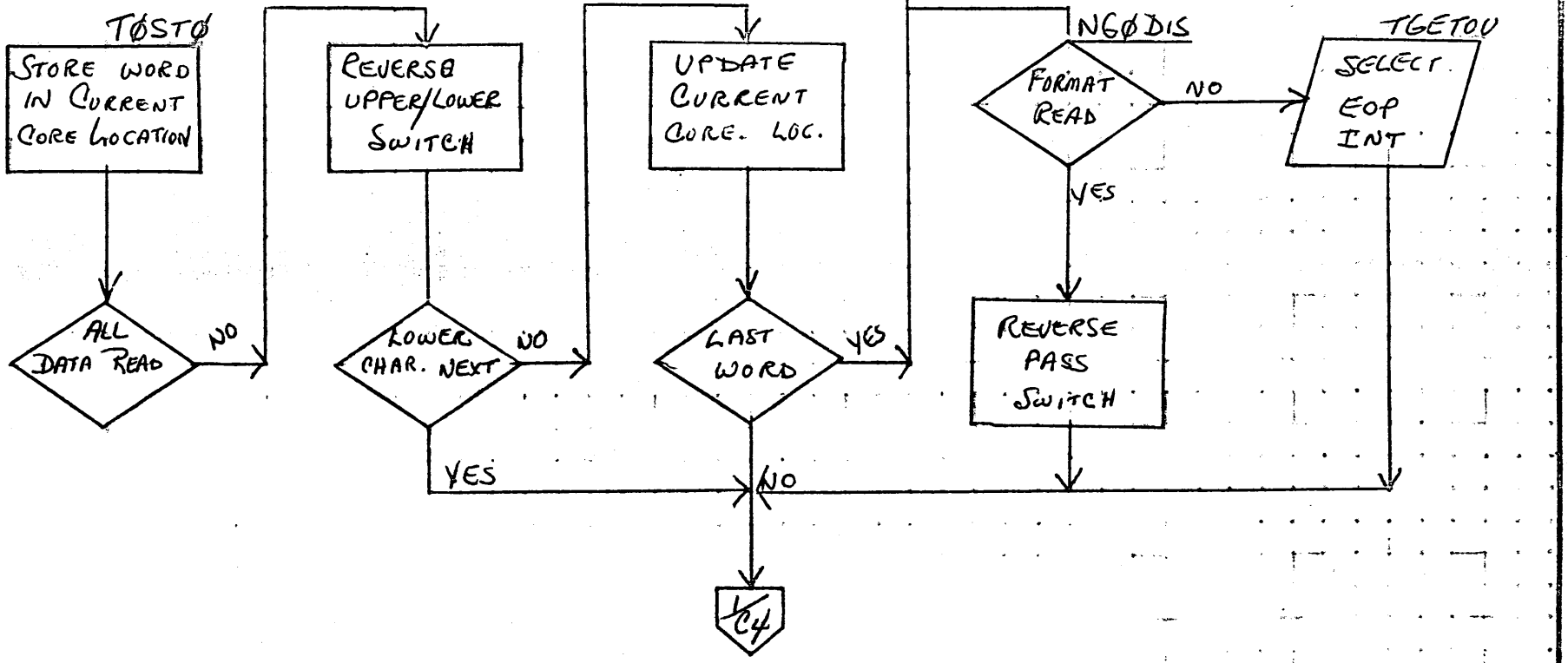
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1711 TELETYPE		PROJECT MGR			
FLOWCHART ☐		PAGE 8 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

4.4.13

AA1385 (FORMERLY CA127-1)

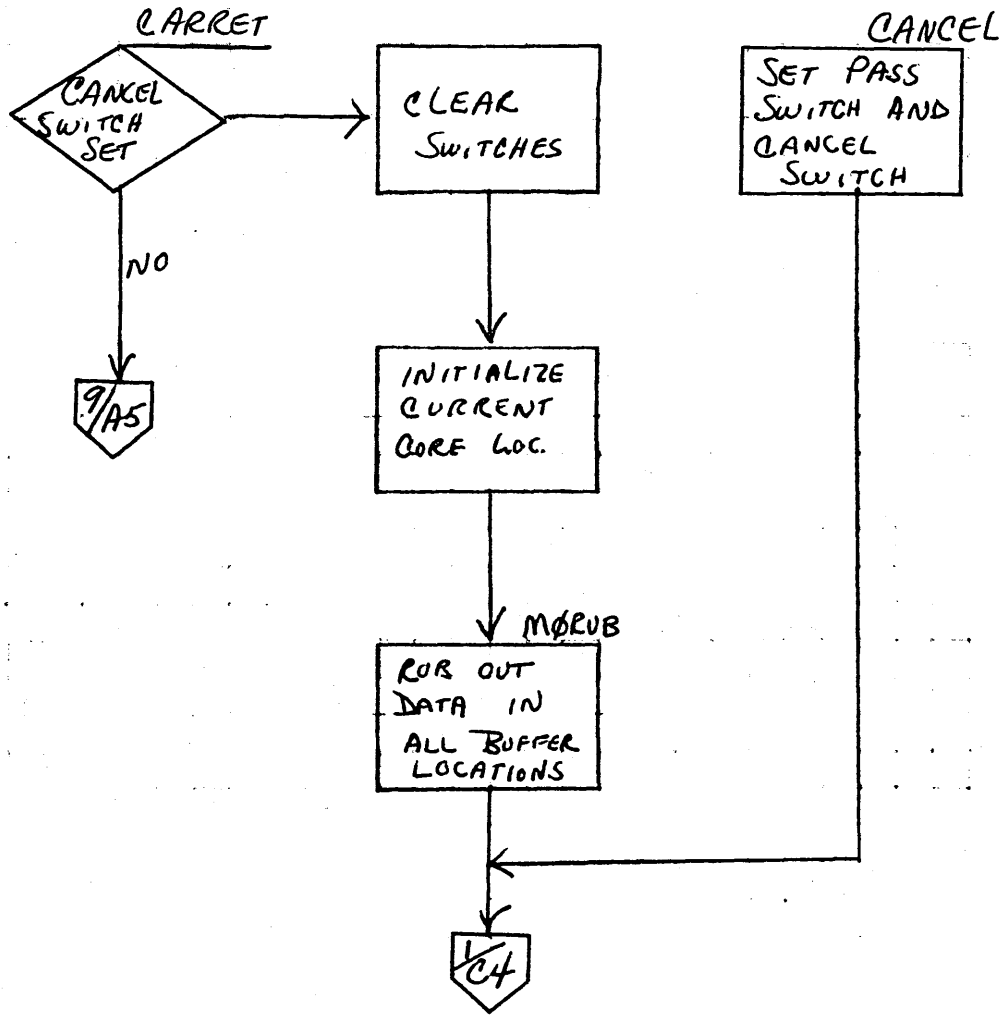
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1711 TELETYPE	PROJECT MGR.			
	PAGE 9 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

44.14



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 711 TELETYPE		PROJECT MGR.		
FLOWCHART ☐		PAGE 10 OF 10	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

44.15

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

45.0 1713 TELETYPEWRITER DRIVER

45.1 The 1713 Teletypewriter Driver operates under the CONTROL DATA 1700 3.0 Operating System to provide the capability for keyboard or paper tape reader input and printer or paper tape punch output. The keyboard/printer module is core resident. The paper tape reader module and the paper tape punch module are mass memory resident.

45.2 MODULES

The 1713 Teletypewriter Driver is composed of 4 modules:

1. MASDRV - Mass Memory Driver Control program.
2. S13001 - Keyboard/Printer Driver.
3. S13002 - Paper Tape Reader Driver.
4. S13003 - Paper Tape Punch Driver.

45.3 1713 KEYBOARD/PRINTER - S13001

45.3.1 ENTRY SYMBOLS

S13KI	Driver initiator address
S13KC	Driver continuator address
S13KER	Driver error routine for diagnostic timer
SETCDI	Error routine which forms the error word for the Alternate Device handler.

45.3.2 EXTERNAL SYMBOLS

MAS500	Entry point in MASDRV/DBLDRV. It will queue the keyboard/printer if the 1713 is busy on entry to the initiator of the driver.
MI	Entry point in MINT, which will process manual interrupts.
MAS300	Entry in MASDRV/DBLDRV. It will check if any mass memory drivers are waiting to use core.
S13MOD	Entry in the systems tables and parameters. Contains the current setting of the mode switch.

S13BZY Entry in the systems tables and parameters. Contains logical unit associated with the module currently busy with the 1713. It will be zero if the 1713 is not busy.

MAKEQ Routine that sets the proper error bits in Word 9 of PHSTAB to be passed to user.

ALTDEV Program that processes irrecoverable errors.

45.3.3 ENTRY INTERFACES

Q = address of the PHYSTB

45.3.4 PHYSICAL DEVICE TABLE ENTRIES

CALL	7	- Q setting to get unit status.
STATUS	12	- equipment status.
SWITCH	9	- switch settings listed below.
CORE	10	- address into which the next data word is to be stored.
LASTPL	11	- the last core location to be stored into plus one
TEMP	15	- temporary storage for character just read.
COREIN	13	- original starting location in case cancel is used.
ERRTAB	8	- request status word.
TIME	4	- diagnostic clock location.
LU	5	- logical unit currently assigned to the device.

°SWITCH° bits are used as follows:

Read

bit 0 - zero for read
1 - one if formatted
2 - one if lower character
4 - one for pass characters until carriage return if formatted
5 - one for cancel
6&7 - null count for turning on the break light

Write

bit 0 - one for write
1 - one if formatted
2 - one if lower
3 - one if operation complete
4 - zero if 1st character {to send carriage return}
5 - zero for control {to send line feed}
6-9 - cancel count - to send 15 cancel characters after TAB, VTAB & FORM

45.3.5 KEYBOARD DESCRIPTION

45.3.5.1 INITIATOR

S13KI is the initiating entry to the driver. At entry the 1713 is checked for busy, if yes, exit is made to MAS300.

If a request is found a check is made to determine if the device is in keyboard mode. If not, keyboard mode and interrupt on EOP are selected and exit is made to the dispatcher.

When the device is already in keyboard mode or keyboard mode has been selected and an EOP interrupt generated to indicate the switch of mode, the PHYSTAB locations are initialized. Write mode is set and interrupt on data and alarm selected. If the operation is a read operation, the diagnostic clock is set and a null character is output to cause an interrupt and exit is made to the dispatcher. If the operation is a write, the write routine in the continuator is entered.

45.3.5.2 CONTINUATOR

Manual Interrupt

S13KC is the continuator entry. At entry, if caused by a manual interrupt the manual interrupt processor is scheduled and a check is made for data or alarm.

If neither of these conditions exist, the device is cleared, interrupt on data is selected, and exit is made to the dispatcher.

Mode Switch

If on entry an EOP interrupt was generated by a mode switch, the interrupt is cleared and the initiator is re-entered.

Error Entry

If the interrupt was caused by an alarm condition the error routine SETCOD is entered. This routine performs these functions:

1. Forms the error word for ALTDEV.
2. Sets the error bits for MAKQ.
3. Sets the counter for the diagnostic timer to -1.
4. Clears the 1713 busy flag.
5. Exits to the Alternate Device Handler (ALTDEV).

Completion

If the interrupt was EOP and not a mode switch or the completion switch is set, the completion routine is entered. This procedure clears interrupts, resets the diagnostic clock, schedules the completion routine, clears the 1713 busy flag and re-enters the initiator.

Data Transfer Operation

If entry was due to data interrupt, the switch word is checked to see if it is a read or write operation. If this is a write operation, the WRITE procedure is entered. If it is a read operation and the unit is busy (a character has been typed) procedure READ is entered. If it is a formatted read, null count is checked to see if two nulls have been sent out to turn on the break light. If not, a null is sent out and exit is made to the dispatcher. If the nulls have been sent out, interrupts are cleared, read mode is set, interrupt on data and alarm is selected and exit is made to the dispatcher.

If it is a write operation and the format switch is set, the 1st character switch is checked. If it is set a carriage return is sent, the 1st character switch is reset and exit is made to dispatcher. If 1st character switch is not set, the control switch is reset and exit is made to dispatcher. If the control switch was not set the spacer count is checked. If it is non zero, it is decremented, a cancel character is sent out and exit is made to the dispatcher. If the spacer count was zero or the write was not formatted, the data word from core is placed in Q. If the upper character switch is set, Q is shifted right 8 bits. The upper 9 bits of Q are then cleared. If the character is a null {zero}, $7F {cancel} is substituted. If the character is a TAB, VTAB, or FORM, the spacer count is set to 15. The character is then sent out. If this is the last word {unformatted single character output} the completion switch is set and exit is made to the dispatcher. If it is not the last character, the upper-lower switch is reversed. If the switch is now set for lower exit is made to the dispatcher. If it is now set for upper, the core location is incremented. If it is now the last location, the completion switch is set. The dispatcher is then entered.

If the operation was read and the unit is busy, the character is input. The parity of the character just input is checked. If it is incorrect exit is made to procedure SETCOD.

If parity is correct the format switch is checked. If set, the character just input is checked. If it is a line feed exit is made to the dispatcher. If it is a carriage return the cancel switch is checked. If it is not set the common completion routine is entered. If it is set the input buffer is backgrounded to all ones. Core location is set back to the original setting effectively repeating the request. If it is a cancel character the cancel and pass switches are set and exit is made to the dispatcher. If it is not a cancel character the pass switch is checked. If it is set exit is made to the dispatcher. If the pass switch is not set or the request is not formatted $FF is put into the upper half of the word containing the character just input. The upper lower switch is then checked. If set for upper the word is shifted left 8 bits and stored into the specified core location.

If set for lower the word is anded with the specified core location. If this is the last word, this was an unformatted single character input operation and exit is made to the common completion routine. If not, the upper lower switch is reversed. If now set for lower exit is made to the dispatcher. If now set for upper the core location is incremented. If it is now the last word exit is made to the dispatcher. If it is the last word and the input is unformatted exit is made to the common completion procedure. If the operation is formatted the pass switch is set and exit is made to the dispatcher.

45.4 1713 PAPER TAPE READER - S13002

45.4.1 Pseudo Entry Symbols

MS13I Relative distance between initiator and first location in S13002.

MS13RC Relative distance between continuator and first location in S13002.

MS13RE Relative distance between error routine and first location in S13002.

45.4.2 External Symbols

MAS400 Entry point in MASDRV. It will queue the reader if the 1713 is busy on entry to the initiator of the driver.

S13MOD Entry in the systems tables and parameters. Contains the current setting of the mode switch.

S13BZY Entry in the systems tables and parameters. Contains logical unit associated with the module currently busy with the 1713. It will be zero if the 1713 is not busy.

MAKEQ Routine that sets the proper error bits in Word 9 of PHSTAB to be passed to user.

ALTDEV Program that processes irrecoverable errors.

45.4.3 PHYSICAL DEVICE TABLE

The 1713 paper tape reader refers to the physical device table entries with the following names:

CALL	@ setting the get unit status.
STATUS	equipment status
SWITCH	switch settings listed below.
LENGTH	complement of remaining length of binary formatted record.
ADDRESS	address into which the next word read from tape is to be stored.
CORE	address into which the next data word is to be stored.
LASTP1	the last core location plus 1 to be stored into.
TEMP	temporary storage.
CHKSUM	checksum accumulator.
ERRTAB	request status word.
TIME	diagnostic clock locations.

'SWITCH' bits are used as follows:

bit 0:	0 for a read operation
1:	1 if formatted operation
2:	1 if lower character
3:	1 if ASCII mode
4:	0 if 1st word
5:	1 if not control information

45.4.4 DESCRIPTION

Initiator

S13RI is the first location of the initiator. At entry the 1713 busy flag (S13BZY) is checked for zero. If not zero, exit is made to MAS400. If zero, routine FNR is called. If no request is found the device is cleared and exit is made to MAS300.

If a request is found a check is made to determine if the device is in TTS mode. If not, TTS mode and interrupt on EOP are selected and exit is made to the dispatcher.

When the device is already in TTS mode, or TIS mode has been selected and an EOP interrupt has been generated by the switch of mode, the PHSTAB locations are initialized. Read mode and interrupt on data and alarm are selected, and a start motion function is output to cause an interrupt. Start motion must be selected for each frame. The diagnostic clock is set and exit is made to the dispatcher.

Continuator

S13RC is the first location of the continuator. If the interrupt was caused by an alarm condition, the error routine ▽SETC0D▽ is entered. If the interrupt was caused by data, the data transfer {▽DATINT▽} section is entered. If an E0P interrupt was generated by a switch of mode the initiator is re-entered.

Data Transfer 0peration

If entry was not because of an alarm interrupt and the request is formatted binary, the first character switch is checked. If equal to zero, the character just input is checked to see if the high order bit is a one. If not, exit is made to the dispatcher. This is because the first word of a binary formatted record is the complement of the number of words in the record. Tape frames are passed until a frame having a one bit in the high order is found. When one is found, the first character switch is reset. The character is then checked to see if it is equal to $AA {M with parity}. If it is, the mode of the request is changed to ASCII and the appropriate point in the code is entered. If it is not equal to $AA but is greater, 1 is subtracted from the character. This is to compensate for the fact that one is added by the punch driver in the equivalent situation. This is to prevent an actual length whose high order character is $AA from being misinterpreted as being an ASCII record. If first character switch was not set control comes to this point. If the control switch is not set and the record length has not been decremented to zero the address of ▽length▽ is put into ▽address▽. If the request was ASCII or was changed to ASCII, ▽C0RE▽ is checked to see if the requested number of words has been filled. If it has not, ▽core▽ is put into the ▽address▽. If the record length had been decremented to zero or the requested number of words had been filled ▽temp▽ is put into ▽address▽. This is to pass characters in ASCII or allow the checksum to come in binary.

If the request is ASCII, parity is checked. If incorrect, procedure ▽SETC0D▽ is entered. If the request is formatted, the character is checked to see if it is a carriage return. If it is and it is the upper character ▽C0RE▽ is incremented and the completion procedure is entered. If the character is a null {blank} line feed or cancel it is ignored and exit is made to the dispatcher. 0ne bits are put in the upper 8 bits of the word containing the character. If it is the lower character, this word is added with the contents of ▽address▽ and stored back into the same location. This word is then added to the checksum. If the request is formatted binary the record length is checked to see if it has been decremented to zero. If it has, the checksum is checked to see if it is zero. If not, procedure ▽SETC0D▽ is

entered. If it is zero the completion procedure is entered. If the record length has not been decremented to zero the control switch is checked. If it is set, ▽CORE▽ is checked to see if the requested number of words has been filled, if not ▽CORE▽ is incremented. Record length is then decremented. The upper-lower character switch is reversed and the record length is checked to see if it has been decremented to zero. If it has the control switch is reset {to indicate that checksum will come in next}. If it has not the control switch is set. In both cases exit is made back to the dispatcher. If the upper lower switch had been set for upper, the word containing the character is shifted left eight bits and control is transferred below to see if this is the last location. If the request was not formatted binary, upper lower switch is reversed and exit is made to dispatcher. If not the core location is incremented. If it is now last and the request is not formatted exit is made to the completion routine. If it is formatted or the core location was not last, the upper lower switch is checked. If not set for upper, exit is made to the common completion routine. If it is set for upper the core location is incremented and exit is made to the completion routine.

45.5 1713 PAPER TAPE PUNCH - S13003

45.5.1 Pseudo Entry Symbols

MS13PI — Relative distance between initiator and first location in S13003.

MS13PC — Relative distance between continuator and first location in S13003.

MS13PE — Relative distance between diagnostic timer error routine and first location in S13003.

45.5.2 External Symbols

MAS400 — Entry point in MASDRV. It will queue the reader if the 1713 is busy on entry to the initiator of the driver.

S13MOD — Entry in the systems tables and parameters. Contains the current setting of the mode switch.

S13BZY	Entry in the systems tables and parameters. Contains logical unit associated with the module currently busy with the 1713. It will be zero if the 1713 is not busy.
MAKE@	Routine that sets the proper error bits in Word 9 of PHSTAB to be passed to user.
ALTDEV	Program that processes irrecoverable errors.

45.5.3 PHYSICAL DEVICE TABLES

The 1713 paper tape punch refers to the physical device table entries with the following names:

CORE	current core location to be punched from
LASTP1	last location plus 1
SWITCH	switch settings listed below
CHECKSUM	checksum accumulator
CALL	@ setting to get unit status
STATUS	machine status of the punch
ERRTAB	request status word
TEMP	temporary storage
TIME	diagnostic clock location

°SWITCH° bits are used as follows:

bit 0:	1 for write operation
1:	1 if formatted operation
2:	1 if lower character
3:	1 if ASCII mode
4:	1 if not 1st word
5:	1 if line feed to be sent {control switch}
6&7:	count of spacer {cancel} characters to be sent
8:	1 if the operation is complete but the last interrupt has not yet come in.

45.5.4 DESCRIPTION:

Initiator

S13PI is the first location of the initiator. At entry the 1713 busy flag {S13BZY} is checked for zero. If not zero, exit is made to MAS400. If zero, routine FNR is called. If no request is found the device is cleared & exit is made to MAS300.

If a request is found a check is made to determine if the device is in TTR mode. If not, TTR mode and interrupt on EOP are selected and exit is made to the dispatcher.

When the device is already in TTR mode, or TTR mode has been selected and an EOP interrupt has been generated by the switch of mode, then write mode and interrupt on data and alarm are selected and the continuator is entered to output a character.

Continuator

S13PC is the first location of the continuator. If the interrupt was caused by an alarm condition, the error routine ▽SETCOD▽ is entered. If the interrupt was caused by data, the data transfer section is entered. If an EOP interrupt was generated by a switch of mode the initiator is re-entered.

Data Transfer

If the entry was caused by a data interrupt, the control switch is checked. If set, a line feed character is output and routine GODISP entered. If not the completion interrupts are cleared, TIME is set negative, the completion procedures are executed and the initiator re-entered.

Routine GODISP will set time to the length of time before which the next interrupt must be received. If the completion bit is not set, the complement of the length is computed and placed in Q. If the high order character of this negative length is equal to or greater than $AA {*with parity} the length is increased by $100. If the operation is not formatted, control is sent to that part of the code which gets data from the specified core location. If the operation is formatted ASCII bits 6 & 7 of SWITCH are checked to see if any waste time characters are to be sent. If so, the count is decremented and a cancel character is sent out. Exit is then made to GODISP. If no spacers are to be sent, the length is checked to see if all data has already been sent out. If so, carriage return line feed is put into A and control is passed to the code just after the point at which data is obtained from core. If not, control is transferred to the point which obtains data from core.

If the operation if formatted binary the length is checked to see if the last word has been sent. If so, the complement of the checksum is placed in Q and the point just after data is obtained from core is entered. If the last word has not been sent, SWITCH is checked to see if the first word has been sent. If so, data is not obtained from core so that the complement of the length is punched. If not, data is obtained from the location in ▽CORE▽. It is placed in Q. SWITCH is checked to see if the upper character is to be punched. If so, Q is shifted right 8 bits. If not the contents of Q is added to ▽CHKSUM▽. The high order 9 bits

of Q are cleared. If formatted AACII and the character is a carriage return the control bit is set in SWITCH. If formatted ASCII and the character is TAB, VTAB, or FORM, a spacer count of 3 is set into SWITCH. If the request is ASCII parity is computed and inserted into bit 7. The character is then punched. The length is then computed and placed in Q. The upper lower switch is reversed. If now set for lower, 1st word switch is set and the remaining length shows the operation is complete this was a single character output operation. The completion bit is set and exit is made to EXIT. Otherwise, the remaining length is checked to see if the last word has been sent. If so the completion bit is set. If not, and the operation is formatted binary and the first word switch is set exit is made to GODISP after resetting the 1st word switch. If not, the core location is incremented. If it now says the operation is complete, the complete bit is set and in either case exit is made to GODISP.

45.6 CAPABILITIES

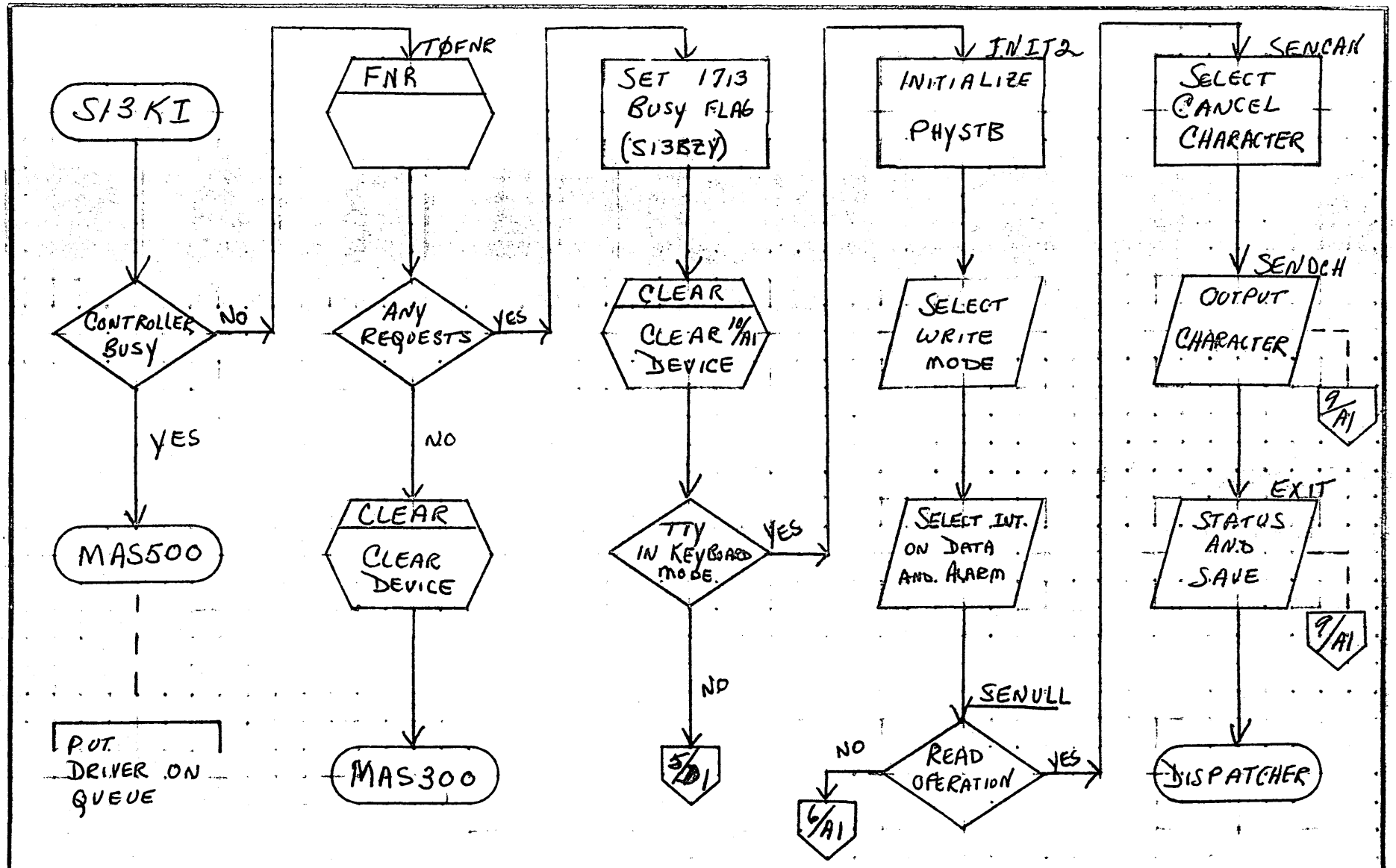
The capabilities of this driver are identical to the capabilities of the following drivers:

1. 1711/1712/1713 Teletypewriter Driver.

2. 1721/1722 Paper Tape Reader Driver.

3. 1723/1724 Paper Tape Punch Driver.

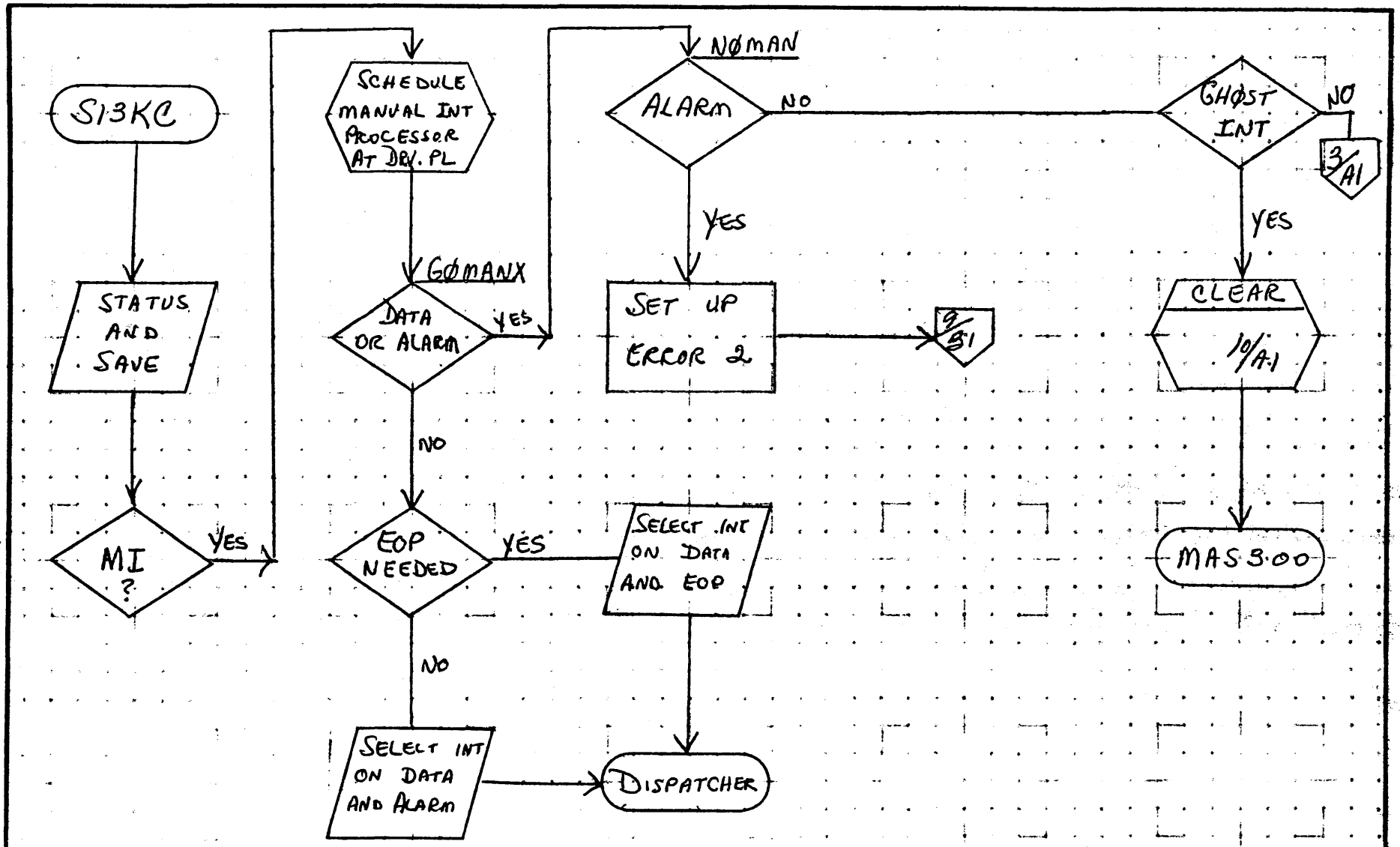
All requests honored by the above drivers are honored by the 1713 Driver and vice versa. The results in all cases are the same.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1713 KEYBOARD		PROJECT MGR			
FLOWCHART ☐	S13001	PAGE 1 OF 10	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

45.15



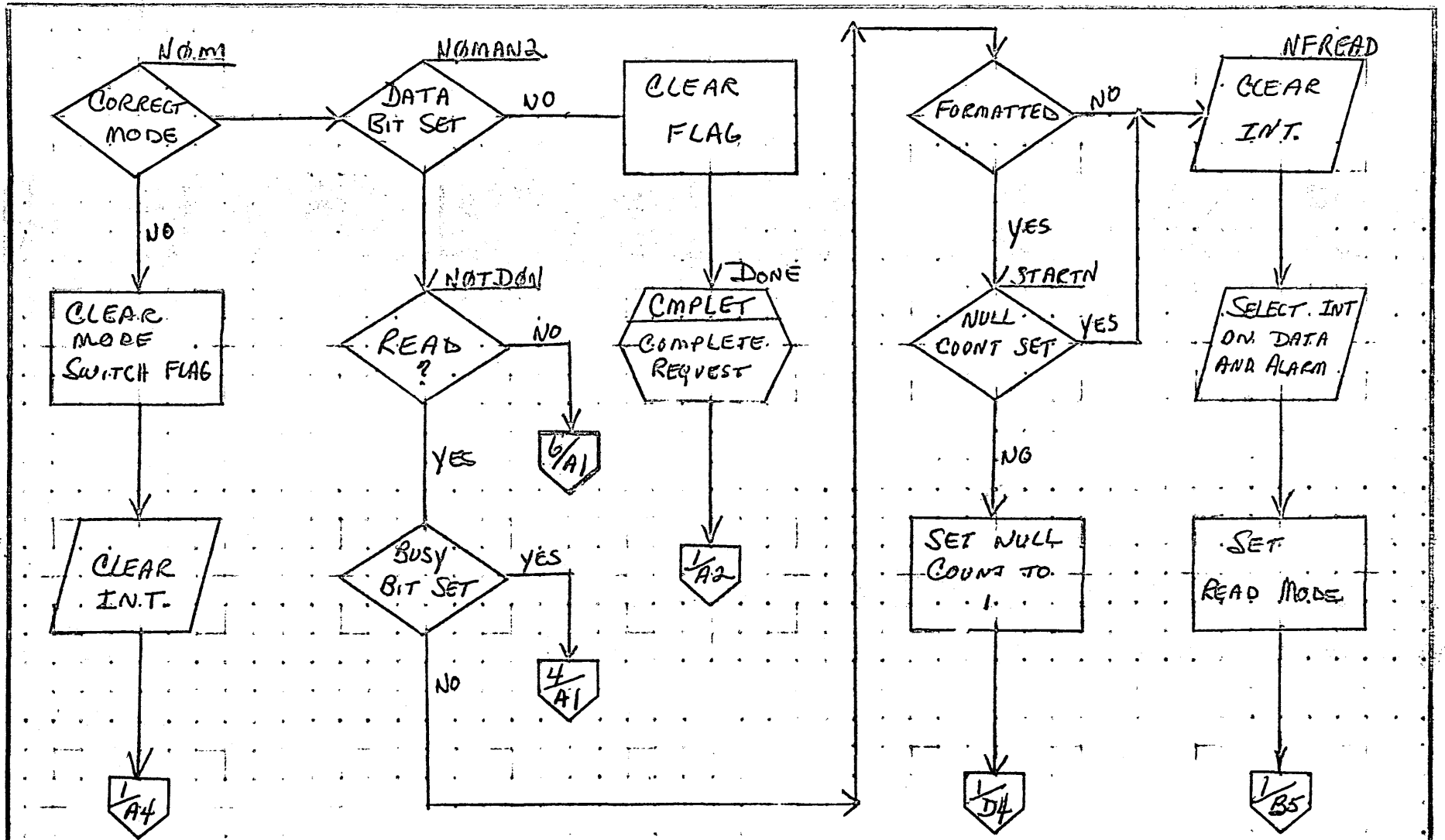
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 KEYBOARD	PROJECT MGR.			
	S13001 PAGE 2 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

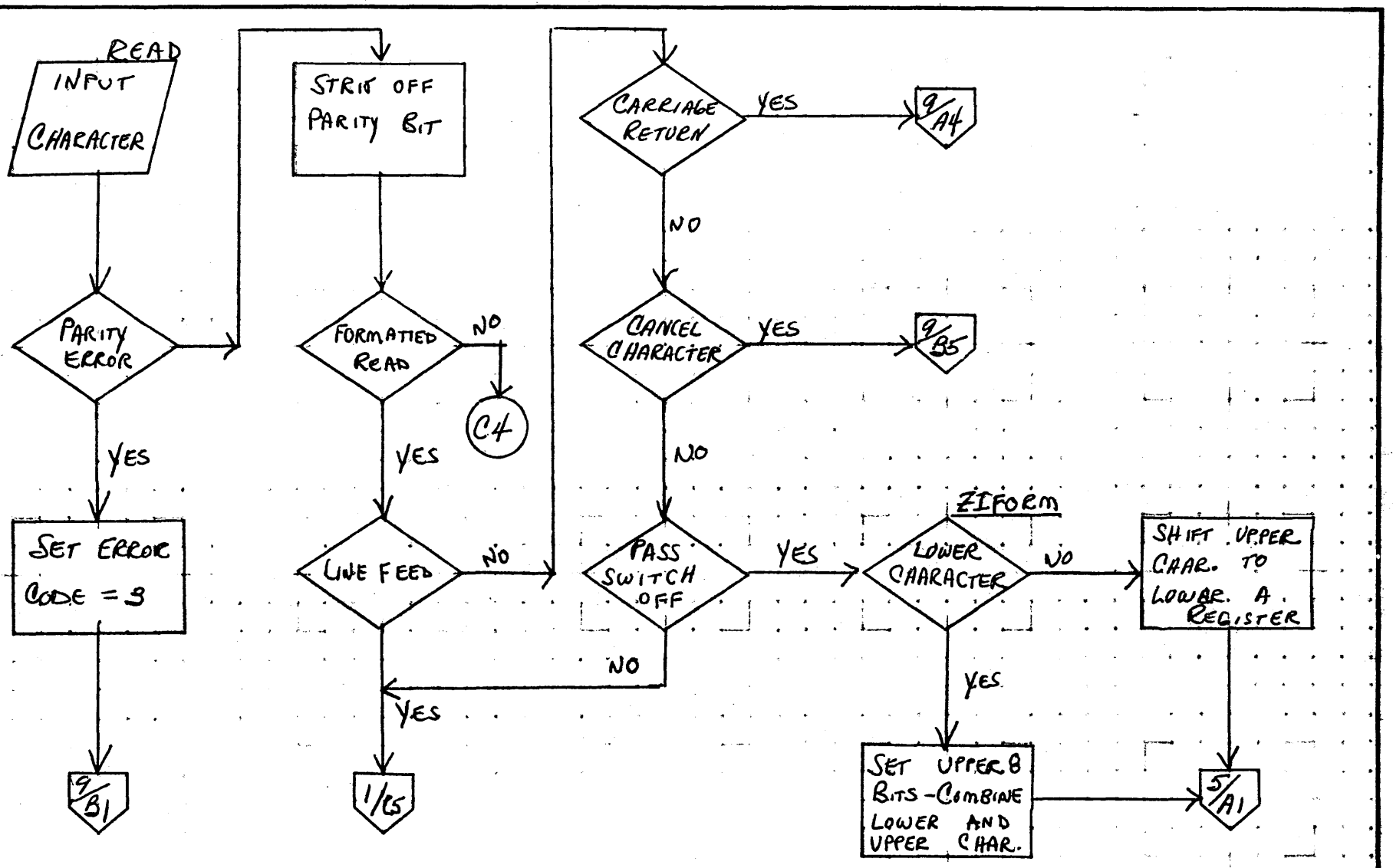
45.16

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 KEYBOARD	PROJECT MGR.			
	S13001 PAGE 9 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



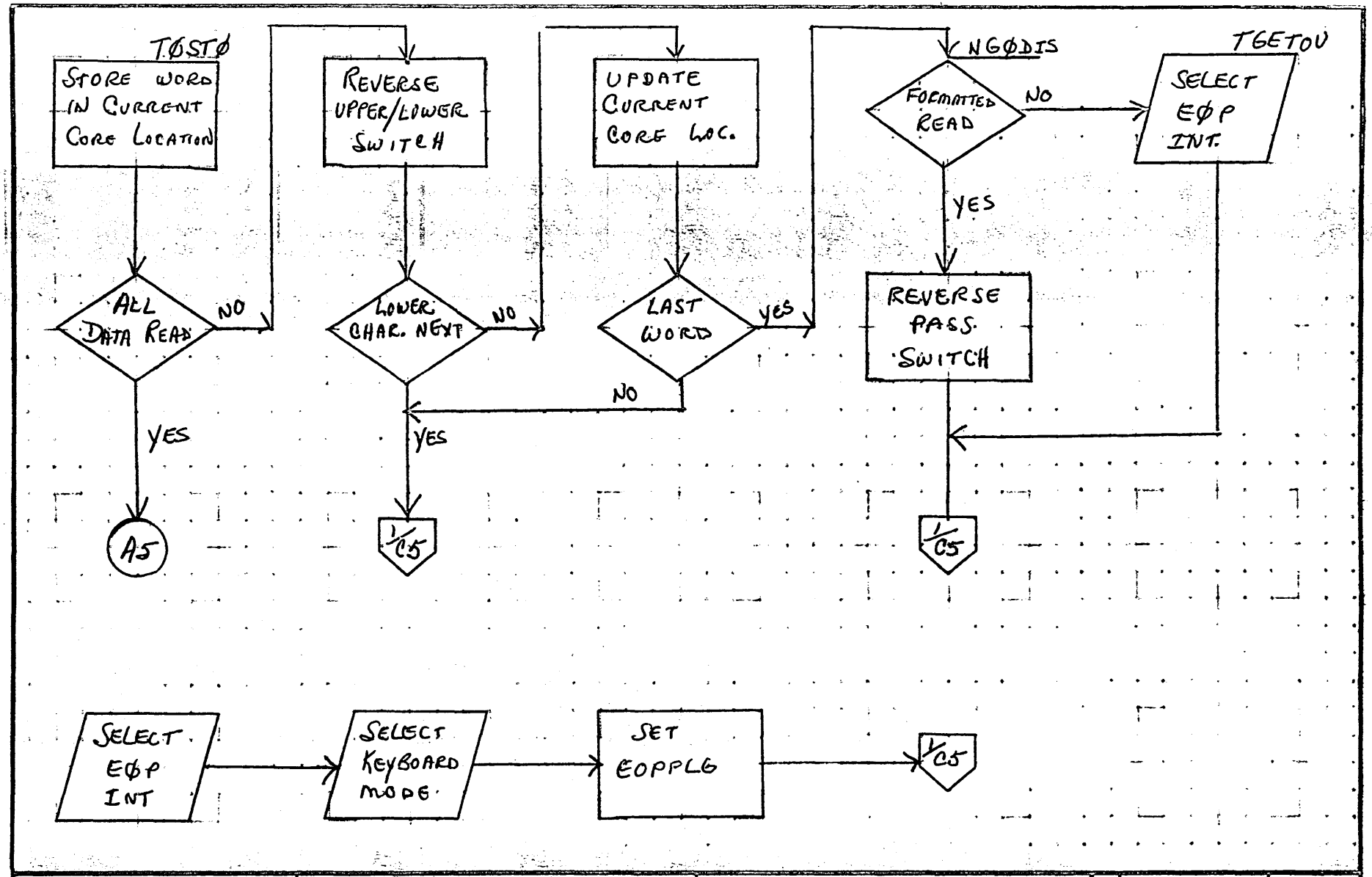
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1713 KEYBOARD		PROJECT MGR.		
	S13001	PAGE 4 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

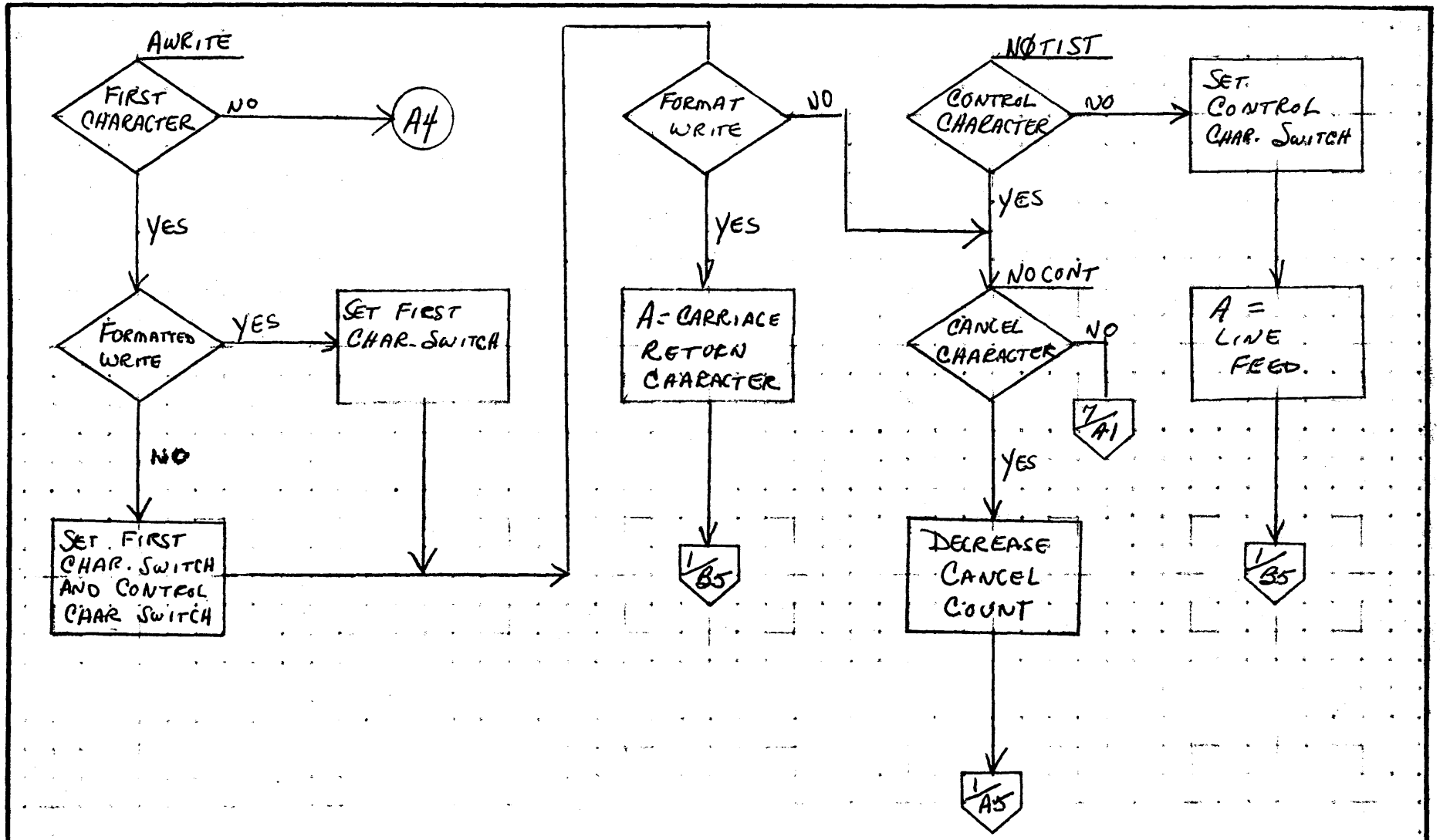
45.18



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: 1713 KEYBOARD		PROJECT MGR.		
	513001	PAGE 5 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

45.19



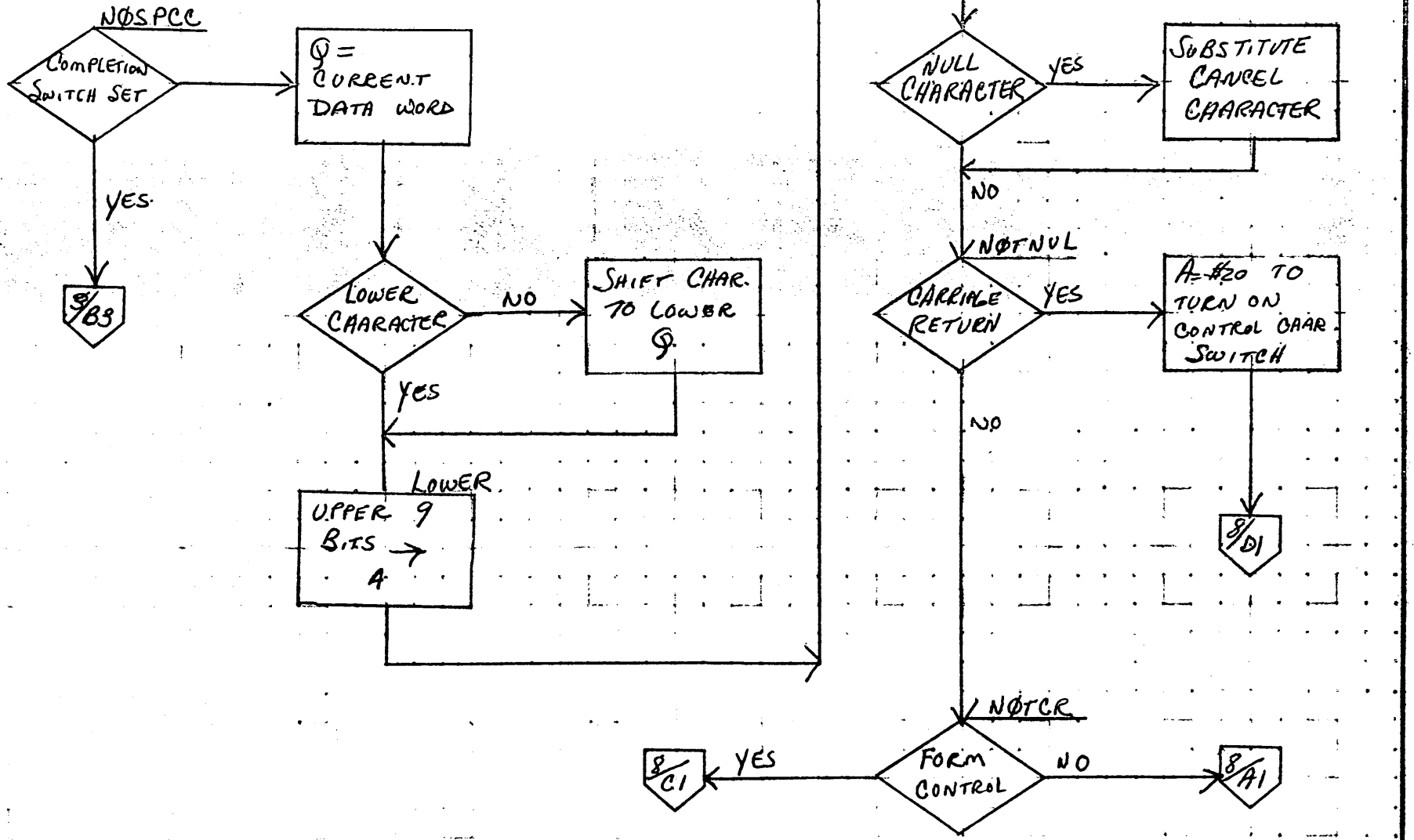
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 KEYBOARD	PROJECT MGR.			
	513001 PAGE 6 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

45.20

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



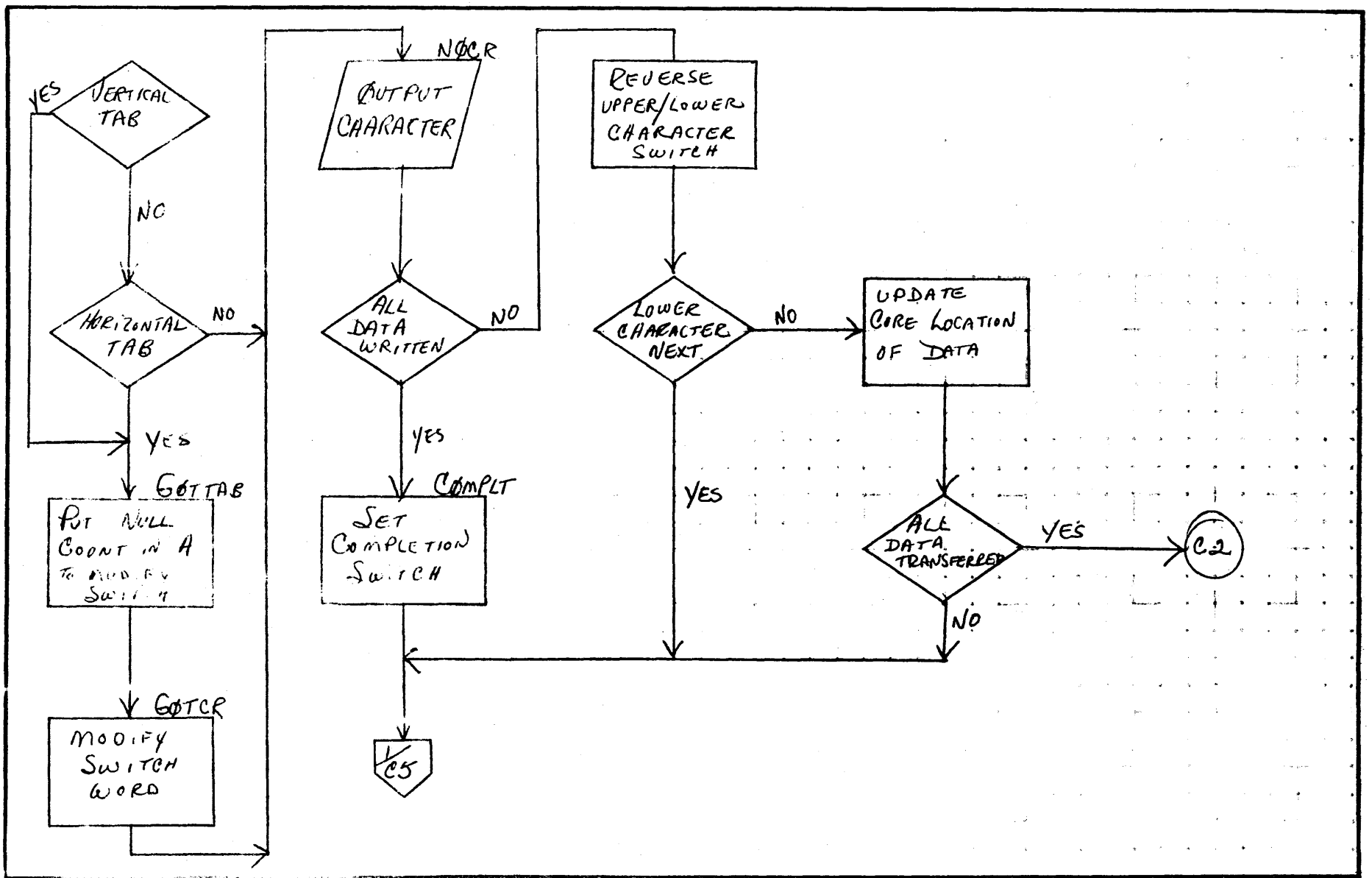
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1713 KEYBOARD		PROJECT MGR.		
	813001	PAGE 7 OF 10	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 FORMERLY CA127

PRINTED IN U.S.A.

MAR 5 1971

45.21



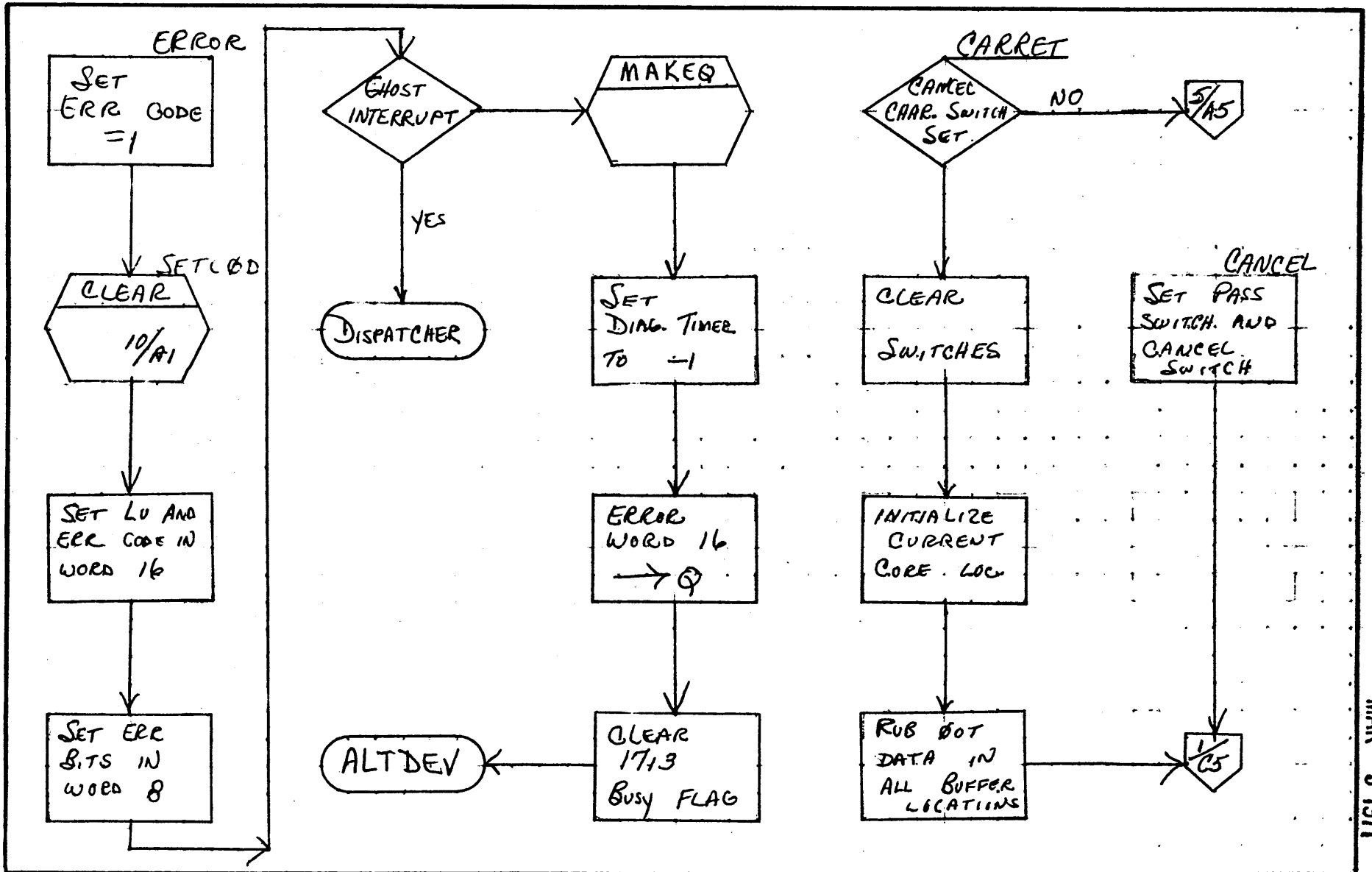
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 KEYBOARD	PROJECT MGR.			
	S1200, PAGE 8 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

45.22

AA 385 FORMER CA127

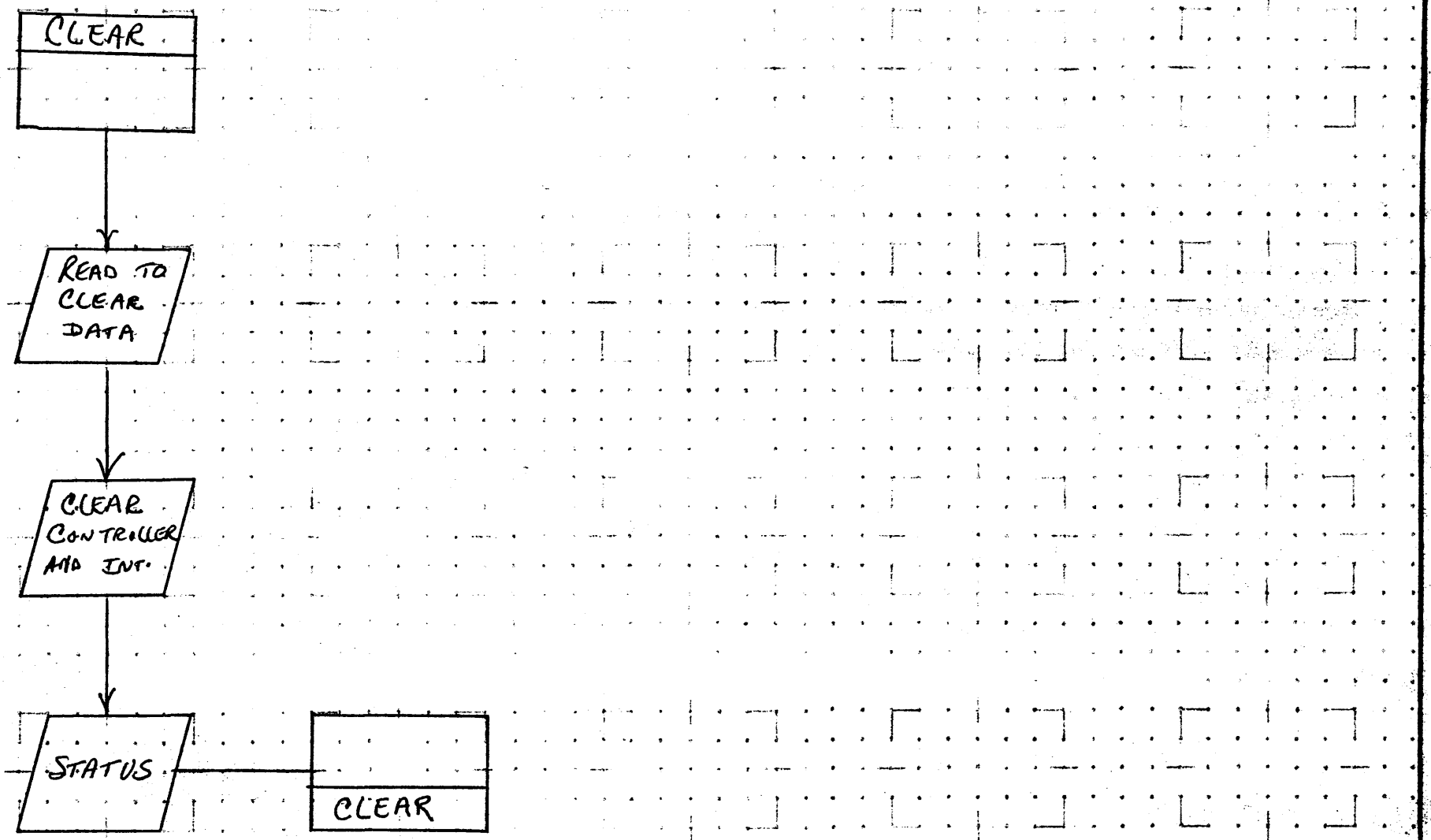
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IAS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 KEYBOARD	PROJECT MGR			
	S13001 PAGE 9 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

45.23



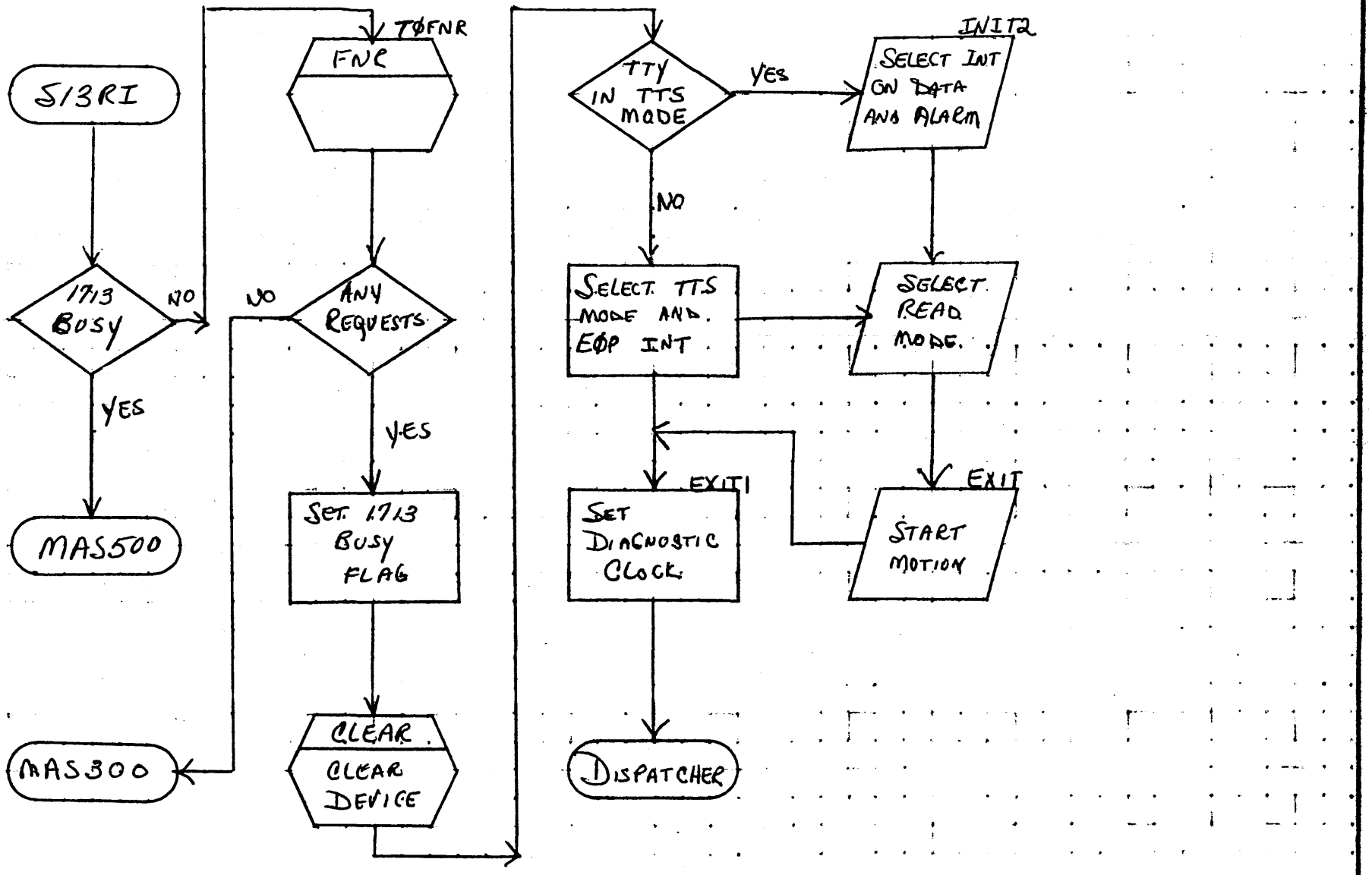
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 KEY BOARD	PROJECT MGR.			
	S13001 PAGE 10 OF 10	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

45.24



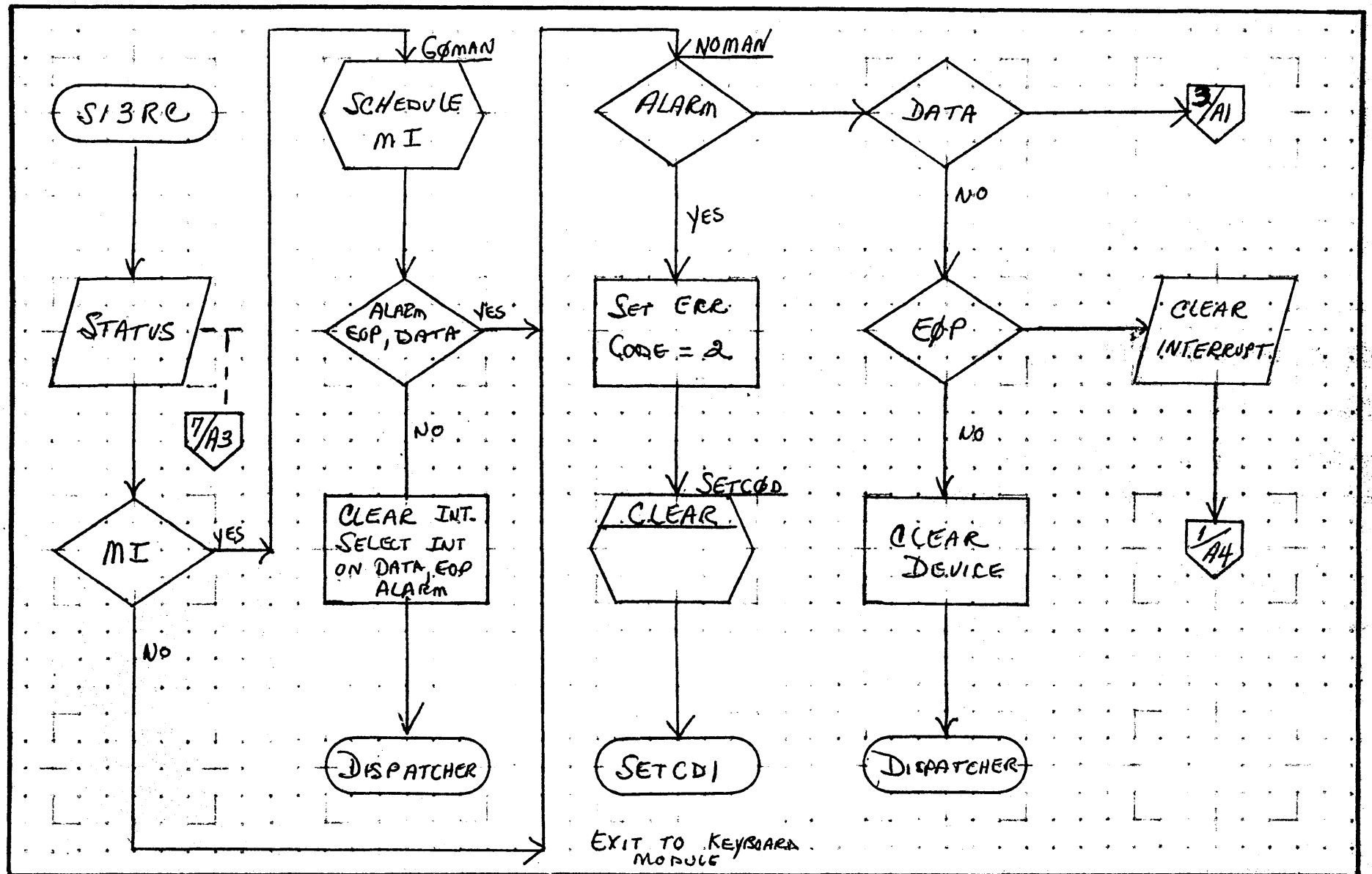
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 READER	PROJECT MGR.			
	513002 PAGE 1 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

45.25

AA 1385 (FORMERLY CA127)

PRINTED IN U.S.A.



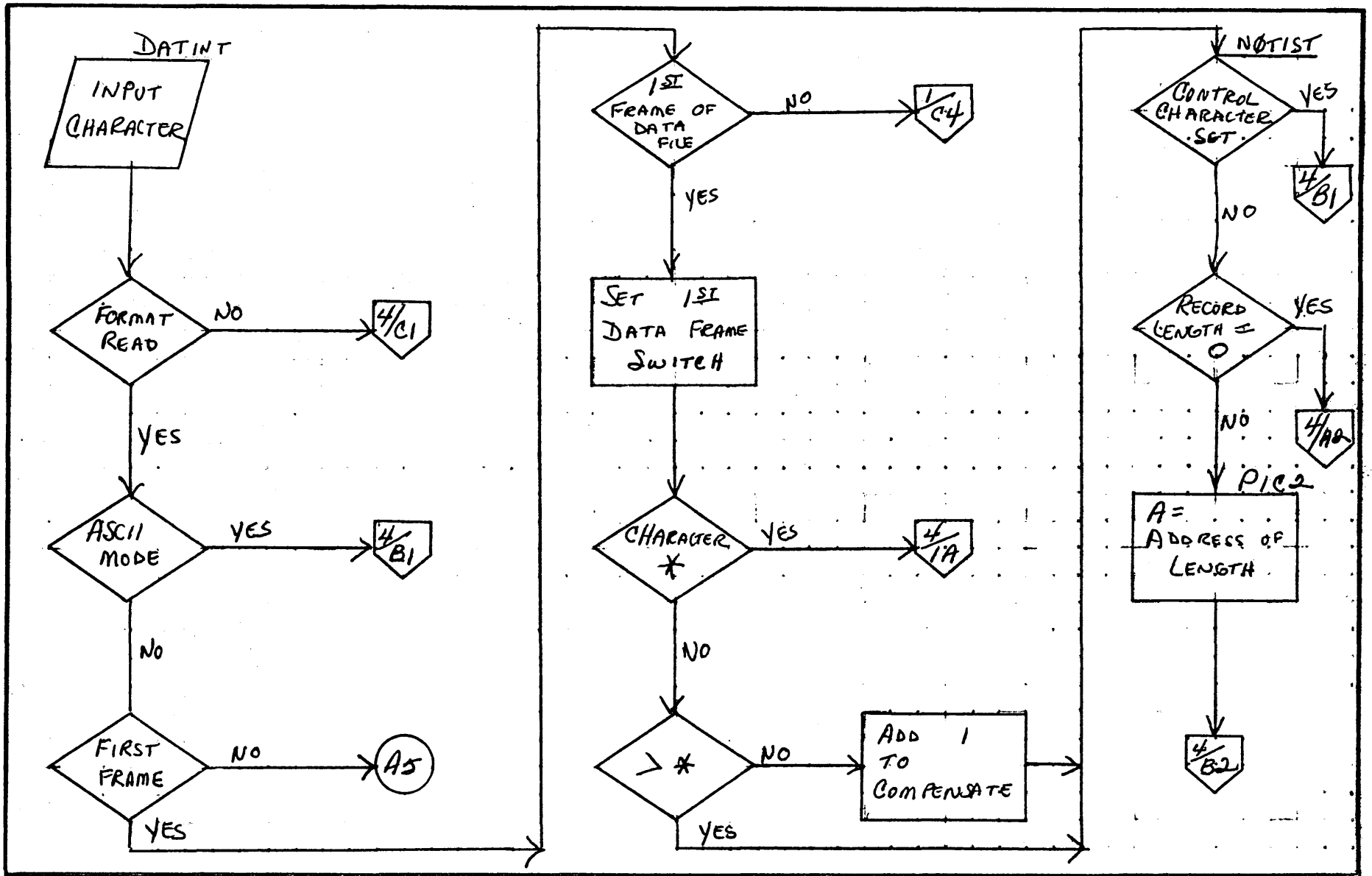
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1713 READER		PROJECT MGR.			
FLOWCHART ☐	513002	PAGE 2 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

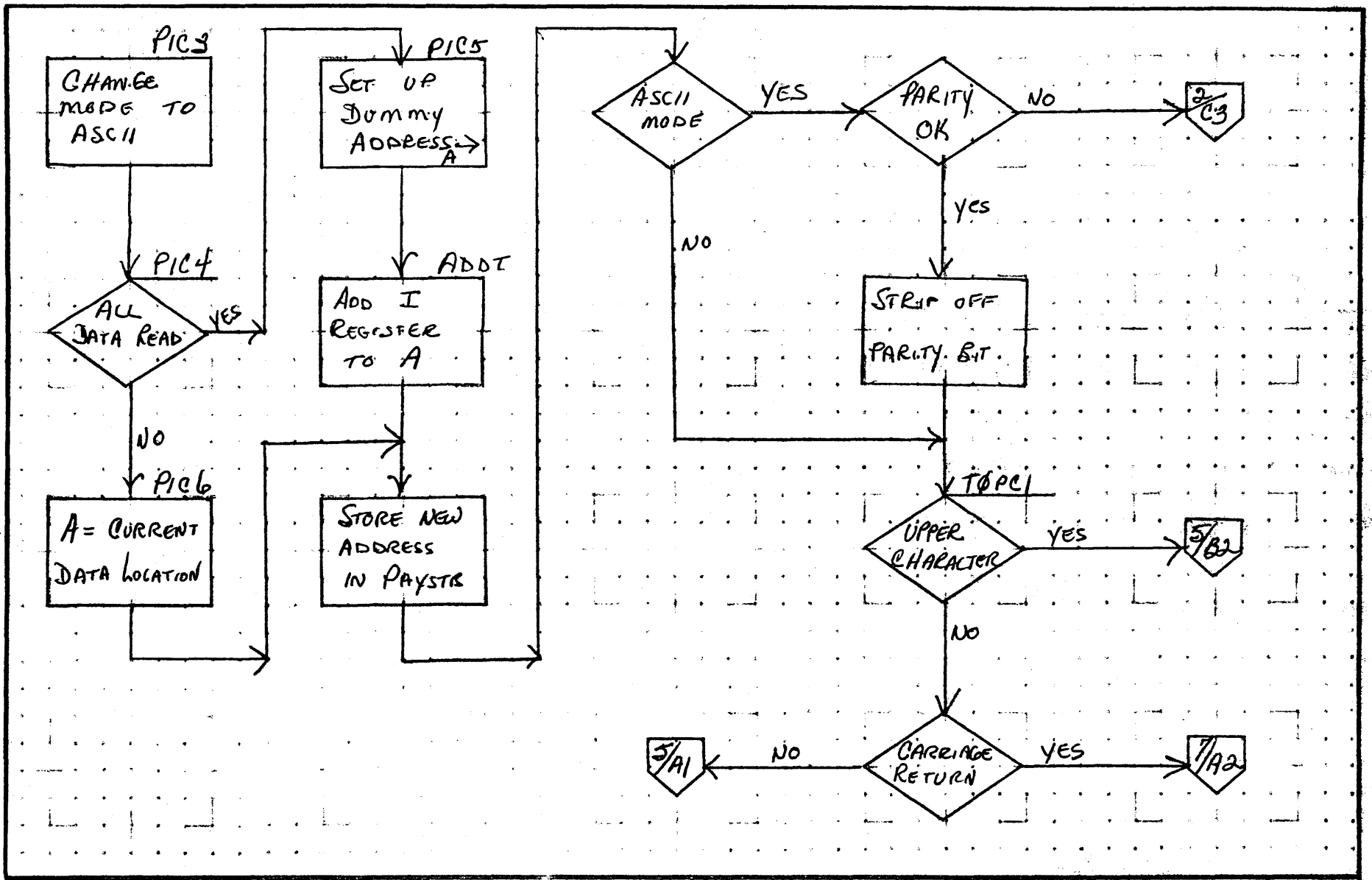
45-29



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS INS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1713 READER		PROJECT MGR.		
	S13003	PAGE 3 OF 7	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

45.27



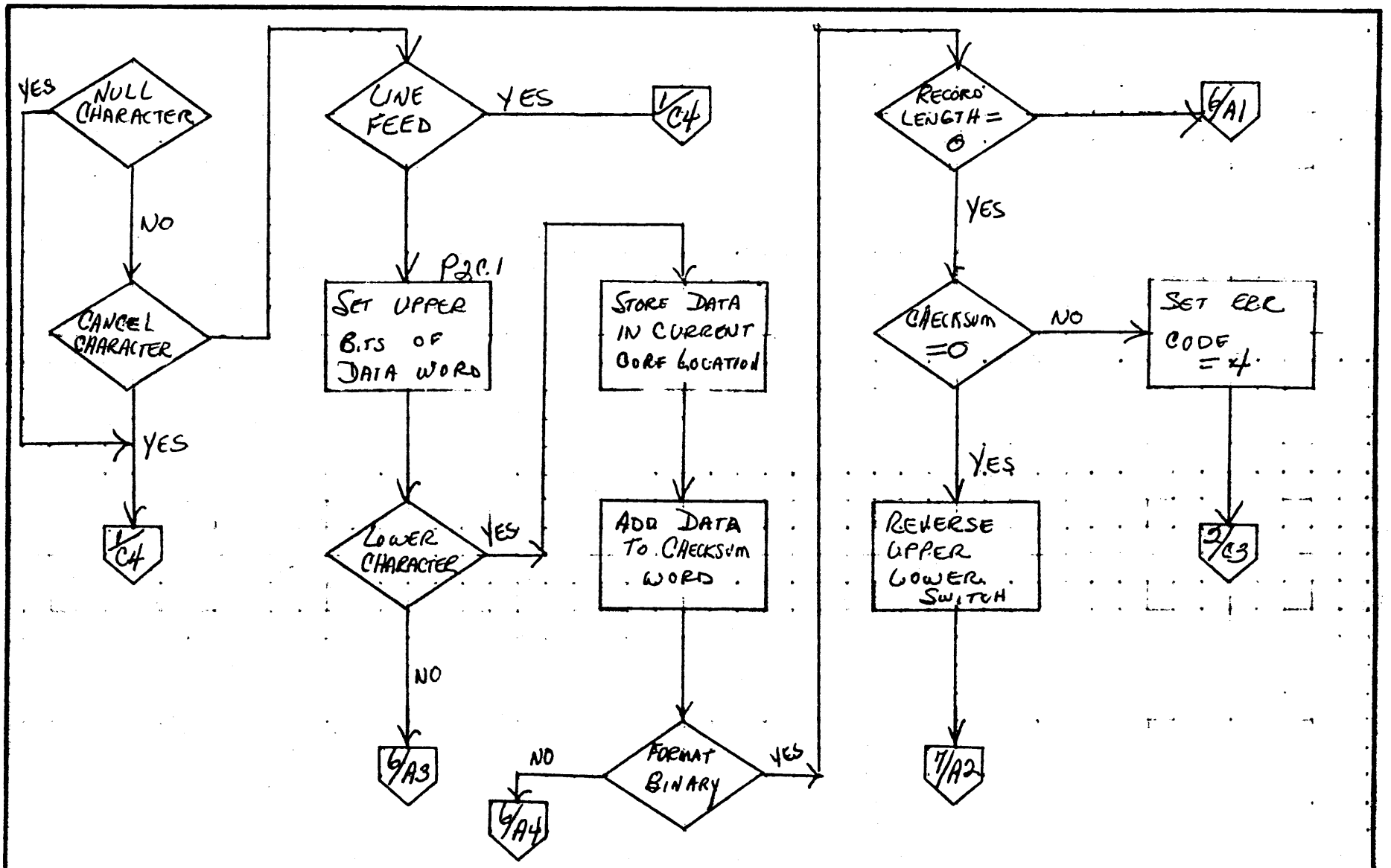
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1713 READER		PROJECT MGR.		
	SI3002	PAGE 4 OF 7	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

45.28

AA1385 (FORMERLY CA127-1)

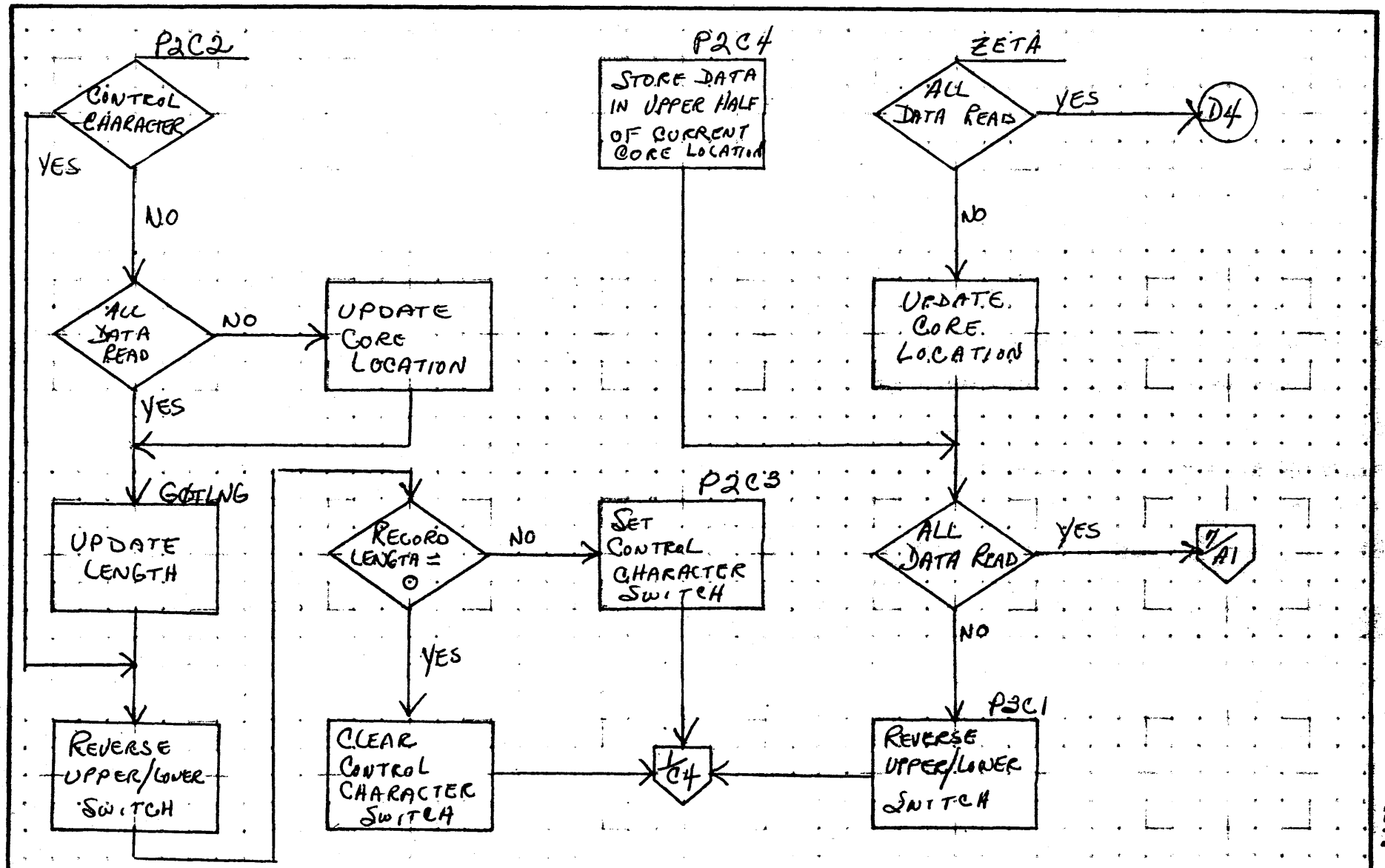
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1713 READER		PROJECT MGR.			
FLOWCHART ☐	513002	PAGE 5 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

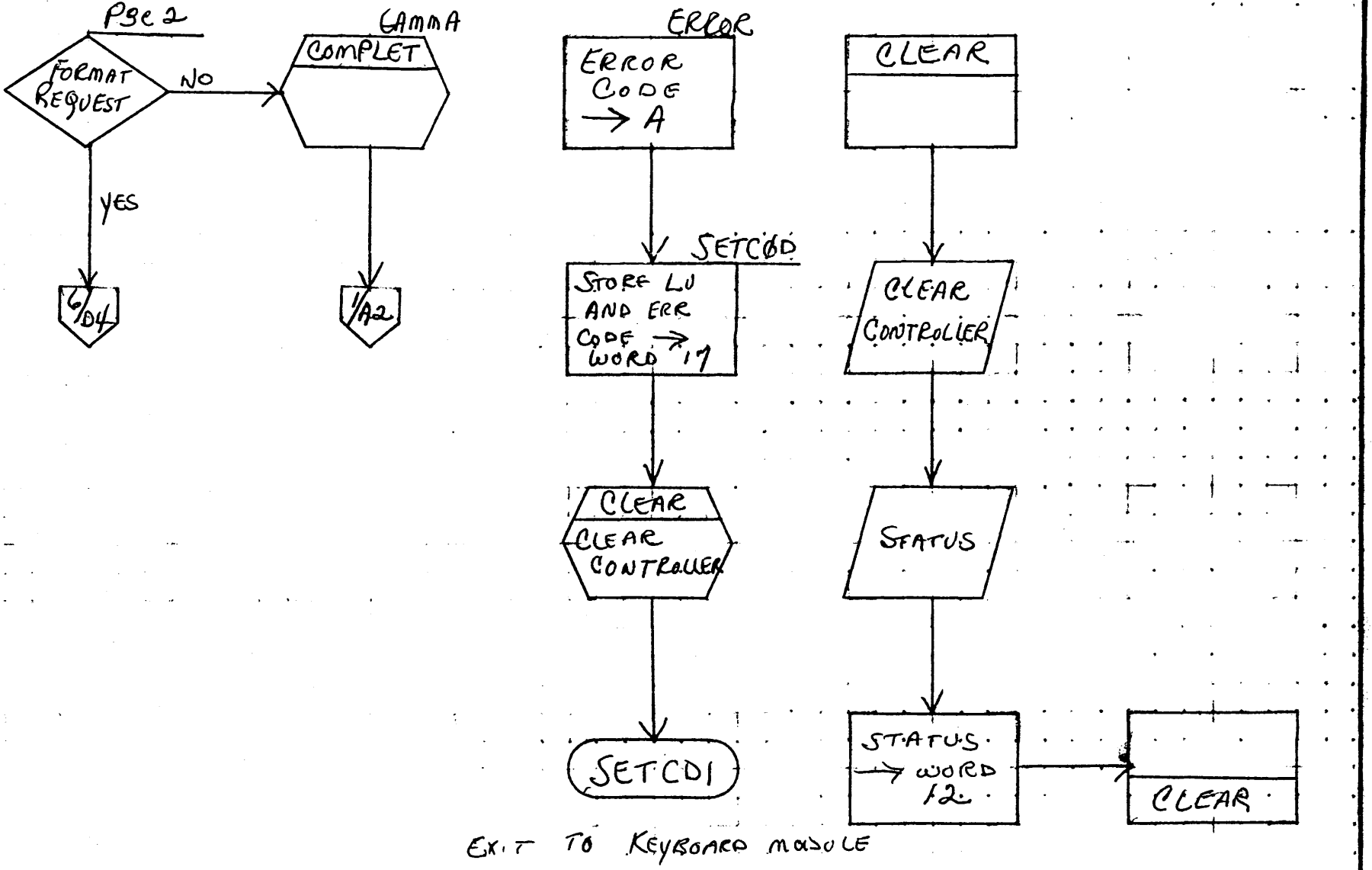
45.29



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1713 READER		PROJECT MGR.			
FLOWCHART ☐	51300 2	PAGE 6 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

45.30



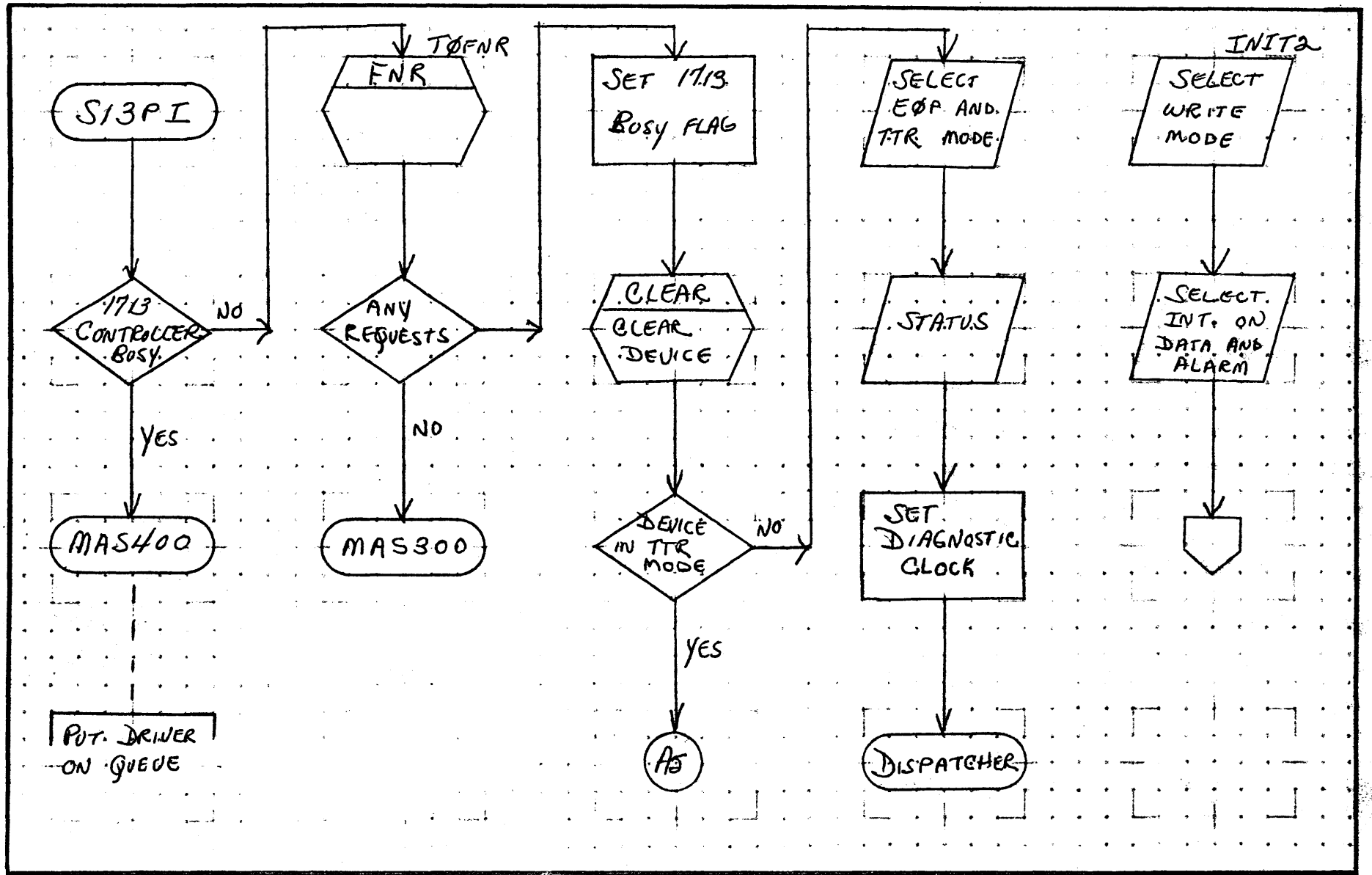
CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE 1713 READER	PROJECT MGR	
	S13002 PAGE 7 OF 7	PROJECT NAME	
	NUMBER ISSUE DATE	TASK NO.	
	DRAWN BY DATE	TASK NAME	

MAR 5 1971

45.31

AA1385 (FORMER · CA127

PRINTED IN U.S.A.



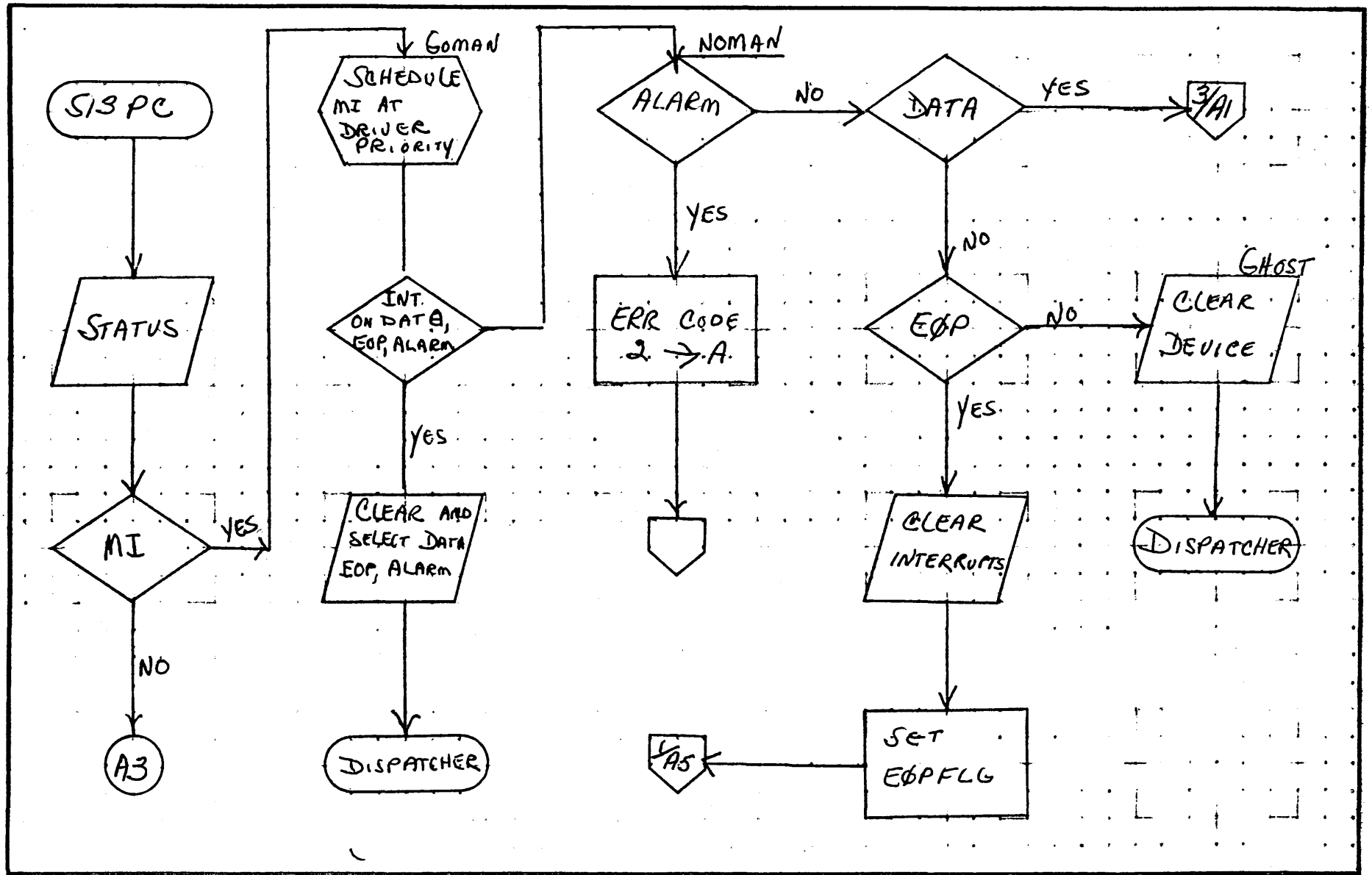
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: 1713 PUNCH	PROJECT MGR.			
	S13003 PAGE 1 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

45.32

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



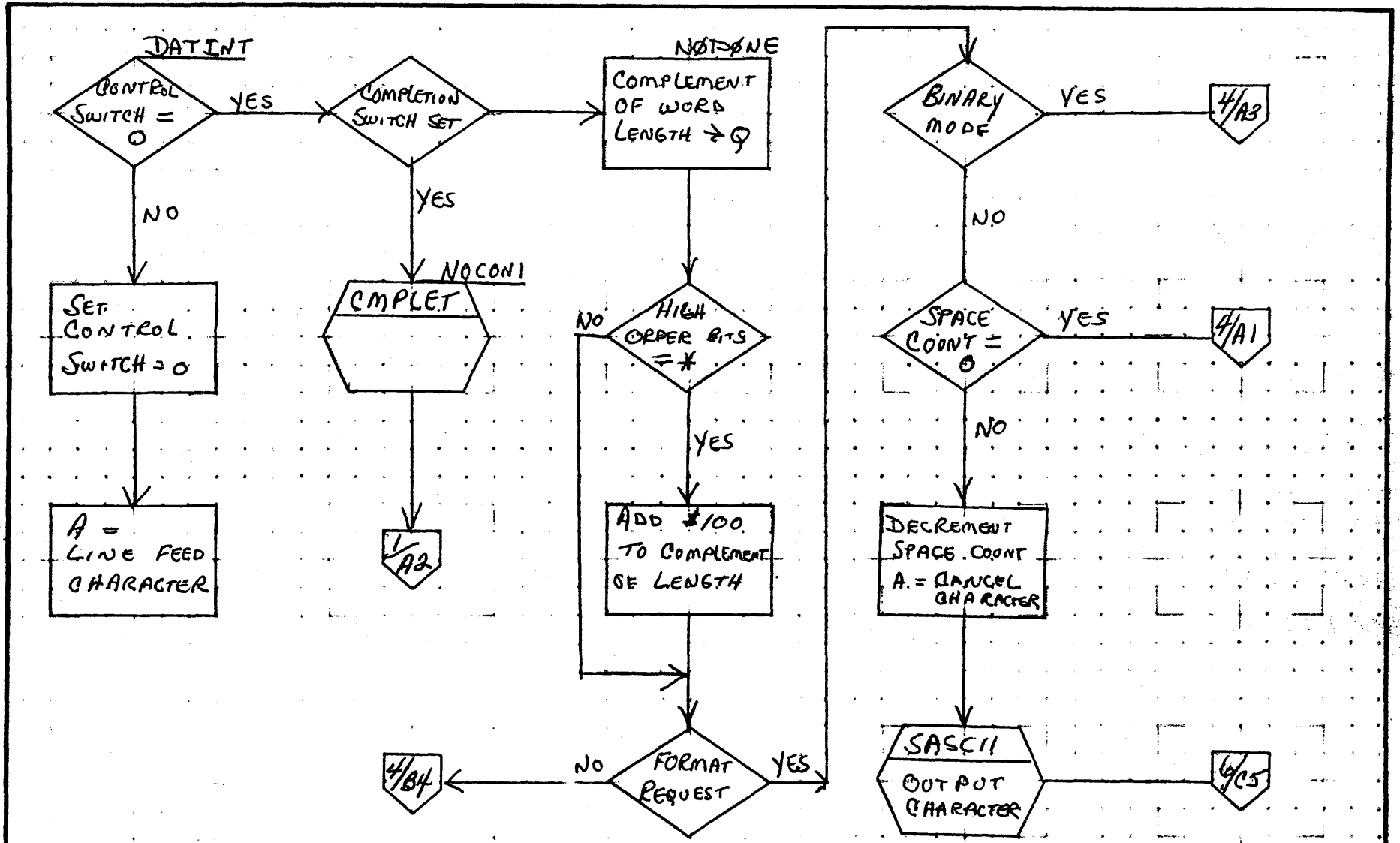
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 PUNCH	PROJECT MGR.			
	S13003 PAGE 2 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

45-33

AA 385 (FORMERLY CA 27)

PRINTED IN U.S.A.



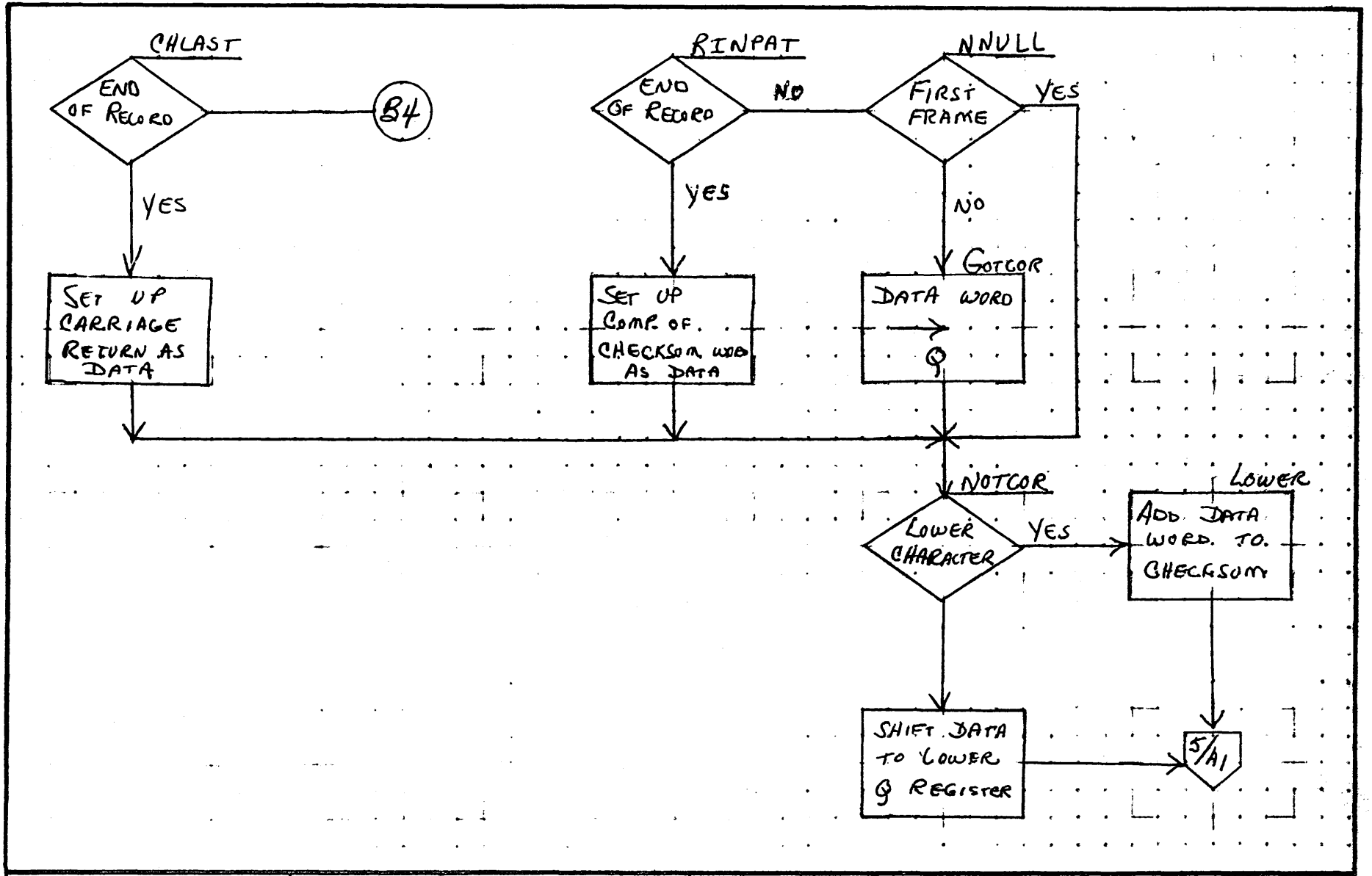
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1713 PUNCH		PROJECT MGR.			
FLOWCHART ☐	51300 3	PAGE 3 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

45.34

AA1385 (FORMERLY CA127-1)

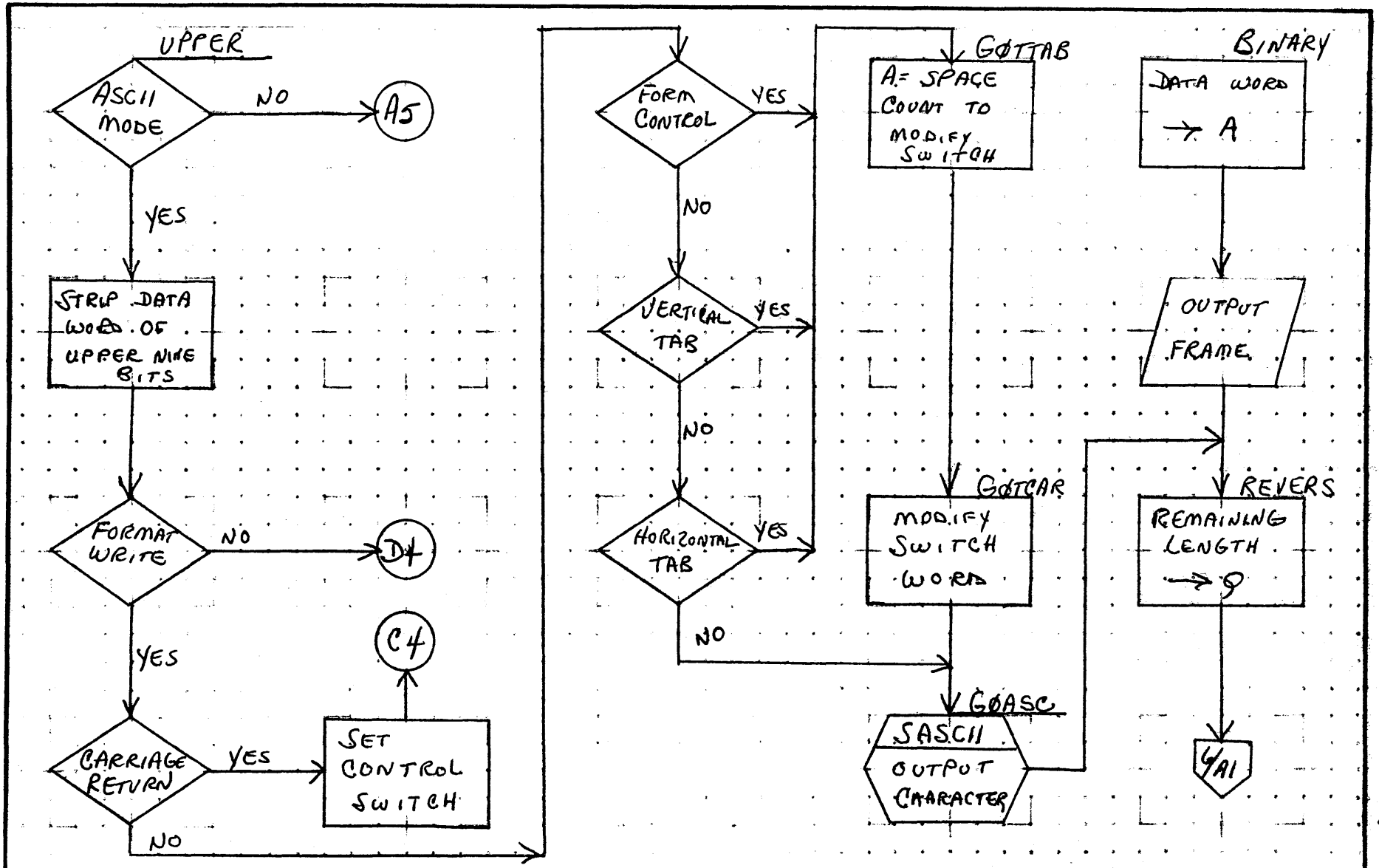
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 PUNCH	PROJECT MGR			
	513003 PAGE 4 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

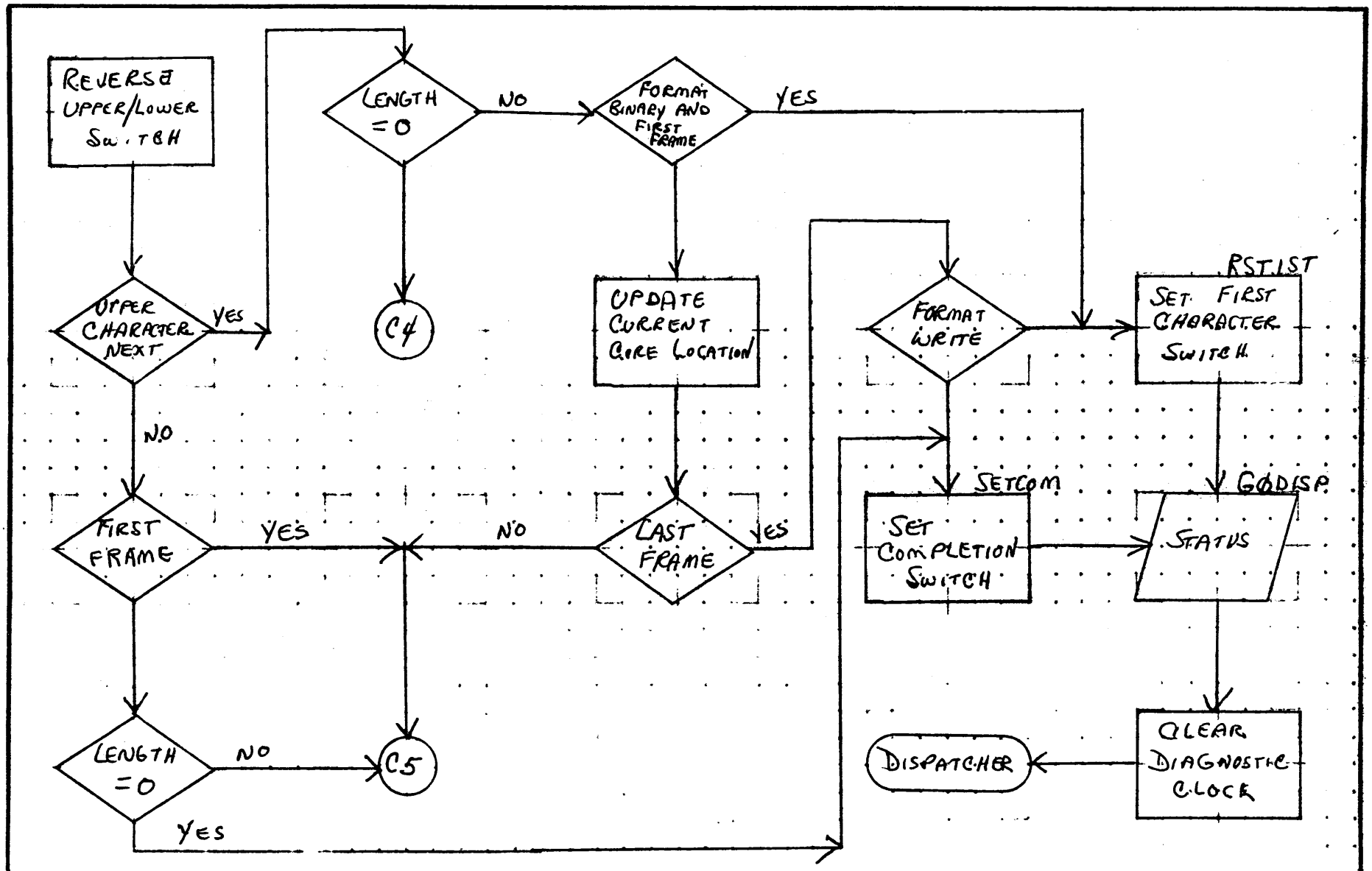
45.35



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1713 PUNCH		PROJECT MGR.			
FLOWCHART ☐	513003	PAGE 5 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

45-36



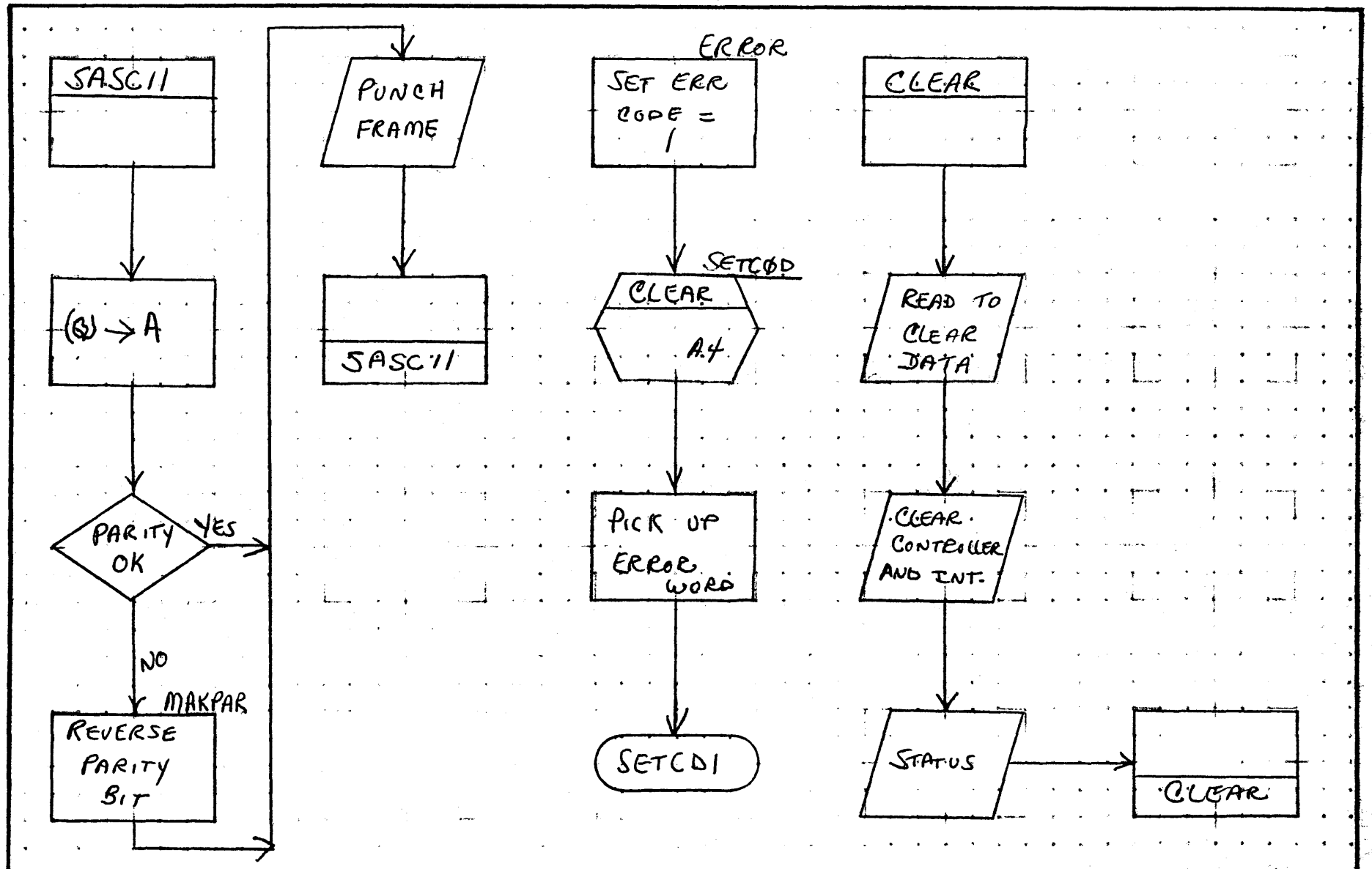
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS IMS MACH TYPE 1700 DOCUMENT TITLE 1713 PUNCH S13003 PAGE 6 OF 7 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO PROJECT MGR PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

MAR 5 1971

45.37

AA 385 (FORMER) A127

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1713 PUNCH	PROJECT MGR.			
	513003 PAGE 7 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

45.38

46.0 DISK DRIVER WITH WORD ADDRESSABILITY (DISKWD)

This version of the disk driver includes the following features:

1. The driver allows word addressing of all mass storage (simulating a 1751 drum capability) in addition to sector addressing.

2. The driver compares data on disk to core after a read or write operation and repeats the operation if the data does not compare.

3. The driver identifies errors (hardware malfunctions) and records all occurrences in an error histroy table.

4. The driver repeats all requests up to ten times before calling the DKDIAG program (to print out the information).

46.1 ENTRY POINTS

DKINTR	initiator entry
DKCONT	continuator entry
DKDIAR	error entry

46.2 EXTERNALS

LOG	Engineering File
DKDIAG	Disk diagnostic printout and recovery routine

46.3 ENTRY INTERFACES

Q = address of the Physical Device Table

46.4 EXIT INTERFACES

None

46.5 GENERAL PROGRAM INFORMATION

46.5.1 INTERNAL SYMBOLS

CLCKVA — Number of seconds after which an I/O hangup is assumed.

COMPAR — An internal flag to indicate this is a compare data operation and not a read/write.

TOTERR — A history table of error occurrences.

Error Code	Meaning
0	Clock run out {never printed out}
1	Ghost interrupt {never printed out}
2	Alarm printed if alarm status bit is set
3	On cylinder bit not set after a seek operation
4	Checkword error
5	Internal reject
6	External reject
7	Load file address register command not causing file address register to set to correct value. Disk read/write head is not positioned correctly.
8	Incorrect status. {Look at the current status word for more information on the incorrect status.}
9	Compare error after a read or write.

Other internal symbols are defined in the PHYSTB description.

PHYSICAL EQUIPMENT TABLE

46.5.2

PHYSTB	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	LABEL
+0	0	Request Code for Schedule Request							RP				CP				PHYSTB
+1	Entry Point of Initiator																
+2	Entry Point of Continuator																
+3	Entry Point of Diagnostic Routine																
+4	Diagnostic Clock																CLCK
+5	Device Logical Unit Number																LUN
+6	Address of Current Parameter List																
+7	0	0	0	0	0	Device Equip. No.			0	0	0	0	0	0		1	GENST
+8	E	S				T							R			S	SYST
+9	V					T	R						S		F	W	RQST
+10	First Word Address of Data																FWD
+11	Last Word Address of Data +1																LWD
+12	Dynamic Hardware Status																STS
+13	N												J				ERCONT
+14	Disk Data Transfer Function																FDATAF
+15	Disk File Address																FILEAD
+16	NC	Disk Unit Control Function															FCONN
+17	H	Address of PHYSTB for other Unit															OTHER
+18	Priority of Overlay Request																ECALL1
+19	Absolutized Completion Address																ECALL2
+20	Thread																ECALL3
+21	Logical Unit From Request																ECALL4

Words 0 to 3, 5 to 8, and Parameter N must be preset before entering initiator.

Word 8
- E=1 Operation is in Progress
- E=0 Operation complete
- S Equipment Class
- T Equipment Type Constant
- R Bit 1=1 Device may be read from unprotected programs
  Bit 2=1 Device may be written from unprotected programs
  Bit 3=1 Equipment table includes words 18-33 for message buffering.

Word 9
- V Error Indicator = 0 in Driver
- T=1 Temporary Parity Error
- R=1 Mass Storage Device
- S=1 Seek in Progress

Word 13
- N Unit Number Relative to Controller
- J Recovery Error Counter

Word 16
- NC=1 Do not compare after Read or Write Operations.

Word 17
- H=1 Other Entry has Request Waiting

Word 18-21
- Storage for an Overlaid Request

46.6 DRIVER DESCRIPTION

If the paired disk is not busy, a request is presented by FNR for completion. If a system directory request has been specified, the sector number will be removed from the directory entry and the transfer effected. Otherwise, the request is categorized as REGULAR or FORMAT. REGULAR requests are procedded by the routine LIKDUM and FORMAT requests are processed by NORMAL.

46.6.1 FORMAT REQUESTS

The most and least significant bits {MSB, LSB} of a FORMAT request are interpreted as a sector address. The routine NORMAL saves the LSB in the PHYSTB then tests the MSB to ensure it is zero. A non-zero MSB is not possible if addressing the disk on this mode. A request type check is then made by NORMAL. If a FWRITE request is to be executed, the routine WRITOP is entered to write the record. A FREAD request must first be checked to ensure it is not being overlayed. The routine CKOVRL will move the request if it is in the input buffer. The routine READOP is then entered to input the disk record.

46.6.2 REQUEST COMPLETION

Once a record has been transferred, control is passed to USEOWN. This routine causes the completion address to be scheduled and the disk unit switching to occur. Transfer is then effected to the routine responsible for finding the next request.

46.6.3 REGULAR REQUESTS

REGULAR requests are processed at label LIKDUM. The MSB and LSB of a REGULAR request is interpreted as a word address by the disk driver. In this mode, the MSB and LSB are converted to a sector and word within a sector address. A buffer, internal to the driver, is required for the processing of records transmitted in this mode. Unconditionally, the first sector {96 words} is read into the internal buffer and a test made for read or write mode.

A regular READ is first checked, and moved if necessary, to ensure that it is not being overlayed. That part of the first sector, from the °word within a sector° pointer to the 96th word, is then moved from the internal buffer to the requestor's buffer. A second request, initiated locally by the driver, transfers the remainder of the record directly to the requestor's buffer. On completion of the transfer, the routine USEOWN is entered.

Regular WRITE requests are processed in three steps. First the sector in the internal buffer is updated from the requestor's buffer, then output via the routine WRITOP. Second the number of full sectors which remain are written directly from the requestor's buffer to the disk via WRITOP. Finally, the last sector is read into the internal buffer, updated, and output. The second and third steps are performed locally by the driver.

46.6.4 SHARED ROUTINES FOR READING AND WRITING

The read routine READOP and the write routine WRITOP share common logic and differ only in the setup of the data transfer function. This routine computes a load address function from the sector number and initializes the error re-try counter. It then initiates a disk transfer at SEEKOP.

46.6.4.1 INITIATION OF A SECTOR SEEK

SEEKOP clears the COMPAR FLAG {used to cause data compare function after a read or write}. SEEKOP selects the proper unit, selects interrupts on alarm and ready not busy. It then sends the controller a seek function and goes to Label EXIT.

46.6.4.2 CONTINUATOR OPERATIONS

The continuator DKCONT checks for and ignores ghost interrupts, disables the diagnostic clock, saves the disk status, then acknowledges the interrupt. If the status is incorrect, control is passed to the error sector at DETERR. If the seek operation is complete, control is passed to Label DATA. If the ▽do not compare▽ flag is set the routine is exited. Otherwise, the compare flag COMPAR is set when control is passed to COMPOP.

At the completion of a seek operation control is passed to label DATA. The seek request flag is reset. Then several tests on the status of the controller are made. If the status word indicates that the unit is not on cylinder, control is passed to the error section at label REPEAT with error code 3 in Q. If the file address register in the controller is different than the selected file address, an error has occurred and the desired file address register is stored in the error table and control is passed to label REPEAT with error code 7 in Q.

Otherwise, a unit is selected, the alarm and end-of-operation interrupts are selected, the first word address minus one {FWD-1} is patched with the last word address plus one, and a data transfer function sent to the controller. The FWD-1 is then restored to its original value and control is passed to Label EXIT.

At EXIT the clock is turned on and control is passed to the Dispatcher.

46.7 ERROR HANDLING

When an incorrect status occurs after an operation, control is passed to Label DETERR to determine what type of error occurred. Then at Label REPEAT an entry in the error table TOTERR is incremented. If this request has been repeated less than ten times the operation will be attempted again.

If the unit is ready and not busy, control is passed to SEEKOP. Otherwise, control is passed to label WAIT to wait for one second before returning control to the driver. {Note that this driver requires a diagnostic timer to recover from this condition.}

If this request has been repeated at least ten times, then the fatal error indicator bits in the PHYSTB are set and control is passed to the error printout routine DKDIAG with the error code in Q. The error codes are shown in Section 24.3.

The disk error routine DKDIAG is entered with the error code right justified in the Q register and the address of the Phys cal Device Table in I. The DKDIAG routine is user submitted and performs functions such as message printout and recovery. Index I must not be changed on return to the driver from DKDIAG.

46.8 HARDWARE REQUIREMENT

The minimum configuration necessary for operation of this driver is:

1 1704 Computer
1 1705 Interrupt Data Channel
1 1738 Disk Controller
1 853 or 854 Disk Drive
1 850 Disk Pack

The driver will operate one 1738 controller, which may have either 853 or 854 disk drives.

Additional hardware recommended:

1 1573 Line synchronizer or equivalent so that the diagnostic timer program can be used to detect I/O hangups on the disk.

46.9 TWO DRIVES ON ONE CONTROLLER

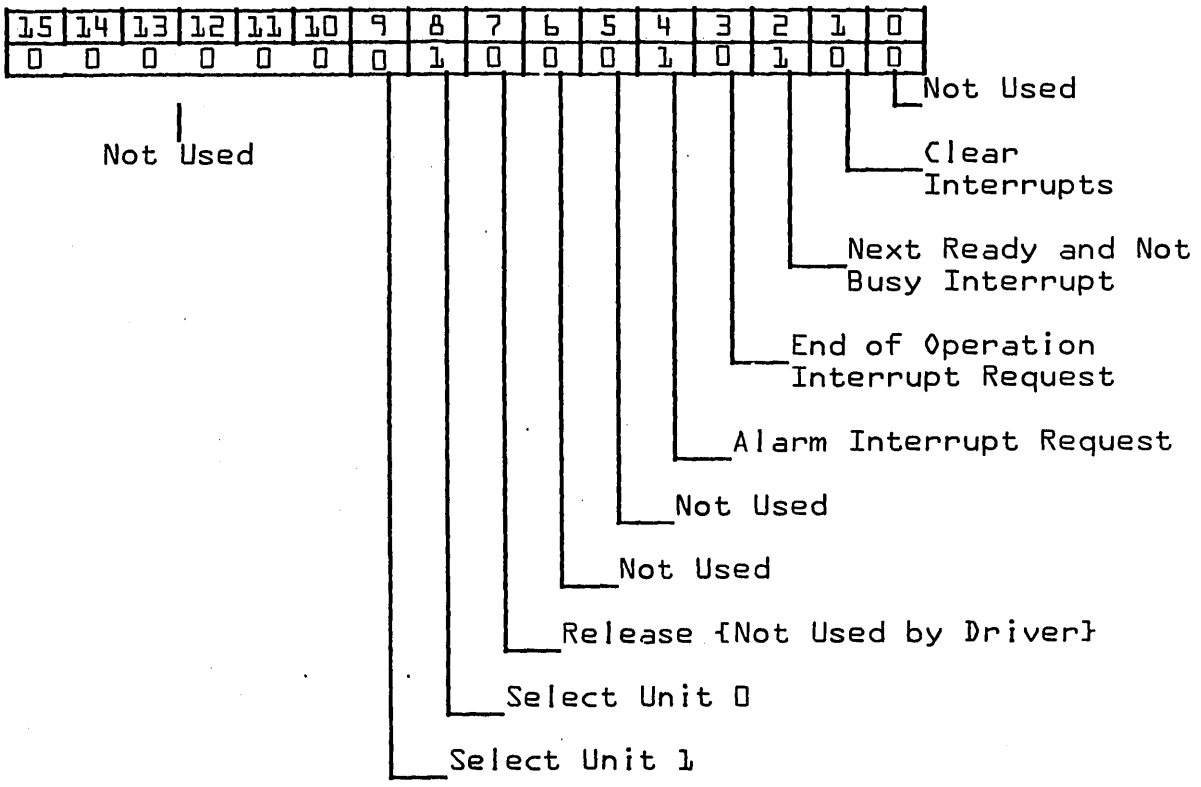
The driver, through PHYSTB settings, can identify two disk drives which are connected to a common controller. Since the controller can operate only one drive at a time, the driver will not attempt to operate one drive when the other is busy. However, at the completion of each request, a check is made to see if the other drive has a request waiting. If so, the other drive is put into operation. It can be seen then that if two disks are connected to the same controller and each has several requests queued, then the two disk drives will be operated alternately until the queues are empty.

45.10 INTERRUPT OPERATION

The driver does all operations using interrupts, i.e., the completion of seeks and data transfers are signalled by the occurrence of certain selected interrupts. Errors are detected in the same manner.

Data transfers are ended when end-of-operation is signalled. Seeks are ended when ready and not busy is signalled.

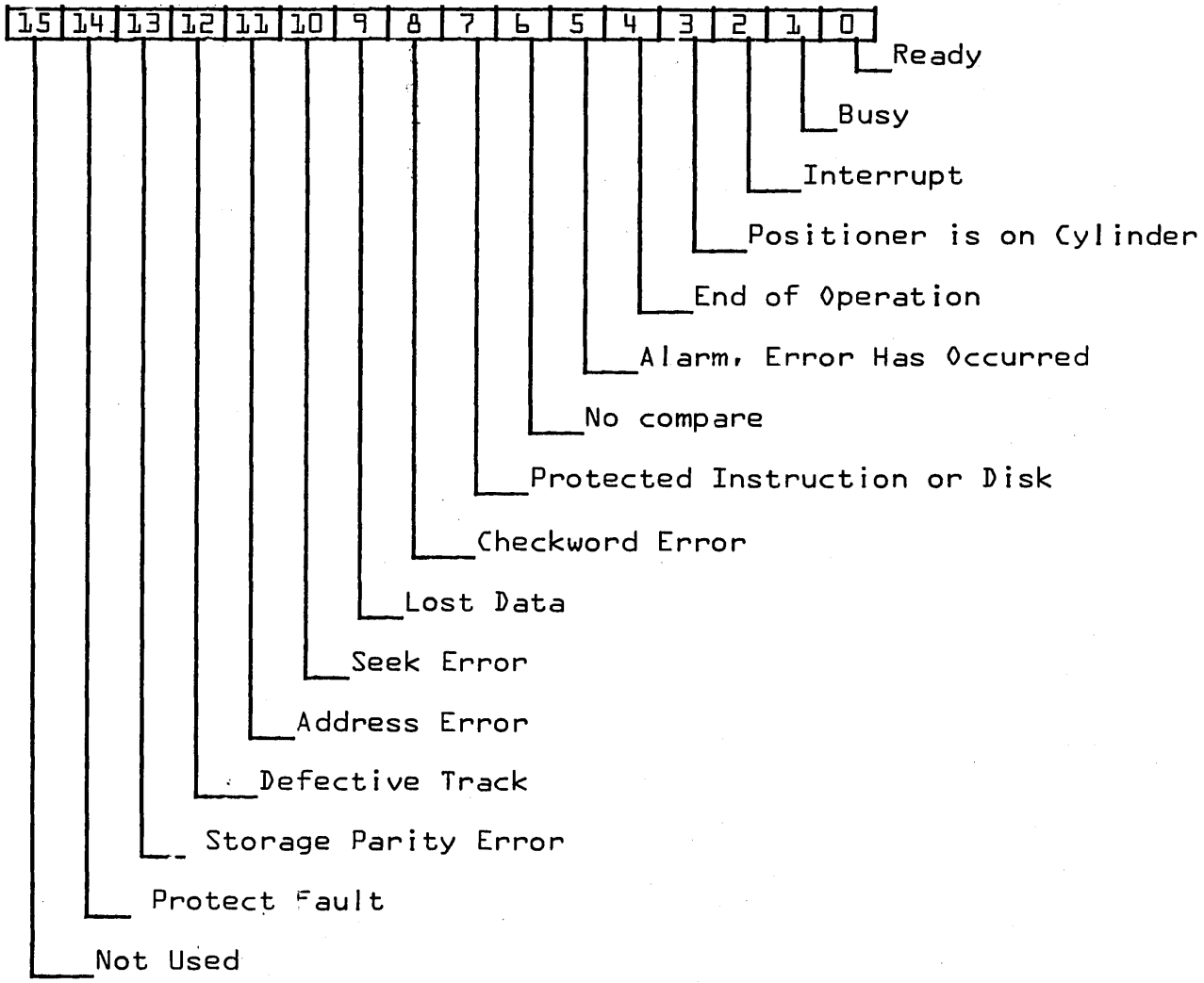
CONNECT CODE



Driver uses following two codes:

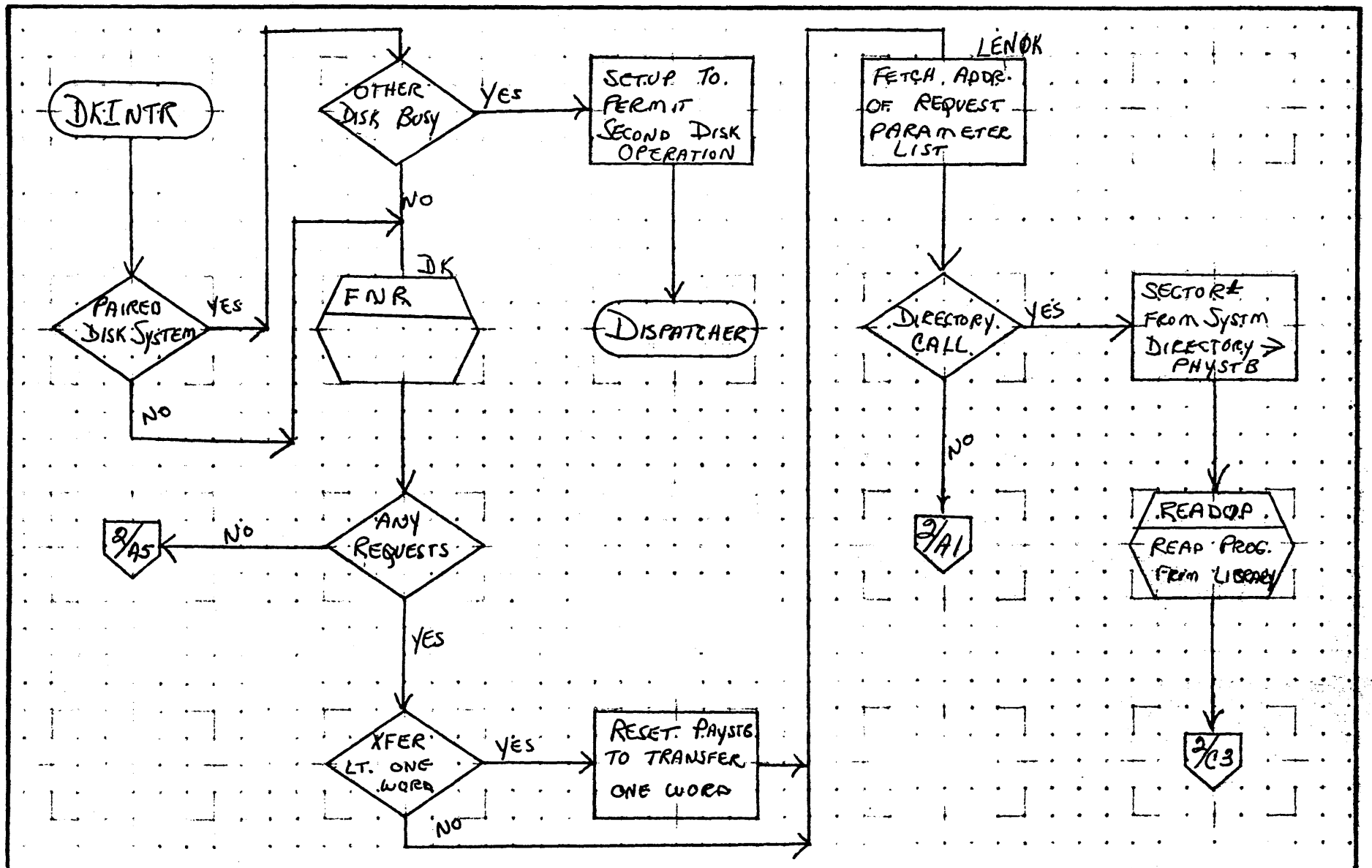
114 for Unit 0
214 for Unit 1

STATUS REPLY



ACCESS TIME FOR AN 853/854 DISK DRIVER

	Maximum	Average
Cylinder Positioning	145 ms	95 ms {2/3 Max Movement}
Latency	25 ms	12.5 ms {1/2 Disk Rev.}
Access	170 ms	107.5 ms



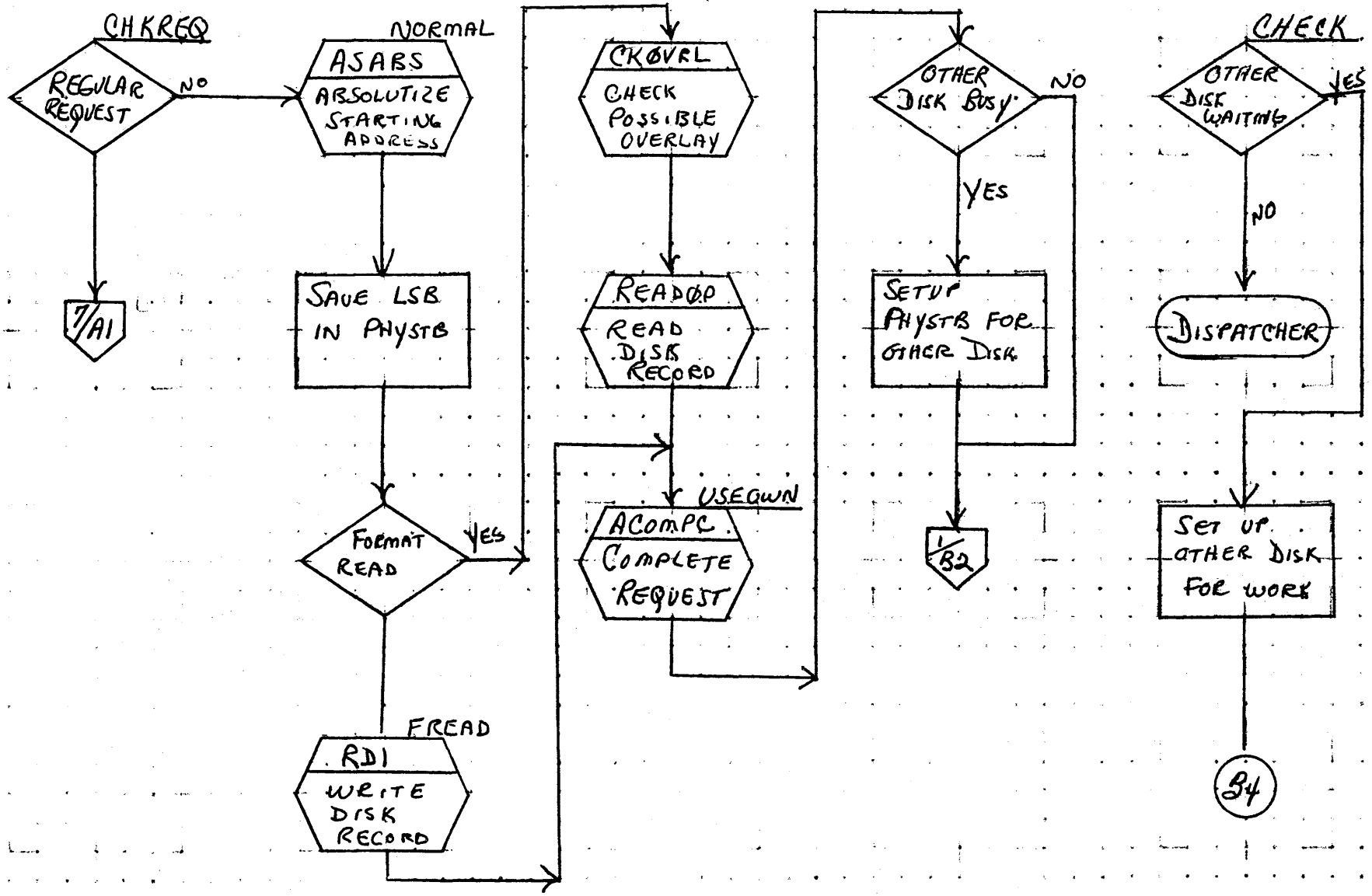
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: DISKWD		PROJECT MGR.			
FLOWCHART ☐	DISK DRIVER	PAGE 1 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

46-11



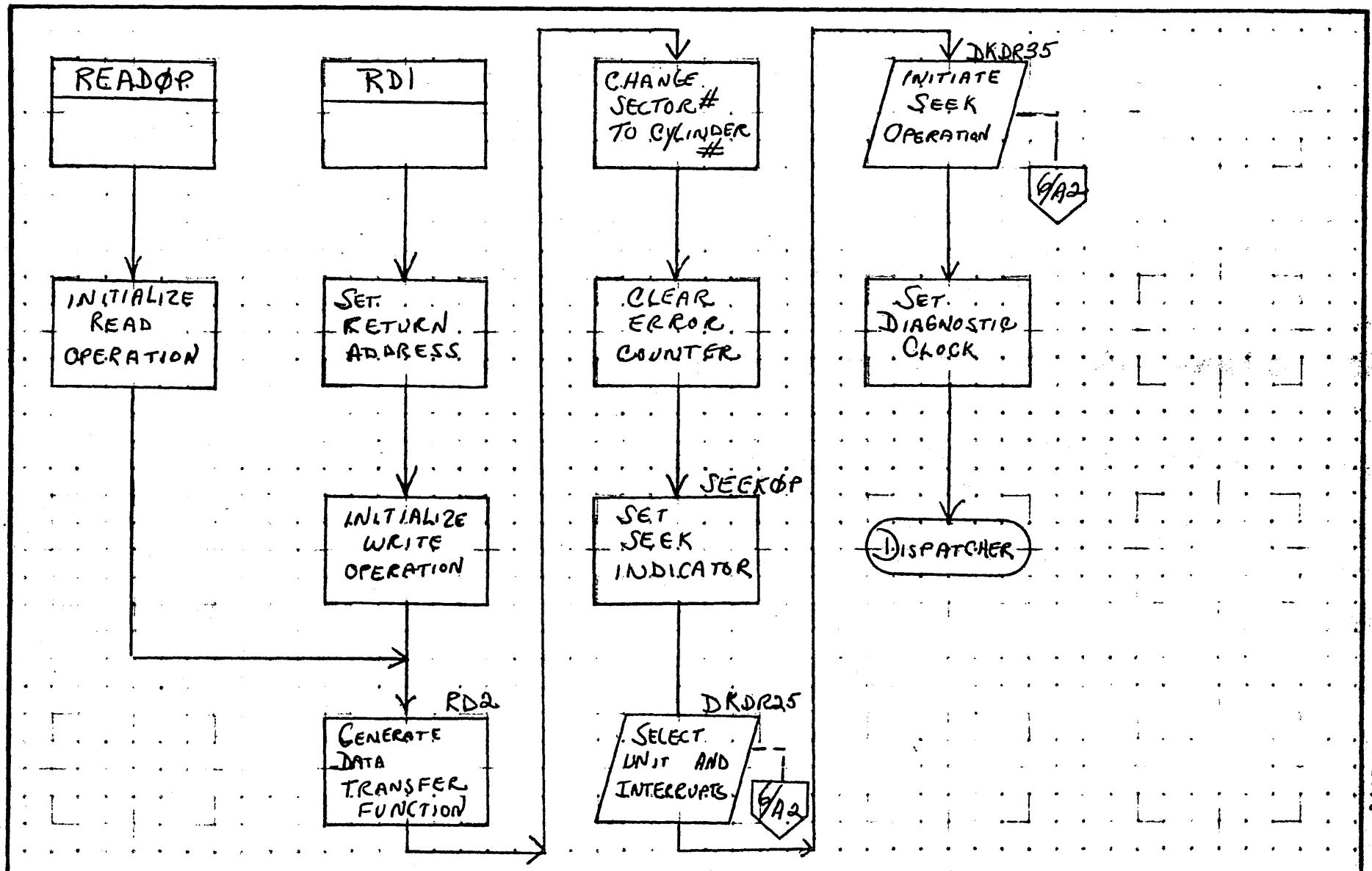
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DISKWD	PROJECT MGR.			
	DISK DRIVER PAGE 2 OF 9	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

49.12



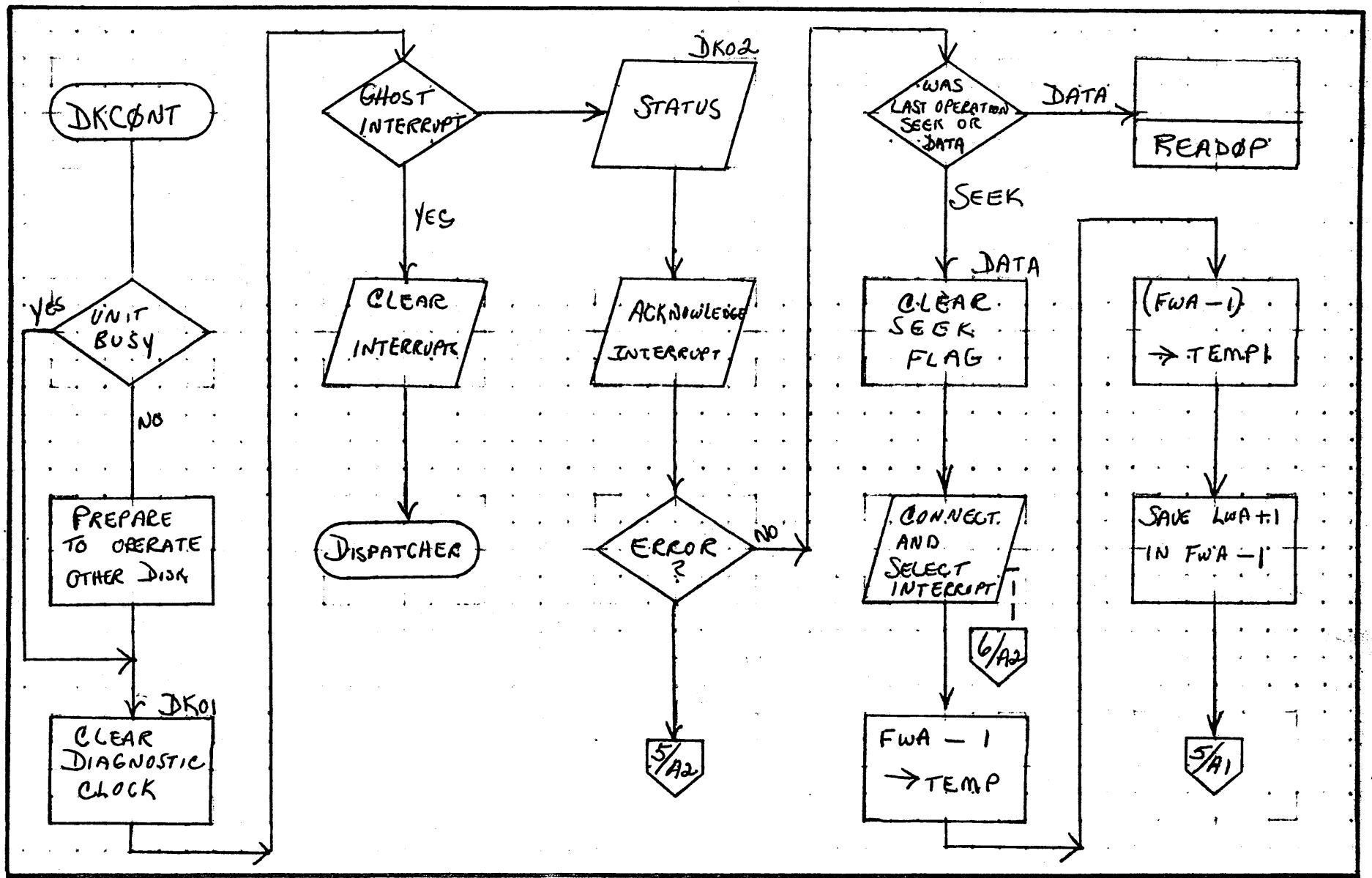
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DISKWD		PROJECT MGR.		
	DISK DRIVER	PAGE 9 OF 9	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

60.13

AA1385 (FORMERLY CA127-1)

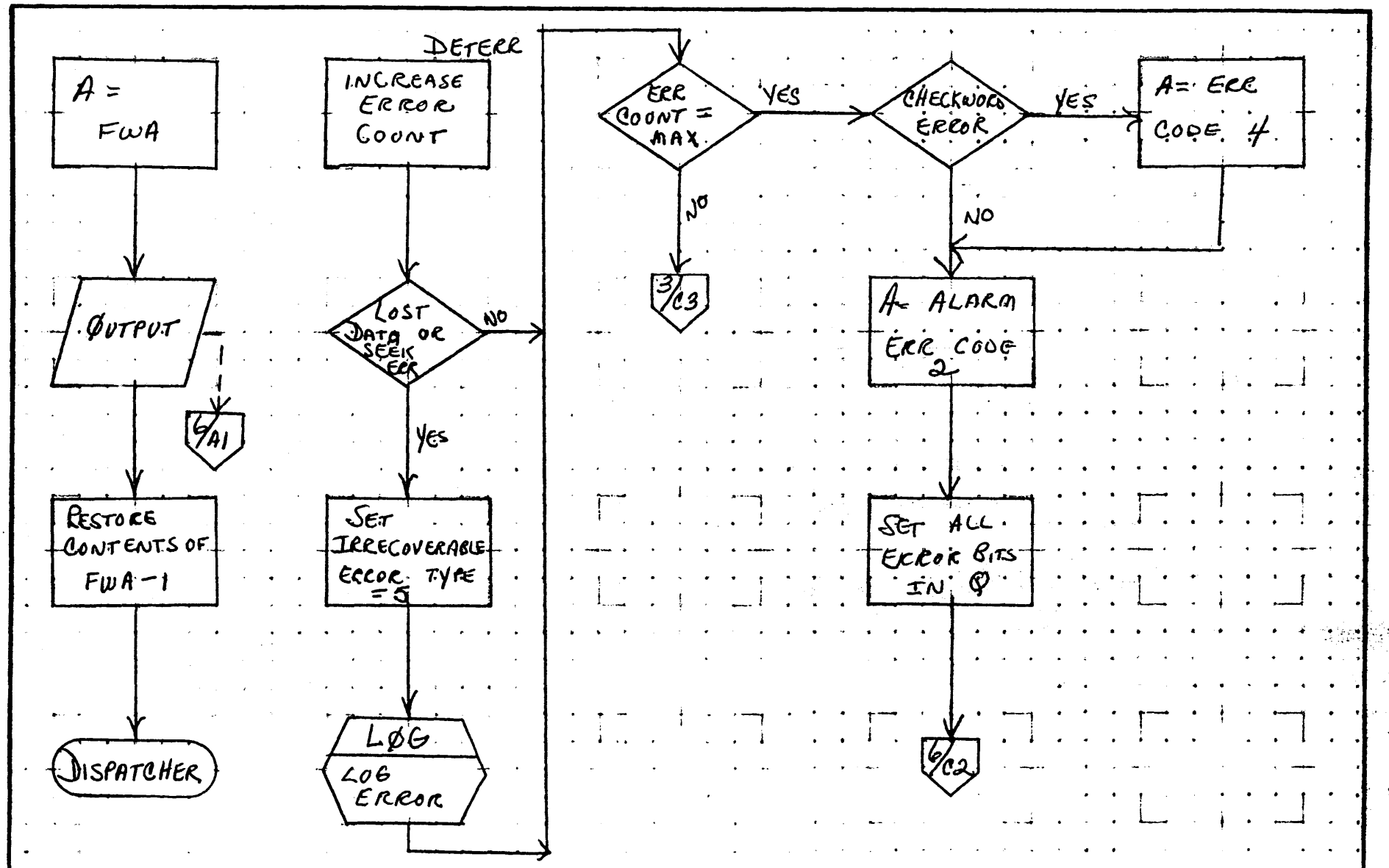
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS Ims	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DISKWD		PROJECT MGR.			
FLOWCHART ☐	DISK DRIVER	PAGE 4 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

46.14



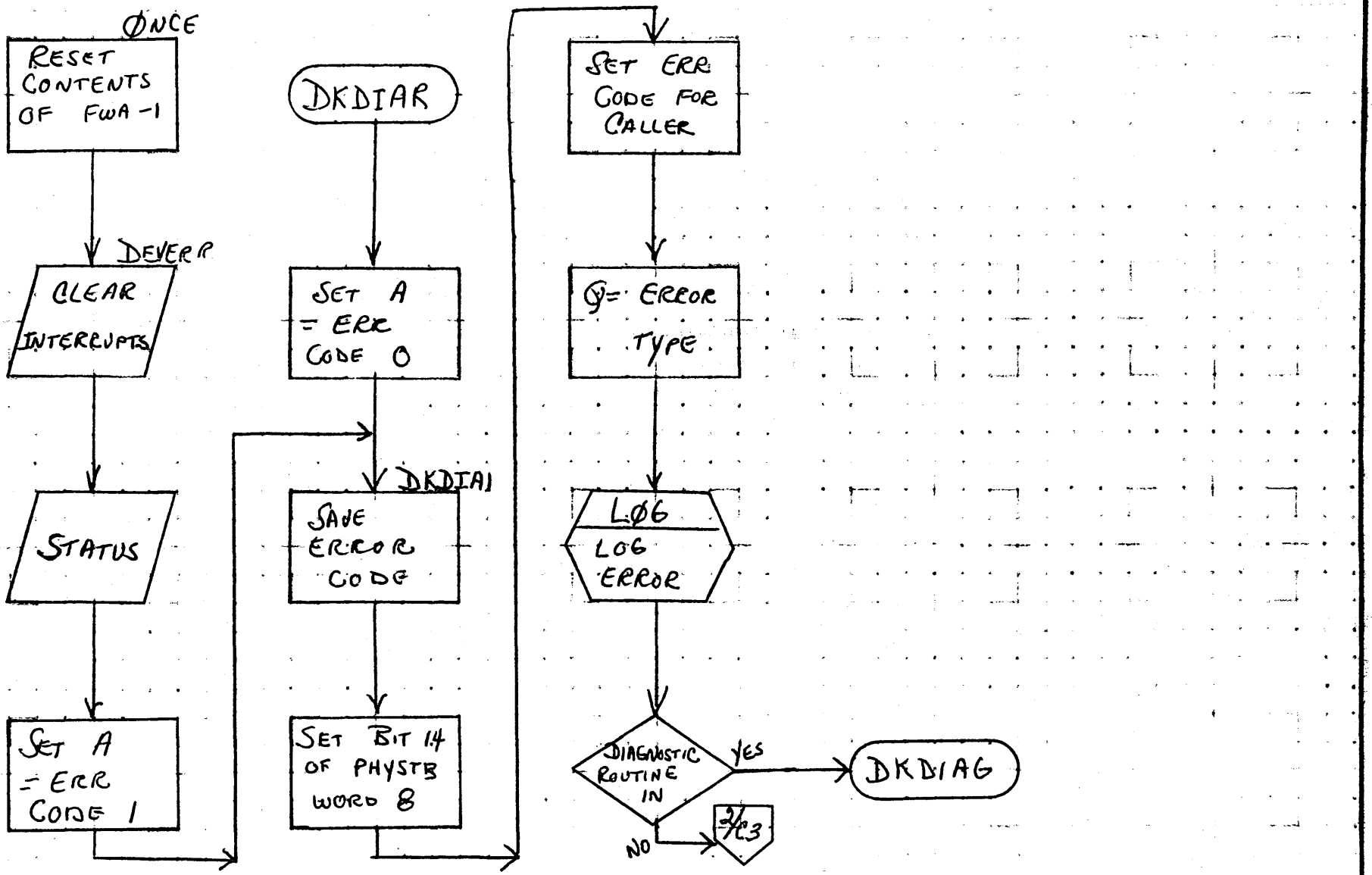
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DISKWD		PROJECT MGR.			
FLOWCHART ☐	DISK DRIVER	PAGE 5 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

46.15



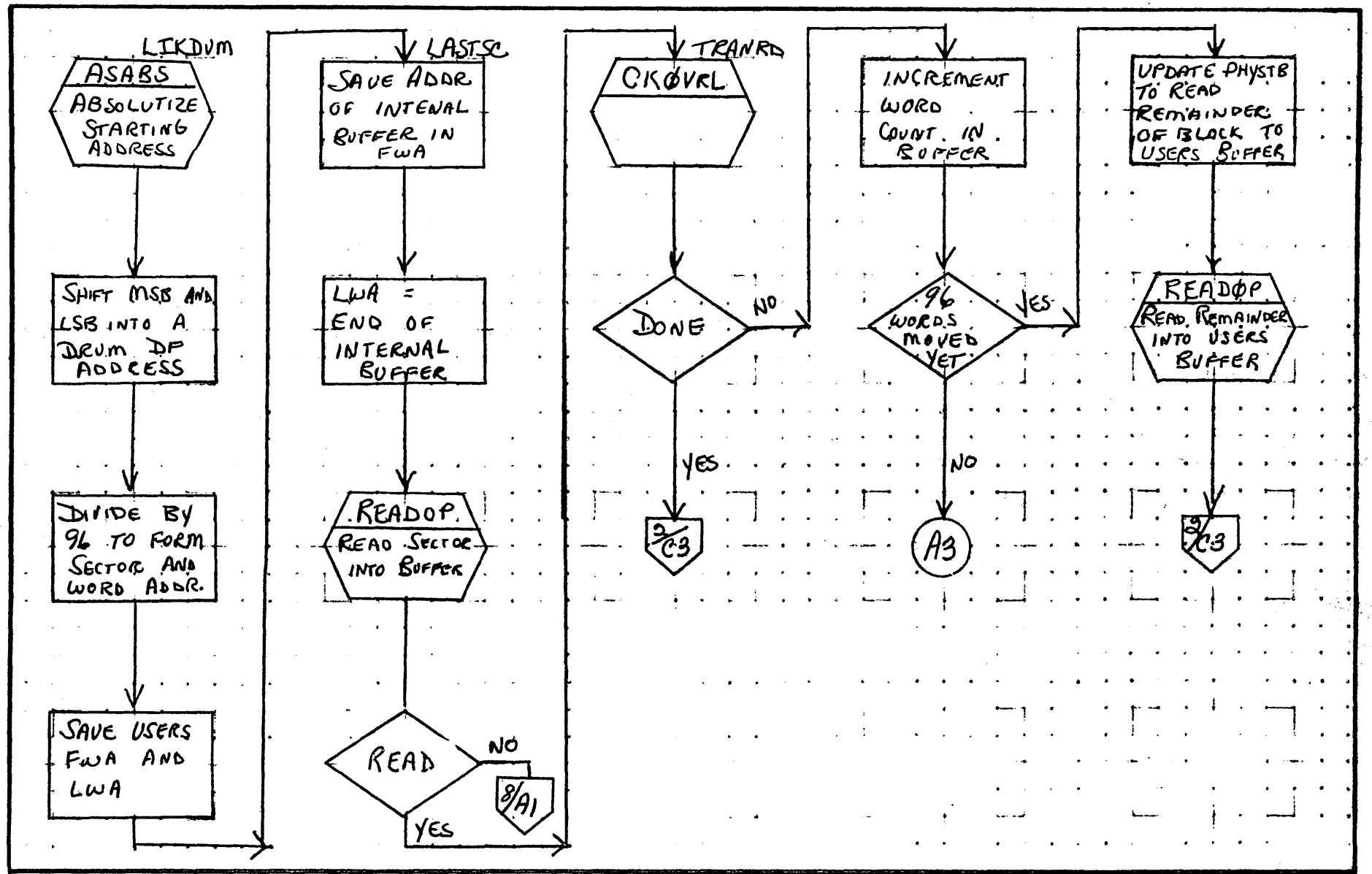
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DISKWD		PROJECT MGR			
FLOWCHART ☐	DISK DRIVER	PAGE 6 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

41-16

AA1385 (FORMERLY CA127-1)

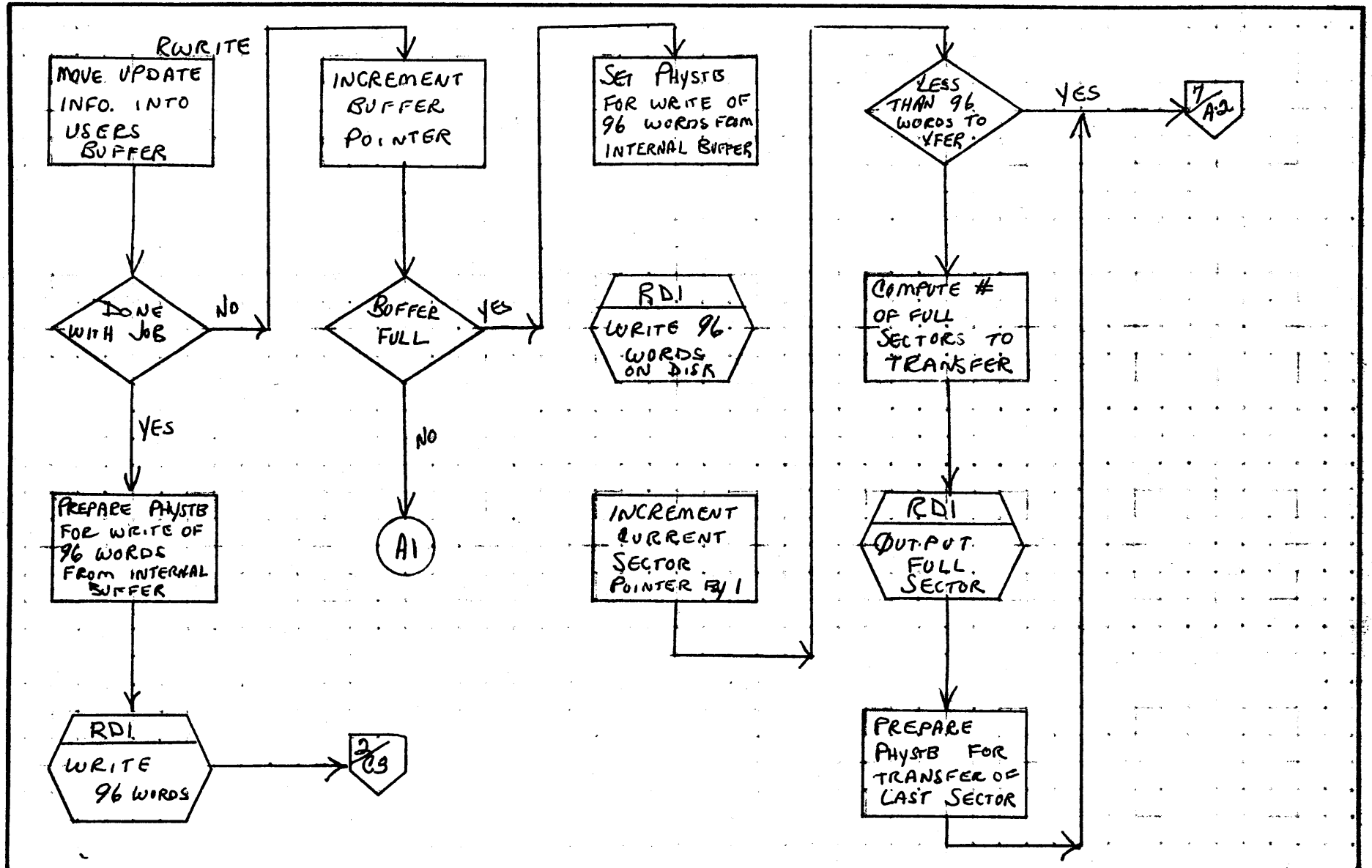
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DISKWD		PROJECT MGR.			
FLOWCHART ☐	DISK DRIVER	PAGE 7 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

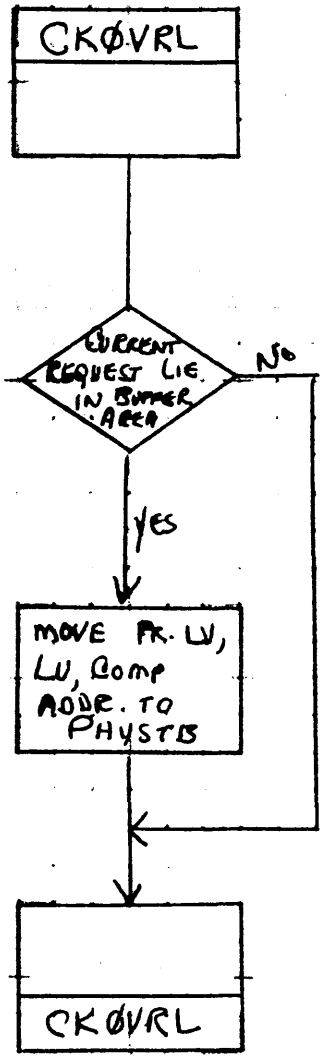
46.17



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: DISKWD		PROJECT MGR.			
FLOWCHART ☐	DISK DRIVER	PAGE 8 OF 9	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

45-18



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DISKWD	PROJECT MGR.			
	DISK DRIVER PAGE 9 OF 9	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

46.19

47.0 1751 DRUM DRIVER

47.1 FUNCTION

The Drum Driver will execute the I/O commands necessary to process a user's request to read or write the drum memory. The requests may be of variable length and may transfer information anywhere in core to or from anywhere on drum.

The driver relinquishes control of the Monitor during all I/O transfers, since the I/O proceeds without program attention via the Direct Storage Access. An interrupt returns control to the driver upon end of operation or the occurrence of any error.

The drum driver will handle five request codes, Read System Directory Format (0), Read (1), Write (2), Format Read (4), Format Write (6).

The drum driver will handle the recoverable error conditions on the drum and will update the Physical Device Table. The drum driver also will simulate disc operations.

47.2 ENTRY POINTS

DRMINT	initiator address
DRMCON	continuator address
DRMERR	error entry

47.3 EXTERNALS

DMDIAG	drum diagnostic routine

47.4 ENTRY INTERFACES

All entry points require that the Q-register contains the address of the drum physical device table.

47.5 EXIT INTERFACES

None

47.6 GENERAL PROGRAM INFORMATION

47.6.1 INTERNAL SYMBOLS

OREJ	Control comes to here on output reject.
TIMERA	Timing error flag (while idle).
TIMERB	Timing error flag (while busy).
MAXERR	Contains the maximum number of times a request will be repeated if a timing error or parity error is detected.
TOTERR	Contains the accumulation of all errors.
ERRCNT	Current number of errors on this request.
STATUS	A subroutine to read drum status
INTREJ	Control comes here on input rejects (internal).
EXTREJ	Control comes here on input rejects (external).
SETEFD	Set error field subroutine.
DDIAGL	Call Drum Diagnostic subroutine
amoni	Indirect address of the Monitor Entry, location $F4_{16}$.

47.6.2 USER INSTRUCTIONS.

Physical device table set up. A listing of the drum device table is attached. DRUM must be an entry point.

At system initialization time or after a master clear, the drum controller should have its interrupt enabled. The following code performs this operation:

```
LDA = N$101
ENA   $A        Clear interrupt
Out   -1        and request interrupt on end of
                operation.
```

User supplied interrupt response routine.

```
INTDRM   LDQ =XDRUM              DEVICE TABLE

         JMP* (DRUM+2)           GO TO CONTINUATOR
```

47.6.3 PHYSICAL DEVICE TABLE

```
DRUM    NUM     $120A       SCHEDULE DRIVER AT LEVEL 10
        ADC     DRMINT              EDIN
        ADC     DRMCON              EDCN
        ADC     DRMERR              EDPGM
        NUM     -1                  EDCLK
        NUM     0                   ELU
        ADC     0                   EPTR
        ADC     E+1                 EWES
        NUM     $66         DRUM STORAGE T=6, READ
                            AND WRITE =3
        NUM     $200                ESTAT1    MMBIT
        ADC     0                   ECCOR
        ADC     0                   ELSTWD
        NUM     0                   ESTAT2
        BSS     DRMCLL (6)  CALL MOVED HERE ON OVERLAY
TRACK   ADC     0,8+E       12 BITS-TRACK
SECTOR  ADC     0,$A+E      11 BITS-SECTOR
S       ADC     0,$C+E      15 BITS-INITIAL CORE
FNCORE  ADC     0,$C+E      15 BITS-FINAL CORE
        ADC     8,1+E       END OF OPERATION INTERRUPT
                              A,Q
RWCTRL  NUM     0,0         END OF SEQUENCE
        NUM     0,-1        SYSTEM DIR READ CODE
        ADC     RCRR
RC      NUM     0           REQUEST CODE
DN      NUM     0           NUMBER OF WORDS
WRITEC  ADC     E           WRITE CODE IN Q
READC   ADC     E+$4        READ CODE IN Q
```

Word 33 of the device table contains the Read Request Code to be used by the driver for reading in System Directory programs. I must be set to either RCRR (Request Codes for Regular Read, 1) for a stand-alone Monitor or RCFR (Request Code for Format Read) for use in the Operating System .

47.8 Program Description

Initiator section. Upon entry, the Q-register contains the address of the physical device table for the drum. The drum driver calls the find next request routine, FNR, which removes the top entry from the drum's thread, stores the location of the parameter list in the physical device table, and returns control to the drum driver. If no more requests are on the drums thread, then the driver exits to the Dispatcher. If this is an overlay, the parameter list is moved to the device table. The driver extracts the starting address of core S and the mass memory address MMA. Then the driver executes the output commands which initiate the I/O transfer. The driver gives control to the dispatcher. The I/O transfer proceeds without program attention.

REJECTS

If an external reject occurred on the output command, the status is read and stored in the device table. And if a timing error was present, timing error flag A is set, flag B is cleared, and control is given to the dispatcher. If no timing error occurred, the drum diagnostic is called with error 7 in Q and the status in A. The error field is set in the device table and the driver proceeds to A1 to schedule the completion address. If an internal reject occurred, the diagnostic is called with error 4. The error field is set and the call is completed.

INTERRUPT ENTRY SECTION

At the end of operation or when an error occurs an interrupt occurs. The user supplied response routine enters the Interrupt entry DRMCON. If timing error flag A is set, it is cleared {this is the resynch interrupt} and the diagnostic is called with a six in Q. Control then goes to the error counter check section. If timing error flag B is set, it is cleared {this is the resynch interrupt} and diagnostic three is called. Control then goes to the error counter check. If no errors exist in the status word, control is given to A 1 to complete the call. Otherwise, an error exists. If it is the guarded address error, the program protect fault routine PFAULT is called, the error field set and control goes to A1.

If the timing error is set, then flag B is set, flag A is cleared and control goes to the dispatcher to await the resynch interrupt.

ERROR COUNTER SECTION

If the error counter, ERRCNT, now equals the maximum number of errors (MAXERR), the error field is set, the diagnostic two is called and then control goes to A1. Otherwise the error counter is incremented, the status is stored in the device table, diagnostic one is called and control goes to A2 to repeat the request.

COMPLETION OF THE CALL SECTION

If the completion address in the users calling sequence lies within the area of a read, or if this is a directory call and the read was unsuccessful, then drum diagnostic will be called (error 8) the completion address will not be scheduled and the drum driver will proceed to the next request. If the transfer was unsuccessful, due to a timing error, parity error, or some other undiagnosed error condition, the drum diagnostic will be called (error 2), the error field V will be set (passed to user in Q at completion address). Upon encountering an error the driver calls upon the drum diagnostic program DRMDIG to print out the error code and drum status.

The error codes are as follows:

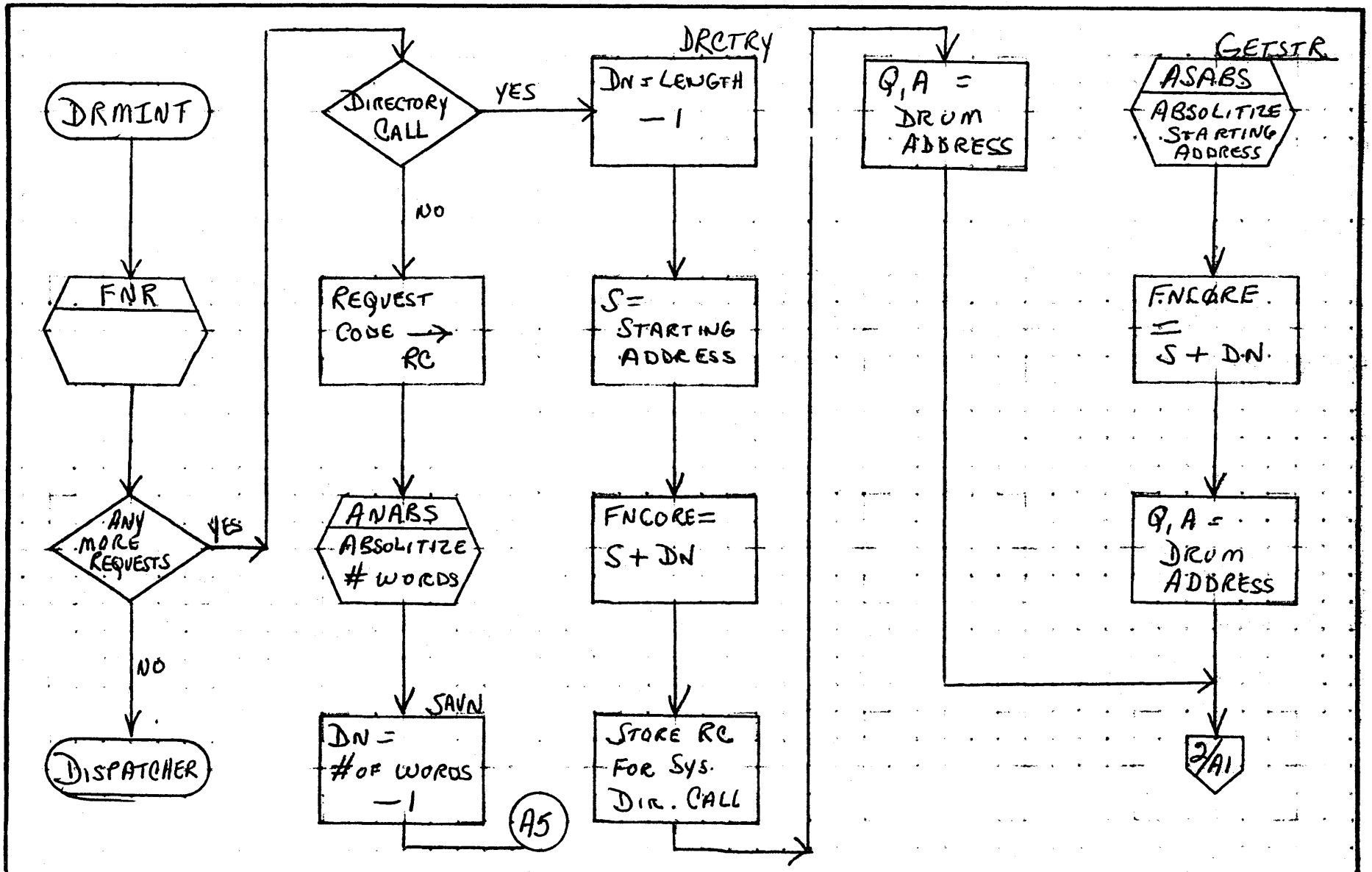
Error Number	Description
1	Repeated the request due to an error.
2	Request not successfully completed. Attempted to repeat the maximum number of times.
3.	A timing error occurred while the drum was busy.
4.	An internal reject on input or output command occurred. (Equipment not turned on or equipment code not set to 2.)
5.	An external reject on input command. Controller has malfunctioned.
6	A timing error occurred while drum was not busy.

Error Number	Description
7	An external reject occurred on an output command and no timing error was set. This is a controller malfunction.
8	Either a directory call or an overlay read request was not completed due to an irrecoverable error.
9	A guarded address error on a write, location of the request is in the A-register.

If this is a directory read request, all of the Q-register will be passed to the user. Otherwise the error field is placed in the high order three bit of Q. The Complete Request Routine is called to schedule the completion address. Control is given to the initiator section to find the next request.

47.9 Program Limitations and Restrictions:

The program as coded is restricted to running only one drum.



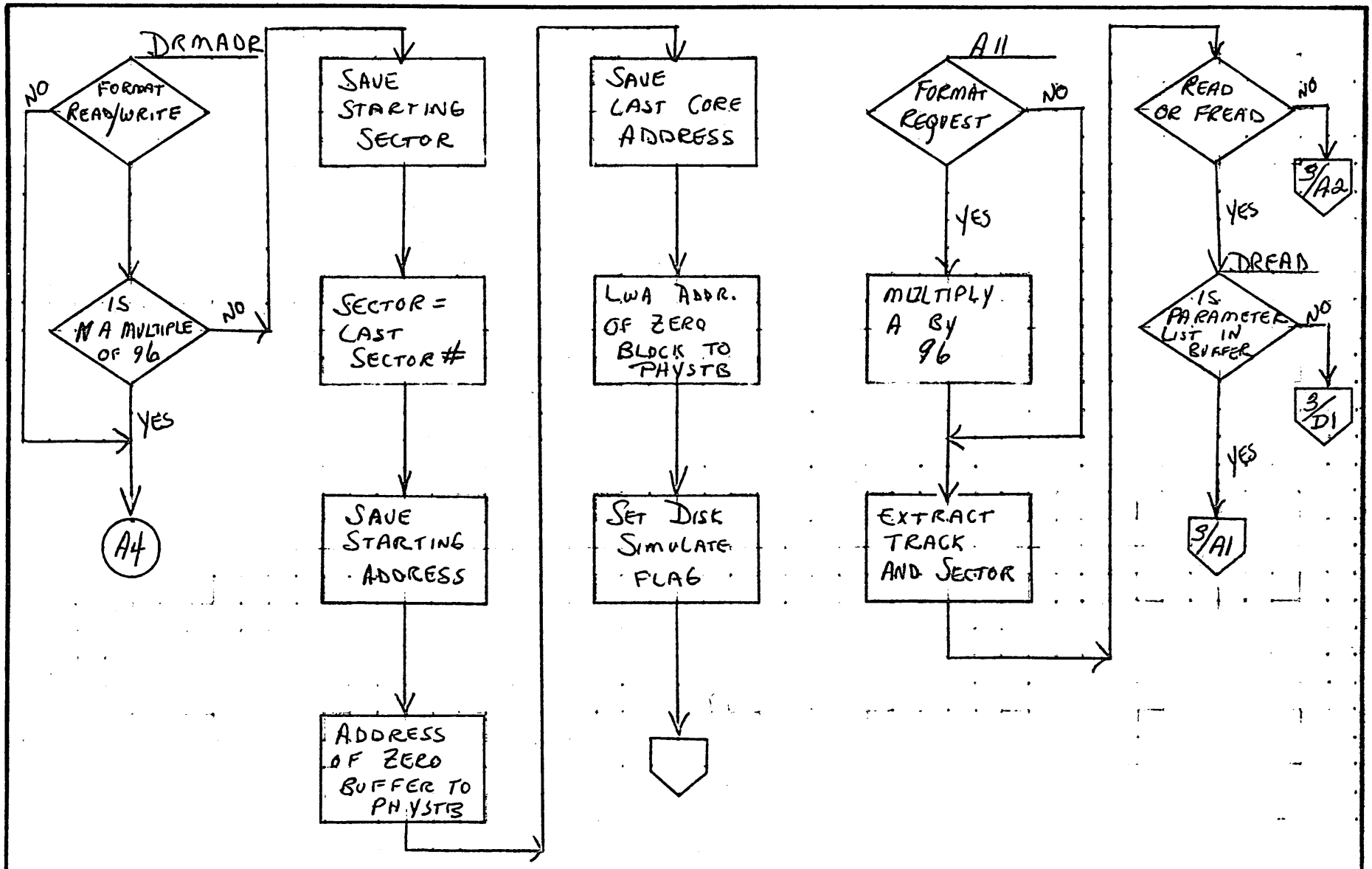
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE DRUM DRIVER	PROJECT MGR.			
	PAGE 1 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

47.7

AA1385 (FORMERLY CA127-1)

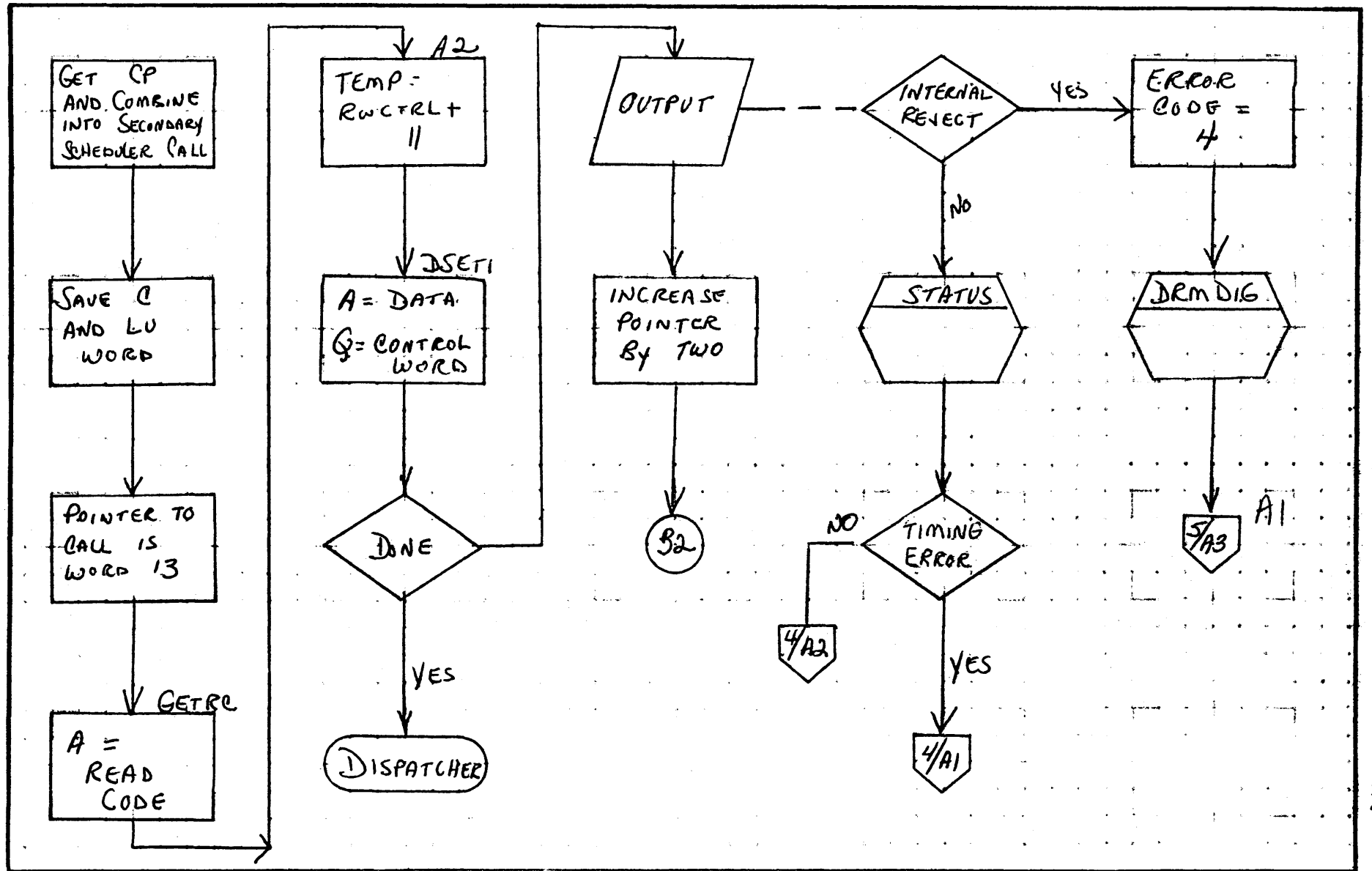
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DRUM DRIVER		PROJECT MGR.		
		PAGE 2 OF 7	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

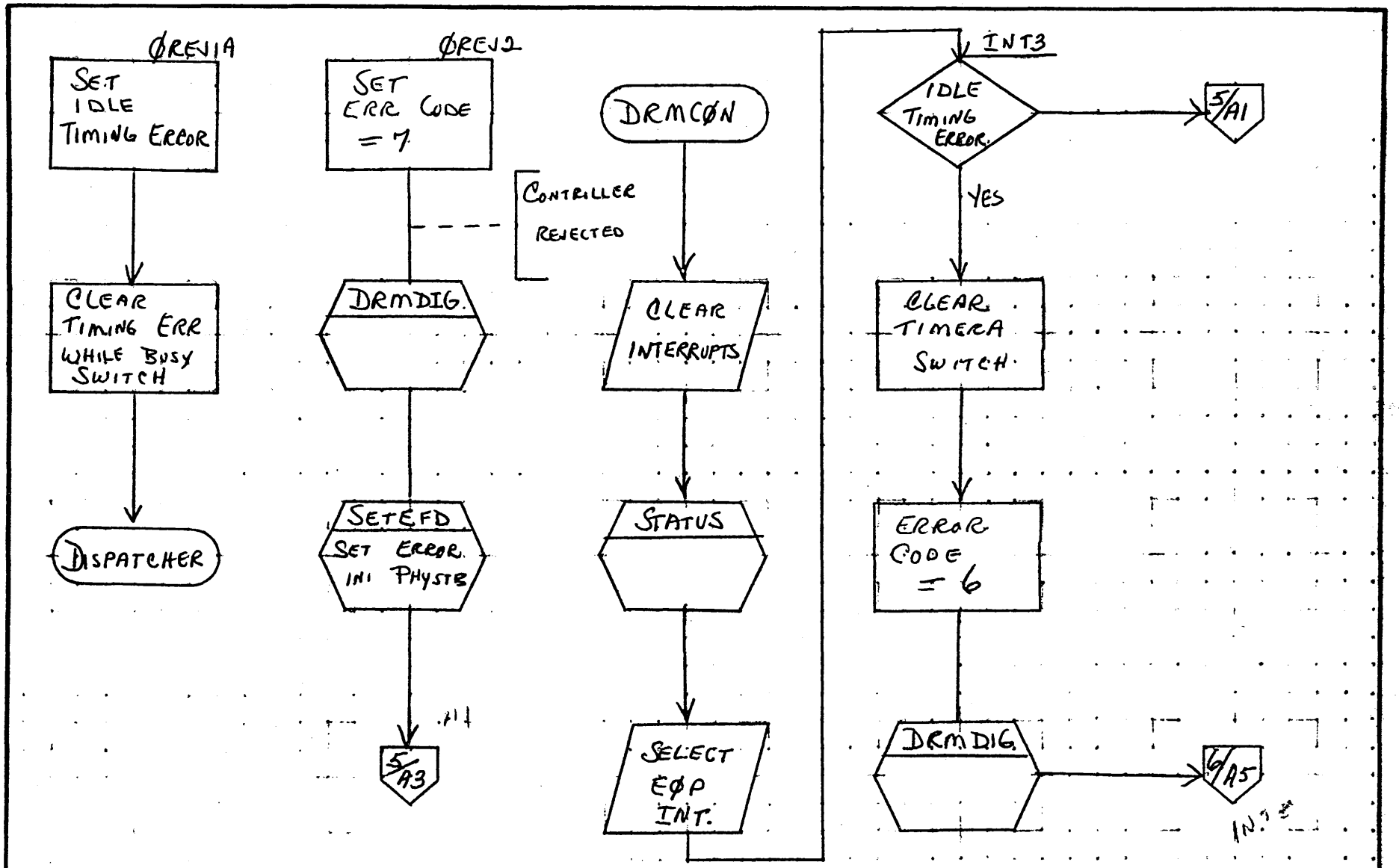
47.8



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS JMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DRUM DRIVER		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

42-19



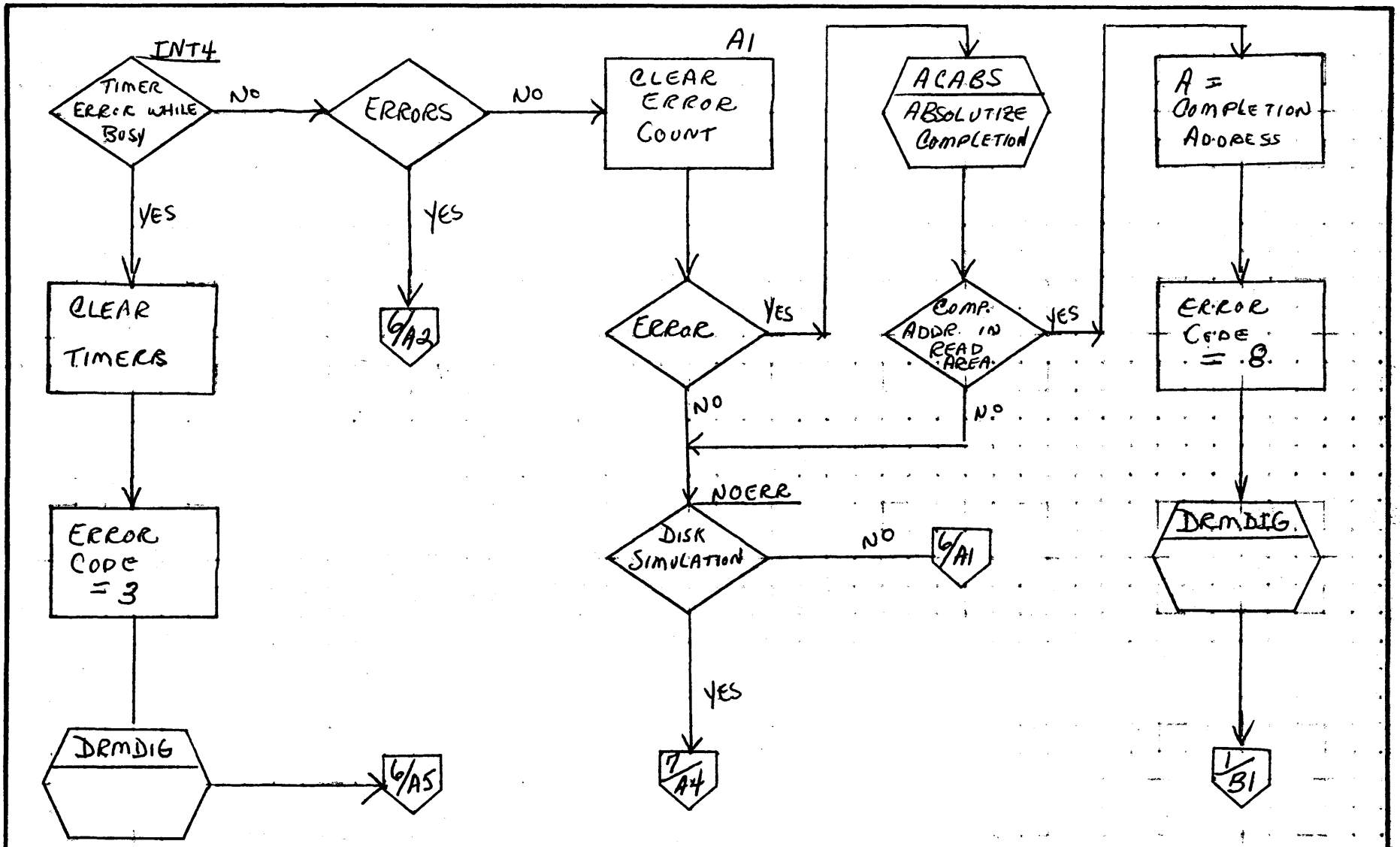
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DRUM DRIVER		PROJECT MGR.		
		PAGE 4 OF 7	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

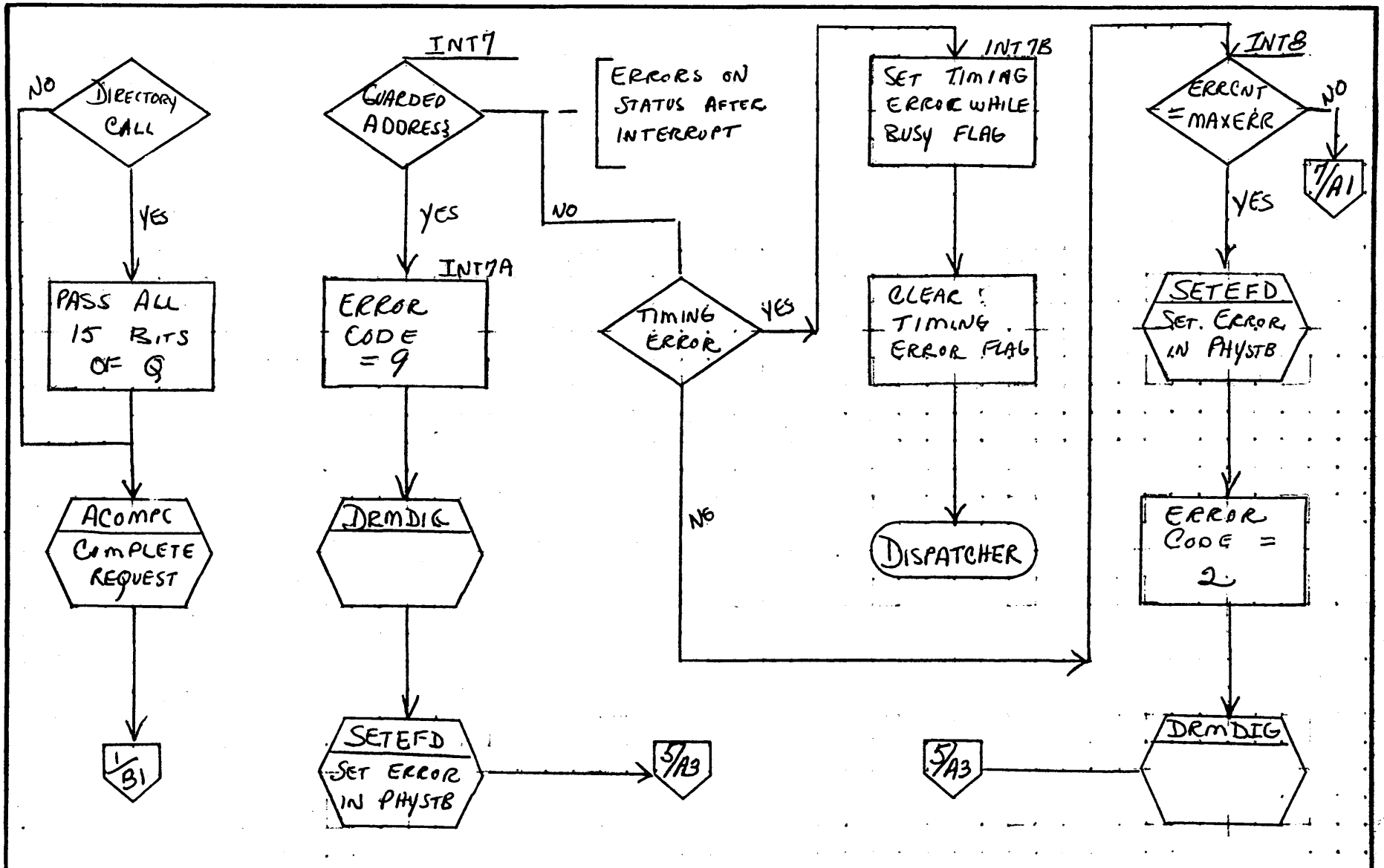
PRINTED IN U.S.A.

MAR 5 1971

47.10



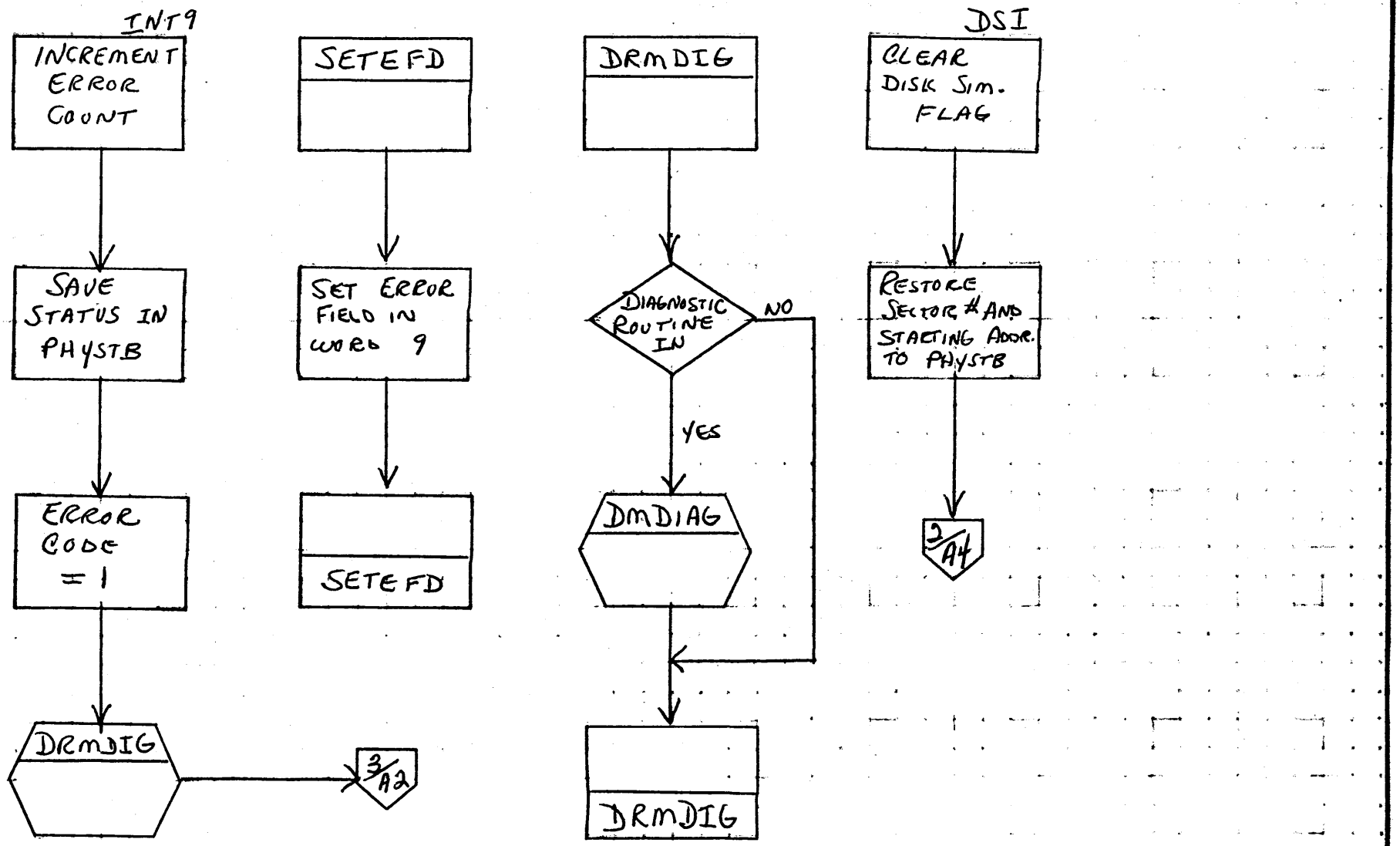
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DRUM DRIVER	PROJECT MGR.			
	PAGE 5 OF 7	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DRUM DRIVER		PROJECT MGR.			
FLOWCHART ☐		PAGE 6 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

47.12



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DRUM DRIVER		PROJECT MGR.			
FLOWCHART ☐		PAGE 7 OF 7	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

47.13

48.0 1777 PAPER TAPE DRIVER

48.1 FUNCTION

The 1777 Paper Tape Station Driver is a standard driver which will operate under V3.0 Operating System. This driver is essentially the 1721/1722 Reader Driver and the 1723/1724 Punch Driver. Compatability is such that the 1777 Paper Tape Station Driver can replace the 1721/1722 Reader Driver and the 1723/1724 Punch Driver.

48.2 MODULES

The 1777 Paper Tape Station Driver is composed of four (4) modules.

1. MASDRV — Mass memory driver control program
2. PUNCDR — Paper tape punch driver
3. PTREAD — Paper tape reader driver
4. STCK — Common routines to the Paper Tape Punch and Reader

When the 1777 Paper Tape Station Driver is mass memory, a small routine precedes the punch driver and the reader driver. This routine calculates and inserts the absolute addresses of the drivers Initiator, Continuator and Error Routine into the PHYSTAB. Also STCK is internal to the punch driver module and reader driver module.

When the 1777 Paper Tape Station Driver is core resident STCK is external to the punch driver module and reader driver module.

48.3 1777 PUNCH DRIVER MASS MEMORY

48.3.1 ENTRY SYMBOLS

PUNCHI	Driver Initiator Address
PUNCDR	Driver Continuator Address
PCHERR	Driver Error routine for Diagnostic Timer.

48.3.2 EXTERNAL SYMBOLS

MAS300	Entry in MASDRV. It will check if any mass memory drivers are waiting in core.
MAKEQ	Routine that sets the proper error bits in word nine (9) of PHYSTAB to be passed to user.
ALTDEV	Program that processes irrecoverable errors.

48.3.3 PHYSICAL DEVICE TABLE

The Physical Device Table for the punch driver mass memory is set up as follows:

```
         ENT   PPTPCH
         EXT   MASDRV,TP1777
         ADC   TP1777
PPTPCH   NUM   $12AA          Punch Entry
         ADC   MASDRV         Initiator Entry
         ADC   0              Continuator Entry
         ADC   0              Time Out Entry
         NUM   -1             Diagnostic Clock
         NUM   0              Logical Unit
         NUM   0              Call Parameter List
         NUM   $XXXX          Hardware Address
         NUM   $XXXX          Request Status
         NUM   0              Status Word
         NUM   0              Current Core Location
         NUM   0              Last Word Address +1
         NUM   0              Hardware States
         NUM   $F8            Driver Length
         ADC   TP1777         Name associated with sector #
         NUM   0              Checksum Word
         NUM   0              Length of Record
         NUM   0              Return Address FNR
         NUM   0              Return Address Common
         NUM   0              Error Code Storage
         NUM   0              Temporary Storage
         NUM   0              Flag of Lost Data
```

48.3.4 SUBROUTINES

48.3.4.1 STCK10 {Status Check}

Clears the interrupt and checks status to see if alarm condition occur. If alarm occurs it checks for lost data if so exit if not set alarm code in A and go to STCD10.

Normal return is at P+2

48.3.4.2 STCD10 {Set Code}

This routine is entered when there is a non recoverable error. Upon entering, A contains the error code. The logical unit is then Exclusive Ored in A. This then goes to MAKEQ and then to Alternate Device Handler.

48.3.4.3 SINT10 {Set Interrupts}

This routine goes to find next request. Upon P_2 return it sets the interrupts, starts motion, and exits.

Upon P+1 return it exits to MASDRV because no more requests are threaded.

48.3.4.4 CREQ10 {Clear Interrupt Request}

This routine clears interrupts, inputs status, goes to MAKEQ and exits.

48.3.5 DESCRIPTION

Initiator

PUNCHI is the initiating entry to the driver. At entry exit is made to SINT10. Upon return checksum is cleared and exits to the dispatcher.

Continuator

PUNCDR is the continuator entry. At entry exit is made to STCK10 upon return at P+2 no errors have occurred, if return was at P+1 a jump to the validation error routine would occur. PTP500 is the tag address.

Data Transfer Operation Punch

Formatted Binary

The complement of the length is punched on the first two (2) frames of each record. Each word is added to the checksum and outputted by character. The path that formatted binary uses is PUNCDR, PTP010, PTP020, PTP030, PTP050, PTP070, PTP080, PTP100, PTP160, PTP200, and PTP300. Upon completion the checksum is complimented and written out for parity purposes.

Binary

Binary is a direct data to punch. The path that binary uses is PUNCDR, PTP010, PTP020, PTP030, PTP070, PTP080, PTP100, PTP160, PTP200 and PTP300. Upon completion the checksum is not punched.

Formatted ASCII

Each character punched has its own parity and it is even parity. The path that formatted ASCII uses is PUNCDR, PTP010, PTP020, PTP030, PTP040, PTP070, PTP080, PTP100, PTP105, PTP110, PTP140, PTP200, and PTP300. Upon completion a carriage return and line feed is punched.

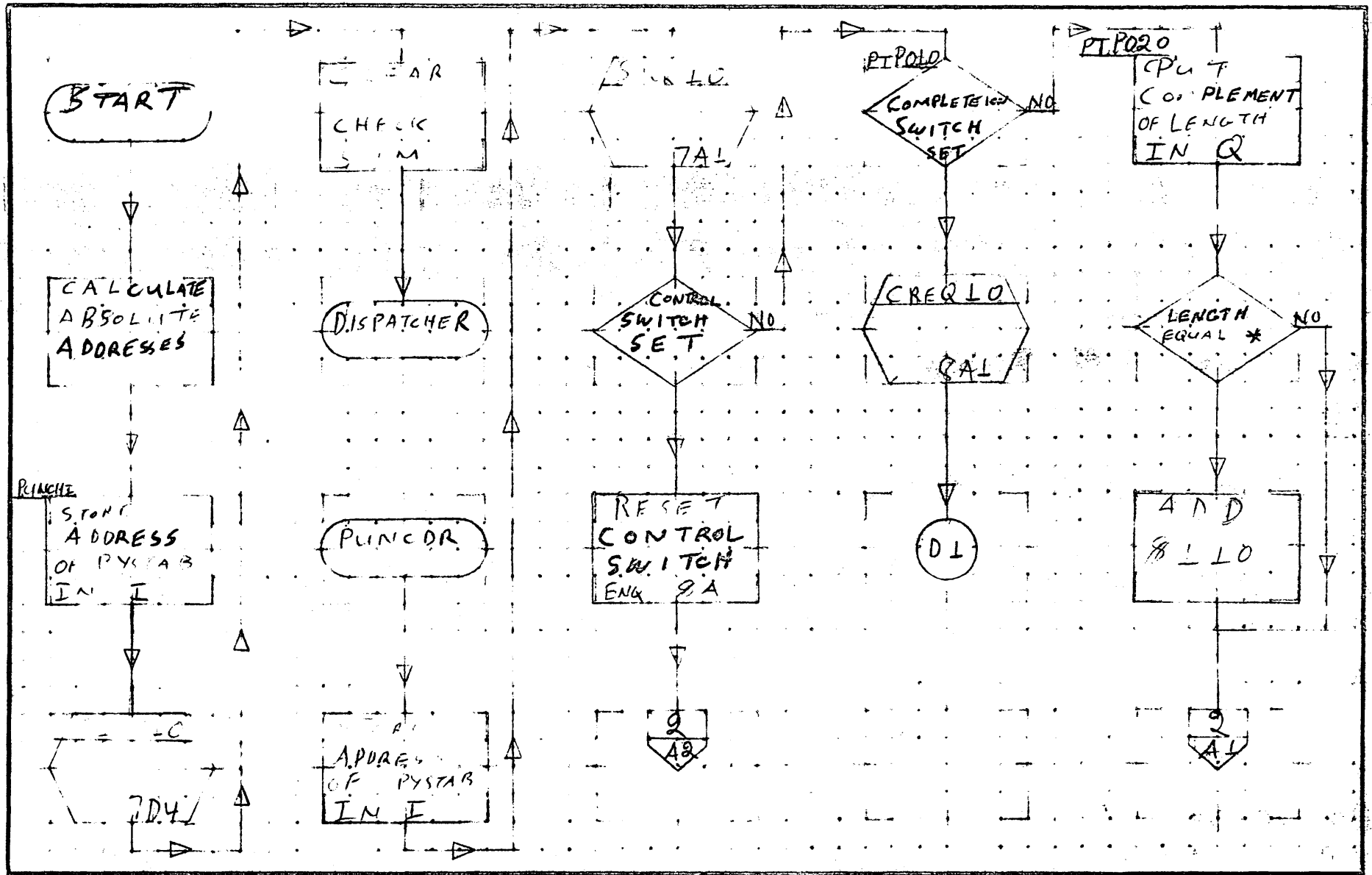
If a carriage return is encountered while punching, a line feed character is punched following the carriage return.

ASCII

Each character punched has it's own parity and it is even. The path that it uses is PUNCDR, PTP010, PTP020, PTP030, PTP070, PTP080, PTP100, PTP105, PTP140, PTP200, and PTP300.

Error Routine

If the Diagnostic Timer times out it will enter this routine PCHERR, which sets alarm status code and goes to STCD10.



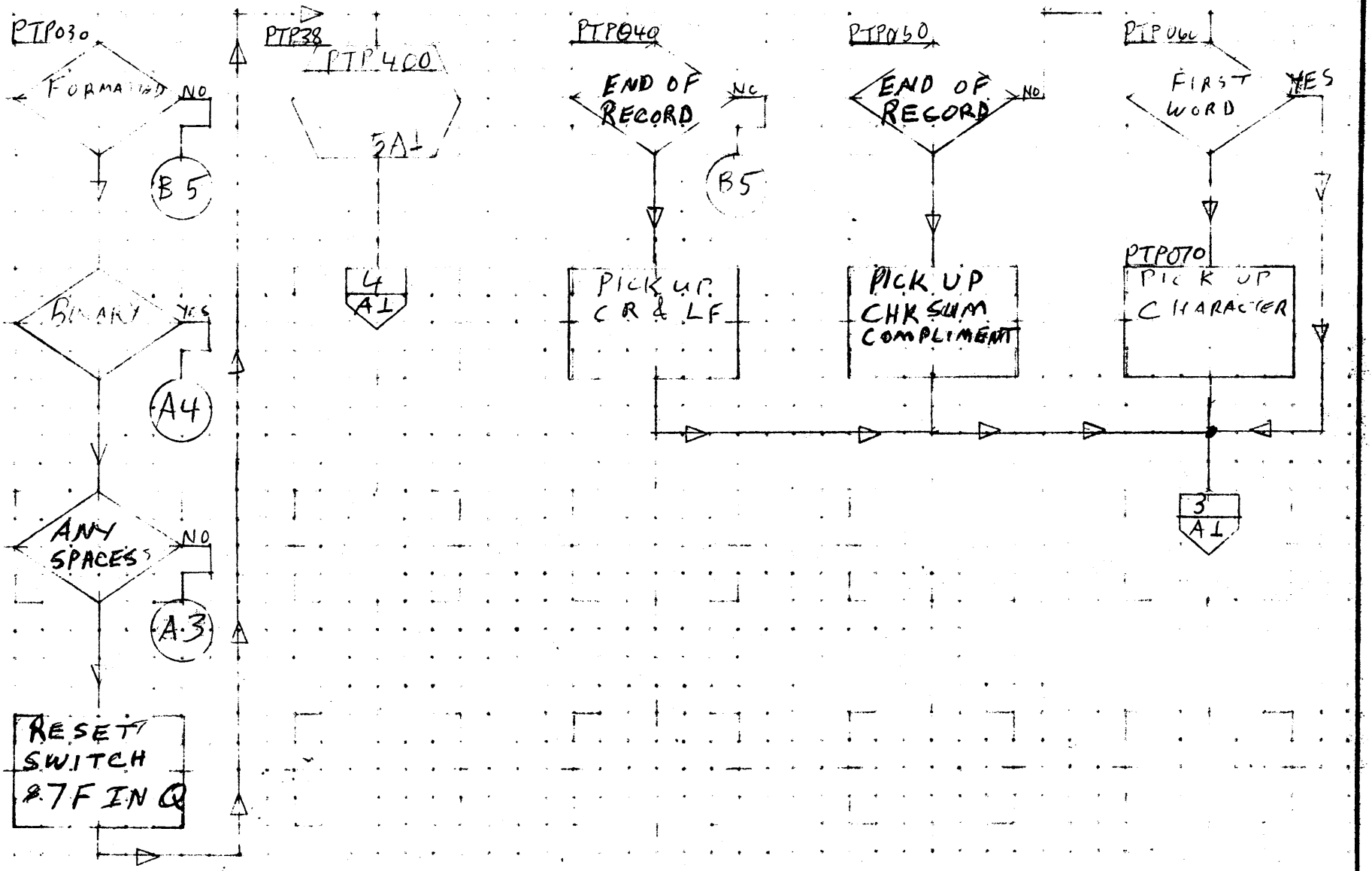
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS 1777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PU. CDR	PROJECT MGR.			
	MASS MEMORY PAGE 1 OF 8	PROJECT NAME PAPER TAPE STATION			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

49.5

AA-385 (FORMER CA127)

PRINTED IN U.S.A.



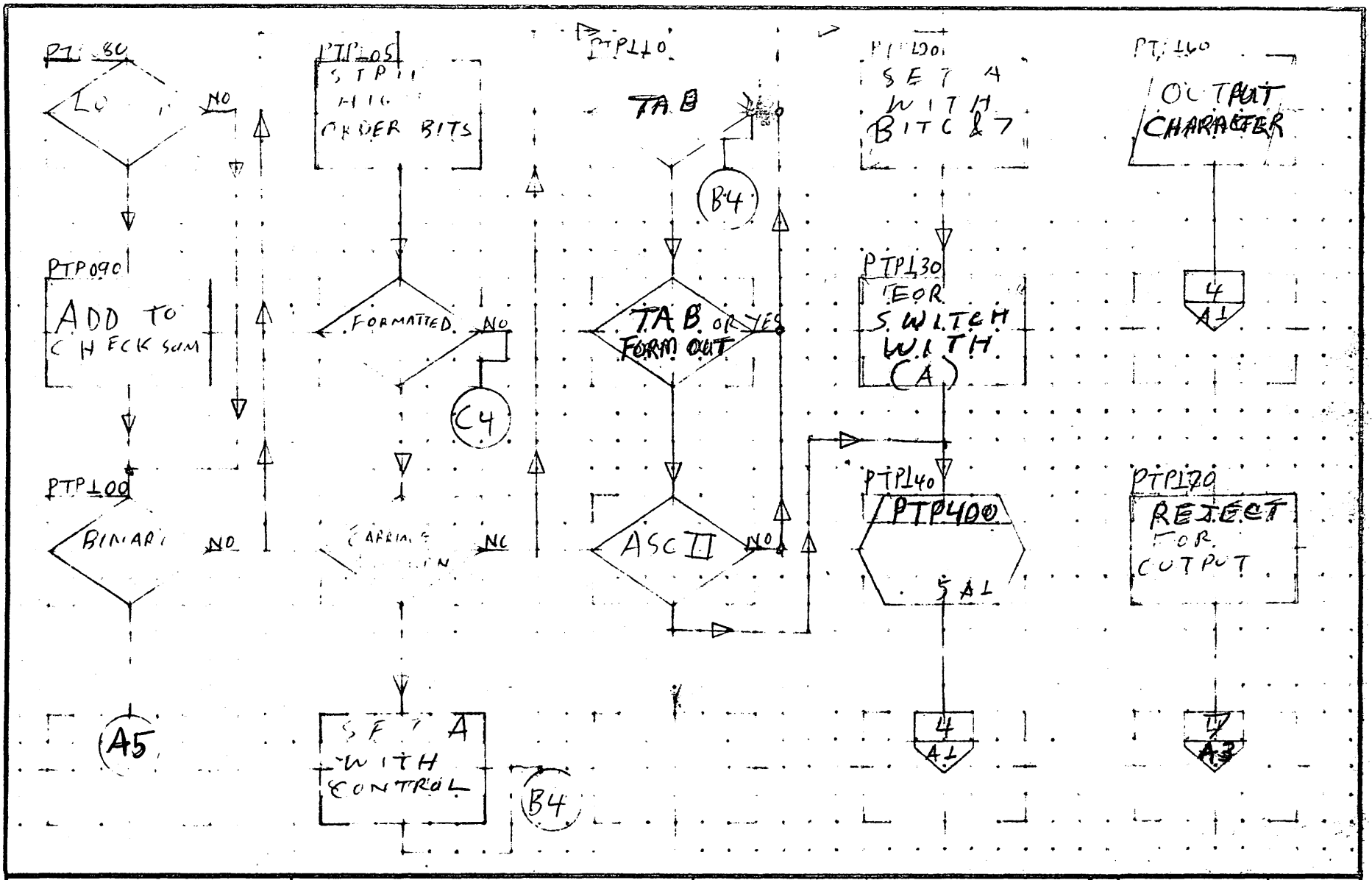
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 2 OF 6	PROJECT NAME			
	NUMBER 2.1. ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

484

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



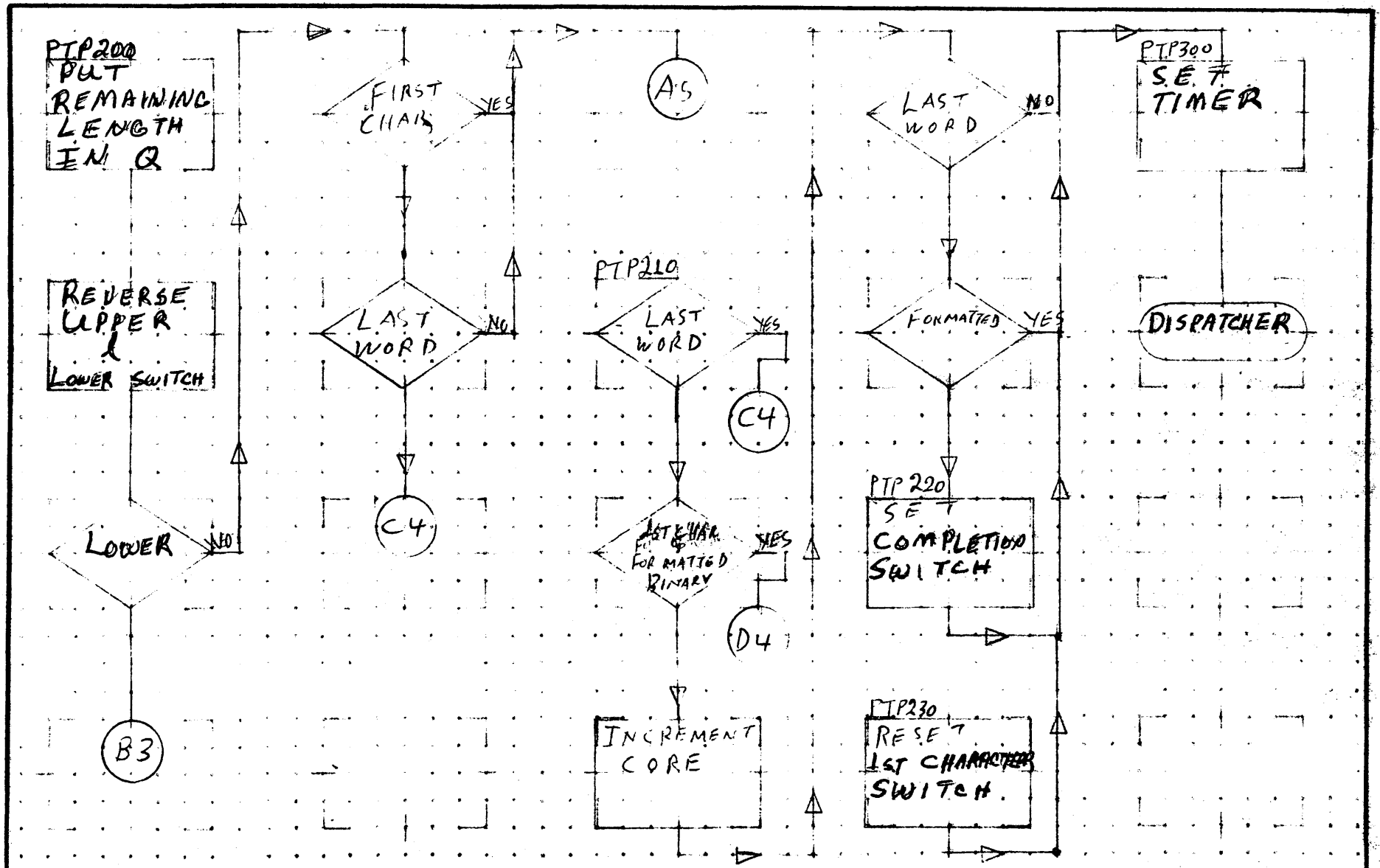
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 3 OF 8	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

48.7

AA1385 (FORMERLY CA127)

PRINTED IN U.S.A.



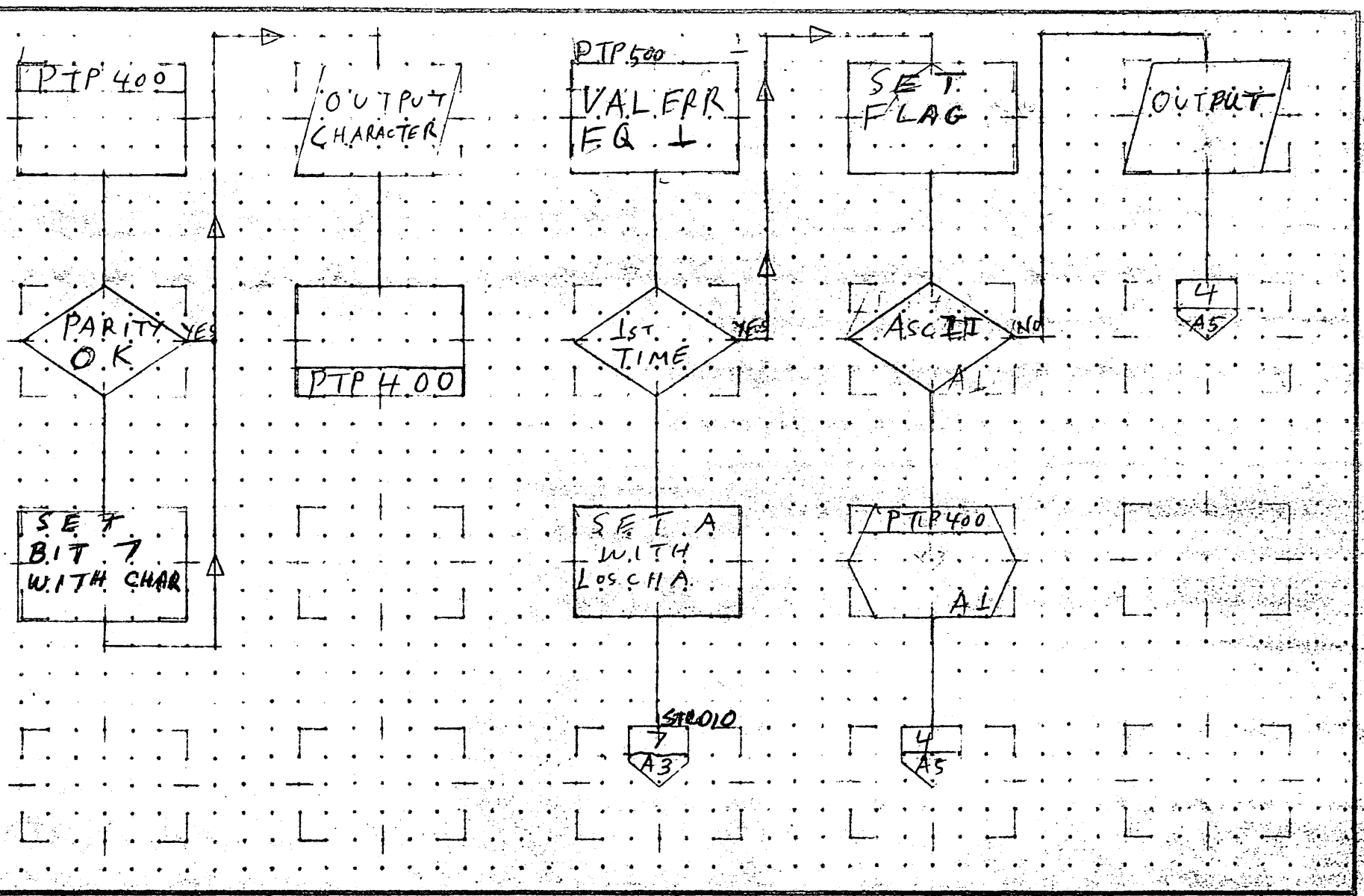
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 4 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

48-8

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

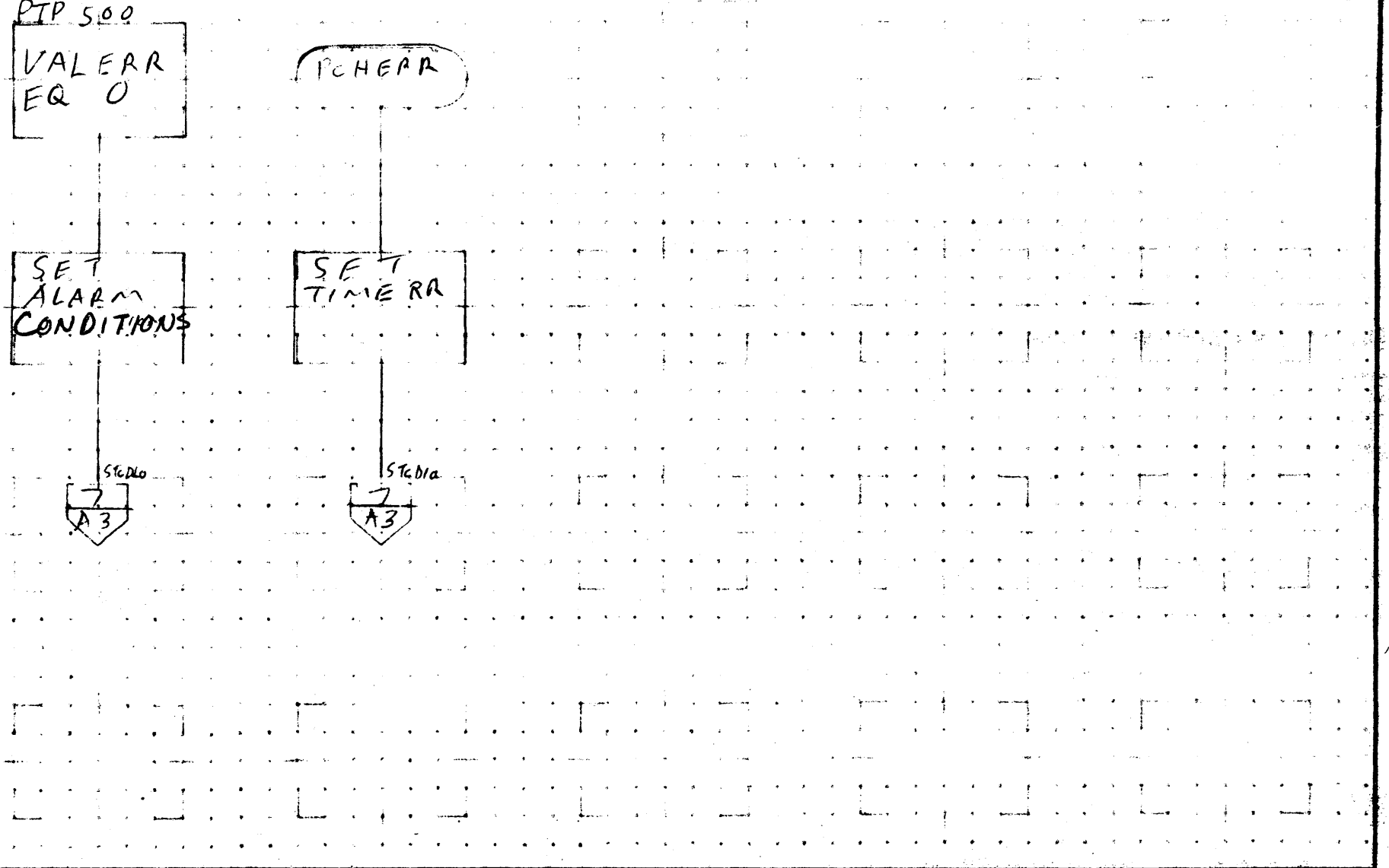


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 5 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



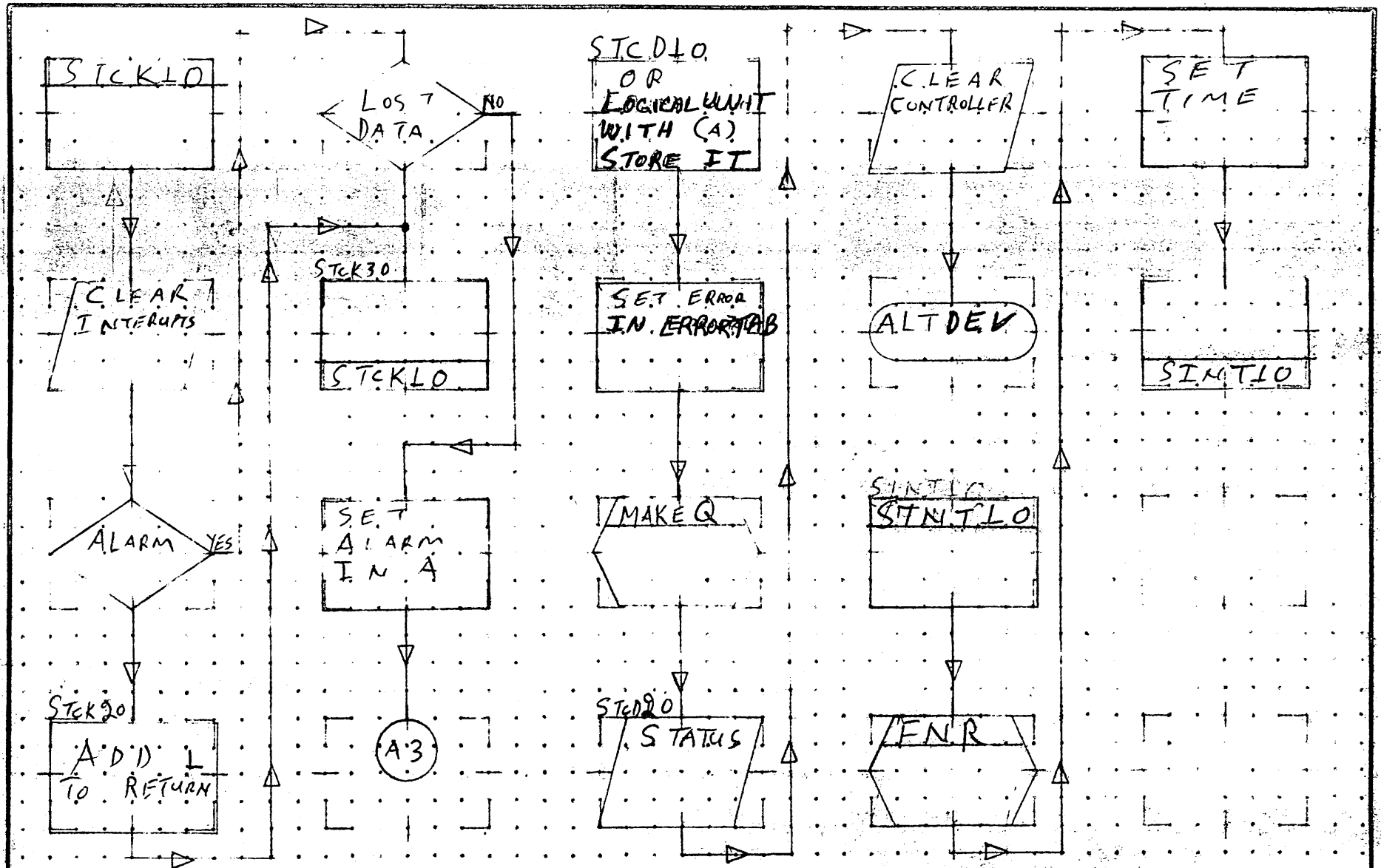
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 6 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

48.10

AA1385 (FORMERLY CA127-11)

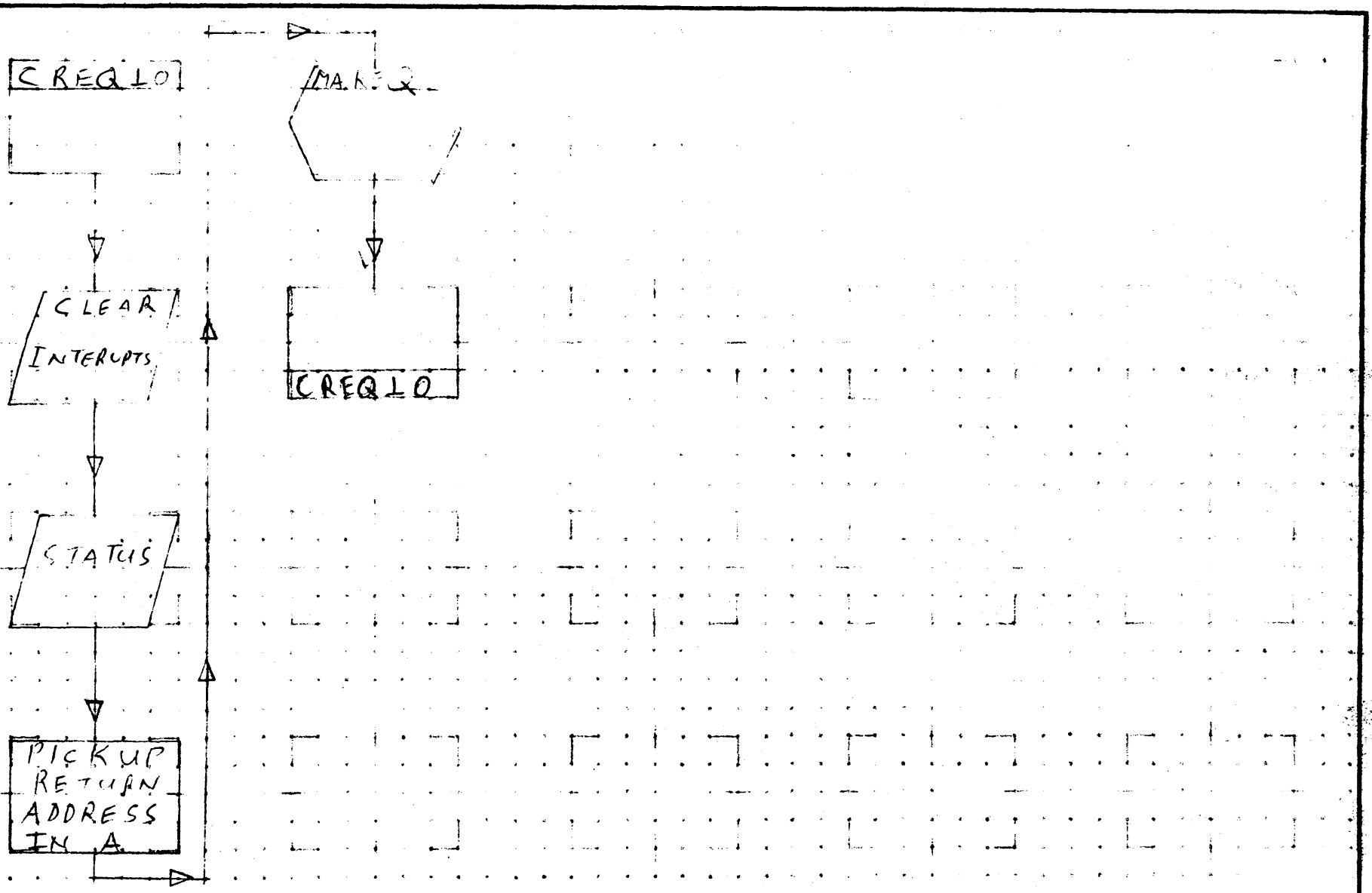
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 177 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 7 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 8 OF 8	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

48.12

AA1385 FORMERLY CA127-11

PRINTED IN U.S.A.

48.4 1777 READER MASS MEMORY

48.4.1 ENTRY SYMBOLS

PREADI	Driver initiator address
PTREAD	Driver continuator address
PTERR	Driver Error Routine for Diagnostic Timer.

48.4.2 EXTERNAL SYMBOLS

MAS300	Entry in MASDRV. It will check if any mass memory drivers are waiting in core.
MAKEQ	Routine that sets the proper error bits in word nine (9) of PHYSTAB to be passed to user.
ALTDEV	Program that processes irrecoverable errors.

48.4.3 PHYSICAL DEVICE TABLE

The Physical Device Table for the Reader Driver Mass Memory is set up as follows:

```
            ENT   PPTRDR
            EXT   MASDRV,TP1777
            ADC   TP1777
PPTRDR      NUM   $12AA          Reader Entry
            ADC   MASDRV         Initiator Entry
            ADC   0              Continuator Entry
            ADC   0              Time Out Entry
            NUM   -1             Diagnostic Clock
            NUM   0              Logical Unit
            NUM   0              Call Parameter List
            NUM   $XXXX          Hardware Address
            NUM   $XXXX          Request Status
            NUM   0              Status Word
            NUM   0              Current Core
            NUM   0              Last Word Address +1
            NUM   0              Hardware Status
            NUM   $FF            Driver Length
            ADC   TF1777         Some associated with sector ⋈
            NUM   0              Check Sum
            NUM   0              Length of Record
            NUM   0              Return for FNR
            NUM   0              Return Address Common
            NUM   0              Error Code Storage
            NUM   0              Temporary Storage
            NUM   0              Flag for Lost Data
```

48.4.4 SUBROUTINES

48.4.4.1 STCK10 {Status Check}

This routine clears the interrupt and checks status to see if an alarm condition occured. If alarm occurred, it checks for lost data, if so, exit; if not, set alarm in A and go to STCD10.

Normal return is at P+2.

48.4.4.2 STCD10 {Set Code}

This routine is entered when there is a non recoverable error. Upon entering A contains the error code. The logical unit is Exclusive Ored into A. This then goes to MAKEQ and then to Alternate Device Handler.

48.4.4.3 SINT10 {Set Interrupts}

This routine goes to find next request. Upon P+2 return, it sets the interrupts, starts motion and exits.

Upon P+1 return it exits to MASDRV because no more requests are threaded.

48.4.4.4 CREQ10 {Clear Interrupt Request}

This routine clears interrupts, input status goes to MAKEQ and exits is taken.

48.4.5 DRIVER DESCRIPTION

Initiator

PREADI is the initiating entry to the driver. At entry exit is made to SINT10. Upon return checksum and length are cleared and exit to the dispatcher is made.

Continuator

PTREAD is the continuator entry. At entry, exit is made to STCK10 upon return at P+2 no errors occurred. If return was at P+1 a jump to the error routine would occur. PTR190 is the tag address.

Data Transfer Operation Reader

Formatted Binary

The compliment of the length is read in on the first two (2) frames of each record. The path that the formatted binary uses is PTREAD, PTR030, PTR040, PTR050, PTR080, PTR090, PTR120, PTR130, PTR180, PTR200, PTR210, and PTR270. Upon completion the checksum is compared and if zero no error.

Binary

Binary is a direct data to core. The path that binary uses is PTREAD, PTR090, PTR120, PTR130, PTR180, PTR240, PTR260, and PTR270. There is no checksum to compare.

Formatted ASCII

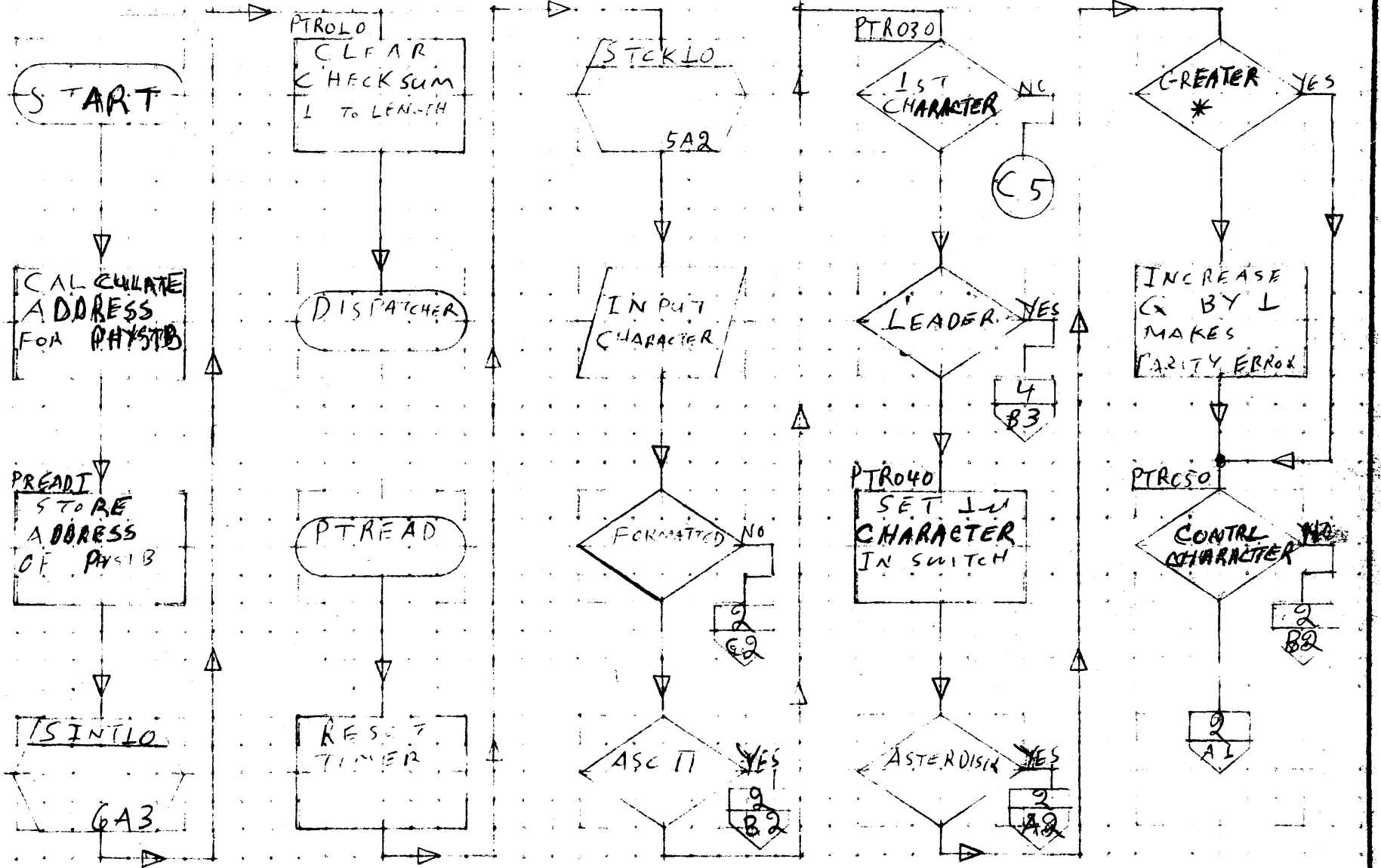
Each character read has it's own parity and it is even parity. The path that formatted ASCII uses is PTREAD, PTR080, PTR090, PTR120, PTR130, PTR135, PTR140, PRT180, PTR200, PTR210 and PTR270. Upon reading a carriage return it exits to PTR290 and completes the request.

ASCII

Each character punched has its own parity and it is even. The path that it uses is PTREAD, PTR090, PTR120, PTR130, PTR135, PTR140, PTR180, PTR240, PTR270.

Error Routine

If the Diagnostic Timer times out it will enter this routine PTRERR, which sets the alarm status code and goes to STCD10.

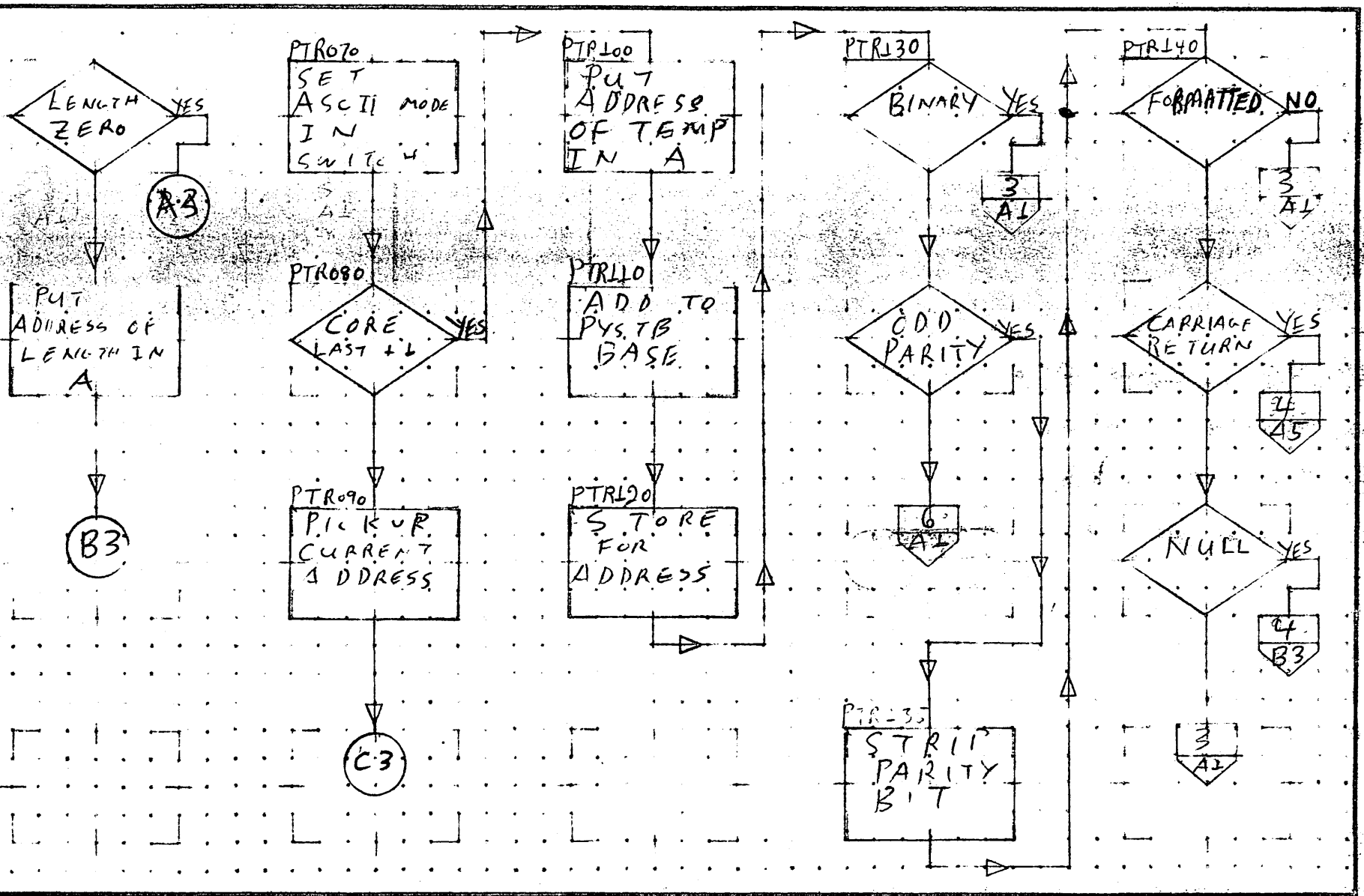


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PTREAD	PROJECT MGR.			
	MASS MEMORY PAGE 1 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

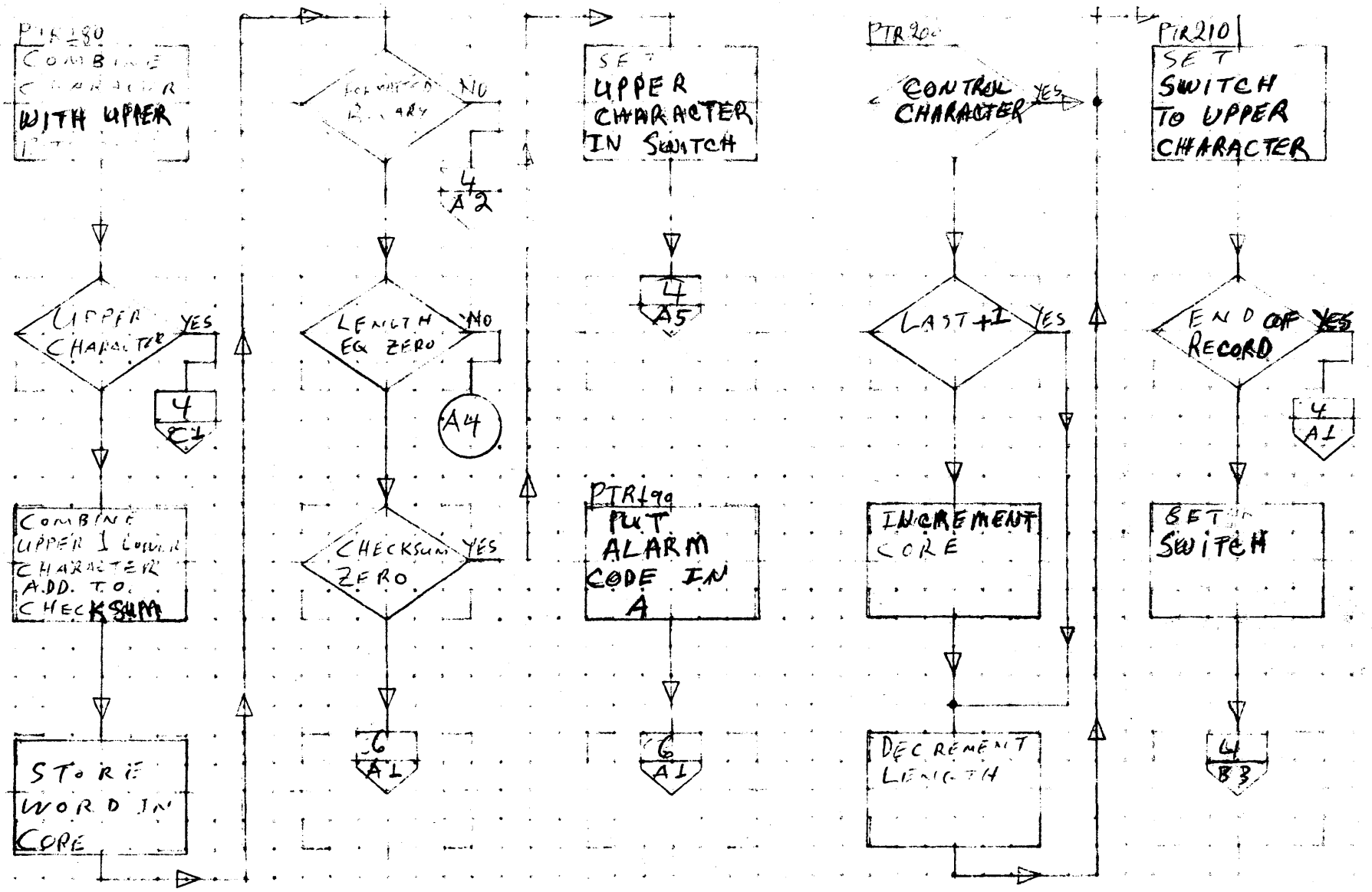
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PTREAD	PROJECT MGR.			
	MASS MEMORY PAGE 2 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

48.17



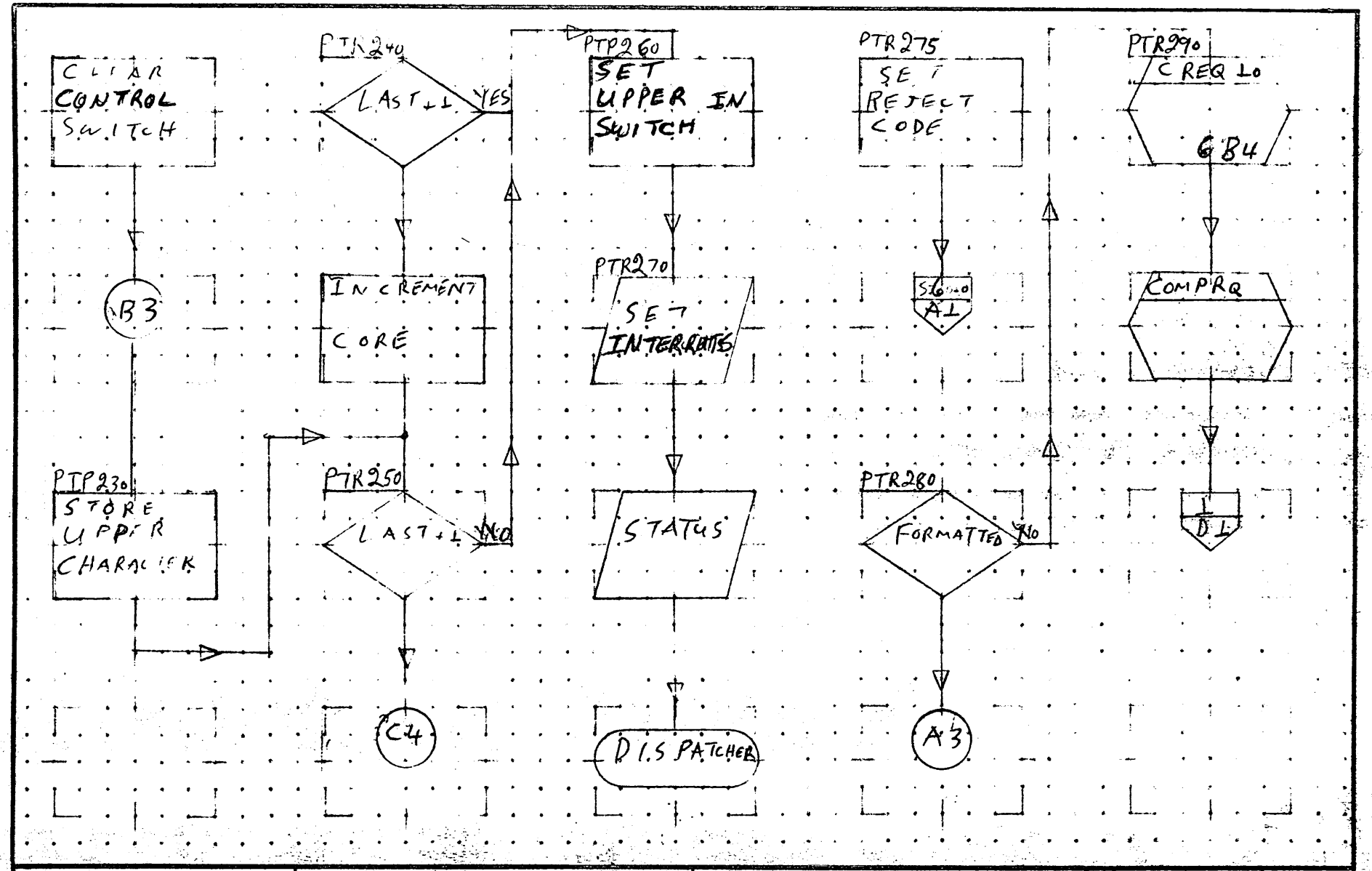
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	PAGE 3 OF 6		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48-18



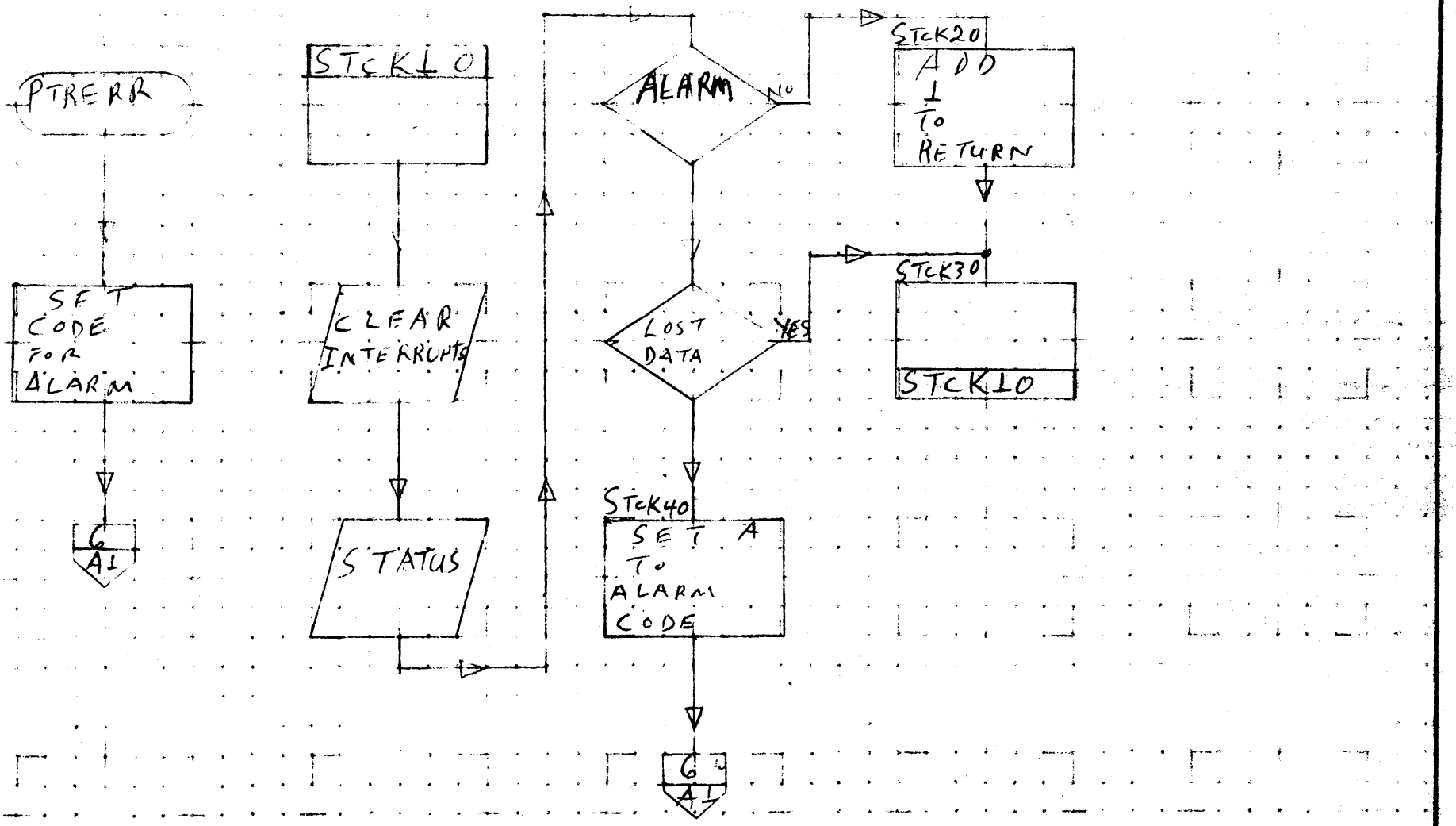
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1712 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 4 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.1.5



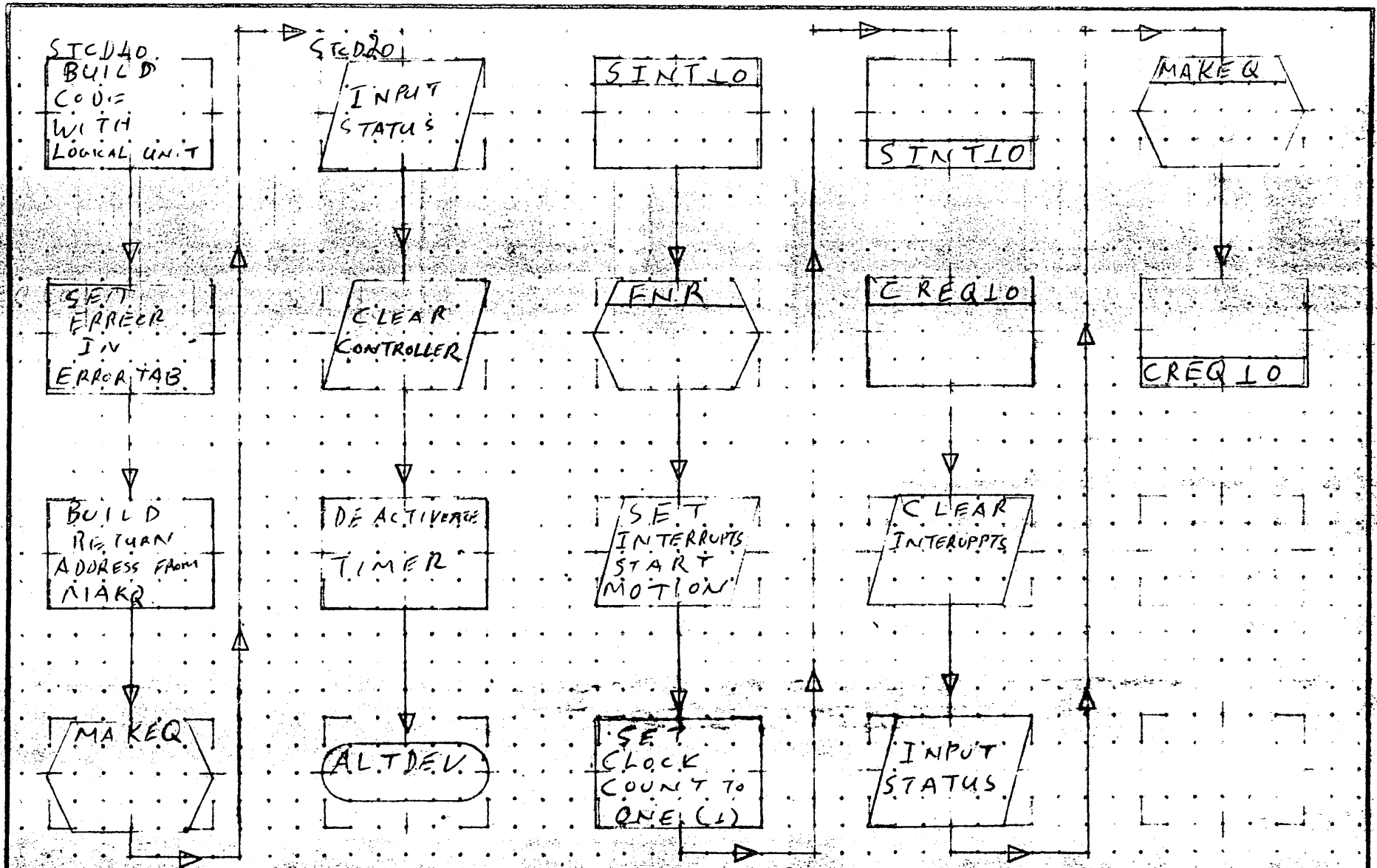
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE P	PROJECT MGR.			
	PAGE 5 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.20



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 READER MACH. TYPE	PROJECT NO.	REV.	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 6 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.21

48.5 1777 PUNCH DRIVER CORE RESIDENT

48.5.1 ENTRY SYMBOLS

PUNCHI	Driver Initiator Address
PUNCDR	Driver continuator address
PCHERR	Driver error routine for diagnostic timer

48.5.2 EXTERNAL SYMBOLS

STCK10	Entry in STCK it checks for alarm.
STCD10	Entry in STCK it sets the error code for alternate device.
SINT10	Entry in STCK it sets interrupts.
CREQ10	Entry in STCK it clears interrupts.

48.5.3 PHYSICAL DEVICE TABLES

The Physical Device Table for the punch driver core resident is set up as follows:

```
          ENT   PPTPCH
          EXT   PUNCHI,PUNCDR,PCHERR
PPTPCH    NUM   $12AA          Punch Entry
          ADC   PUNCHI         Initiator Entry
          ADC   PUNCDR         Continuator Entry
          ADC   PCHERR         Time-out Entry
          NUM   -1             Diagnostic Clock
          NUM   0              Logical Unit
          NUM   0              Call Parameter List
          NUM   $XXXX          Hardware Address
          NUM   $XXXX          Request Status
          NUM   0              Status Word
          NUM   0              Current Core Location
          NUM   0              Last Word Address +1
          NUM   0              Hardware Status
          NUM   0
          ADC   TP1777         Some associated with sector #
          NUM   0              Checksum Word
          NUM   0              Length of Record
          NUM   0              Return Address FNR
          NUM   0              Return Address Common
          NUM   0              Error Code Storage
          NUM   0              Temporary Storage
          NUM   0              Flag for Lost Data
```

48.5.4 DRIVER DESCRIPTION

Initiator

PUNCHI is the initiating entry to the driver. At entry, exit is made to SINT10, upon return the checksum is cleared and an exit to the dispatcher is made.

Continuator

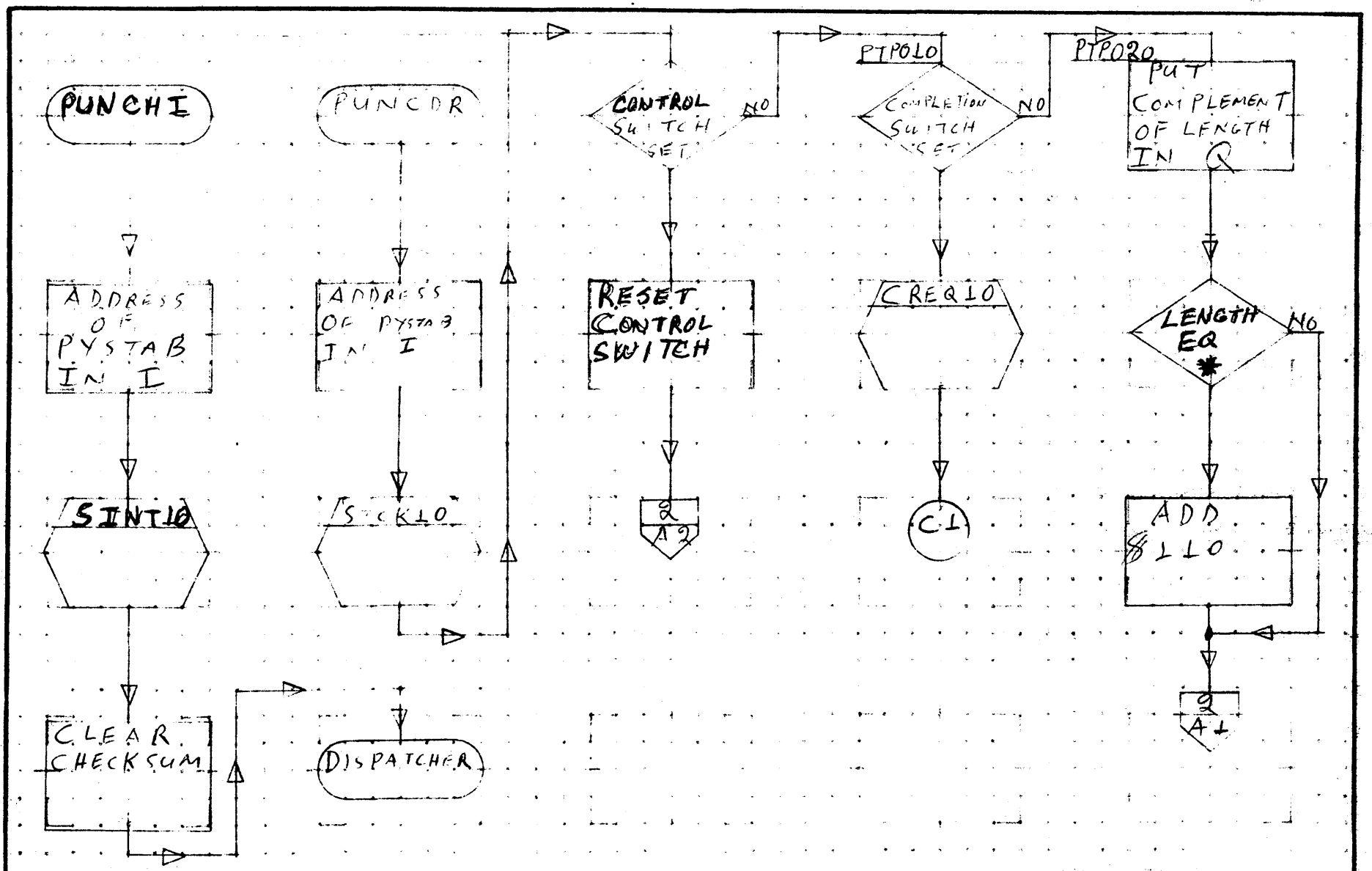
PUNCDR is the continuator entry. At entry, exit is made to STCK10 upon normal return P+2 no errors occurred. If return was at P+1 a jump to the validation error routine would occur. PTP500 is the tag address.

Data Transfer Operation Punch

See Section 48.3.5 Data Transfer Operation Punch.

Error Routine

If the diagnostic timer times out it will enter this routine PCHERR, which sets the alarm status code and goes to STCD10.



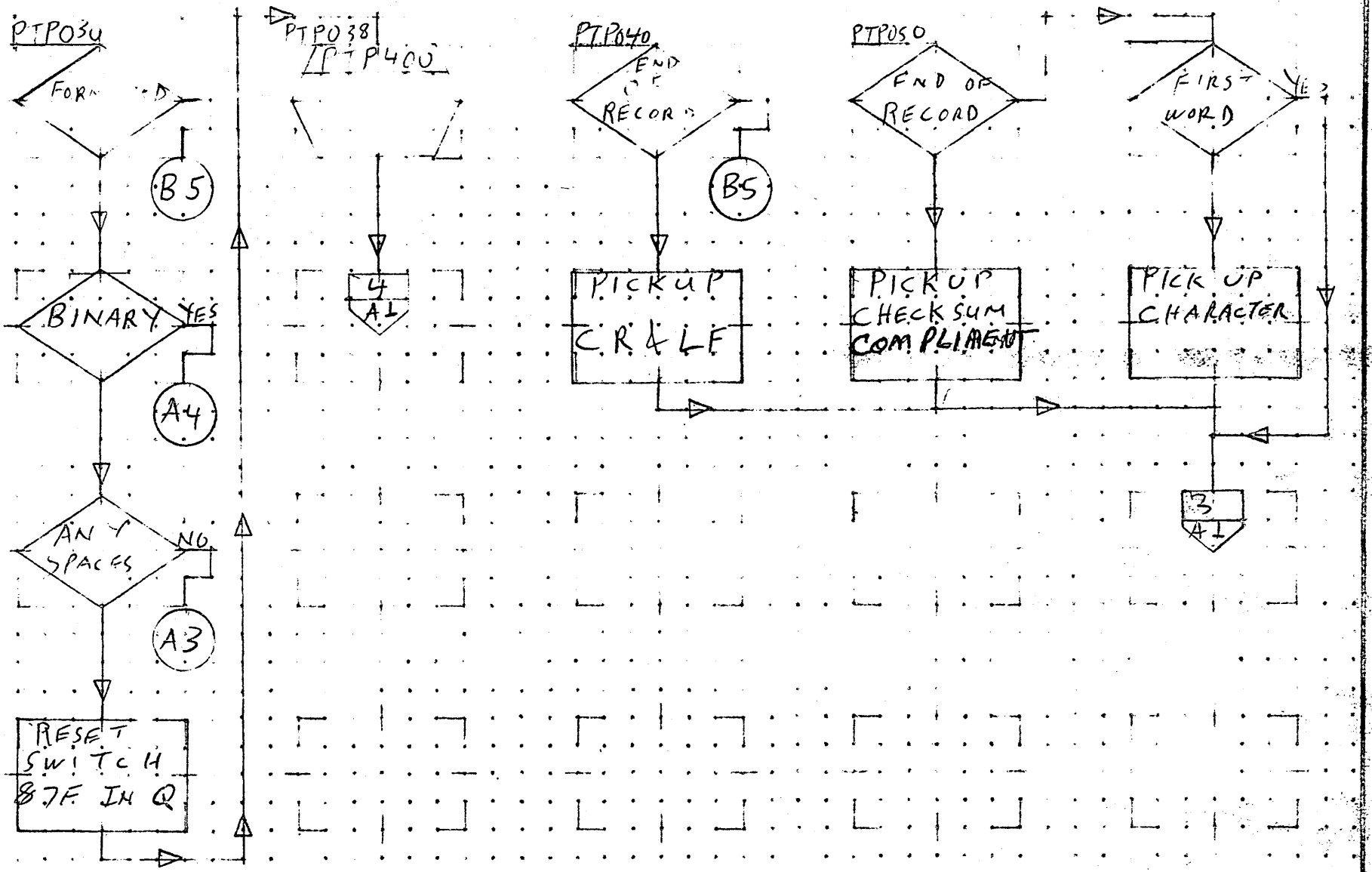
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PUNCDR	PROJECT MGR.			
	CORE RESIDENT PAGE 1 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.24



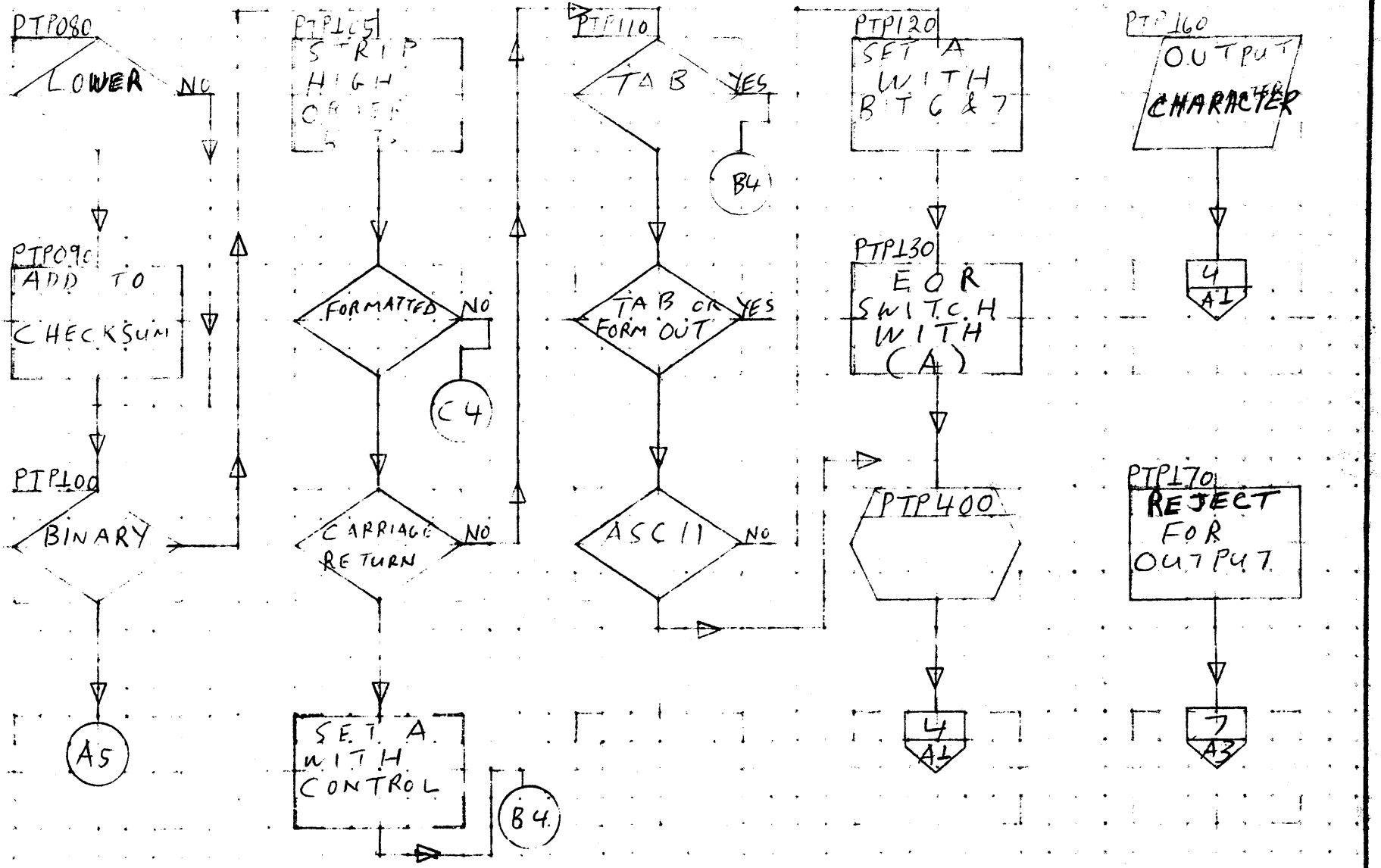
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / / FUNCT MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 2 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

40-25

4A-385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



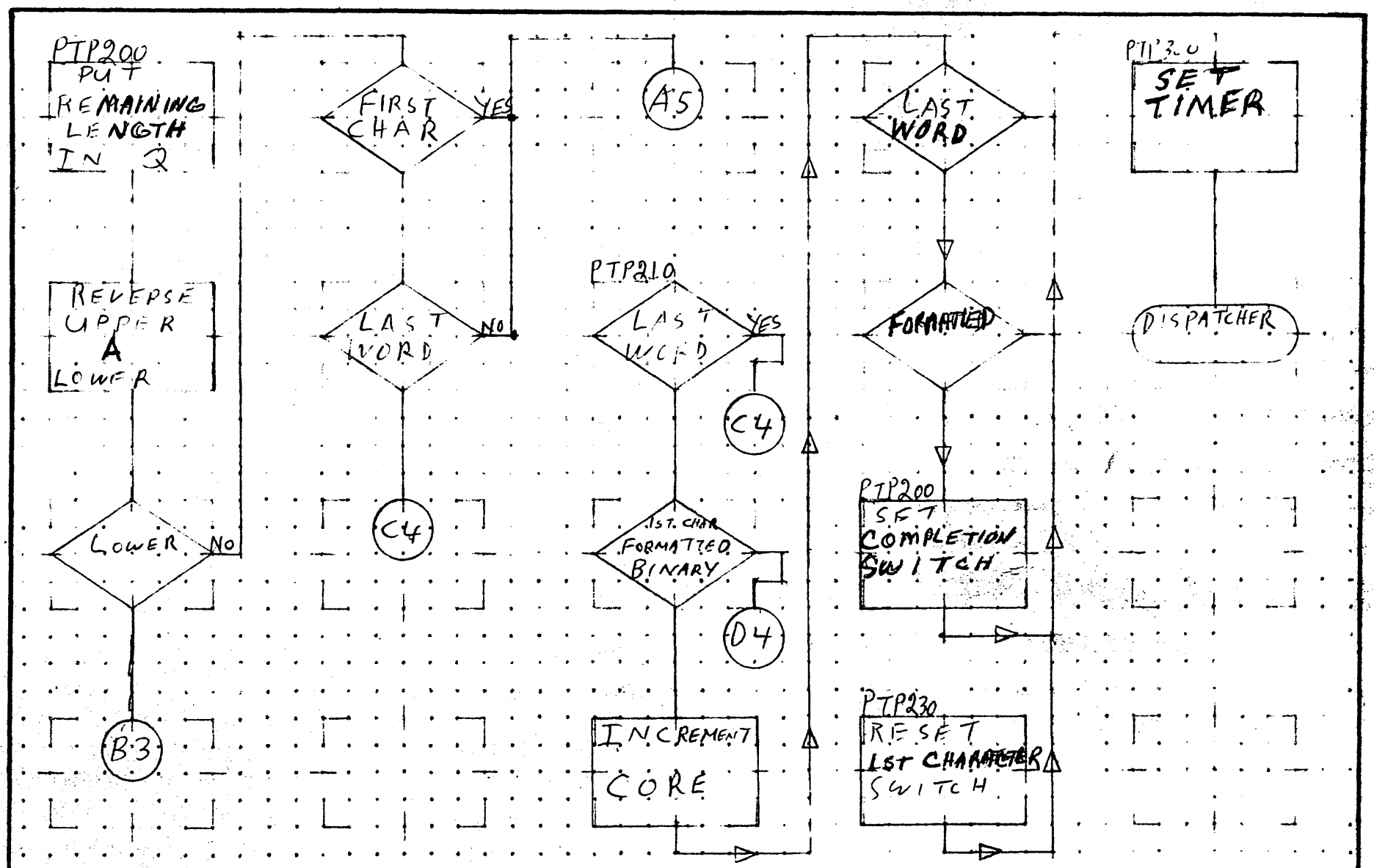
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: 7 7 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 3 OF 6	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.26



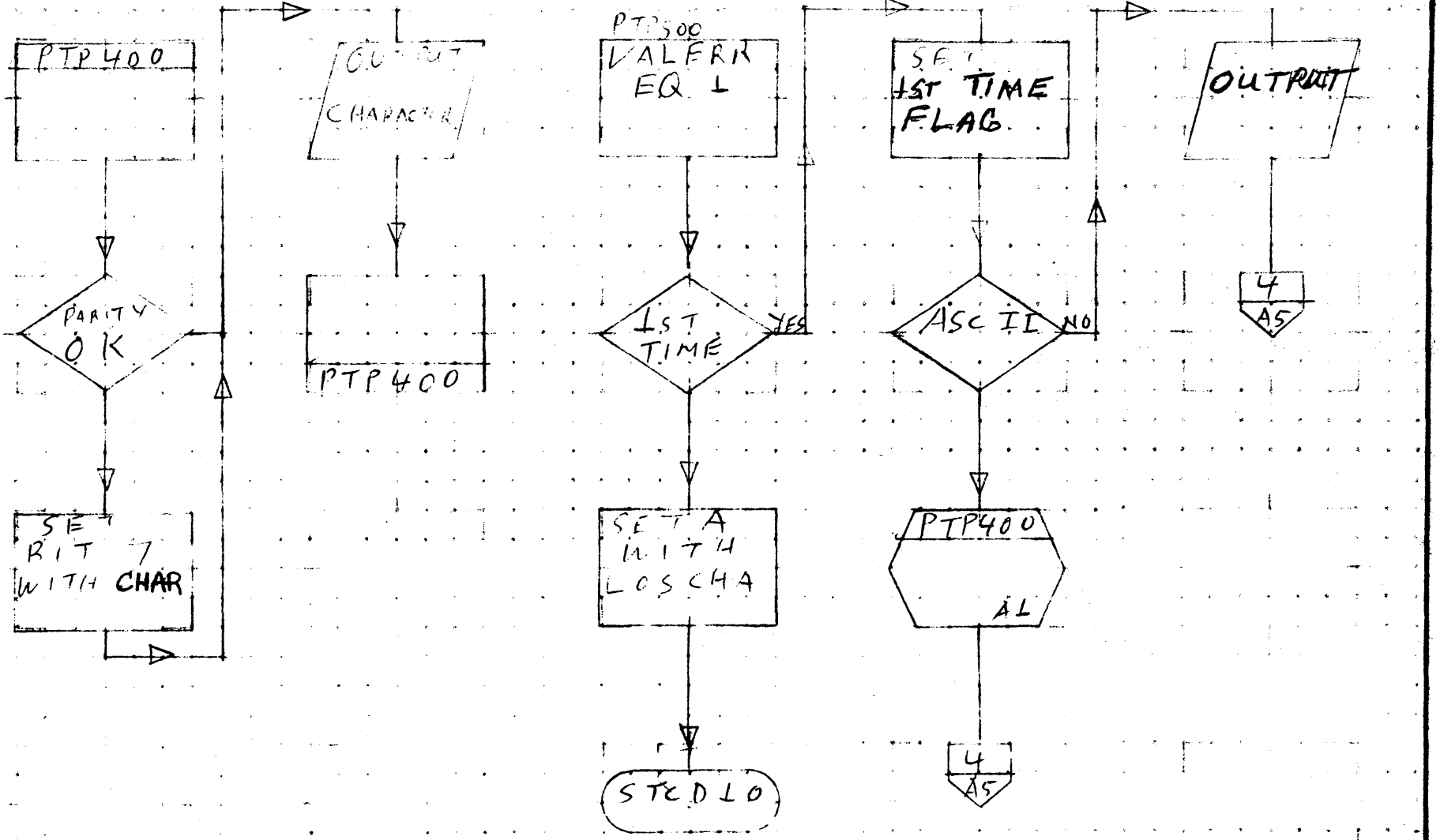
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR			
	PAGE 4 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.27



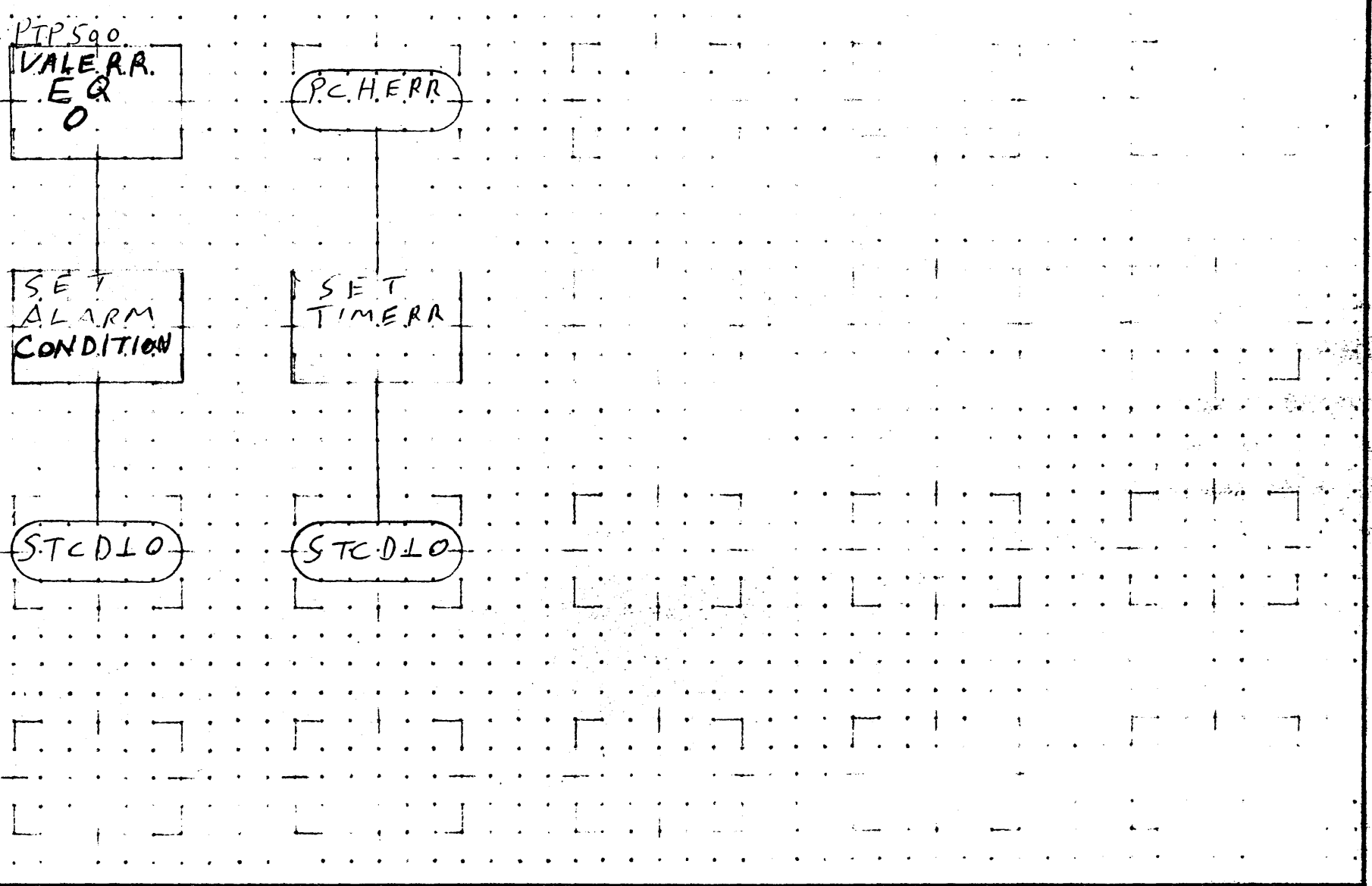
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 5 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48*28



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 777 PUNCH MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 6 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.29

48.6 1777 READER CORE RESIDENT

48.6.1 ENTRY SYMBOLS

PREADI	Driver initiator address
PTREAD	Driver continuator address
PTRERR	Driver error routine for diagnostic timer.

48.6.2 EXTERNAL SYMBOLS

STCK10	Entry in STCK it checks for alarm.
STCD10	Entry in STCK it sets the error code for alternate device.
SINT10	Entry in STCK it sets interrupts.
CREQ10	Entry in STCK it clears interrupts.

48.6.3 PHYSICAL DEVICE TABLE

The Physical Device Table for the reader driver core resident is set up as follows:

```
          ENT   PPTRDR
          EXT   PREADI,PTREAD,PTRERR
PPTRDR    NUM   $12AA          Reader Entry
          ADC   PREADI         Initiator Entry
          ADC   PTREAD         Continuator Entry
          ADC   PTRERR         Time Out Entry
          NUM   -1             Diagnostic Clock
          NUM   0              Logical Unit
          NUM   0              Call Parameter List
          NUM   $XXXX          Hardware Address
          NUM   $XXXX          Request Status
          NUM   0              Status Word
          NUM   0              Current Core
          NUM   0              Last Word Address +1
          NUM   0              Hardware Status
          NUM   0
          ADC   TF1777         Some associated with Sector #
          NUM   0              Checksum
          NUM   0              Length of Record
          NUM   0              Return for FNR
          NUM   0              Return Address Common
          NUM   0              Error Code Storage
          NUM   0              Temporary Storage
          NUM   0              Flag for Lost Data
```

48.6.4 DRIVER DESCRIPTION

Initiator

PREADI is the initiating entry to the driver. At entry exit is made to SINT10. Upon return checksum and length are cleared and exit to the dispatcher is made.

Continuator

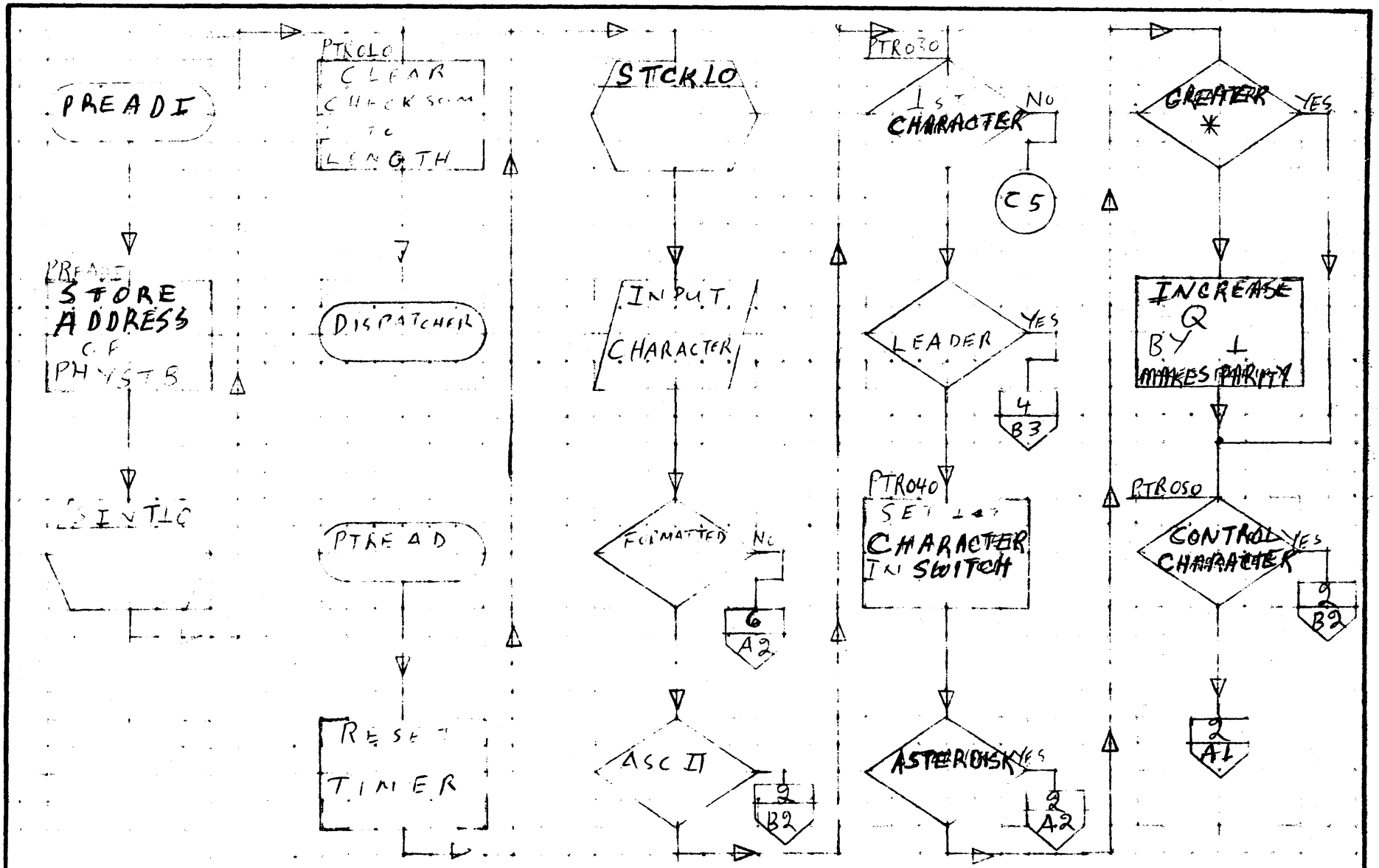
PTREAD is the continuator entry. At entry exit is made to STCK10 upon return at P+2 no errors occur. If return was at P+1 a jump to the error routine would occur. PTR190 is the tag address.

Data Transfer Operation Reader

See Section 48.4.5 Data Transfer Operation Reader

Error Routine

If the Diagnostic Timer times out it will enter this routine PTRERR, which sets alarm status code and goes to STCD10.



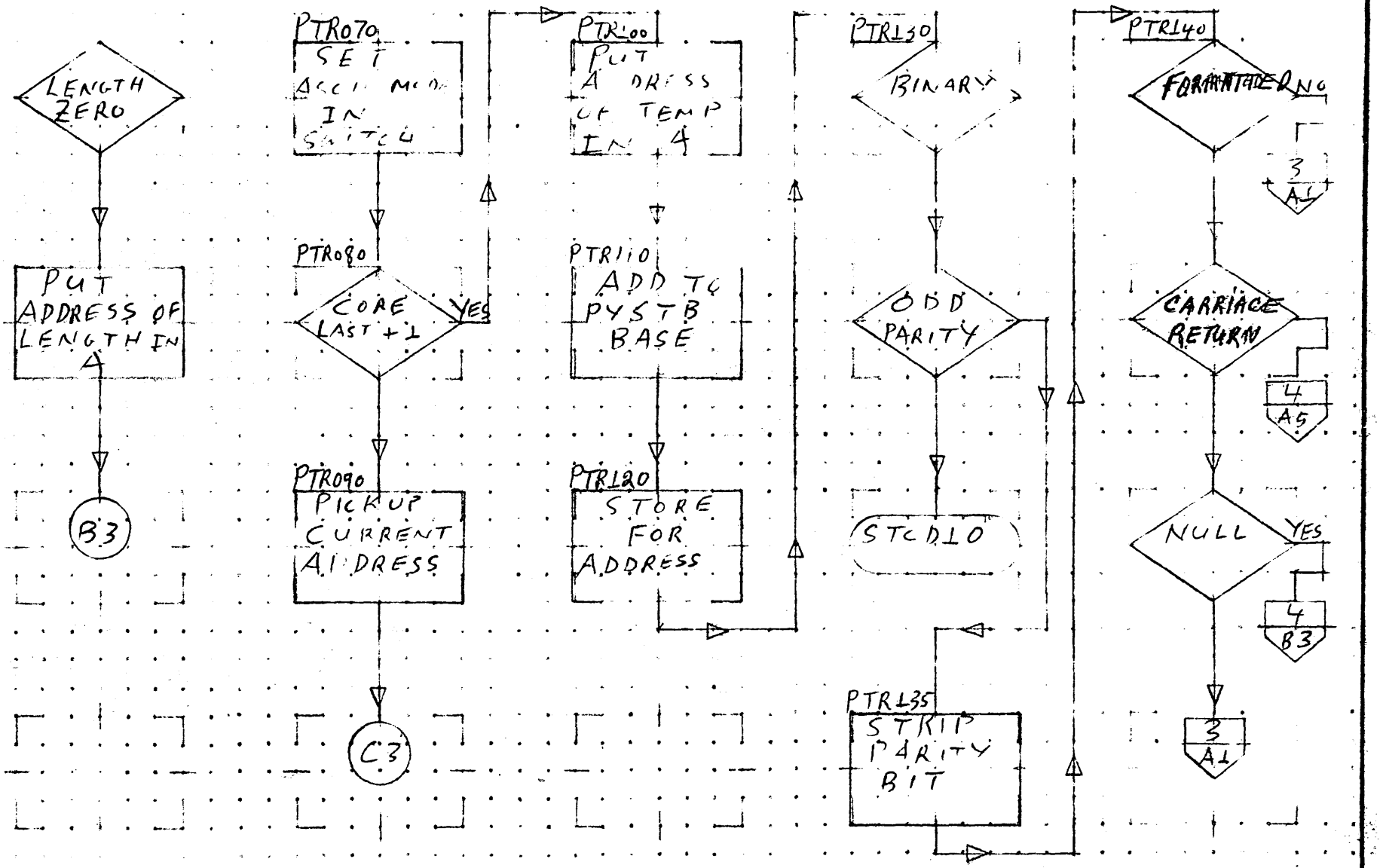
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 777 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PTREAD	PROJECT MGR			
	CORE RESIDENT PAGE 1 OF 5	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 9 1971

48-32



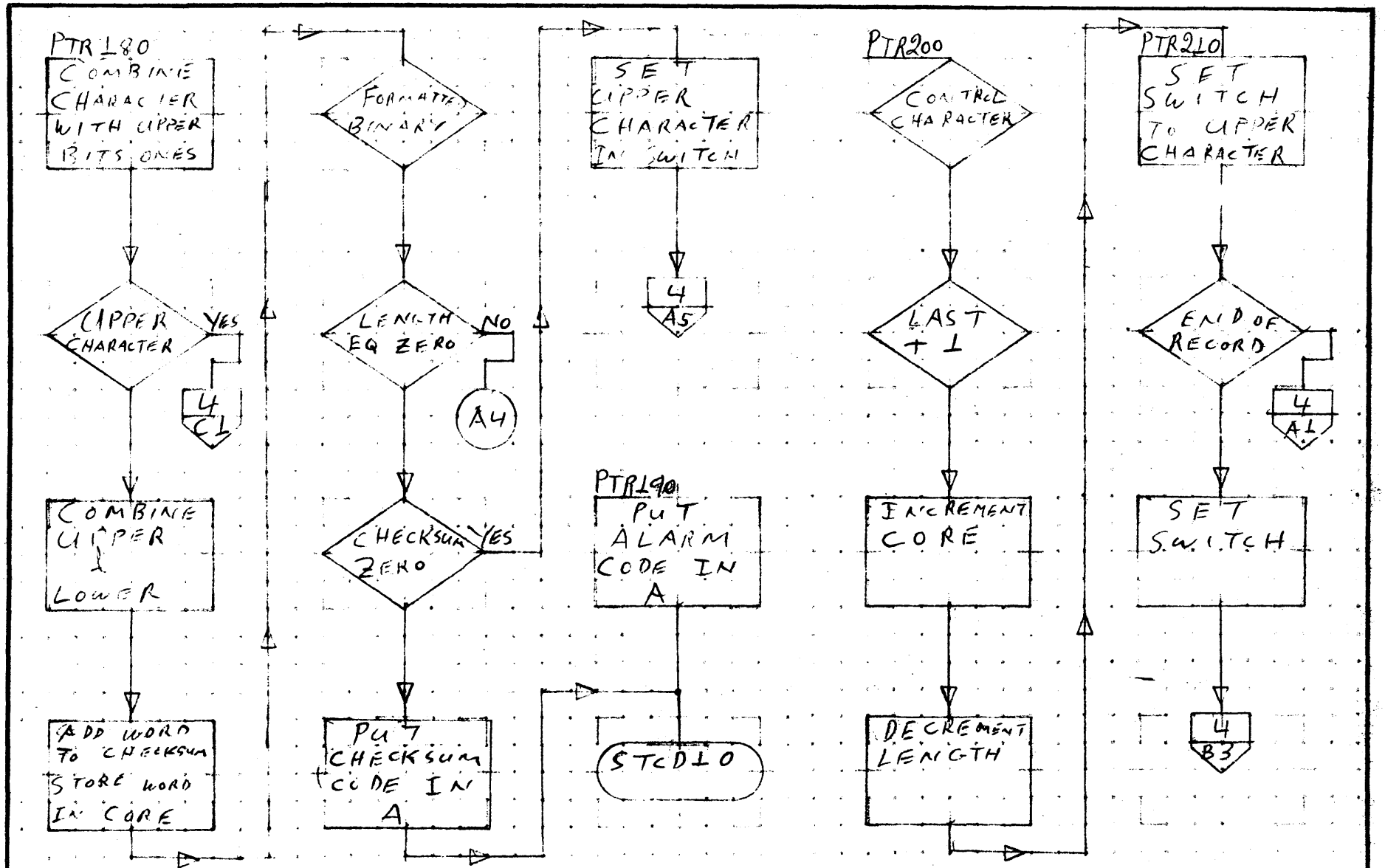
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	CORE RESIDENT PAGE 2 OF 5	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.33.



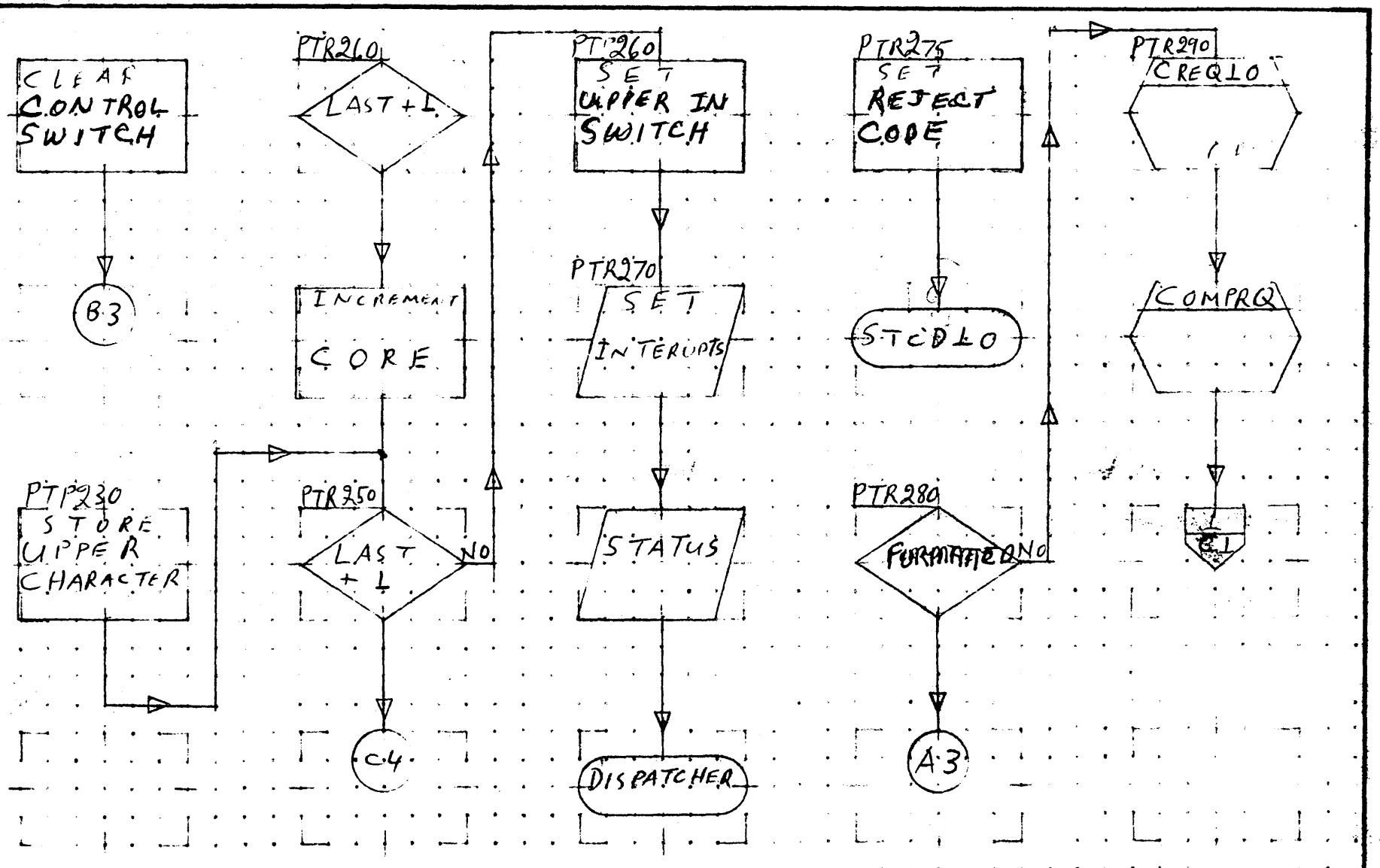
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	CORE RESIDENT PAGE 3 OF 5	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

hb-34

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



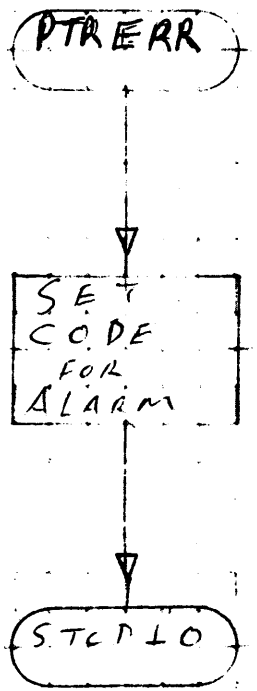
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1711 READER MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 4 OF 5	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

48.35



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1777 READER MACH. TYPE	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 5 OF 5	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

48.7 SUBPROGRAM {STCK}

48.7.1 STCK10 {Status Check}

Clears the interrupt and checks status to see if alarm condition occured. If alarm occured it checks for lost data, if so exit, if not set alarm in A and go to STCD10.

Normal return is P+2

48.7.2 STCD10 {Set Code}

This routine is entered when there is a non recoverable error. Upon entering A contains error code. The logical unit is Exclusive Ored into A. This then goes to MAKEQ then to Alternate Device Handler.

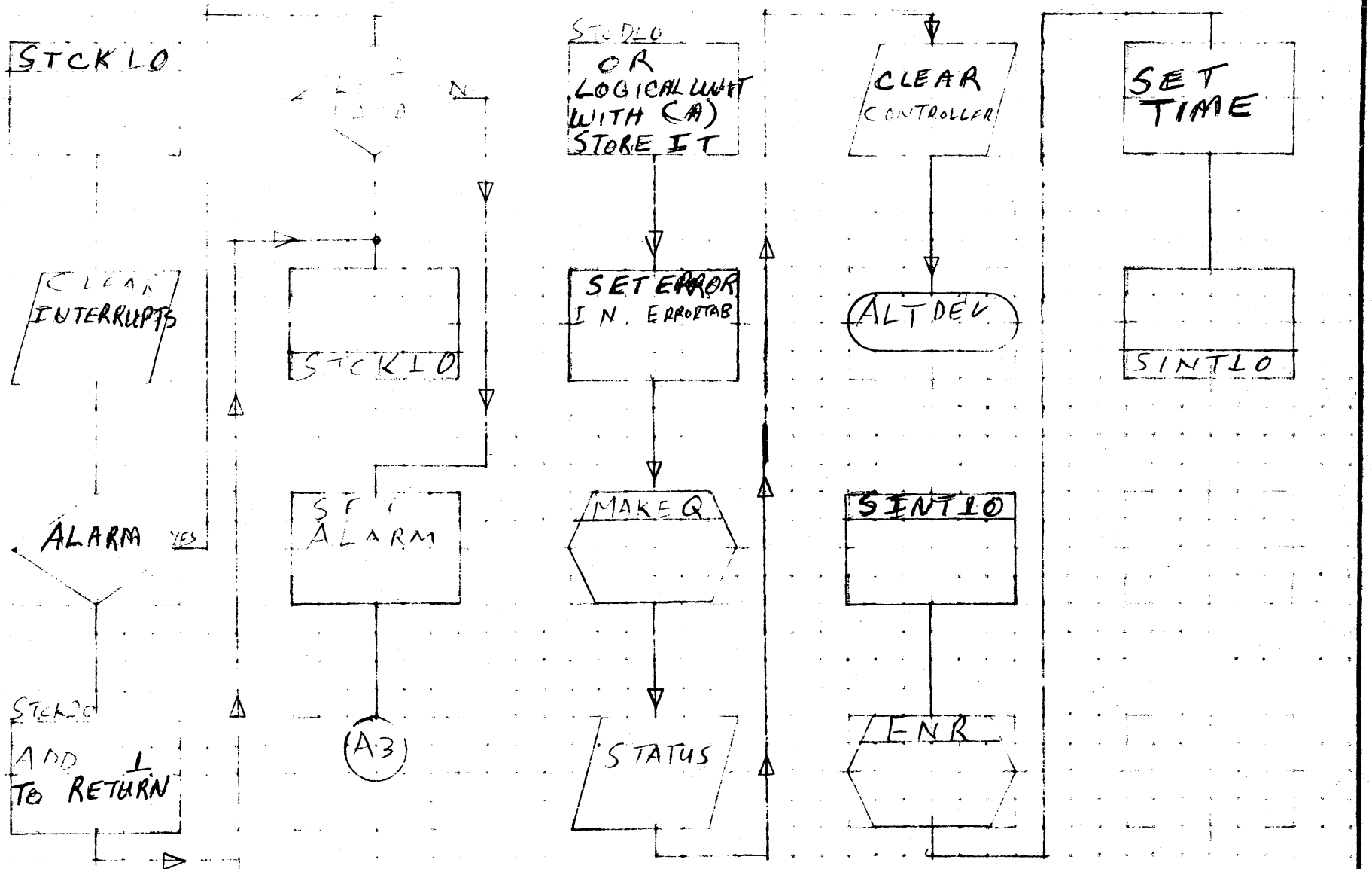
48.7.3 SINT10 {Set Interrupts}

This routine goes to find next request. Upon P+2 return it sets the interrupts and starts motion then exits.

Upon P+1 return, it exits to the dispatcher because no more requests are threaded.

48.7.4 CREQ10 {Clear Interrupt Request}

This routine clears interrupts, inputs status, goes to MAKEQ and exits.



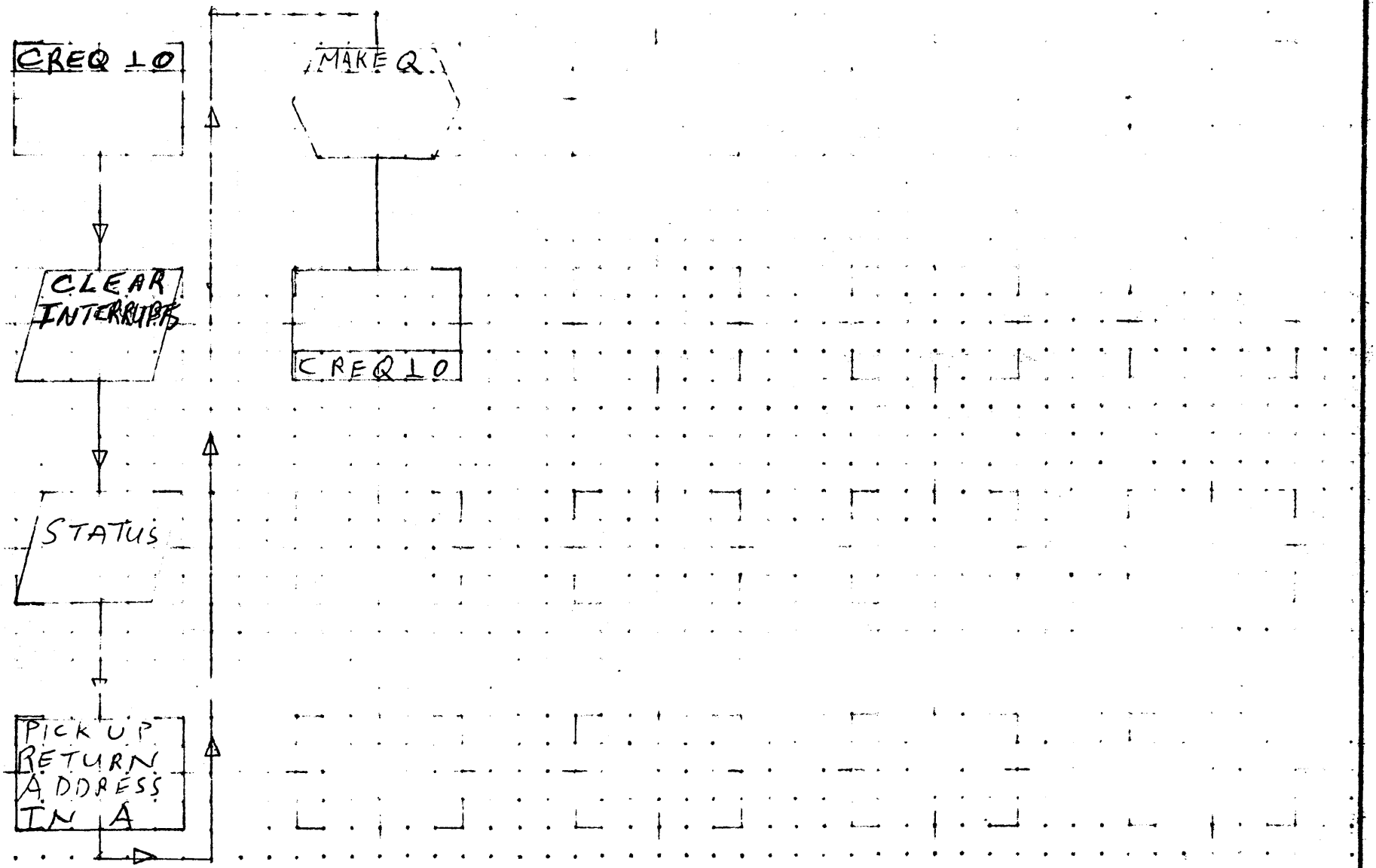
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS STC MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE COMMON SUBROUTINE	PROJECT MGR.			
	PAGE 1 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

48.38

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS STCK MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

48.39

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

49.0 1726/405 CARD READER DRIVER

49.1 FUNCTION

The 1726/405 Card Reader Driver is a standard software driver which will operate under the 1700 MSOS 3.0 Operating System. This driver cannot perform read operations on any other card reader.

49.2 ENTRY POINTS

IN1726	initiator entry
CN1726	continuator entry
EX1726	error entry

49.3 EXTERNALS

MAKEQ	
ALTDEV	
MAS300	declared external if the driver is mass memory resident.
BUFALC	declared if the driver is buffered

49.4 GENERAL PROGRAM INFORMATION

49.4.1 ASSEMBLY OPTIONS

The 1726/405 Card Reader Driver has two assembly option control cards, which control six assembled versions of the 1726/405 Driver.

There are three optional versions of converting Hollerith to ASCII. The following is a description of the differences that exist in the three versions.

{1} ASCII63

The 1726 Controller has a hardware conversion of the American Standard Code for Information Interchange as it was determined to be in 1963. The hardware conversion is designed to recognize a separater card 6-7-8-9 punch, because of this the normal EOF card 7-8 punch {word 14 of the PHYSTAB}, will not be recognized.

{2} ASCII68

This version employs the standard software algorithim that is used in other drivers to convert Hollerith to ASCII. The only difference is the table of hexidecimal codes. This table reflects the revisions in the American Standard Code for Information Interchange as revised in 1968. If the software conversion is used, the end of file pattern is set in word 14 of the PHYSTAB. It can be set to anyone column configuration the user desires.

{3} CDC SUBSET

The CDC Subset is the same as ASCII68 with the following exceptions.

A. The 11-8-2 punch will be interpreted as a hexidecimal 7D.

B. The 12-8-2 punch will be interpreted as a hexidecimal 7B.

C. The 11-0 punch will be interpreted the same as the 11-8-2 punch.

D. The 12-0 punch will be interpreted the same as the 12-8-2 punch.

The CDC Subset option enables the information read on the 1726/405 Card Reader to be compatible with the CDC 3000 Series Computers and 6000 Series, which use CDC ASCII.

The 1726/405 Driver has an assembly option for two mode of operation, buffered and non-buffered. Listed below is a general description of each mode.

{1} BUFFERED

If the buffered version is selected, the first word of the card is read direct. The reason for this is that the binary card status comes up after the first word of a card has been transferred into the computers memory. This status is used to help determine the length of the buffered transfer. An exit is made from the driver after the buffered transfer has started. Control is returned to the driver when an

(2) NON-BUFFERED

If the non-buffered option is selected, the information on one punched card is read and processed before an exit is made from the driver. The reason for this is that when the data status is up it remains up until the entire card is read from the controllers buffer memory. This feature allows the driver to run at a low priority, for there is no longer any danger of losing data.

49.5 DESCRIPTION

At the entry point EX1726, time expired, error code zero is set and the Alternate Device Handler is entered.

At the initiator entry, routine FNR is entered to see if a request is stacked. If not, exit is made to the dispatcher. If there is a request, the checksum accumulator, packing cycle indicator, hollerith error flag, and sequence check words are cleared.

At the label FEEDCD, the relative address of label NEXT from label NOTALR is stored in the subroutine return address word (RETURN). Status is taken and a test is made to see if the card reader is ready. If it is not ready, a jump is made to label ALA. If the card reader is ready, a request for interrupt on Data and Alarm is made. The Diagnostic timer is set and an exit is made to the Dispatcher.

At the entry point, CN1726, status is taken and a test is made to check for an alarm. If there is no alarm, jump to label NOTALR. If there is an alarm, a test is made to check for the following conditions.

a. MANUAL SWITCH OR POWER OFF
b. STACKER FULL OR JAM
c. FAIL TO FEED
d. SEPARATOR CARD
e. ERROR (PRE-READ OR COMPARE)
f. PROTECTED

If none of these conditions exist, a jump is made to label NOTALR. If one or more of these conditions are set, a test is made to see if there has been a Pre-Read or Compare error. If there was a Pre-Read or Compare error, the error code COMPAR (3) is entered in the "A" register and a jump to label ERROR is made otherwise jump to label ALA.

Label ALA is the entry point to a routine that handles alarm conditions. There are two versions, one is selected at assembly time with an equate card. If EQU BUFER (0) the non-buffered version is assembled. When entry is made at label ALA, error code ALARM (2) is inserted in the "A" register and a jump is made to label ERROR. If EQU BUFER (1) the buffered version is assembled, a test is made to see if the 1706 is not busy. If it is not busy, the error code ALARM (2) is inserted in the "A" register and a jump is made to label ERROR. If the 1706 is busy, the data transfer is terminated and alarm error (2) code is inserted in "A" and a jump is made to label ERROR.

At label NOTALR the current buffer address BUFFAD is made equal to the starting address of the 80 word buffer (BUFFER) a jump is then made to label NOTALR modified by RETURN.

At label NEXT, a test is made to determine if the request is formatted. If it is formatted, jump to label FRDBIN. If it is not formatted, a test is made to see if the request is ASCII or Binary. If it is binary, jump to label FRDBIN. If ASCII skip to label FRDBIN.

At label FRDBIN, a test is made to determine the mode ASCII or binary. If it is binary, jump to label A8. If it is ASCII, an assembly option is used. If it is ASCII68, the driver treats it like binary and the next instruction is at label A8, but if it is ASCII63, the relative address from NOTALR to TOSKIP is stored in the subroutine return address RETURN. The release negate function bit is set in "A" and a jump is made to label DA1726.

At label A8, the relative address from NOTALR to BB is stored in RETURN. An Assembly option sets the release negate function bit in "A" if it is ASCII63 and the negate function bit for ASCII68, a jump is then made to label DA1726.

At label DA1726, the function code is increased to include end of operation and then stored in TEMPWD. Status is then taken and a test is made to check if the data bit is set. If it is not set, the subroutine return address is made to equal NEXT-NOTALR and exit is made to the dispatcher to await for Data Interrupt. If data is up, the card reader is functioned with the code stored in TEMPWD. Status word is then read from the card reader and stored in TEMPWD. Status is then taken and stored. A test is made to check for a separator card. If it is a separator card, reload memory and jump to label EOFRD.

If it is not a separator card, there is an assembly option, either buffered or non-buffered. If it is non-buffered, input the next word and take status. A test is then made to check for end of operation. If there is no end of operation, the card transfer is not complete, and the cycle is repeated with a jump to label PETE. If end of operation status is up, jump to label NOTALR. If it is buffered, a check is made for a binary card. If it is a binary card, jump to label LONGBF. If it is not a binary card, a check is made to see if the ASCII63 conversion is used, if not, jump to label LONGBF. If the ASCII63 conversion is used, a check is made to see if it is an ASCII Request. If it is an ASCII Request, a jump is made to test if the request is formatted. If the request is not formatted, jump to label LONGBF. If it is formatted, there is an ASCII card in a binary deck and a jump is made to label SHORT.

At label SHORT, load A with $0928. This is computer code for increase A by $28. Jump to label STBF.

At label LONGBF, load A with $0950, increase A by $50, jump to label STBF.

At label STBF, A is stored in BLENTH. The last address +1 of the buffer is placed in the first word of the buffer. Start the 1706 Data Transfer and exit.

At label TOSKIP, the first word that was read in routine DA1726 is taken from storage TEMPWD and placed in the first word of the 80 word buffer. Then jump to label ASCII.

The ASCII conversion that is used is an assembly option controlled by an equate instruction. If EQU ASCI68(0) then the standard ASCII hardware conversion is used. If EQU ASCI68(1 or 2) then the software conversion is used.

At label ASCII, if EQU ASCI68(0), the status word that was taken after word one was read in routine DA1726 and stored in word STATUS. The binary and separator status bits are set after the first word is transferred to computer memory. A test is made on this word for binary status. If it is binary, a no 79 error code (12) is entered in °A° and a jump is made to label ERROR. If it is not a binary card, the first word of the 80 word buffer is transferred to the first word of the users buffer. A test is then made to see if the users buffer is full. If it is full, terminate transfer and jump to label DONE. If it is not full, a test is made to see if 40 words have been transferred. If 40 words have not been transferred, jump to label ASCII and repeat the cycle. If 40 words have been transferred a test is made to determine if the request was formatted. If the request is formatted, jump to label DONE. If the request is non-formatted, jump to label FEEDCD.

At label ASCII, if EQU ASCII68(1 or 2), a column is picked up from the buffer, converted to ASCII via subroutine CONVERT, shifted left 8 bits, and stored into the indicated core location. The PHYSTAB word CORE is then incremented. If the number of words requested has been read, label DONE is entered. If not, the buffer address is incremented. If the buffer has been emptied and this is a formatted request, label DONE is entered. If this is an unformatted request, label FEEDCD is entered.

Subroutine CONVERT is a subroutine to convert 12 bit hollerith columns to ASCII. It does this by computing an index to a table of ASCII codes. Punches in column 2 thru 7 add $62 through $67 to the index respectively. A punch in columns 1 and 9 add $71 and $79 respectively. A punch in column 8 adds $18 to the index. If the resultant sum exceeds $7F an illegal Hollerith punch combination was present and the Hollerith error flag will be set. If the index is below $80, the bottom 4 bits are saved and the zone punches are then processed. If column 12 is punched, $70 is added to the index. If column 11 is punched, $60 is added, and if column zero is punched, $50 is added. If the resultant sum exceeds $7F, more than one zone punch was present in the column and the punch combination is reported to be illegal and the Hollerith error flag is set as before, unless the CDC subset option was chosen then the illegal punch is checked for 11-0 and 12-0 punch. If 11-0 is punched, it is processed as $7D and if 12-0 is punched, it is processed as $7B. Whenever an illegal Hollerith punch is detected the ASCII representation for a question mark (?) is set in that column. By doing so the user is able to complete his read, identify the error and continue if he desires. The Hollerith error flag is processed at label TOMAKQ. If no error is detected during the conversion the low order 6 bits of the index are used to access the ASCII character table and the conversion is complete. At label BB transfer the first word read in DA1726 and store it in the first word of the buffer. A word is obtained from the buffer via subroutine GETWRD. If this is the first card of the request, the card is checked to see if this is an end of file. If so, label EOFRD is entered, if not, the card is checked to see if it contains a 7,9 punch in column one. If it does, label FRDB2 is entered. If not, and it is not the first card, the 7,9 punch is missing and is reported via error code 12 to the Alternate Device Handler. If it is the first card, the mode is set to ASCII and label ASCII is entered.

At label FRDB2 the word just obtained is checked to see if checksum override is indicated. If so, the indicator is set. The sequence number is then checked. If incorrect and this is not the first card of a record; this fact is reported to the Alternate Device Handler with error code (9) set. If it is the first card the current sequence number is then set to the one just read. The sequence number is then incremented by one, and another word is obtained from the buffer via subroutine GETWRD.

If this is the first card of the record, this word should be the complemented record length. If non-negative, this fact is reported to the Alternate Device Handler with error code 10. If it is negative, it is stored in PHYSTAB word 18. At label FRDB7 the loop begins which gets the rest of the word in the record.

The purpose of subroutine GETWRD is to construct successive 16 bit words from the 12 bit columns stored in the buffer. To perform this, 4 columns are combined to produce three words. This is performed in three successive cycles as words are requested. Upon completion of each cycle, the next cycle is set for future entry.

At label DONE, the interrupts are cleared and a jump is made to label MAKEQ. Upon return the diagnostic clock is set negative, and a jump is made to the IN1726 + 1.

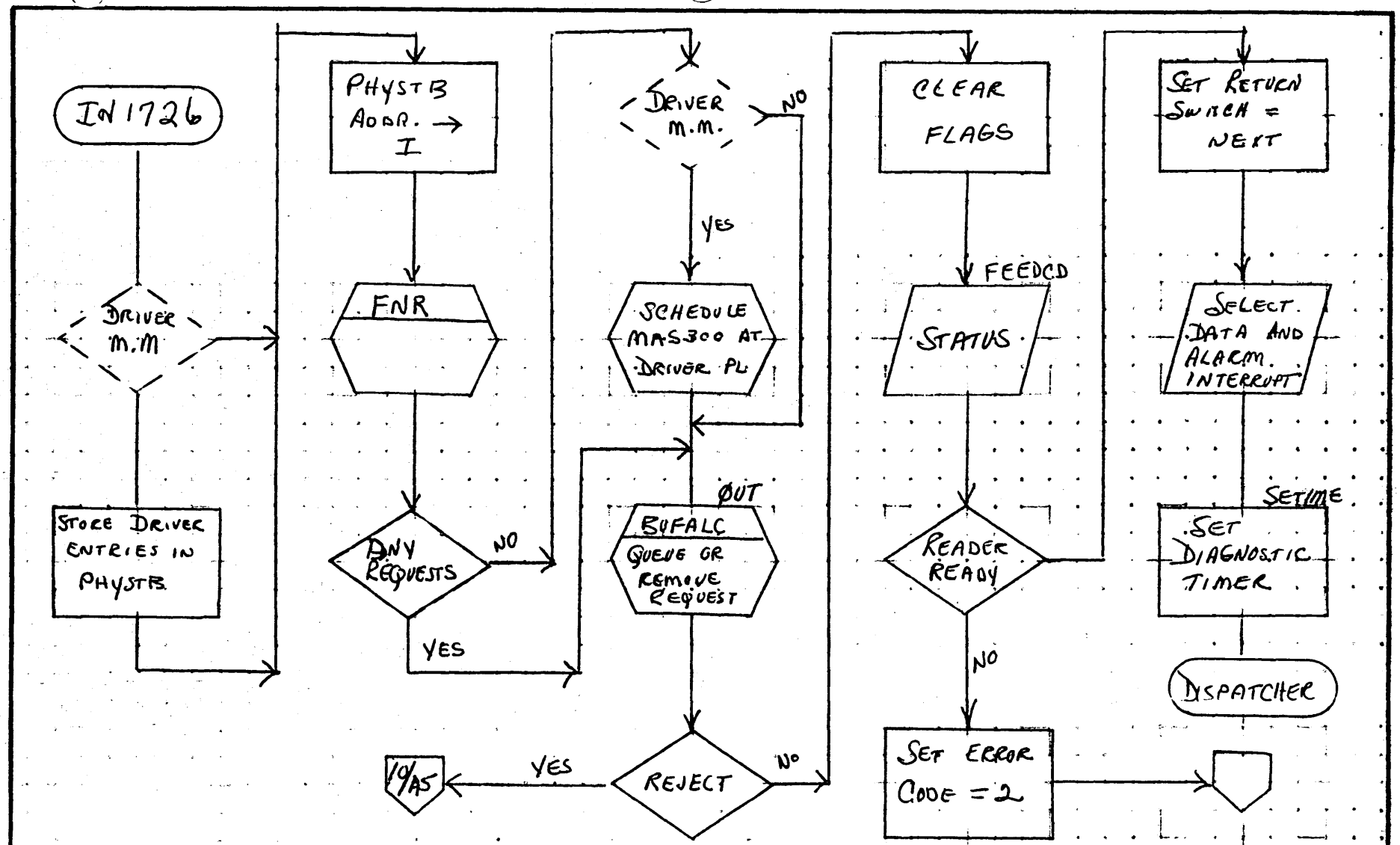
At label ERROR, the error code and logical unit number are combined for output by the Alternate Device Handler, and bits 14 and 15 of the request status PHYSTAB word are cleared. Status is saved, the controller is cleared, and subroutine MAKEQ is entered. Upon return, exit is made to the Alternate Device Handler.

The purpose of subroutine MAKEQ is to set the address of the next word of the buffer if a READ request does not get as many words as requested. Bits 13, 14, 15 of ERRTAB are set if there was a device failure, short read or end of file condition. Exit is made to the address stored in RETURN.

At label FRDB7, if the remaining record length is zero, exit is made to label FRDB8 to see if the checksum is correct. If not, the core address is checked to see if the number of words requested has been read. If so, record length is incremented and checked for zero. If not, reading continues until the entire record has been read. At this time, the complement of the checksum is read and the sum checked for zero. If not, and the checksum override was not indicated, error code four is reported to the Alternate Device Handler. If zero or checksum override was indicated, exit is made to label DONE.

At label RDBIN, the current core location is checked against LAST+1. If equal, one column only is to be read and label ONECHR is entered. If not, a word is obtained from the buffer via subroutine GETWRD. It is stored into core and the core location is incremented. This is repeated until the number of words requested has been read, at which time label DONE is entered. At label ONECHR, the low order four bits are cleared from the word just read and the word is stored into core. Label DONE is then entered.

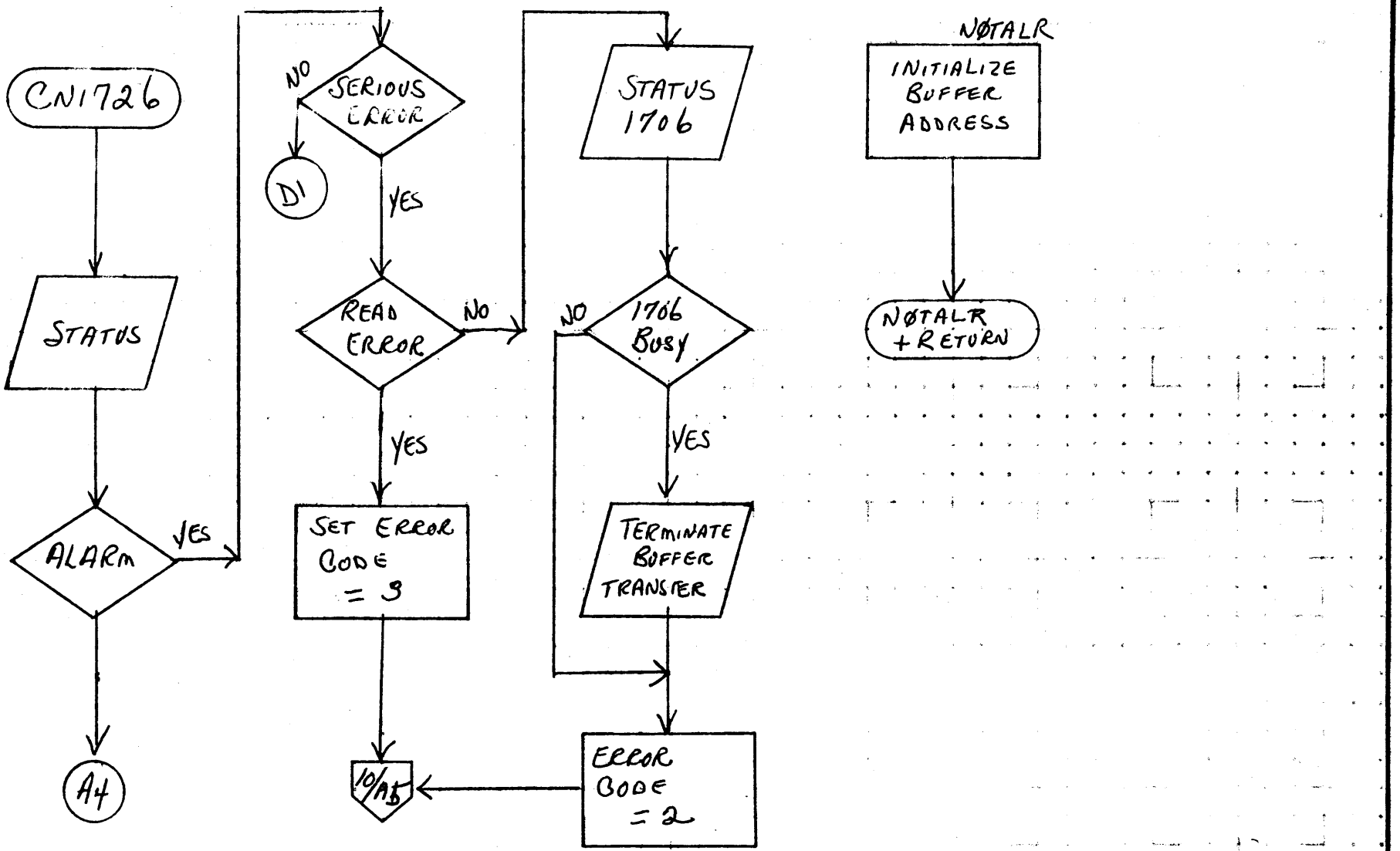
At label EOFRD, the error bit is set in the request status word of physical device table. Exit is then made to label DONE. This will cause entry to be made to the completion routine with the error indicator set without first going to the Alternate Device Handler.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: 1726/405 DRIVER		PROJECT MGR.		
	BUFFERED	PAGE 1 OF 15	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

49.8



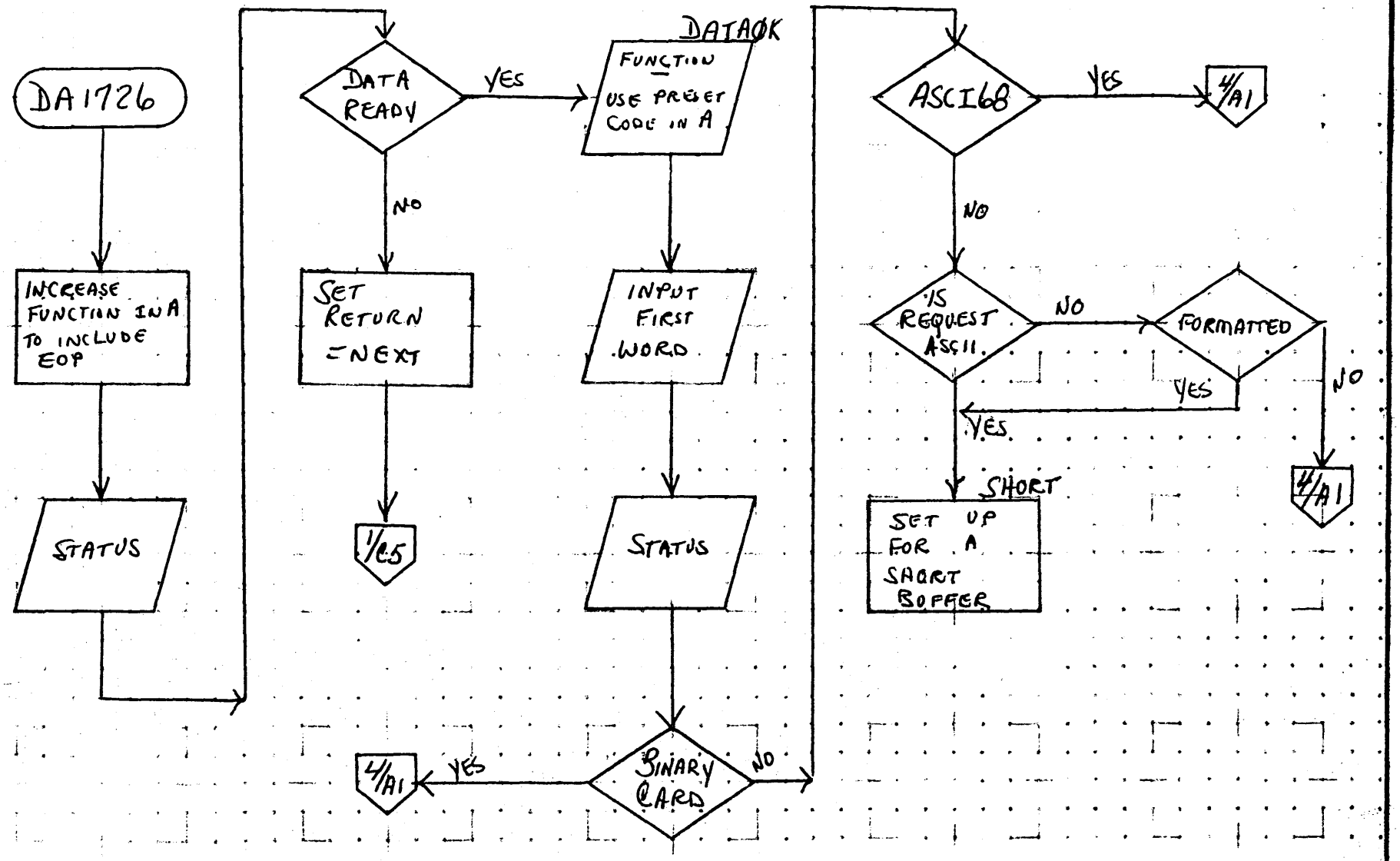
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1726/405 DRIVER		PROJECT MGR.		
	BUFFERED	PAGE 2 OF 15	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

4.9.9

AA1385 (FORMERLY CA127-)

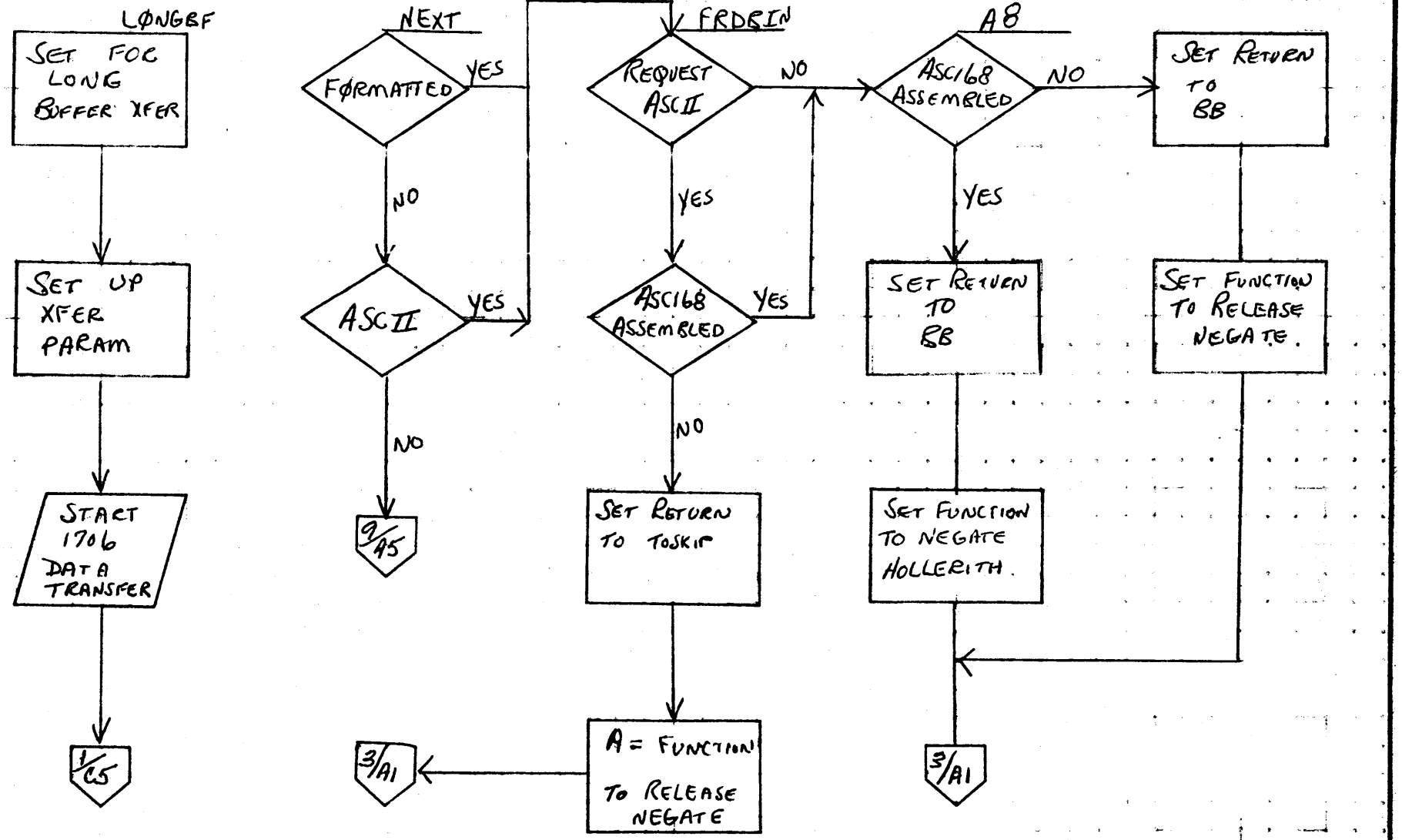
PRINTED IN U.S.A.



CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE 1726/405 DRIVER		PROJECT MGR.			
SAMPLE CODE ☐	BUFFERED	PAGE 3 OF 15	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

MAR 5 1971

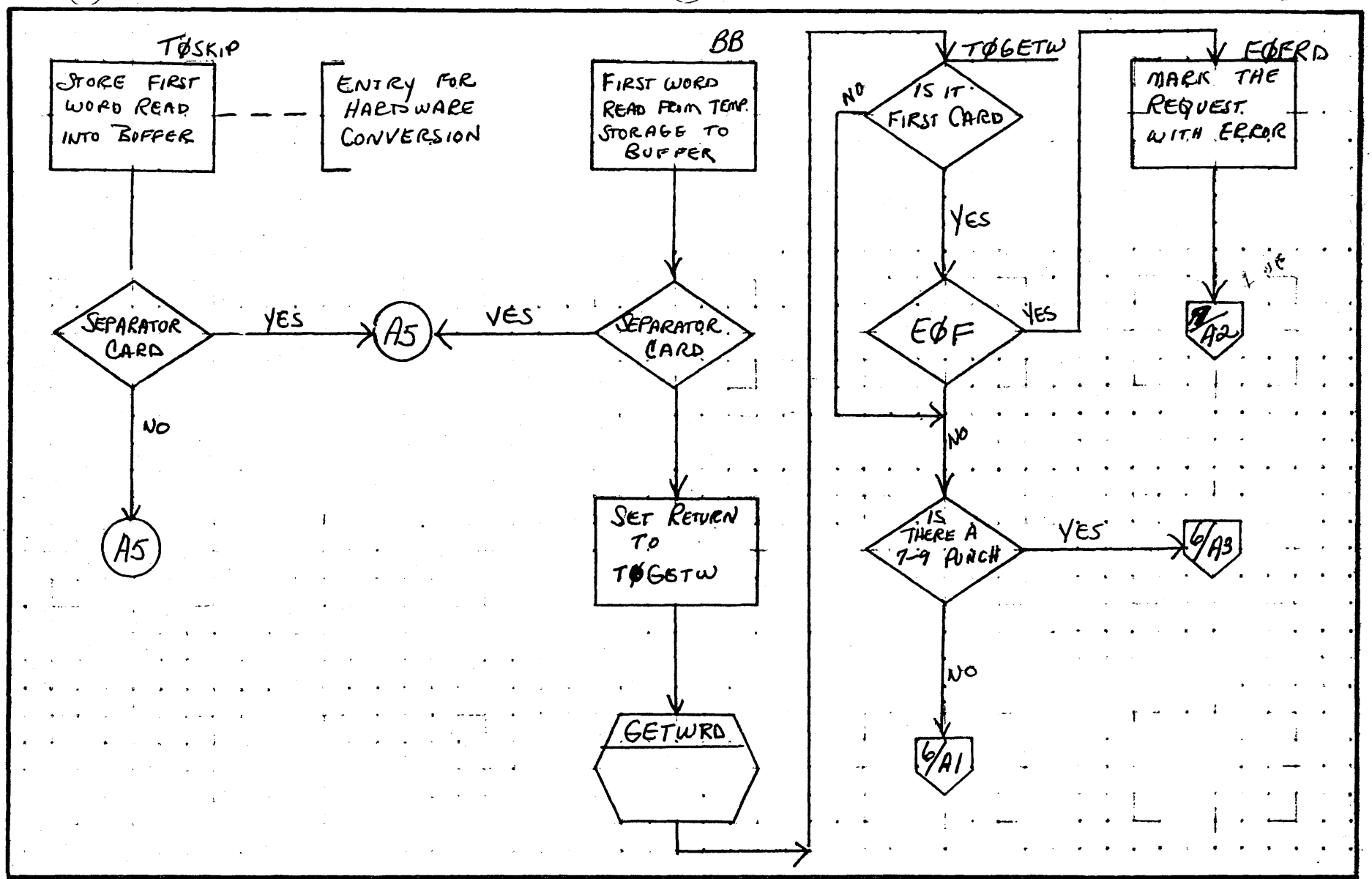
49.10



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1726/405 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 4 OF 15	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

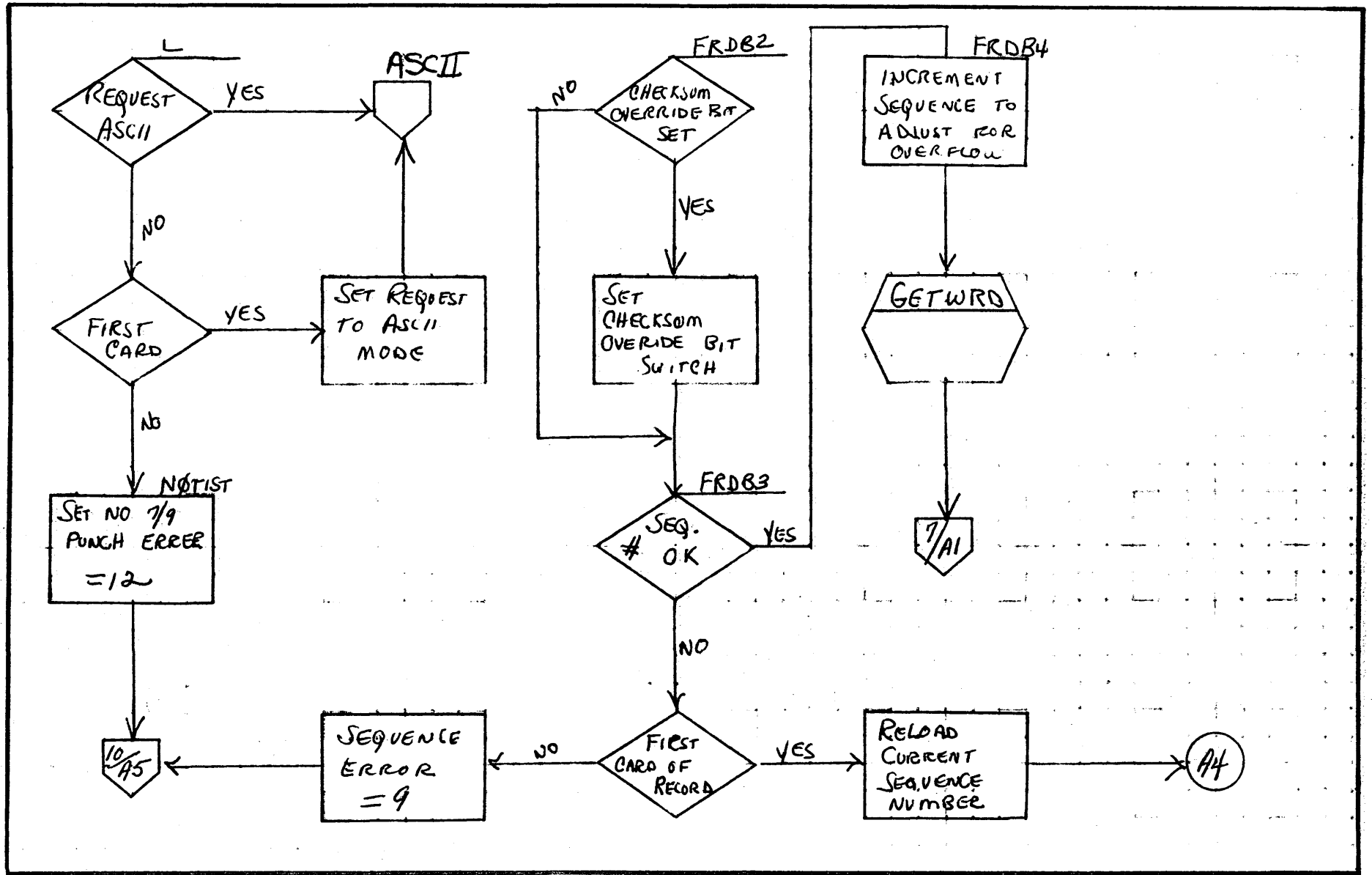
49.11



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1726/405 DRIVER		PROJECT MGR.			
	BUFFERED	PAGE 5 OF 15	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

49.12



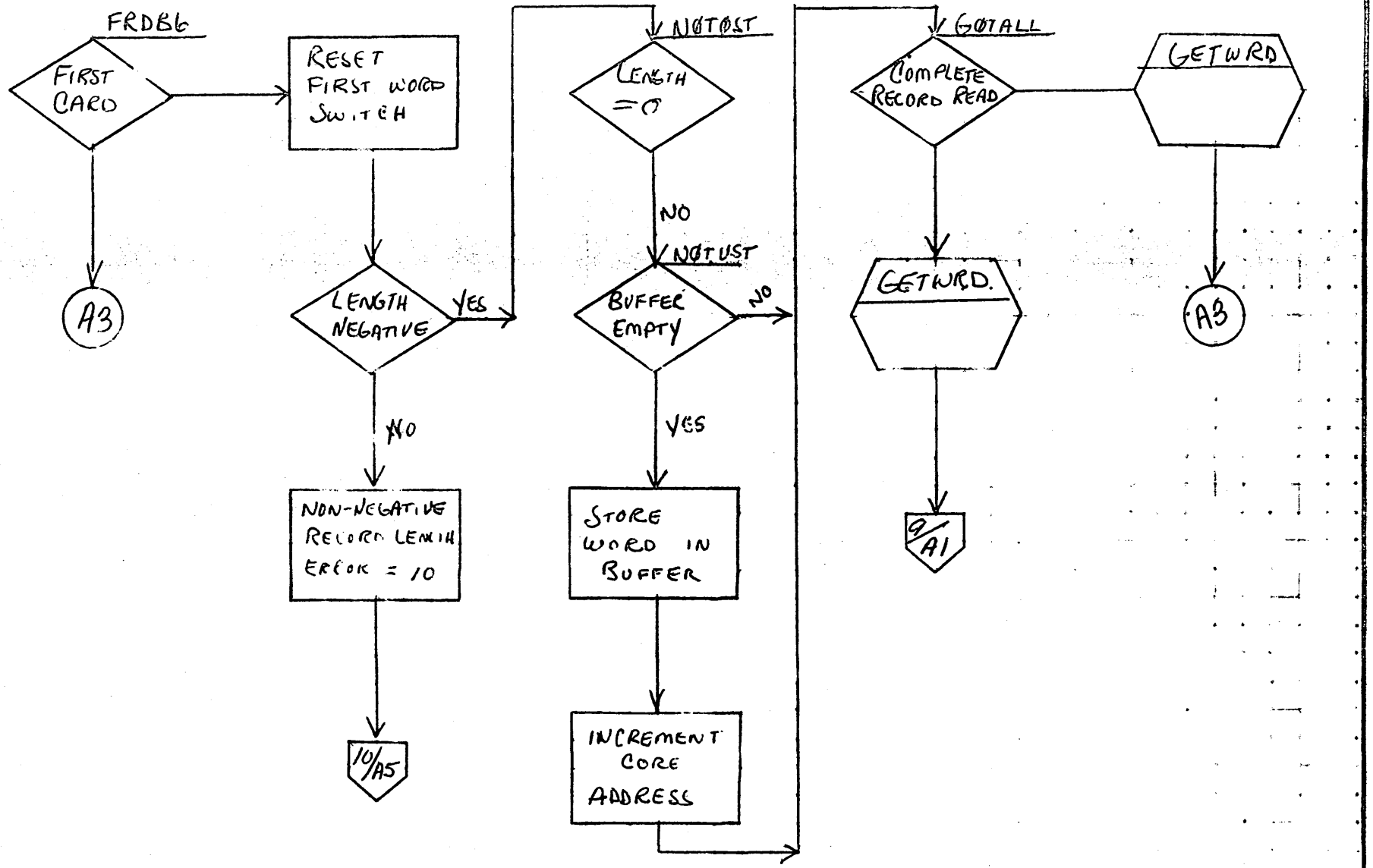
CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE 1726/405 DRIVER BUFFERED PAGE 6 OF 15	PROJECT MGR.	
	NUMBER ISSUE DATE	PROJECT NAME	
	DRAWN BY DATE	TASK NO.	
		TASK NAME	

AA1385 (FORMER CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

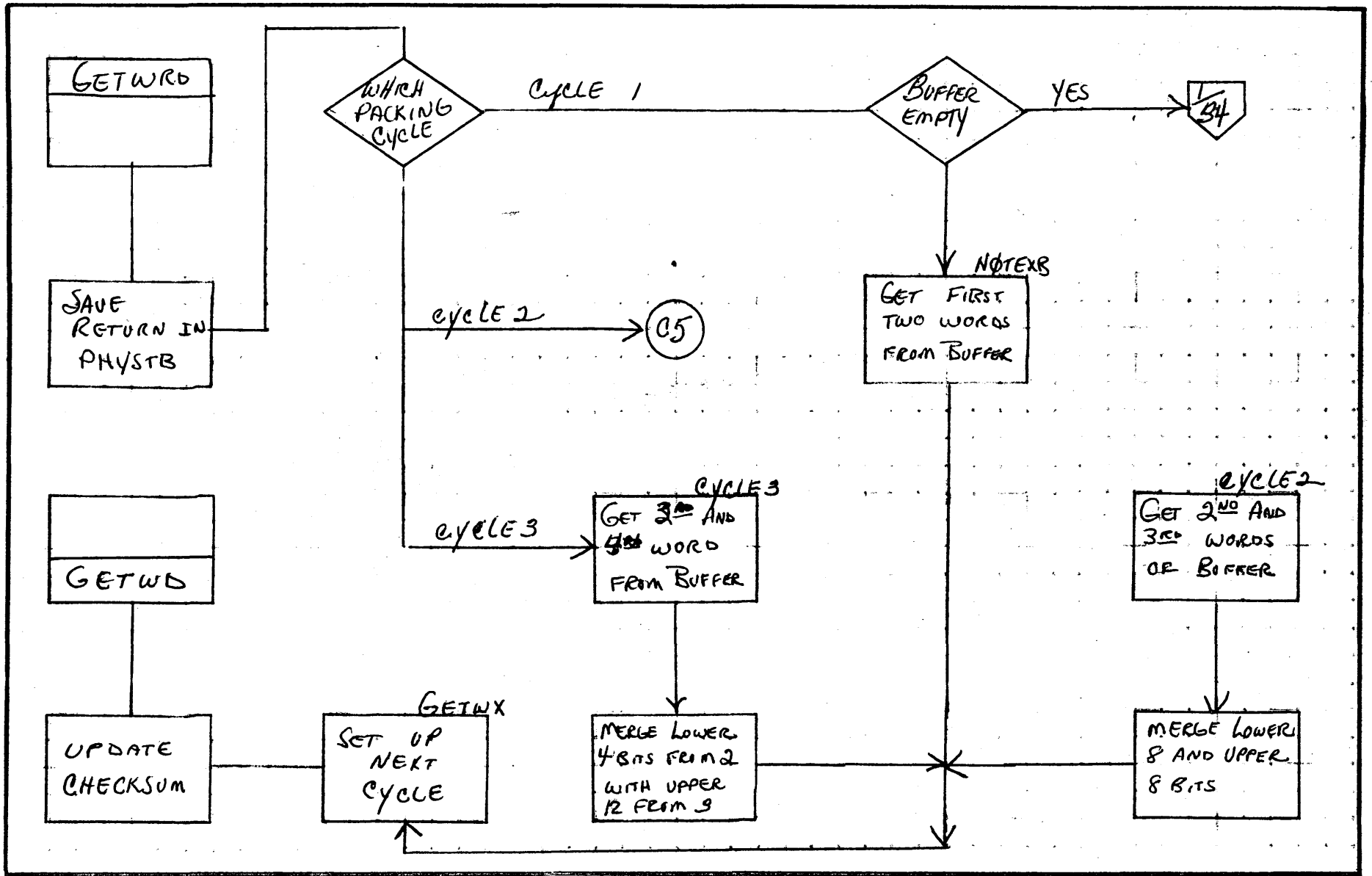
49.13



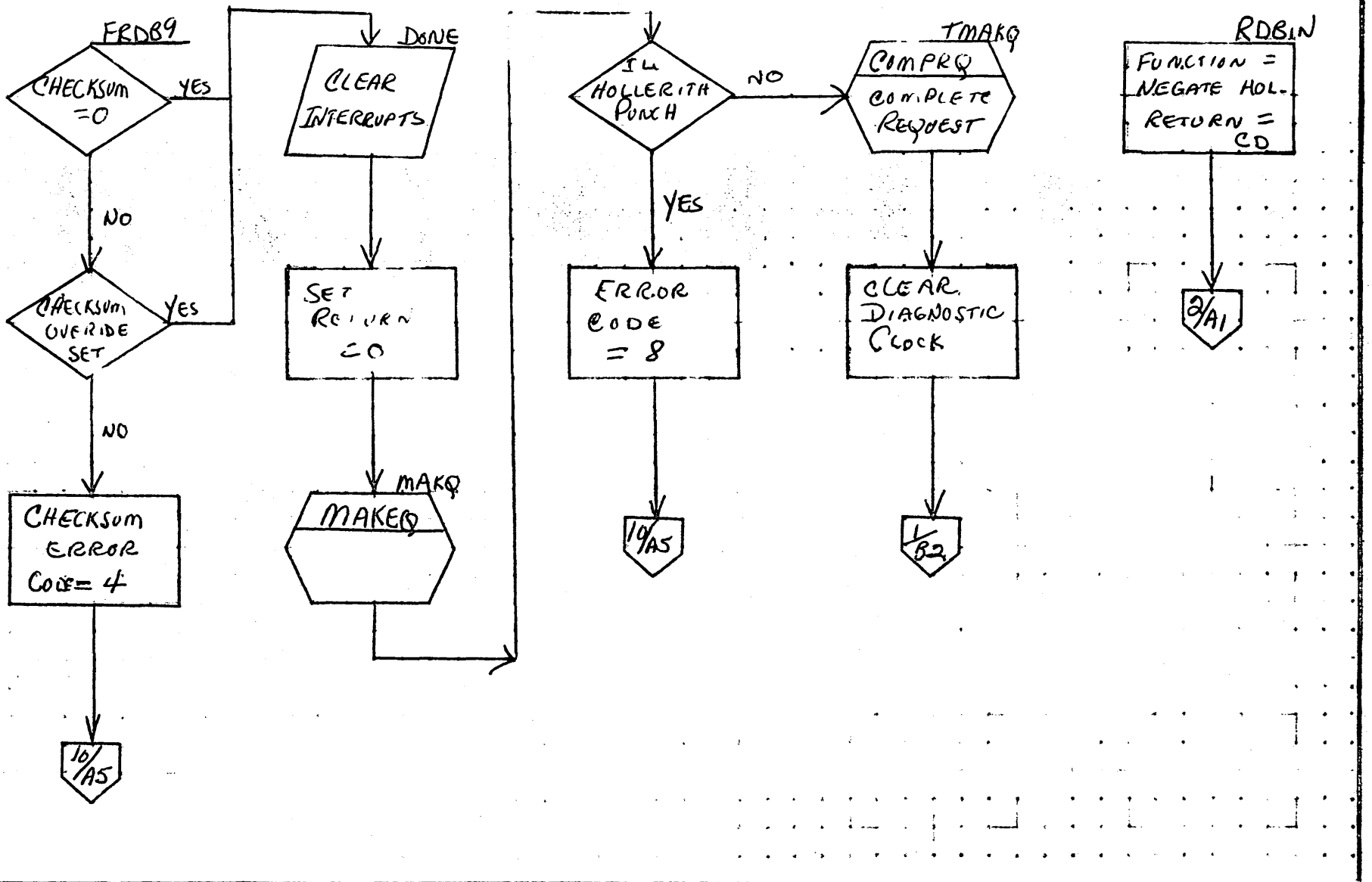
CONTROL DATA CORPORATION SOFTWARE DOCUMENTATION SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1406/405 DRIVER	PROJECT MGR.			
	BUFFERED PAGE 7 OF 15	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

49.14



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: 1726/405 DRIVER BUFFERED	PAGE 8 OF 15	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

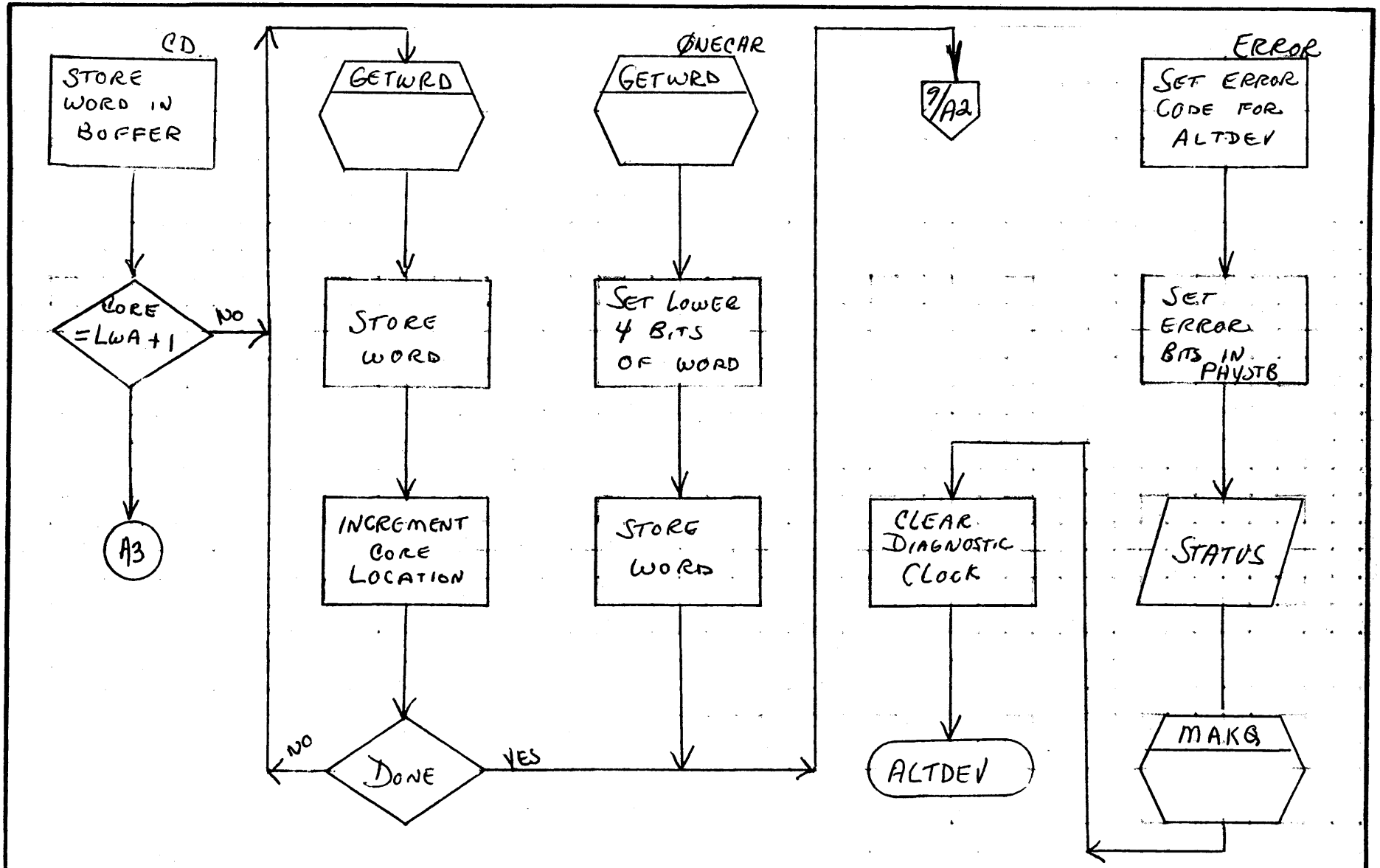


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECK IN AB F OTHER	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1726/405	DRIVER	PROJECT MGR.			
	BUFFERED	PAGE 9 OF 15	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN B	ATE	TASK NAME			

MAR 5 1971

49.16

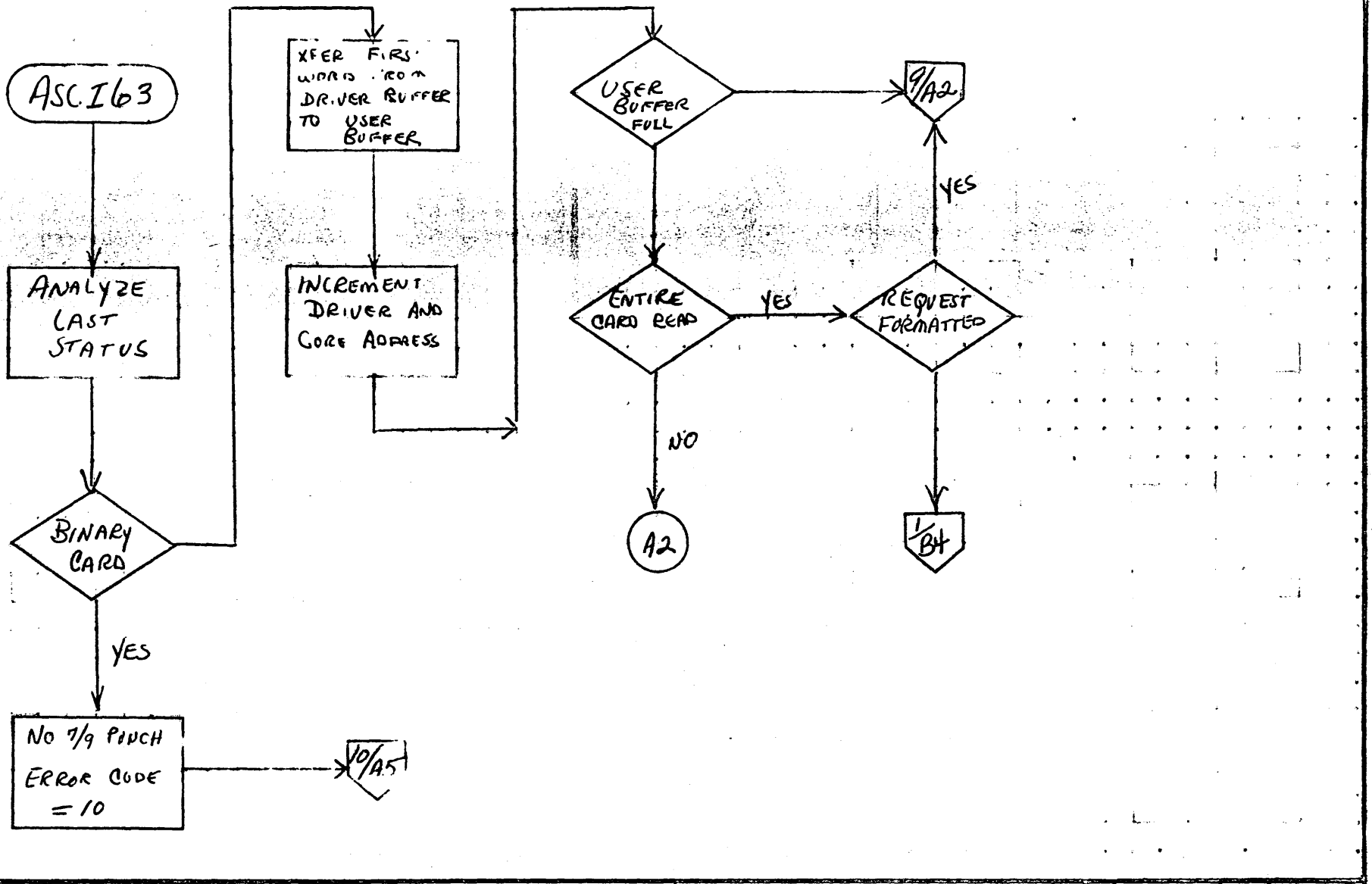
PRINTED IN USA



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE 1726/405 DRIVER BUFFERED PAGE 10 OF 15 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

MAR 5 1971

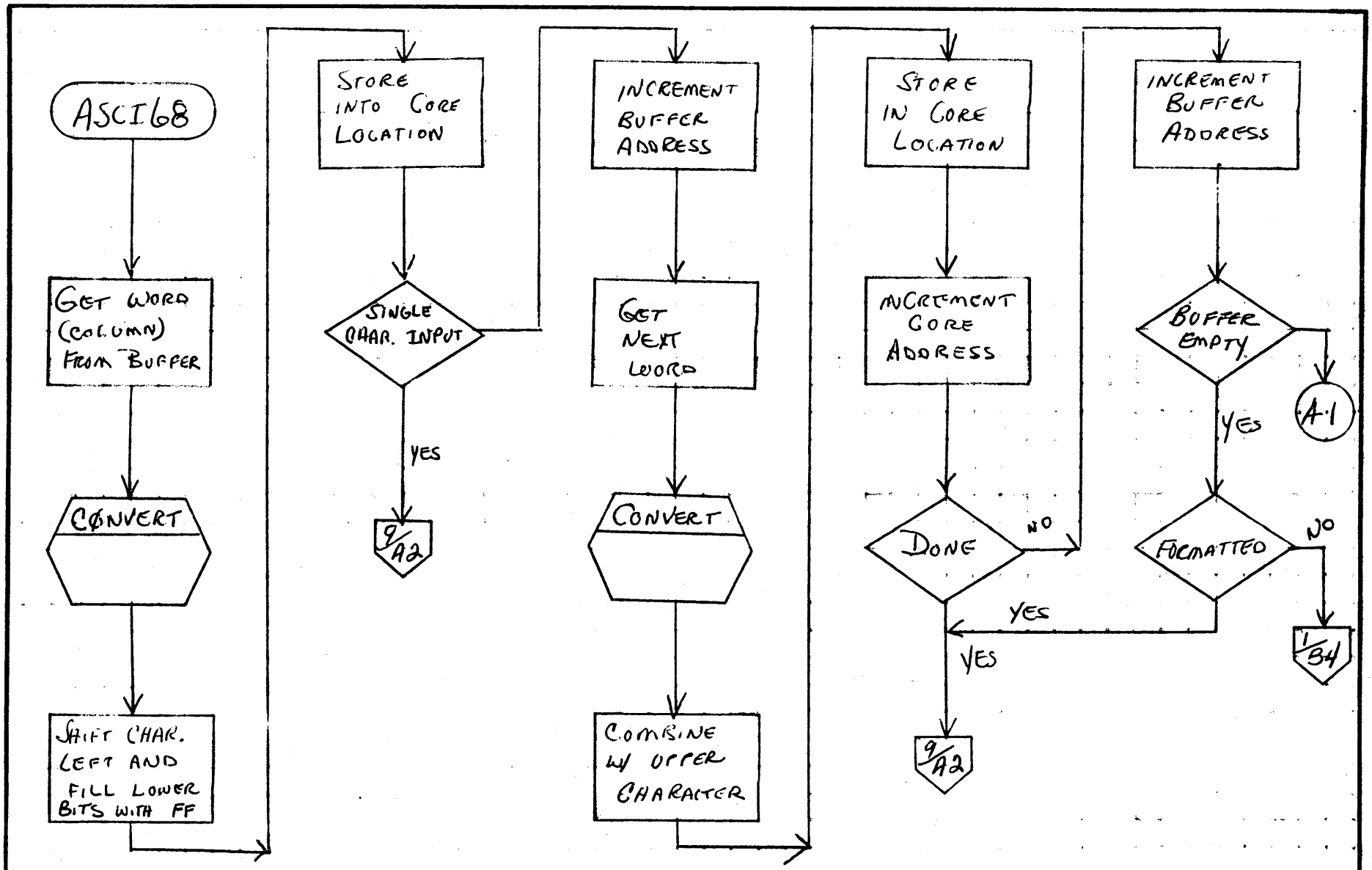
49.17



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT TITLE 1726/405 DRIVER		PROJECT MGR			
	BUFFERED	PAGE 11 OF 19	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO			
	DRAWN B	DATE	TASK NAME			

MAR 5 1971

49.10

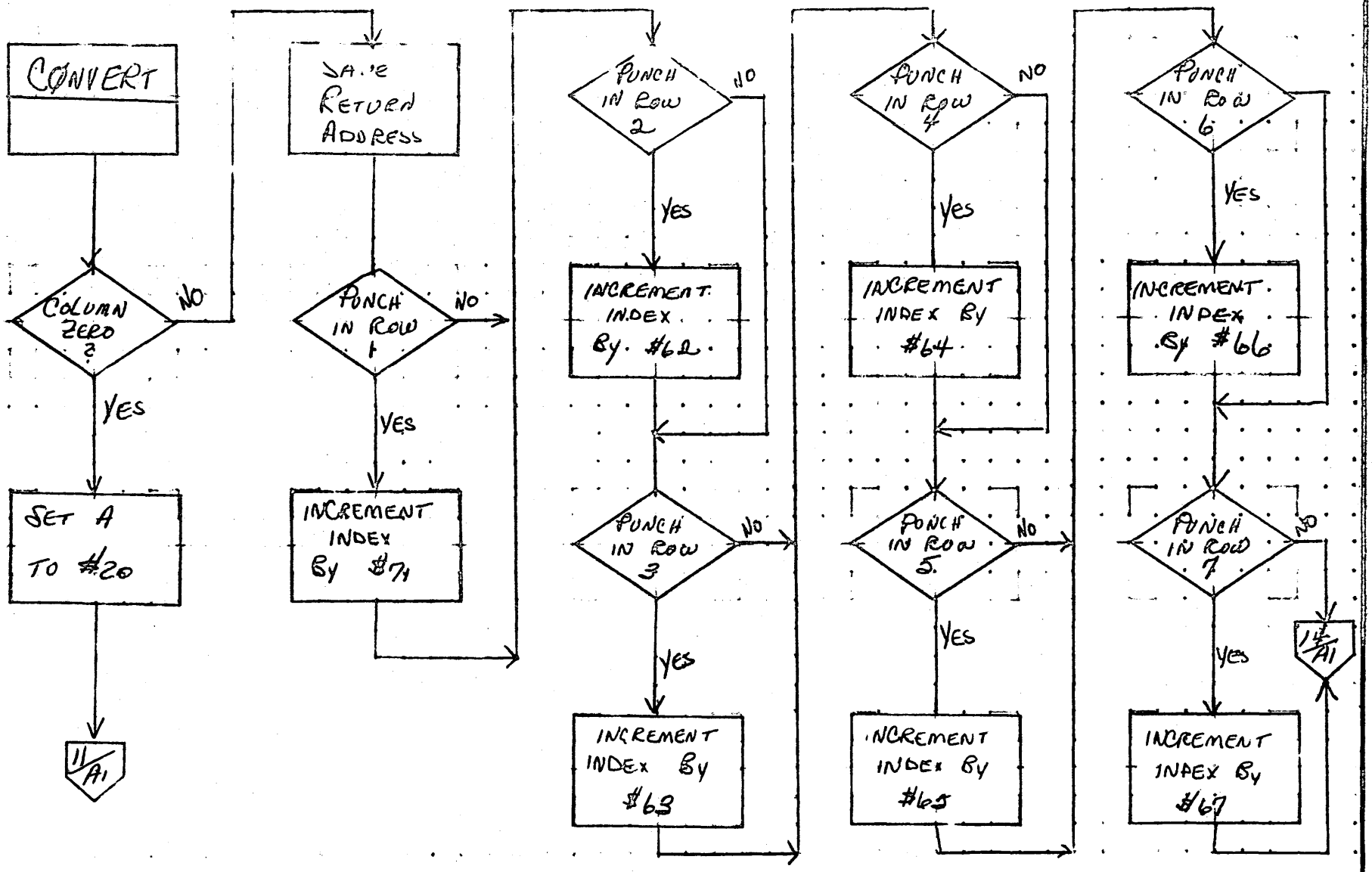


CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE 1720/405 DRIVER, BUFFERED PAGE 12 OF 15	PROJECT MGR.	
	NUMBER ISSUE DATE	PROJECT NAME	
	DRAWN BY DATE	TASK NO.	
		TASK NAME	

MAR 5 1971

AA1385 (FORMERLY CA127)

PRINTED IN U.S.A.



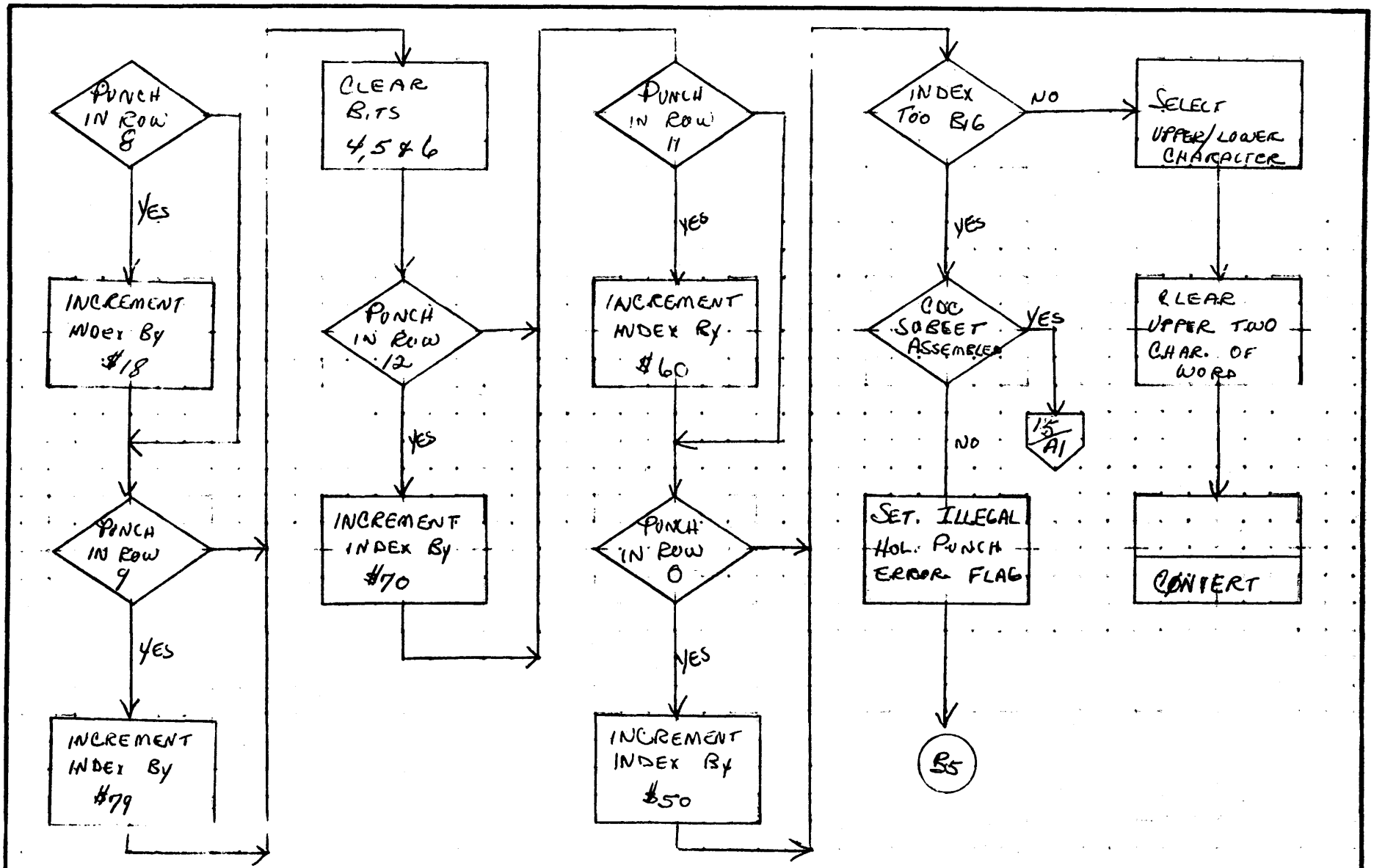
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1726/405 DRIVER BUFFERED	PAGE 13 OF 15	PROJECT MGR.		
			PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN B.	DATE	TASK NAME		

MAR c 1971

49.20

FORM 385 FORMER

PRINTED IN U.S.A.

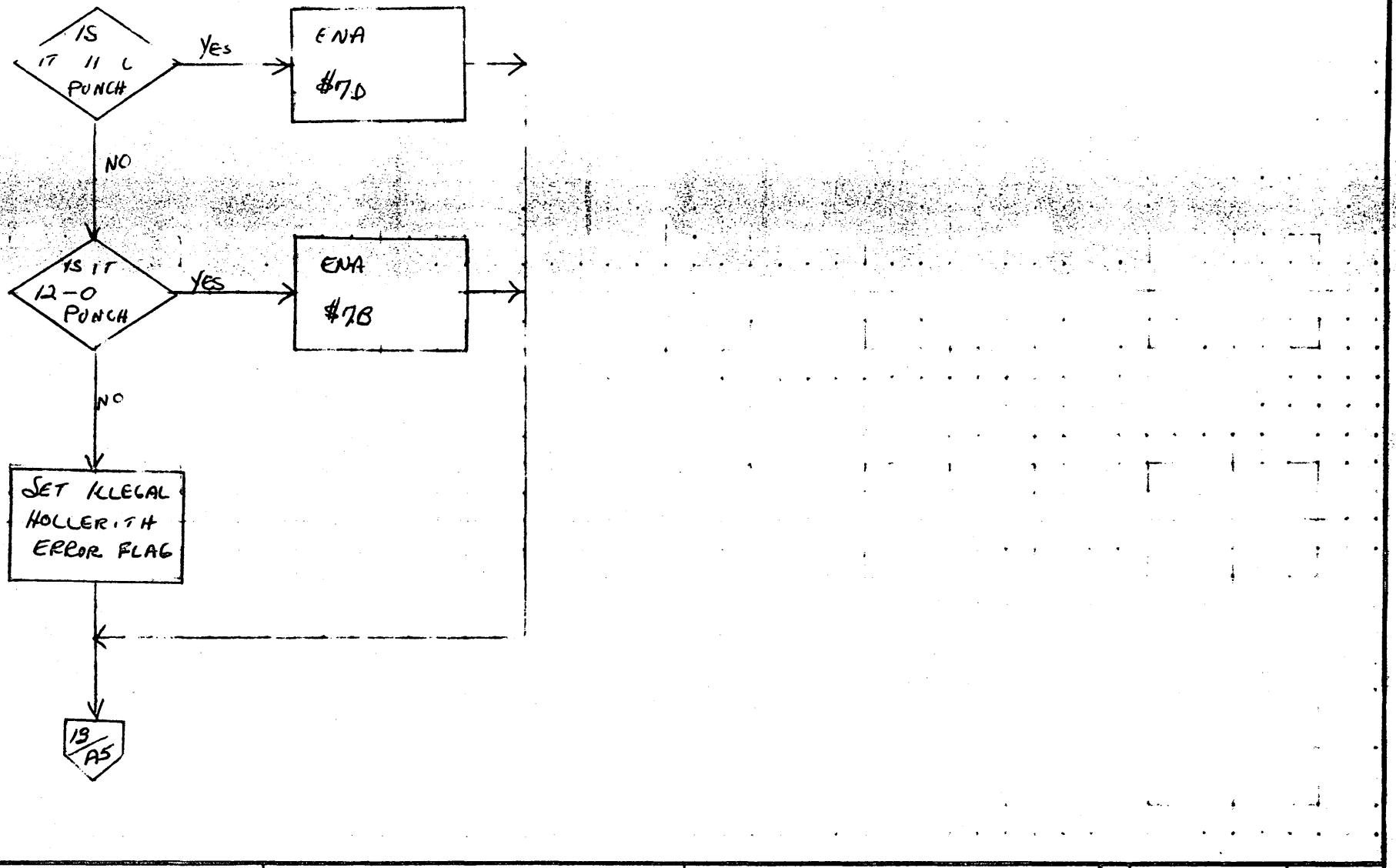


CONTROL DATA CORPORATION			
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.
SAMPLE CODE ☐	DOCUMENT TITLE 1726/405 DRIVER		PROJECT MGR
FLOWCHART ☐	BUFFERED	PAGE 14 OF 15	PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO
OTHER ☐	DRAWN BY	DATE	TASK NAME

REV	APPROVED	DATE

MAR 5 1971

49.21



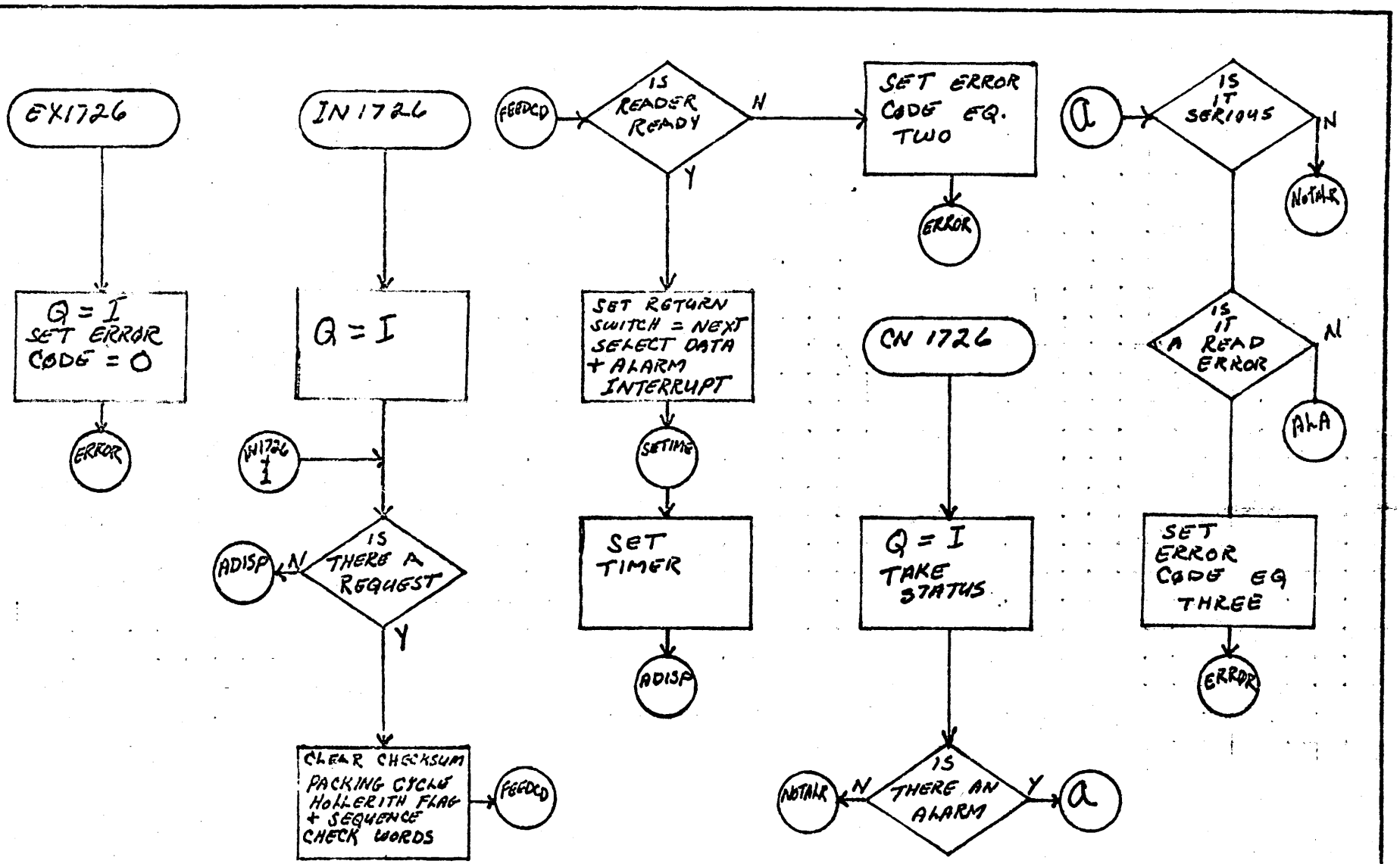
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1726/405 DRIVER BUFFERED PAGE 15 OF 15	PROJECT MGR.			
	NUMBER ISSUE DATE	PROJECT NAME			
	DRAWN BY DATE	TASK NO			
		TASK NAME			

MAR 5 1971

49.22

A 1385 FORM

PRINTED IN U.S.A.



SOFTWARE DOCUMENT	CLASS EFI2	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR.		
FLOWCHART ☐	NON-BUFFERED	PAGE 1 OF 14	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY 24/11.	DATE	TASK NAME		

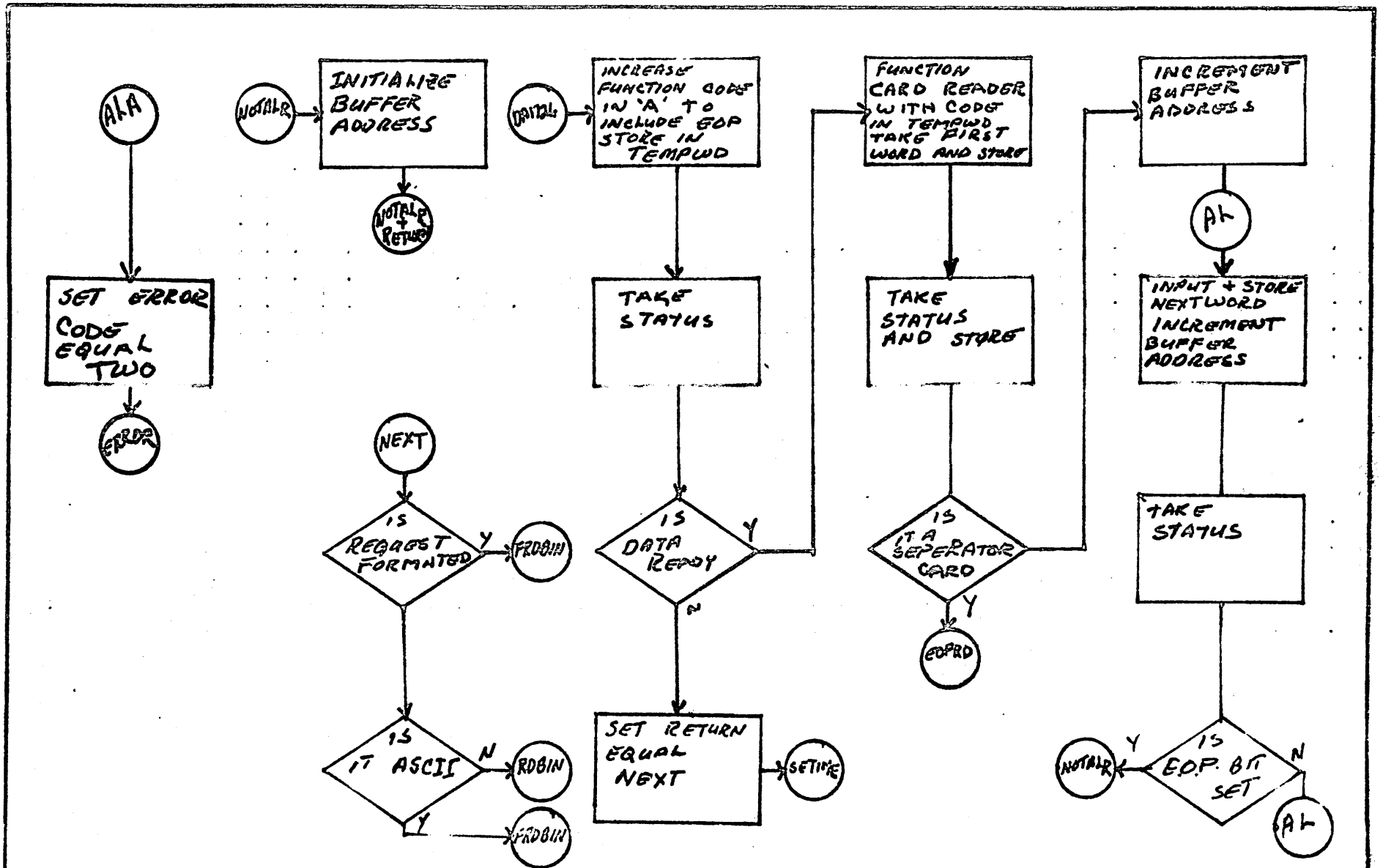
CONTROL DATA CORPORATION

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

49.23



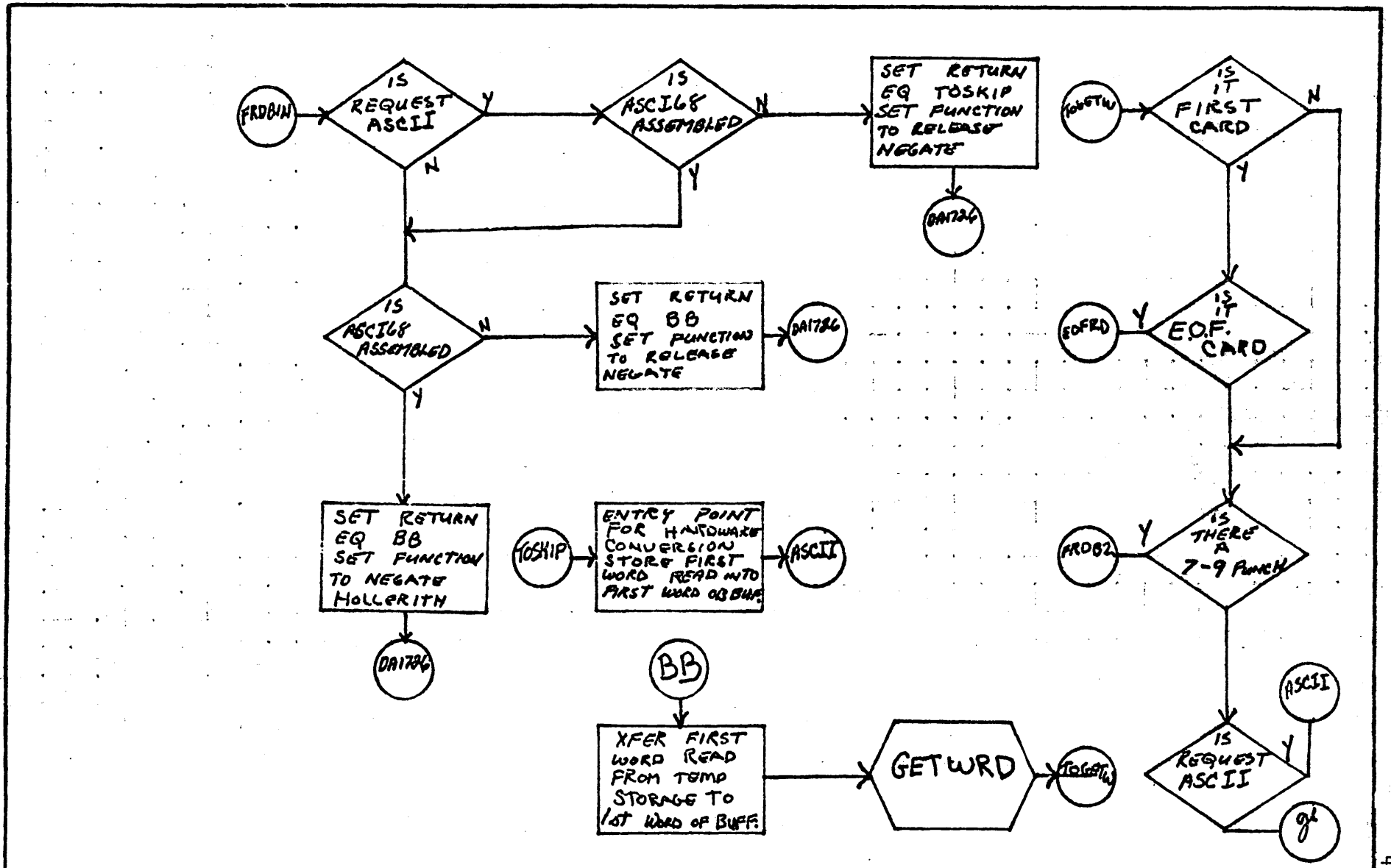
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1726/1405 DRIVER		PROJECT MGR			
FLOWCHART ☐	NON-BUFFERED	PAGE 2 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY 24771.	DATE	TASK NAME			

MAR 5 1971

49.24

AA 385 FORME[illegible]

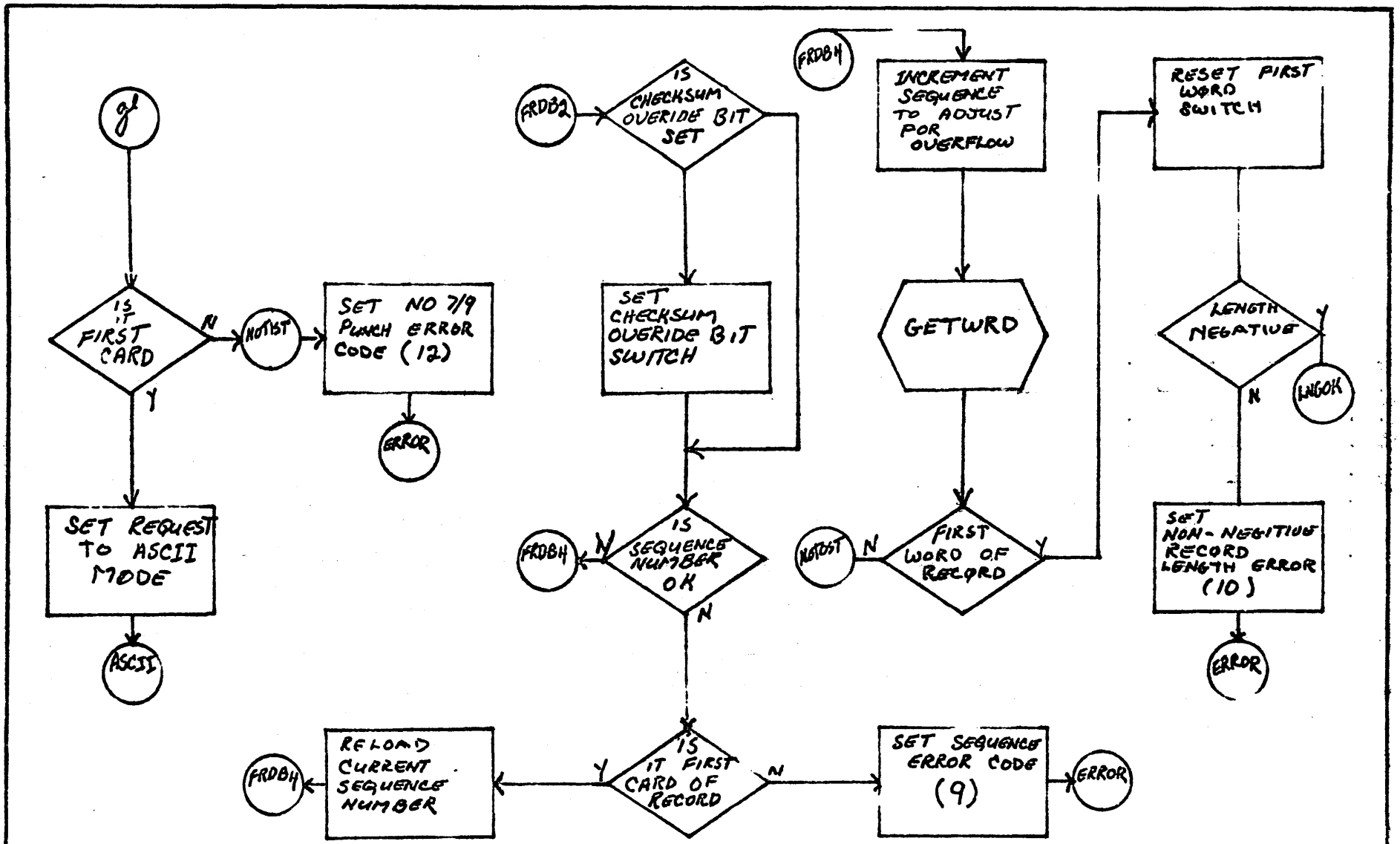
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR.		
	NON-BUFFERED	PAGE 3 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY 2HM.	DATE	TASK NAME		

MAR 5 1971

4-25

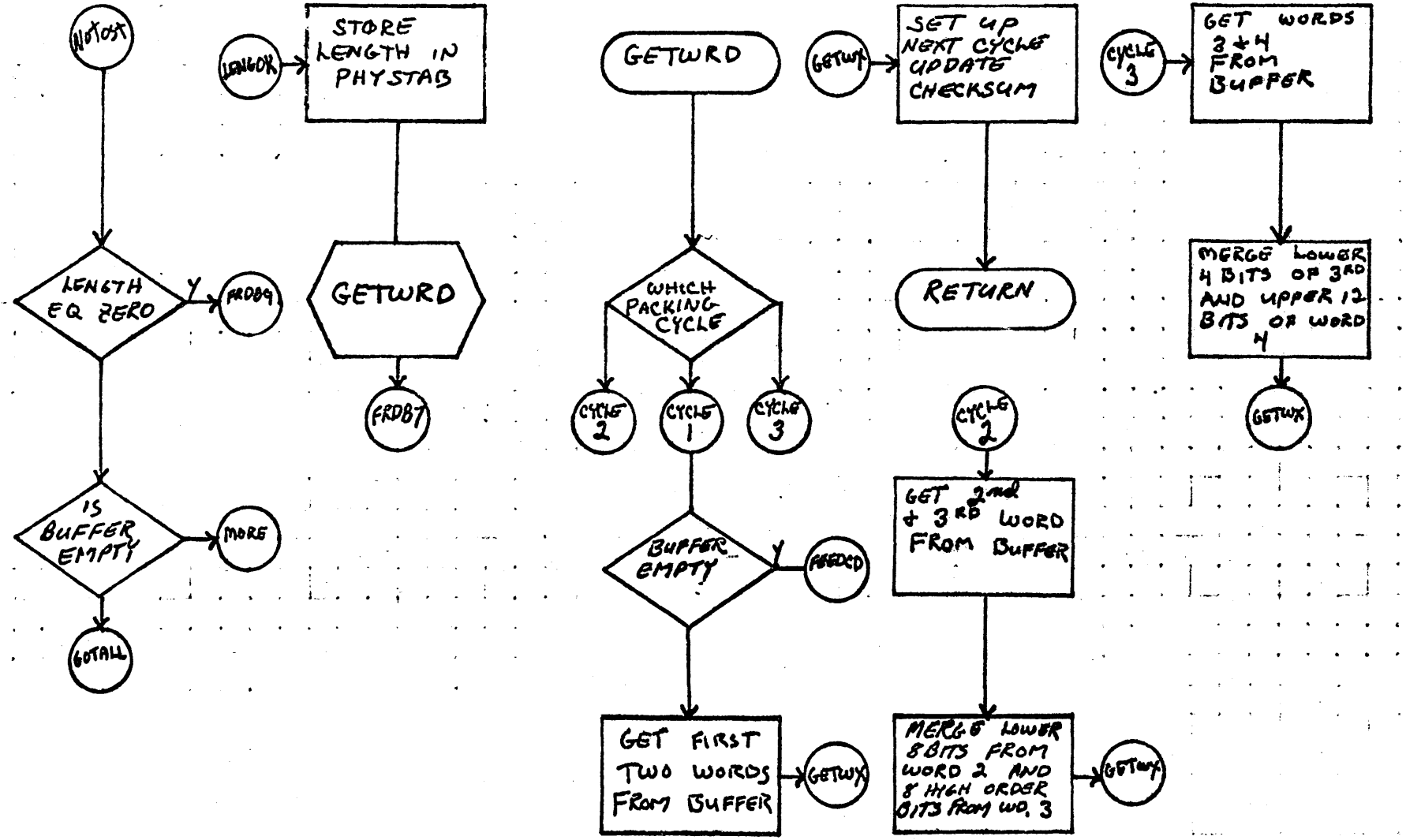


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOW CHART DECISION TABLE OTHER	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE 1726/405 DRIVER		PROJECT MGR		
	NON-BUFFERED	PAGE 4 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	JOB NO		
	DRAWING		TASK NAME		

PRINTED IN U.S.A.

MAR 5 1971

49-26



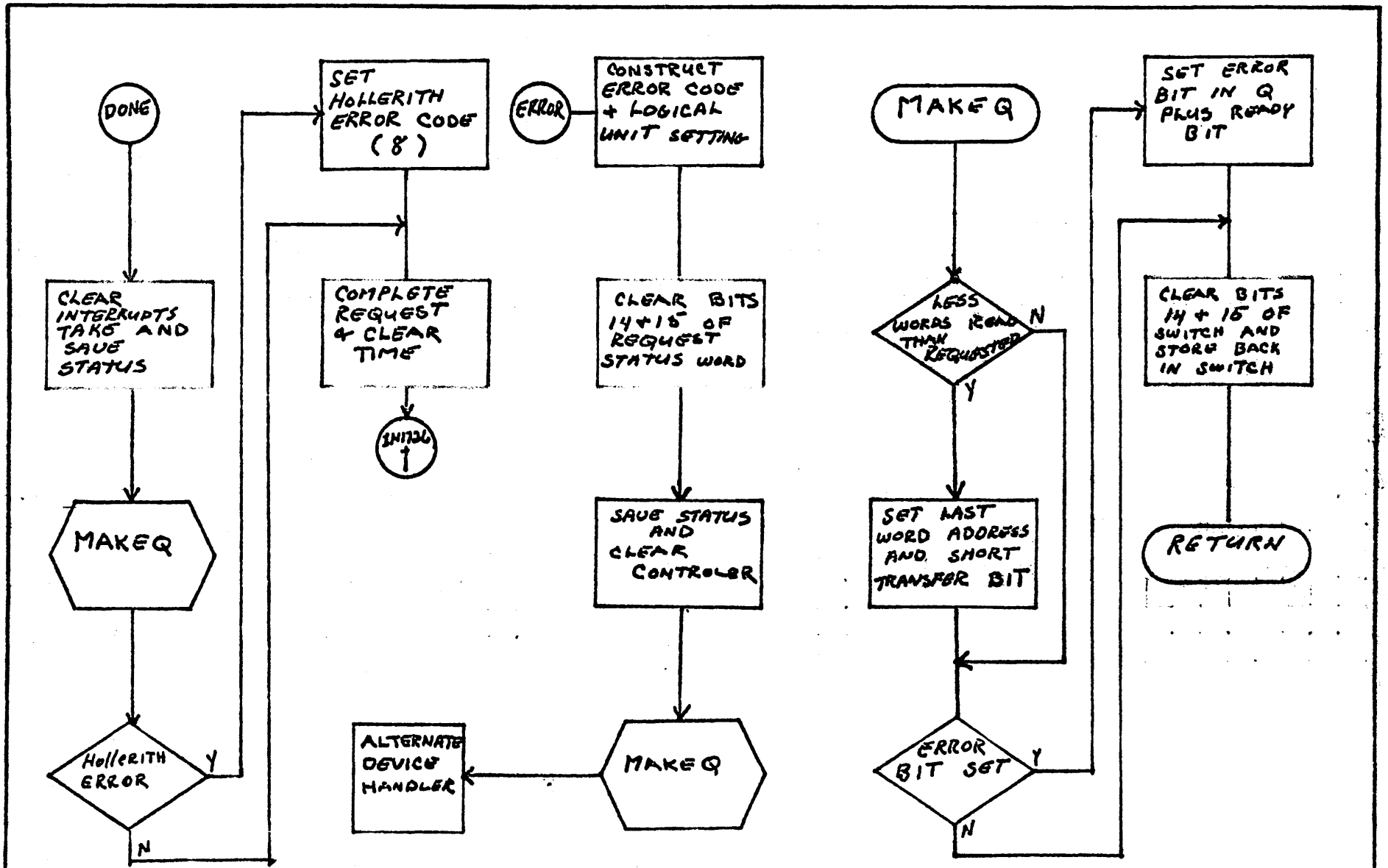
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR.		
	NON-BUFFERED	PAGE 5 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY 2471	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

49-27



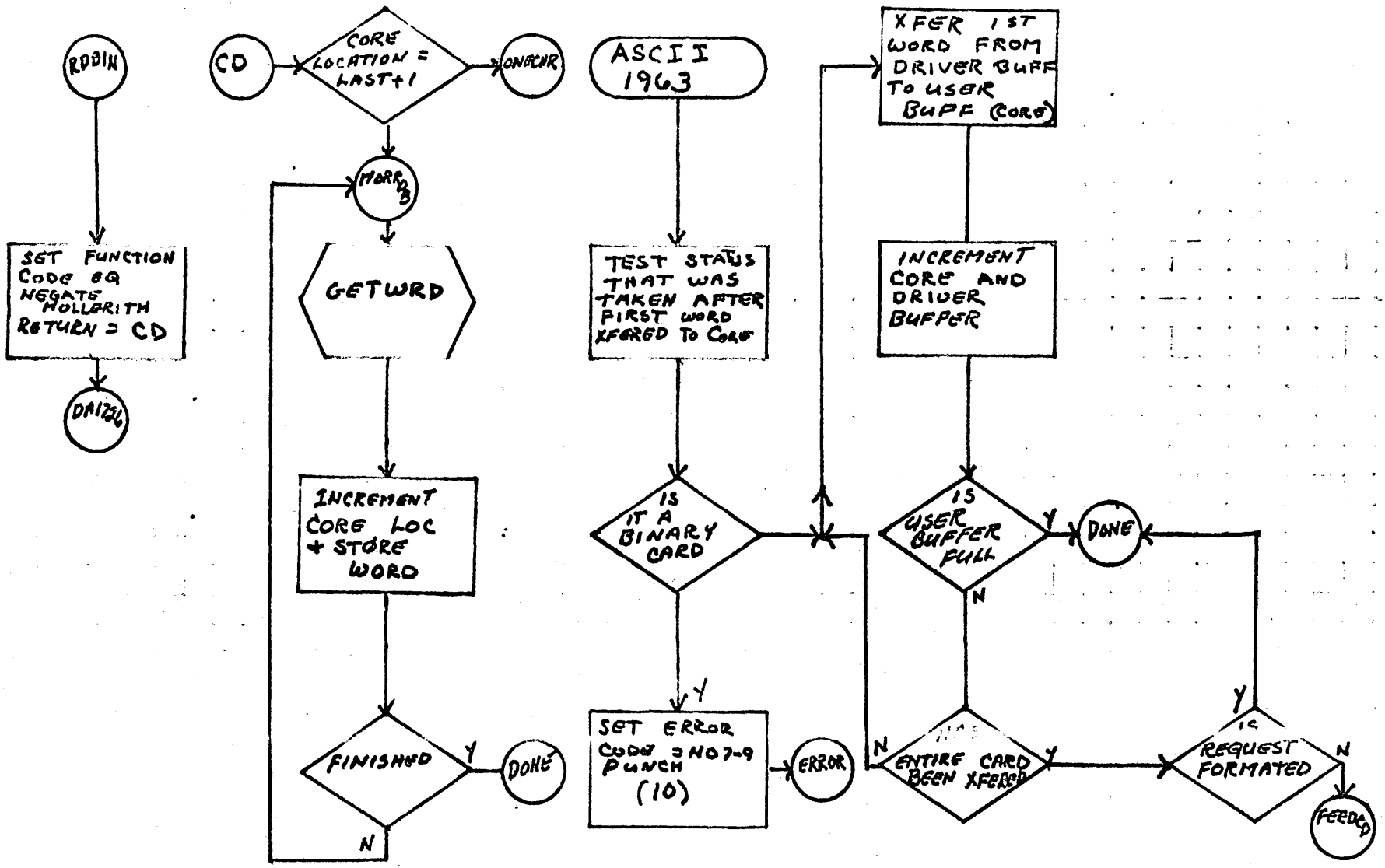
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR			
FLOWCHART	NON-BUFFERED	PAGE 6 OF 11	PROJECT NAME			
DECISION TABLE	NUMBER	ISSUE DATE	TASK NO			
OTHER	DRAWN BY 24M	DATE	ASK NAME			

MAR 5 1971

49.28

A-385 FORMERLY A-27

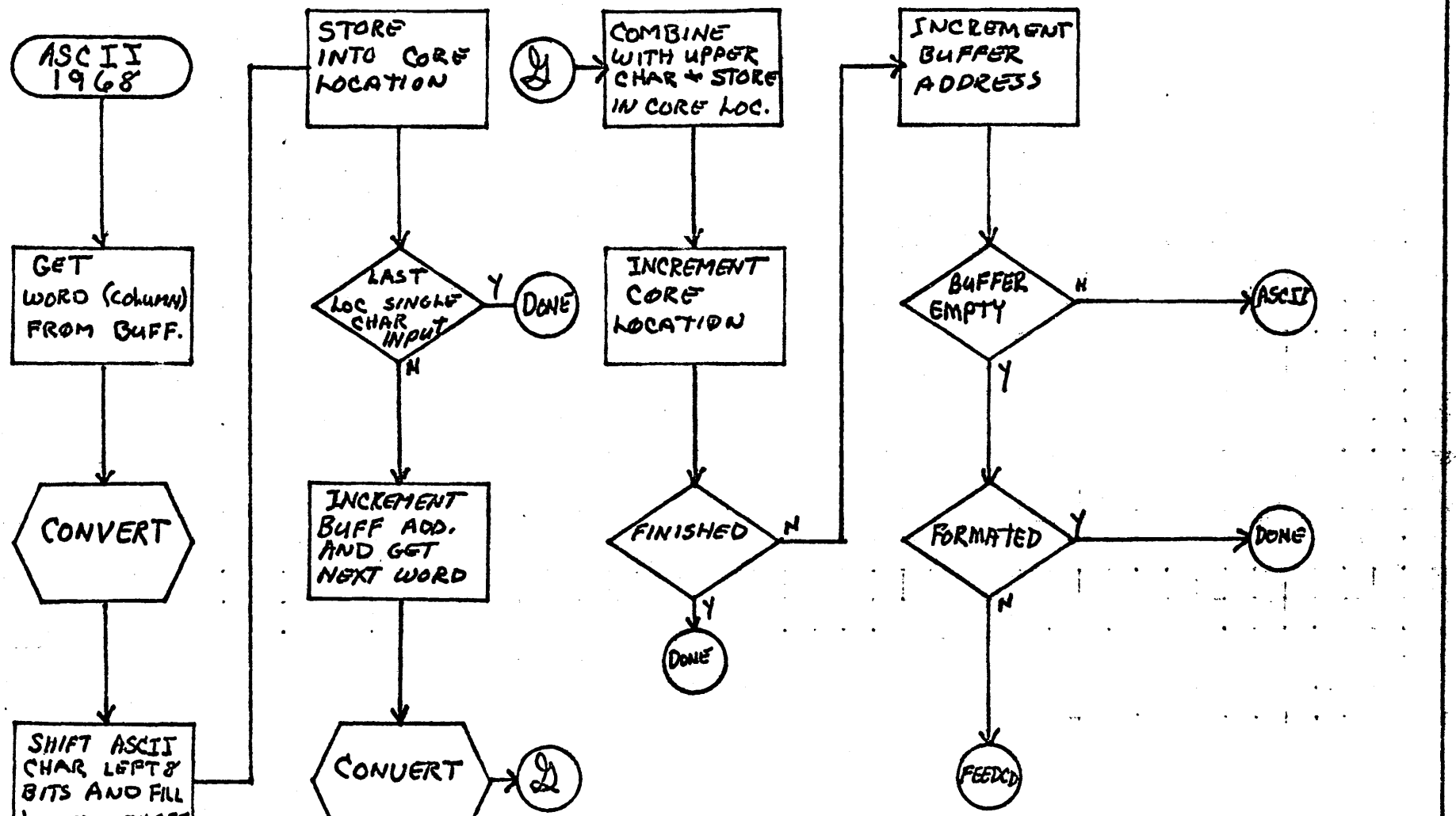
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR.			
FLOWCHART ☐	NON-BUFFERED	PAGE 7 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY JHM.	DATE	TASK NAME			

MAR 5 1971

49-29



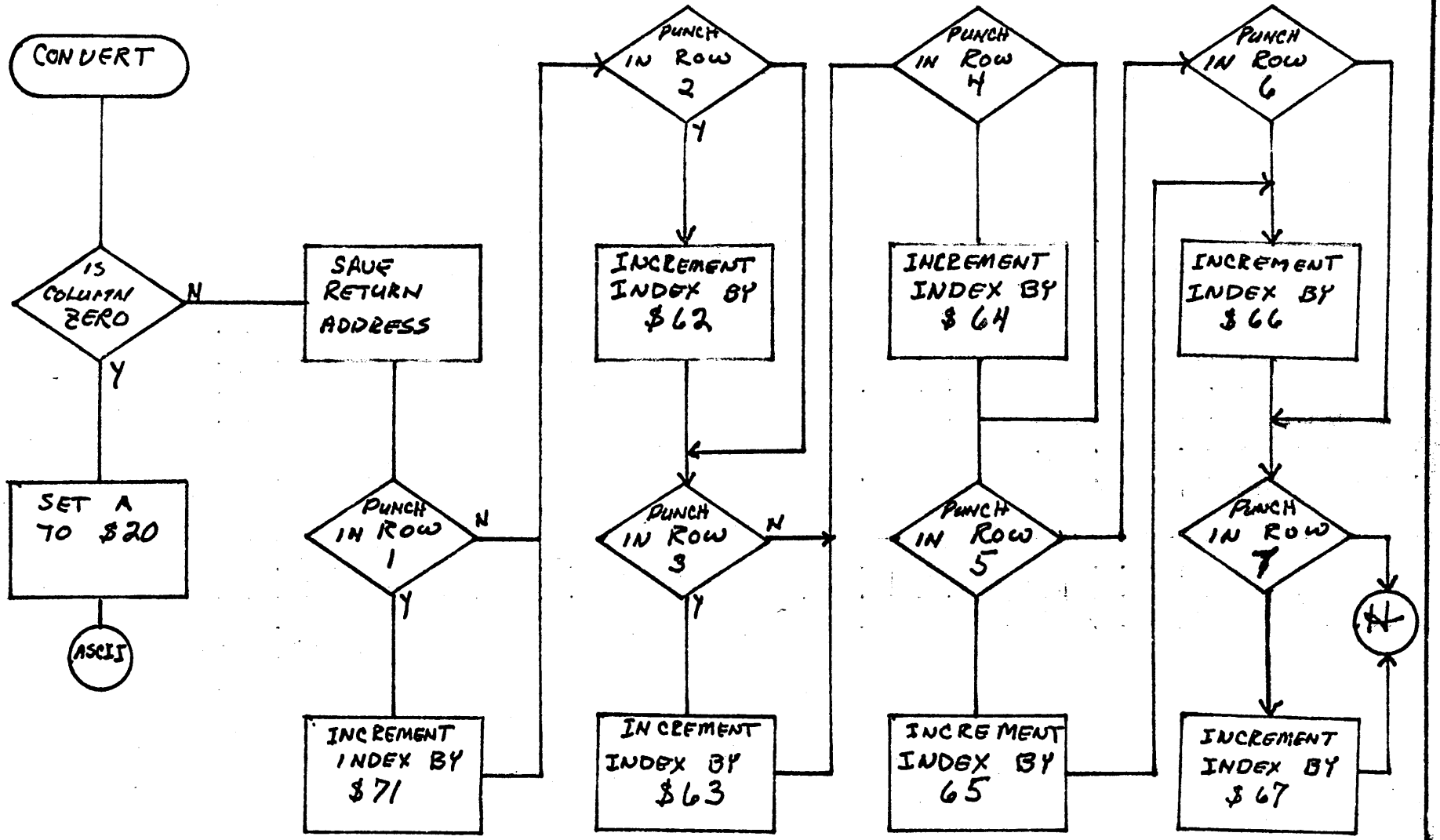
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR			
FLOWCHART ☐	NON-BUFFERED	PAGE 8 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY [illegible]	DATE	TASK NAME			

MAR 1971

49.30

A1385 (FORMER A127-1)

PRINTED IN U.S.A.



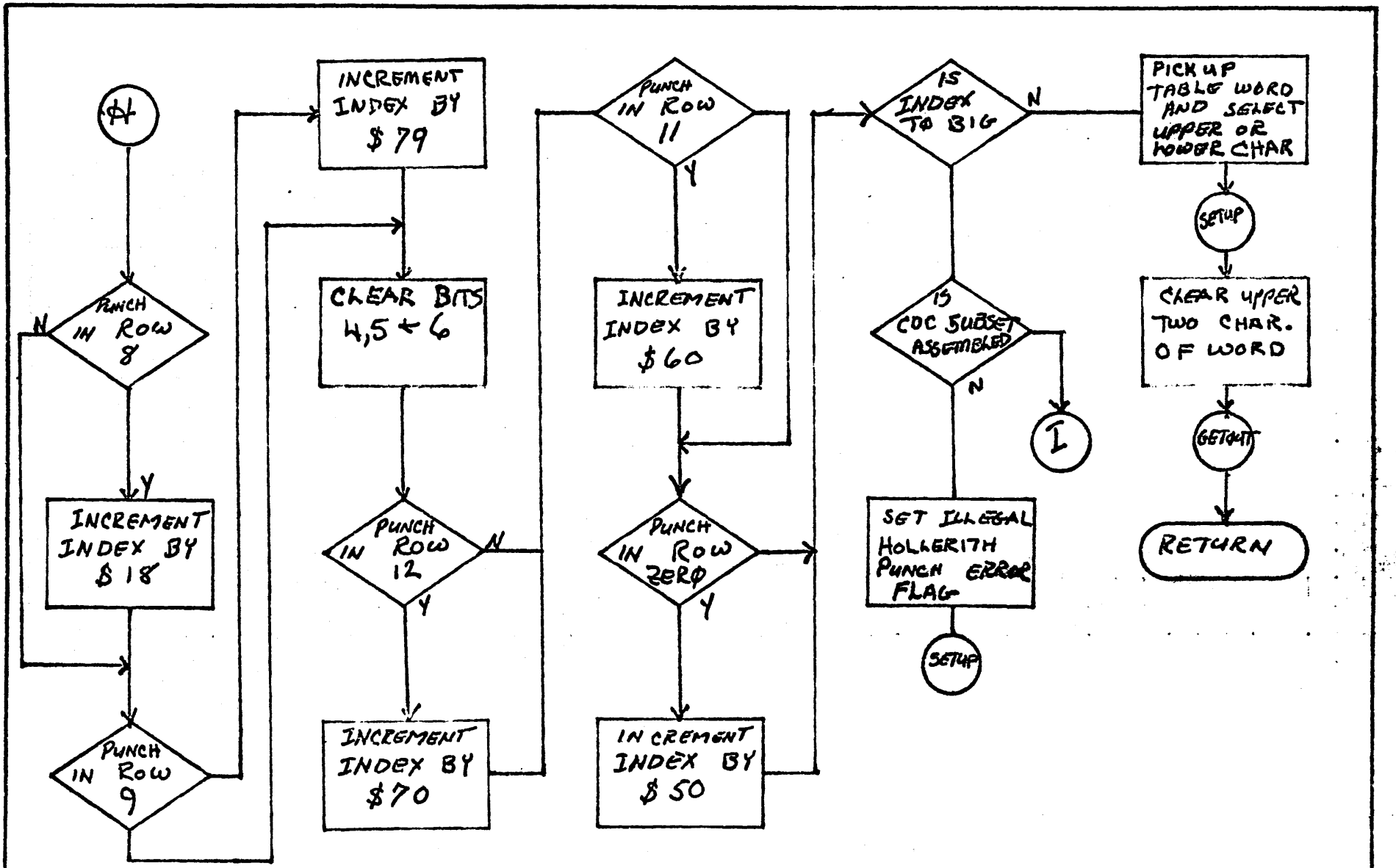
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR			
FLOWCHART ☐	NON-BUFFERED	PAGE 9 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 3 1971

49.33

AA1388 (FORMERLY CA127-1)

PRINTED IN U.S.A.

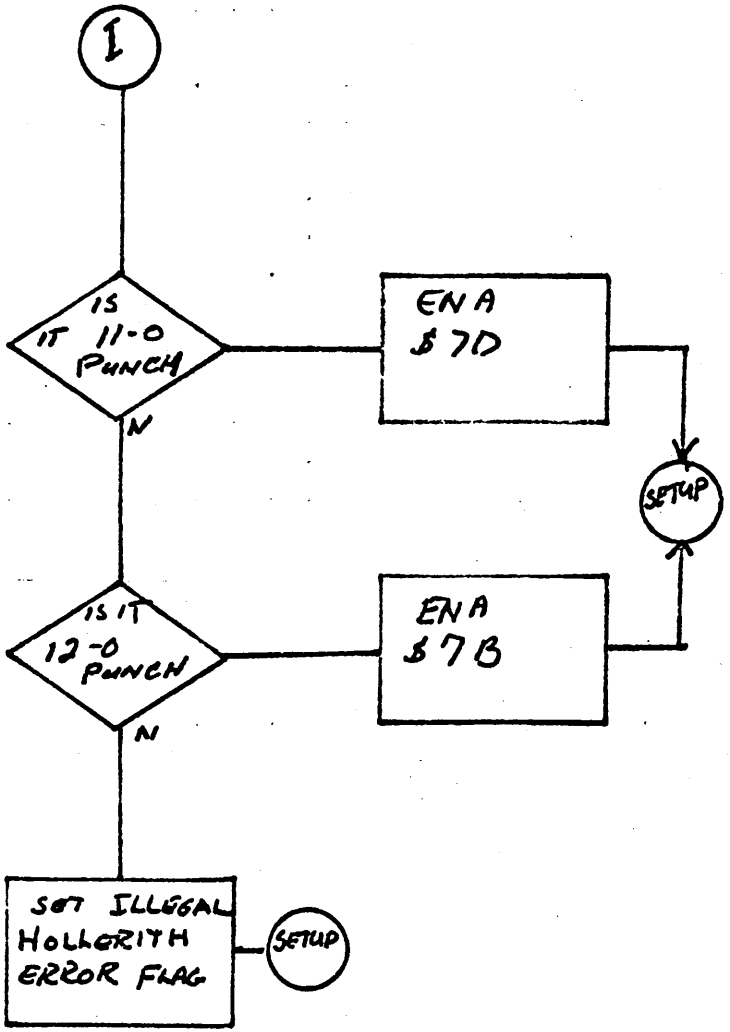


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO	REV	APPROVED / DATE
	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR		
	NON-BUFFERED	PAGE 10 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

49.32

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1726-405 DRIVER		PROJECT MGR		
	NON-BUFFERED	PAGE 11 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY [illegible]	DATE	TASK NAME		

MAR 5 1971

49.33

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

50.0 1729-2 CARD READER DRIVER

50.1 FUNCTION

The 1729-2 Card Reader Driver is a standard software driver which will operate under the 1700 MSOS 3.0 Operating System. Compatability is such that the 1729-2 driver can be used to perform read operations on the 1728/430.

50.2 ENTRY POINTS

IN1729	initiator entry
CN1729	continuator entry
EX1729	error entry

50.3 EXTERNALS

ALTDEV	Alternate Device Handler
MAKEG	

50.4 DRIVER DESCRIPTION

At the entry point EX1729, time expired, error code zero is set and the Alternate Device Handler is entered.

At the initiator entry, routine FNR is entered to see if a request is stacked. If not, exit is made to the dispatcher. If there is a request, the checksum accumulator, packing cycle indicator, hollerith error flag, and sequence check words are cleared, and exit is then made to symbol FEEDCD {feed a card}.

At symbol E0FRD, the error bit is set in the request status physical equipment table word. The offset bit is set to cause the end of file card just read to be offset. Exit is then made to symbol DONE. This will cause entry to be made to the completion routine with the error indicator set without first going to the Alternate Device Handler.

Routine CNVRT is a subroutine to convert 12-bit hollerith columns to ASCII. It does this by computing an index to a table of ASCII codes. Punches in columns 2 through 7 add $62 through $67 to the index respectively. A punch in columns 1 and 9 add $71 and $79 respectively. A punch in column 8 adds $18 to the index. If the resultant sum exceeds $7F an illegal Hollerith punch combination was present and the Hollerith error flag will be set. If the index is below $80, the bottom 4 bits are saved and the zone punches are then processed. If column 12 is punched, $70 is added to the index. If column 11 is punched, $60 is added, and if column zero is punched, $50 is added. If the resultant sum exceeds $7F, more than one zone punch was present in the column and the punch combination is reported to be illegal and the Hollerith error flag is set as before. Whenever an illegal Hollerith punch is detected the ASCII representation for a question mark(?) is set into that column. By doing so the user is able to complete his read, identify the error and continue if he desires. The Hollerith error flag is processed at symbol TOMAKQ. If no error is detected during the conversion the low order 6 bits of the index are used to access the ASCII character table and the conversion is complete.

At entry point CN1729, the continuator or interrupt time entry, the base address of the physical device table, is placed in the index register and status is read. If alarm error is set, an error is reported via error code 2. If it is a data interrupt a column in input and stored in the buffer, and procedure EXIT is entered.

At procedure EXIT, the buffer address in incremented, the interrupt on data, EOP, and alarm is selected. Exit is then made to the dispatcher to await the next interrupt.

If the interrupt was an end of operation (EOP) a jump is made to symbol NOTALR. At NOTALR the buffer address is re-initialized, and a check made to determine whether the operation is a formatted or unformatted READ. If it is a formatted READ procedure FRDBIN is entered. If unformatted ASCII, procedure ASCII is entered. If unformatted binary, procedure RDBIN is entered.

At procedure FRDBIN, a word is obtained from the buffer via subroutine GETWRD. If this is the first card of the request, the card is checked to see if this is an end of file. If so, procedure EOFRD is entered. If not, the card is checked to see if it contains a 7,9 punch in column one. If it does, procedure FRDB2 is entered. If not, and the request is ASCII, procedure ASCII is entered. If not, and it is not the first card, the 7,9 punch is missing and is reported via error code 12 to the Alternate Device Handler. If it is the first card, the mode is set to ASCII and procedure ASCII is entered.

At procedure FRDB2 the word just obtained is checked to see if checksum override is indicated. If so, the indicator is set. The sequence number is then checked. If correct and this is not the first card of a record; this fact is reported to the Alternate Device Handler with error code 9 set. If it is the first card the current sequence number is then set to the one just read.

The sequence number is then incremented Modulo 256, and another word is obtained from the buffer via subroutine GETWRD. If this is the first card of the record, this word should be the complemented record length. If non-negative, this fact is reported to the Alternate Device Handler with error code 10. If it is negative, it is stored in a PHYSTAB location. At this point FRDB7 the loop begins which gets the rest of the word in the record.

The purpose of subroutine GETWRD is to construct successive 16 bit words from the 12 bit columns stored in the buffer. To perform this, 4 columns are combined to produce three words. This is performed in three successive cycles as words are requested. Upon completion of each cycle, the next cycle is set for future entry.

At procedure DONE, the interrupts are cleared and the status is saved. Subroutine MAKEQ is then entered, the diagnostic clock is set negative, and routine FNR is re-entered to see if another request is waiting.

At routine ERROR, the error code and logical unit number are combined for output by the Alternate Device Handler, and bits 14 and 15 of the request status PHYSTAB word are cleared. The controller is cleared, status is saved, and subroutine MAKEQ is entered. Upon return, exit is made to the Alternate Device Handler.

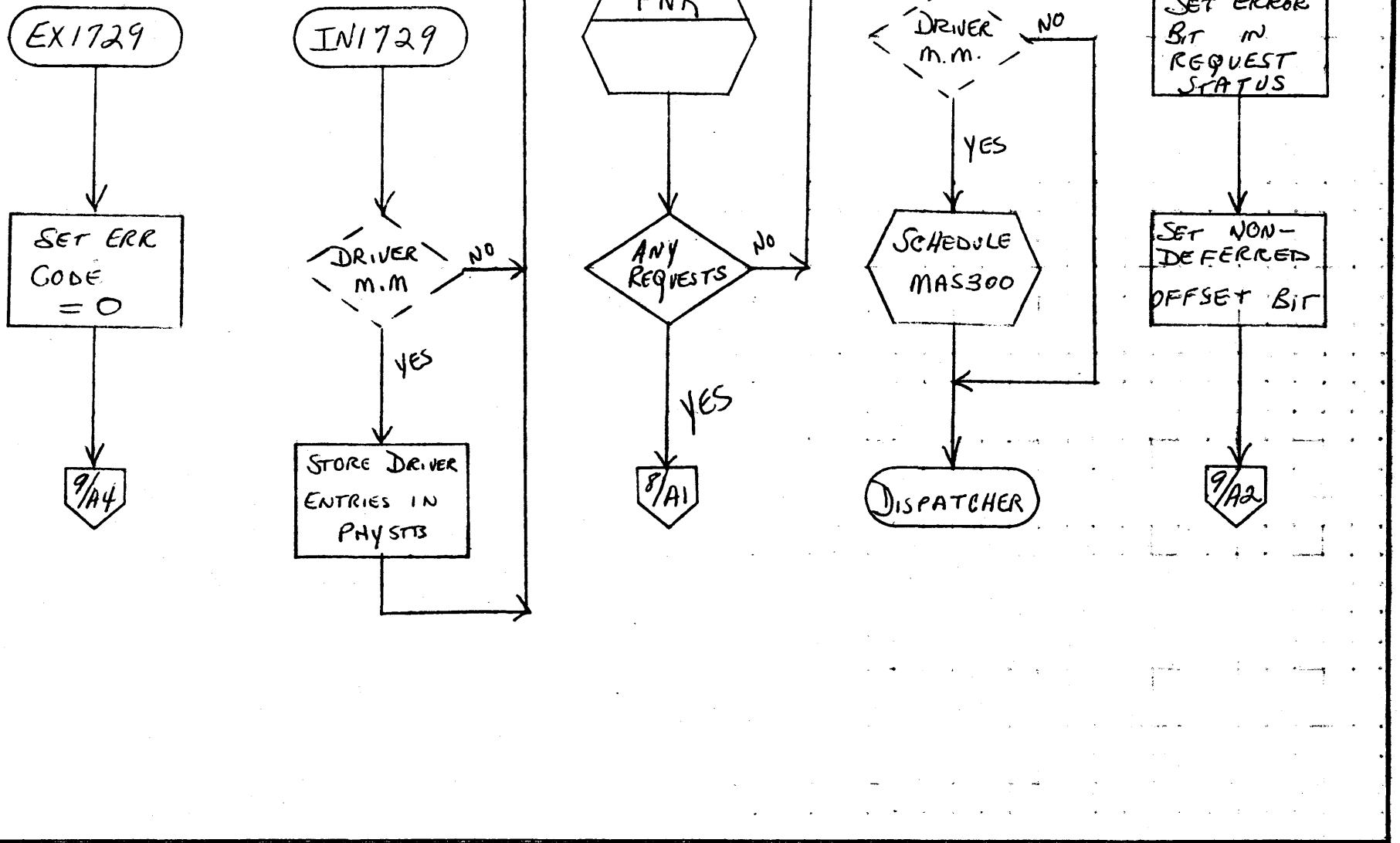
The purpose of subroutine MAKEW is to set the address of the next word of the buffer if a READ request does not get as many words as requested. Other completion housekeeping common to normal and abnormal termination is performed by this subroutine.

At label FRDB7, if the remaining record length is zero, exit is made to label FRDB8 to see if the checksum is correct. If not, the core address is checked to see if the number of words requested has been read. If so, record length is incremented and checked for zero. If not, reading continues until the entire record has been read, reading continues until it has, or the entire record read. At this time, the complement of the checksum is read and the sum checked for zero. If not, and the checksum override was not indicated, error code four is reported to the Alternate Device Handler. If zero or checksum override was indicated, exit is made to procedure DONE.

At procedure FEEDCD, feed a card, the current buffer address is set back to the beginning of the buffer. The offset indicator is then checked. If set, it is first reset to zero then the offset gate is set. The controller is cleared, and interrupts selected on data, alarm, and end of operation. Exit is then made to the Dispatcher.

At procedure FDBIN, the core location is checked against LAST+1. If equal, one column only is to be read and procedure ONCHR is entered. If not a word is obtained from the buffer via subroutine GETWRD. It is stored into core and the core location is incremented. This is repeated until the number of words requested has been read, at which time procedure DONE is entered. At procedure ONECHR, the low order four bits is cleared from the word just read and the word is stored into core. Procedure DONE is then entered.

At procedure ASCII, read ASCII, a column is picked up from the buffer, converted to ASCII via subroutine CONVRT, shifted left 8 bits, and stored into the indicated core location. If this was a request for one character, procedure DONE is entered. If not another column is picked up, converted, and stored into the lower 8 bits of the core location. The core location is then incremented. If the number of words requested has been read, procedure DONE is entered. If not, the buffer address is incremented. If the buffer has been emptied and this is a formatted request, procedure DONE is entered. If this is an unformatted request procedure FEEDCD is entered.



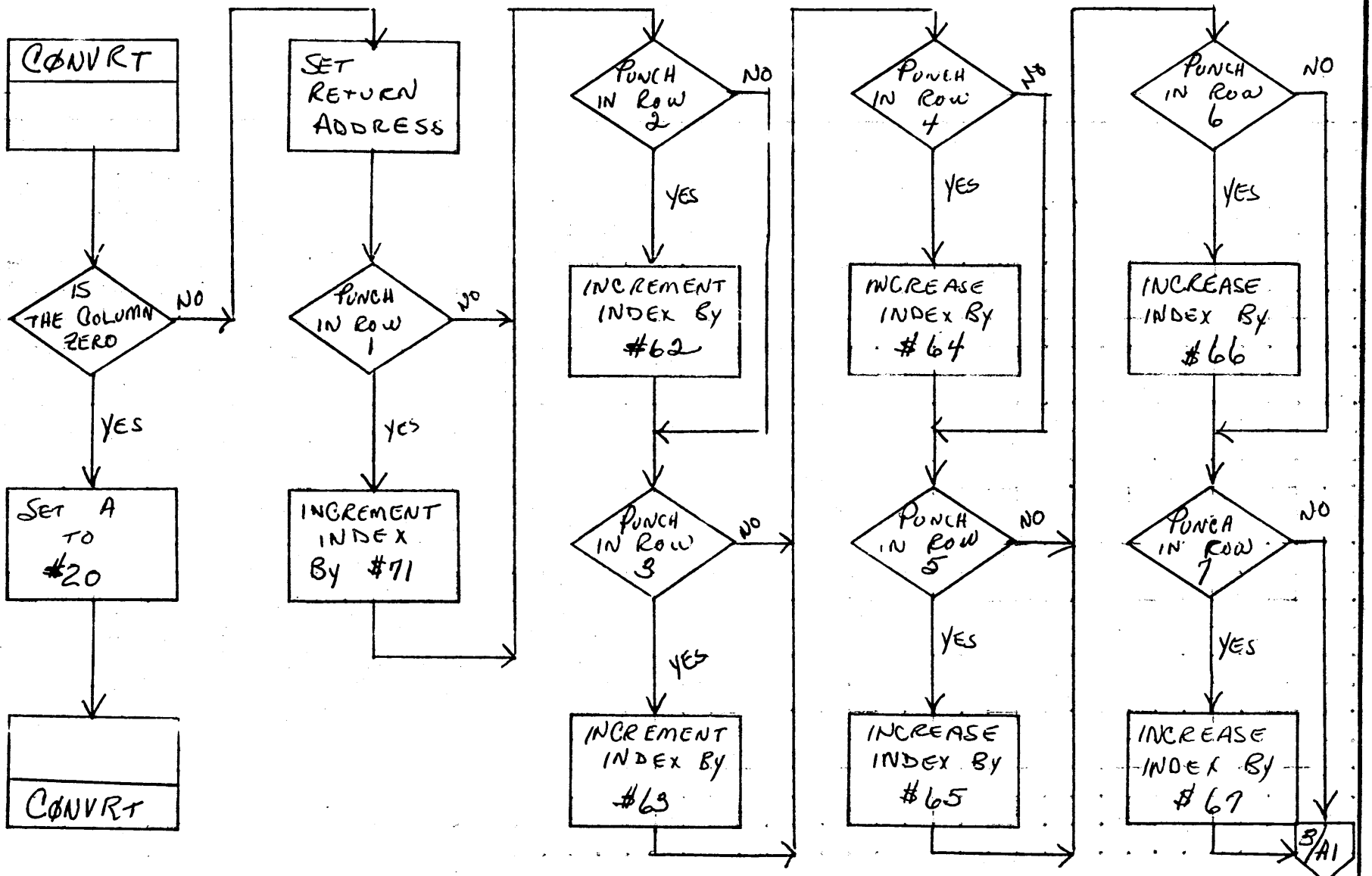
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1729-2 DRIVER	PROJECT MGR			
	PAGE 1 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

50.6

AA1385 (FORMERLY CA127-1)

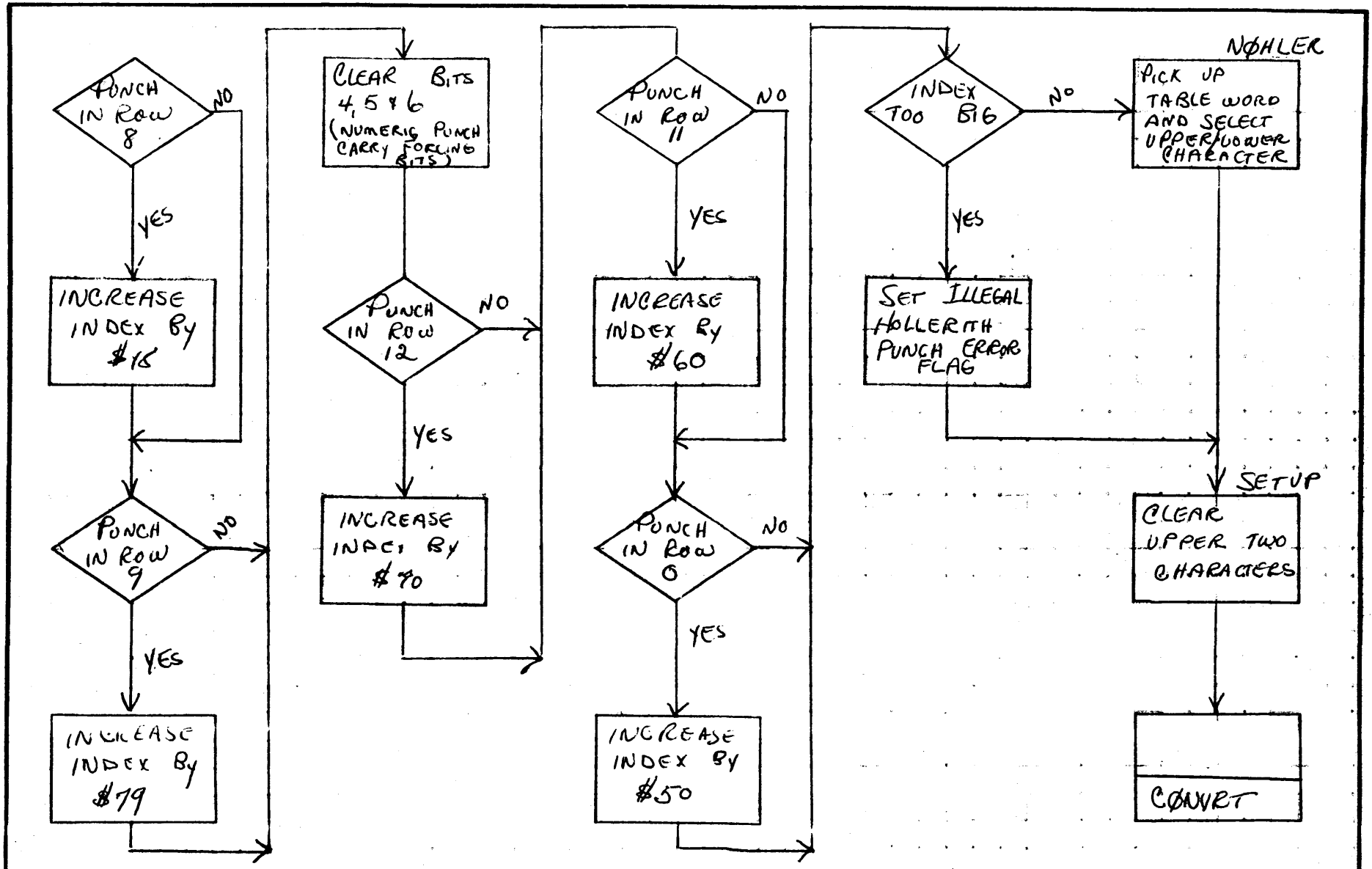
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1729-2 DRIVER		PROJECT MGR.			
		PAGE 2 OF 11	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

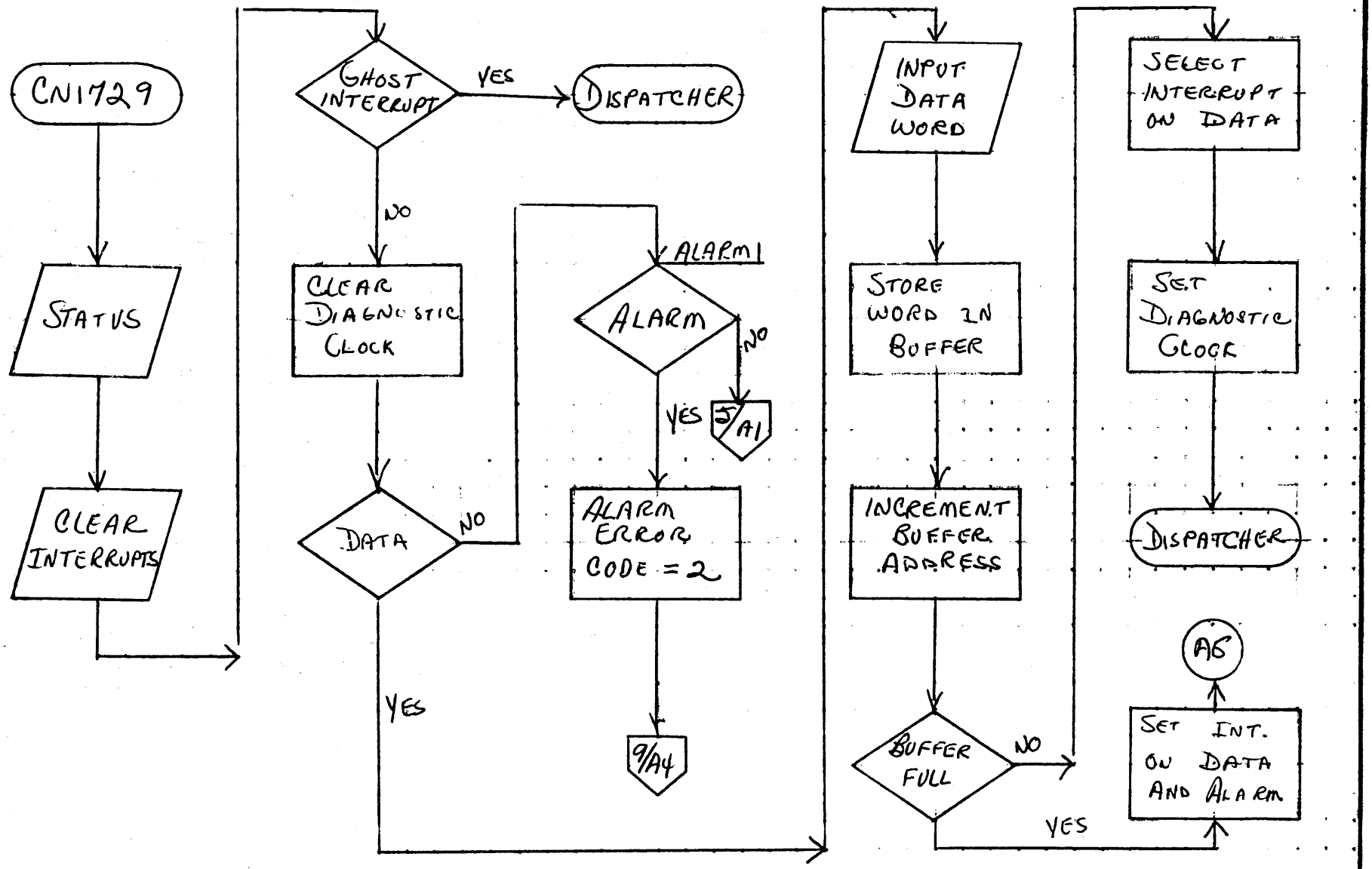
50.7



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1729-2 Driver		PROJECT MGR.		
		PAGE 9 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

50-8



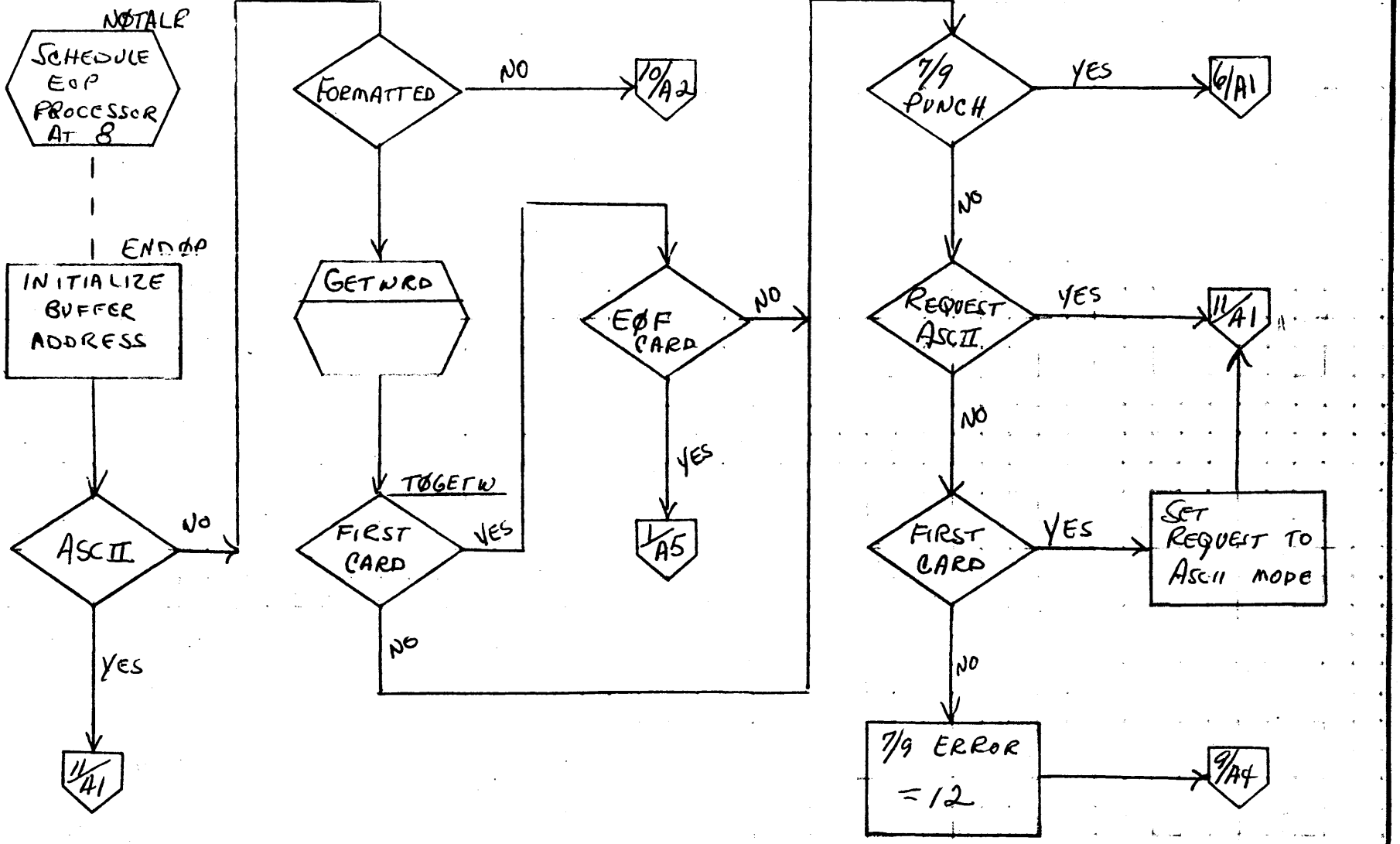
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1729-2 DRIVER	PROJECT MGR.			
	PAGE 4 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

50.9



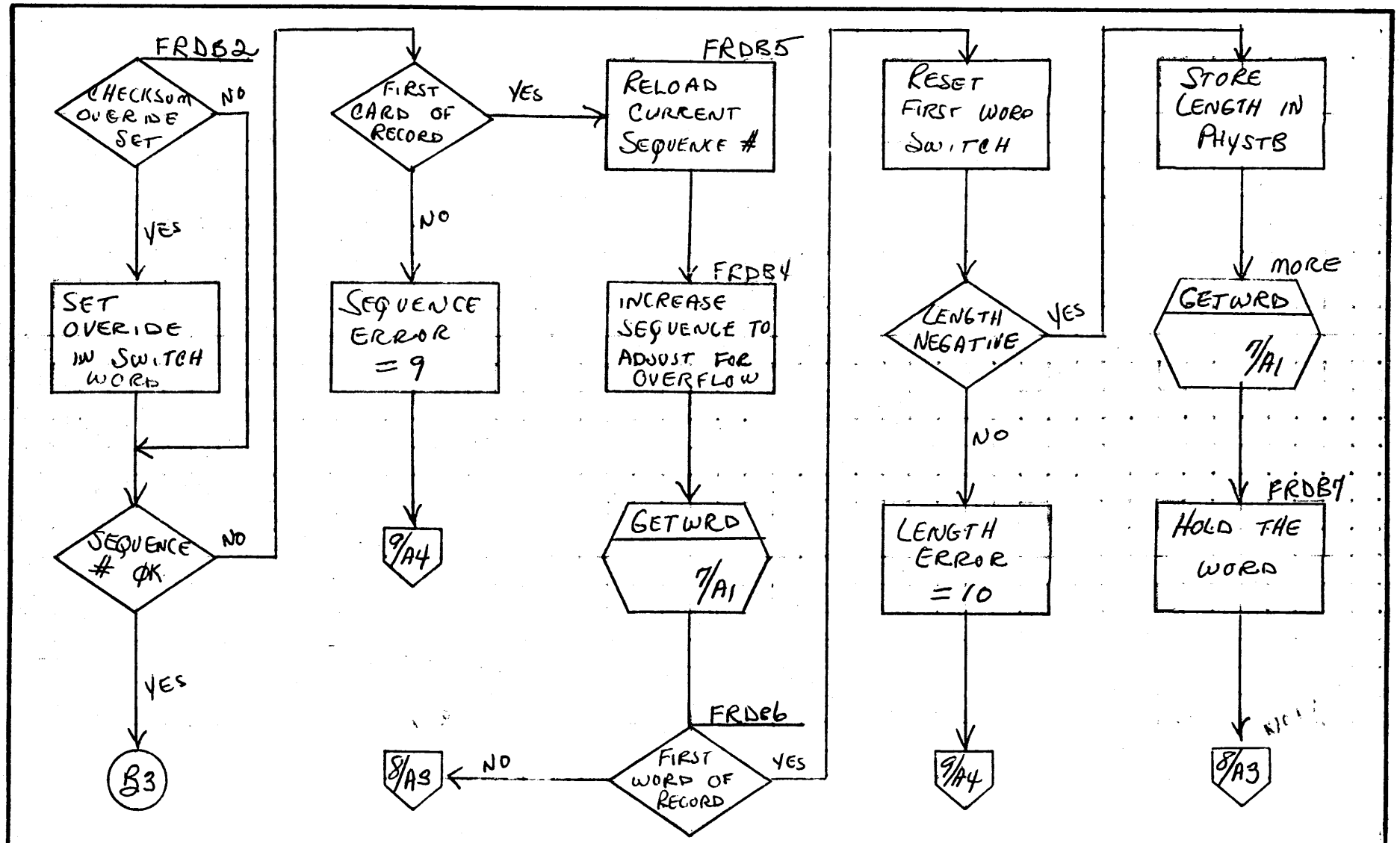
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1729-2 DRIVER		PROJECT MGR.			
FLOWCHART ☐		PAGE 5 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

50.10

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



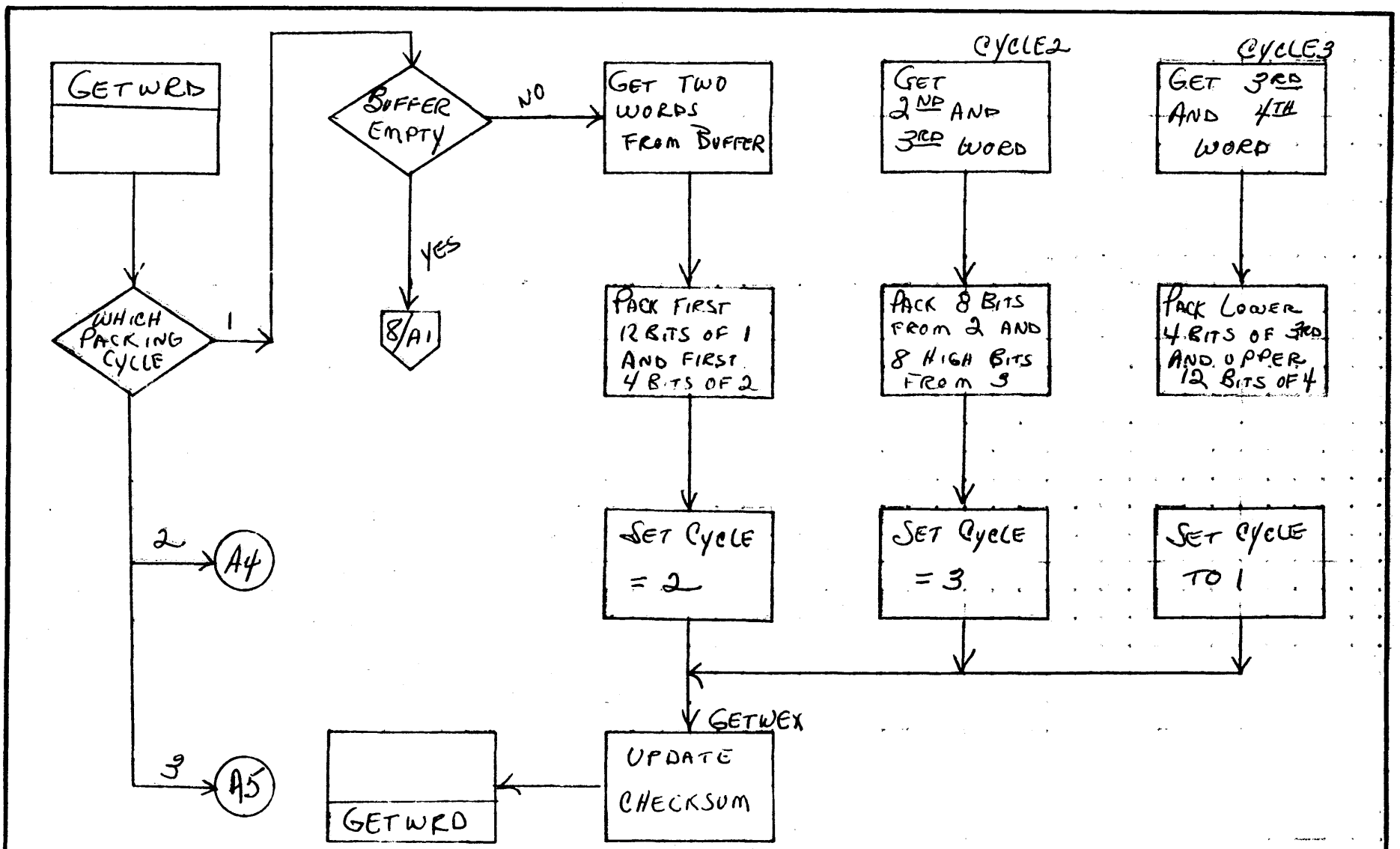
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1729-2 DRIVER	PROJECT MGR.			
	PAGE 6 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

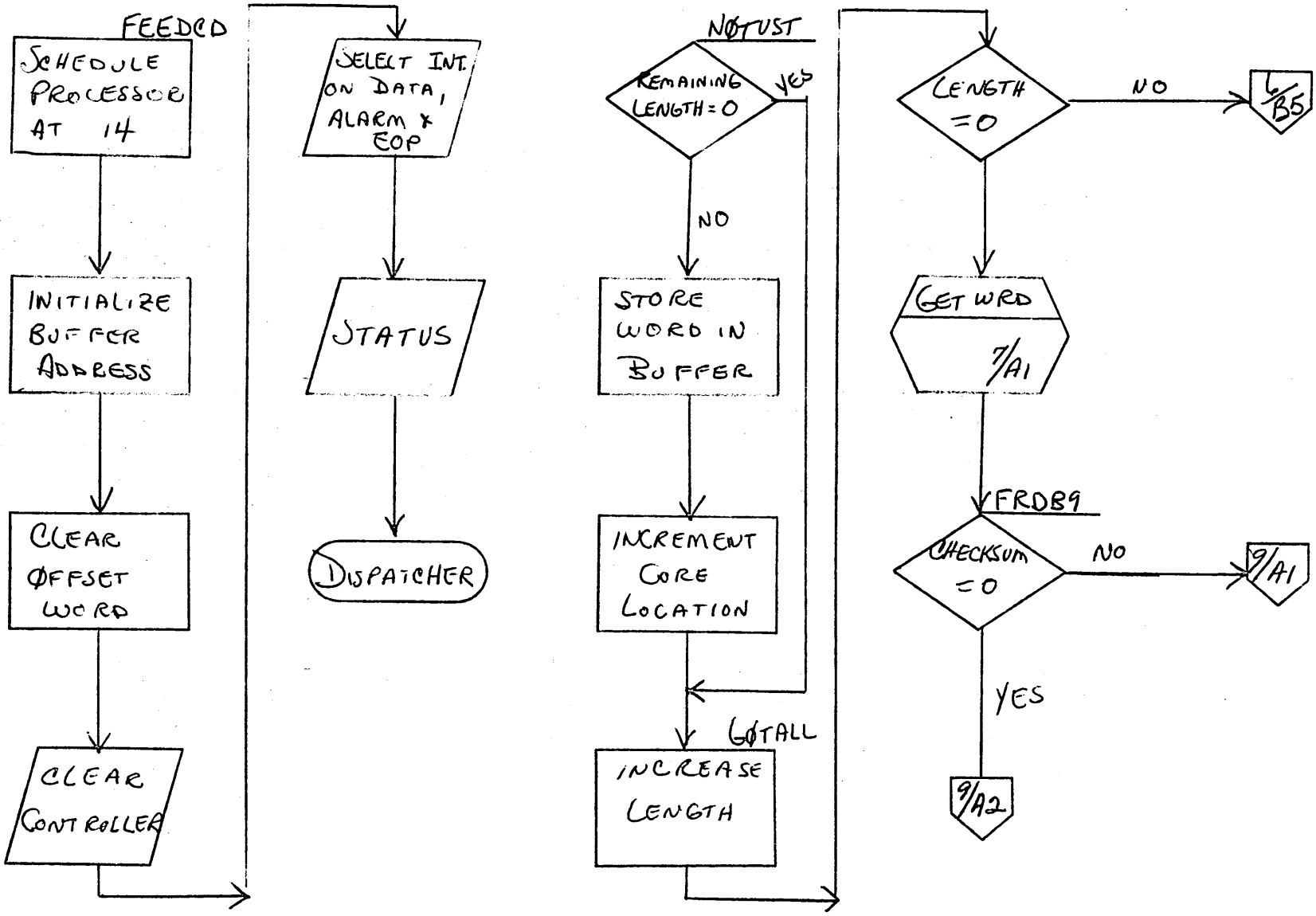
50.11



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: INLS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1729-2 DRIVER		PROJECT MGR.			
FLOWCHART ☐	PAGE 7 OF 11		PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

50-12

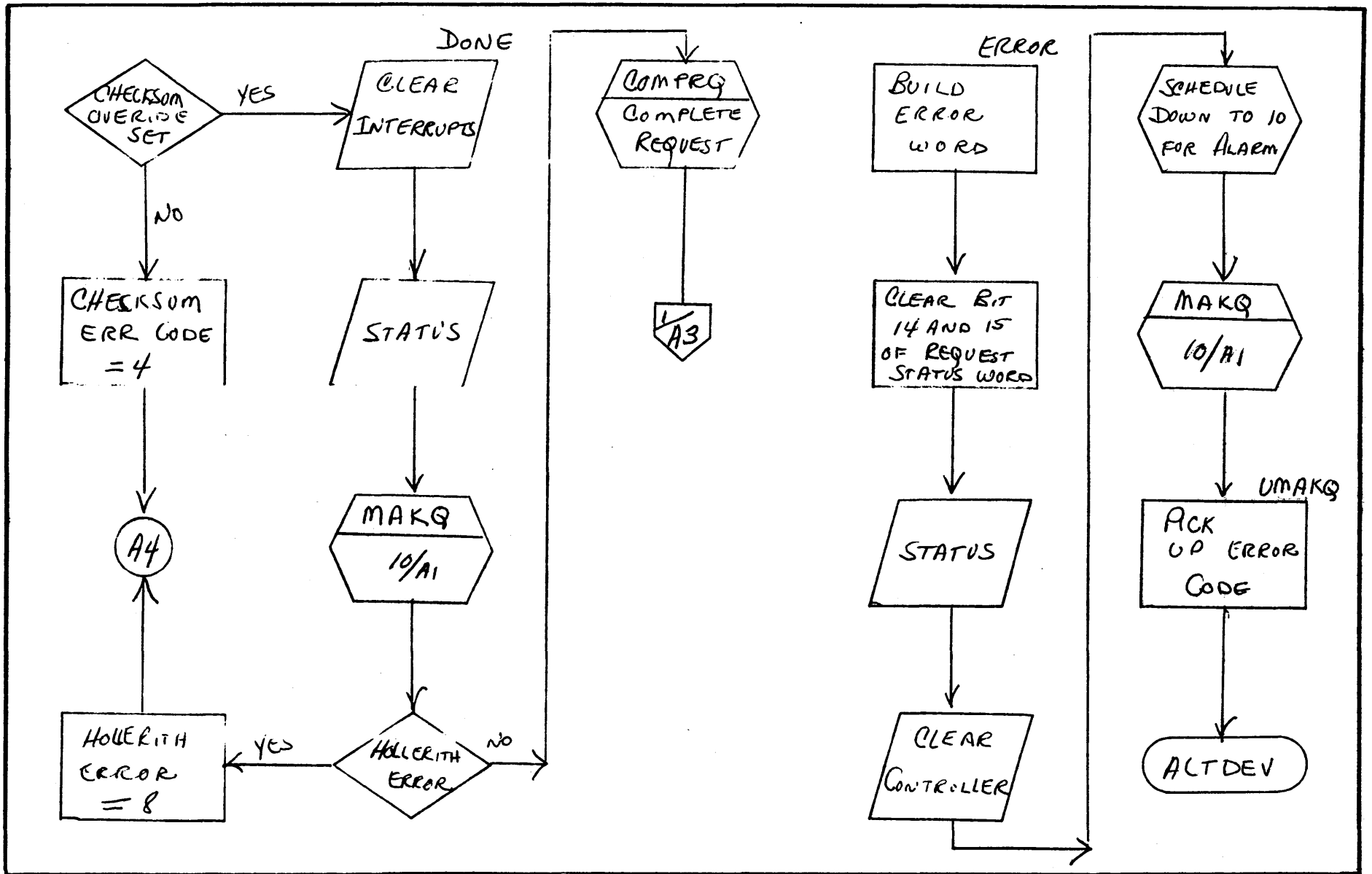


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700	PROJECT NO	REV	APPROVED	DATE
	DOCUMENT TITLE 1729-2 DRIVER	PROJECT MGR			
	PAGE 8 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

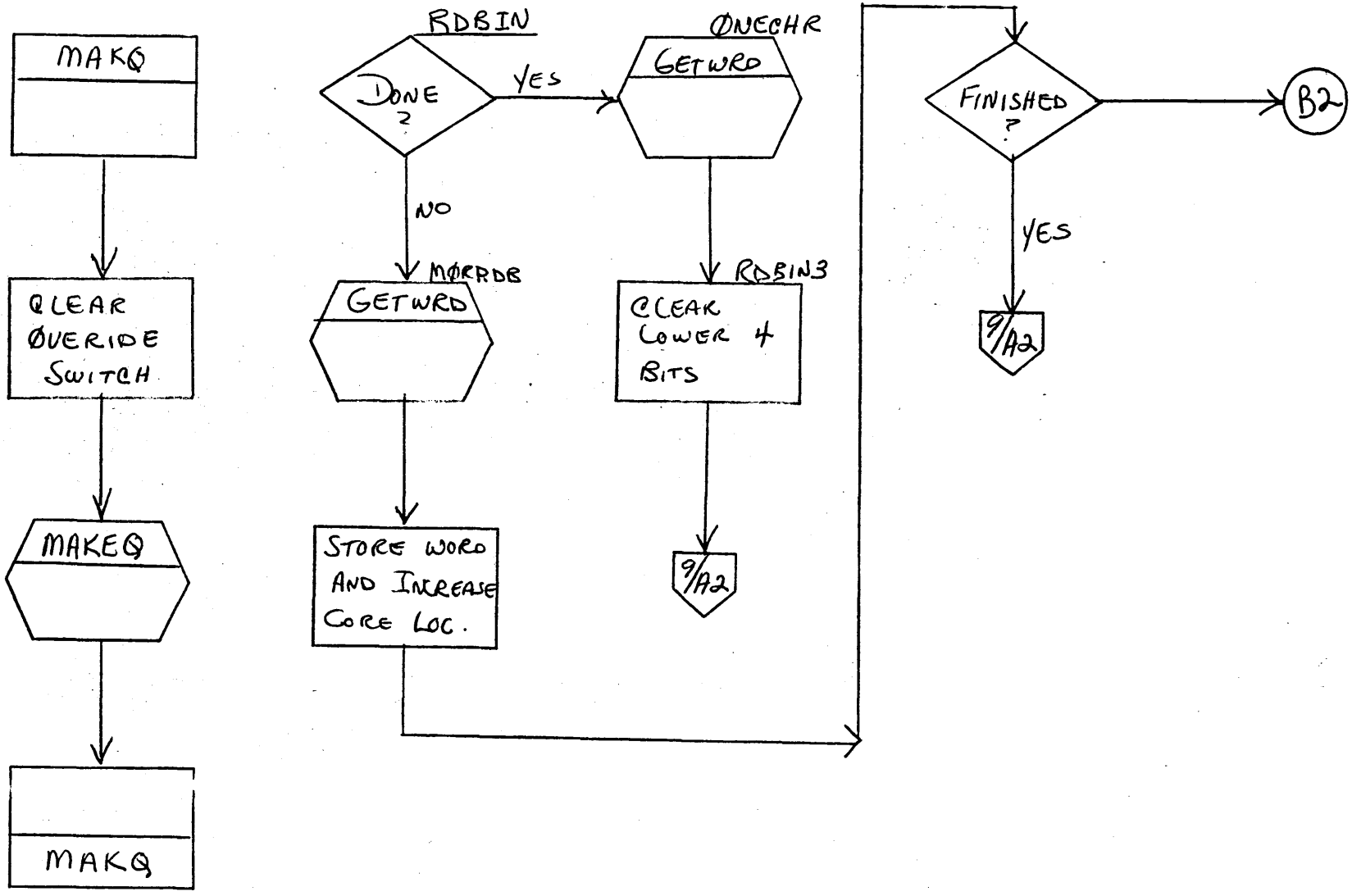
50.13



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE / FLOWCHART / DECISION TABLE / OTHER	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1129-2 DRIVER		PROJECT MGR		
	PAGE 9 OF 11		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

50.14

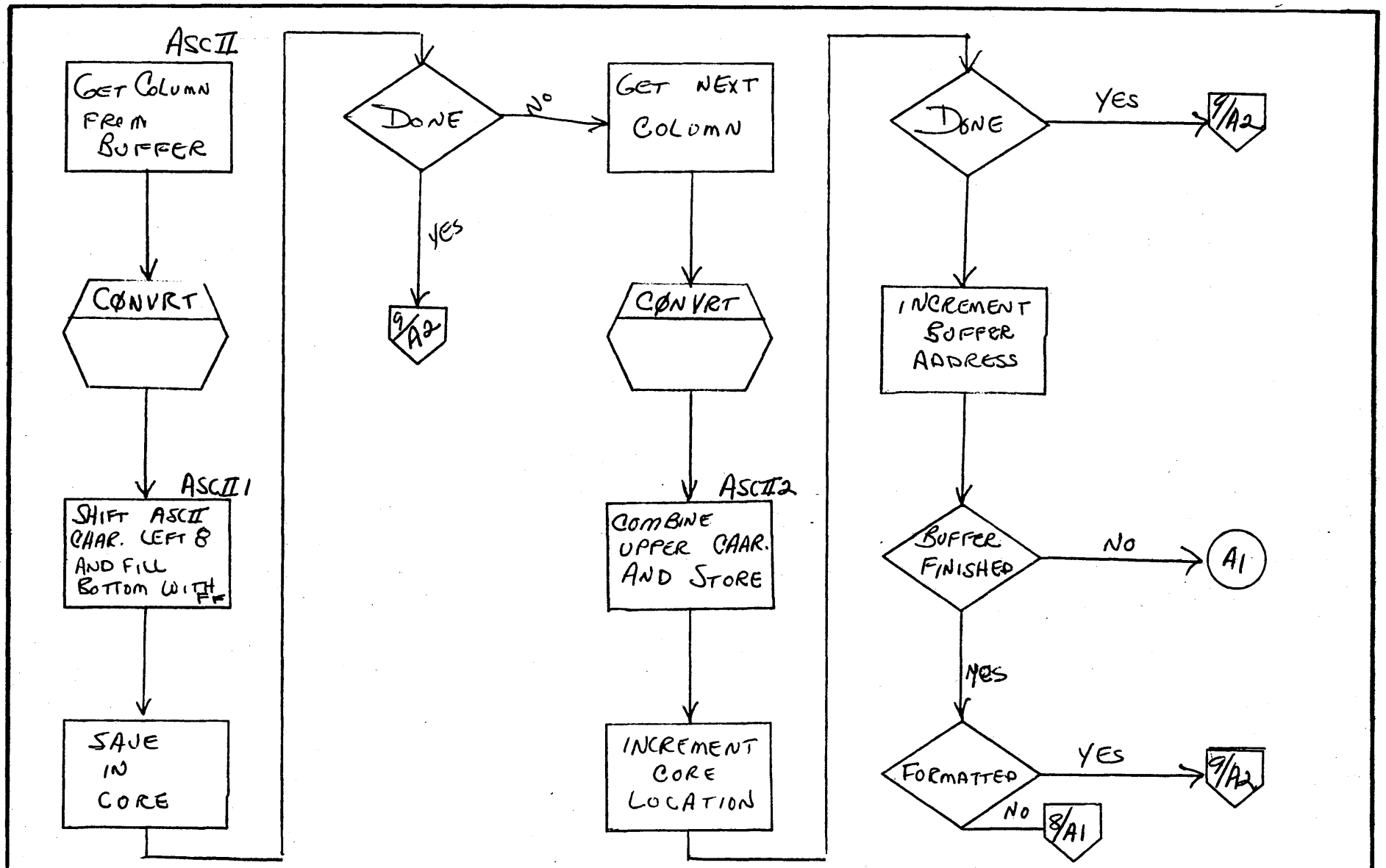


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1729-2 DRIVER		PROJECT MGR.		
		PAGE 10 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

50.15

AA-385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1729-2 DRIVER		PROJECT MGR.		
		PAGE 11 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

50.15

51.0 1728/430 Reader/Punch Driver

51.1 Function

To operate the 1728/430 reader and punch.

51.2 Entry Points

EX1728: Time expired error entry.

IN1728: Initiator entry.

CN1728: Continuator or interrupt time entry.

FF1728: Return to driver after sequence logging request.

CM1728: Completion address for sequence logging.

51.3 Externals

ALTDEV: Alternate Device Handler

51.4 Driver Description

At the initiator entry, routine FNR is entered to see if a request is stacked. If not, exit is made to the dispatcher. If ther is a request, the checksum accumulator, packing cycle indicator, and tape motion control parameter are cleared. The current buffer address set to the beginning of the buffer. The request code is checked to see if this is a motion control request (for punching end of file cards). If so, the three motion codes are saved and symbol M0RMC is jumped to process them. If not, a check is made to see if an attempt is bsing made to switch from reading to punching or vice versa. If so, the connect code is changed to reference the appropriate station and bit 15 of Equipment Table Word 14 is checked to see if the attempted switch is to be reported as an error. If so, the Alternate Device Handler is entered with error code 11 set. If not, or the operator repeats the request, the request code is checked to see whether this is a

READ or a WRITE request. If it is a WRITE request, symbol PUNCH is jumped to. If it is a READ request, symbol FEEDCD (feed a card) is jumped to.

At the entry point EX1728, time expired, error code zero is set and the Alternate Device Handler is entered.

At symbol EOFRD, the error bit is set in the request status physical equipment table word. The non-deferred offset bit is set to cause the end of file card just read to be offset. Exit is then made to symbol DONE. This will cause entry to be made to the completion routine with the error indicator set without first going to the Alternate Device Handler.

Routine CONVRT is a subroutine to convert 12-bit Hollerith columns to ASCII. It does this by computing an index to a table of ASCII codes. Punches in columns 2 through 7 add $62 through $67 to the index respectively. Punches in columns 1 and 9 add $71 and $79 respectively. A punch in column 8 adds $ 8 to the index. If the resultant sum exceeds $7F, an illegal Hollerith punch combination was present and error code 8 is passed to the Alternate Device Handler. If the index is below $80, the bottom 4 bits are saved and the zone punches are then processed. If column 12 is punched, $70 is added to the index; if column 11 is punched, $60 is added, and if column zero is punched, $50 is added. If the resultant

sum exceeds $7F, more than one zone punch was present in the column and the punch combination is reported to be illegal as before. If not, the low order 6 bits of the index are used to access the ASCII character table and the conversion is complete.

At entry point CN1728, the continuator or interrupt time entry, the base address of the physical equipment table, is placed in the index register and status is read. If alarm status is set, echo check status is checked. If not set, the error is reported via error code 2. If set, and this is a WRITE request, the offset bit is set. Regardless of whether it is a READ or WRITE, the error is then reported via error code 7. If it is a data interrupt and a READ request is in progress, the column is input and stored into the buffer, and procedure EXIT is entered. If it is a WRITE request, a word is obtained from the buffer and is output to the punch station. Procedure EXIT is then entered.

At procedure EXIT, the buffer address is incremented, the interrupt is cleared, and interrupt on data, EOP, and alarm is selected. Exit is then made to the dispatcher to await the next interrupt.

If the interrupt was end of operation and a WRITE request is in progress, the request is checked to see if it is formatted. If so, procedure FORMOP is entered. If unformatted and the core

location is equal to LAST+1, procedure DONE is entered. If not, procedure PUNCH is entered.

At procedure FORMOP, the request is checked to see if it is ASCII. If so, procedure DONE is entered. If not, the switch is checked to see if the checksum has been punched. If so, procedure DONE is entered. If not, procedure PUNCH is entered.

If it is a formatted READ, procedure FRDBIN is entered. If unformatted ASCII, procedure ASCII is entered. If unformatted binary, procedure RDBIN is entered.

At procedure FRDBIN, a word is obtained from the buffer via subroutine GETWRD. If this is the first card of the request, the card is checked to see if it is an end of file. If so, procedure EOFRD is entered. If not, the card is checked to see if it contains a 7,9 punch in column one. If it does, procedure FRDB2 is entered. If not, and the request is ASCII, procedure ASCII is entered. If not, and it is not the first card, the 7,9 punch is missing and is reported via error code 12 to the Alternate Device Handler. If it is the first card, the mode is set to ASCII and procedure ASCII is entered.

At procedure FRDB2, the word just obtained is checked to see if checksum override is indicated. If so, the indicator is set. The sequence number is then checked. If incorrect, and this is

not the first card of a record, this fact is reported to the Alternate Device Handler with error code 9 set. If it is the first card, the message SQ,AA,BB is output on the list device where AA is the expected sequence number and BB is the one actually read. The current sequence number is then set to the one just read.

The sequence number is then incremented modulo 256, and another word is obtained from the buffer via subroutine GETWRD. If this is the first card of the record, this word should be the complemented record length. If non-negative, this fact is reported to the Alternate Device Handler with error code 10. If it is negative, it is stored in a PHYSTAB location. At this point, FRDB7, the loop begins which gets the rest of the words in the record.

The purpose of subroutine GETWRD is to construct successive 16 bit words from the 12 bit columns stored in the buffer. To perform this, 4 columns are combined to produce 3 words. This is performed in three successive cycles as words are requested. Upon completion of each cycle, the next cycle is set for future entry.

At procedure DONE, the request is checked to see if it is a motion control request with write end of file codes remaining

to be processed. If so, procedure WEOF is entered. If not, interrupts are cleared and status is saved. If sequence logging is in effect, exit is made to the dispatcher to await completion. If not, subroutine MAKEQ is entered, diagnostic clock time is set negative, and routine FNR is re-entered to see if another request is waiting.

At routine WEOF, the sequence base is reset and codes set up to punch one word unformatted with the word to be punched the column one configuration for an end of file card. The deferred offset switch is set and the punch request is begun.

At routine ERROR, the error code and logical unit number are combined for output by the Alternate Device Handler, and bits 14 and 15 of the request status PHYSTAB word are cleared. The controller is cleared, status is saved and subroutine MAKEQ is entered. Upon return, exit is made to the Alternate Device Handler.

The purpose of subroutine MAKEQ is to set the address of the next word of the buffer if a READ request does not get as many words as requested. Other completion housekeeping common to normal and abnormal termination is performed by this subroutine.

At label FRDB7, if the remaining record length is zero, exit is made to label FRDB8 to see if the checksum is correct. If not,

the core address is checked to see if the number of words requested has been read. If so, the record length is incremented and checked for zero. If not, reading continues until the entire record has been read. If the number of words requested has not been read, reading continues until it has, or the entire record read. At this time, the complement of the checksum is read and the sum checked for zero. If not, and checksum override was not indicated, error code 4 is reported to the Alternate Device Handler. If zero or checksum override was indicated, exit is made to procedure DONE.

At procedure FEEDCD, feed a card, the current buffer address is set back to the beginning of the buffer and the deferred offset indicator is checked. If not set, the current offset indicator is checked. If set, it is first reset, then the offset gate is set. Interrupts are cleared and interrupt on data, EOP, and alarm selected. If data status is up and this is a punch operation, exit is made to the point in code to punch the card. This indicates that punching of a card just read is to be performed and no feed is necessary. If data status was not up, exit is made to the dispatcher. If deferred offset is set, it is reset and offset for the next card is set. The point in code above which checks for current offset is then entered.

At procedure RDBIN, the core location is checked against LAST+1. If equal, one column only is to be read and procedure ONCHR below is entered. If not, a word is obtained from the buffer via subroutine GETWRD. It is stored into core and the core location is incremented. This is repeated until the number of words requested has been read, at which time procedure DONE is entered. At procedure ONECHR, the low order 4 bits are cleared from the word just read and the word is stored into core. Procedure DONE is then entered.

At procedure ASCII, read ASCII, a column is picked up from the buffer, converted to ASCII via subroutine CONVRT, shifted left 8 bits, and stored into the indicated core location. If this was a request for one character, procedure DONE is entered. If not, another column is picked up, converted, and stored into the lower 8 bits of the core location. The core location is then incremented. If the number of words requested has been read, procedure DONE is entered. If not, the buffer address is incremented. If the buffer has been emptied and this is a formatted request, procedure DONE is entered. If this is an unformatted request, procedure FEEDCD is entered.

The purpose of subroutine PCNVRT is to convert ASCII characters to 12 bit Hollerith columns. The ASCII character is checked to see if it is greater than or equal to $20 and less than $60.

If not, it is converted to zero (blank column). If it is in range, a direct table look-up is performed to get the 12 bit Hollerith equivalent. The column is then stored into the buffer, the buffer address is incremented, and return is made to the caller.

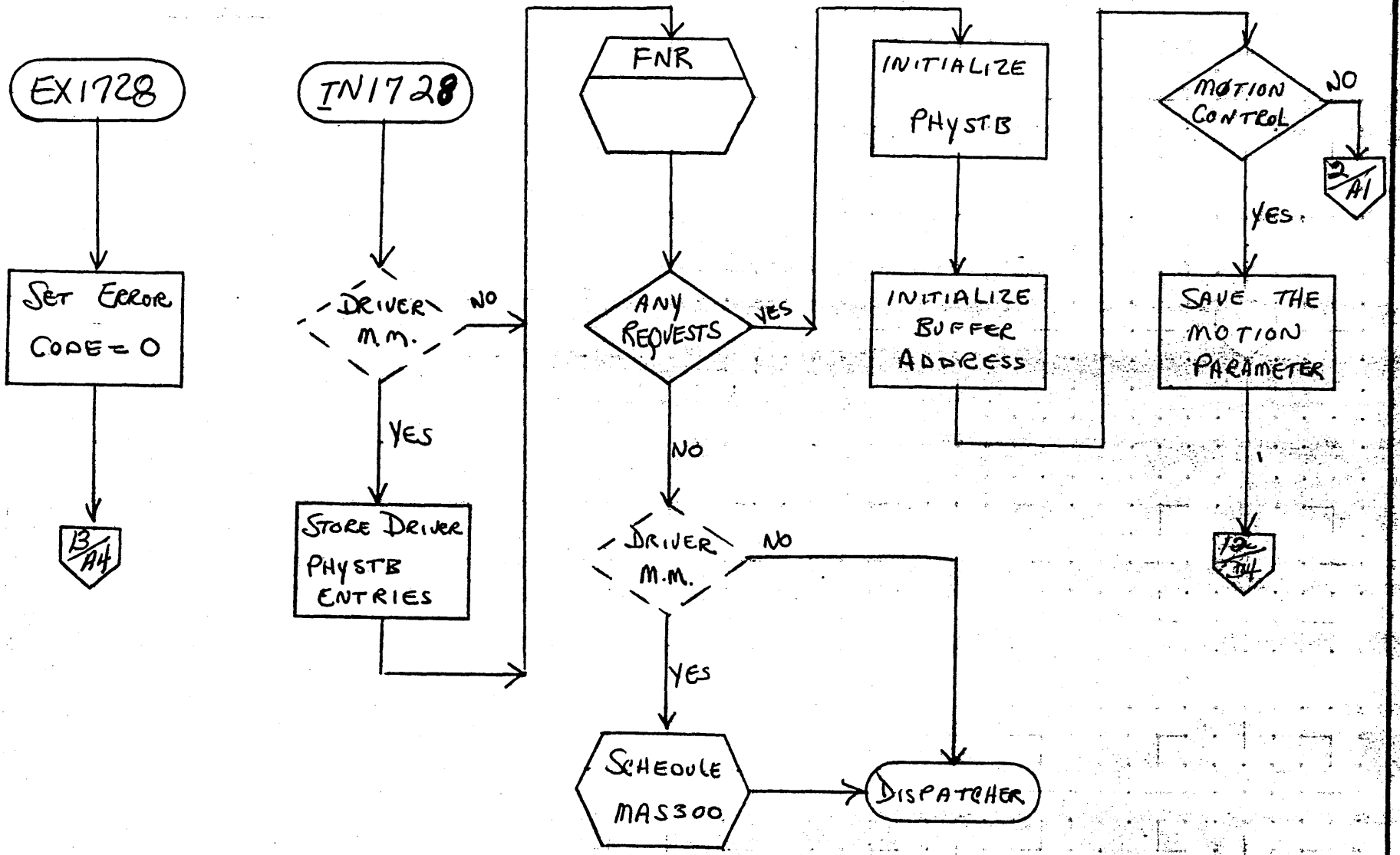
The purpose of subroutine PUT is to separate the 16 bit binary input words into successive 12 bit columns and store them into the buffer for punching. At entry, the checksum is updated and the unpacking cycle is determined. At cycle 1, the high order 12 bits of the input word are placed into bits 0-11 of buffer word one. The low order 4 bits are placed into bits 8-11 of buffer word two. At cycle 2, the high order 8 bits are put into bits 0-7 of buffer word two and the low order 8 bits into bits 4-11 of buffer word three. At cycle 3, the high order 4 bits are placed into bits 0-3 of buffer word three and the low order 12 bits into bits 0-11 of word four. The buffer address is then incremented by four. If the buffer is now full and the request has not been completed, the core address is incremented by one. Routine FEEDCD is then entered.

At label PUNCH, cycle 1 is set in subroutine PUT and the request is checked to see if it is ASCII. If so, exit is made to label PUNCH6. If not, the request is checked to see if it is formatted. If so, exit is made to label PUNCH3. If not, the contents of the

indicated core location are picked up and subroutine PUT is called. Upon return, the core location is compared to last location plus one. If equal, switches are set to indicate the request is formatted, this is not the first word, and the checksum is punched. This is to utilize common code in the formatted binary punch portion of the driver. If not equal, the core location is incremented and checked as before. If not equal to last location plus one, the sequence above is repeated.

At label PUNCH6, the switch is checked to see if this is the first card of the request. If so, exit is made to label PNC12. If not and the request is formatted, exit is made to label PNC16. If not, the core location is checked to see if the request is complete. If so, exit is made to procedure DONE. If it was the first card, the switch is reset. The contents of the core location are then picked up and the two characters converted and stored into the buffer via subroutine PCNVRT. The core location is then checked to see if the request is complete. If not, the core location is incremented. If the buffer is full, exit is made to routine FEEDCD. If not, the core location is checked to see if the request is complete. If not, the sequence is repeated. If it is, zero is put in A and the cycle is repeated to fill the rest of the card with blanks.

At label PUNCH3, formatted binary punch requests are processed. The first word is constructed to contain the sequence number and a 7,9 punch. It is put into the buffer via subroutine PUT. The switch is then checked to see if this is the first card of the record. If so, the switch is reset and the complement of the record length is computed and placed in the buffer via subroutine PUT. A word is then picked up from core and put into the buffer via subroutine PUT. The core location is then incremented and checked to see if the requested number of words has been completed. If not, the cycle is repeated. If so, the switch is checked to see if the checksum has been punched. If not, it is marked punched and put into the buffer by subroutine PUT. If the card is not yet filled, zeroes are issued to the buffer by PUT until it is filled.

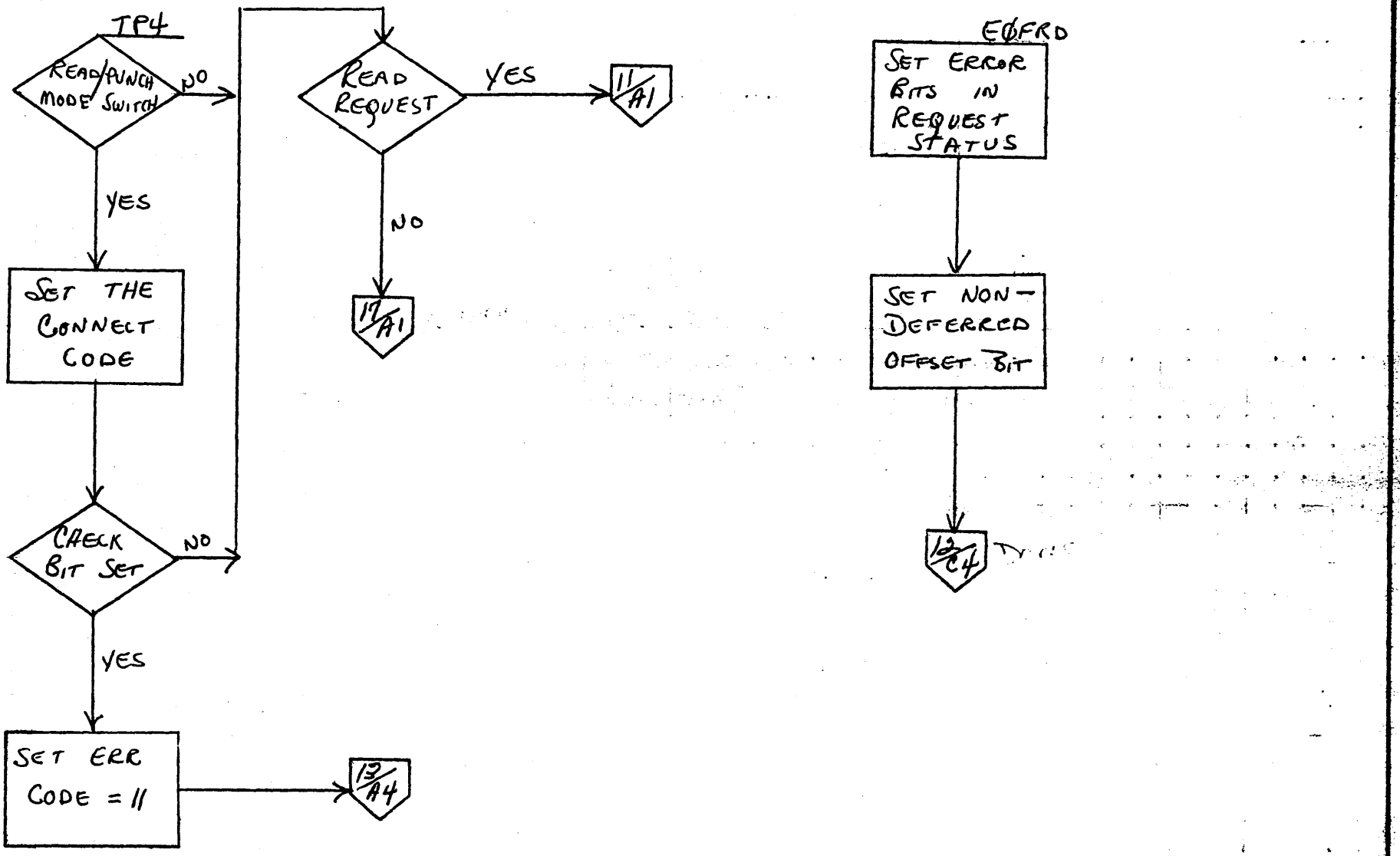


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
	PAGE 1 OF 19	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.12

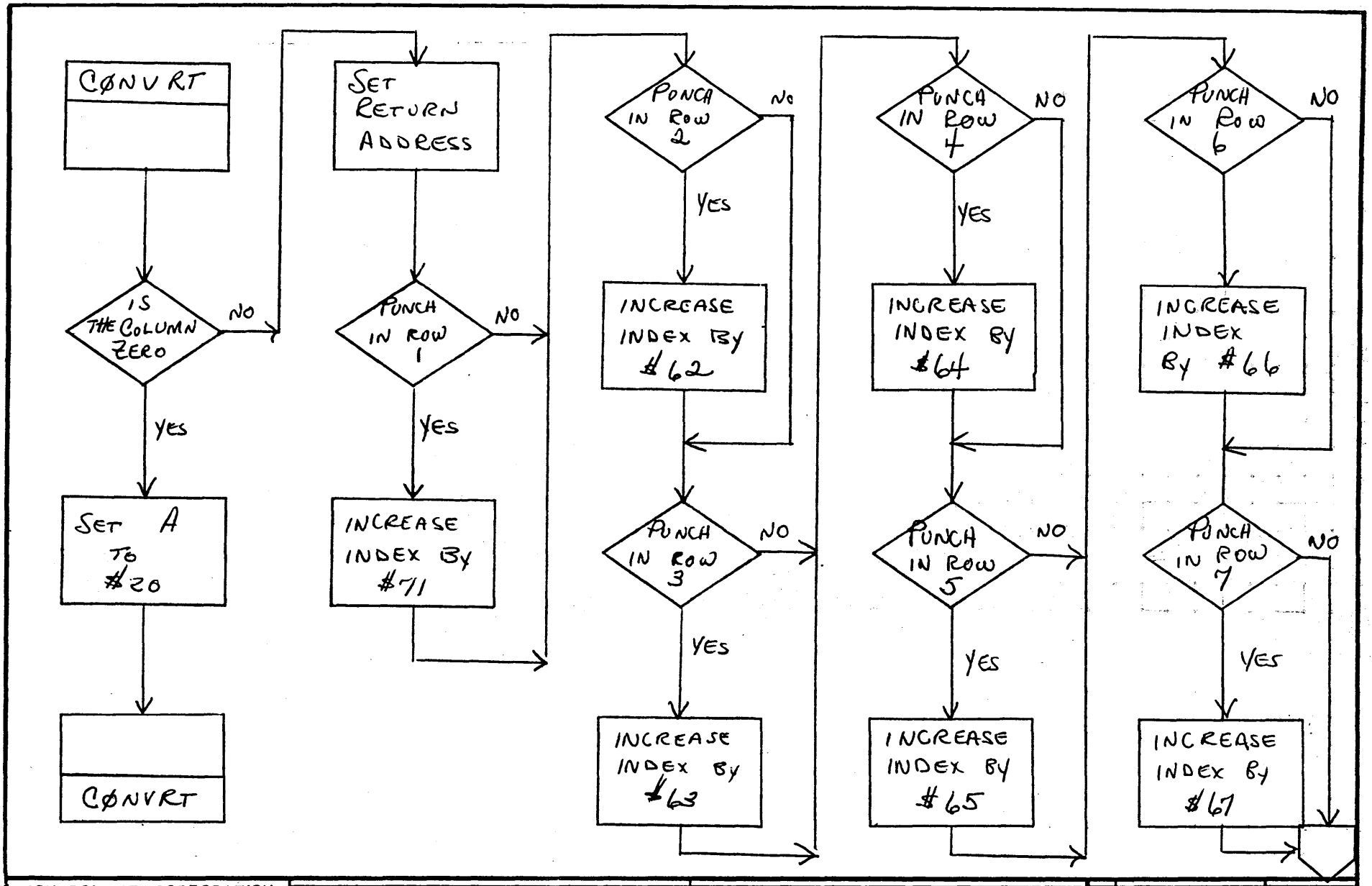


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
FLOWCHART ☐	PAGE 2 OF 19	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.13

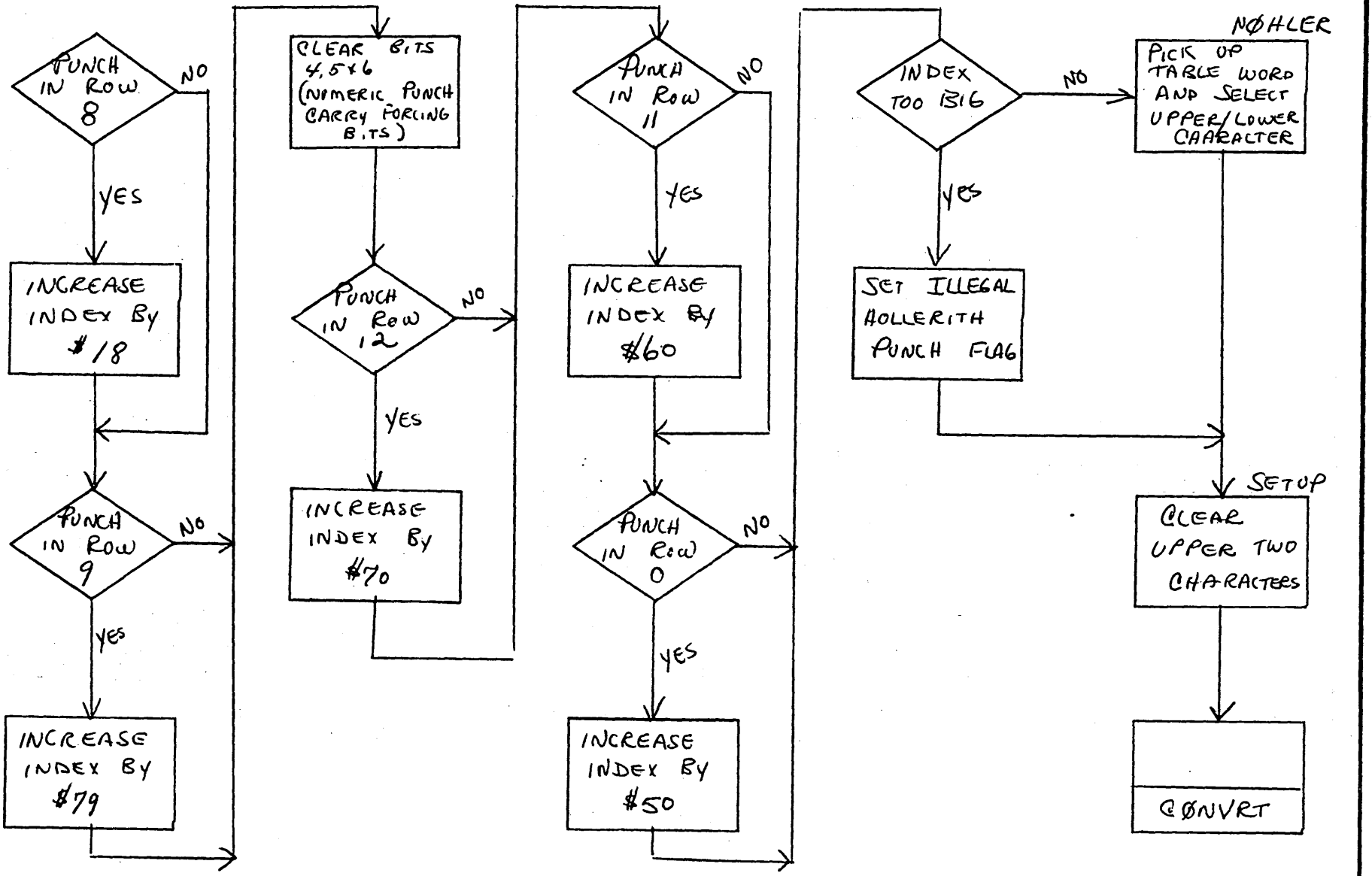


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR.		
	PAGE 3 OF 19		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.14

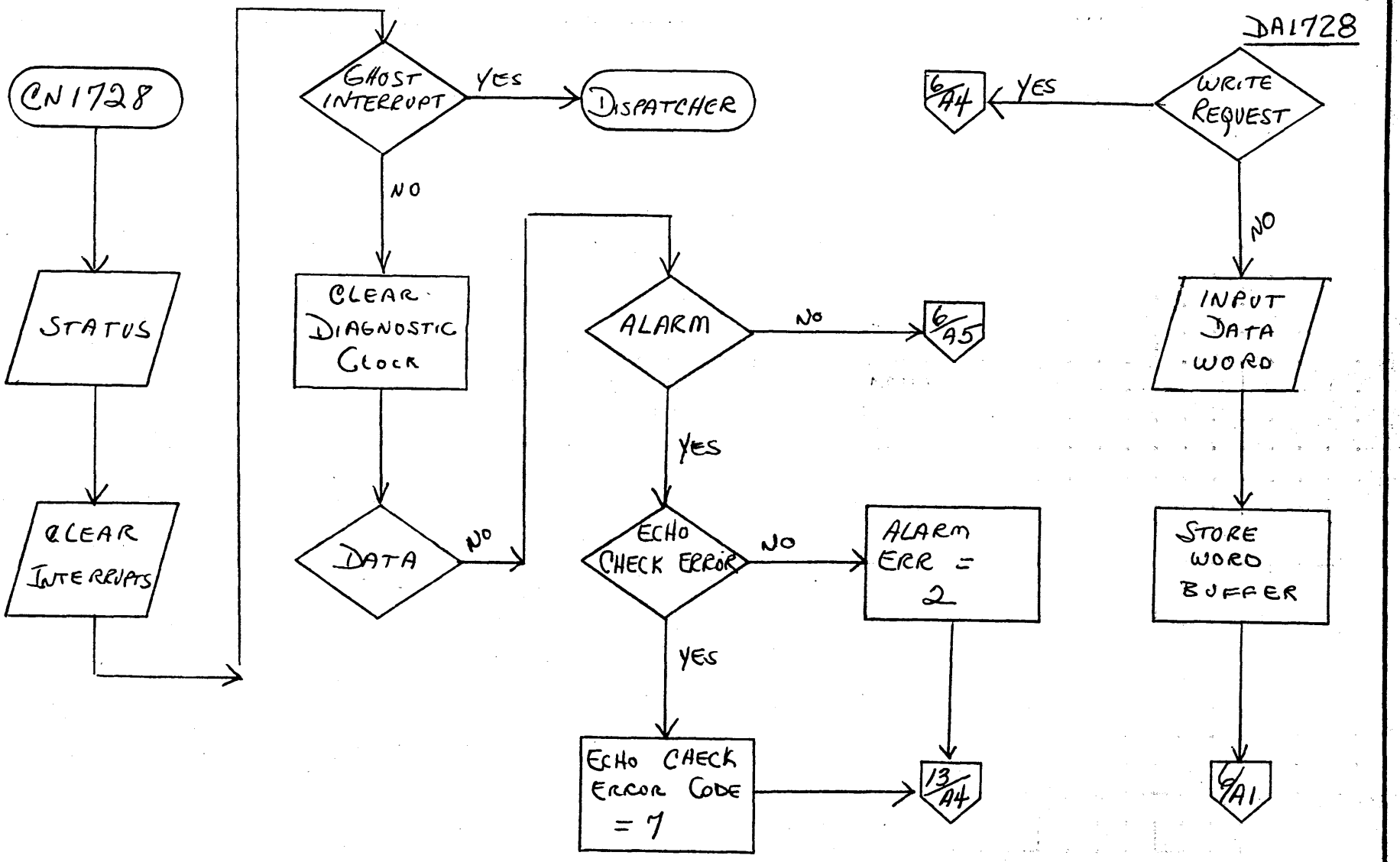


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR			
FLOWCHART ☐		PAGE 4 OF 19	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

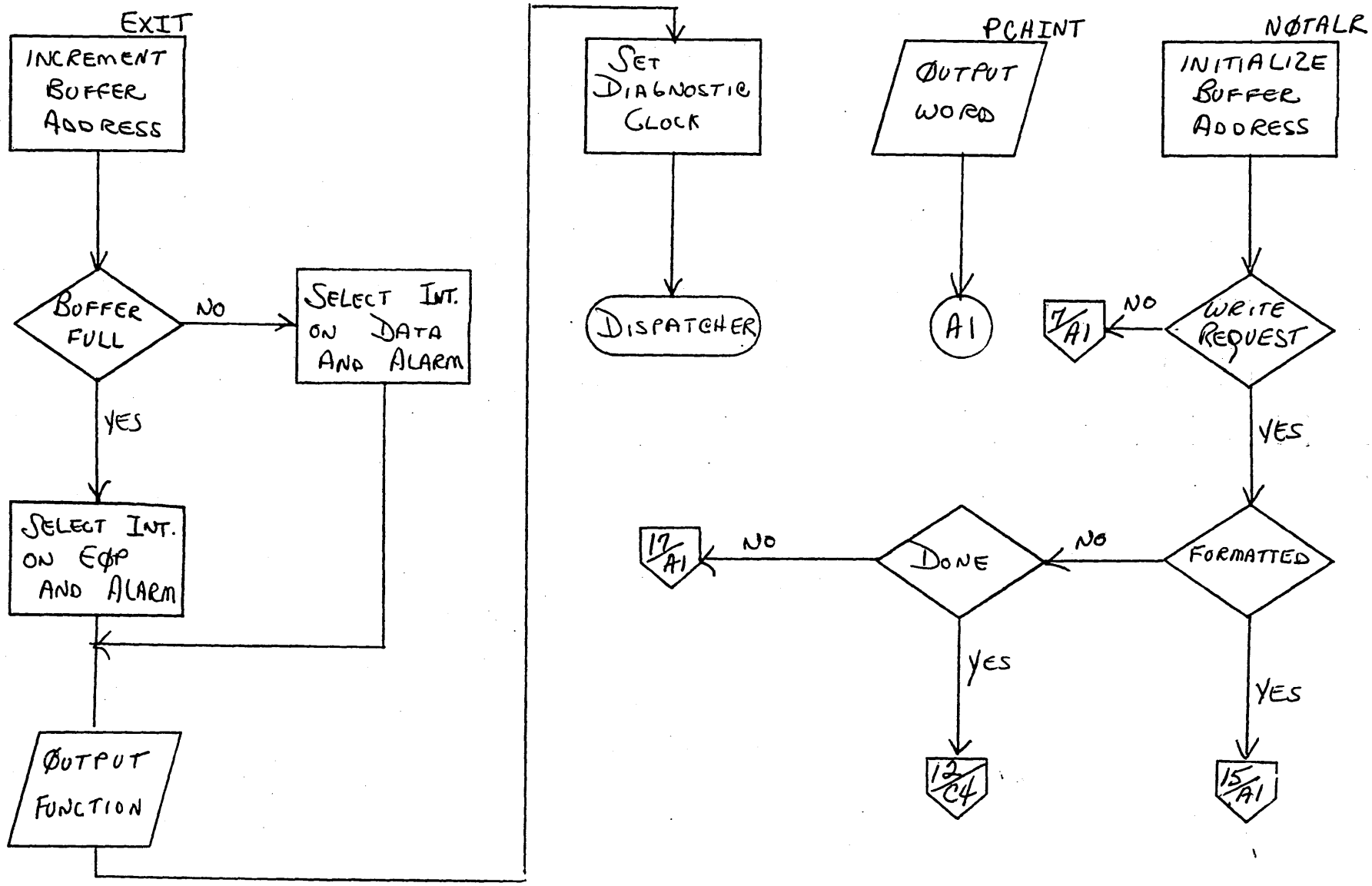
AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.15



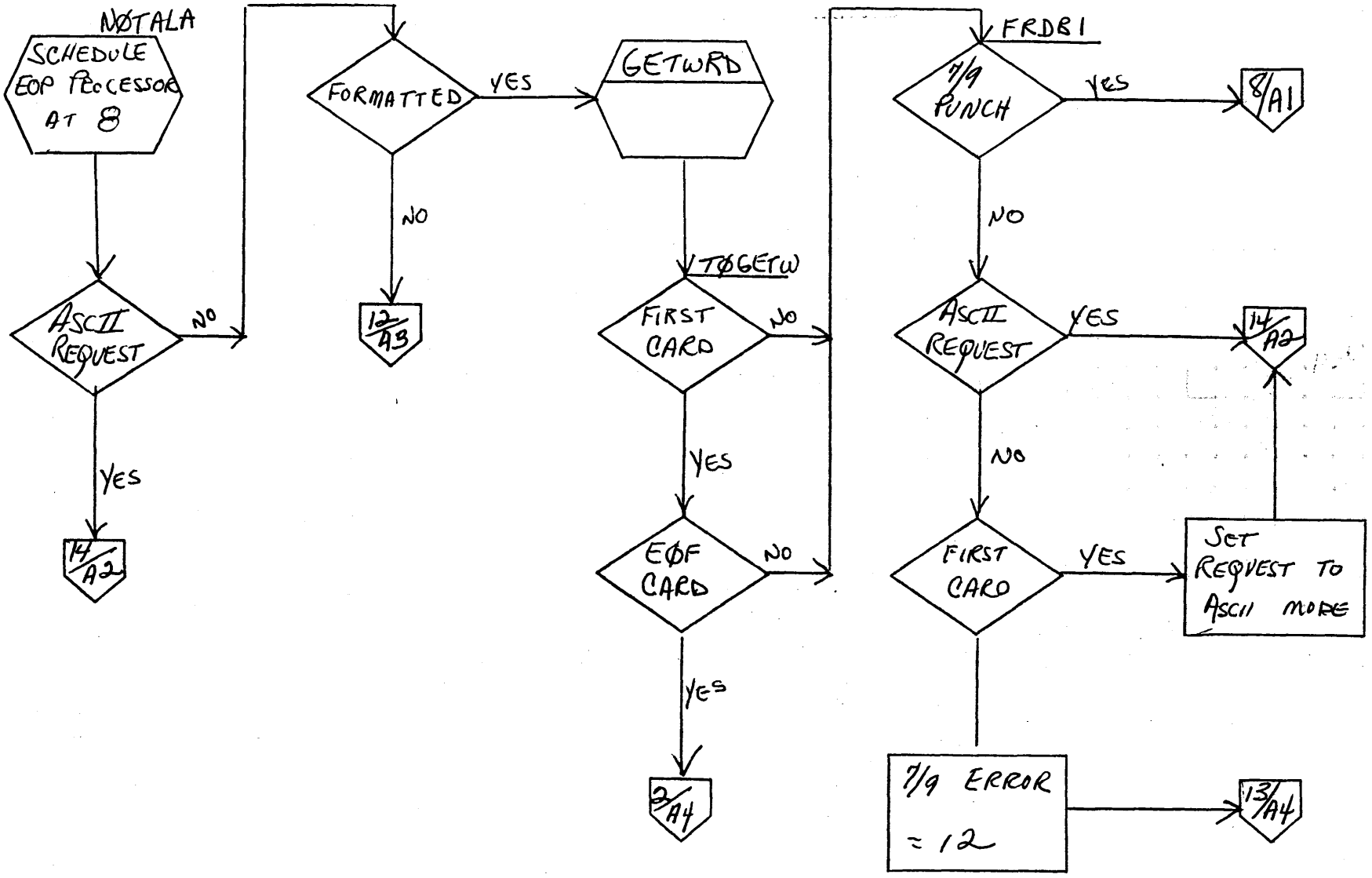
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
	PAGE 5 OF 19	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR.			
FLOWCHART ☐		PAGE 6 OF 19	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

51.17

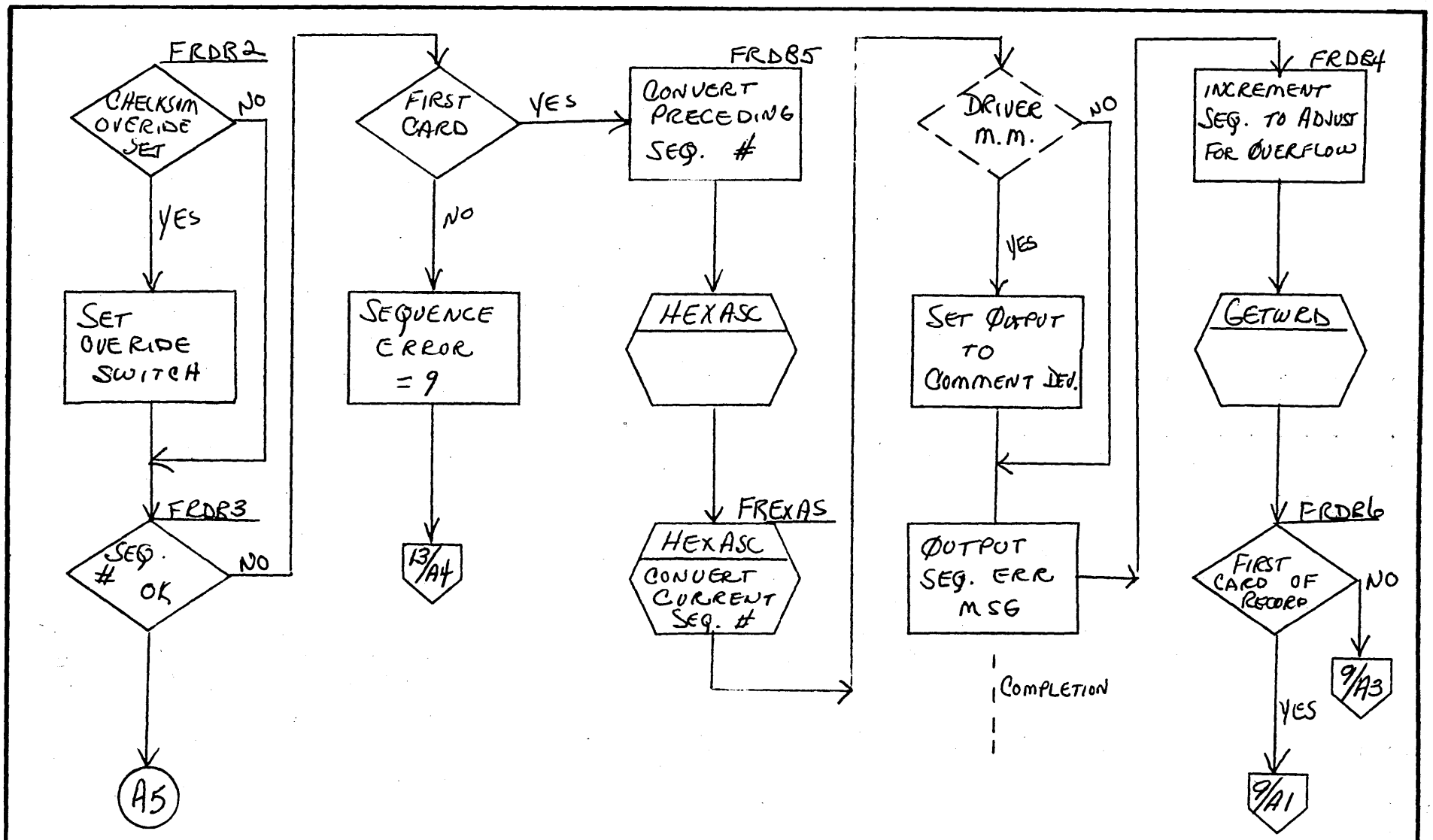


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
	PAGE 7 OF 19	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.18

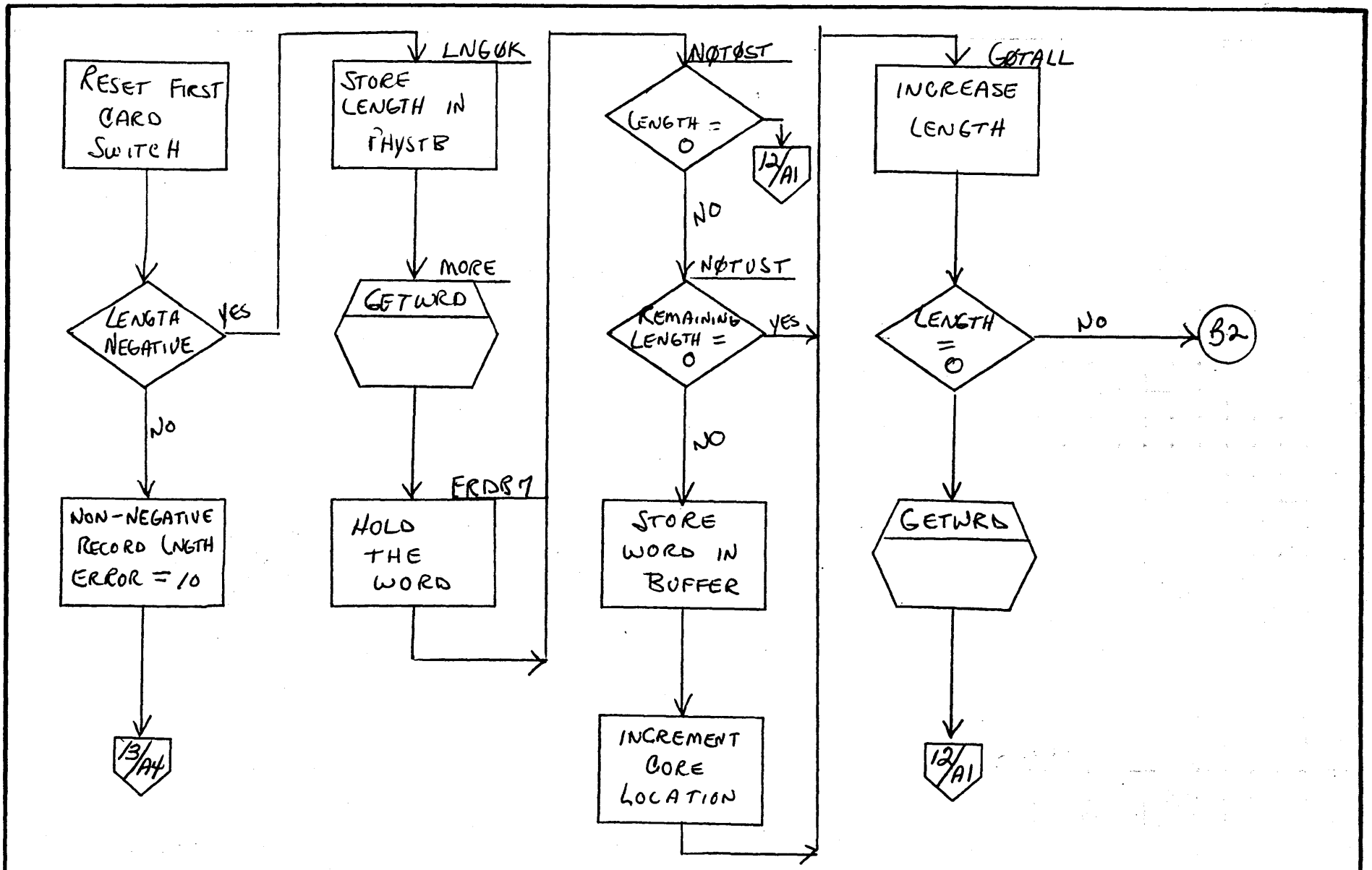


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: 1728/430 DRIVER		PROJECT MGR.		
		PAGE 8 OF 19	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

51.19

AA1385 (FORMERLY CA127)



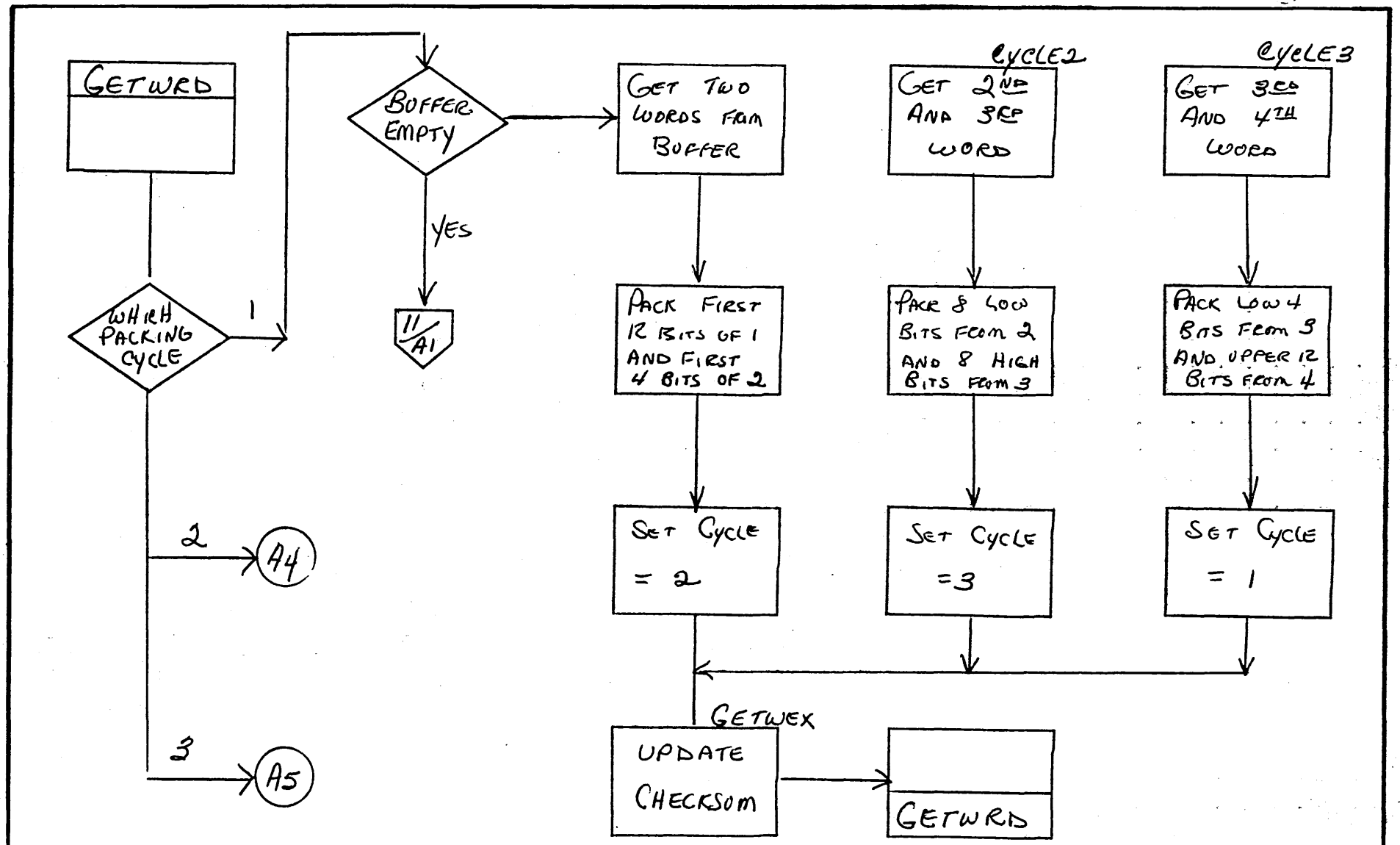
CONTROL DATA CORPORATION			
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.
SAMPLE CODE ☐	DOCUMENT TITLE: 1728/430 DRIVER		PROJECT MGR.
FLOWCHART ☐	PAGE 9 OF 19		PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.
OTHER ☐	DRAWN BY	DATE	TASK NAME

REV	APPROVED	DATE

MAR 5 1971

51.20

AA1385 (FORMERLY CA127-1)

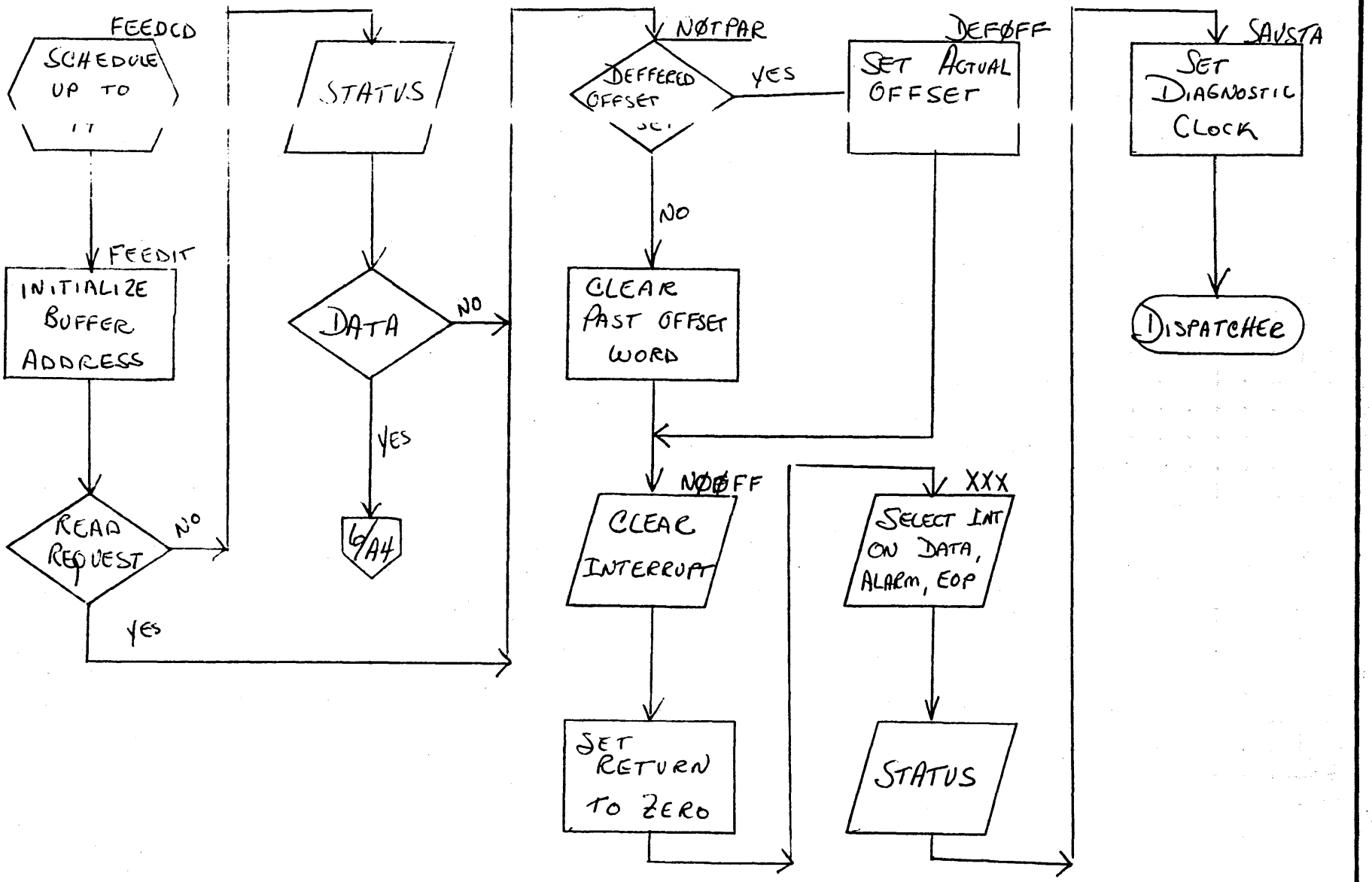


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR.		
	PAGE 10 OF 19		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.21

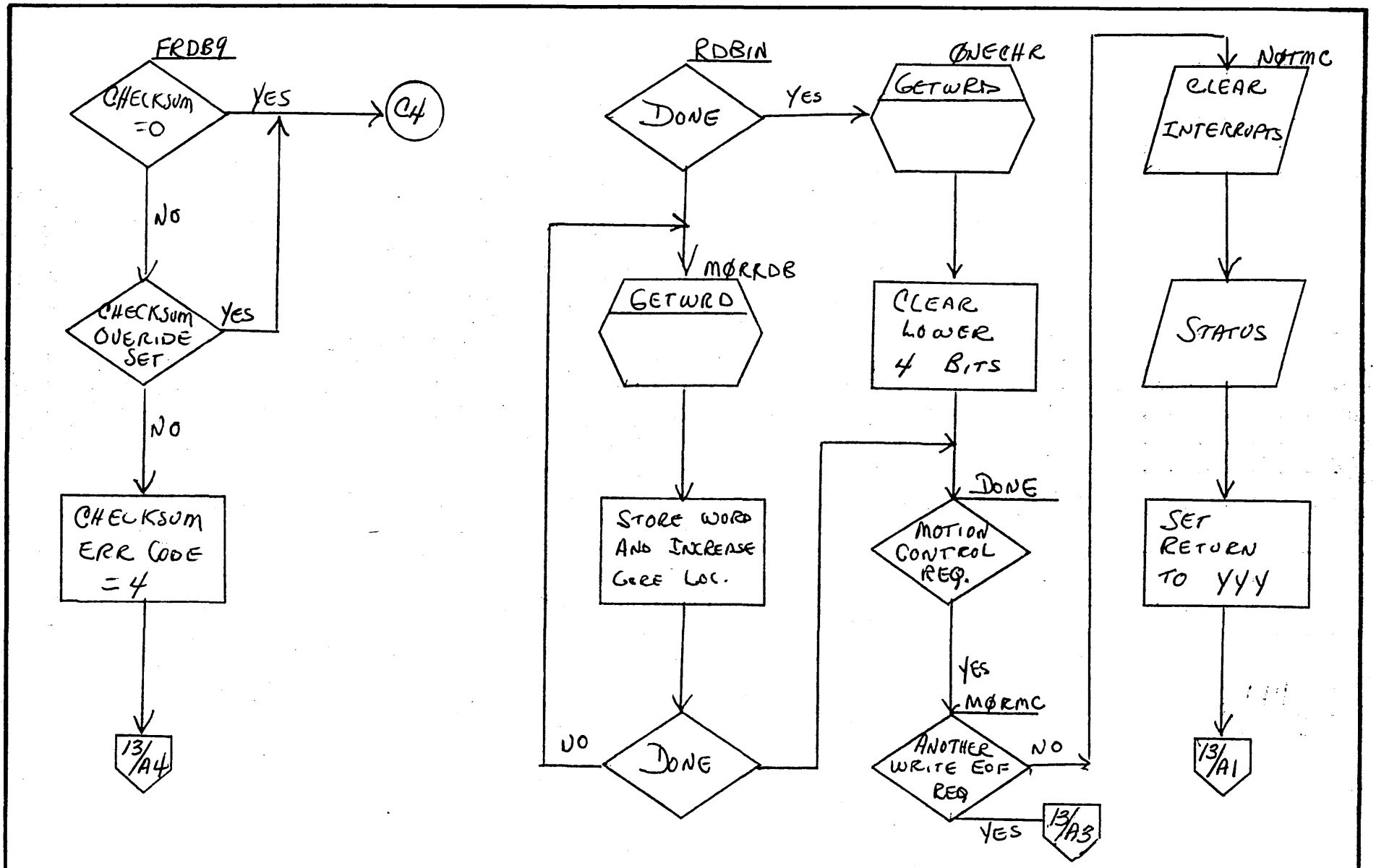


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR.			
		PAGE 11 OF 19	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

51.22

AA1385 (FORMERLY CA127-1)

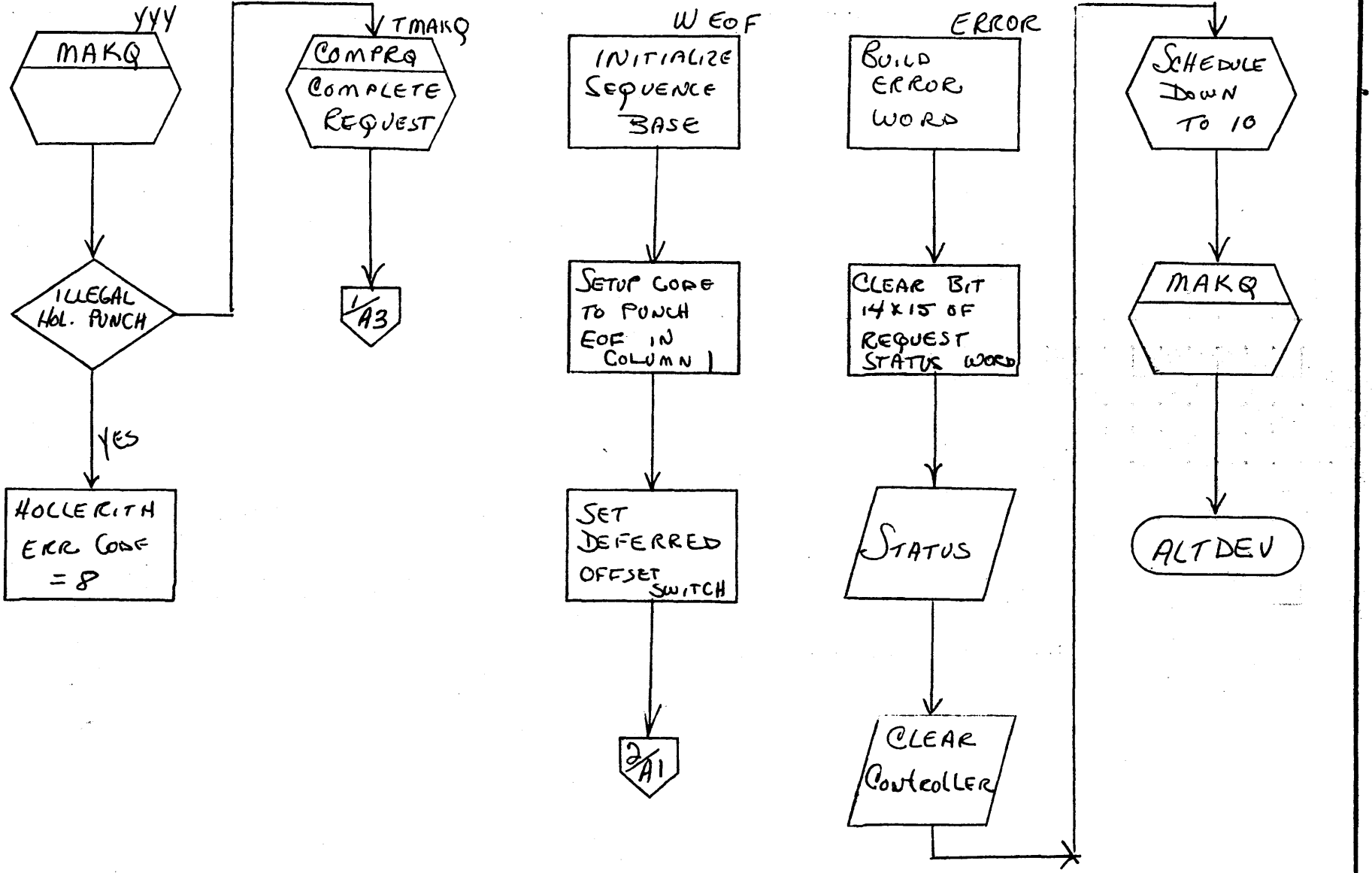


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR.		
		PAGE 12 OF 19	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.23

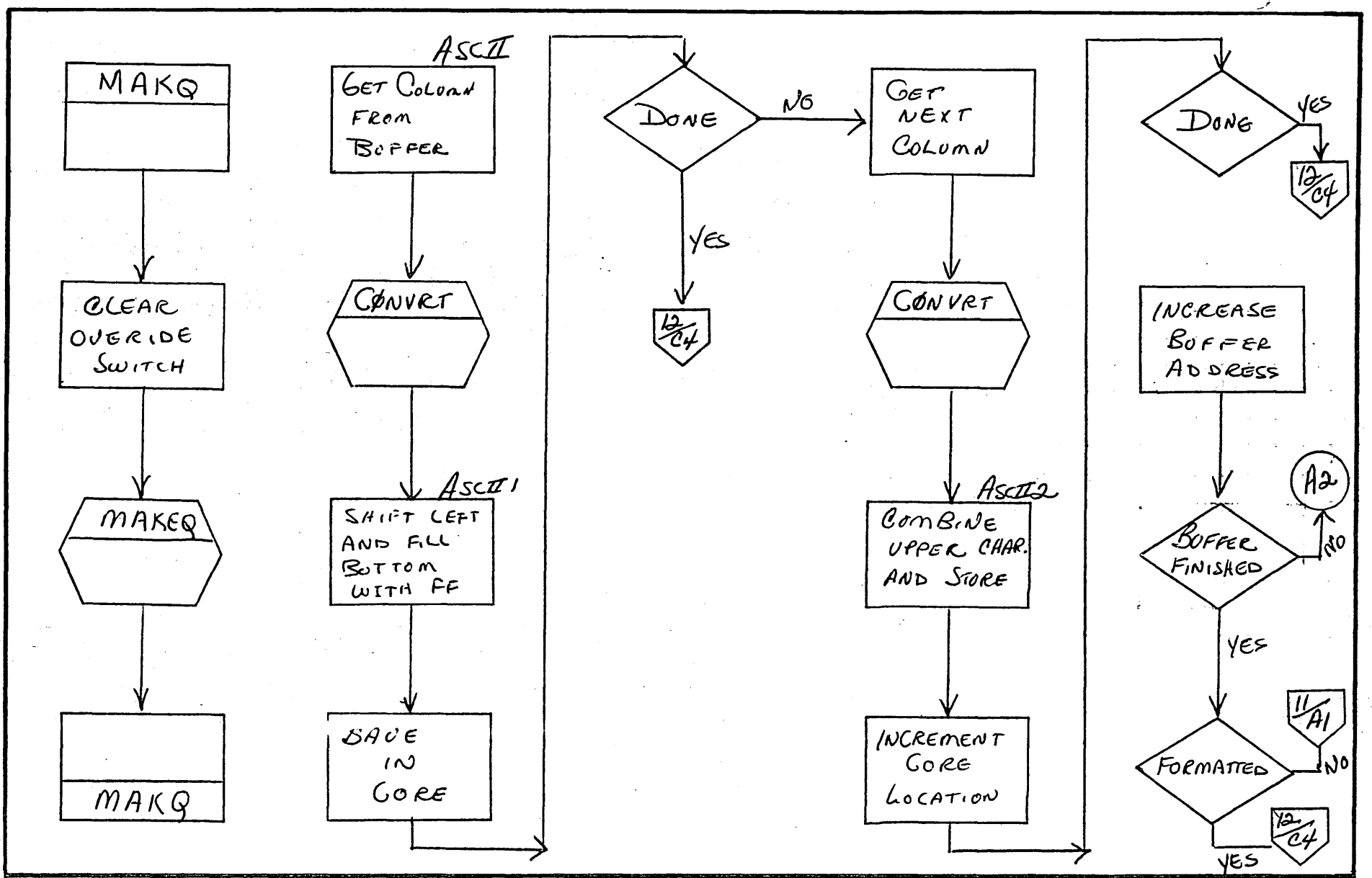


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
	PAGE 3 OF 19	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.24



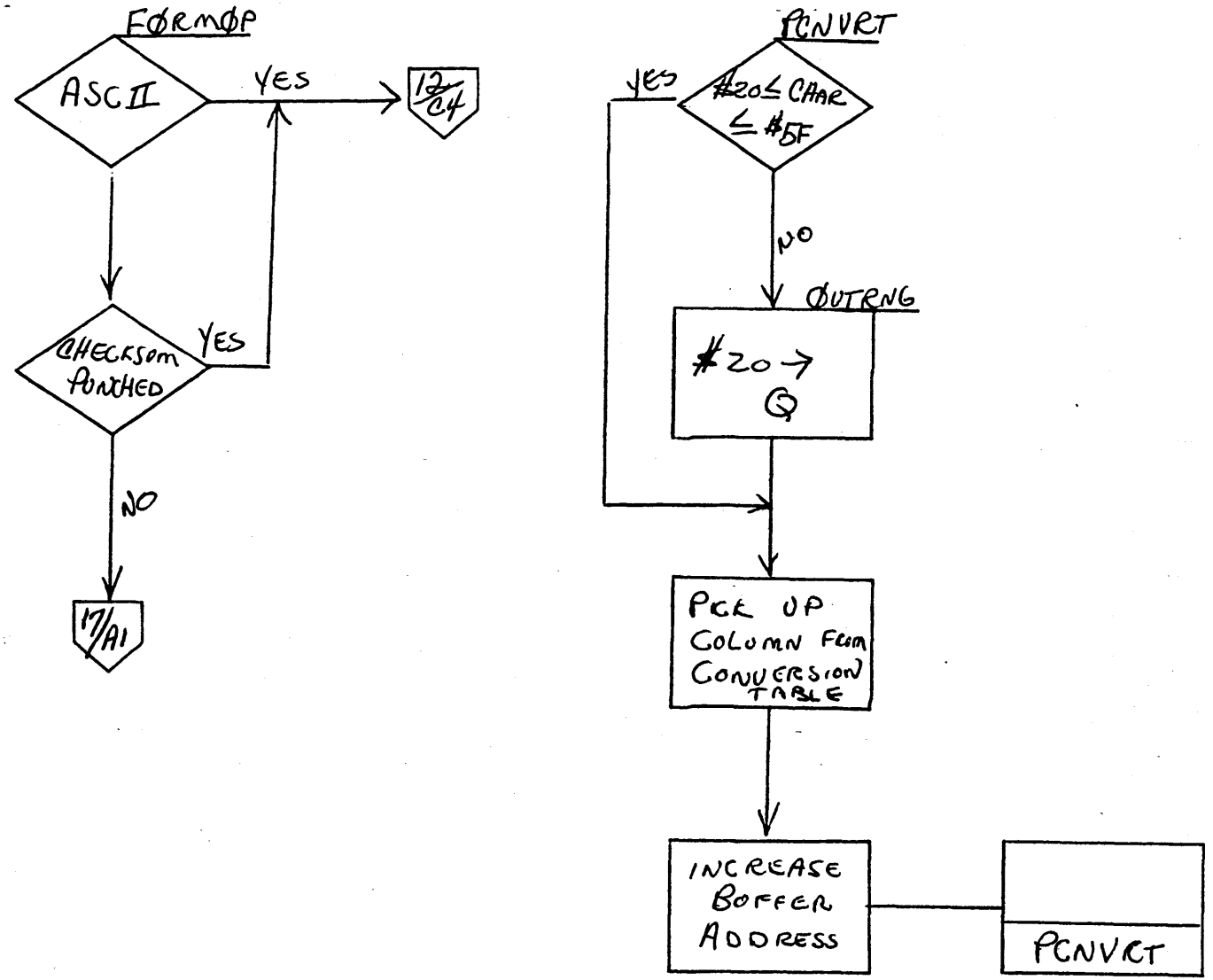
CONTROL DATA CORPORATION			
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.
SAMPLE CODE ☐	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR.
FLOWCHART ☐		PAGE 14 OF 19	PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.
OTHER ☐	DRAWN BY	DATE	TASK NAME

REV	APPROVED	DATE

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.25

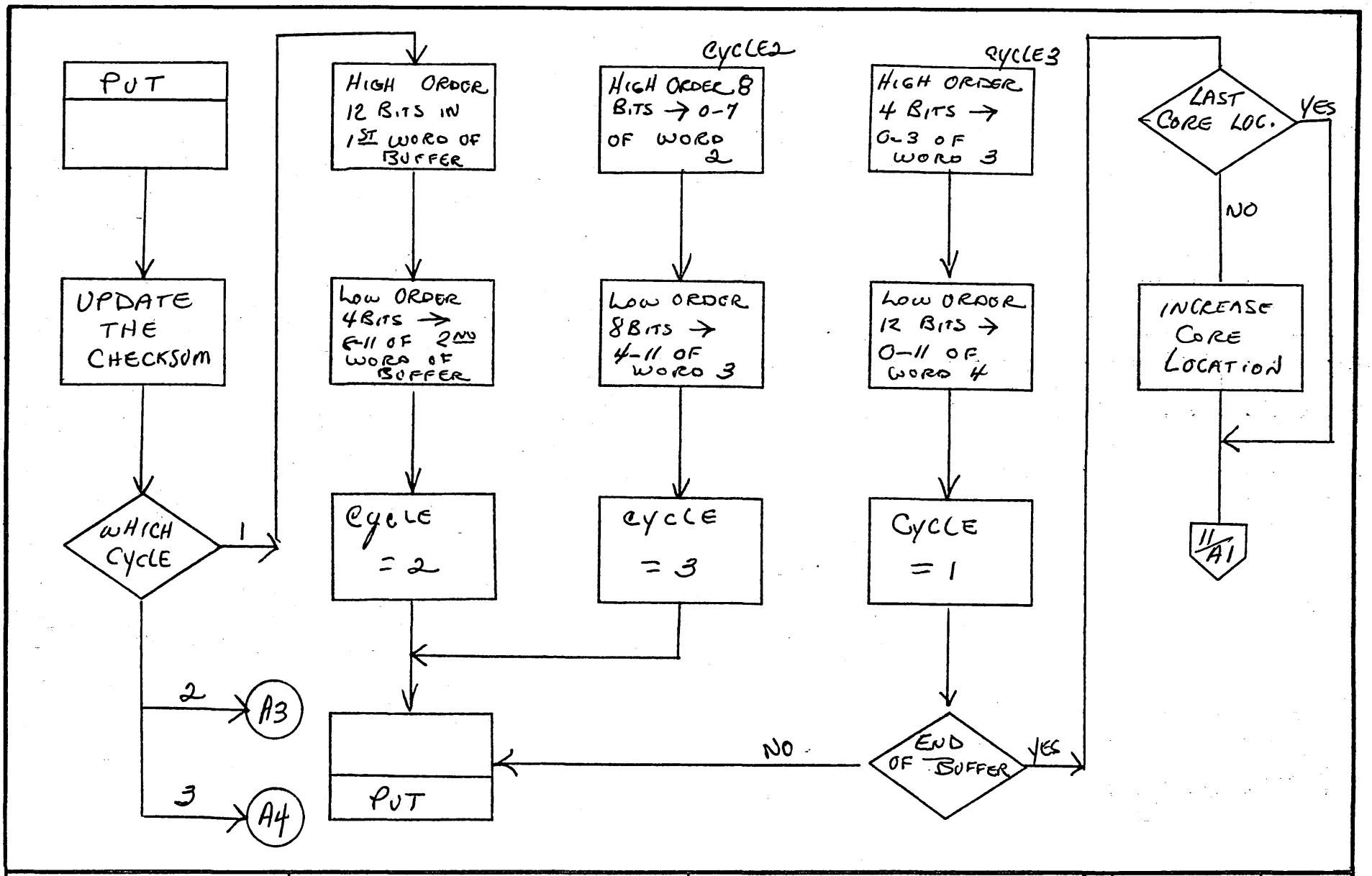


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
	PAGE 15 OF 19	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.26

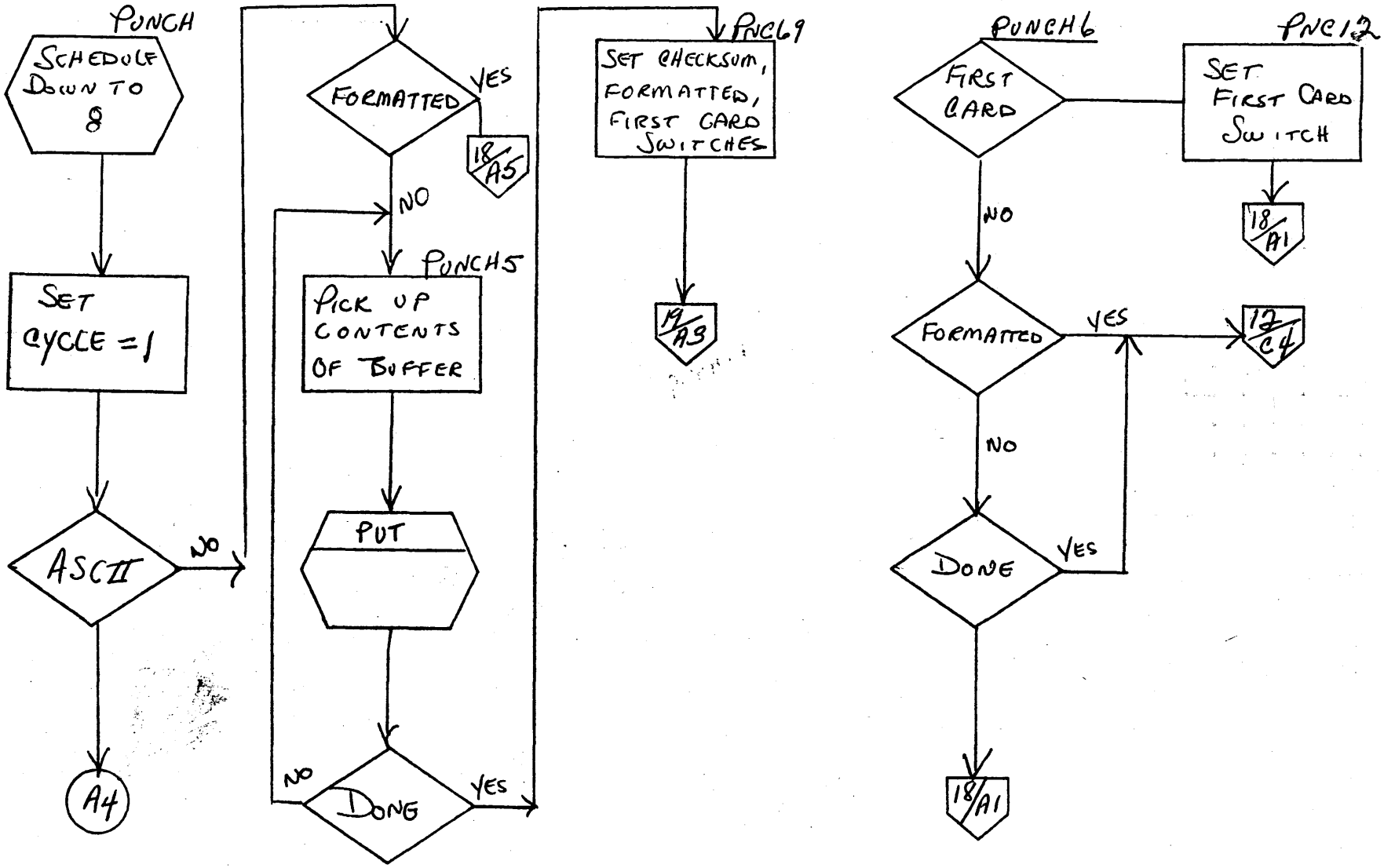


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
	PAGE 16 OF 19	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

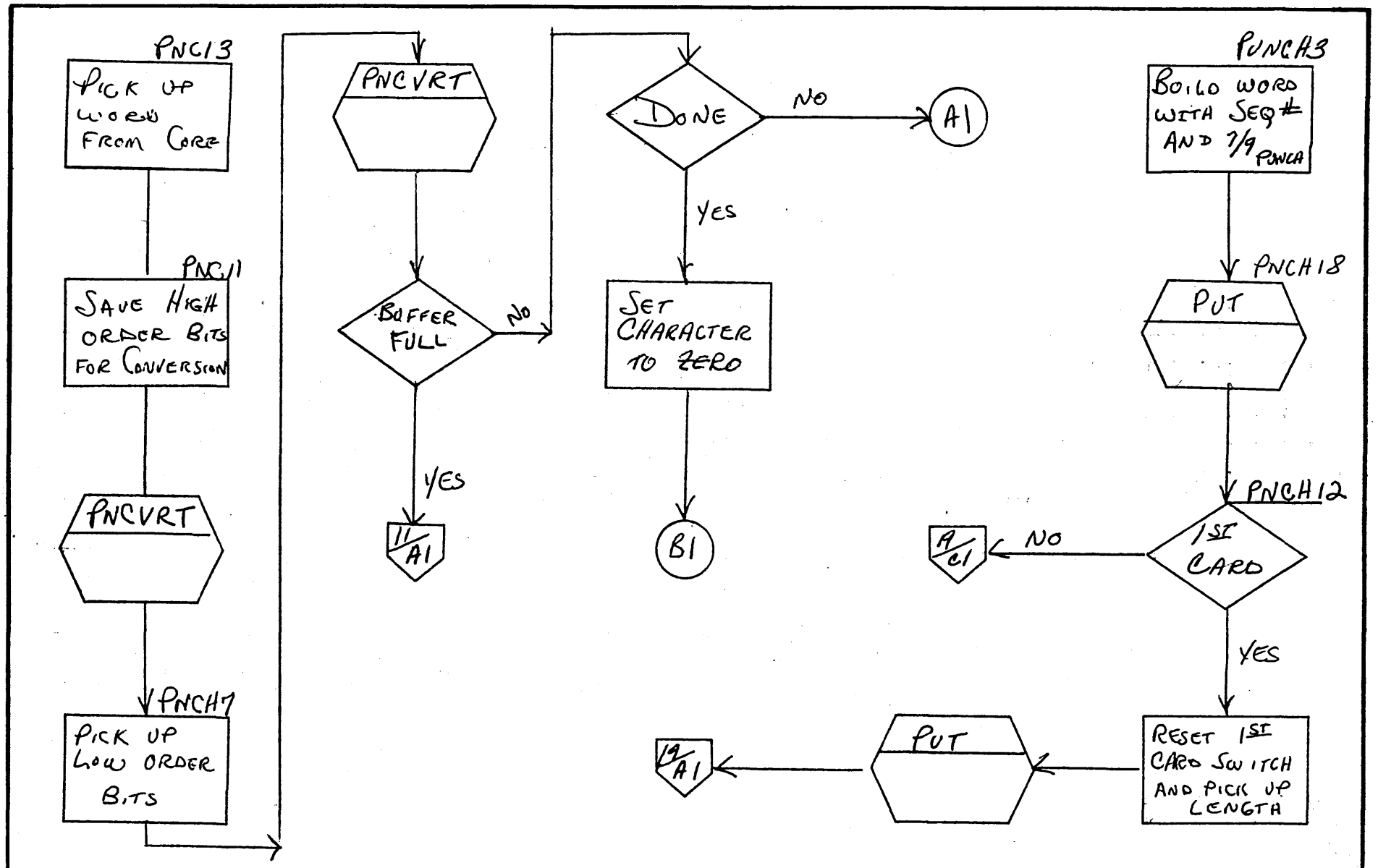
51.27



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1728/430 DRIVER	PROJECT MGR.			
	PAGE 17 OF 19	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

51.28

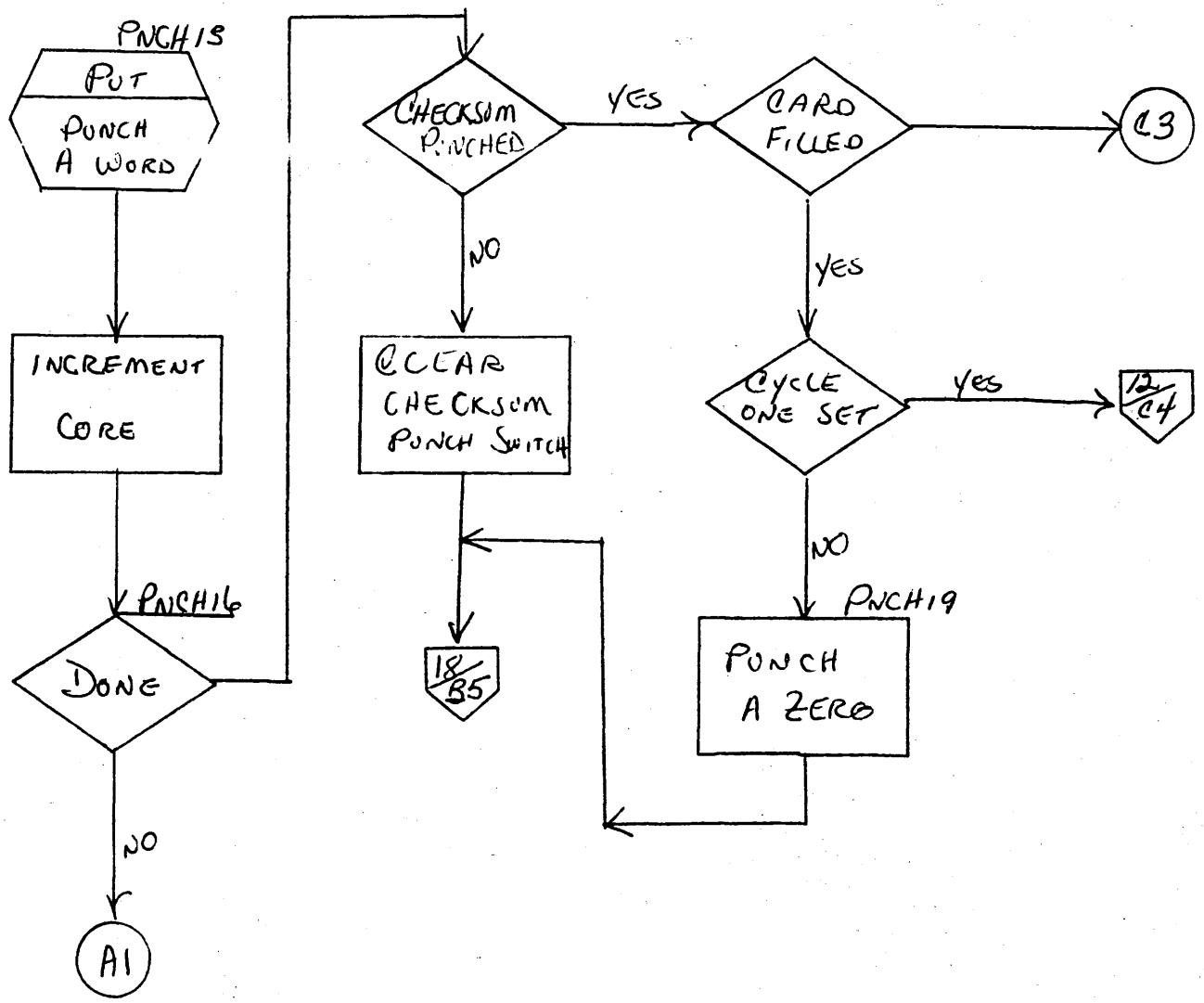


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1728/430 DRIVER		PROJECT MGR.		
		PAGE 18 OF 19	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

51.29



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1728/1430 DRIVER		PROJECT MGR.			
FLOWCHART ☐		PAGE 19 OF 19	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

51.30

AA1385 (FORMERLY CA127-1)

52.0 1740/501 Printer Driver

52.1 Function

The 1740/501 Line Printer Driver is a standard software driver which will operate under the 1700 MSOS 2.1 Operating System. This driver can be used by both the 1742 Line Printer and the 501 Line Printer. This driver can be set up to reside on Mass Memory.

52.2 Entry Points

The following are the 1740/501 Line Printer Driver entry points:

IN501	Initiator Entry Point
CN501	Continuator Entry Point & Interrupt Entry Point
ER501	Error Routine Entry Point

52.3 Externals

The following are externals used by this driver:

ALTDEV	Alternate Device Handler Entry
MAKEQ	Common Driver Completion Routine used before entering COMPRQ.

52.4 General Program Information

52.4.1 Word Nine - Various Aspects

The following is a listing of which bits of word 9, of the Physical Device Table, are used by the driver:

BIT	USE
0	1 for a write operation
1	1 if formatted
2	1 if pending upspace
3	ASCII-Binary bit, not used by the Printer Driver
4	1 if lower character
5	1 if first entry into the driver {for preceding carriage control}

52.4.2 LOCATIONS in the COMMUNICATIONS REGION

The following are communication region locations which are referenced by this driver:

LOCATION	CONTENTS OF THE LOCATION
$09	$007F
$0A	$00FF
$20	$C000
$22	$0000
$30	$2000
$38	$FFDF
$40	$DFFF
$B5	Location of FNR
$B6	Address of FNR Subroutine
$EA	Location of the Dispatcher

52.5 General Driver Description

The 1740/501 Line Printer driver uses two types of requests: the FWRITE and WRITE. Binary-ASCII mode has no significance and will be ignored. For a more detailed description of the format of these requests, see the 1700 Operating System Reference Manual. Two types of modes can be used by this driver, the FORTRAN and the NON-FORTRAN modes. Word 17 of the Physical Device Table for the printer contains the logical unit number of the FORTRAN line printer. When the logical unit number specified by the request is the same as the number in word 17 of the Physical Device Table for the printer, the FORTRAN mode of operation is used. All other logical unit numbers will be handled inthe NON-FORTRAN mode.

Formatted Write

The Formatted Write is used in the FORTRAN and NON-FORTRAN modes as described below:

{1}FORTRAN Mode

When performing a Formatted Write, this mode will interpret the first character of the record for print control and therefore will not print the first character. The following chart lists the characters used for print control and the action which will be performed before printing.

CHARACTER	ACTION BEFORE PRINTING
0	Double Line Space
1	Top of Form
+	No Upspace
All Others	Upspace of 1 Line

{2}NON-FORTRAN Mode

In this mode, an upspace of 1 line will be performed before printing and the first character of the record will be printed.

UNFORMATTED WRITE

In both the FORTRAN and the NON-FORTRAN modes, an Unformatted Write causes no preceding upspace to be performed and printing to be done only if one of the following appears: $03, $04, $0D, or $1B followed by a $30.

52.6 Driver Description

Initiator Entry

At the Initiator entry, IN501, I is set equal to the address of the device's Physical Device Table. FNR subroutine is entered to check if there is a request stacked. If there are no requests stacked, exit is made to the Dispatcher. If there is a request available, location {TEMP}, a temporary buffer, is cleared; the controller is also cleared. A check is then made of MAXLIN to see if page control is wanted. If MAXLIN is a negative number, the line counter {COUNT} is cleared; if MAXLIN is a positive number, that number is stored for future use. Label AFALAR is then entered.

Routine DONE: This routine is entered after successful completion of a request. The controller is cleared, the Diagnostic Clock is set to a -1 and the hardware status is saved. After executing the external subroutines MAKEQ and COMPRQ, the location of the Initiator Entry point +1 is entered to see if another request is waiting.

Error Entry

At the Error routine entry, ER501, I is set to equal the address of the device's physical device table. The error code is set to zero and then routine ERROR is entered.

Routine ERROR: The error indicator (ERRTAB) is set, the status is checked and stored and the controller is cleared. The error code is picked up, the Diagnostic Clock is set to a -1 and then the error code, in Q, is passed to the Alternate Device Handler.

Continuator Entry

At the Continuator entry, CN501, I is set to equal the address of the device's physical device table and the status is checked. If the entry was caused by an Alarm interrupt, the error code is set to a 2 and an exit is made to routine ERROR. If entry was not caused by an Alarm, label AFALAR is entered.

Routine AFALAR: Upon entering this routine, a check is made to see if the request is formatted. If it is not formatted, jump to the NOFORM routine because an unformatted write causes no preceding upspace to be given. If it is formatted, check if this is the first entry into the Continuator. This check is to determine if carriage control functions are to be issued. If the request is formatted, but not the first entry into the Continuator, there is to be no preceding carriage control operation before printing, therefore jump to the NOFORM routine to continue processing the request.

If the request is formatted and the first entry into the Continuator, status is checked to see if the printer is already engaged in a carriage control operation. If the printer is busy, the temporary buffer is cleared, previous interrupts are cleared and Interrupt on End of Operation and Alarm is set, the hardware status is saved, the Diagnostic Clock is set to a 1, and an exit to the Dispatcher is then made. If the printer is not busy, a check is then made to see which mode is being used, the FORTRAN or NON-FORTRAN mode. If the FORTRAN mode is being used, enter routine FTNMOD; if the NON-FORTRAN mode is being used, enter routine NONFTN.

Routine FTNMOD: In order to have reached this routine, the following must be true: (1) the request must be formatted, (2) this must be the first entry into the Continuator by the request, (3) the request must be used in the FORTRAN mode. This routine picks up a word and then stores it in (ODD); the present core location is increased by 1 for future use. A check is then made to see what the first character is in order to decipher which carriage control function should be performed.

If the first character is a +, no preceding upspace is to be performed, therefore, enter the NOFORM routine in order to continue processing.

If the first character is a 1, a page eject is to be executed. The line counter {COUNT}, used with page format control, is cleared and a page eject function is performed. The NOFORM routine is then entered in order to continue processing.

If the first character is a zero, double spacing occurs. The line counter {COUNT}, used with page format control, is increased by 1. The MAX routine is then entered. In this routine the line counter {COUNT} will be increased by 1 again; so effectively the line counter {COUNT} has been increased by a 2. Upon returning from the MAX routine, the double space function is then processed. The NOFORM routine will then continue processing the request.

If the first character is anything other than one of the above, a single space function will occur. The NOFORM routine will then continue processing the request.

Routine NONFTN: In order to have reached this routine, the following must be true: {1} the request must be formatted, {2} this must be the first entry into the Continuator by the request, {3} the request must be used in the NON-FORTRAN mode. The storage location {ODD}, used for temporarily storing odd characters, is cleared and an upspace of 1 line is performed. Routine MAX is then entered to check the line counter for page format control. Upon returning from the MAX routine, the NOFORM routine is entered so that processing will be able to continue.

Routine NOFORM: Upon entering this routine, a check is made to see if there is a non-print function waiting to be completed. If so, entry is made to routine ILEGAL. If not, check if an upspace function is to be executed. If the upspace switch is set, clear the switch, execute an upspace function and enter routine MAX to check the line counter {COUNT}. Upon returning from routine MAX, the NOFORM routine is entered, with the upspace switch {SWITCH} now set to zero, to continue processing the request. If the upspace switch is not set, equal to zero, the NOUPSP routine is entered to examine each remaining word of the print line.

Routine NOUPSP: In order to have reached this routine, carriage control has been taken care of. From this point on, each remaining word of the print line is examined to see if it contains characters outside the printable range of $20-$5F. Upon entering this routine, a check is made to see if all the characters have been checked and transferred. If the transfer has been completed, routine FINISH is entered. If the transfer has not been completed, the next word is picked up. At this word a check is made for odd characters. If there are odd characters, store the word in {ODD} to be worked with later and then combine the upper character of the word just picked up and the lower character of the word previously stored in {ODD}, which is now in the Q register.

Routine NOUPSP: {cont.} The LEGAL routine is then entered to check this newly formed word. If there are no odd characters, routine LEGAL is entered directly.

Routine LEGAL: At this point, a check is made to see if the characters are legal. If either of the characters are illegal, enter the ILEGAL routine; if both characters are legal, output the word to the buffer, increase the character count and the core location. The NOUPSP routine is then entered to check if the transfer is complete.

Routine MAX: This routine checks if page format is wanted. The value in {MAXLIN} will be negative if page format control is not wanted; the value in {MAXLIN} will be positive if page format control is wanted. If page format is not wanted, control will return to the section in the program it came from +1. If format is selected, increase the line counter {COUNT} by 1 and check if the number of lines printed is equal to or greater than the stated number of lines in {MAXLIN}. If equal to or greater, a page eject function is performed and the line counter is cleared. If less than the stated number in {MAXLIN}, output the paper motion function and continue processing.

Routine FINISH: To have reached this routine, all the characters have been checked and transferred to the buffer. In order to complete this request, a check must be made to see if the request is formatted or unformatted. If the request is unformatted, enter the QABUG routine. If the request is formatted, check if there are any odd characters. If any odd characters are present, {ODD} will not equal zero, clear storage location {ODD}, send the odd character and a space to the buffer, and return to the LEGAL routine to check if this last character is legal. If there are no odd characters, {ODD} will equal zero, check if the printer is busy. If it is busy, clear {TEMP}, clear the previous interrupts, set interrupt on End of Operation and Alarm, save the hardware status, set the Diagnostic Clock to a 1 and jump to the Dispatcher. If the printer is not busy, print the buffer, mark the request unformatted and enter the CLEAR routine.

Routine CLEAR: In this routine, the character counter and the temporary storage area are cleared. Previous interrupts are cleared, interrupts on Data and Alarm are selected, the hardware status is saved, the Diagnostic Clock is set to a 1 and then a jump is made to the Dispatcher.

Routine QABUG: In order to have reached this routine, the request must have been unformatted and all of the characters have been examined and transferred to the buffer. A check is made to see if there are any odd characters. If there are no odd characters, exit is made to routine DONE. If there is an odd character, check what the character is. If it is not one of the following, exit to routine DONE.

Routine @ABUG: (cont.)

$03	End of Text
$04	End of Transmission
$0B	Vertical Tab
$0C	Form Feed
$0D	Carriage Return

If it is one of the above, they are considered legal and will be processed. The odd storage area (ODD) is cleared and routine ILEGAL is entered to process the above codes.

Routine ILEGAL: This routine checks the upper character for an illegal character. If the upper character is legal, a switch is set and then the lower character is checked. If one of the following illegal characters, below the printable range of $20-$5F, is detected, enter the routine specified on the right. All other characters below the printable range will enter the IGNORE routine. A synopsis of the routines, listed below, will follow the chart.

Character	Routine
$03	EOT
$04	EOT
$09	TAB
$0B	VTAB
$0C	FORMFD
$0D	CARRET
$1B	ESCAPE

Routine EOT: This routine will process the $03 and $04 codes. Upon entering this routine, the word, containing the illegal code, $03 or $04, will be saved in the temporary storage location (TEMP). A check is made to see if the printer is busy. If it is busy, previous interrupts are cleared. Interrupts on End of Operation and Alarm are selected, the hardware status is saved, and the Diagnostic Clock is set to 1. Exit is made to the Dispatcher. If the printer is not busy, the upspace switch is set so that the paper will be upspaced one line, the next available core location is set up, and the odd character storage area (ODD) is cleared. If the $03 or $04 is in the upper character, enter the PRTOUT routine and do not increase the character counter or the core location; if the $03 or $04 is in the lower character, place a space in the lower character position. The legal character in the upper character position, and the space are then sent to the buffer. The character counter and core location are both increased. Enter the PRTOUT routine, so that the buffer may be printed.

Routine PRTOUT: In order to have reached this print routine, the request must be unformatted. This routine prints the buffer and exits to the CLEAR routine.

Routine TAB: The routine computes the number of spaces to the next tabstop and then sends the computed number of spaces to the buffer. The number of spaces between each tabstop is set at assembly time by an EQU TABSTOP{X}, X = number of spaces. Upon entering this routine, the word, containing the $09, is stored in {TEMP}. A check is made to see if the $09 is in the upper character position. If it is, enter the MORSPC routine. If the $09 is in the lower character, place a space in the lower character position so that the upper character and a space are placed in the A register. Enter the ATOBUF routine.

Routine MORSPC: This routine increases the space counter. A check is made to see if the space counter equals zero {it will equal zero if the next tabstop has been reached}. If the next tabstop has not been reached, {LC does not equal zero}, load the A register with $2020 {space, space} and then enter routine ATOBUF. If the next tabstop has been reached, a check is made to see if the $09 is the upper character. If it is the upper character, place a space before the lower character and transfer this newly formed word to the Q register and enter routine LEGAL. If the $09 is the lower character, a check must also be made to see if any odd characters are present. If there are odd characters present {ODD does not equal zero}, pick up the character{s} in {ODD} by placing them in the Q register and then clear the storage location {ODD}. A space is placed before the odd character and transferred to the Q register before entering the LEGAL routine. If there are no odd characters present {ODD = 0}, place a space in {ODD}, increase the core location by 1 and enter the NOUPSP routine.

Routine ATOBUF: This routine sends the word in the A register to the buffer. The space count is increased by 1. A check is made to see if the space counter is equal to zero; if equal to zero, the next tabstop has been reached. If equal to zero, a check is made to see if the upper character is the last space before the next tabstop. If it is the upper character, enter the IGNORE routine to blot out the space in the lower character. If the last space is in the lower character, increase the core location and then enter the NOUPSP routine. If the next tabstop has not been reached {space counter will not equal zero}, enter MORSPC routine.

Routine VTAB: This routine processes the $0B code; it uses a function to select format tape level 2. Upon entering this routine, the word, containing the $0B code, is stored in {TEMP} for future use. The printer is checked to see if it is busy. If it is, previous interrupts are cleared, Interrupts on End of Operation and Alarm are selected, the hardware status is saved and the Diagnostic Clock is set to 1. Exit to the Dispatcher is then made. If the printer is not busy, activate format channel for tape level 2 and then enter the IGNORE routine.

Routine FORMFD: This routine processes the $0C code; it issues a page eject function. Upon entering this routine, the word, containing the $0C code, is stored in {TEMP}. A check is made to see if the printer is busy. If it is, previous interrupts are cleared, interrupts on End of Operation and Alarm are selected, the hardware status is saved and Diagnostic Clock is set to 1; exit is made to the Dispatcher. If the printer is not busy, the line counter, used with page format control, is cleared and a page eject function is executed.

Routine CARRET: This routine processes the $0D code; it prints the buffer. Upon entering this routine, the word, containing the $0D code, is saved in {TEMP}. A check is made to see if the printer is busy. If it is busy, previous interrupts are cleared, interrupts on End of Operation and Alarm are selected, the hardware status is saved and the Diagnostic Clock is set to 1; an exit is then made to the Dispatcher. If the printer is not busy, set the upspace switch, and check if the $0D is in the lower character. If it is in the lower character, combine the upper character and a space to finish packing the buffer. The character counter and the core location are both increased. The buffer is printed and then the CLEAR routine is entered. If the $0D is in the upper character, the buffer is printed and routine CLEAR is entered.

Routine ESCAPE: This routine processes the $1B code, by examining the character following the escape character and by issuing the appropriate function. Upon entering this routine, the word, containing the $1B code, is stored in {TEMP}. A status check is made to see if the printer is busy. If it is busy, previous interrupts are cleared, interrupts on End of Operation and Alarm are selected, the hardware status is saved, the Diagnostic Clock is set to 1 and an exit to the Dispatcher is made. If the printer is not busy, a check is made to see if the upper character is the escape character {$1B}. If it is the upper character, pick up the word stored in {TEMP} and enter the GOTES routine. If the $1B is the lower character, pick up the upper character of the next word to determine what the function character will be. Combine the $1B and this upper character and enter routine GOTES.

Routine GOTES: This routine checks which director function should be issued. The first check is for the print function {$30}; if equal to $30 enter the Print routine, so that the buffer may be printed. If found to be less than $30, increase the core location and enter the NOUPSP routine because any character, when accompanied with the $1B, below $30 is not considered to be a director function. If the function is greater than $30, but less than $40, this routine will keep shifting the bit 1 place at a time, to determine which function should be issued. The following is a chart of the possible 1740/501 Line Printer functions, all other codes will be ignored:

Routine GOTES: {cont.}

ESCAPE CODE	1740 FUNCTION
$1B30	Print
$1B31	Single Space
$1B32	Double Space
$1B33	Advance to Level 1
$1B34-$1B39	Advance to Level 2-7
$1B3E	Advance to Level 12

Once the function is determined, the director function is executed. A check is then made to see if the $1B is the upper character. If it is, increase the core location and enter NOUPSP routine. If it is in the lower character, pick up last legal character and then increase the core location and enter the NOUPSP routine.

Routine PRINT: The word in {TEMP} is picked up. A check is made to see if the upper character is the $1B. If it is the upper character, increase the core location and enter the PRTOUT routine. If it is the lower character, check if there are any odd characters. If there is, clear {ODD}, send the upper character and a space to the buffer, increase the character counter and the current address and then enter the PRTOUT routine. If there are no odd characters, store the lower character {unused} in {ODD}, send the upper character and a space to the buffer, increase the core location and increase the character counter and then enter the PRTOUT routine.

Routine ILEGAL also checks the range of characters to see if they can be printed. If the character found is less than $20 and <u>not</u> one of the following, enter routine IGNORE.

$03
$04
$09
$0B
$0C
$0D
$1B

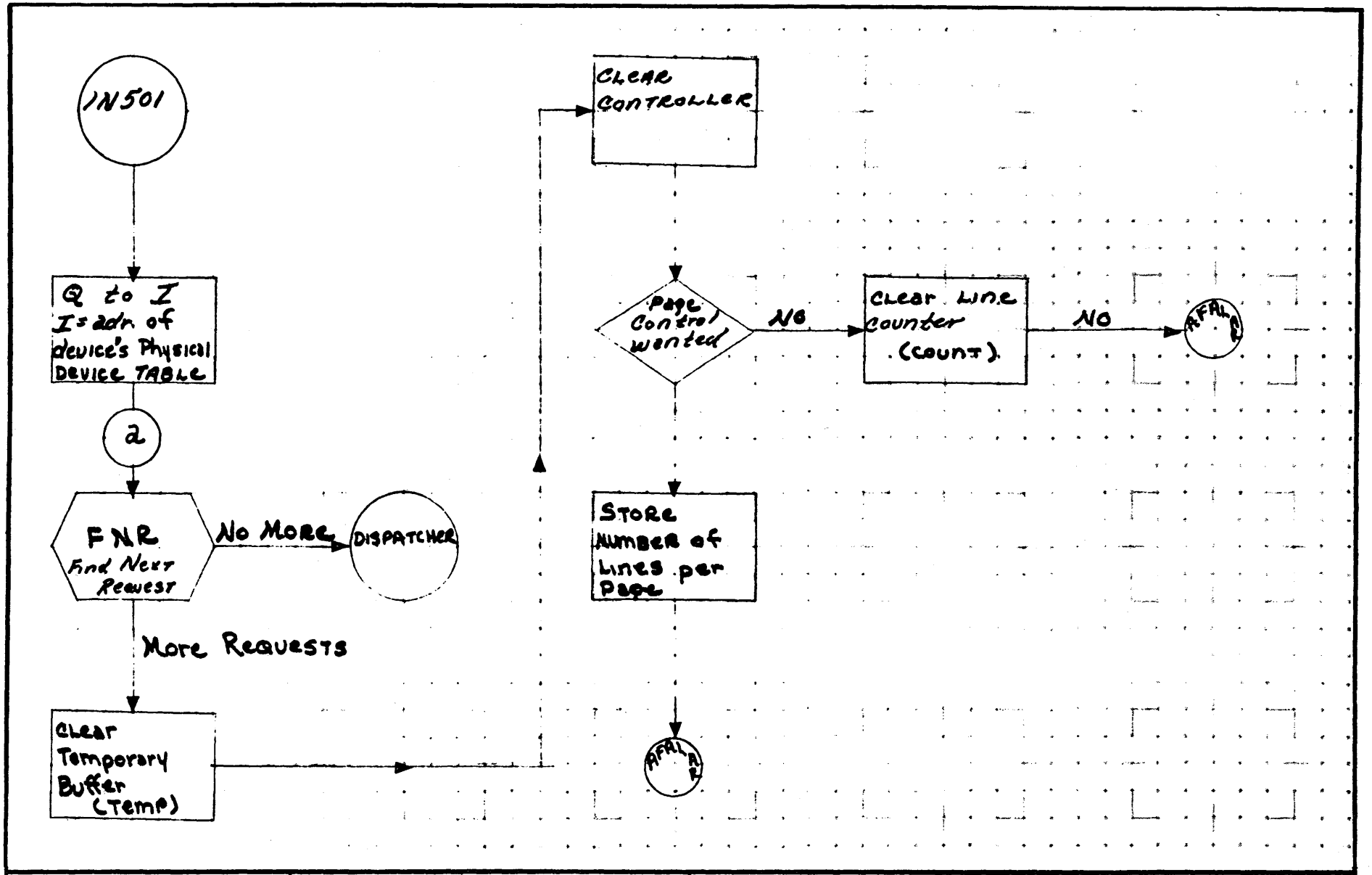
If the character is greater than $20, check if the character is between the range of $60 and $7F. If the character is found to be in the range of $60 and $7F, enter routine TRNSLT so that the lower case character will be converted to the upper case character in the printable range.

If the character is found to be greater than $7F, enter routine IGNORE.

Routine TRNSLT: This routine processes the character found in the range of $60-$7F by translating the character in its lower case {$60-$7F} to its upper case {$40-$5F}. A check is made to see if the illegal character is in the upper character of the word. If the illegal character is the upper character, subtract twenty from its ASCII value by masking with $DFFF. Enter routine LEGAL to continue processing. If the illegal character is the lower character, subtract twenty from its ASCII value by masking with $FFDF and then enter routine LEGAL.

Routine IGNORE: This routine prohibits the printing of the illegal character. Upon entering this routine, a check is made to see if the illegal character is the upper character of the word. If it is the upper character of the word, enter routine YSILGL. If it is the lower character, shift the illegal character to the upper character position and then enter routine YSILGL.

Routine YSILGL: This routine checks if there is an odd character. If there is an odd character, combine the odd character with the saved, legal character, clear the odd character storage location {ODD} and jump to routine LEGAL to check the legality of the newly formed word. If there are no odd characters, store the saved legal character in {ODD}, increase the core location and then enter routine NOUPSP.

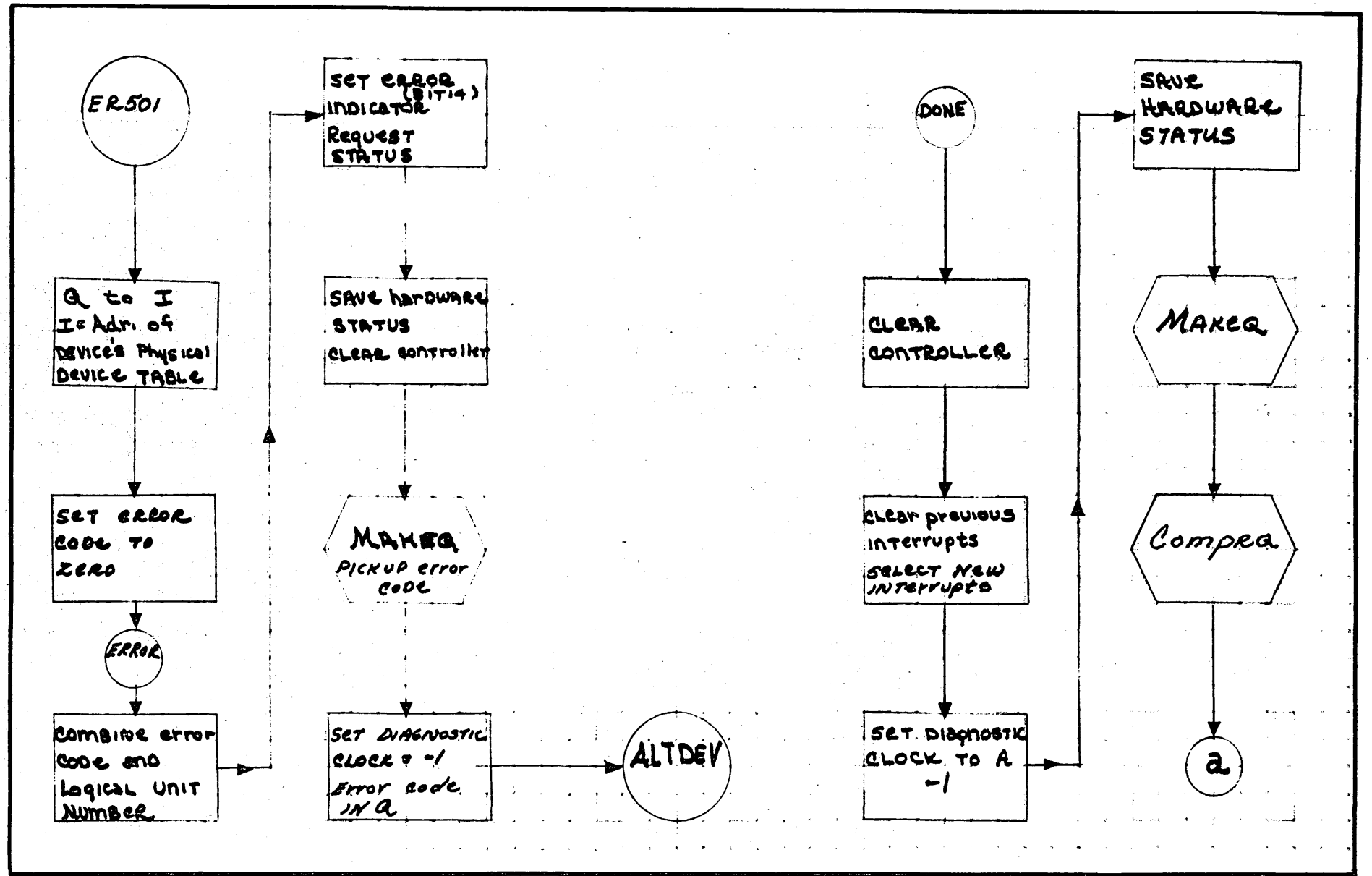


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer	PROJECT MGR.			
	DRIVER PAGE 1 OF 17	PROJECT NAME			
	NUMBER D717A 1-E006 #2.1 ISSUE DATE	TASK NO.			
	DRAWN BY M.L.F. DATE 1-70	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971



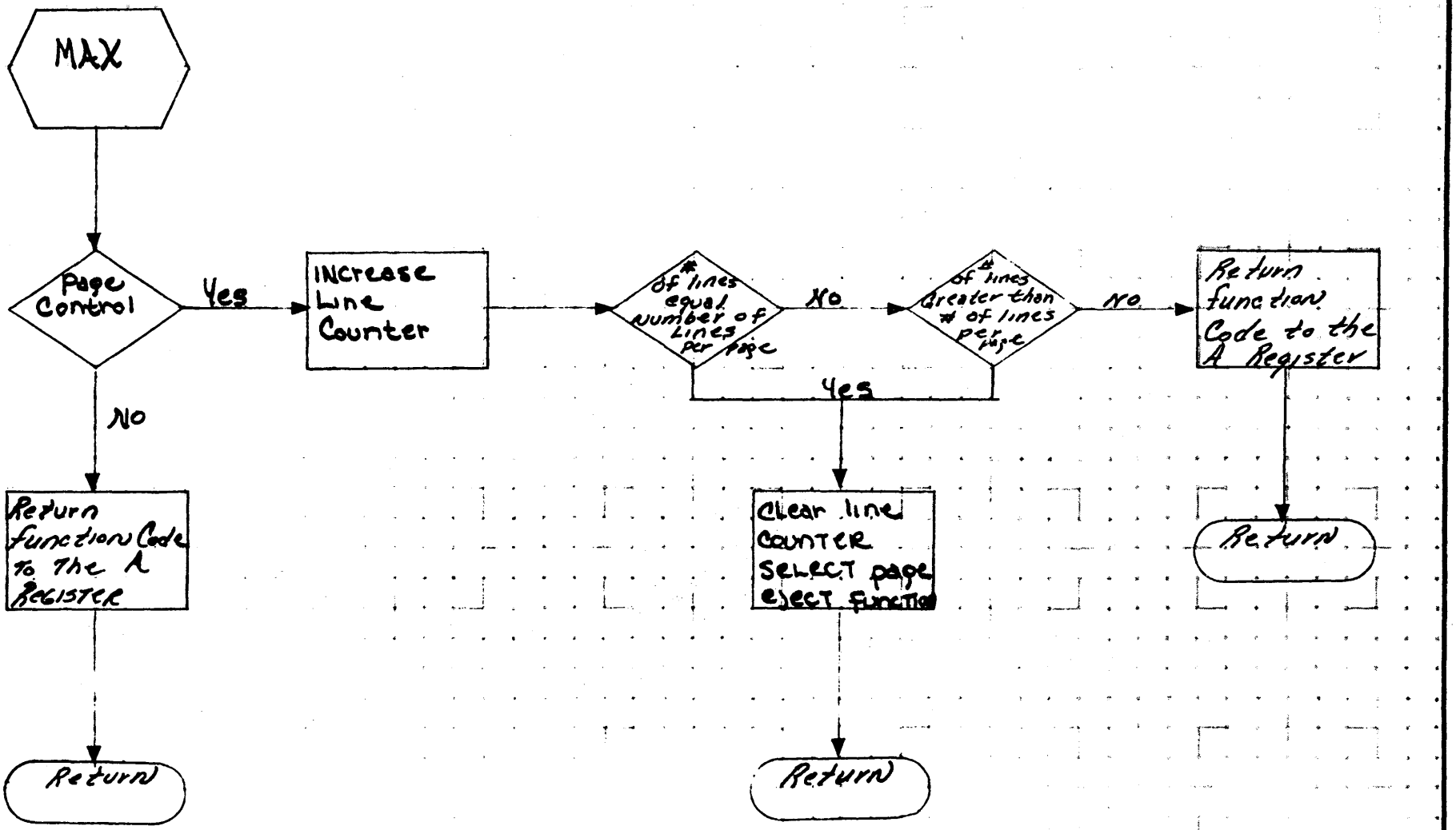
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE 1790/501 Line Printer Driver PAGE 2 OF 17 NUMBER D717-1-E00-02.1 ISSUE DATE DRAWN BY M.L.F. DATE 1-70	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

Page 52.13



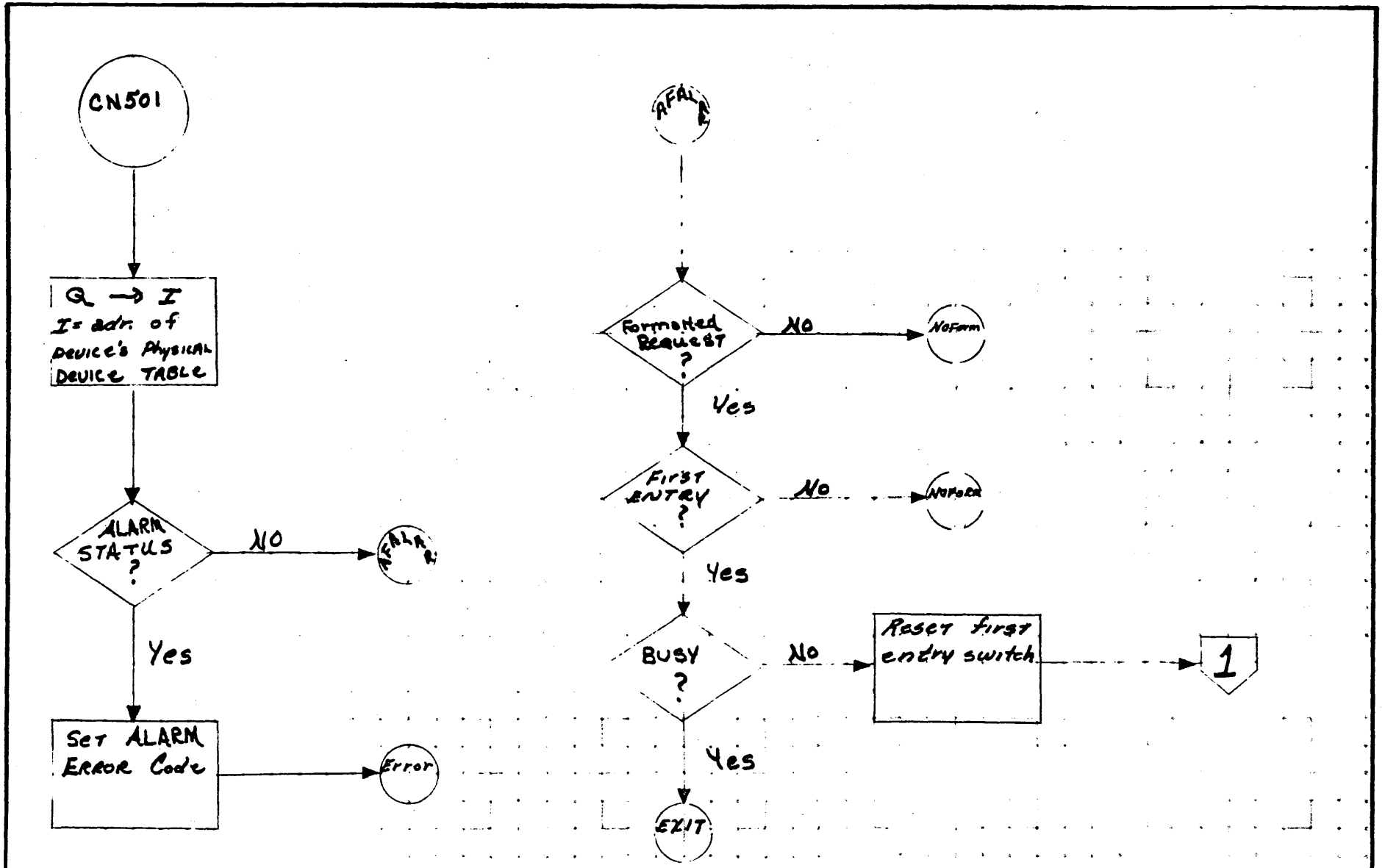
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer		PROJECT MGR.			
	Driver	PAGE 3 OF 17	PROJECT NAME			
	NUMBER D717#1-E006#2.1	ISSUE DATE	TASK NO.			
	DRAWN BY M. L. F.	DATE 1-70	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

Page 52.14

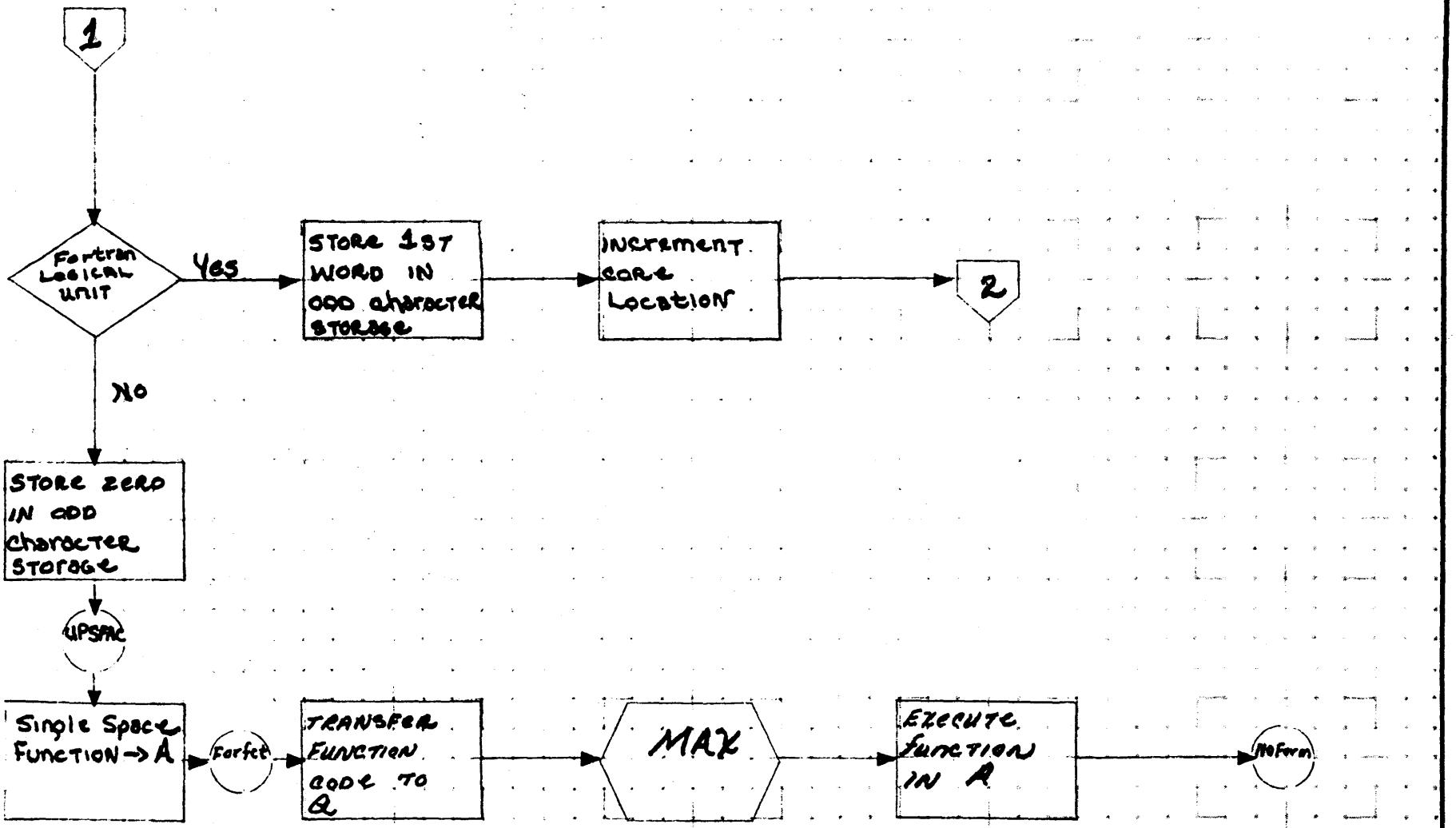


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer Driver PAGE 4 OF 17	PROJECT MGR.			
	NUMBER D71701-E006 #2.1 ISSUE DATE	PROJECT NAME			
	DRAWN BY M. L. F. DATE 1-70	TASK NO.			
		TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

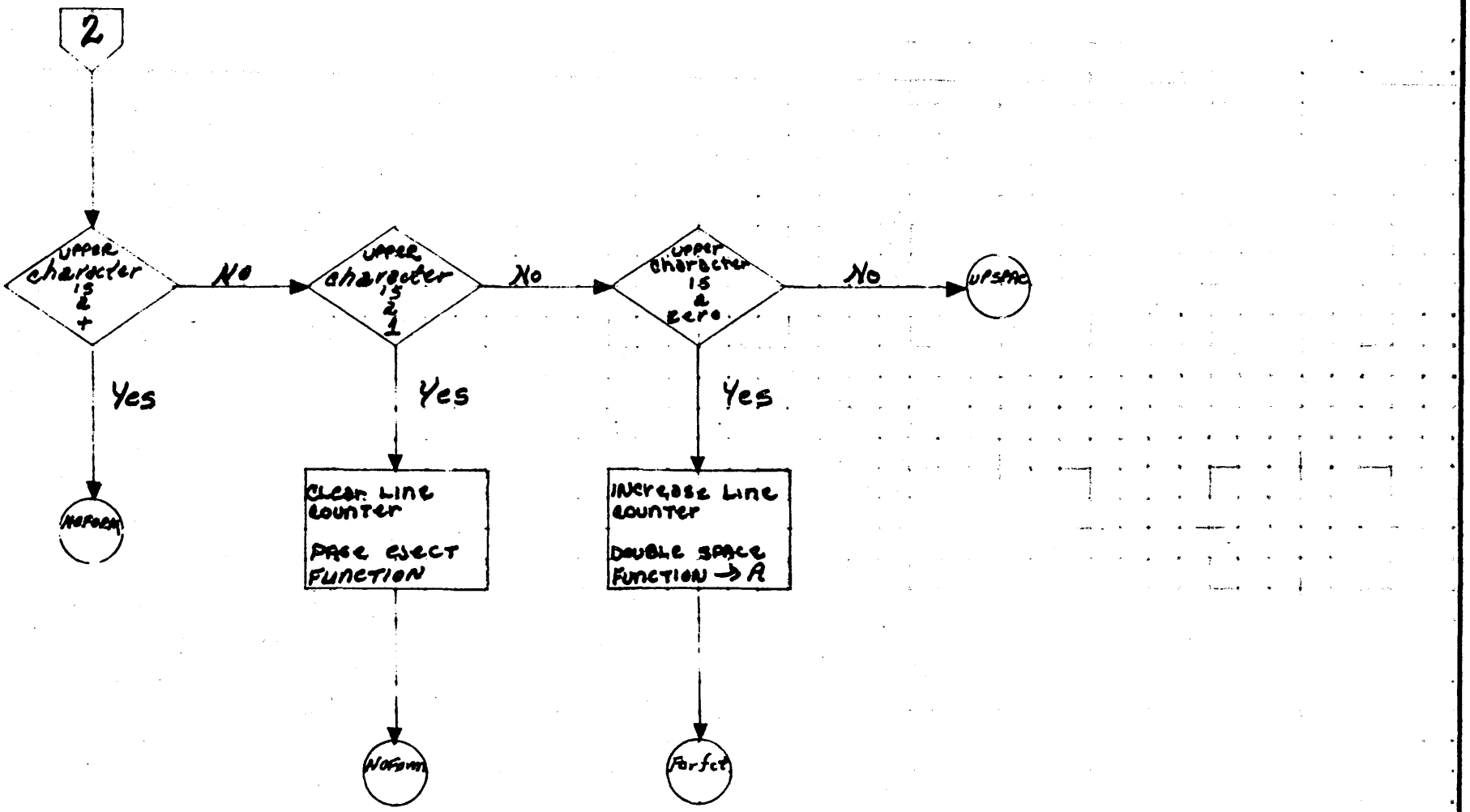
MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer	PROJECT MGR.			
	Driver PAGE 5 OF 17	PROJECT NAME			
	NUMBER D7178-1-E006-M2.1 ISSUE DATE	TASK NO.			
	DRAWN BY M.L.F. DATE 1-70	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

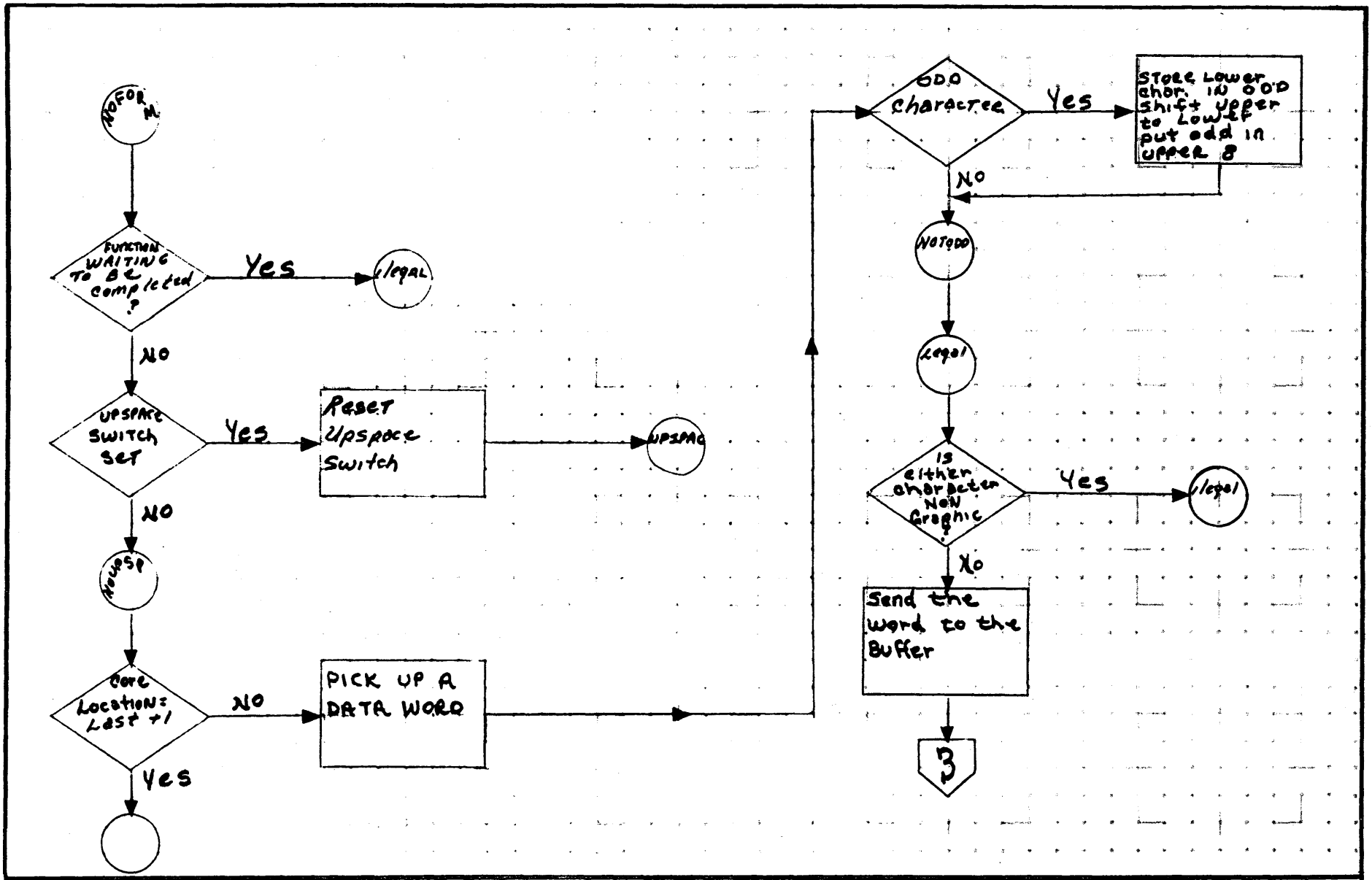


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1740/501 Line Printer Driver	PAGE 6 OF 17	PROJECT MGR.		
	NUMBER D717M1-E006#2.1	ISSUE DATE	PROJECT NAME		
	DRAWN BY M. L. F.	DATE 1-70	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971



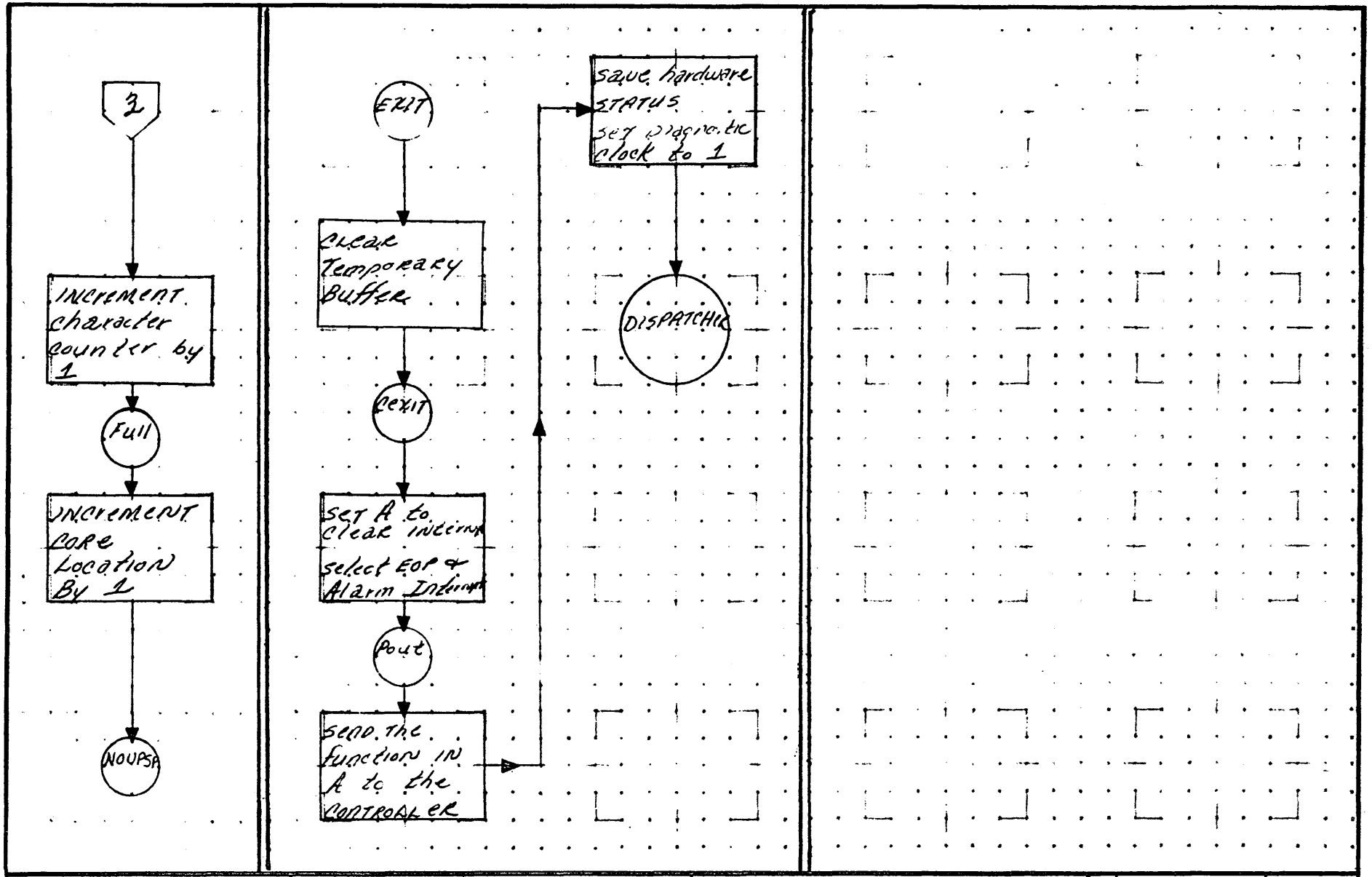
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer Driver PAGE 7 OF 17	PROJECT MGR.			
	NUMBER D717#1-E006#2.1 ISSUE DATE	PROJECT NAME			
	DRAWN BY M.L.F. DATE 1-70	TASK NO.			
		TASK NAME			

MAR 5 1971

Page 52.17

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



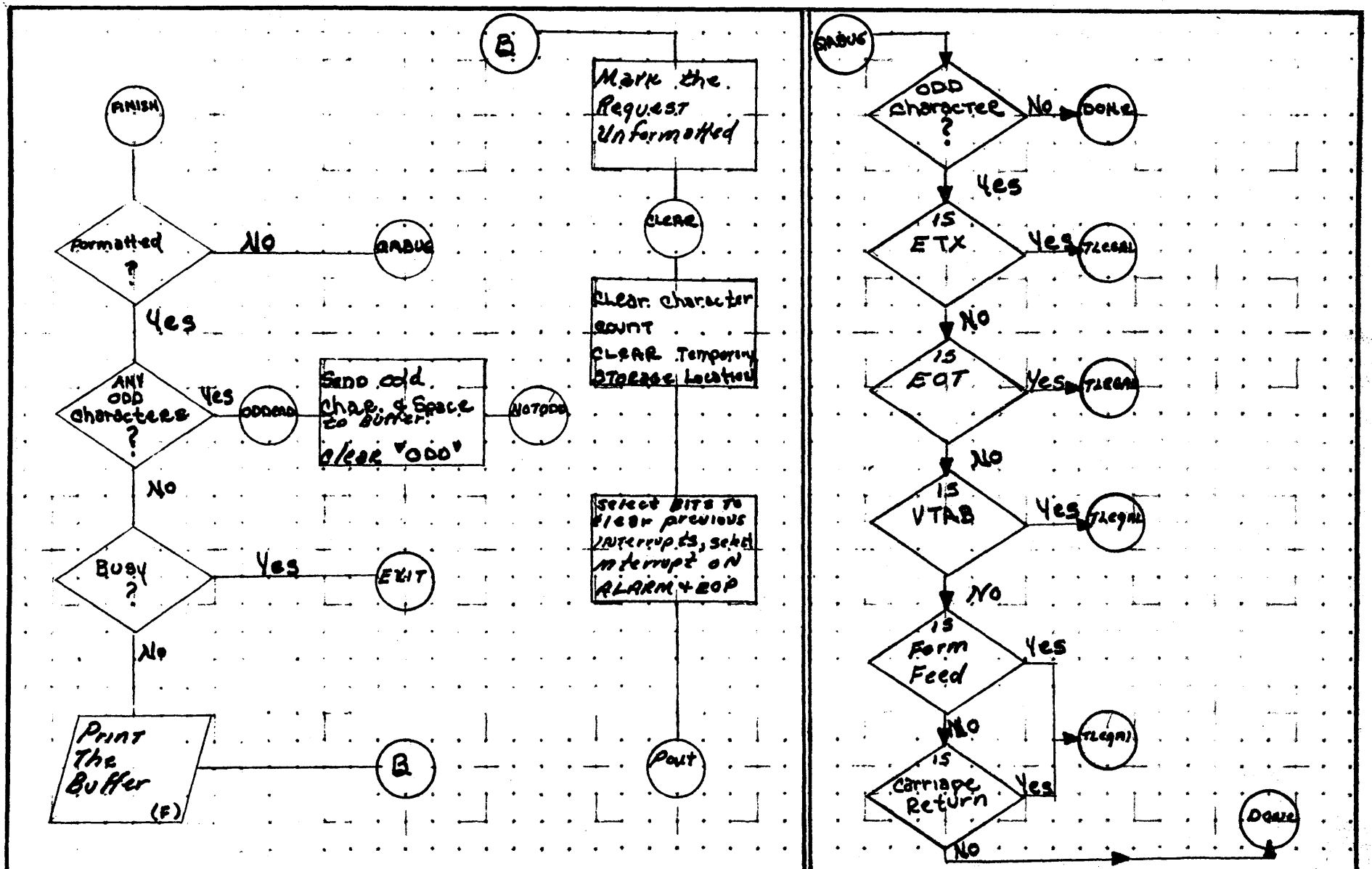
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1730/501 Line Printer		PROJECT MGR.		
FLOWCHART ☒	DRIVER	PAGE 8 OF 17	PROJECT NAME		
DECISION TABLE ☐	NUMBER 91721.Fo.002.1	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY M.L.F.	DATE 1-70	TASK NAME		

MAR 5 1971

Page 52.18

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



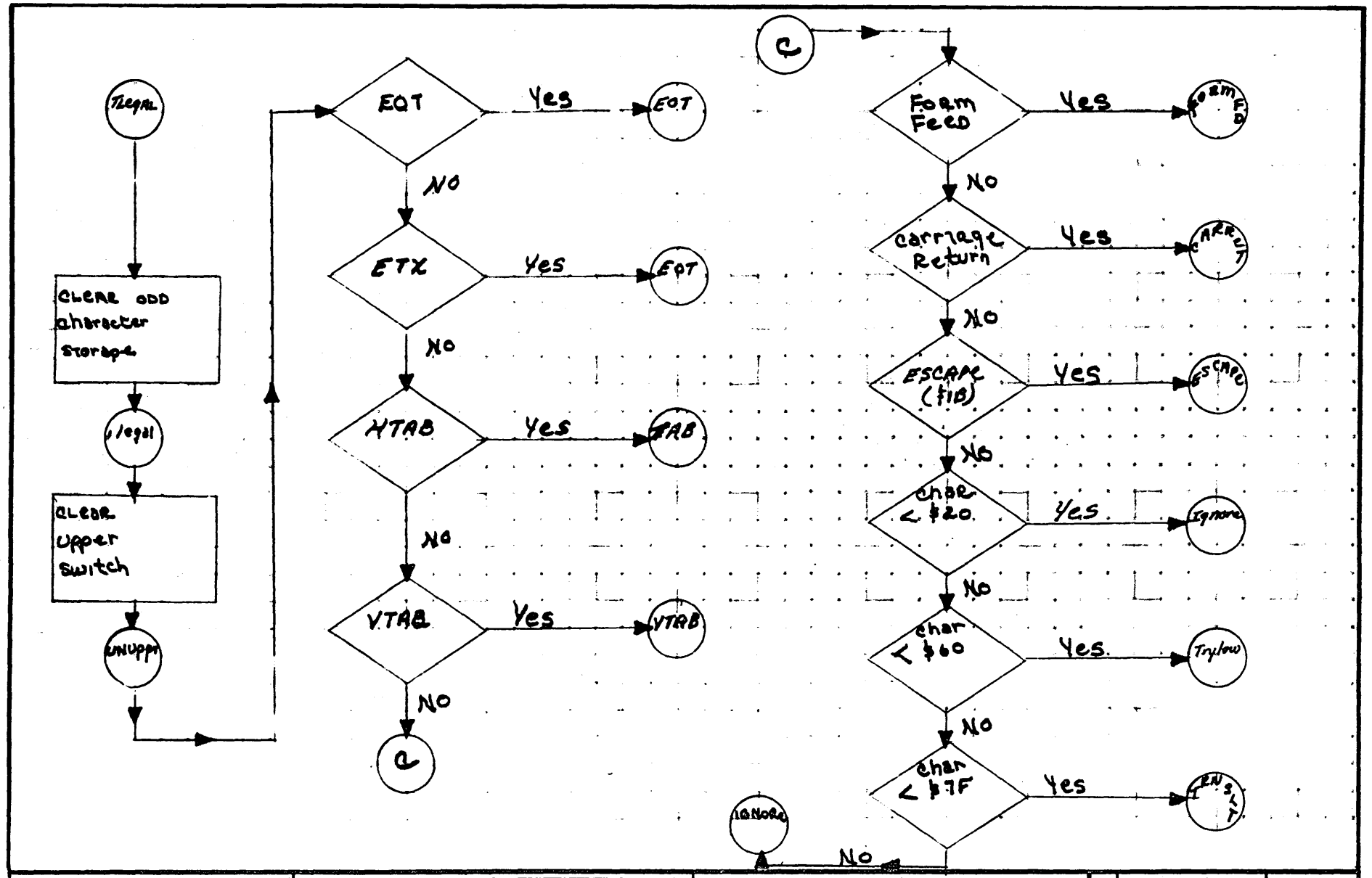
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer	PROJECT MGR.			
	Driver PAGE 9 OF 17	PROJECT NAME			
	NUMBER D717#1-E006#2.1 ISSUE DATE	TASK NO.			
	DRAWN BY M.L.F. DATE 1-70	TASK NAME			

MAR 5 1971

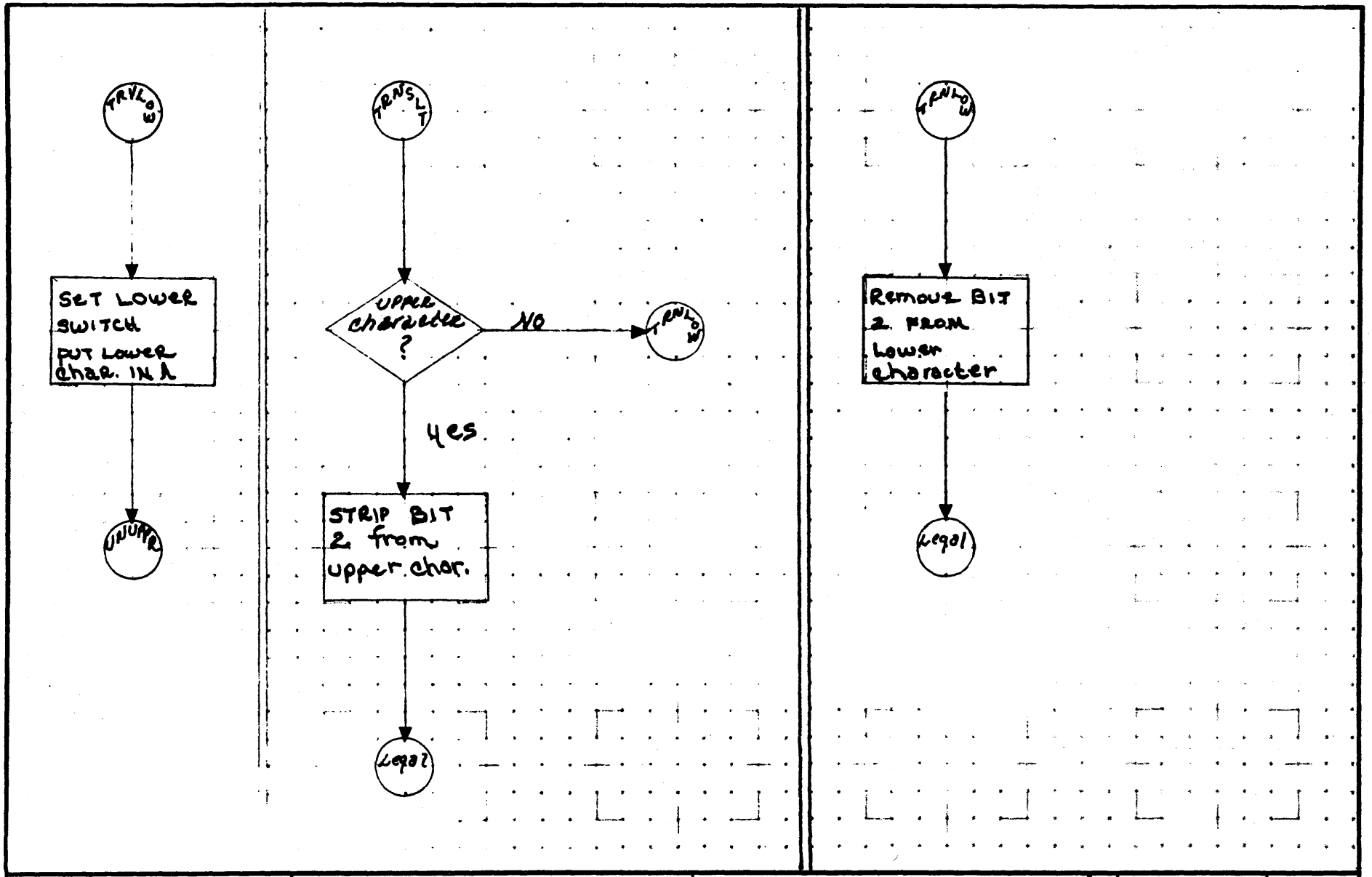
Page 52.19

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE 1240/501 Line Printer		PROJECT MGR.			
SAMPLE CODE ☐	Driver	PAGE 10 OF 17	PROJECT NAME			
FLOWCHART ☒	NUMBER A717B1-E0062.2	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY M.L.F	DATE 1-70	TASK NAME			
OTHER ☐						



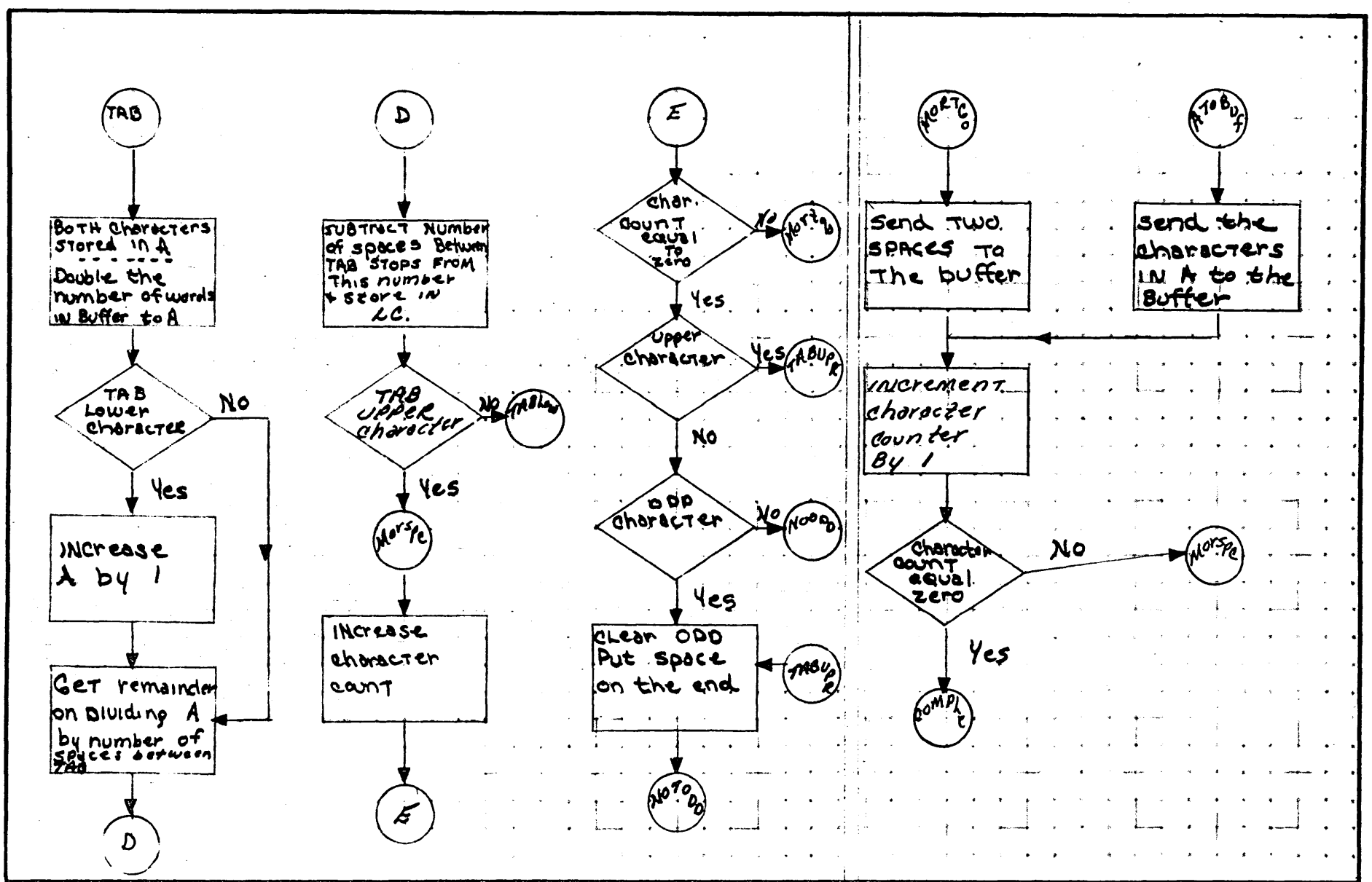
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1740/501 Line Printer		PROJECT MGR.			
FLOWCHART ☒	Driver	PAGE 11 OF 17	PROJECT NAME			
DECISION TABLE ☐	NUMBER D717#1-E006#2.1	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY M.L.F.	DATE 1-70	TASK NAME			

MAR 5 1971

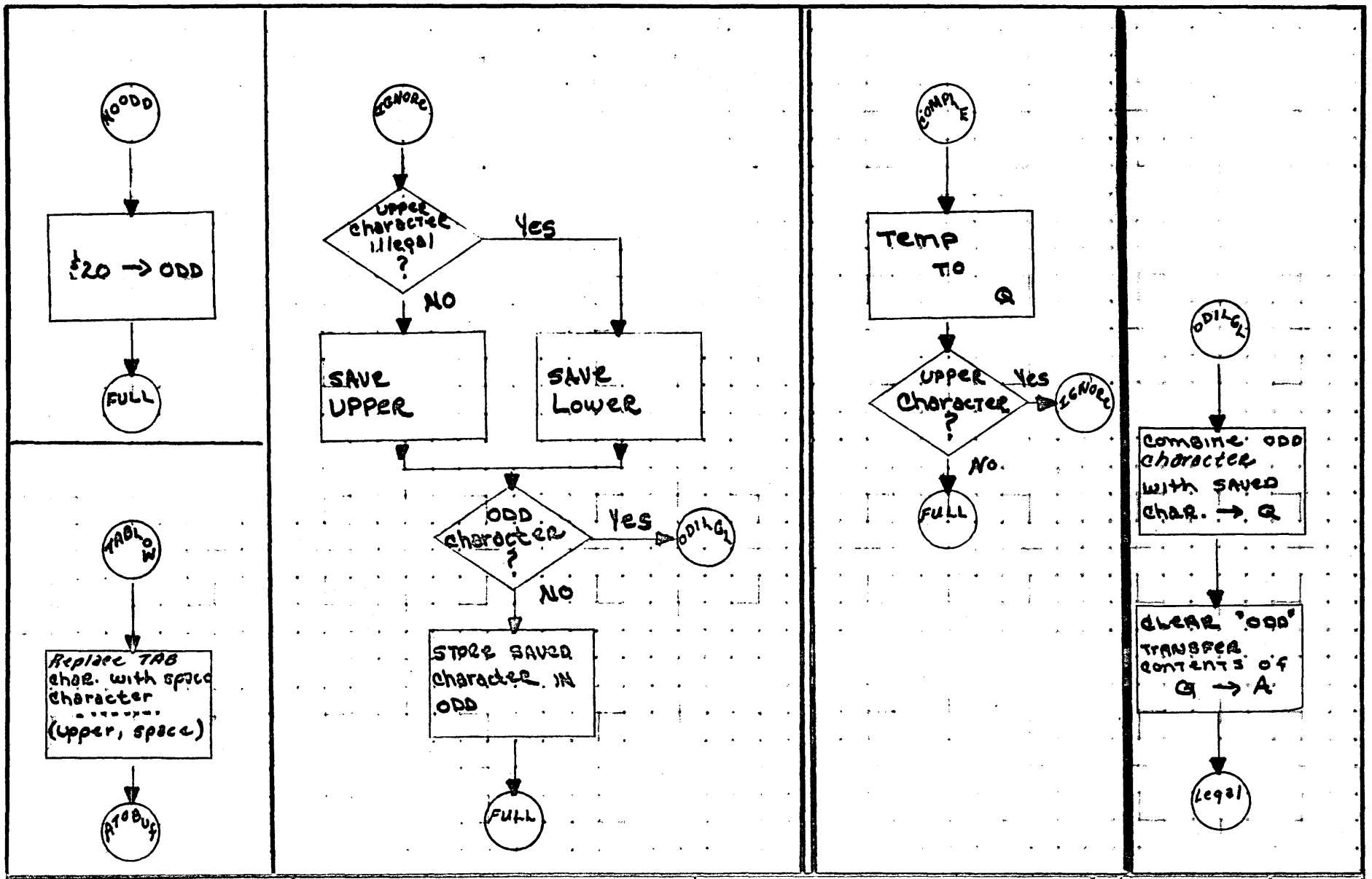
Page 52.21

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



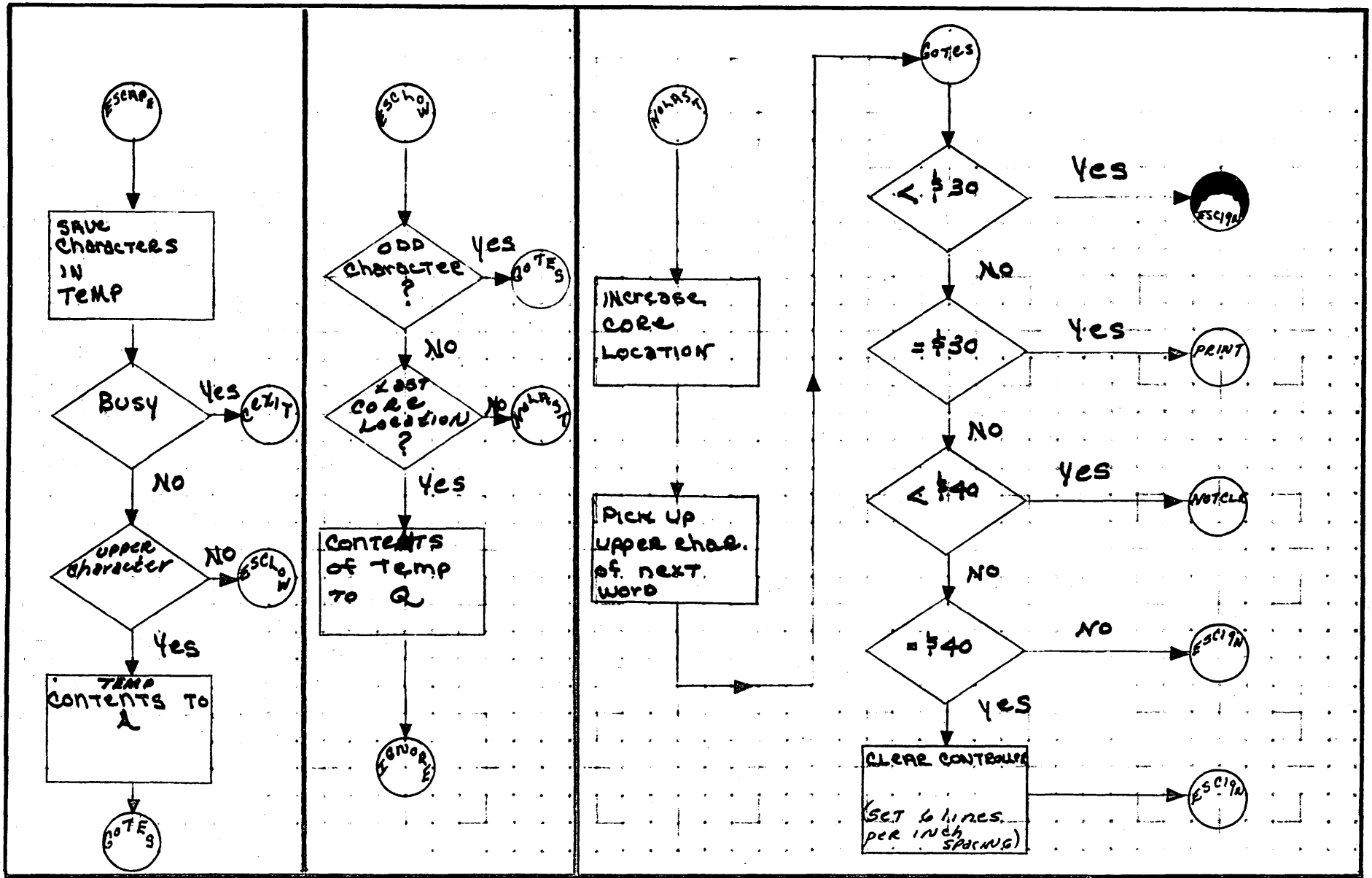
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer Driver PAGE 12 OF 17	PROJECT MGR.			
	NUMBER D717-01-E0002.1 ISSUE DATE	PROJECT NAME			
	DRAWN BY M.L.F. DATE 1-70	TASK NO.			
		TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 7600/501 Line Printer Driver PAGE 13 OF 17	PROJECT MGR.			
	NUMBER D717-1-Ronk02 ISSUE DATE	PROJECT NAME			
	DRAWN BY M.L.F. DATE 1-70	TASK NO.			
		TASK NAME			

AA1385 (FORMERLY CA127-1)

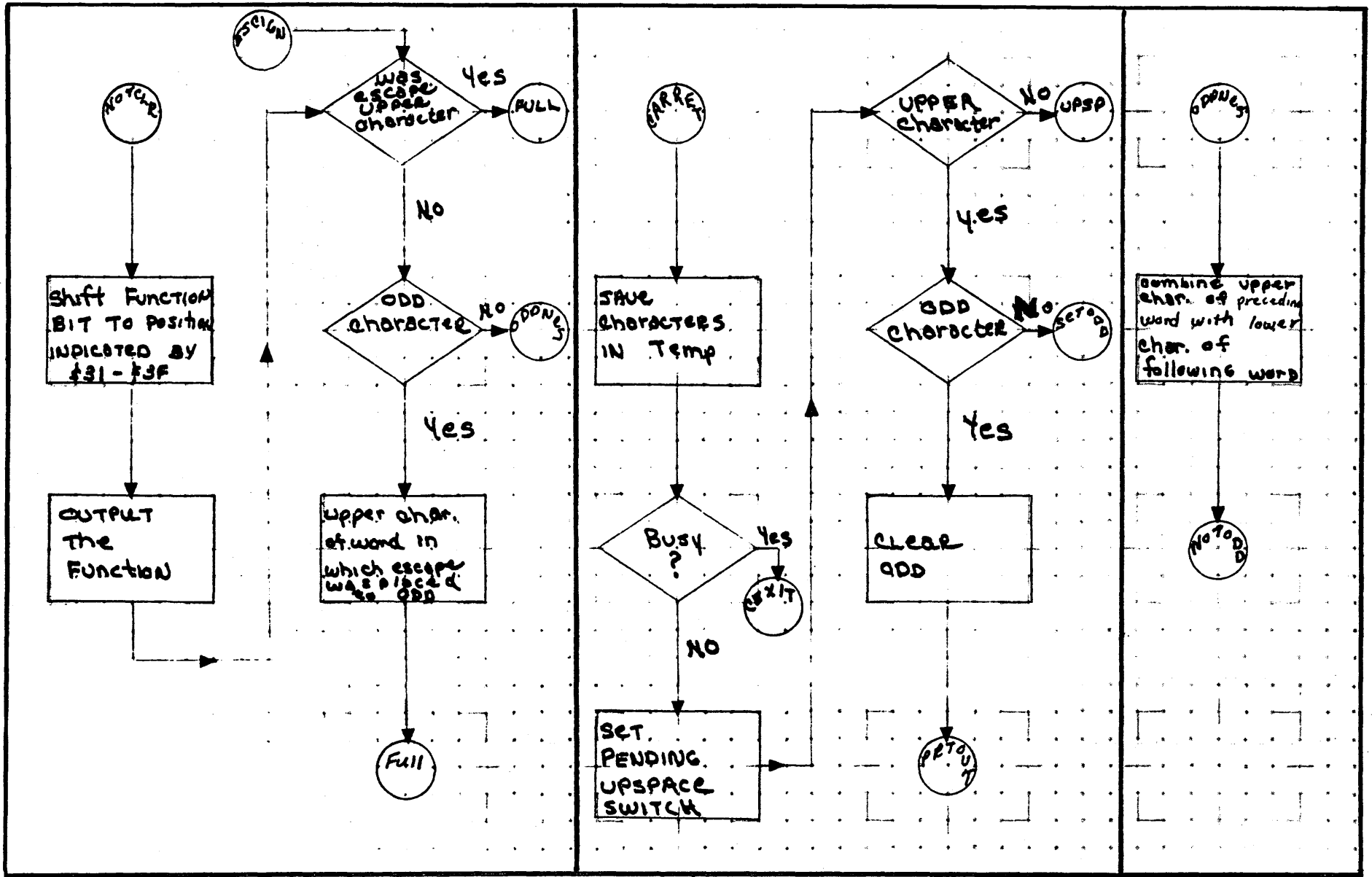
PRINTED IN U.S.A.



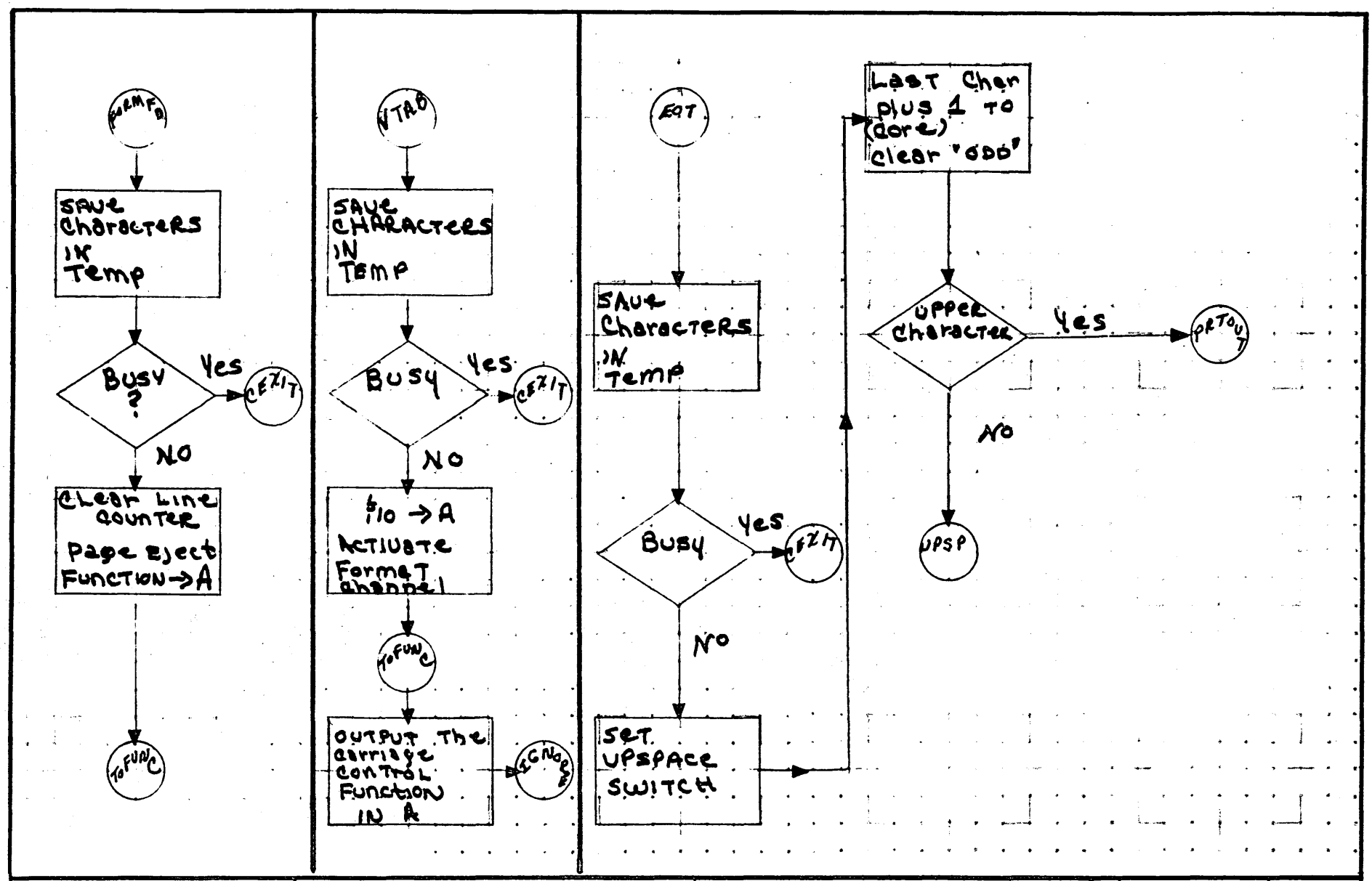
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1700/501 Line Printer Driver	PAGE 14 OF 17	PROJECT MGR.			
	NUMBER 60176101-E000 02.1	ISSUE DATE	PROJECT NAME			
	DRAWN BY M.L.F.	DATE 1-70	TASK NO.			
			TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



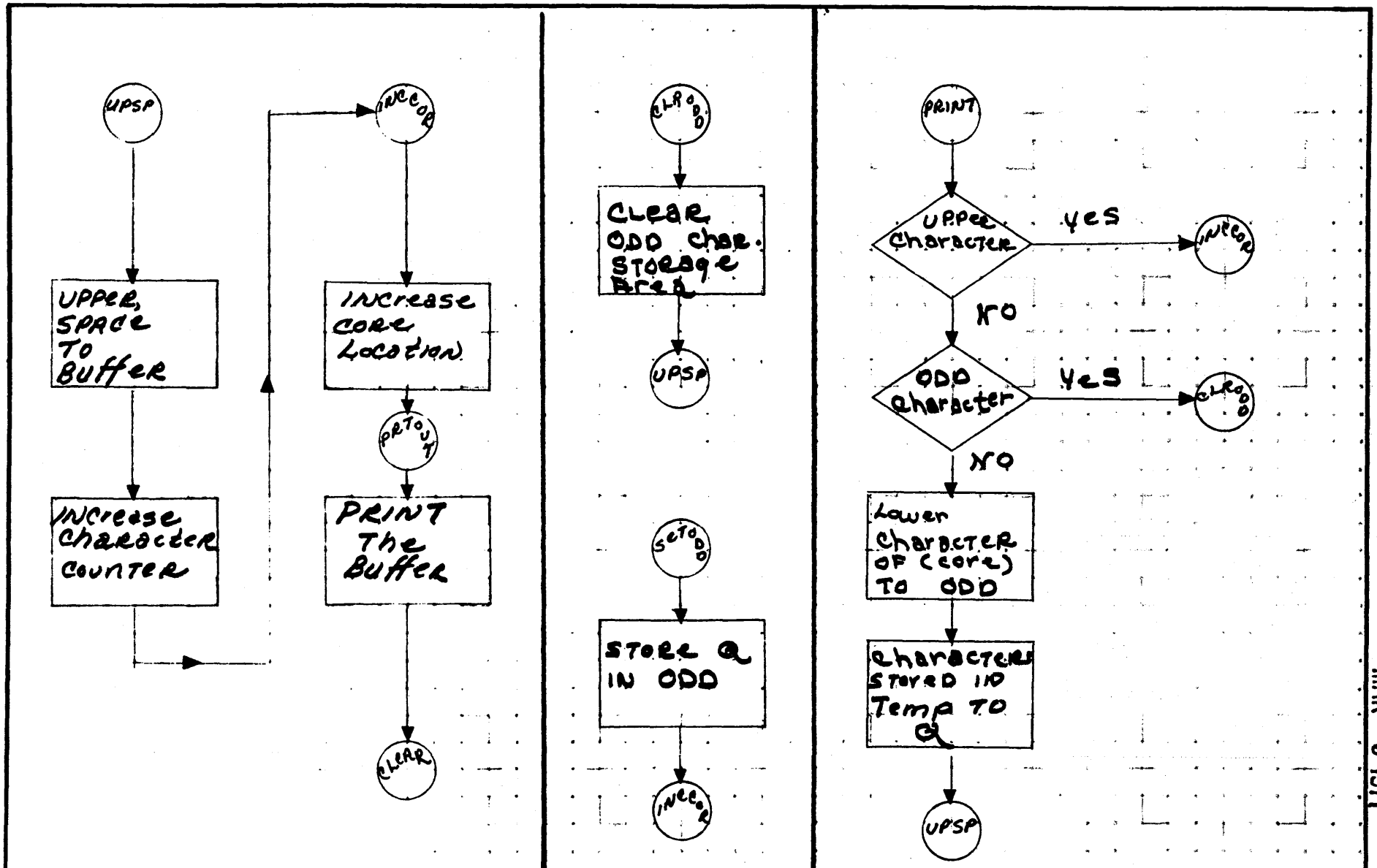
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer Driver	PAGE 15 OF 17	PROJECT MGR.			
	NUMBER D7174-E0696	ISSUE 2.1 DATE	PROJECT NAME			
	DRAWN BY M.L.F.	DATE 1-70	TASK NO.			
			TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1240/501 Line Printer Driver PAGE 16 OF 17	PROJECT MGR.			
	NUMBER D717-71-F00(M32.1 ISSUE DATE	PROJECT NAME			
	DRAWN BY M.L.F. DATE 1-70	TASK NO.			
		TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1740/501 Line Printer	PROJECT MGR.			
	Driver PAGE 17 OF 17	PROJECT NAME			
	NUMBER D717 #1-Feature2. ISSUE DATE	TASK NO.			
	DRAWN BY M.L.F. DATE 1-70	TASK NAME			

MAR 5 1971

Page 52.27

53.0 1731/601 Unbuffered Magnetic Tape Driver

53.1 Introduction

This tape driver is composed of six modules which may be combined in various ways to make up a tape driver. The modules are:

TAPEDR - The traffic director and tape motion control processor.

T14 - The tape motion control request processor.

RWBA - Processes the READ and WRITE requests in either binary or ASCII mode.

FRWA - Processes the FREAD and FWRITE requests in ASCII mode.

FRWB - Processes the FREAD and FWRITE requests in binary mode.

RECOVR - Error recovery routine.

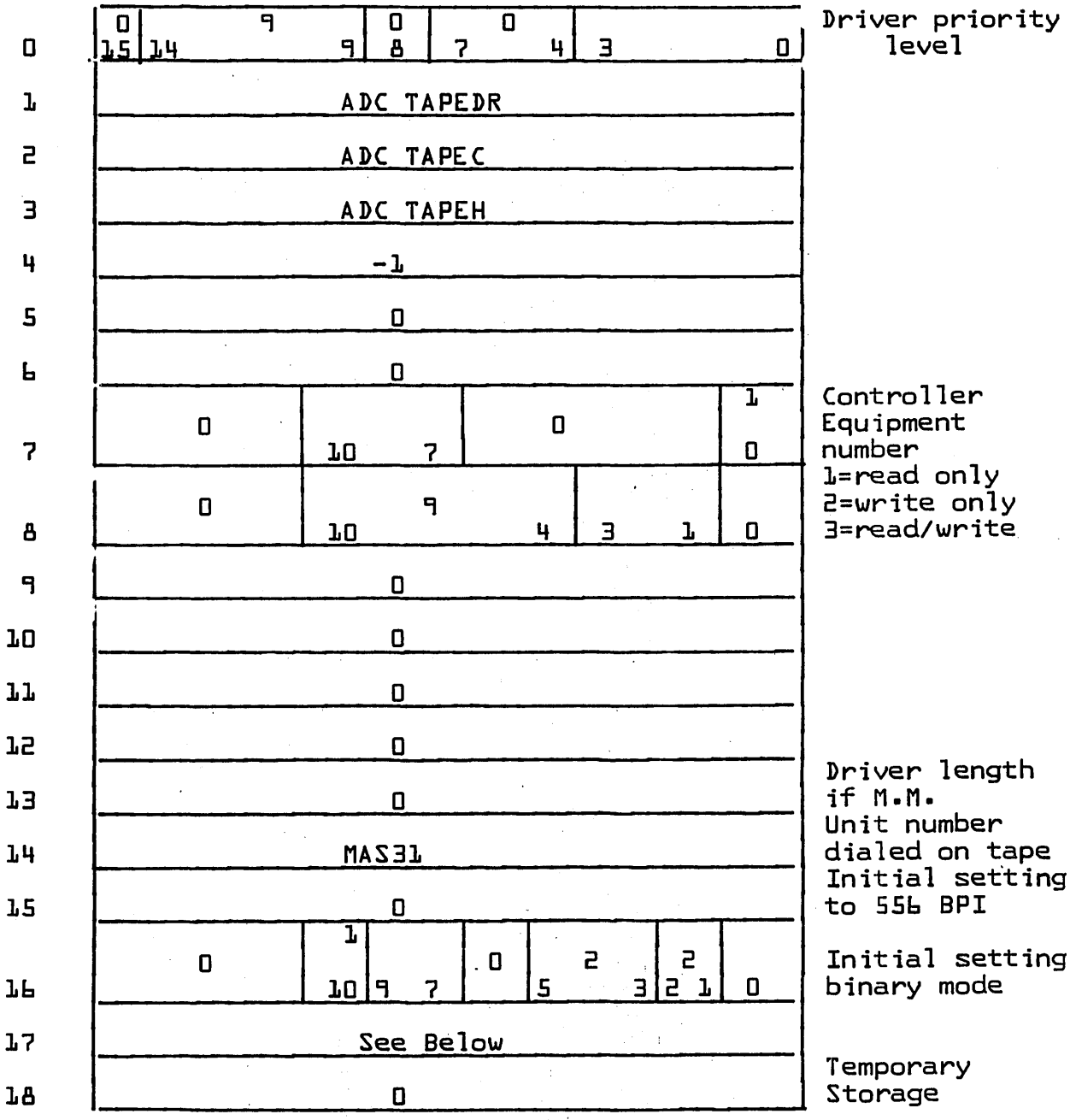
TAPEDR must always be included as all requests pass through it. If density changes and/or tape motion control is desired T14 must also be included. If T14 is not included, a user call will result in an illegal request diagnostic and job termination. In order to perform a read or write operation on magnetic tape any combination of RWBA, FRWA and FRWB routines must be included in the tape driver. A request for one of these routines which is not in resident will result in an illegal request diagnostic and job termination. RECOVR may be included in the driver or replaced or omitted at the users option. If an error is detected and this routine is not in resident, the operator will be notified of an unrecoverable error (Alternate Device Handler) and given an opportunity to take action.

Module	Entry Points	Externals
TAPEDR	TAPEDR	FRWA
	TAPEC	FRWB
	TAPEH	RWBA
	CKBVS4	ALTDEV
	URECOV	MAKEQ
	NORMRT	
	RETRY	
	RETRY2	
	CKEOP	
	TP5	

Module	Entry Point	Externals
	CKEOP1	
FRWA	FRWA	TP5
		URECOV
		CKEOP1
		RECOVT
		NORMRT
FRWB	FRWB	TP5
		URECOV
		CKEOP1
		RECOVT
		NORMRT
RWBA	RWBA	TP5
		URECOV
		CKEOP1
		RECOVT
		NORMRT
		CKBUSY
RECOVT	RECOVT	CKBUSY
	ENDOP	RETRY
		RETRY2
		RWBA
		FRWA
		FRWB
		CKEOP
		TP5
		LOG

53.2 Physical Device Table

Each tape unit in the system will have a PHYSTB (physical equipment table) entry as follows:



WORD 17 contains the location of WORD 0 of the PHYSTB entry of another tape unit connected to the same controller. Through this word, all the tape units connected to a controller are threaded together in a circular manner.

If only one tape unit is connected to a controller, word 15 must contain the address of its own PHYSTB entry.

53.3 Assumptions

1. The driver must not be interrupted while performing a data transfer.

2. The driver can service on only one tape at a time.

3. All callers must be at lower priority levels.

53.4 TAPEDR Description

This module processes all the requests to the tape driver and sends them to the proper routine. It also contains the tape motion control request routine and several subroutines used by all portions of the driver.

53.4.1 INTERNAL DESCRIPTION

On entry, the driver checks to see if another request is active. If another request is active, exit is made to the dispatcher as the driver will pick up the current request later on. If no request is in progress, the driver waits till the controller becomes inactive and then proceeds with request processing.

Two types of requests are processed, tape motion and data transfer. Suitable table setups are made for each type. When table setups are complete, the unit is connected and mode and density selected. The requests are then passed to a corresponding processing routine is not present, job termination results.

53.4.2 USE OF INTERRUPTS

Interrupts are used to signal completion of the following tape motion operations:

1. Backspace record
2. Rewind
3. Mark end of file

While the above operations are in progress the driver has relinquished control. All other operations require that the driver retain control until they are completed. A partial exception is the unload request. The driver initiates this request and exits. No completion is signalled by the controller.

53.4.3 TPMCTL

At TPMCTL a parameter is extracted from WORD 11 and placed in the Q register to use as an ordinal to the request table. The remaining parameters are restored temporarily in WORD 11. Using the ordinal in Q the proper routine address is picked up and placed in Q and a jump is made to the effective address formed by the value in Q plus one. The table format is:

TPMTBL + 0	JMP	DONOTH
+ 1	JMP	BSPR
+ 2	JMP	MARKEF
+ 3	JMP	REWIND
+ 4	JMP	UNLOAD
+ 5	JMP	SKIPF

At DONOTH the tape motion indicator is cleared and exit is made to TP6.

At BSPR the control function to select a backspace and interrupt is placed in the accumulator and control is given to TMP1A.

At MARKEF the control function to select a write or file mark and interrupt is placed in the accumulator and control is given to TPM1.

At REWIND the control function to select a rewind and interrupt is placed in the accumulator and control is given to TPM1A.

At UNLOAD, WORD 11 for this logical unit is set to zero to force DONOTH to be processed as the next parameter. This will terminate this request as the tape unit will require manual intervention by the operator before it may again be addressed. The control function to select a rewind and unload is placed in the accumulator and control is given to TPM1.

At SKIPF a non stop read is performed and at each end of operation status is checked for end of file and end of tape. When either is detected control is given back to TPMCTL to process the next parameter.

At TPM1A status is checked to see if tape is positioned at load point. If it is, control is given back to TPMCTL to process the next parameter thus treating the current parameter as a no operation. If the unit is not at load point, control is given to TPM1.

At TPM1 the function in the accumulator is issued to the tape unit and if interrupt was not selected, control is given to TPM2. If interrupt was selected, this unit is set to busy (bit 15=1), the "I" register (the logical unit number) is saved in SAVEI and exit is made to the dispatcher.

At TPM2 the subroutine CKEOP is entered to check for end of operation. If no end of operation, the unit is checked for ready. If ready control is returned to TPM2. If not ready it is assumed that the last operation performed was an unload, and the CKEOP routine is re-entered to EOP to put status

into WORD 12 for this logical unit. EOP is also used when an end of operation was encountered and after status has been placed in WORD 12 control is returned to the end of operation line in the call to CKEOP where control is given back to TPMCTL to process the next parameter.

At end of operation interrupt time the external interrupt processor (EPROC) gets the interrupt time address and enters TAPEC where the logical unit from SAVEI is restored in the "I" register, this unit is set not busy, a function code to clear the interrupt is placed in the accumulator and control is given to TPM2.

CYCLE is entered at the completion of each request. It examines (via thread in WORD 17) all the tapes connected to the controller. If it finds one that needs service, the PHYSTD location for that tape unit is put in "I" and a request processed for it. In this way, the tapes are activated in a comutative manner until all requests to all units have been processed. When this occurs, exit will be made to the dispatcher.

53.5 T14 - TAPE MOTION CONTROL REQUEST PROCESSOR

This request processor adjusts return address for direct calls.

When the above is completed, TAPE exits to entry point "SAVLU" in the Read/Write Request Processor where threading and driver scheduling take place. All legality checking for unprotected requests is done in the Protect Processor
It is assumed that protected requests do not require legality checking.

53.6 RWBA - READ AND WRITE IN BINARY OR ASCII MODE

Upon entry to RWBA the contents of WORD 7 for this logical unit are stored in local storage EQTAB with bit 1-0 set to zero. The first word address is stored in CORE and the last word address + 1 (CORE plus the requested number of words) is stored in last. If this is a write request control is given to WBA1. If not the function to set read motion is issued. Each character is then input to the A register and packed three to a word drop the two low order bits of the last character. Partial words are saved in TEMP until three characters have been assembled at which time TEMP is stored in the address specified by the contents of CORE. CORE is then increased by one and control is returned to input three more characters until CORE and LAST become equal at which time a tight read loop is entered until there are no more characters to read. No more characters are transferred to the requestor. At end of operation if three characters have not been assembled into TEMP they are left justified and stored in the address specified in CORE with the lower order unfilled bits set to zero unless CORE and LAST are equal.

If CORE and LAST are equal nothing is transferred to the requestor. In either case the Q register is set to zero for the error recovery code and at RBA8 parity and lost data errors are checked for. If an error exists and the external address of the error recovery package (RECOVR) is equal to $7FFF_{16}$ the routine is not resident, CORE is loaded into the A register and LAST into Q and control is given to URECOV. If the external is patched RECOVR is entered. If RECOVR returns to this point CORE and LAST are put in A and Q and control is given to URECOV.

At WBA1 the unit is checked to see if it is write enabled. If it is not, CORE and LAST are placed in A and Q and control is given to URECOV. If it is write enabled write motion is initiated and the word at the location specified in CORE is placed in the A register. Three shift left 6 and output instructions are performed on the A register. CORE is incremented by one and control is given back to WBA to get the next word until CORE and LAST are equal. When they are equal the subroutine CKEOP is entered and looped on until end of operation comes in. When this happens the error recovery code 1 is placed in Q and control is given to RBA8 to check parity and lost data.

53.6.1 ASCII/BCD CONVERSION TABLE

TABLE + 00	26	IF
+ 01	31	31
+ 02	32	32
+ 03	33	33
+ 04	34	34
+ 05	35	35
+ 06	36	36
+ 07	37	37
+ 08	38	38
+ 09	39	39
+ 0A	30	21
+ 0B	3D	22
+ 0C	27	23
+ 0D	3A	24
+ 0E	3E	25
+ 0F	22	26
+ 10	20	27
+ 11	2F	28
+ 12	53	29

TABLE + 13	54	12
+ 14	55	13
+ 15	56	14
+ 16	57	15
+ 17	58	16
+ 18	59	17
+ 19	5A	18
+ 1A	5F	19
+ 1B	2C	3D
+ 1C	28	1E
+ 1D	25	2D
+ 1E	5C	2F
+ 1F	40	1A
+ 20	2D	10
+ 21	4A	2A
+ 22	4B	0F
+ 23	4C	3F
+ 24	4D	2B
+ 25	4E	1D
+ 26	4F	1D
+ 27	50	0C
+ 28	51	1C
+ 29	52	3C
+ 2A	21	2C
+ 2B	24	30
+ 2C	2A	1B
+ 2D	5D	20
+ 2E	3B	3B
+ 2F	5E	11
+ 30	2B	0A
+ 31	41	01
+ 32	42	02
+ 33	43	03
+ 34	44	04
+ 35	45	05

TABLE + 36	46	06
+ 37	47	07
+ 38	48	08
+ 39	49	09
+ 3A	3F	0D
+ 3B	2E	2E
+ 3C	29	3E
+ 3D	5B	0B
+ 3E	3C	0E
+ 3F	23	3A

53.7 FRWA - FORMAT READ AND WRITE IN ASCII MODE

At FRWA the contents of WORD 7 for this logical unit are stored in the local storage EQTAB with bits 1-0 set to zero. The first word address -1 from the request is placed in CORE and the LAST word address (CORE plus the number of words requested) in LAST. If this is a write request, control is given to FRWA1. If not, read motion is initiated at FRDA6. At FRDA the first character is inputted to A and placed in Q to use as an ordinal to the conversion tables. The upper 8 bits of the table entry at TABLE + the value in Q is placed in the lower 8 bits of TEMP and CORE is incremented by one. The difference between CORE and LAST is stored in ENDCK and the second character is inputted and converted in the same fashion as the first. The character in TEMP and this character are packed into one word and stored in the location specified by the contents of CORE. If ENDCK is not zero control is given back to FRDA. If it is zero, a tight input loop is entered to the rest of the record but no more words are transferred to the requestor. In any case, when reading finally terminates on end of operation if two characters have not been read but at least one has the character is left justified and the lower set to FF_{16}. CORE and LAST are both incremented by one, Q is set to 2 for the error recovery code and at FRDA5 an end of file is checked for. If an end of file has been read control is given to FRDA7 where CORE is placed in A, LAST is placed in Q and exit is made to NORMRT. If no end of file, parity and lost data are checked. If no parity or lost data errors exist control is given to FRDA7. If an error exists to the external for RECOVR is checked for 7FFF (unpatched). If unpatched control is given to URE. If patched the routine exits to RECOVR. If RECOVR returns control to URE, CORE and LAST are placed in A and the routine exits to URECOV.

At FWRA1 the unit is checked for a write enable ring. If it is not write enabled control is given to URE. If it is, write motion is initiated and CORE and LAST are both incremented by one. At FWRA the first character or upper half of the word at the location specified by the contents of CORE is placed in Q with the two highest order bits set to zero and it is used as an ordinal to the conversion table the same as in a read operation. On a write, however, the lower half of the table entry contains the desired conversion character. This character is then outputted and the next character is picked up and processed in the same manner. When two characters have been output CORE is incremented by one and control is given back to FWRA for the next character until CORE and LAST become equal. At that time the subroutine CKEOP is looped on until end of operation. At end of operation the error recovery code of 3 is placed in Q and control is given to FRDA5 to check parity, etc.

53.8 FRWB - FORMAT READ AND WRITE IN BINARY MODE

Upon entry to FRWB the contents of WORD 8 for this logical unit are stored in local storage EQTAB with bits 1-0 set to zero. The first word address is stored in CORE and the last word address + 1 (CORE + number of words requested) is stored in LAST. If this is a write request control is given to FWRB1. If it is not read motion is initiated at FRDB1. At FRDB is a chain of eight input sequences.

Every eight characters read will form (3) 1700 words. As sixteen bits are read they are stored, CORE is incremented by one and checked for equality with LAST. When they are equal control is given to SPIN where a tight read loop reads to the end of operation without transferring any characters to the requestor. When end of operation occurs and less than sixteen bits has been read to fill the current word, the data is left justified with the unfilled lower order bits set to zero and stored in the address specified by CORE. At the end of operation the error recovery code 4 is placed in Q and if no file mark has been read parity and lost data are checked at PARCK. If a file mark was read or no error was detected control is given to FWB7 where CORE is placed in A and LAST in Q and the routine exits to NORMRT. If an error exists the external reference to RECOVR is checked for $7FFF_{16}$ (unpatched). If it is unpatched control is given to FWB3. If it is patched the routine exits to the error recovery package RECOVR. If RECOVR returns control to this point control is at FWB3 and CORE and LAST are placed in A and Q and the routine exits to URECOV.

At FWRB1 the tape is checked for a write enable ring. If none exists control is given to FWB3. If a ring is present write motion is initiated and the reverse unpacking of the data at the address

specified by the contents of CORE is performed with a series of eight output sequences. Each six bits unpacked starting from bit 15 of the first word and working toward bit 0 of the last word is outputted as a character on tape. If the output sequence ends (i.e., CORE equals LAST) and a full six bits is not left the lower order bit of the character are set to zero. In any event when CORE is equal to LAST the record has been transmitted and the subroutine CKEOP is looped on until end of operation. At end of operation, the error recovery code 5 is placed in Q and control is given to PARCK.

53.9 RECOVR - THE ERROR RECOVERY PACKAGE

This routine may be replaced at the users option. Its entry point is labeled RECOVR. It is return jumped to with an error recovery code in Q (range is 0-5 with even numbers representing reads and odd numbers representing writes).

Upon entry to RECOVR the error recovery code is divided by 2 and saved in RCODE. If it was odd the routine goes to WRETRY. If it was even control is given to ODD. At ODD if RETRY is not equal to three it is incremented by one at P1, the tape unit is backspaced one record and control is given to XIT. If it is equal to three, control is given to CLEAN where RETRY2 is checked. If RETRY2 is equal to four the routine exits back to the calling routine through its entry point. If RETRY2 is not equal to four control is given to CLEAN2. At CLEAN2, RETRY2 is incremented by one, ENDOP is set to zero and up to three backspaces are issued. If load point is sensed after any of the backspaces, no more backspaces are done. A count of the number of extra backspaces actually executed is kept in ENDOP. A non stop read is done and continues until the number of end operations specified in ENDOP have been read. At this point the error record is the next record to be read and control is given to XIT.

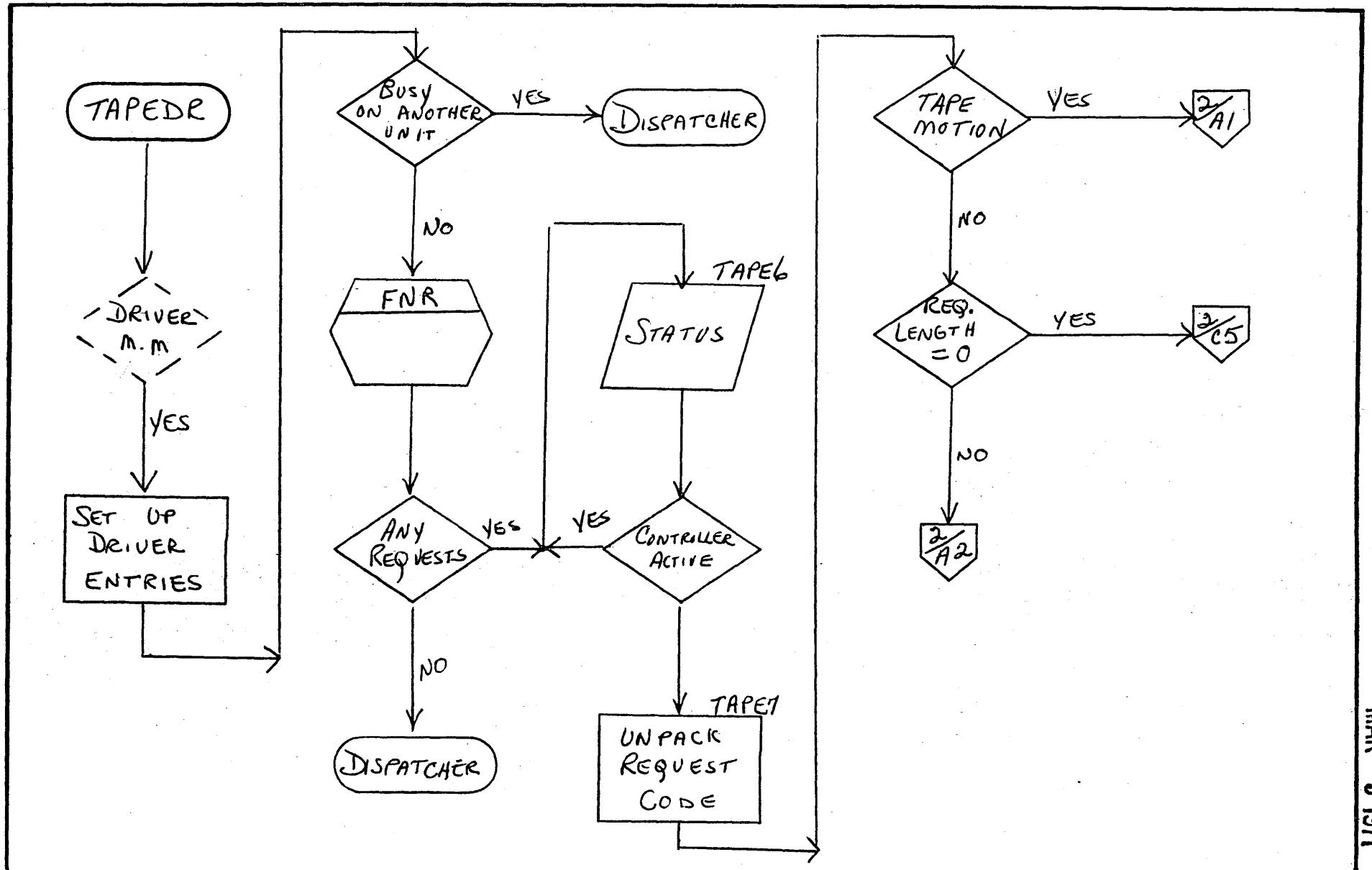
At XIT the routine determines which routine to re-enter by using the value in RCODE as an ordinal to a table of addresses. The format of the table is:

TBL + 0	JMP	RWBA
+ 1	JMP	FRWA
+ 2	JMP	FRWB

At WRETRY if RETRY is not equal to two, control is given to R1. If it is equal to two control is given to W1. At W1 if RETRY2 is equal to two control is given to W6. If it is less than two control is given to W3 and if it is greater than two control is returned to the calling routine through the entry point of RECOVR.

At W3 the tape unit is backspaced and at least half of the record is erased. The number of erases to perform is calculated by the formula $\frac{(N-1)C}{12D} + 1$ where N is the number of computer words to be written, C is the number of characters per computer word and D is the density the data is to be recorded in. The erases are performed by writing a file mark and backspacing. When the computed number of erases have been performed control is given to W6.

At W6 zero is stored in RETRY and control is given to XIT.

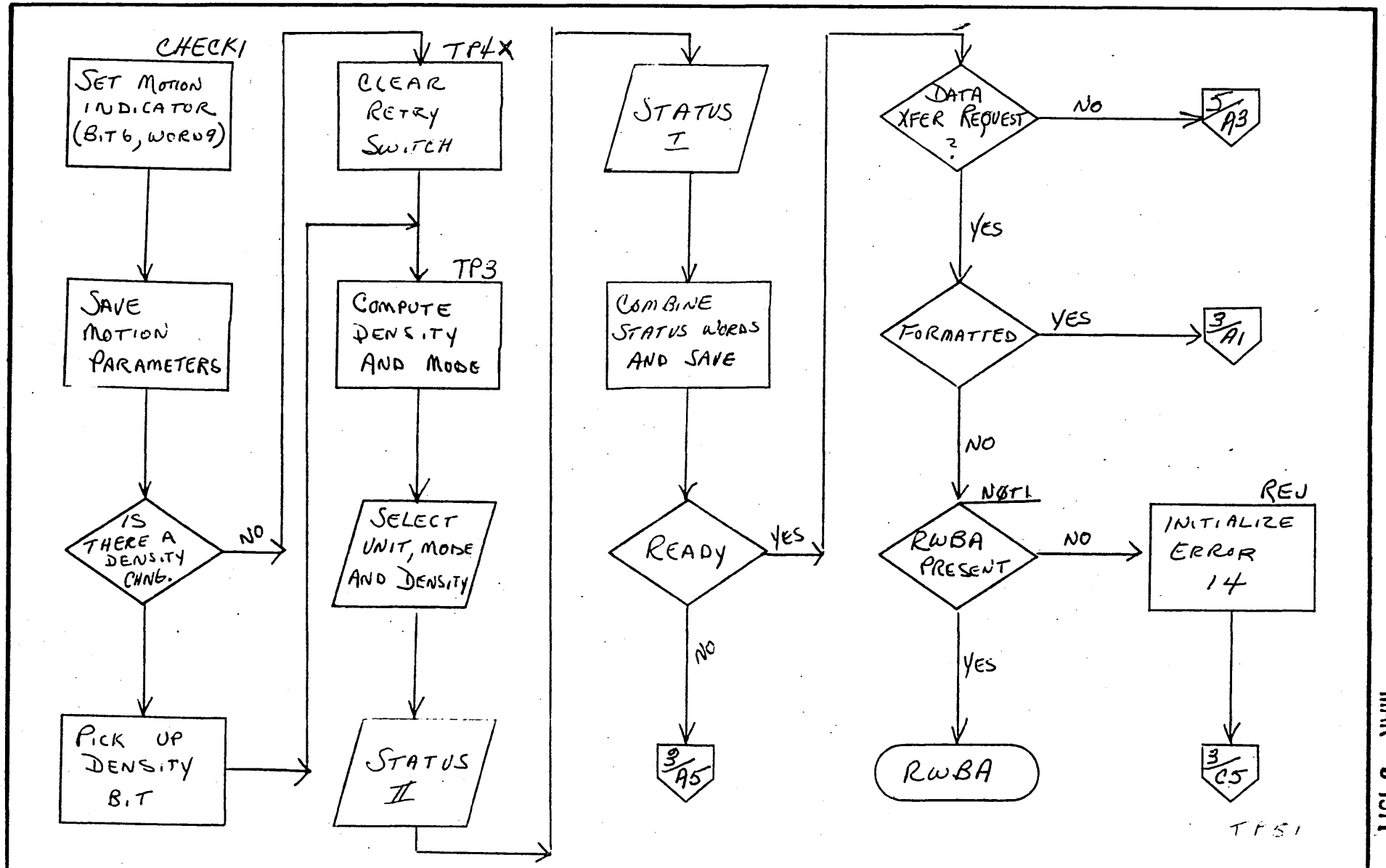


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPEDR	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER UNBUFFERED	PAGE 1 OF 29	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

53.14

AA1385 (FORMERLY CA127-1)

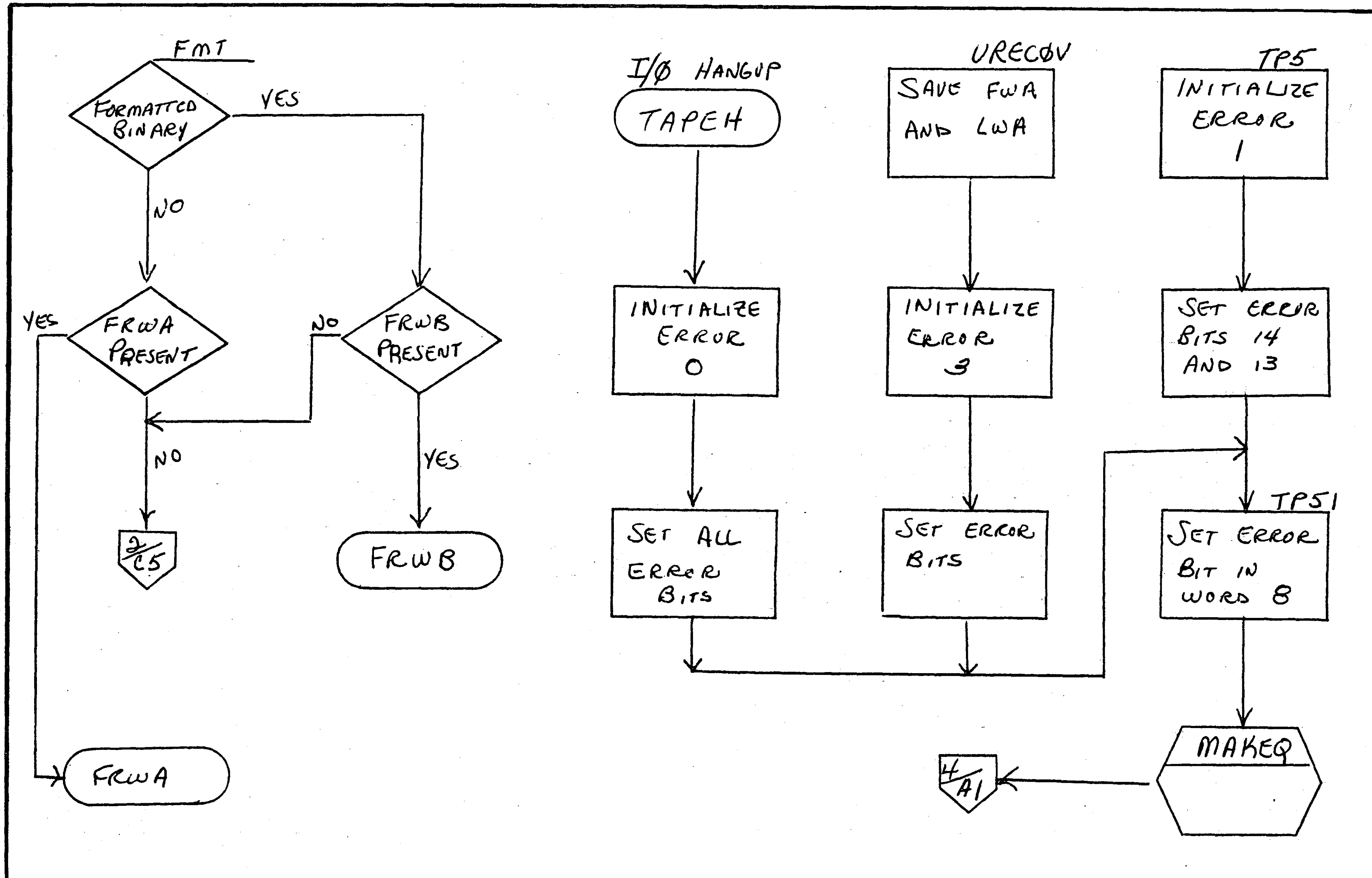


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPEDR	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	UNBUFFERED	PAGE 2 OF 29	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.15

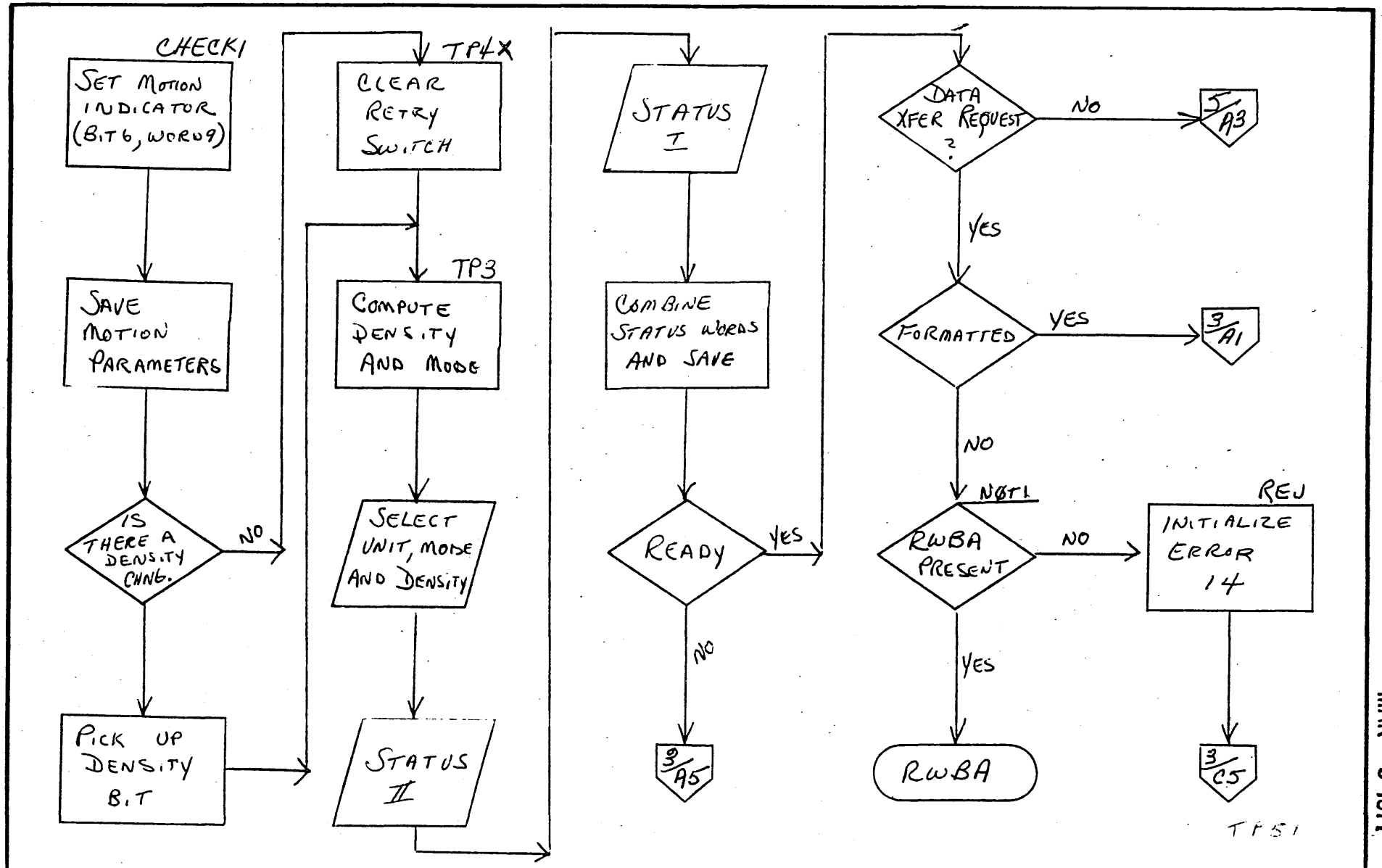


MAR 5 1971

CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPEDR
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.
FLOWCHART ☐	UNBUFFERED	PAGE 3 OF 24	PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.
OTHER ☐	DRAWN BY	DATE	TASK NAME

REV	APPROVED	DATE

53.16

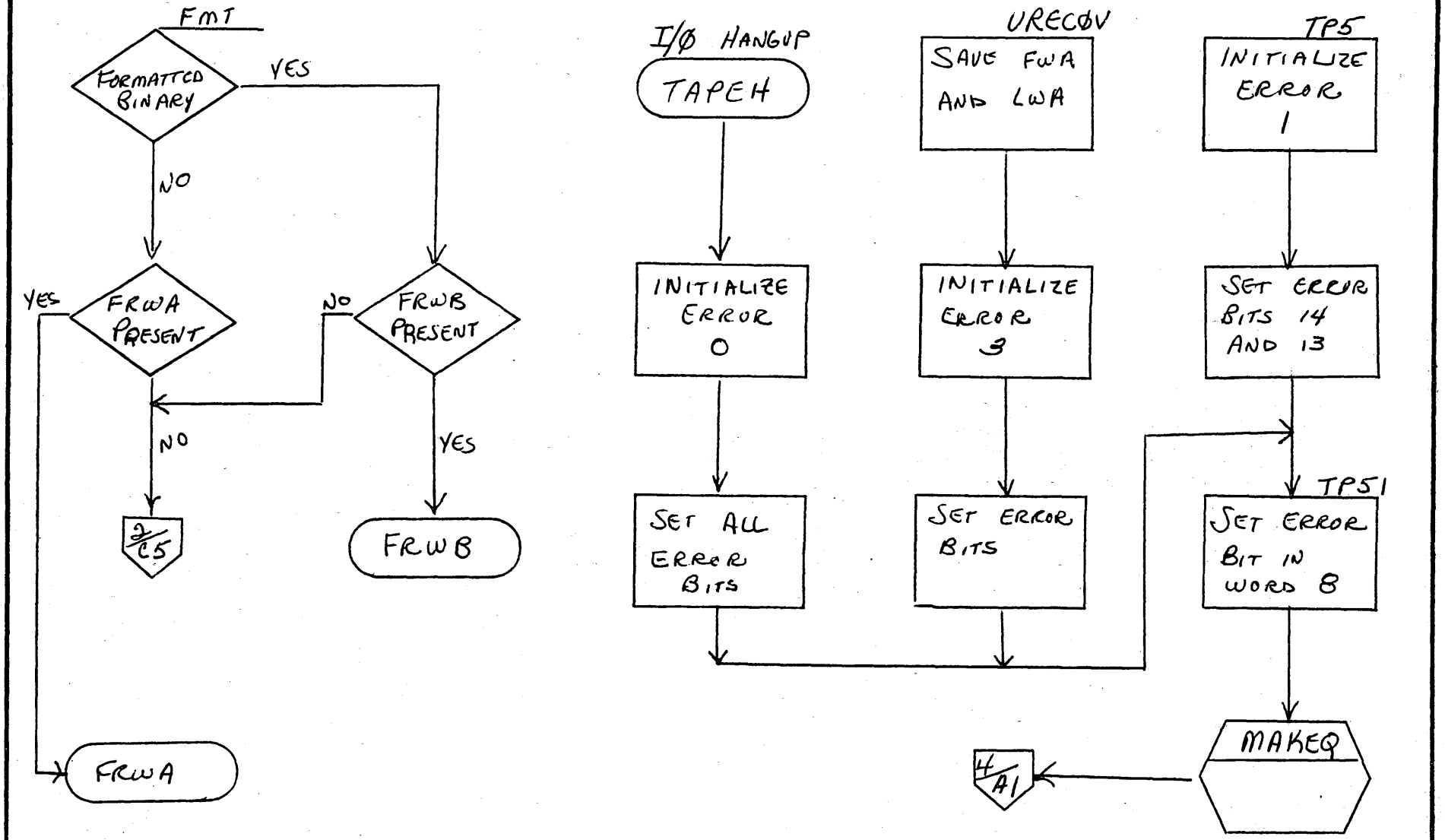


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPEDR	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER UNBUFFERED	PAGE 2 OF 29	PROJECT MGR.			
	NUMBER	ISSUE DATE	PROJECT NAME			
	DRAWN BY	DATE	TASK NO.			
			TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

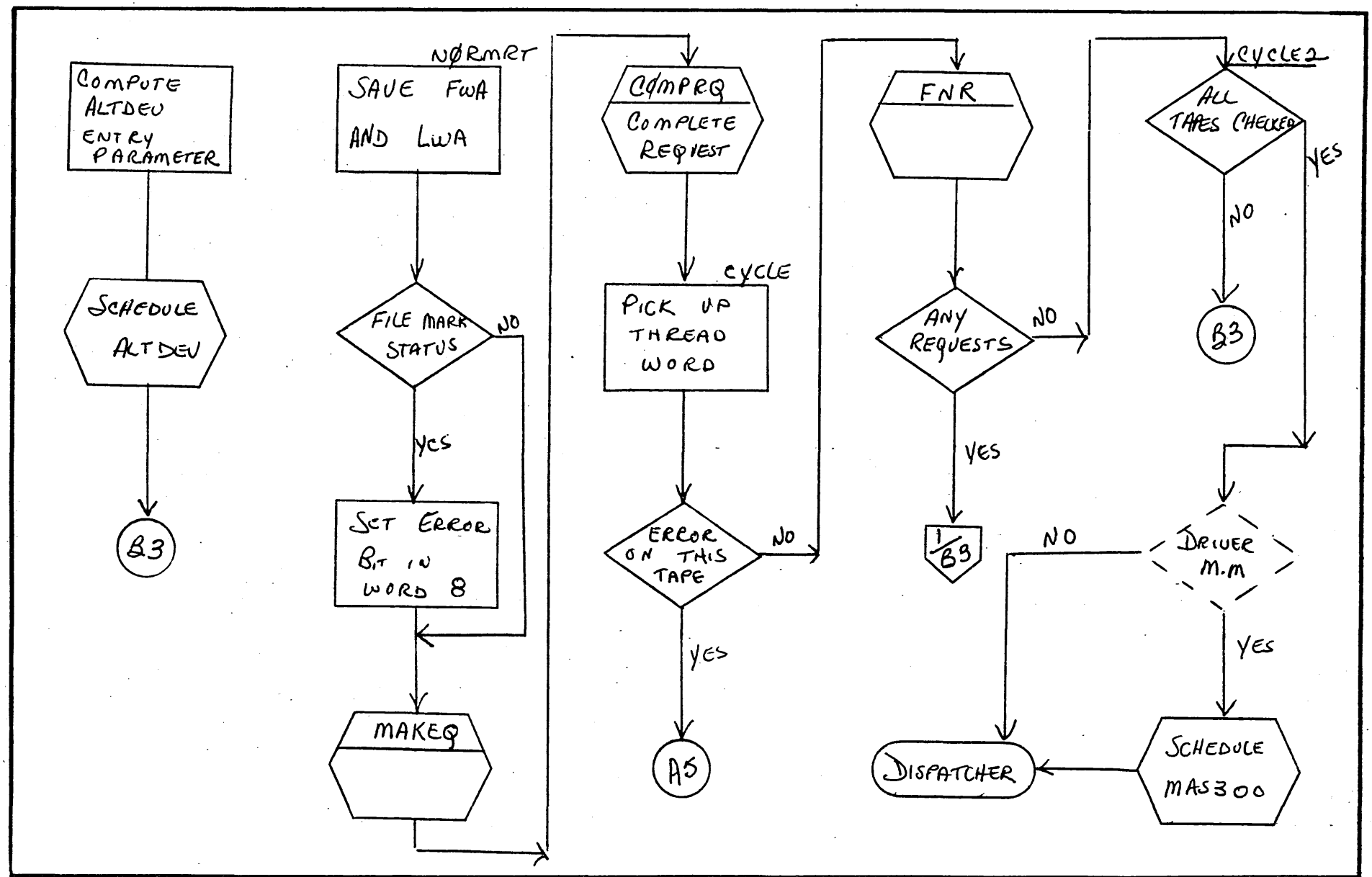
53.15



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LMS	MACH. TYPE 1700	TAPEDR	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.		
	UNBUFFERED	PAGE 3 OF 24	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

53.16

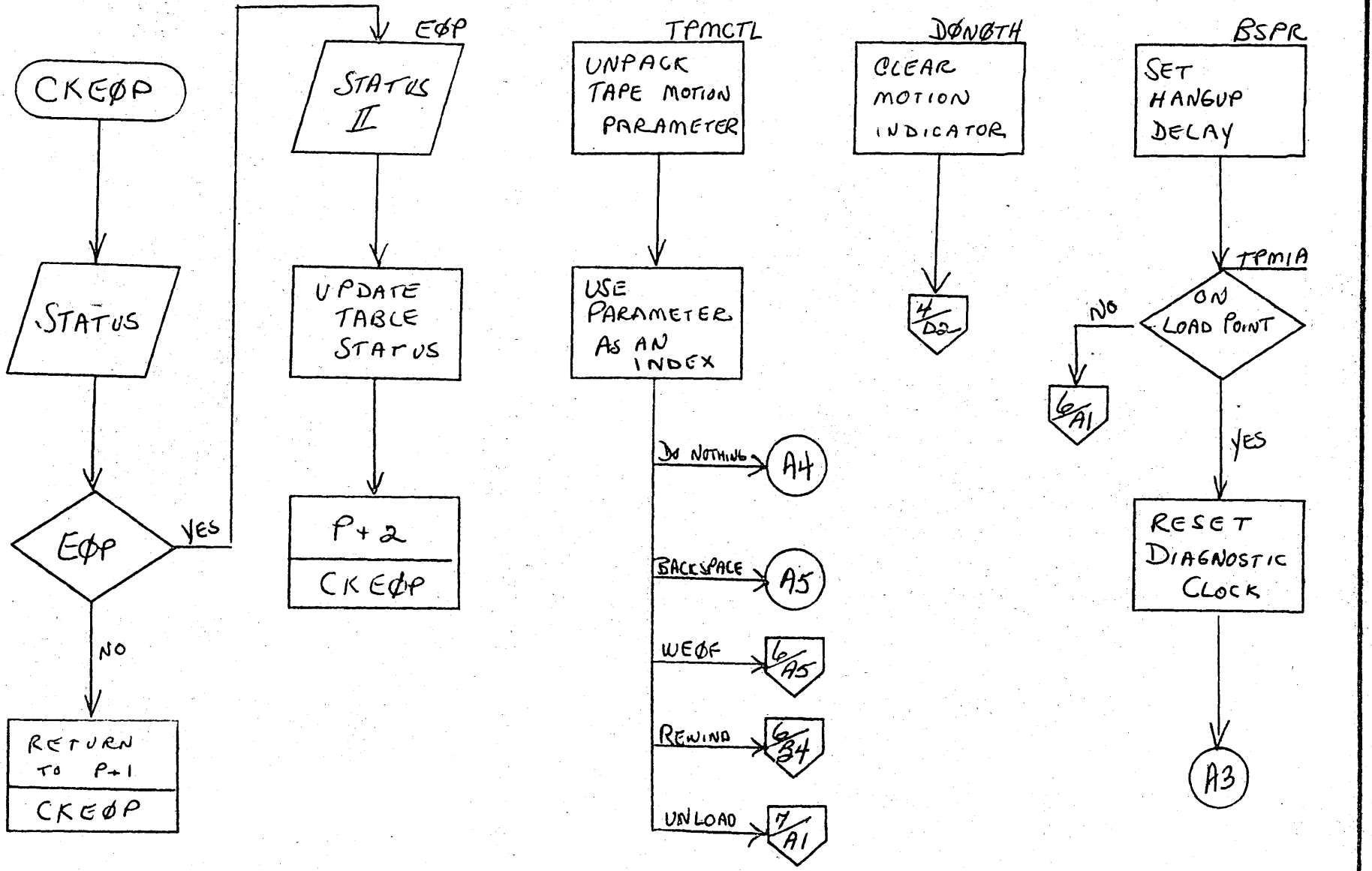


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPEDR	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	UNBUFFERED	PAGE 4 OF 29	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

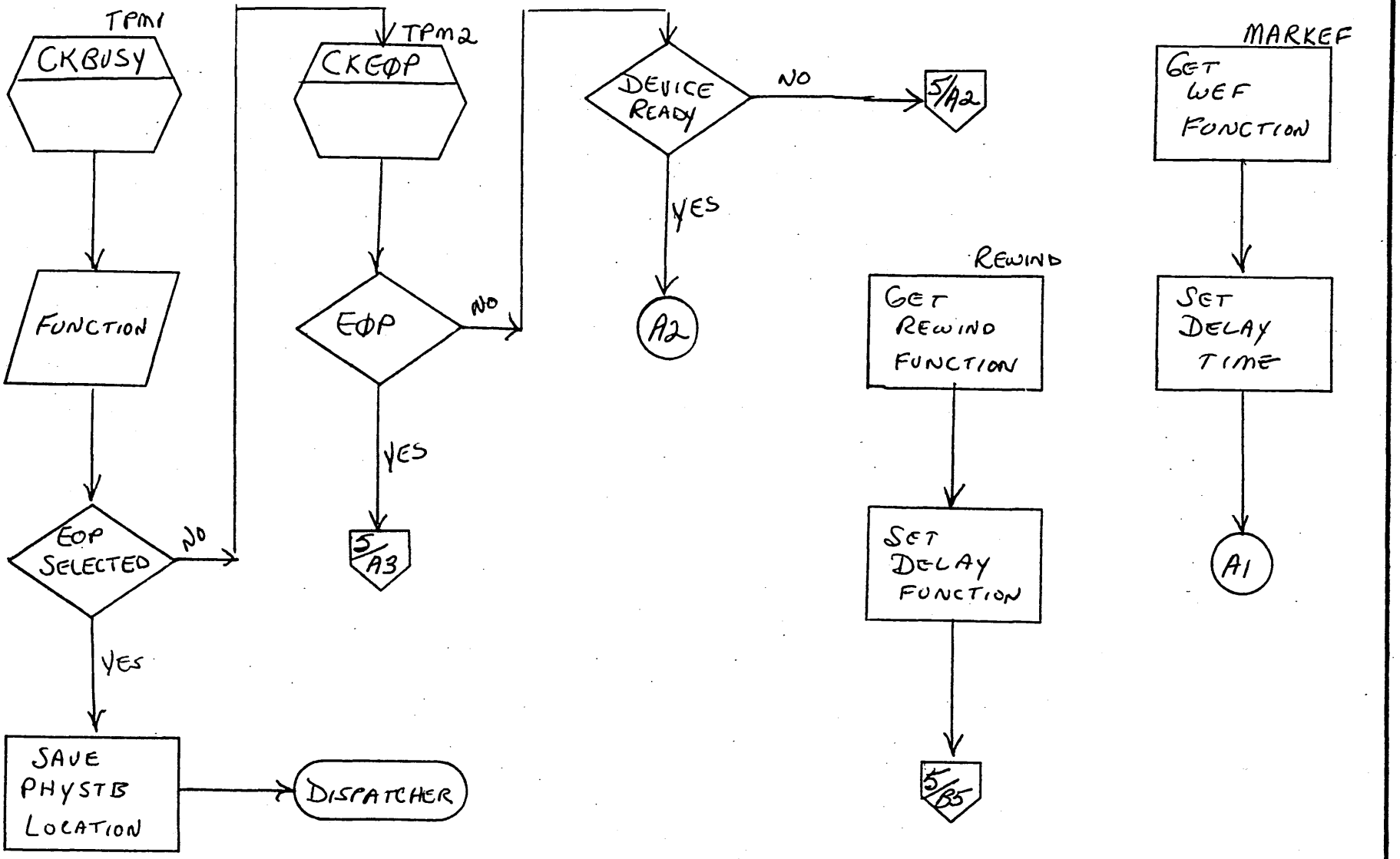
53.17



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPEDR	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	UNBUFFERED	PAGE 5 OF 29	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.18

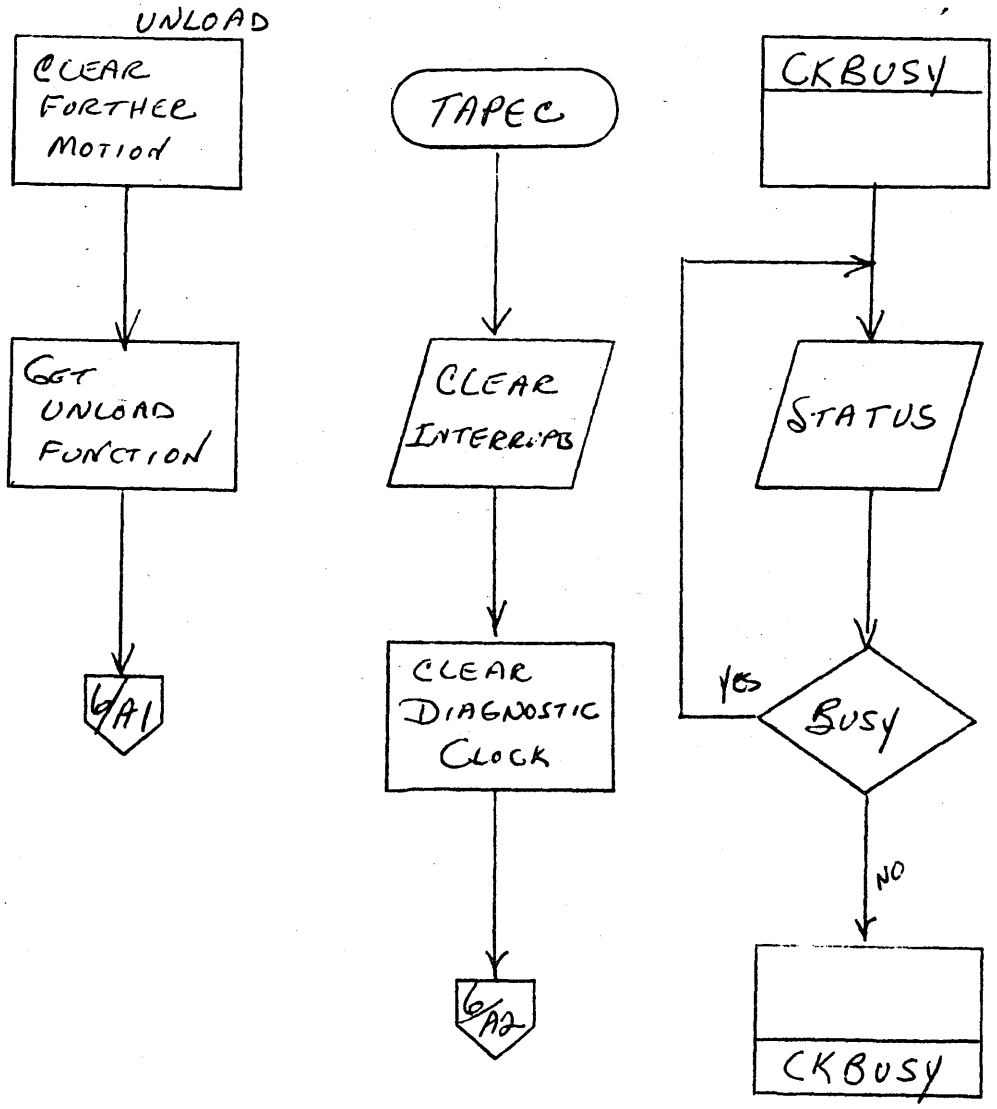


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	: TAPEDR	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	UNBUFFERED	PAGE 6 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.19

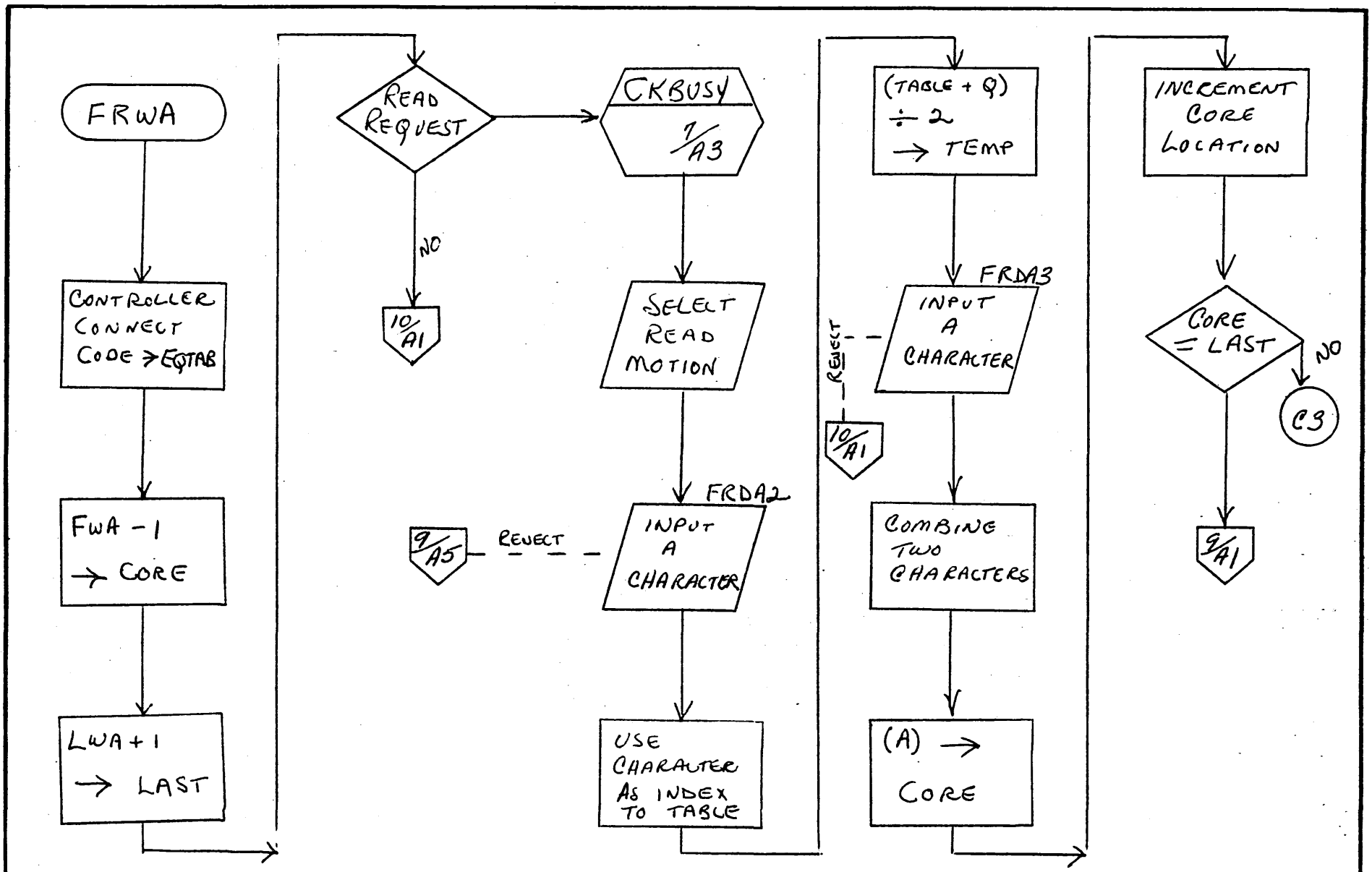


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPEDR	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	UNBUFFERED	PAGE 7 OF 29	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53-20

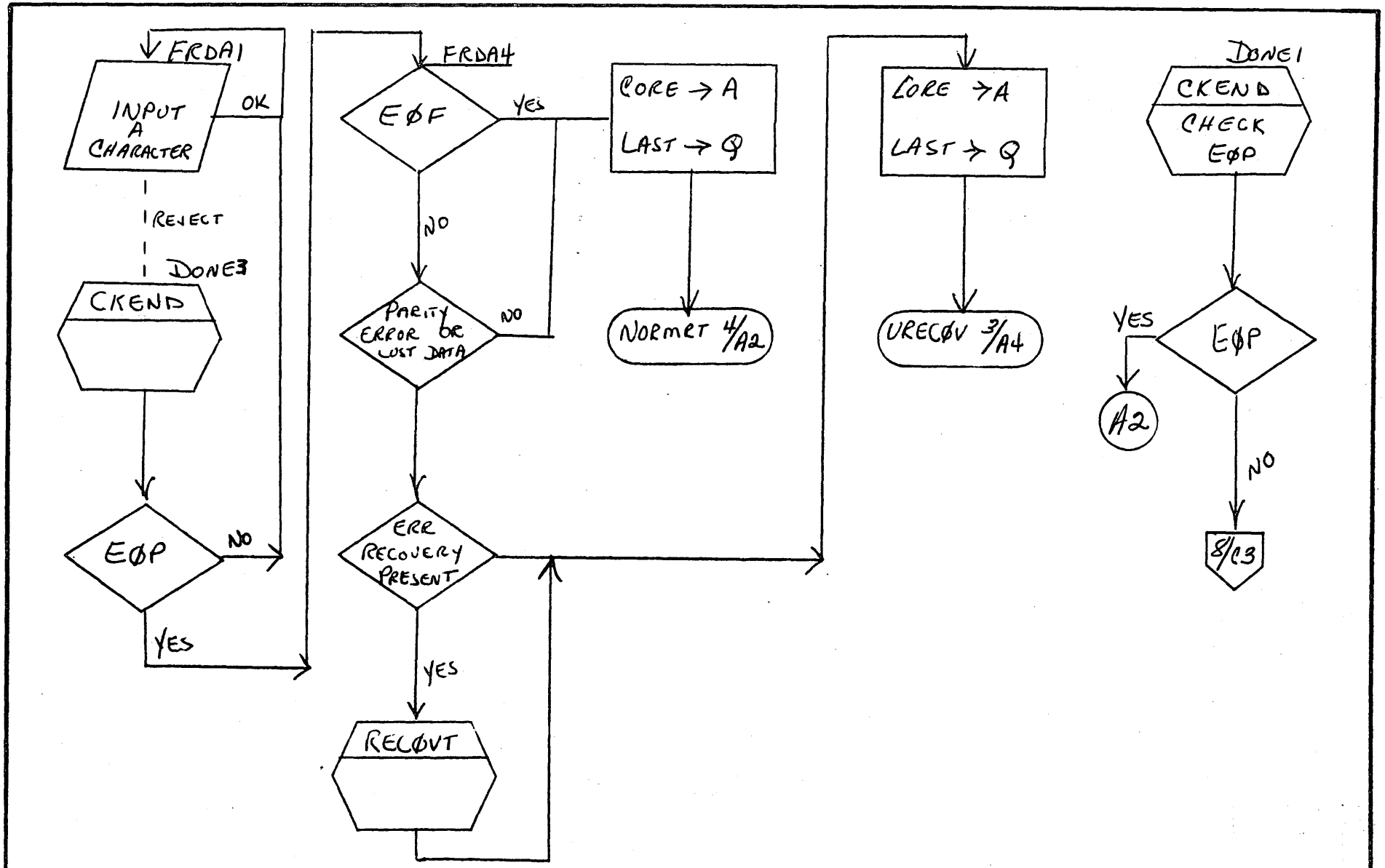


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWA	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR			
FLOWCHART ☐	UNBUFFERED	PAGE 8 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.21

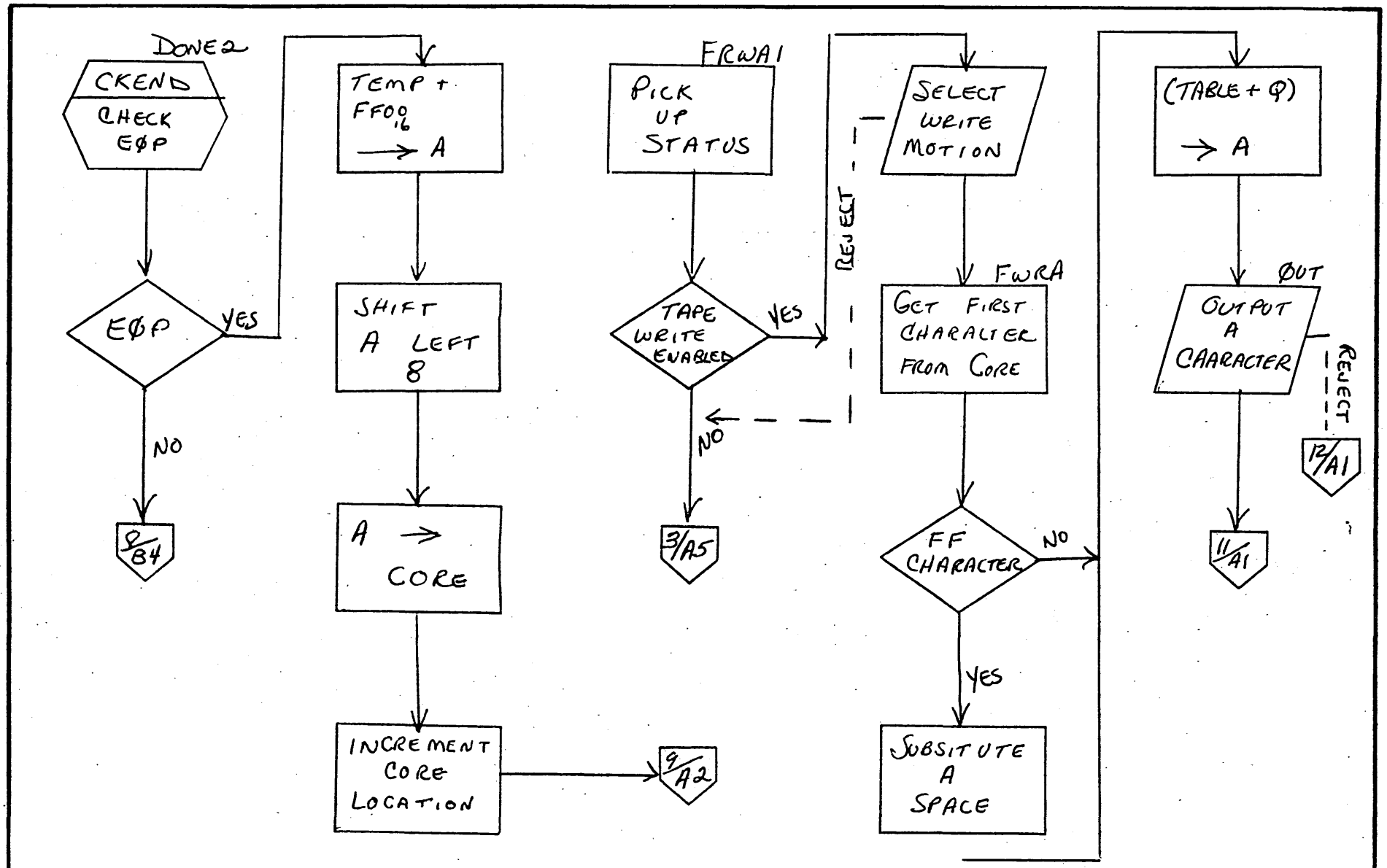


CONTROL DATA CORPORATION	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWA	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
SAMPLE CODE ☐	UNBUFFERED	PAGE 9 OF 29	PROJECT NAME			
FLOWCHART ☐	NUMBER	ISSUE DATE	TASK NO.			
DECISION TABLE ☐	DRAWN BY	DATE	TASK NAME			
OTHER ☐						

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.22



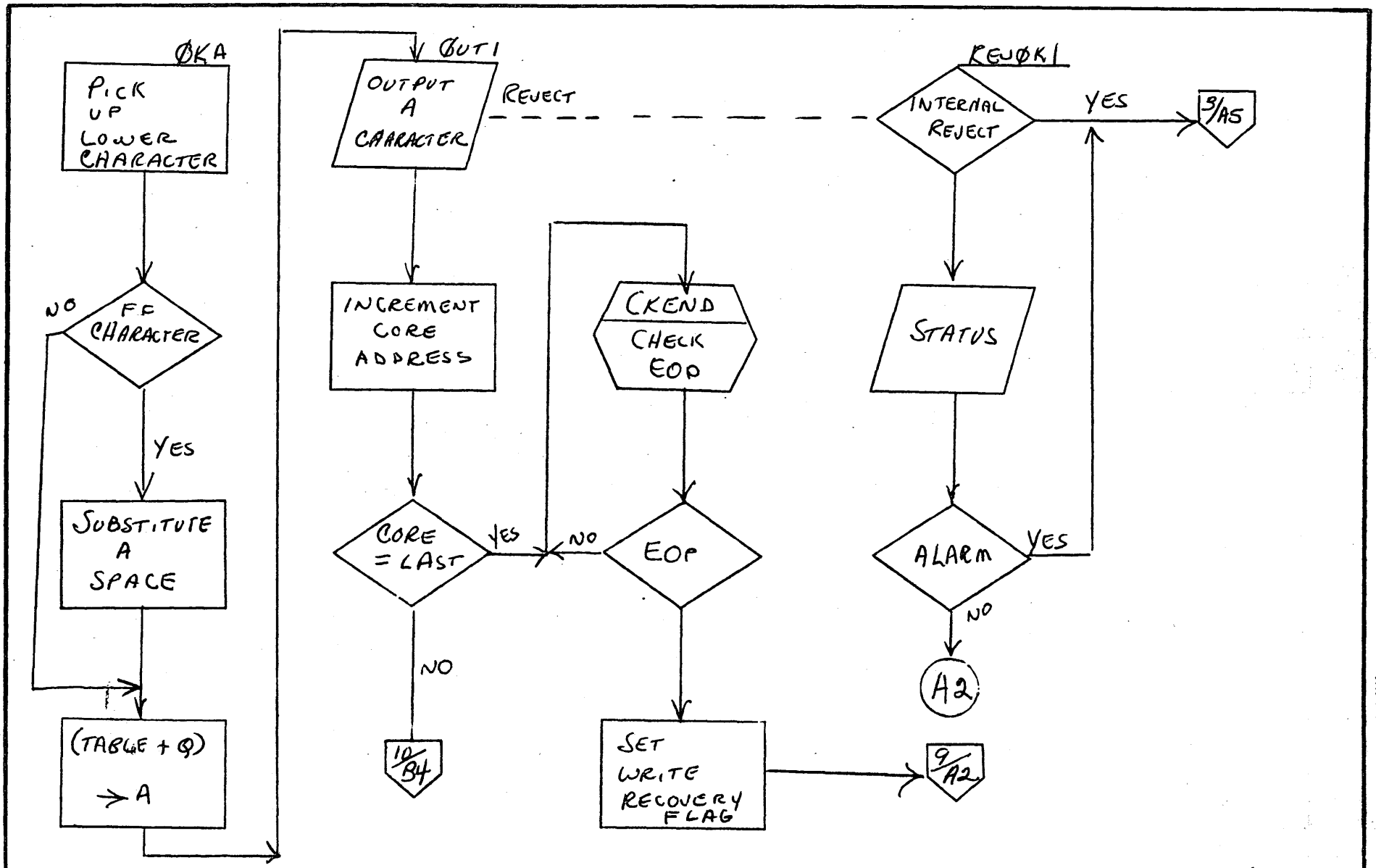
CONTROL DATA CORPORATION			
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWA
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.
FLOWCHART ☐	UNBUFFERED	PAGE 10 OF 29	PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.
OTHER ☐	DRAWN BY	DATE	TASK NAME

REV	APPROVED	DATE

MAR 5 1971

53.23

AA1385 (FORMERLY CA127-1)

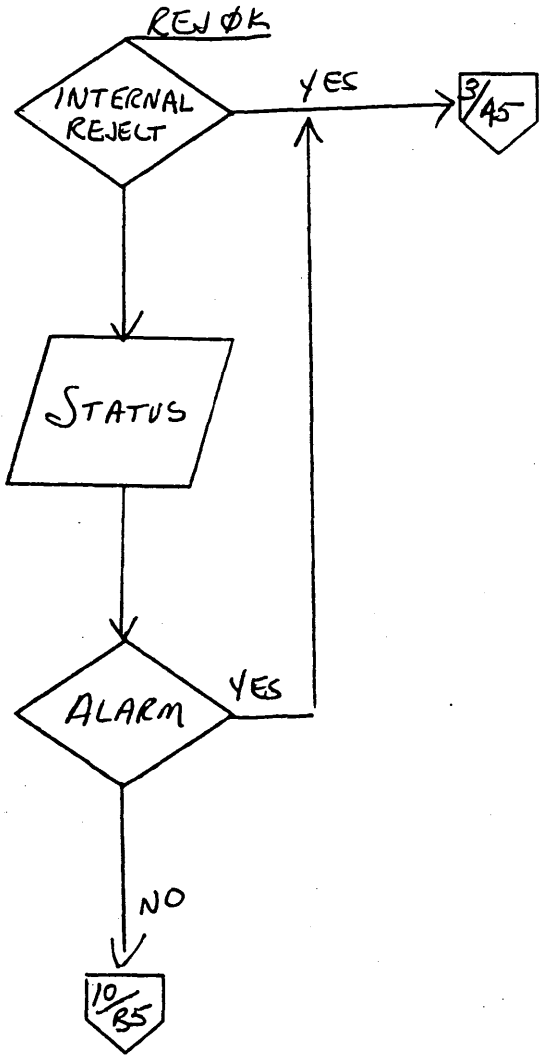


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWA	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.		
	UNBUFFERED	PAGE 11 OF 29	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.24

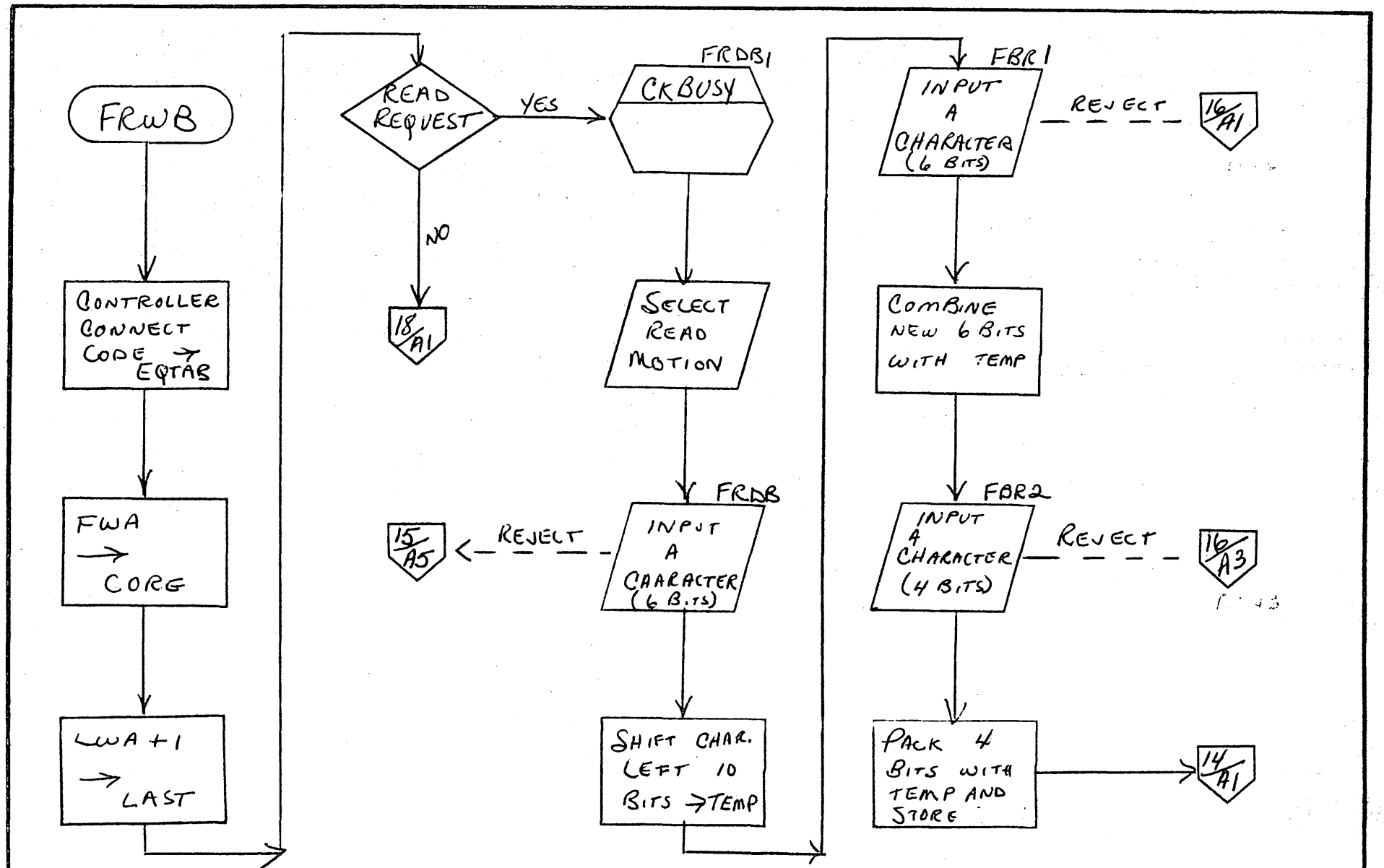


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	FRWA	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.		
	UNBUFFERED	PAGE 12 OF 29	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

53.25

AA1385 (FORMERLY CA127-1

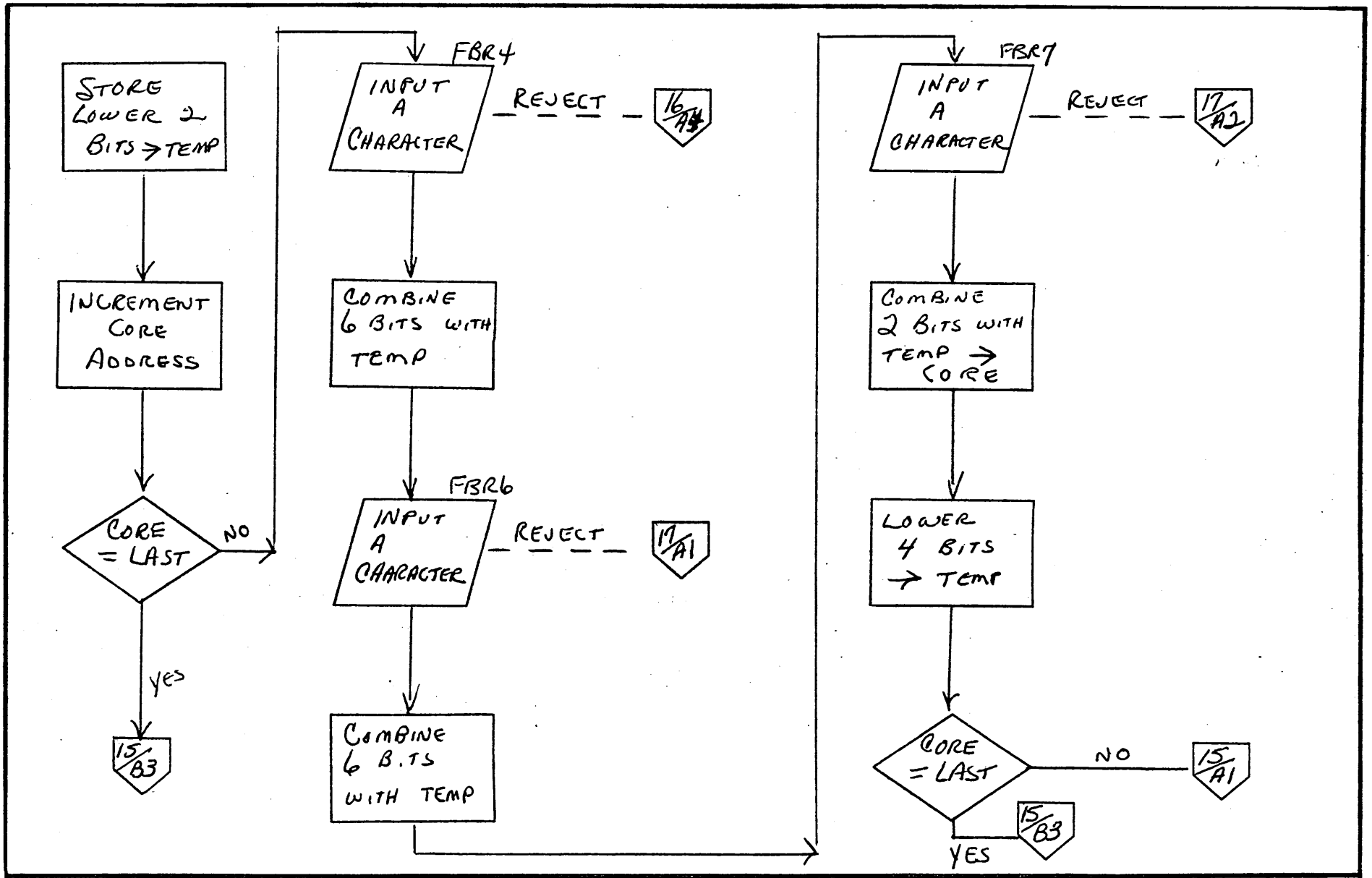


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWB	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.		
	UNBUFFERED	PAGE 13 OF 29	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.26

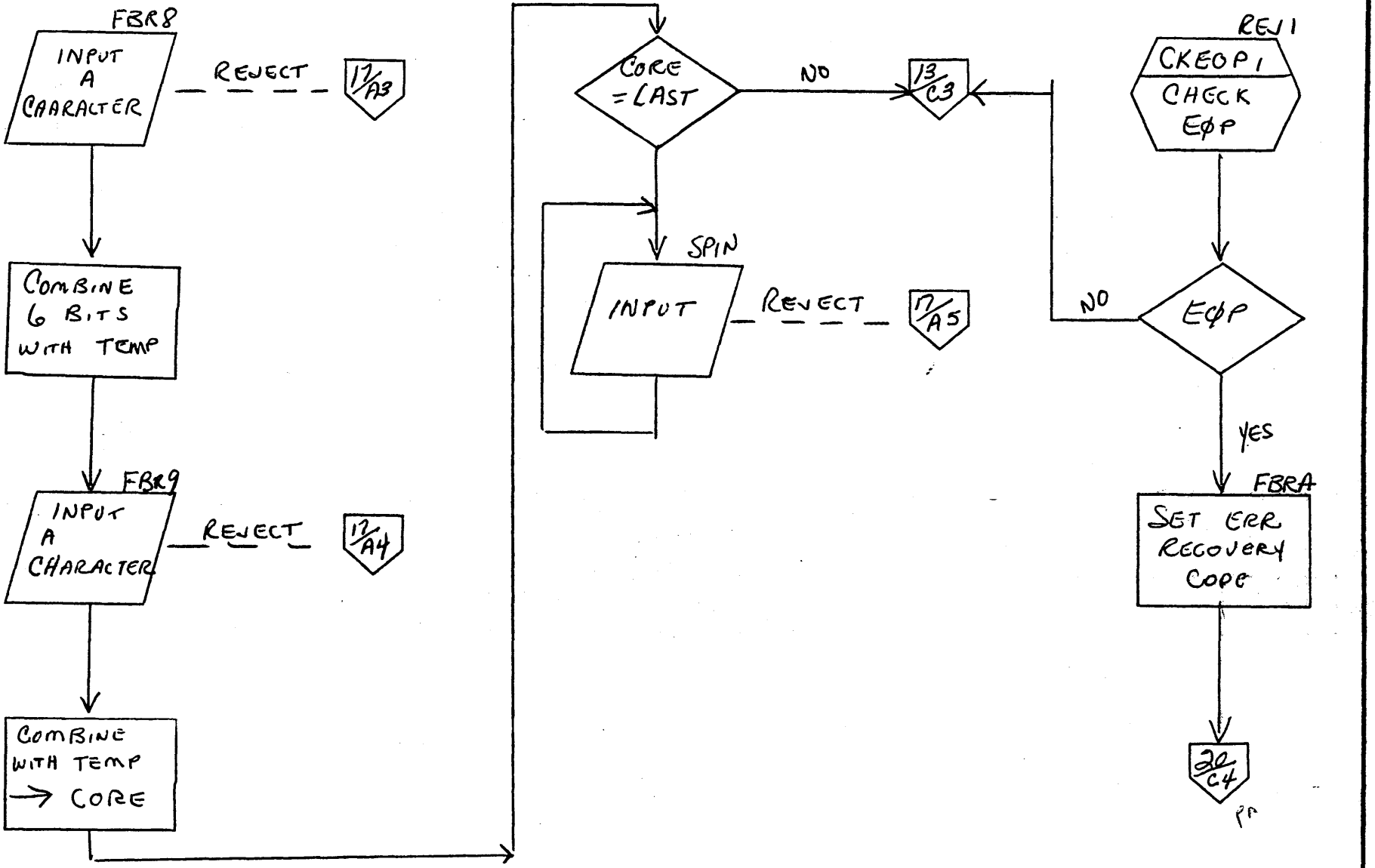


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	UNBUFFERED	PAGE 14 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53-27

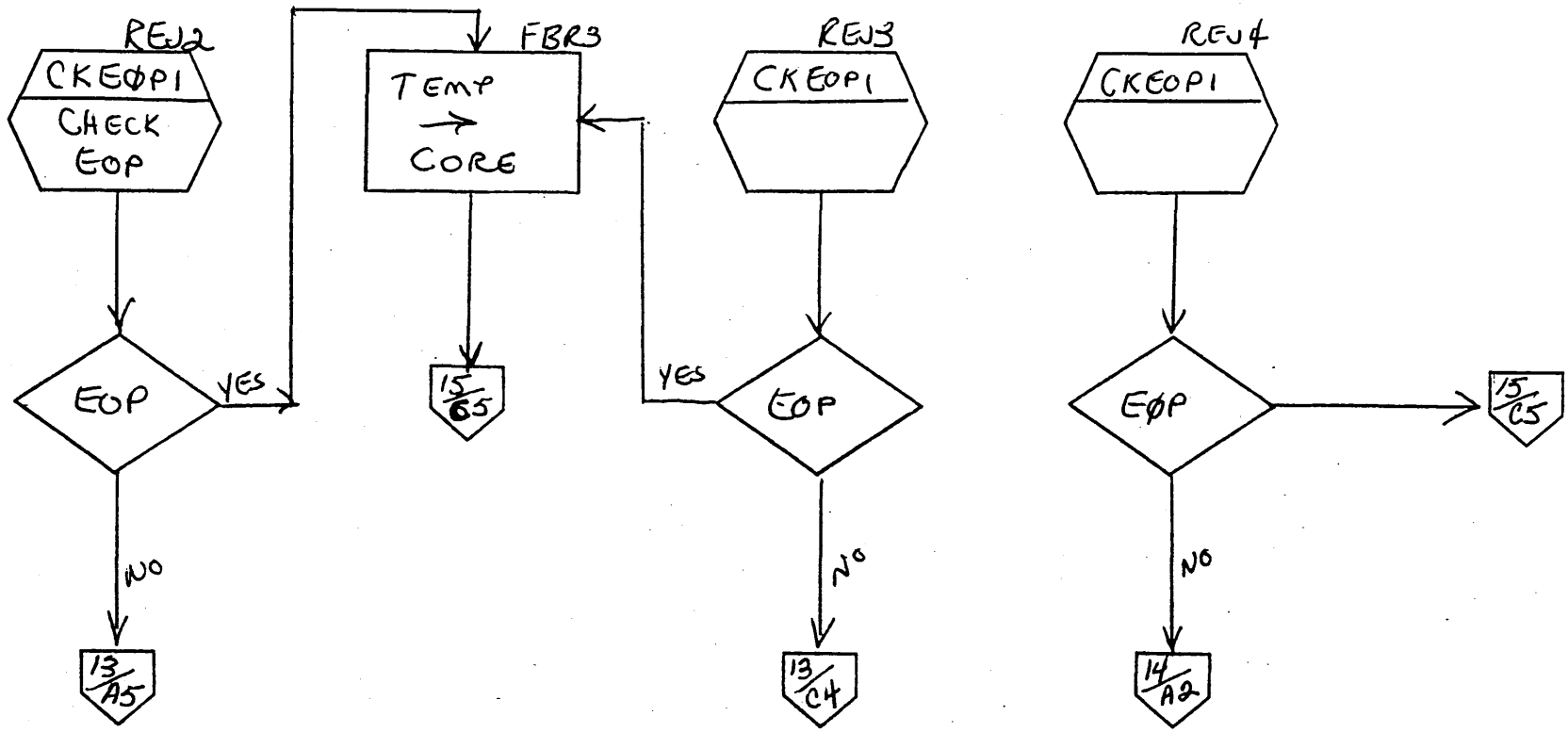


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	FRWB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	PAGE 15 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

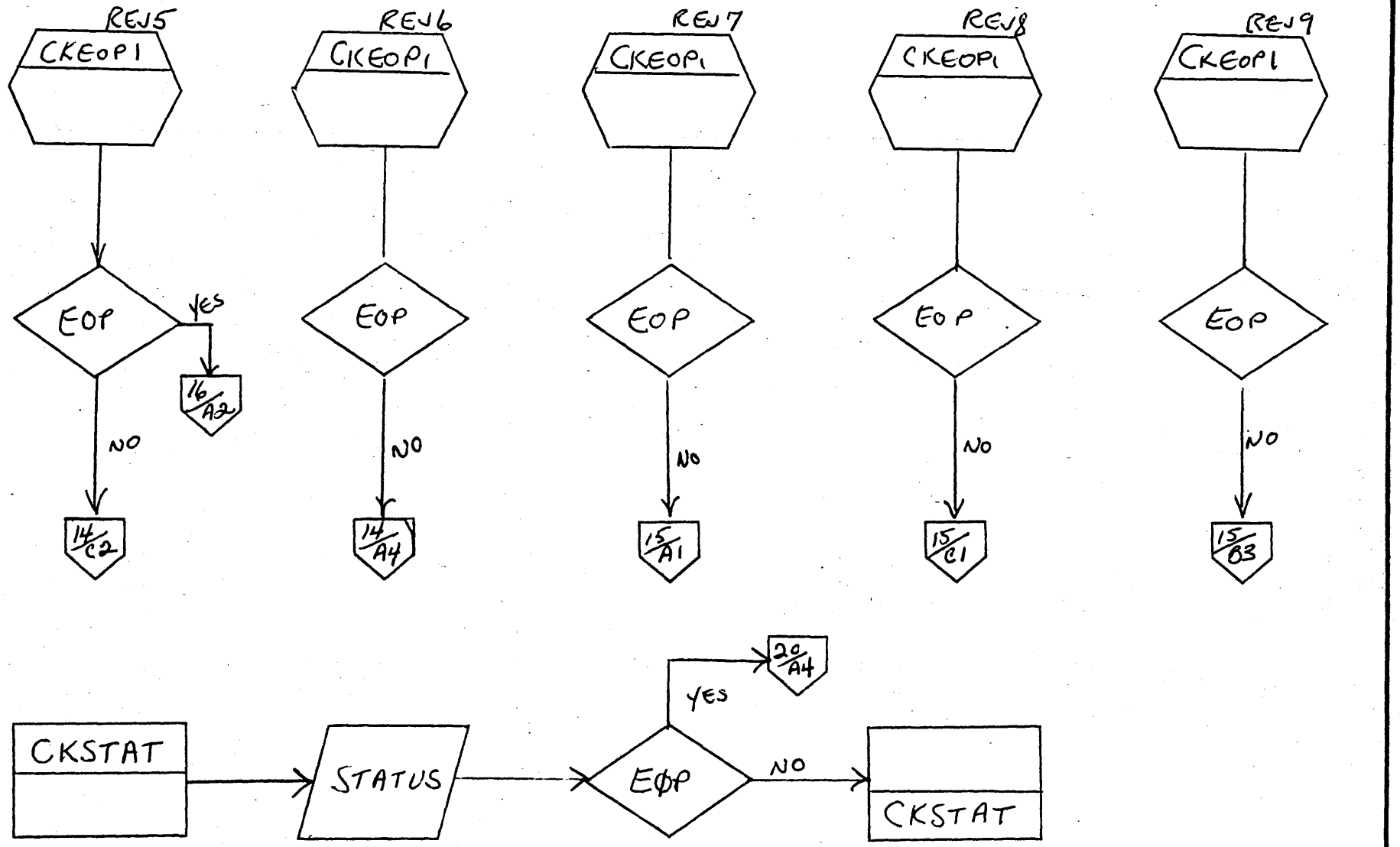
53-28

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWB	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.		
	UNBUFFERED	PAGE 16 OF 29	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

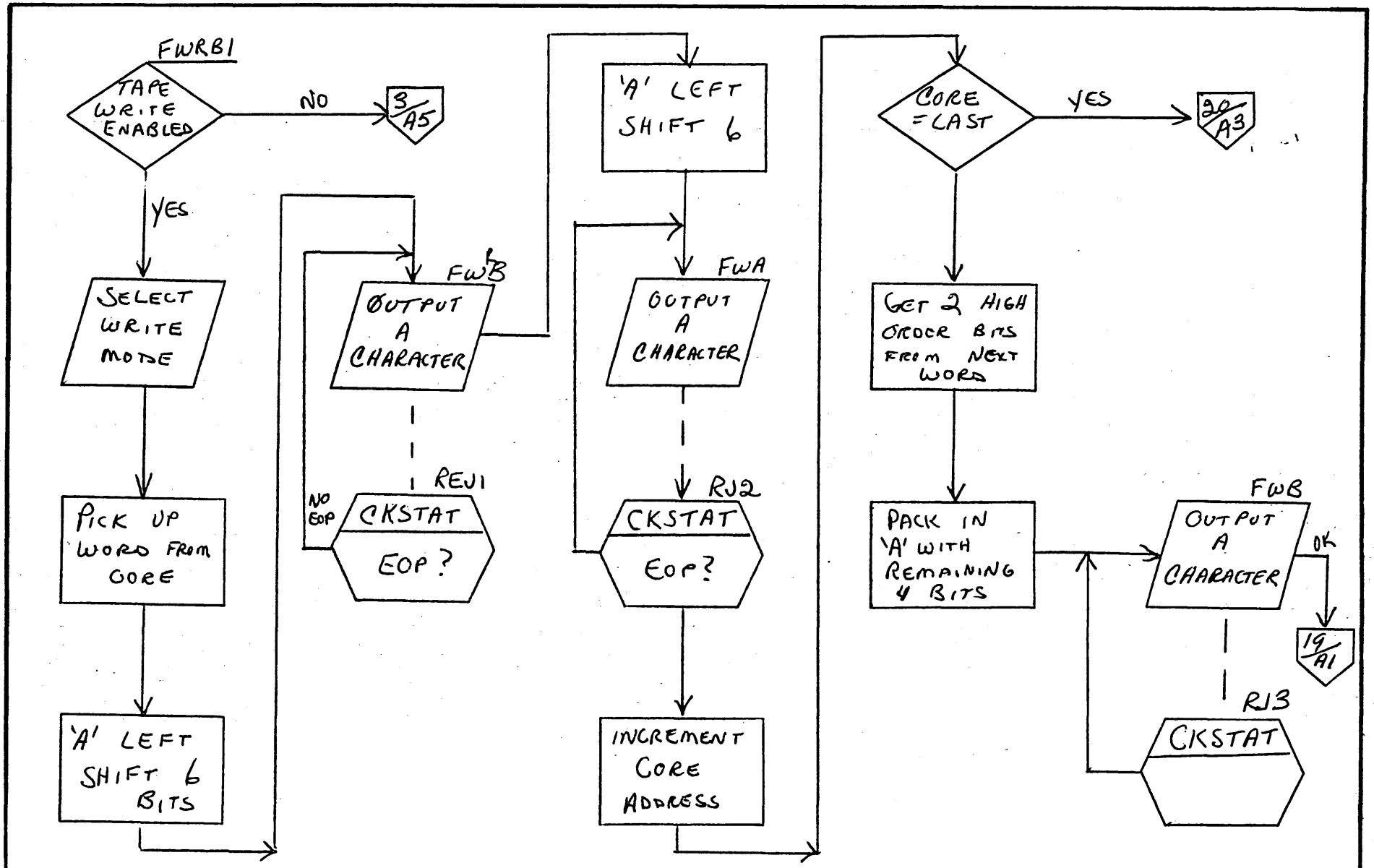


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	FRWB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	UNBUFFERED	PAGE 17 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53-30

AA1385 (FORMERLY CA127-1)

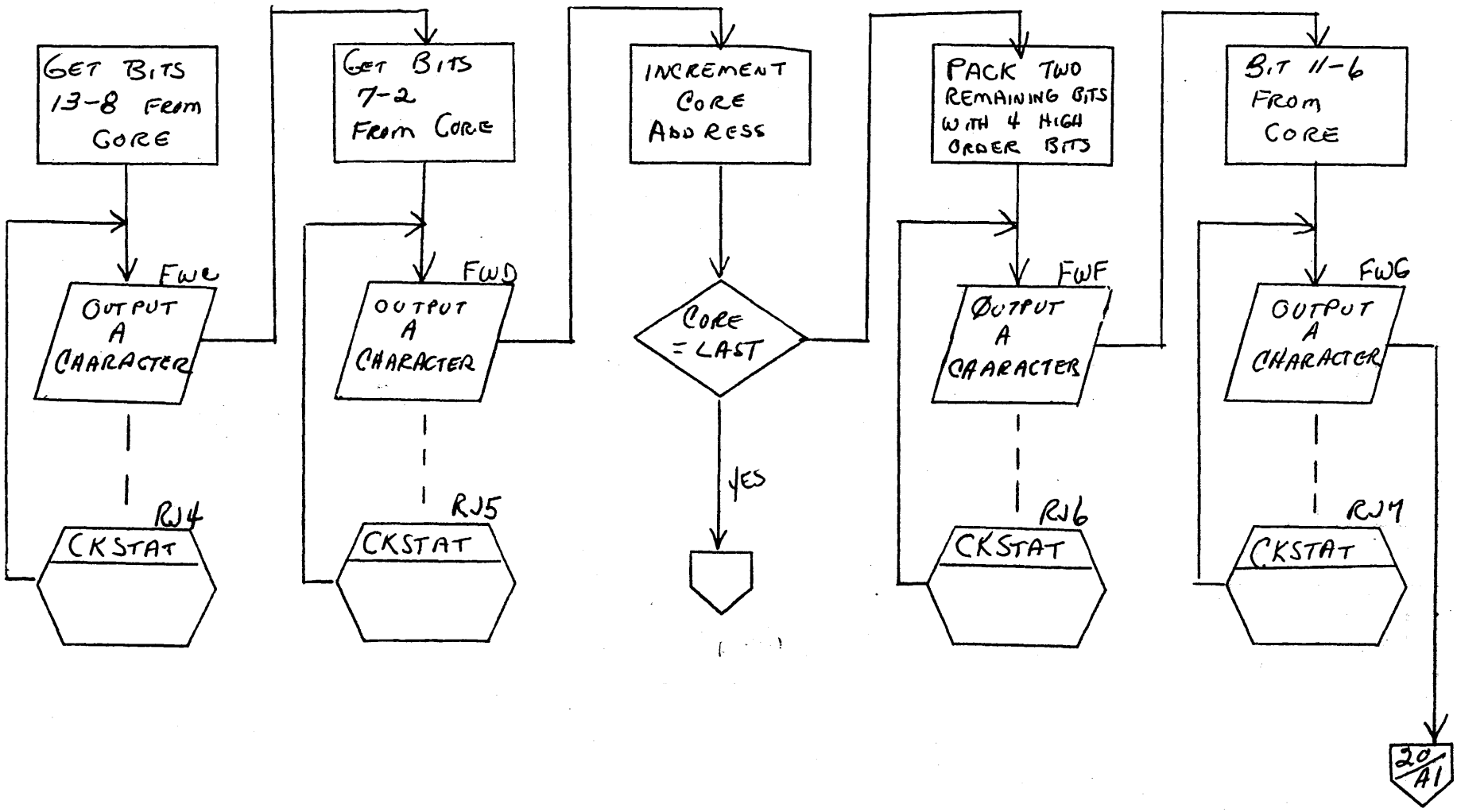


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	FRWB	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601	DRIVER	PROJECT MGR.			
	UNBUFFERED	PAGE 18 OF 29	PROJECT NAME			
	NUMBER	ISSUE DATE /	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.31

AA1385 (FORMERLY CA127-1)

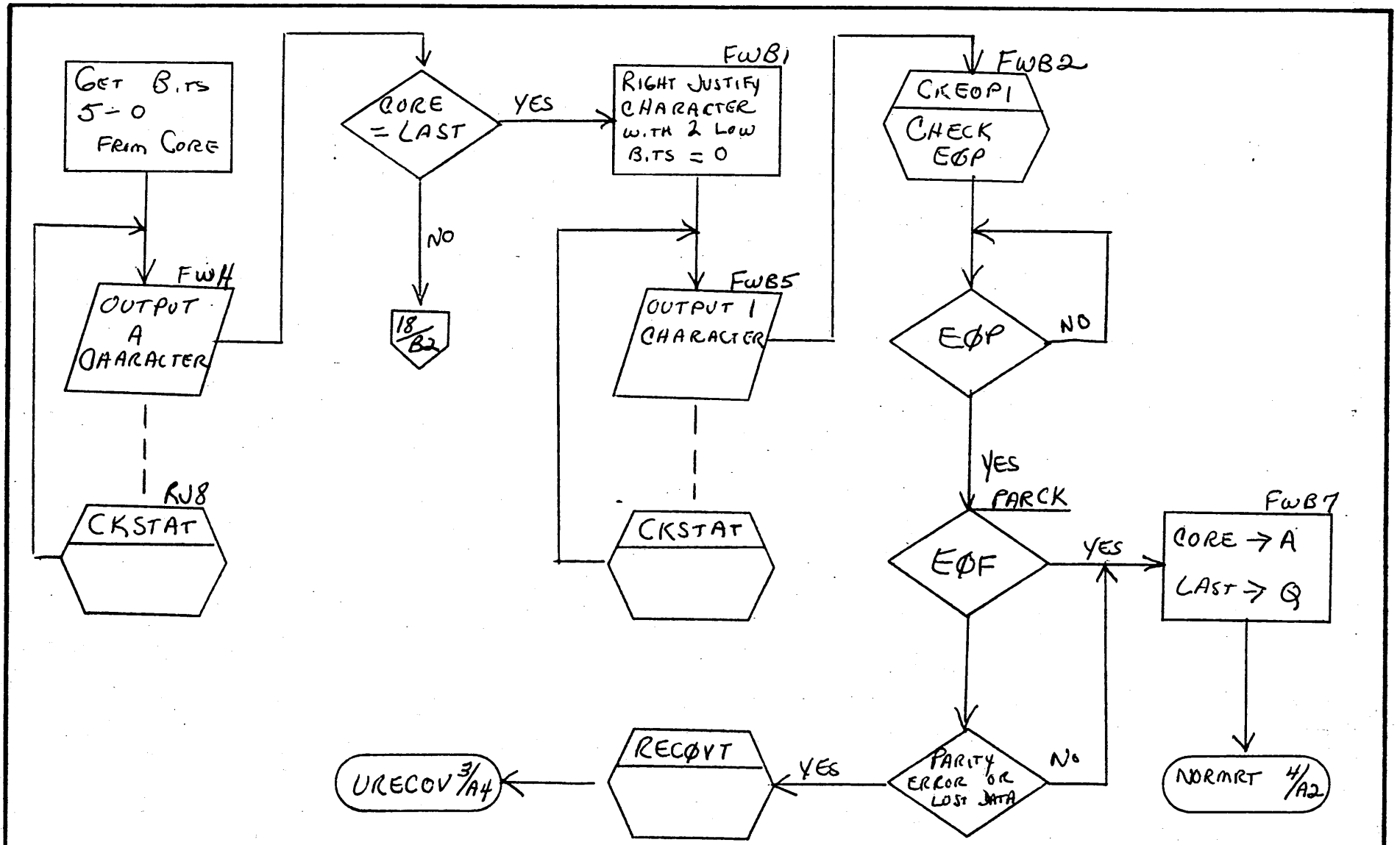


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWB	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	UNBUFFERED	PAGE 19 OF 29	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.32

AA1385 (FORMERLY CA127-1)



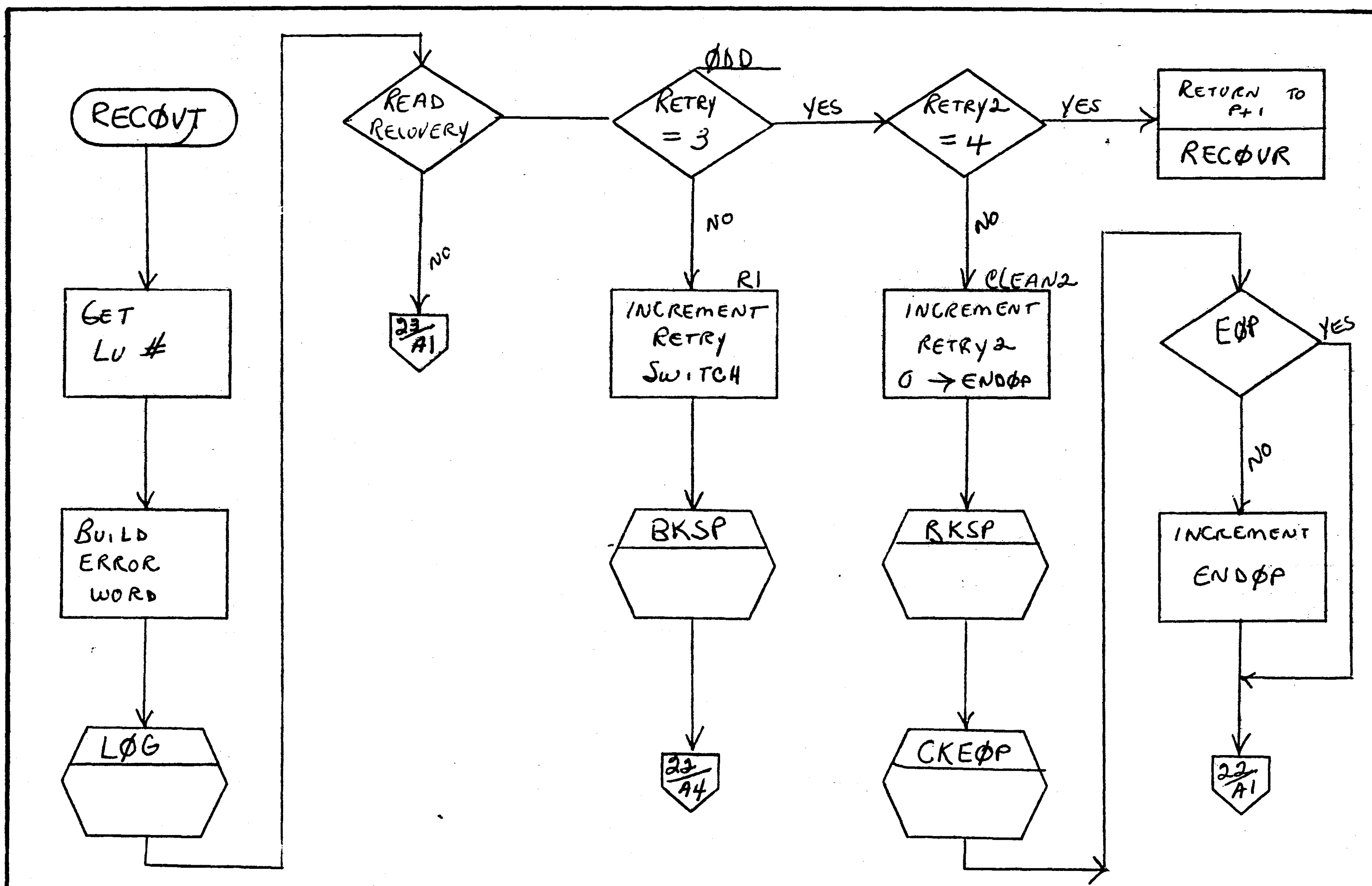
CONTROL DATA CORPORATION			
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWB
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.
FLOWCHART ☐	UNBUFFERED	PAGE 20 OF 29	PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.
OTHER ☐	DRAWN BY	DATE	TASK NAME

REV	APPROVED	DATE

MAR 5 1971

53.33

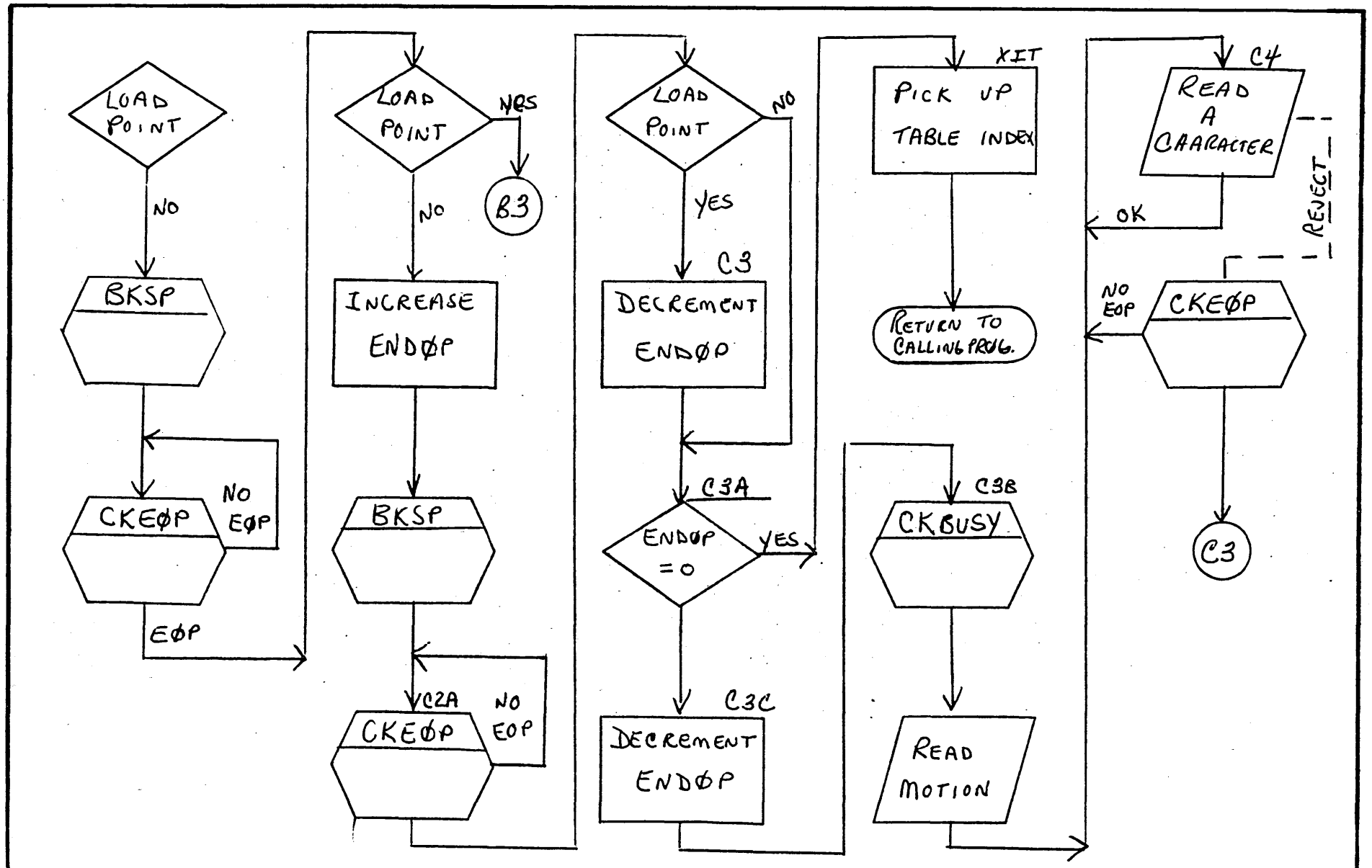
AA1385 (FORMERLY CA127-1)



MAR 5 1971

CONTROL DATA CORPORATION SOFTWARE DOCUMENT			REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	RECØVT			
FLOWCHART ☐	DOCUMENT TITLE 1731/601 DRIVER	PROJECT MGR.			
DECISION TABLE ☐	UNBUFFERED PAGE 21 OF 29	PROJECT NAME			
OTHER ☐	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

53-34

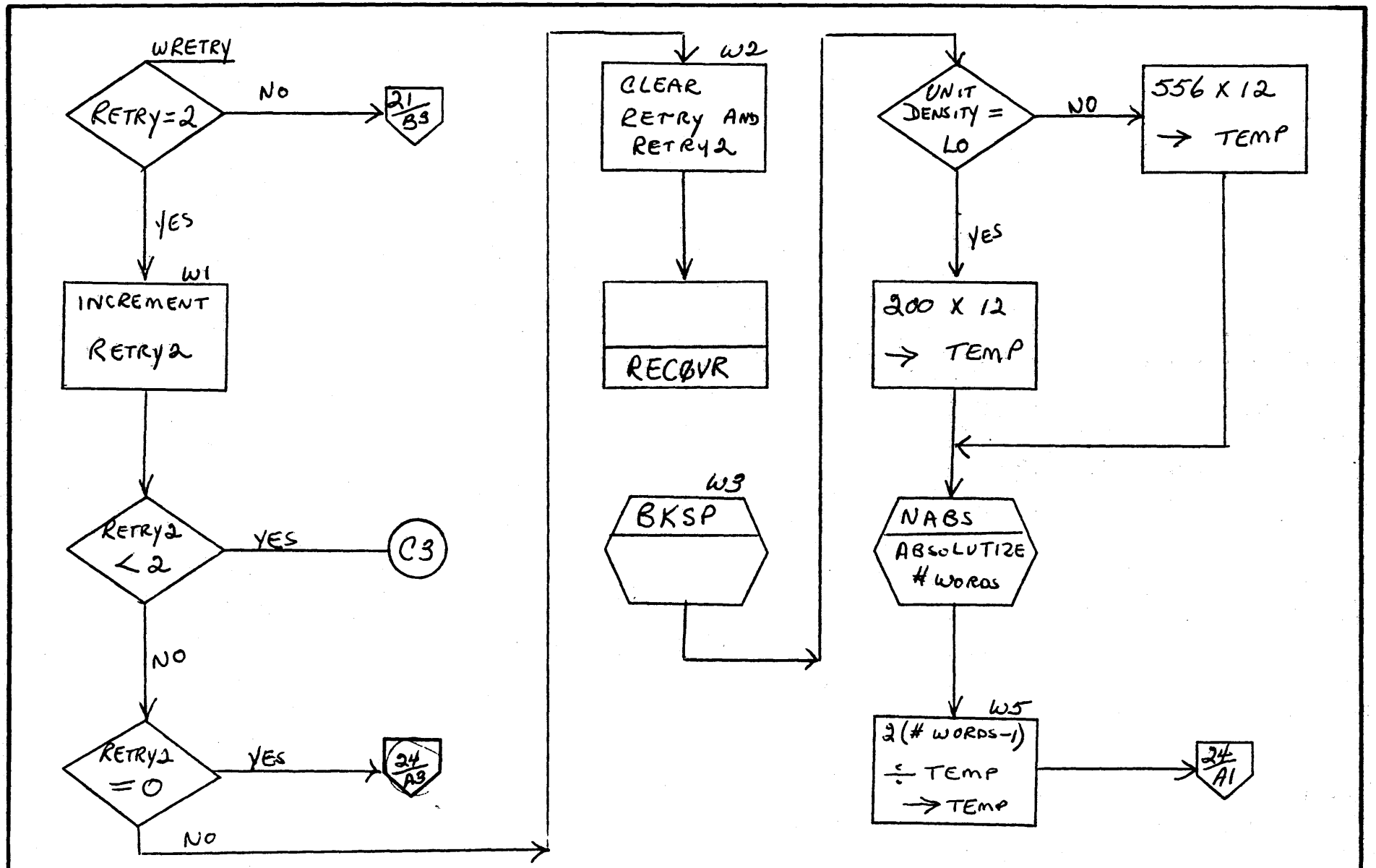


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	RECØVT	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	UNBUFFERED	PAGE 22 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.35

AA1385 (FORMERLY CA127-1

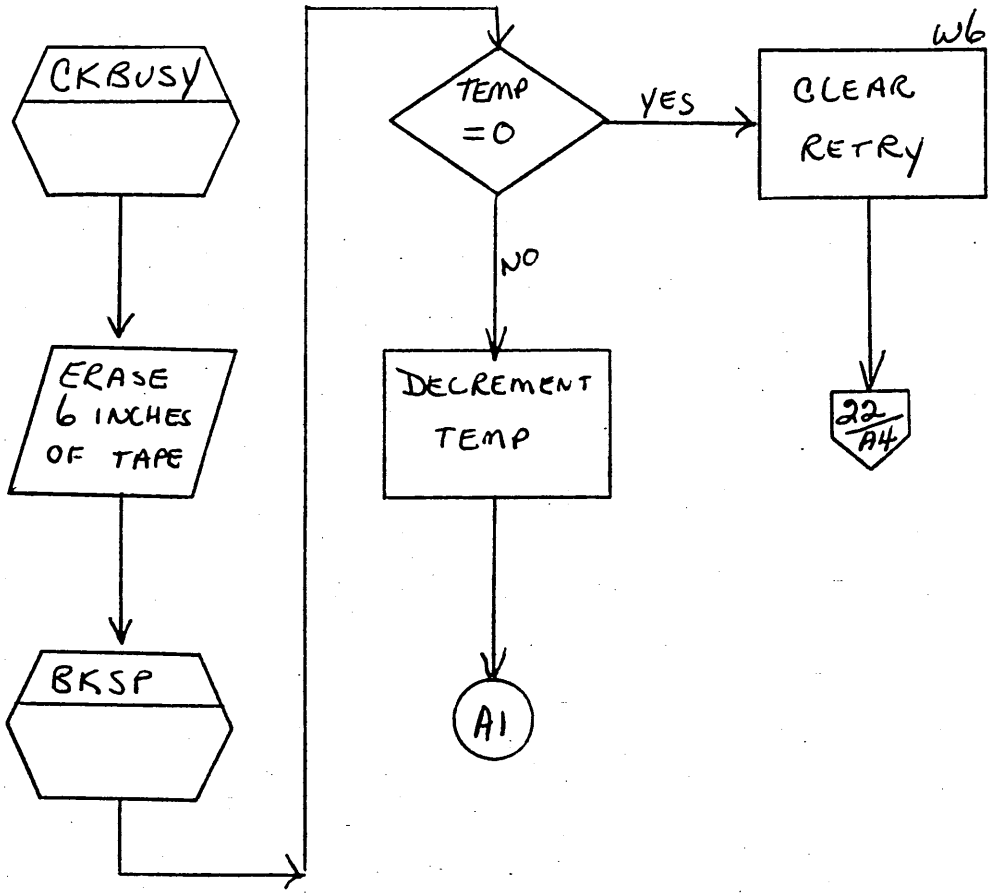


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	RECØVT	REV / APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER UNBUFFERED PAGE 23 OF 29	PROJECT MGR.	
	NUMBER ISSUE DATE	PROJECT NAME	
	DRAWN BY DATE	TASK NO.	
		TASK NAME	

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.36

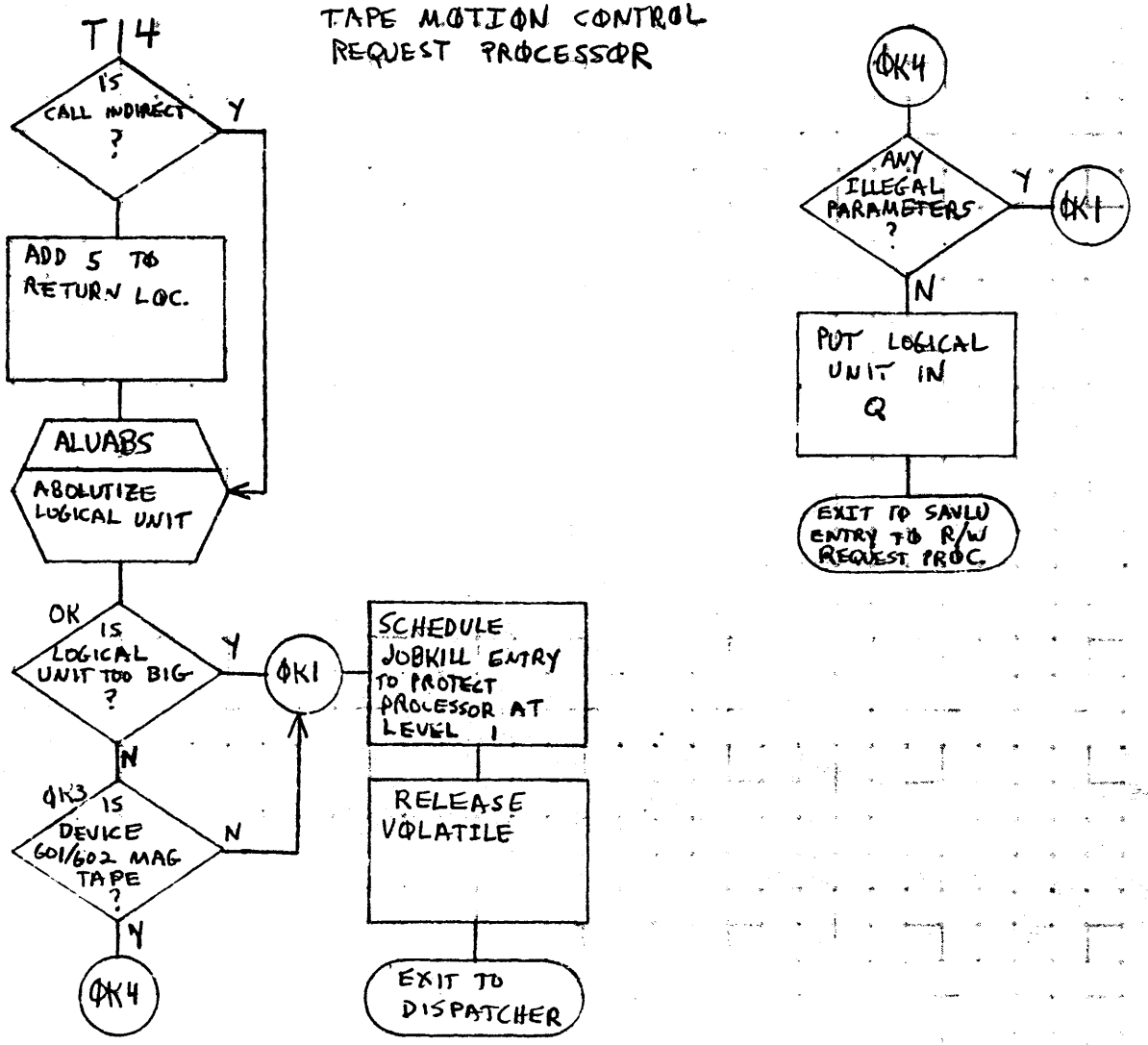


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	RECØVT	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	UNBUFFERED	PAGE 24 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.37

AA1385 (FORMERLY CA127-1)

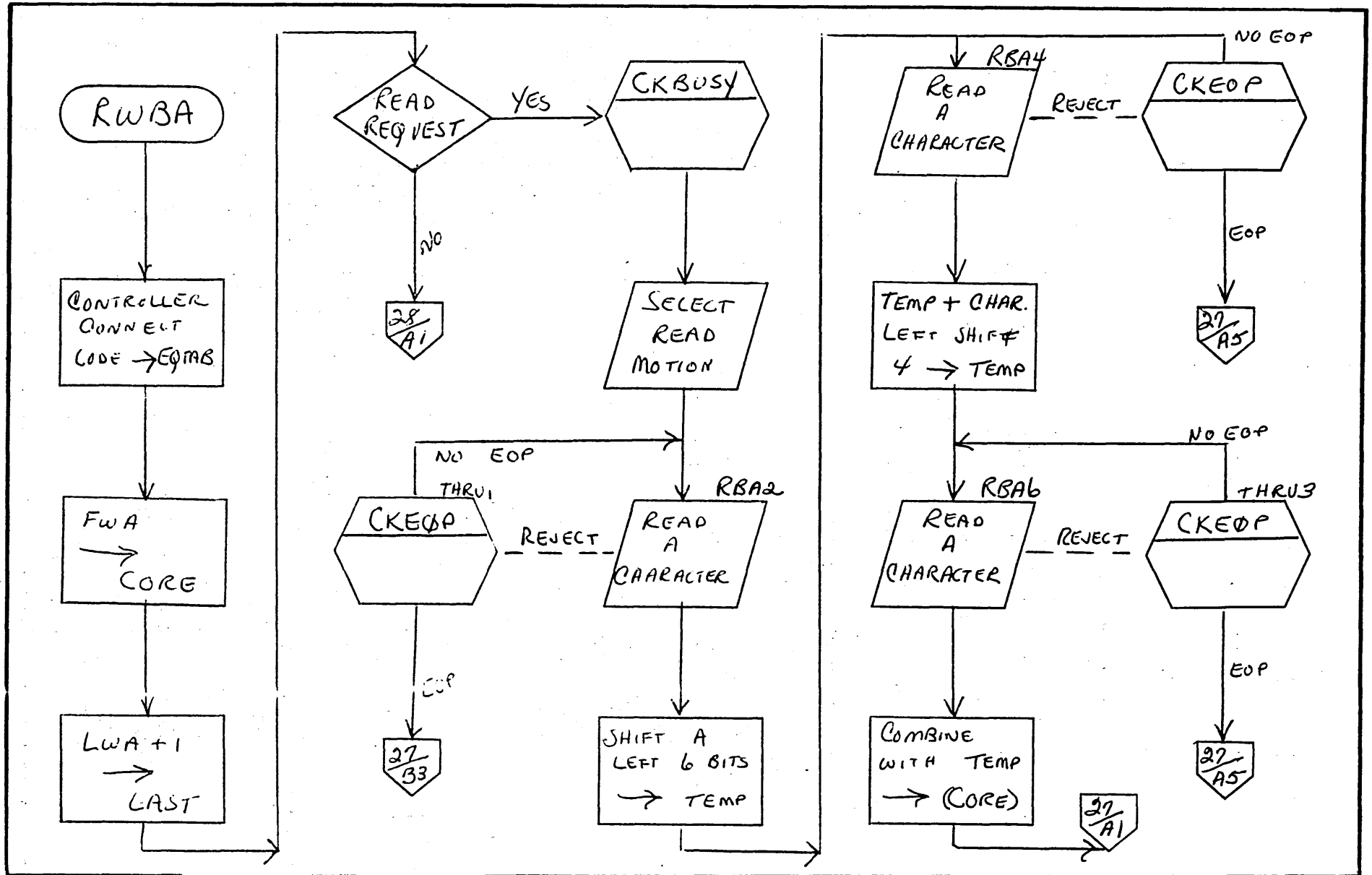


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPE	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER UNBUF AND BUF	PAGE 25 OF 39	PROJECT MGR.			
	NUMBER	ISSUE DATE	PROJECT NAME			
	DRAWN BY	DATE	TASK NO.			
			TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

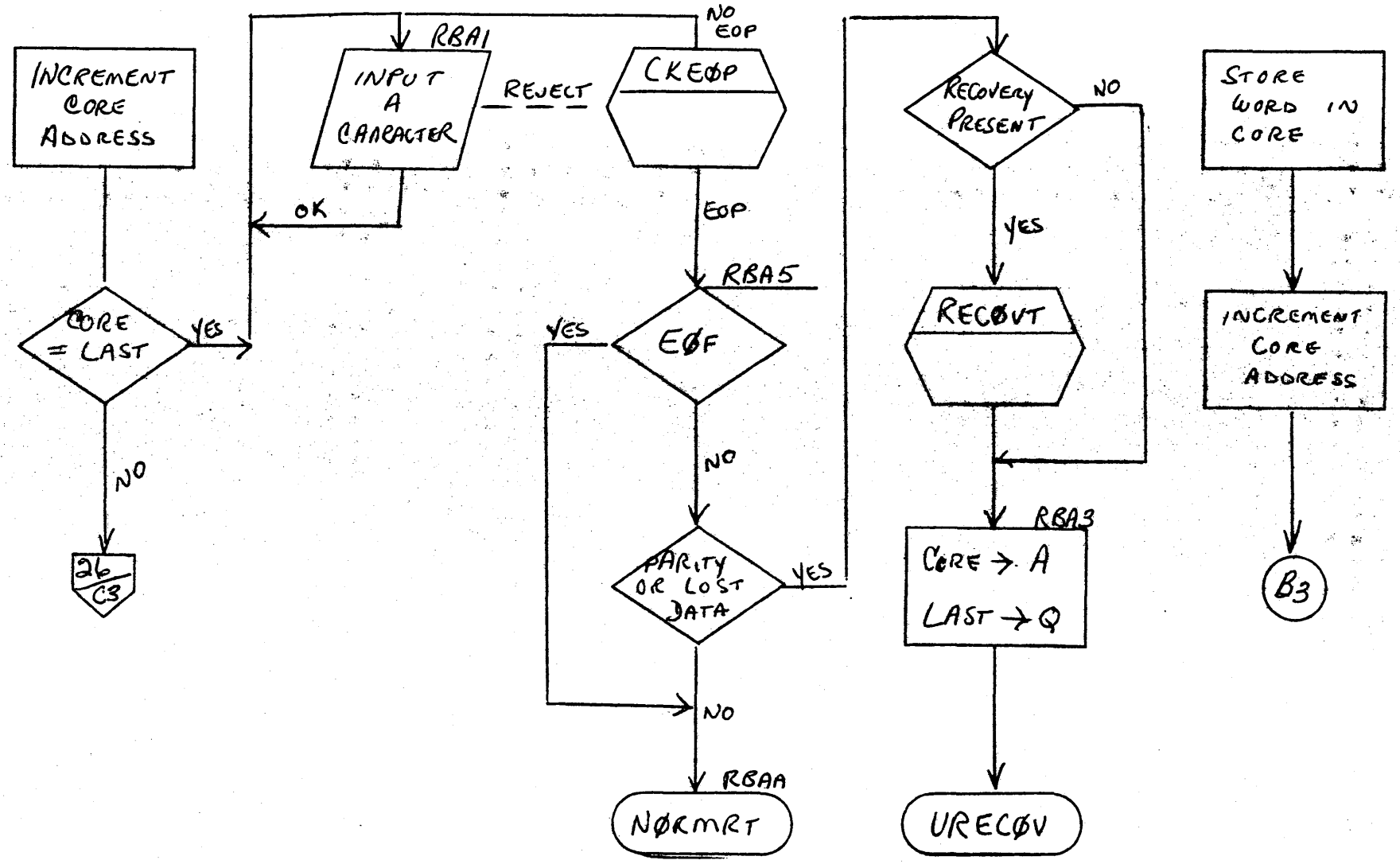
53.38



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	RWBA	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	PAGE 26 OF 29	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

533.39

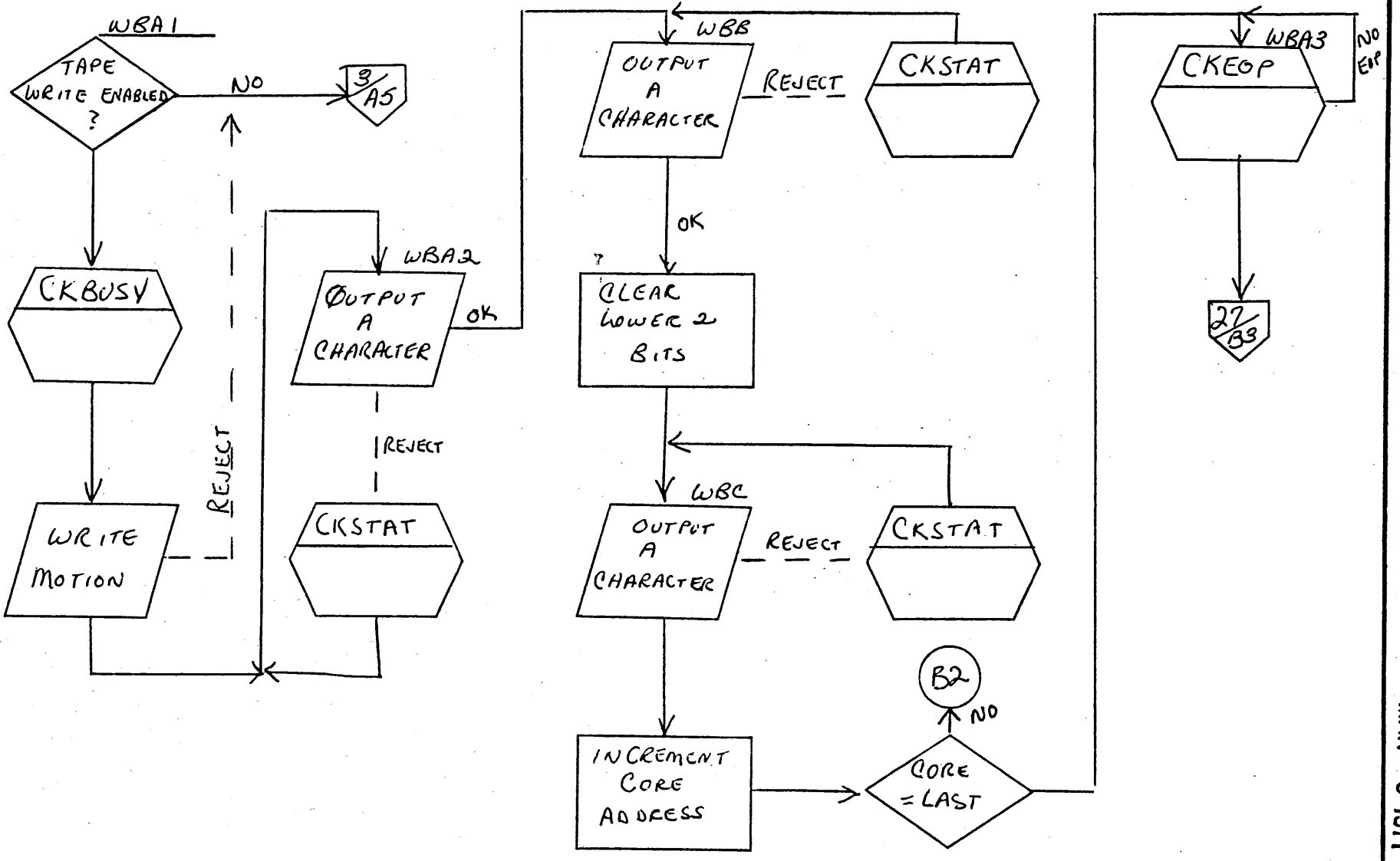


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	RWBA	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	UNBUFFERED	PAGE 27 OF 29	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.40

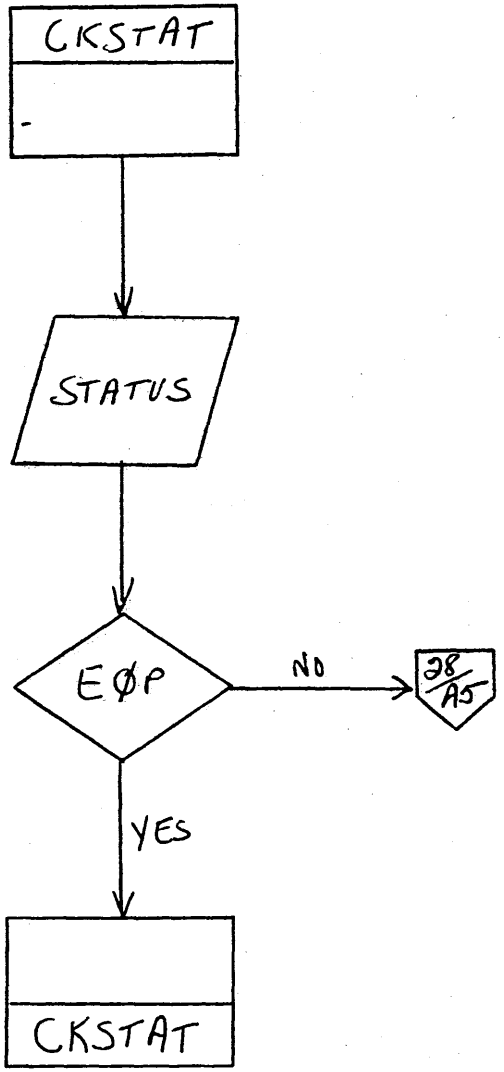


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	RWBA	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.		
	UNBUFFERED	PAGE 28 OF 29	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.41



CONTROL DATA CORPORATION	DOCUMENT CLASS	IMS	MACH. TYPE	1700	RWBA	REV	APPROVED	DATE
SOFTWARE DOCUMENT	DOCUMENT TITLE	1731/601 DELVER			PROJECT MGR.			
SAMPLE CODE ☐	UNBUFFERED		PAGE 29	OF 29	PROJECT NAME			
FLOWCHART ☐	NUMBER		ISSUE DATE		TASK NO.			
DECISION TABLE ☐	DRAWN BY		DATE		TASK NAME			
OTHER ☐								

MAR 5 1971

53.42

AA1385 (FORMERLY CA127-1)

53.10 INTRODUCTION

This tape driver is composed of six modules which may be combined in various ways to make up a tape driver. The modules are:

TAPEDR - The traffic director and tape motion control processor

T14 - The tape motion control request processor.

RWBA - Processes the READ and WRITE requests in either binary or ASCII mode.

FRWA - Processes the FREAD and FWRITE requests in ASCII mode.

FRWB - Processes the FREAD and FWRITE requests in binary mode.

RECOVR - Error recovery routine

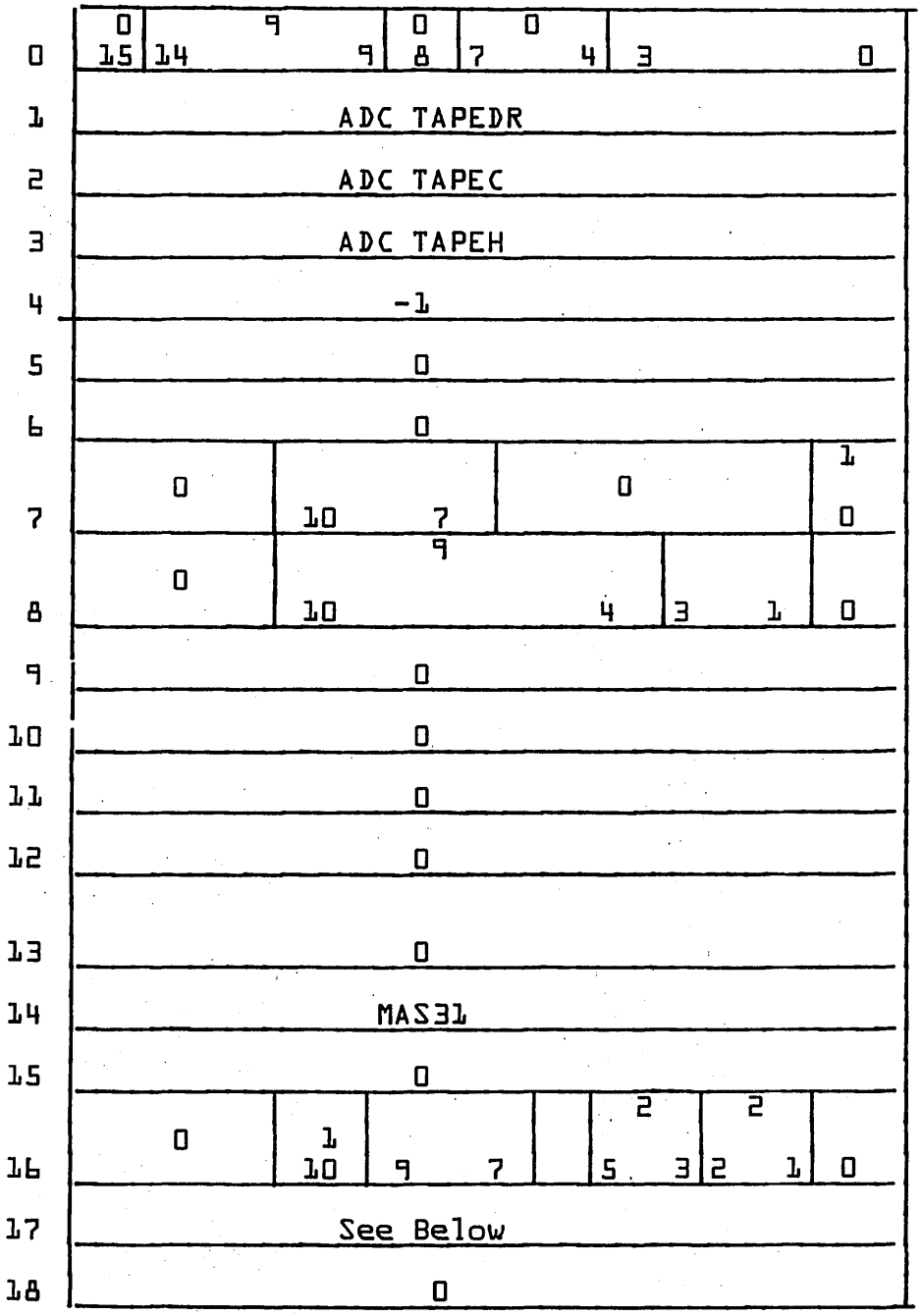
TAPEDR must always be included as all requests pass through it. If density changes and/or tape motion control is desired T14 must also be included. If T14 is not included, a user call will result in an illegal request diagnostic and job termination. In order to perform a read or write operation on magnetic tape any combination of RWBA, FRWA and FRWB routines must be included in the tape driver. A request for one of these routines which is not in resident will result in an illegal request diagnostic and job termination. RECOVR may be included in the driver or replaced or omitted at the user's option. If an error is detected and this routine is not in resident, the operator will be notified of an unrecoverable error (Alternate Device Handler) and given an opportunity to take action.

Module	Entry Points	Externals
TAPDRB	TAPDRB	FRWAB
	TAPCB	FRWBB
	TAPHB	RWBAB
	URECOVB	MAKEQ
	NRMRTB	BUFALC
	RTRYB	ALTDEV
	RTRY2B	
	DYNSTB	

Module	Entry Points	Externals
	CKE0PB	
	INEXRV	
	CKBSY	
	BUFLWA	
	XB	
	BUFFWA	
RWBAB	RWBAB	CKBSY
		INEXRJ
		XB
		DYNSTB
		BUFFWA
		BUFLWA
		RECVTB
		URECVB
		NRMRTB
		CKE0PB
FRWAB	FRWAB	CKBSY
		XB
		DYNSTB
		BUFFWA
		BUFLWA
		URECVB
		RECVTB
		NRMRTB
		CKE0PB
		INEXRJ
FRWBB	FRWBB	INEXRJ
		CKBSY
		XB
		DYNSTB
		BUFFWA
		RECVTB
		URECVB
		NRMRTB
		CKE0PB
RECVTB	RECVTB	CKBSY
	END0PB	RTRYB
		RTRY2B
		RWBAB
		FRWAB
		FRWBB
		INEXRJ
		CKE0PB
		LOG

53.12 Physical Device Tables

Each tape unit in the system will have a PHYSTB (physical equipment table) entry as follows:



WORD 17 contains the location of WORD 1 of the PHYSTB entry of another unit connected to the same buffered data channel. Through this word, all the tape units and other perfericals connected to a Buffered Data channel are threaded together in a circular manner.

If only one tape unit is connected to a controller, word 17 must contain the address of its own PHYSTB entry.

WORD 18 is a temporary storage cell used by the drive to store the tape motion control function or an error code flag.

53.13 Assumptions

1. The Buffered Tape Driver may be interrupted to process higher priority drivers and processes.

2. Only one device on a buffered data channel may be serviced at a time.

3. All callers must be at lower priority levels.

53.14 TAPDRB Description

This module processes all the requests to the tape driver and sends them to the proper routine. It also contains the tape motion control request routine and several subroutines used by all portions of the driver.

53.14.1 INTERNAL DESCRIPTION

On entry, the Driver checks to see if the Buffered Channel is busy. If so, exit is made to the dispatcher as the driver will pick up the current request later on. The Driver then checks all other units connected to channel for a busy unit. If so, exit is made to the dispatcher. If no requests are in progress and the channel is not busy, the driver waits till the controller becomes inactive and then proceeds with request processing.

Two types of requests are processed, tape motion and data transfer. Suitable table setups are made for each type. In addition for data transfers, a check is made to confirm that block length is greater than 9 words. Requests for blocks of less than 9 words cause job termination. When table setups are complete, the unit is connected, mode and density selected. The requests are then passed to a corresponding processing routine. In the case of data transfers, if the corresponding processing routine is not present, job termination results.

53.14.2 USE OF INTERRUPTS

Interrupts are used to signal completion of the following tape motion operations:

1. Backspace record
2. Rewind
3. Mark end of file
4. Skip one file forward

While the above operations are in progress the driver has relinquished control. All other operations require that the driver retain control until they are completed. A partial exception is the rewind-unload request. The Driver initiates a rewind, upon regaining control, it initiates an unload and exits from Driver.

53.4.3 TPMCTL

At TPMCTL a parameter is extracted from WORD 15 and placed in the Q register to use as an ordinal to the request table. The remaining parameters are restored temporarily in WORD 16. Using the ordinal in Q the proper routine address is picked up and placed in Q and a jump is made to the effective address formed by the value in Q plus one. The table format is:

TPMTBL + 0	ADC	DONOTH-1
+ 1	ADC	BSPR-1
+ 2	ADC	MARKEF-1
+ 3	ADC	REWIND-1
+ 4	ADC	UNLOAD-1
+ 5	ADC	SKIPF-1

At TPM1A status is checked to see if tape is positioned at load point. If it is, control is given back to TPMCTL to process the next parameter thus treating the current parameter as a no operation. If the unit is not at load point, control is given to TPM1.

At TPM1 the function in the accumulator is issued to the tape unit and if interrupt was not selected, control is given to TPMCTL. If interrupt was selected, the "I" register (the logical unit number) is saved in SAVEI and exit is made to the dispatcher.

At DONOTH (Do Nothing) the current status is taken and stored in WORD 13 of the PHYSTB, and exit is made to CONTIU.

At BSPR the control function to select a backspace and interrupt is placed in the accumulator and control is given to TMP1A.

At MARKEF the control function to select a write or file mark and interrupt is placed in the accumulator and control is given to TPM1.

At REWIND the control function to select a rewind and interrupt is placed in the accumulator and control is given to TPM1A.

At UNLOAD, WORD 16 for this logical unit is set to zero to force DONOTH to be processed as the next parameter if the tape unit is at the load point. If not at the load point, the REWIND control function is selected, with unload being set up as the next function to be processed. Unload will terminate this request as the tape unit will require namual intervention by the operator before it may again be addressed.

At SKIPF a non stop read is performed and at each end of operation, status is checked for end of file and end of tape. When either is detected control is given back to TPMCTL to process the next parameter.

Upon entrance from an end of operation interrupt TAPEC is entered and the logical unit from SAVEI is restored in the "I" register. A

function code to clear the interrupt is placed in the accumulator and control is given to TPM1.

CONTIU is entered at the completion of each request. The V indicator of WORD 10 bits 13 to 15 are computed. If an error was detected the Alternate Device Handler is scheduled with Q containing the error code and logical unit number, control is given to SEARCH.

SEARCH examines all units connected to the Buffered Channel (via thread in WORD 15) it sees if one needs service. If it finds one, the PHYSTD location for the unit is put in "Q" and the DRIVER is scheduled. The FOVER flag is zeroed and control is passed to the Dispatcher. In this way, the tapes are activated in a commutative manner until all requests to all units have been processed. When this occurs, exit will be made to the dispatcher.

53.15 T14 - TAPE MOTION CONTROL REQUEST PROCESSOR

This request processor adjusts return address for direct calls.

When the above is completed, T14 exits to entry point "SAVLU" in the Read/Write Request Processor (E00610A0060S) where threading and driver scheduling take place. All legality checking for unprotected requests is done in the Protect Processor (E00610A0160S). It is assumed that protected requests do not require legality checking.

53.16 RWBA - READ AND WRITE IN BINARY OR ASCII MODE

Upon entry to RWBA the contents of WORD 8 for this logical unit are stored in local storage EQTAB with bits 1-0 set to zero. The first word address is stored in CORE and the last word address + 1 is stored in LAST. X in TAPEC is modified so that control will be passed to INTRW at interrupt time. The Buffer first word address is stored into HOLD with the last word address + 1 stored in Buffer first word location. If this is a write request, control is given to ENRING. If not, the Read motion and End of Operation interrupt function is given. A buffered input operation will then be initiated. Control will be passed to TPEBSY which saves I in SAVEI and jumps to DYNSTA. Interrupt will be enabled while Initializing a buffered input or output.

At ENRING the unit is checked to see if it is write enabled. If it is not, core and LAST are placed in A and Q and control is given to URECOV. If it is write enabled, the data is packed into the buffer prior to a buffered write.

Upon receiving an End of operation on a buffered read or write, the driver will be reentered at INTRW. SAVEI is restored in I and the next storing address is requested. The previous interrupt condition

is cleared and the status is updated. If an end of file was read, $3CF0 stored in first word of user's buffer and control is passed to NORMRT. If a parity error was detected, control exits to RCVR. If a write operation is being processed control is passed to NORMRT. Otherwise the data buffer is unpacked and placed in users buffer; control is then passed to NORMRT. RCVR checks to see if the error recovery program RECOVR is present, if not present, control is passed to URECOV. If present control is return jumped to RECOVR. If the error could not be recovered from, URECOV is entered.

53.16.1 PACKING AND UNPACKING READ/WRITE REQUESTS

Each 1700 core location will be written or read as three 6 bit characters. Two trailing ones will be added onto the low order bits of each word on a write and stripped off as the data is read; i.e., bits 15-10 will be the first character written, bits 9-4 will be the second and bits 3-0 plus two low order ones will be the third. On reading the tapes, the first characters will occupy bits 15-10, the second will occupy bits 9-4 and the third will have its two low order bits stripped off and will occupy bits 3-0. The mode may be either binary or ASCII but will only result in the binary recording (odd parity) or BCD recording (even parity) respectively of the data on tape. No conversions take place. Writing zeros in even parity will cause difficulty in the controller and should be avoided.

53.17 FRWA - FORMAT READ AND WRITE IN ASCII MODE

The FRWA program is set up the same as RWBA except for packing and unpacking (see 5.1).

53.17.1 PACKING AND UNPACKING FORMATTED READ/WRITE REQUESTS

Formatted reads or writes in ASCII mode will assume ASCII code in 1700 core and external BCD (six bits plus parity) recorded in BCD (even parity) mode on tape. The conversion takes place by direct table look up (see 5.2).

53.17.2 ASCII/BCD CONVERSION TABLE

TABLE + 00	26	IF
+ 01	31	31
+ 02	32	32
+ 03	33	33
+ 04	34	34
+ 05	35	35

TABLE + 06	36	36
+ 07	37	37
+ 08	38	38
+ 09	39	39
+ 0A	30	21
+ 0B	3D	22
+ 0C	27	23
+ 0D	3A	24
+ 0E	3E	25
+ 0F	22	26
+ 10	20	27
+ 11	2F	28
+ 12	53	29
+ 13	54	12
+ 14	55	13
+ 15	56	14
+ 16	57	15
+ 17	58	16
+ 18	59	17
+ 19	5A	18
+ 1A	5F	19
+ 1B	2C	3D
+ 1C	28	1E
+ 1D	25	2D
+ 1E	5C	2F
+ 1F	40	1A
+ 20	2D	10
+ 21	4A	2A
+ 22	4B	0F
+ 23	4C	3F
+ 24	4D	3B
+ 25	4E	1D
+ 26	4F	1D

TABLE	+ 27	50	0C
	+ 28	51	1C
	+ 29	52	3C
	+ 2A	21	2C
	+ 2B	24	30
	+ 2C	2A	1B
	+ 2D	5D	20
	+ 2E	3B	3B
	+ 2F	5E	11
	+ 30	2B	0A
	+ 31	41	01
	+ 32	42	02
	+ 33	43	03
	+ 34	44	04
	+ 35	45	05
	+ 36	46	06
	+ 37	47	07
	+ 38	48	08
	+ 39	49	09
	+ 3A	3F	0D
	+ 3B	2E	2E
	+ 3C	29	3E
	+ 3D	5B	0B
	+ 3E	3C	0E
	+ 3F	23	3A

53.18 FRWB - FORMAT READ AND WRITE IN BINARY MODE

The FRWB program is set up the same as RWBA except for packing and unpacking (see 53.18.1).

53.18.1 PACKING AND UNPACKING FORMATTED READ/WRITE REQUESTS

Formatted reads or writes in binary mode will assume binary code in 1700 core and binary (even parity) mode on tape. On reads each six bit character received from the tape unit is packed left justified

(is starting at bit 14 of word 1 and moving towards 0 of the last word). On writes the procedure is the same with every bit working from left to right and top to bottom, being written on tape contiguously. If a write or a read operation does not terminate on an exact word, the last character written on tape or read from tape is padded with zeros.

53.19 RECOVR - THE ERROR RECOVERY PACKAGE

This routine may be replaced at the users option. Its entry point is labeled RECOVR. It is return jumped to with Q containing pointer to routine entered from.

Q	Routine
0	RWBA
1	FRWA
2	FRWB

Upon entry to RECOVR the error recovery code in Q is saved in RCODE. If the error was detected on a read and RETRY is not equal to three, control is given to R1. R1 increases RETRY and the record is backspaced and control is given to XIT, which causes the record to be read again. If RETRY is equal to three, control is given to CLEAN where RETRY2 is checked. If RETRY2 is equal to four the routine exits back to the calling routine through its entry point.

If RETRY2 is not equal to four control is given to CLEAN2. At CLEAN2, RETRY2 is incremented by one, ENDOP is set to zero and up to three backspaces are issued. If load point is sensed after any of the backspaces, no more backspaces are done. A count of the number of extra backspaces actually executed is kept in ENDOP. A non stop read is done and continues until the number of end operations specified in ENDOP have been read. At this point the error record is the next record to be read and control is given to XIT.

At XIT the routine determines which routine to re-enter by using the value in RCODE as an ordinal to a table or address. The format of the table is:

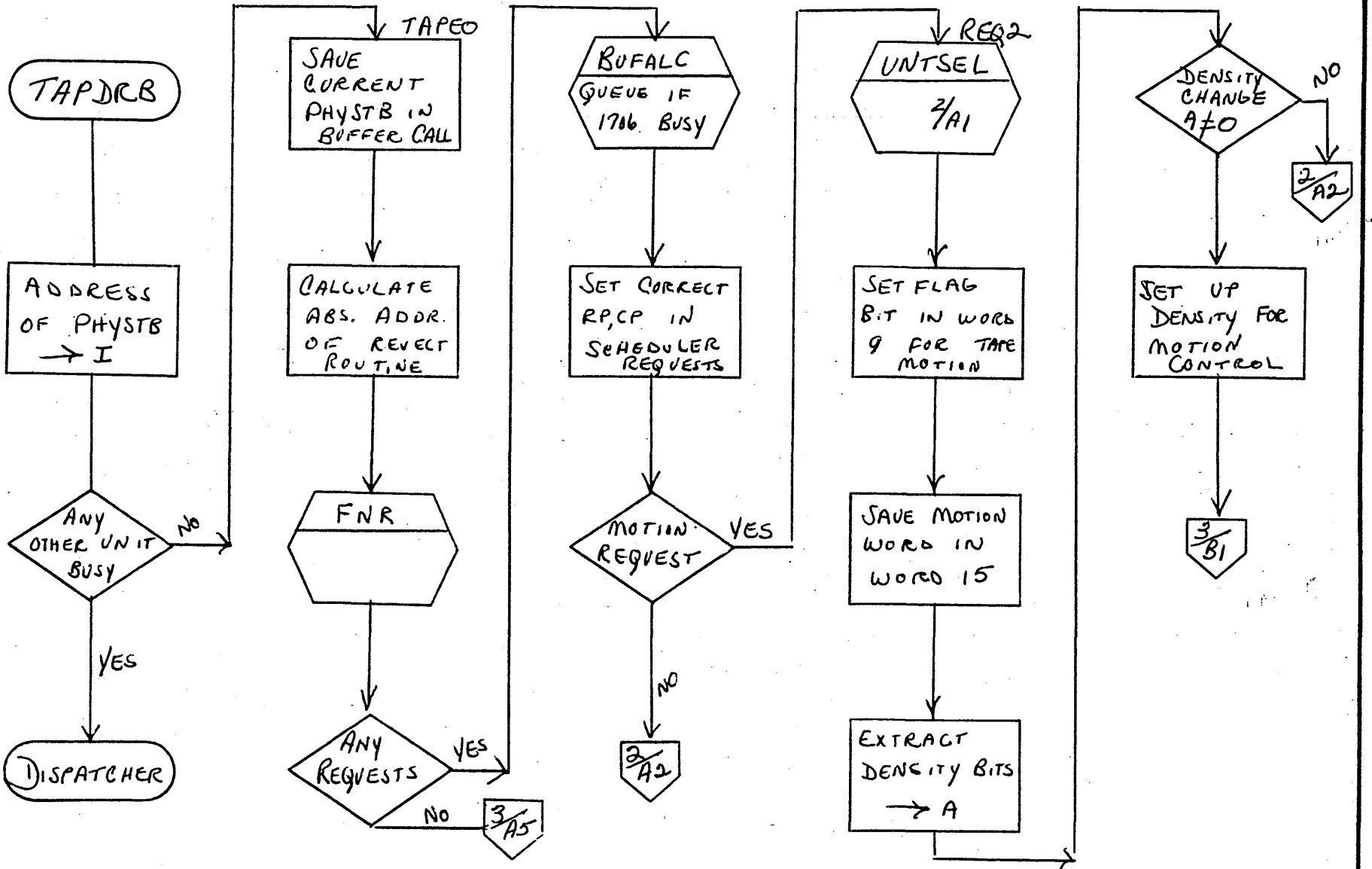
TBL + 0	ADC	RWBA
+ 1	ADC	FRWA
+ 2	ADC	FRWB

If the error was detected on a write and RETRY is not equal to two, control is given to R1.

If it is equal to two control is given to W1. At W1 if RETRY2 is equal to two, control is given to W6. If it is less than two, control is given to W3 and if it is greater than two, control is returned to the calling routine through the entry point of RECOVR.

At W3 the tape unit is backspaced and at least half of the record is erased. The number of erases performed is calculated by the formula $\frac{(N-1)C}{12D} + 1$ where N is the number of computer words to be written, C is the number of characters per computer word and D is the density the data is to be recorded in. The erases are performed by writing a file mark and backspacing. When the computed number or erases have been performed control is given to W6.

At W6, zero is stored in RETRY and control is given to XIT.

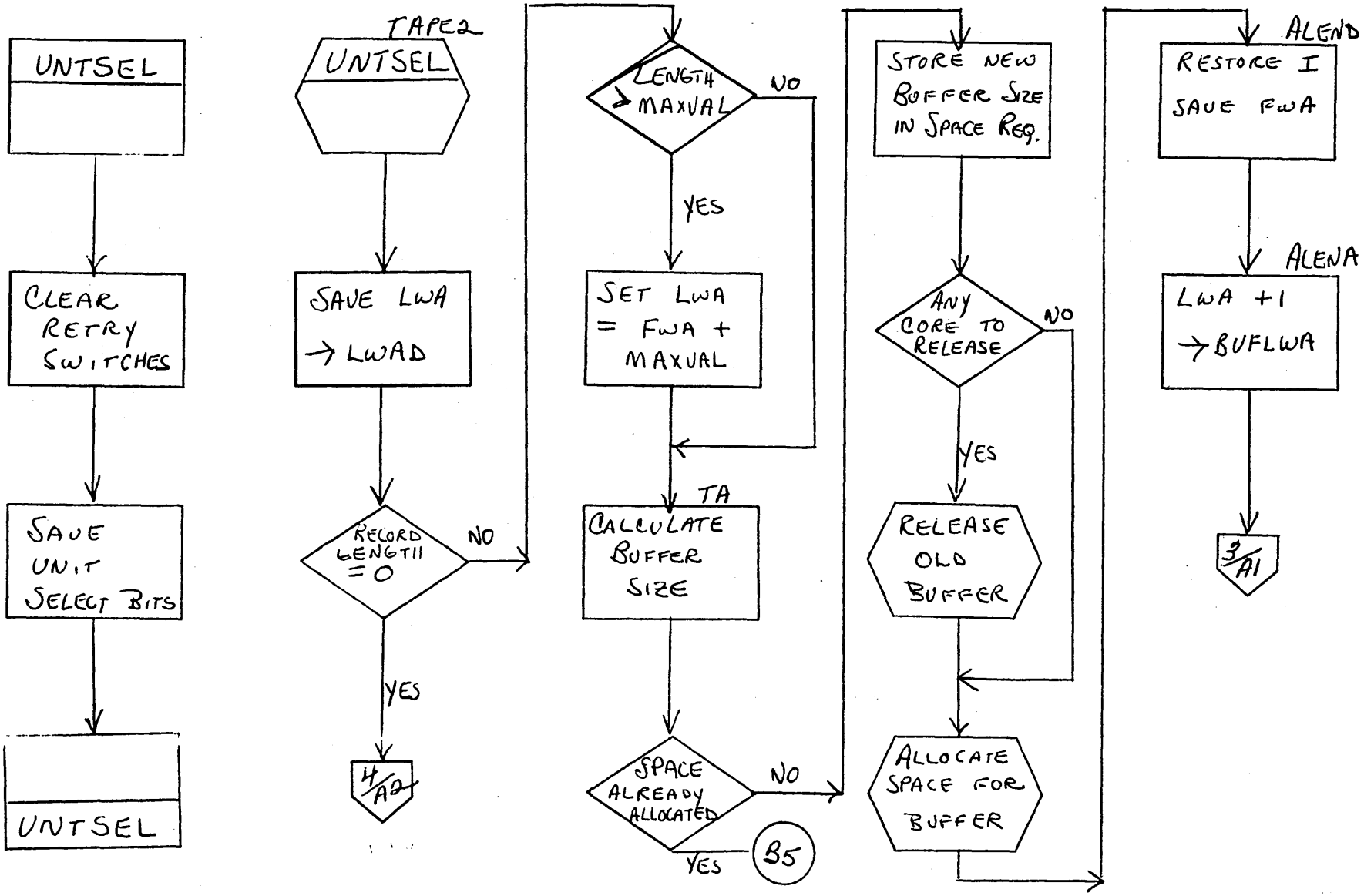


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPDRB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 1 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53-55

AA1385 (FORMERLY CA127-1)

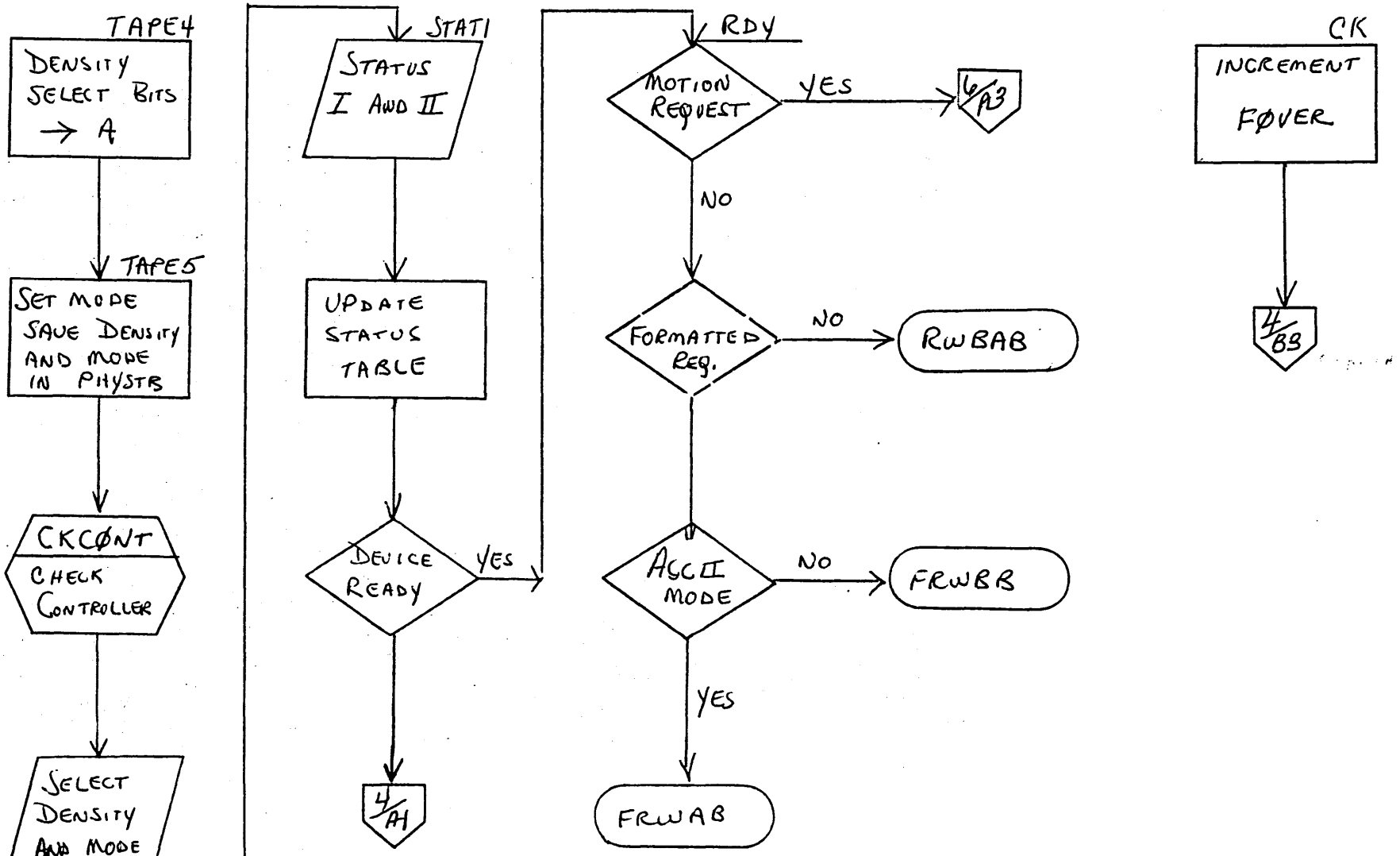


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPDRB	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER BUFFERED	PAGE 2 OF 22	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.56

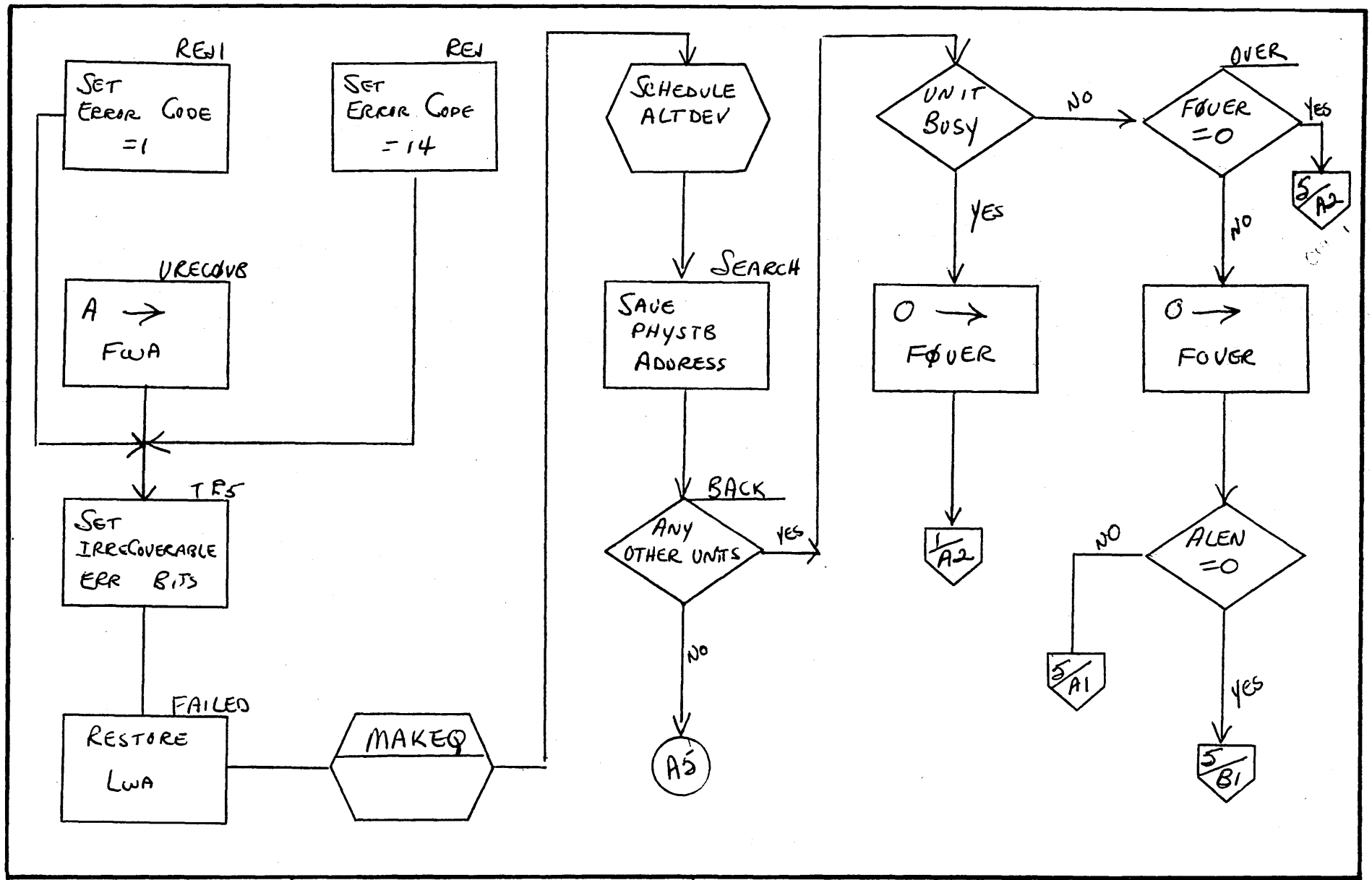


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700	TAPDRB	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER	PROJECT MGR.			
	BUFFERED PAGE 3 OF 22	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.57

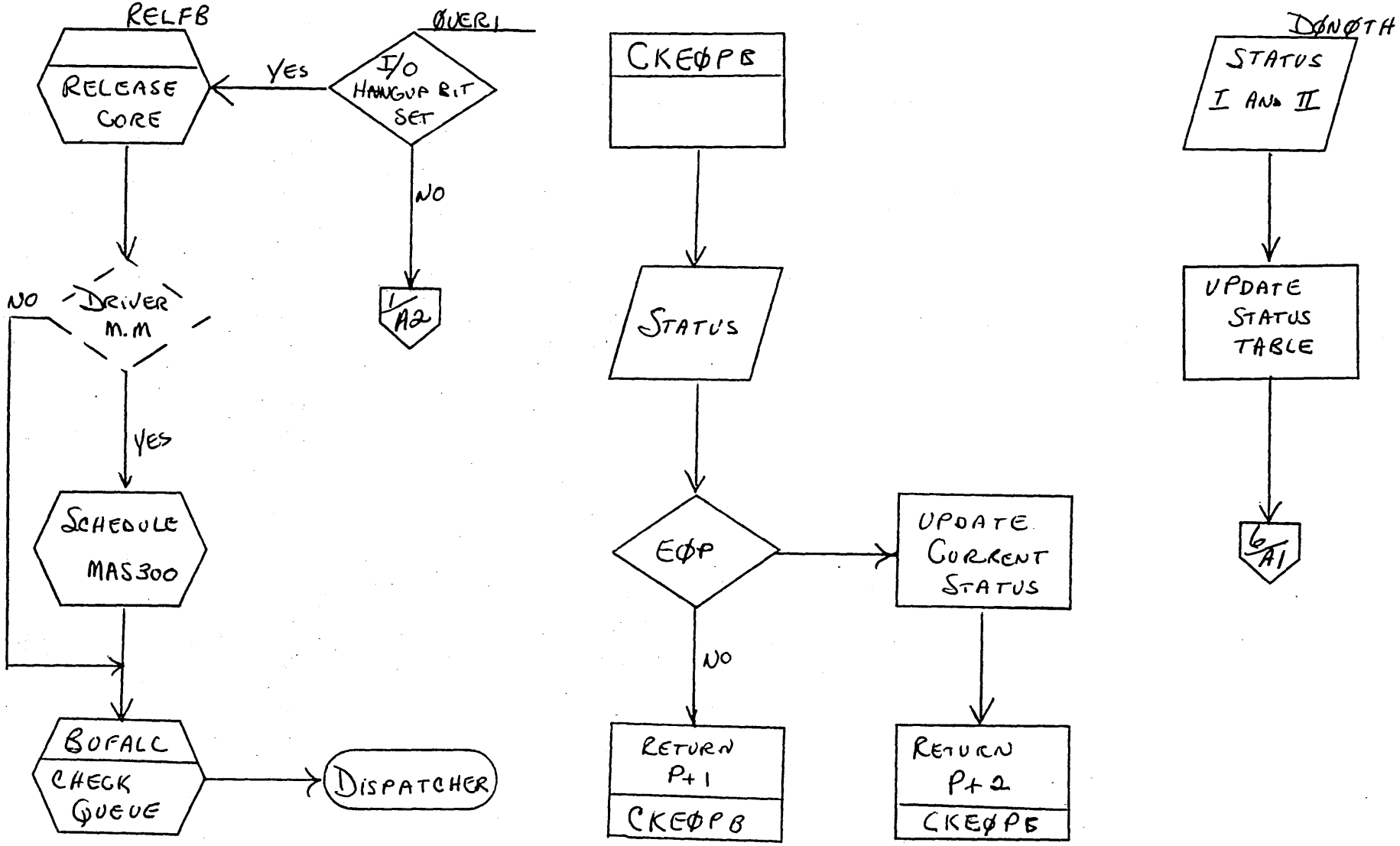


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPDRB	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER BUFFERED	PAGE 4 OF 22	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.58

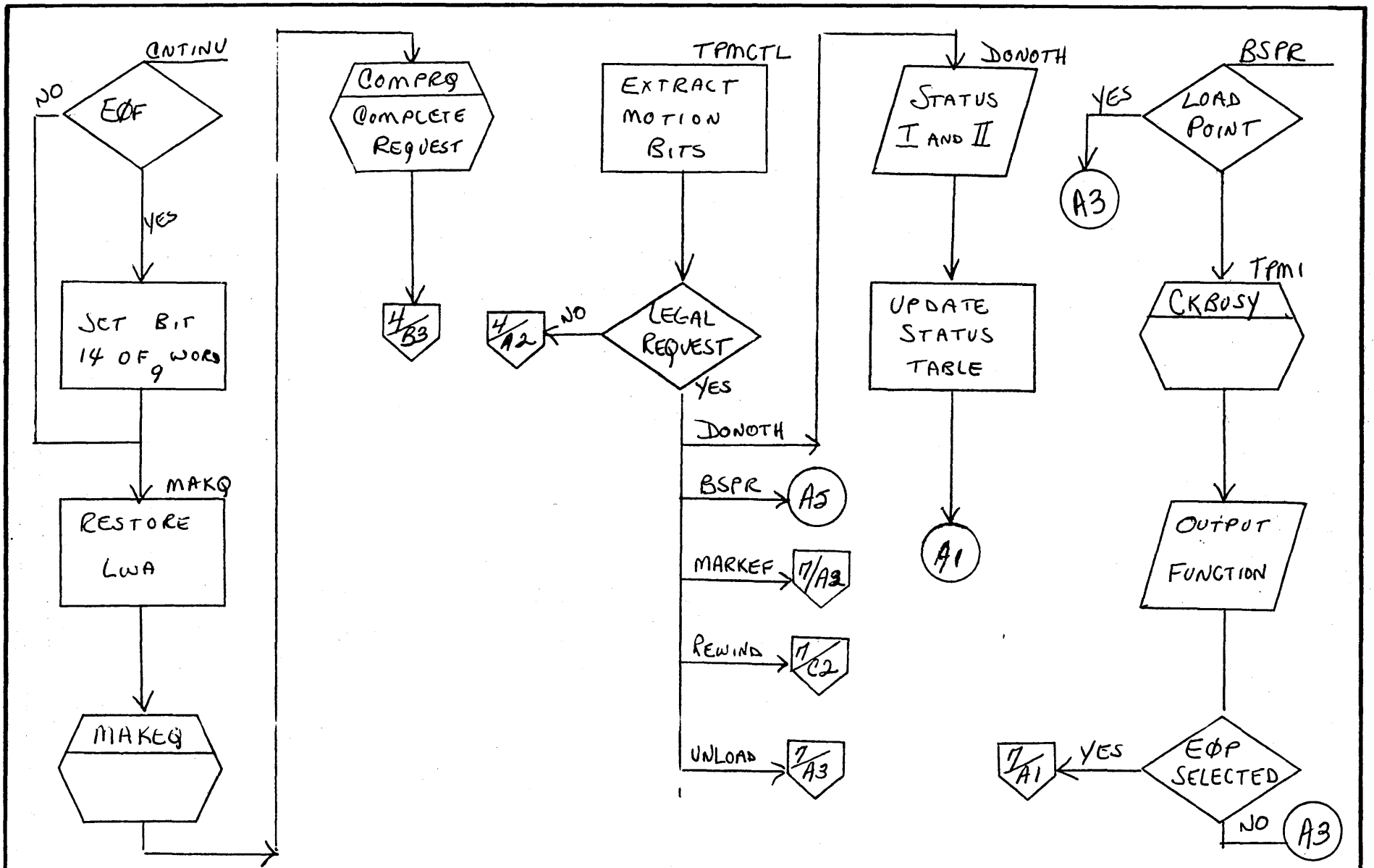


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPDRB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 5 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

53.59

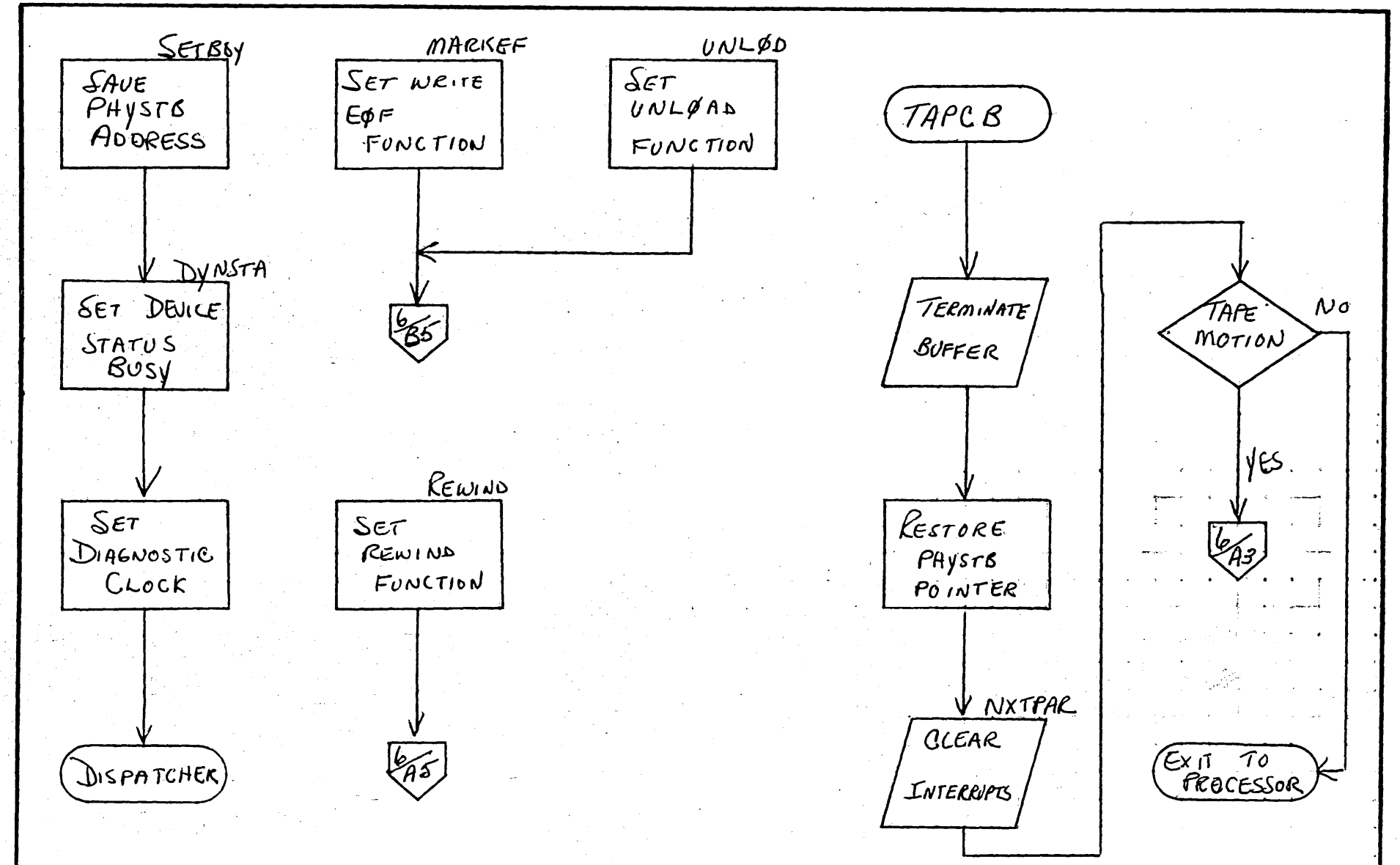


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPDRB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 6 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

533.60

AA1385 (FORMERLY CA127-1)



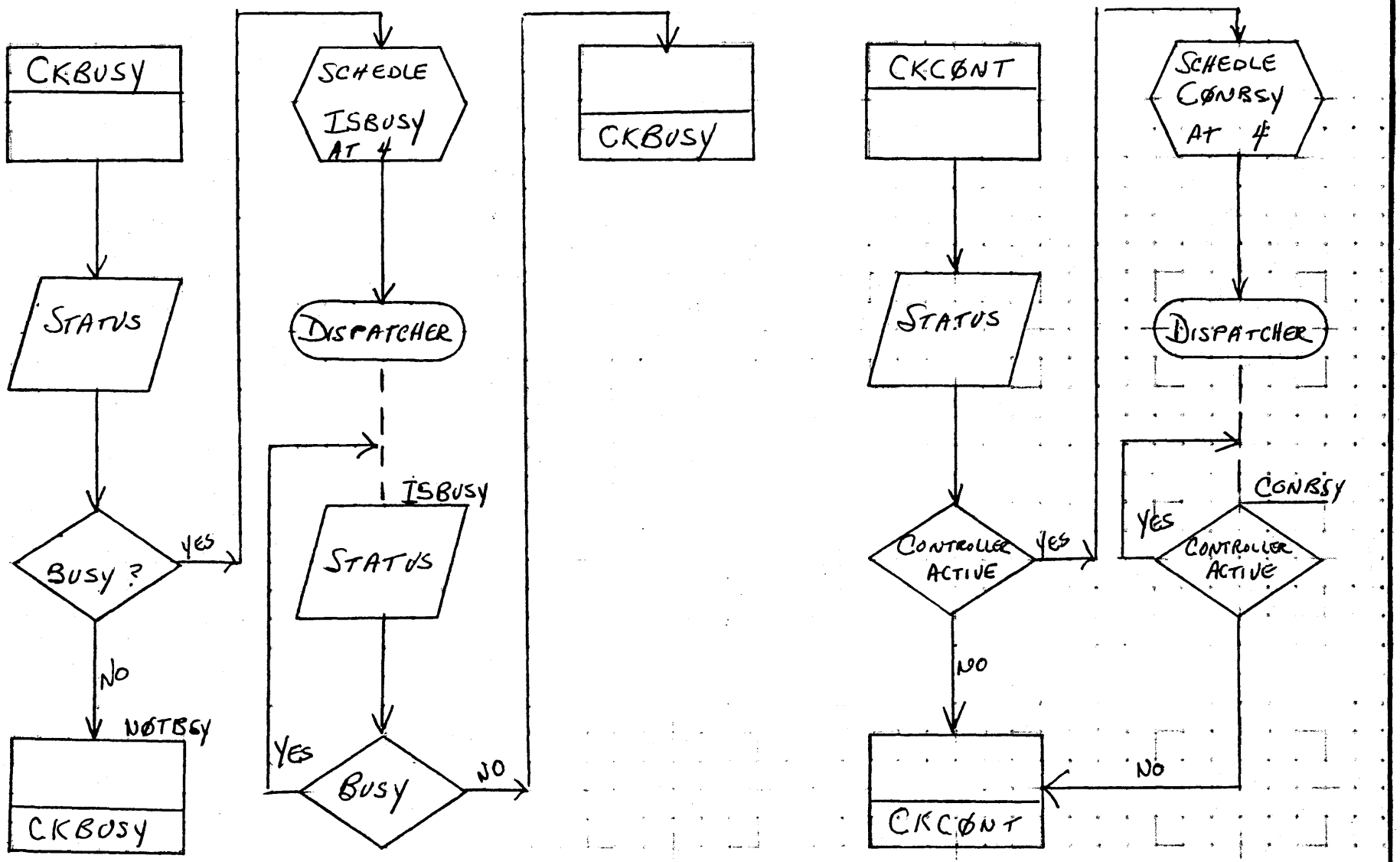
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	TAPDRB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 7 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

53-61



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	TAPDRB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 8 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

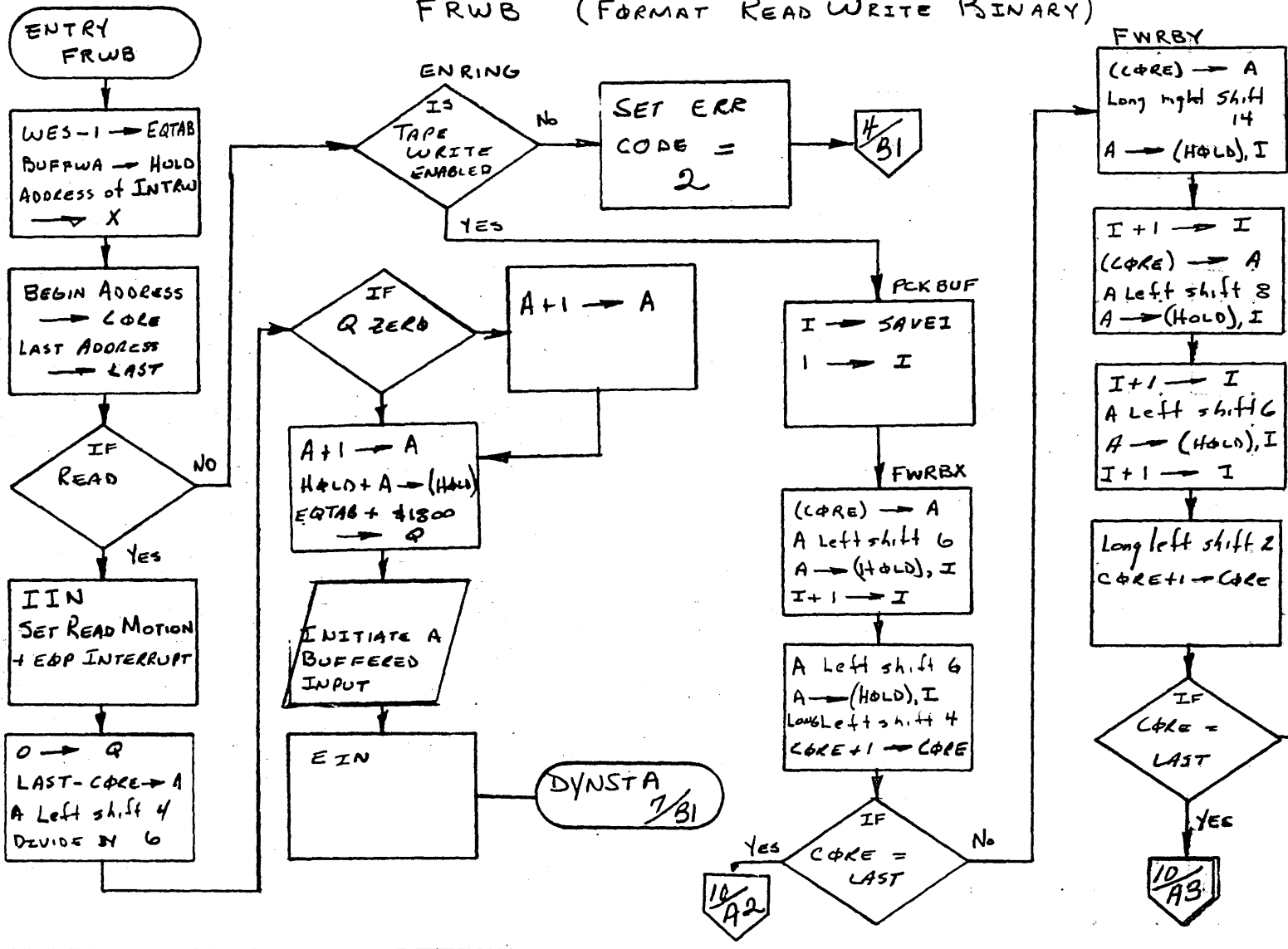
MAR 5 1971

53.62

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

FRWB (FORMAT READ WRITE BINARY)

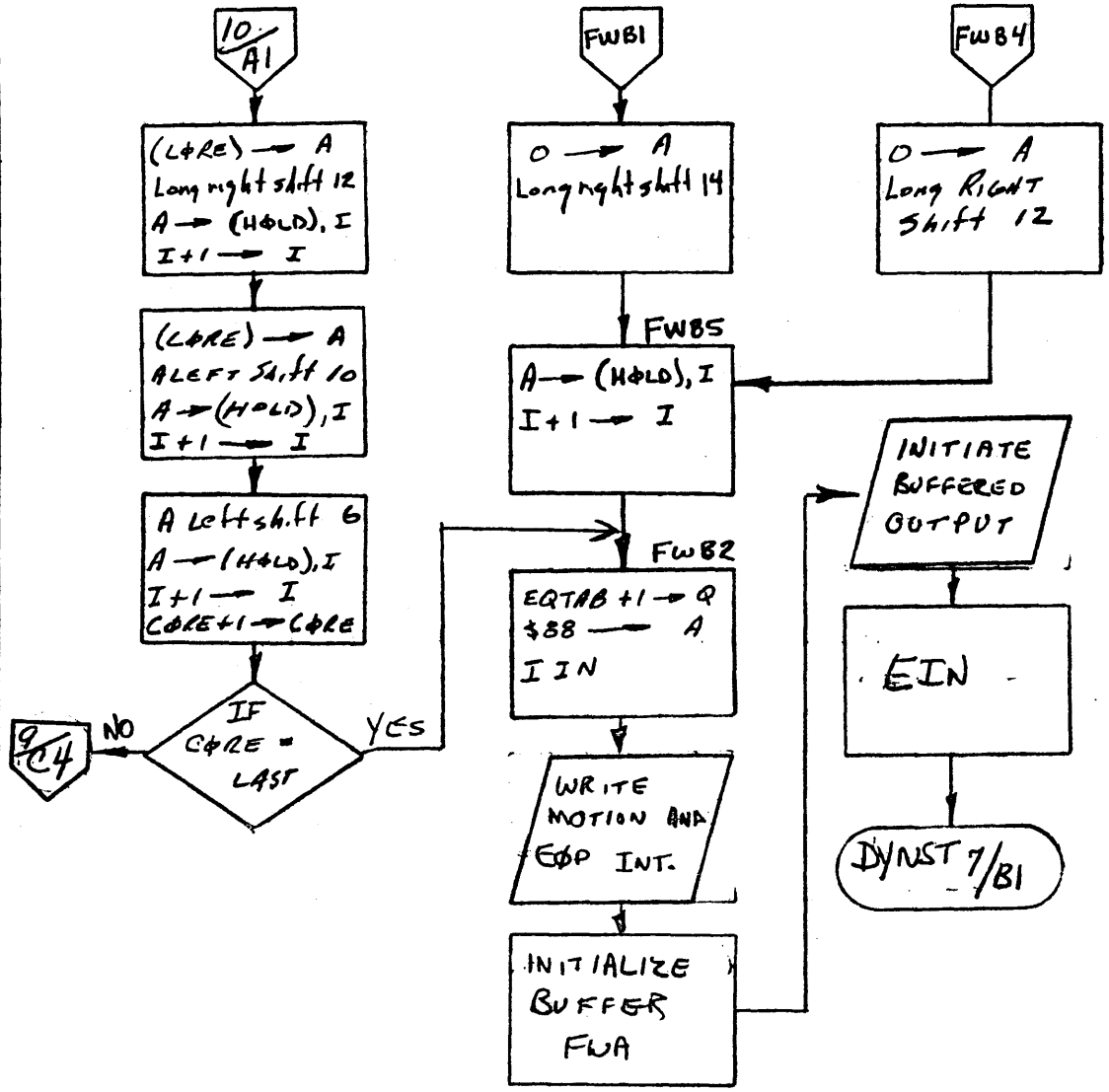


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWBB	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	BUFFERED	PAGE 9 OF 22	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.63

AA1385 (FORMERLY CA127-1)

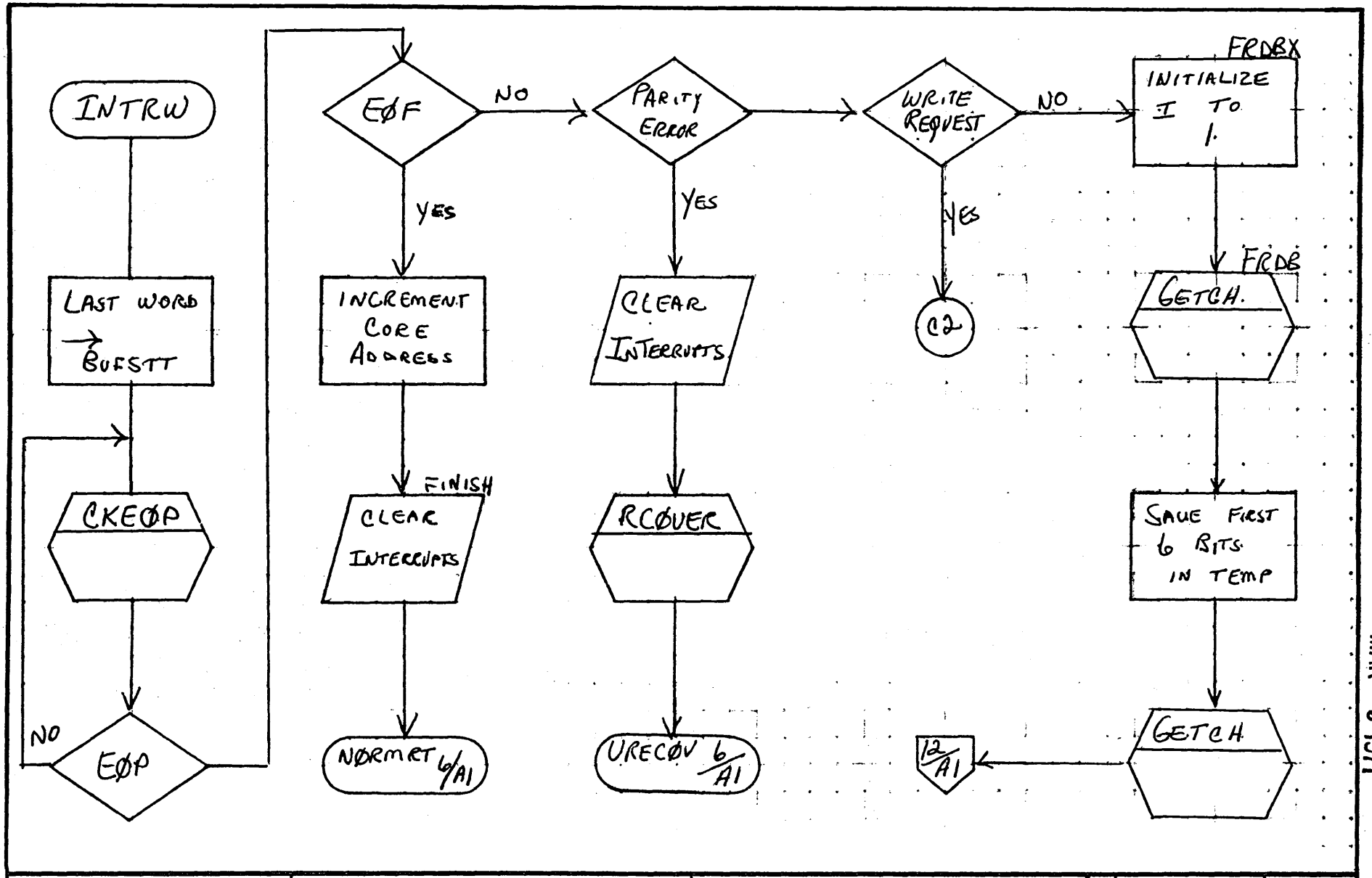


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWBB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 10 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

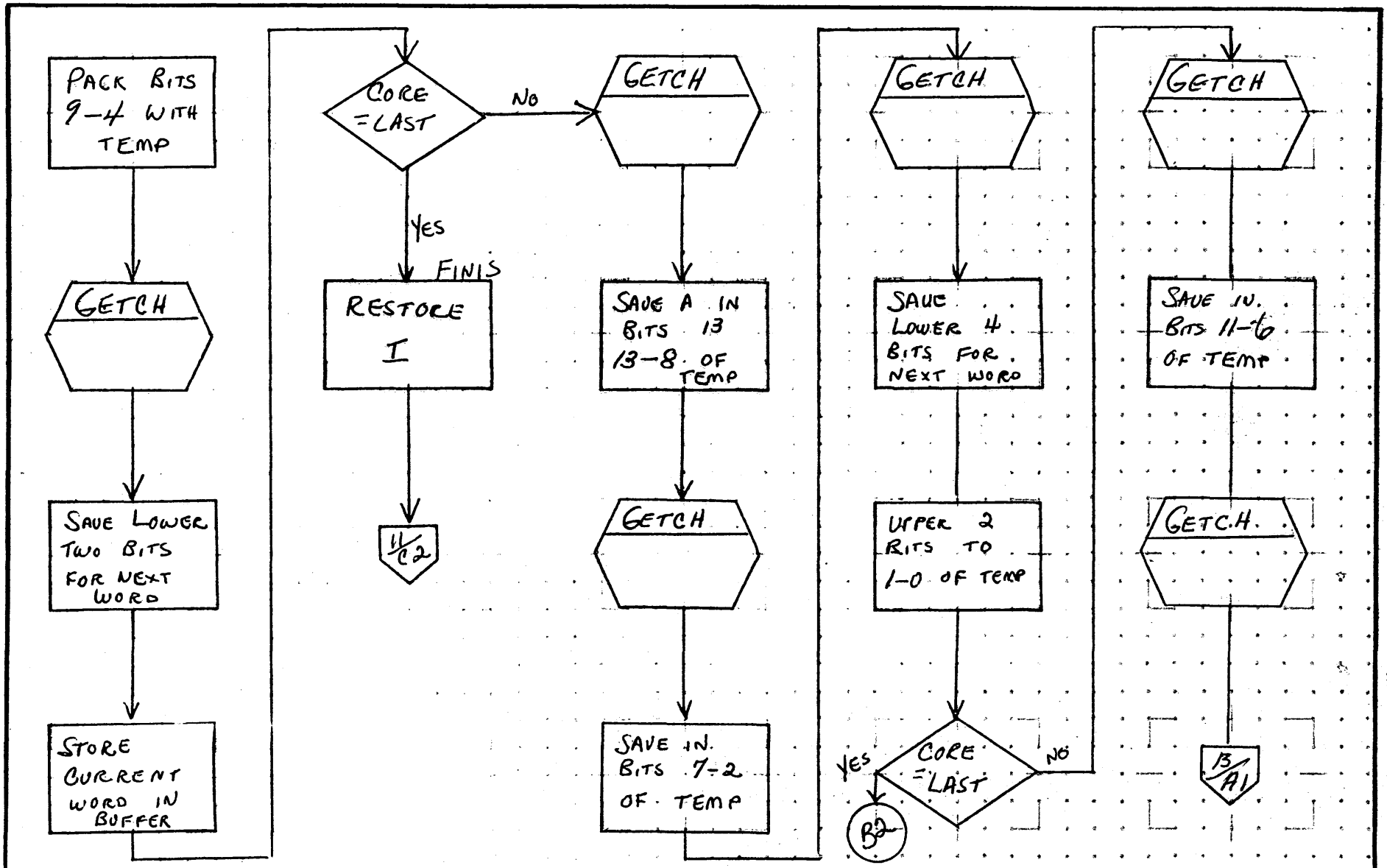
53.64



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	FRWBB	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER	PROJECT MGR.			
	BUFFERED PAGE 11 OF 22	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

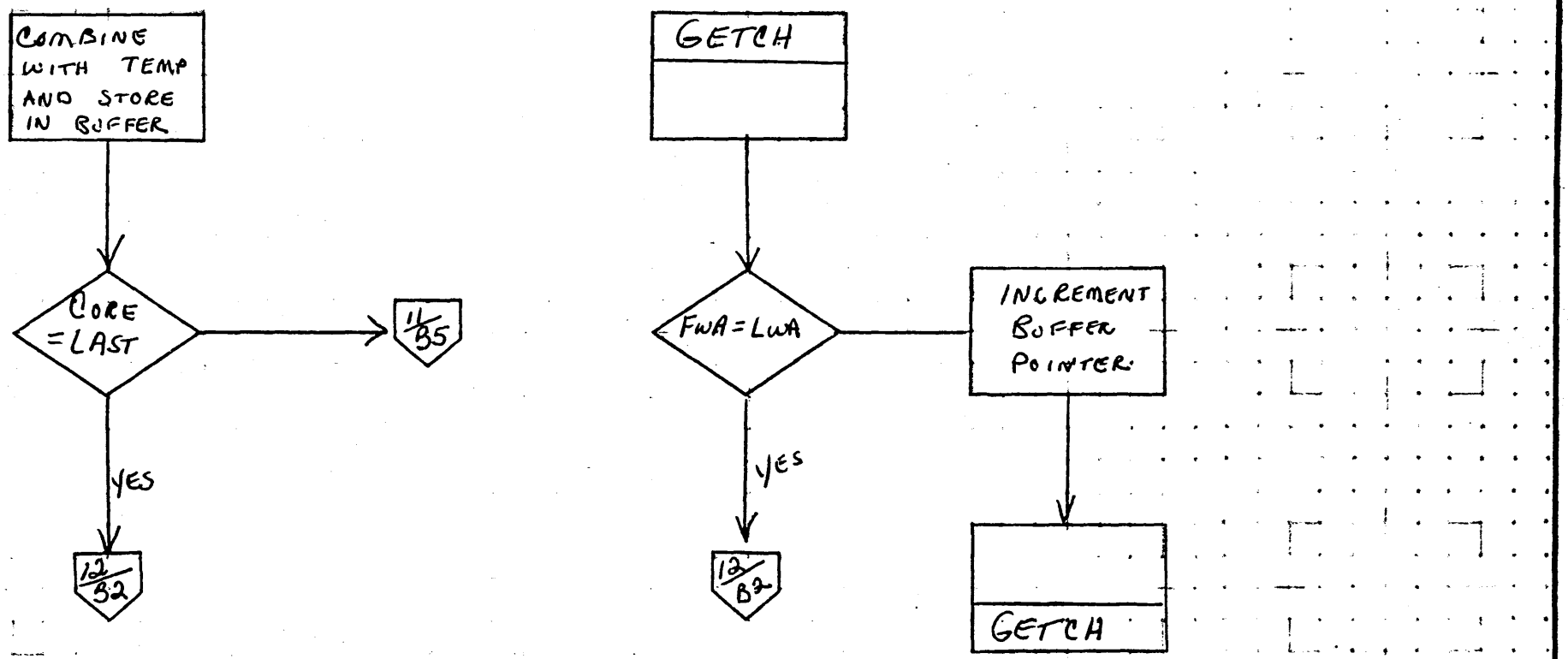
53.65



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWBB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 12 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

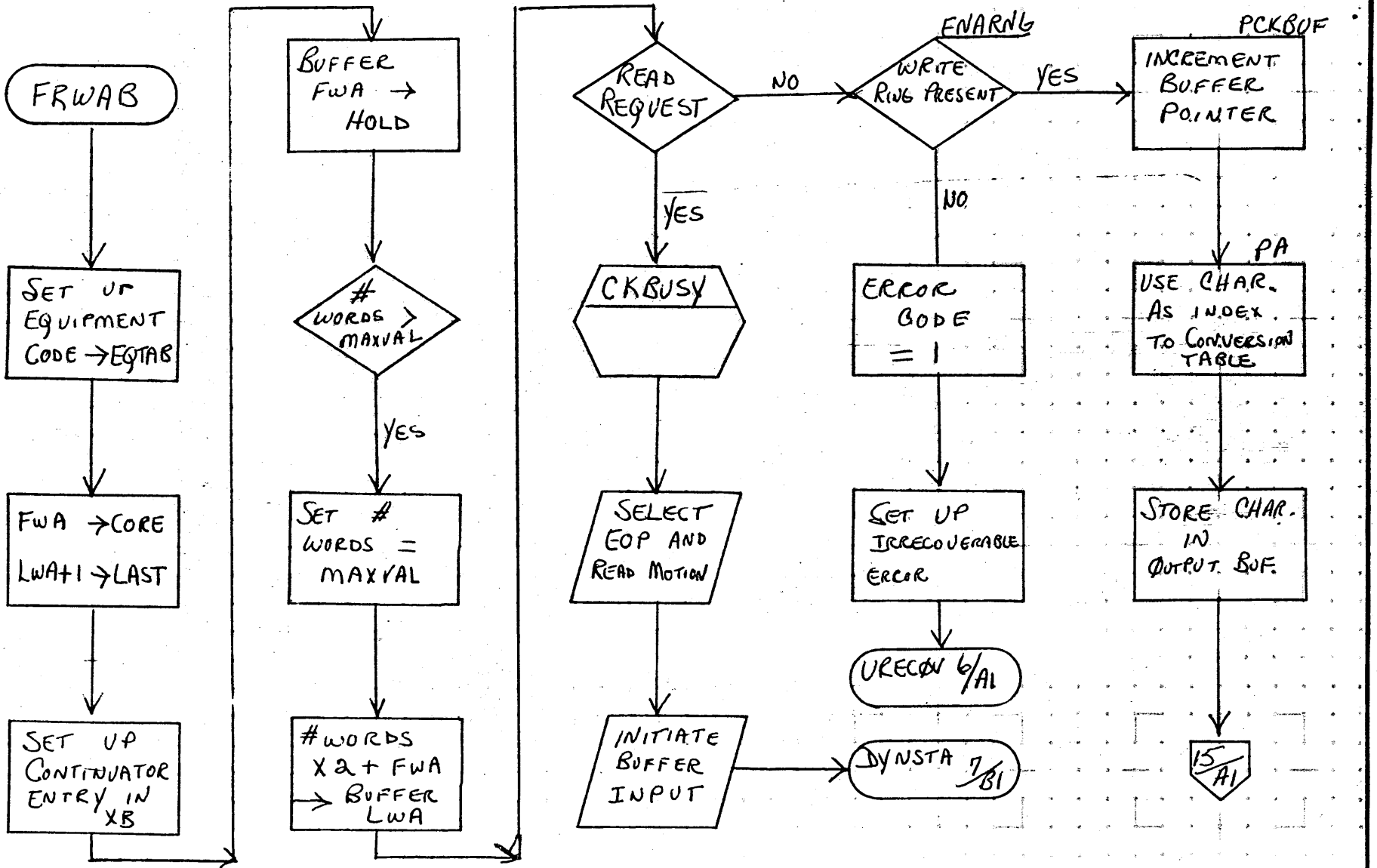
53.66



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWBB	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER BUFFERED	PAGE B OF 22	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

53.67



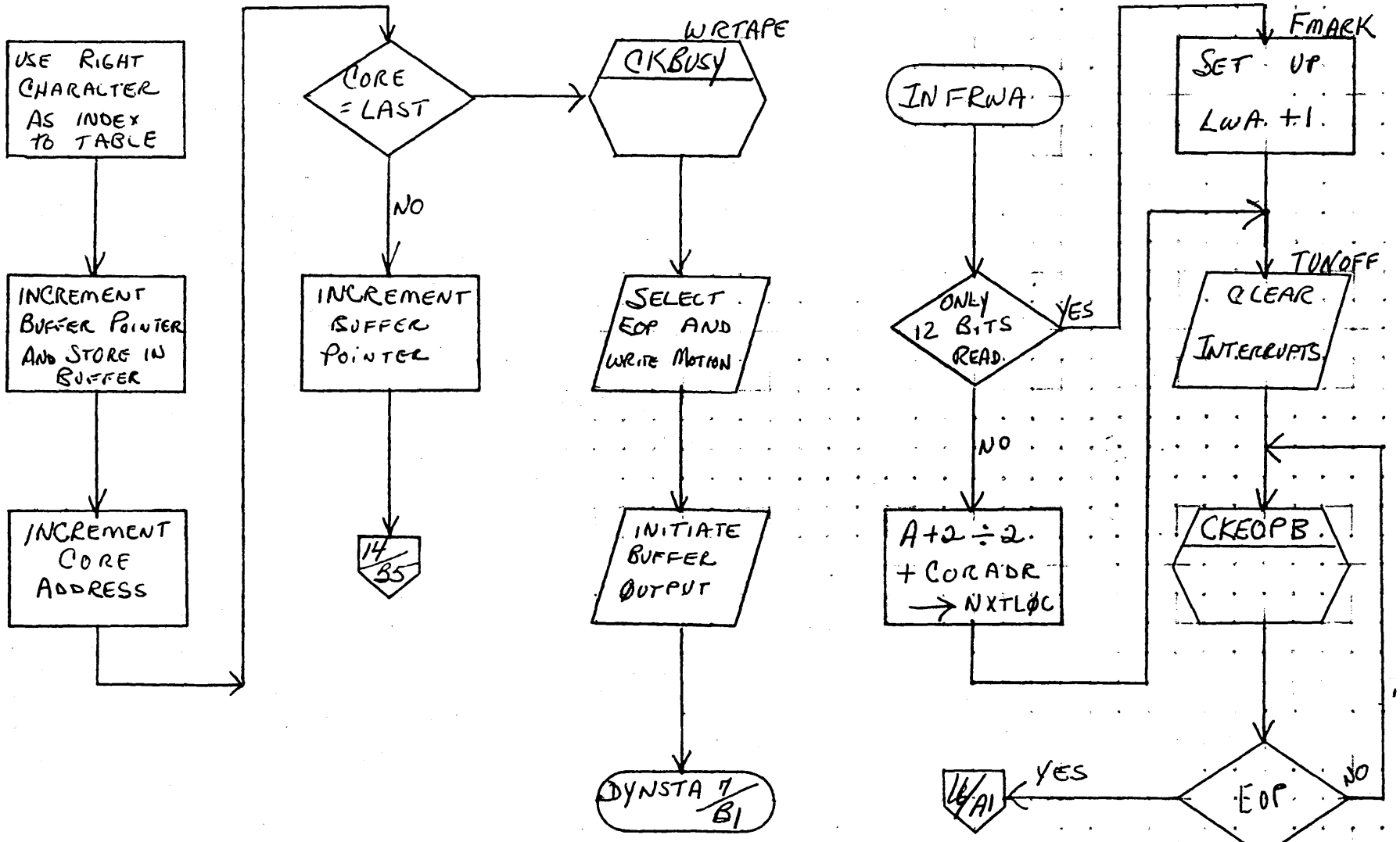
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWAB	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER BUFFERED	PAGE 14 OF 22	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

533.68

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS EMS	MACH. TYPE 1700	FRWAB
	DOCUMENT TITLE 1731/601 DRIVER BUFFERED	PAGE 15 OF 22	PROJECT MGR.
	NUMBER	ISSUE DATE	PROJECT NAME
	DRAWN BY	DATE	TASK NO.
			TASK NAME

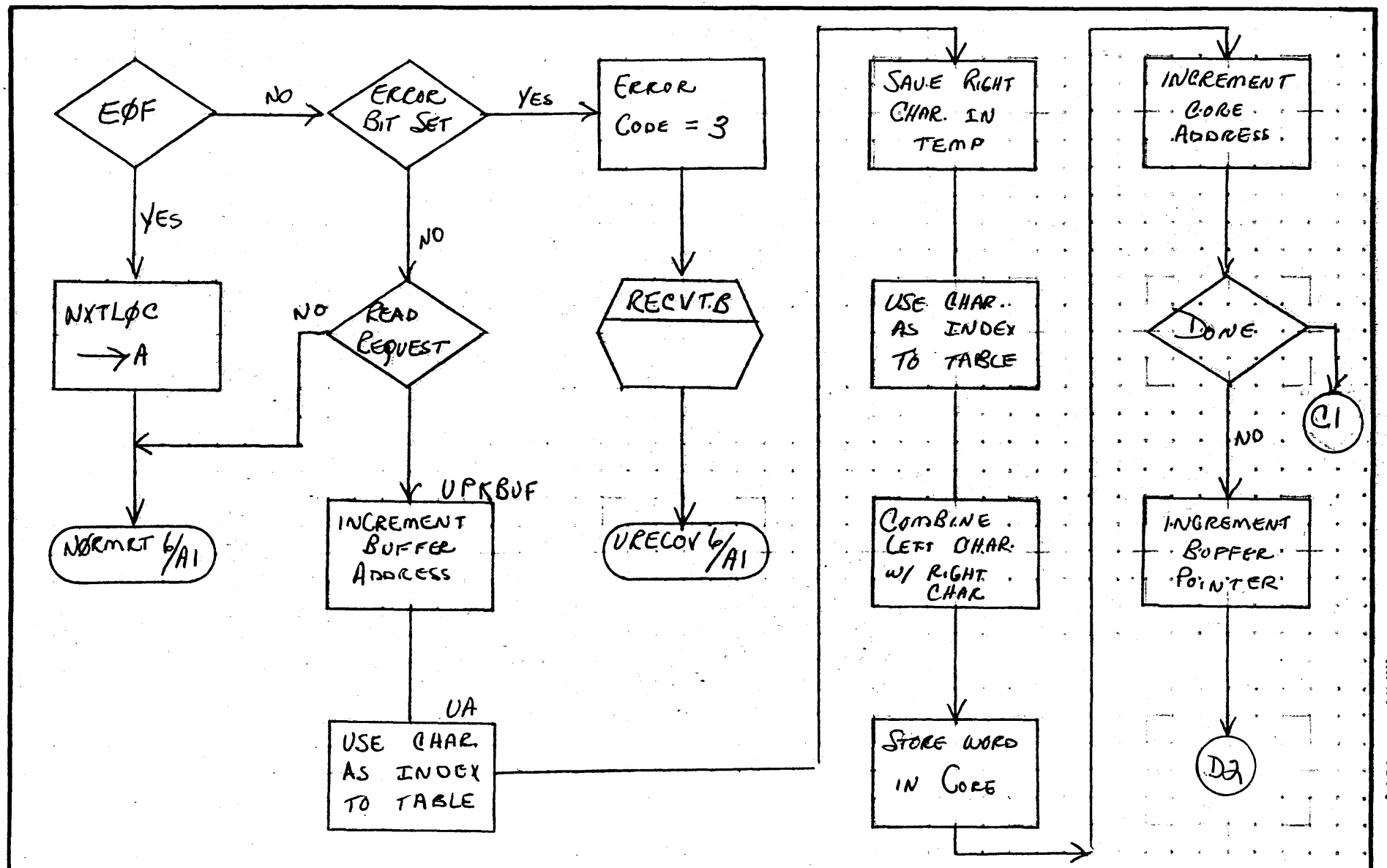
REV	APPROVED	DATE

MAR 5 1971

533.69

AA1385 (FORMERLY CA127-1)

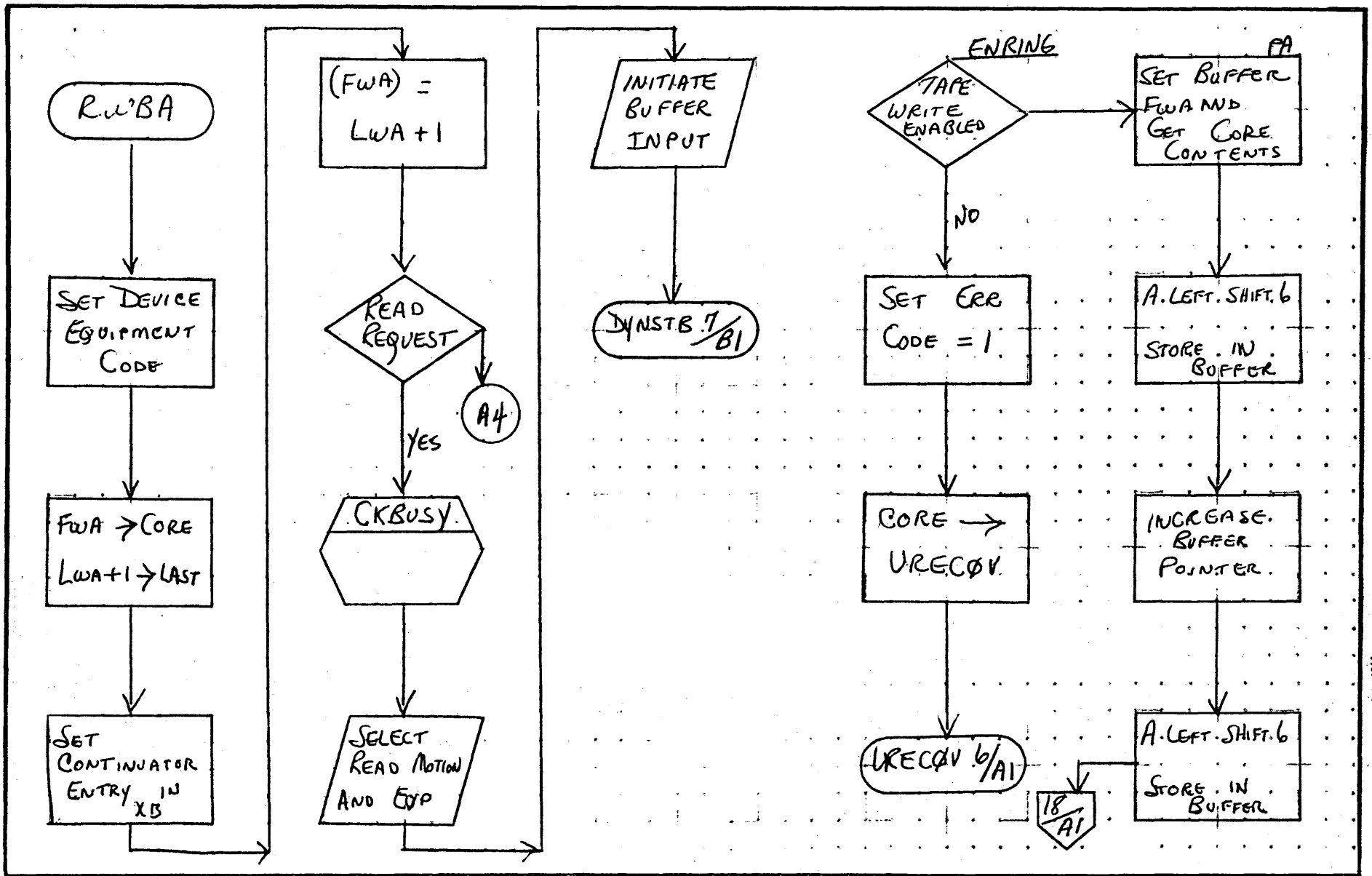
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	FRWAB	REV	APPROVED / DATE
	DOCUMENT TITLE 1731/601 DRIVER BUFFERED	PAGE 16 OF 22	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

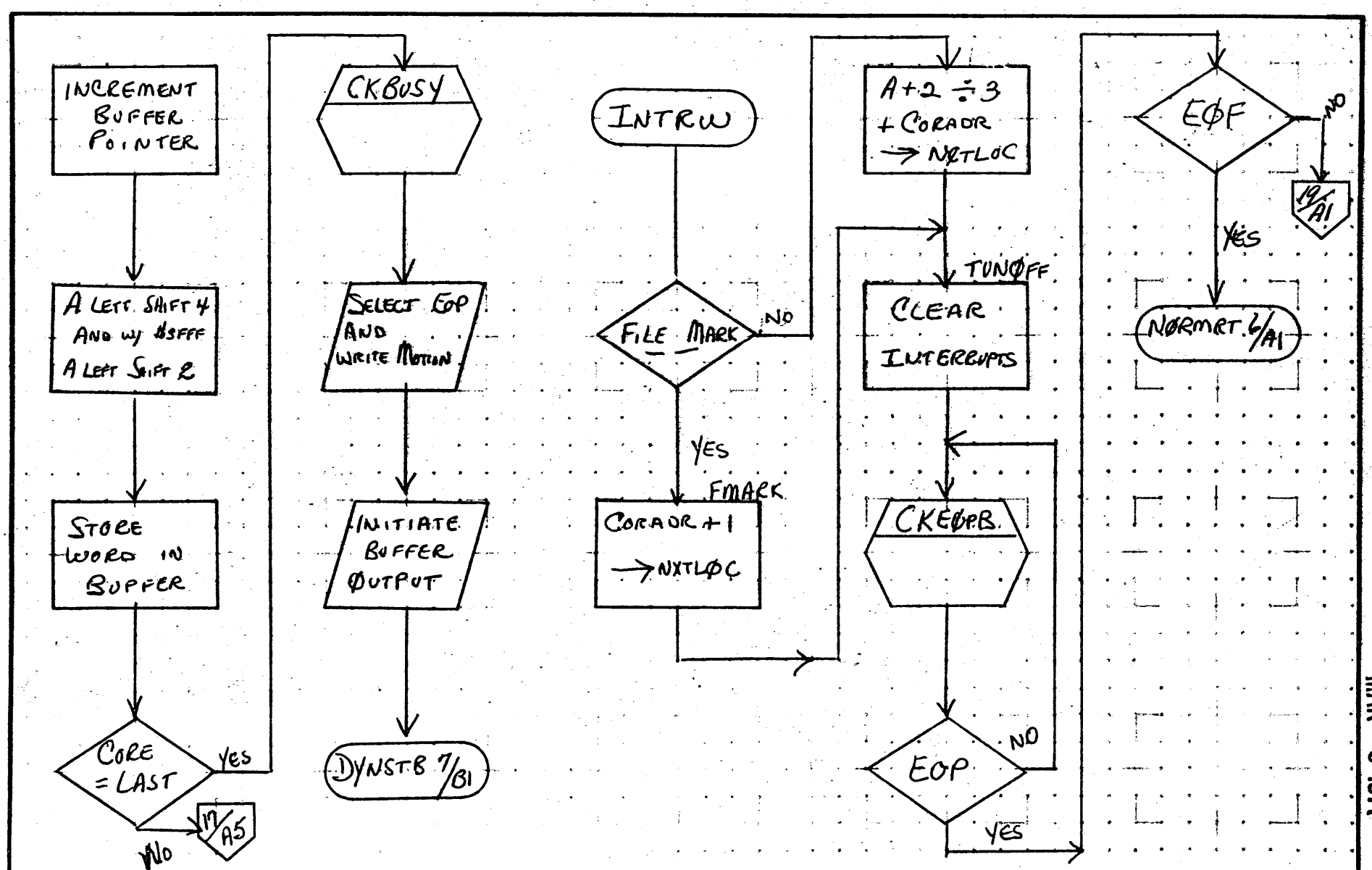
53.70



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	RWBAB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 17 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.71



CONTROL DATA CORPORATION			
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	RWBAB
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.
FLOWCHART ☐	BUFFERED	PAGE 18 OF 22	PROJECT NAME
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.
OTHER ☐	DRAWN BY	DATE	TASK NAME

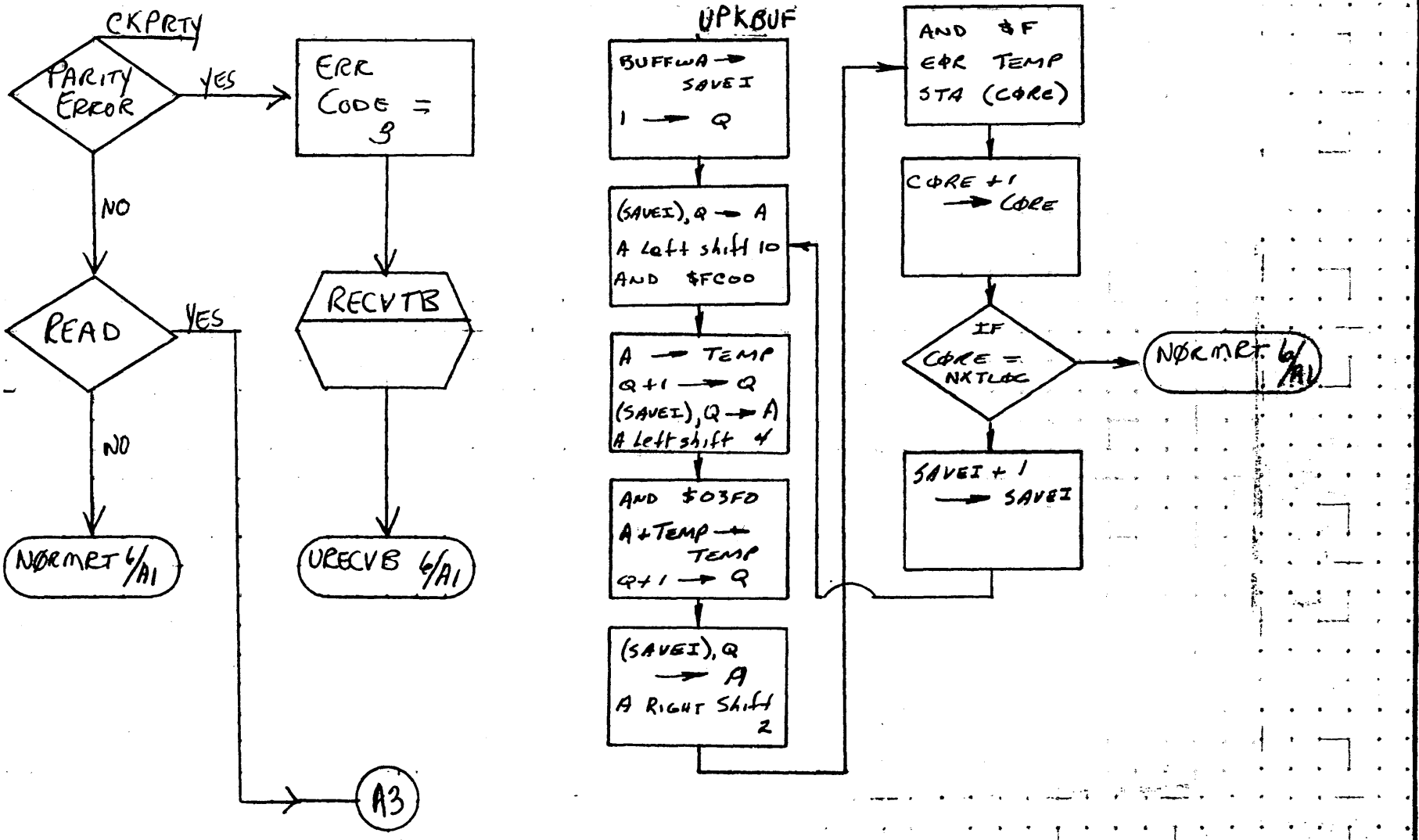
REV	APPROVED	DATE

MAR 5 1971

53.72

AA1385 (FORMERLY CA127-1)

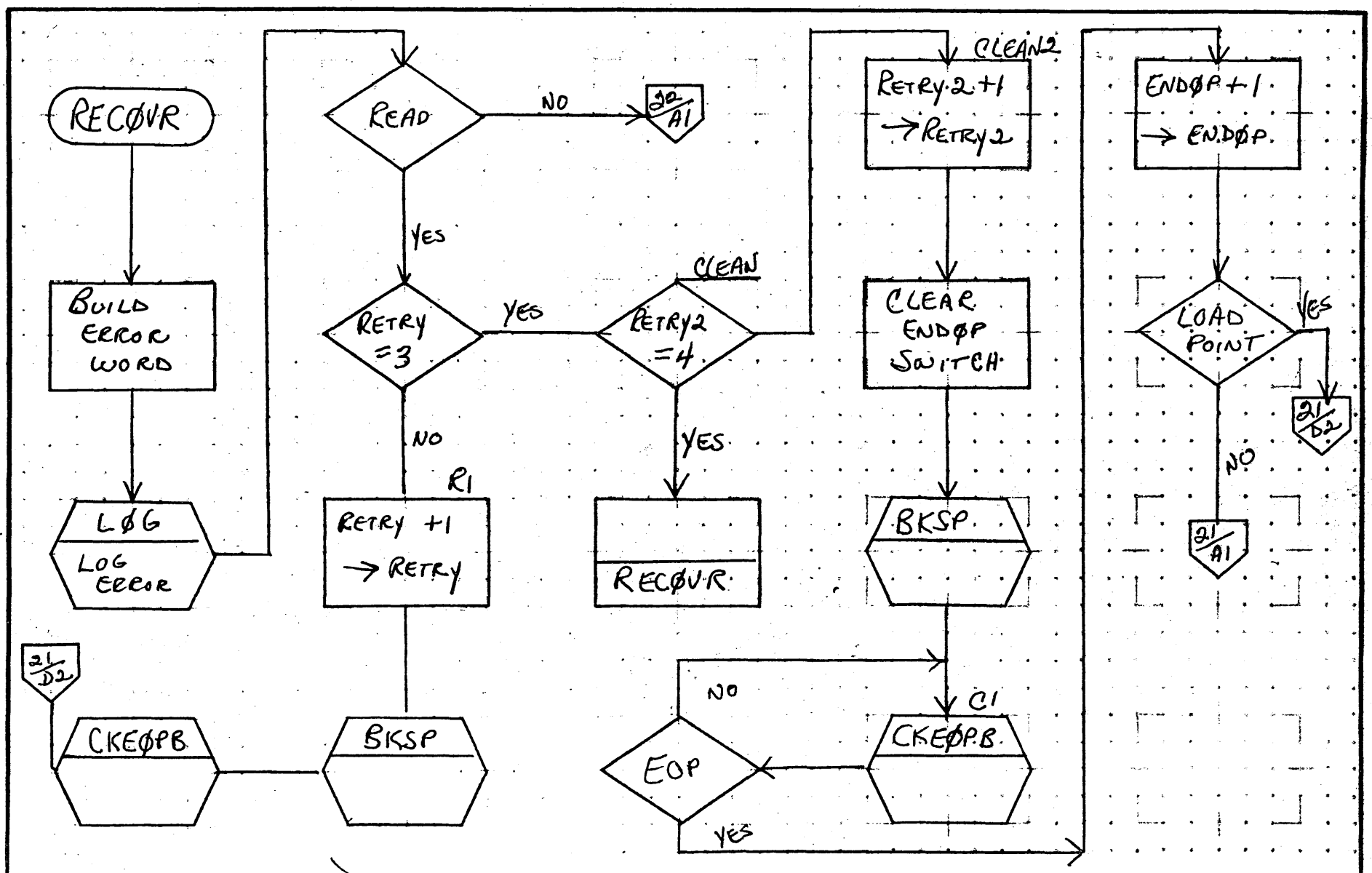
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	RWBAB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 19 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.73



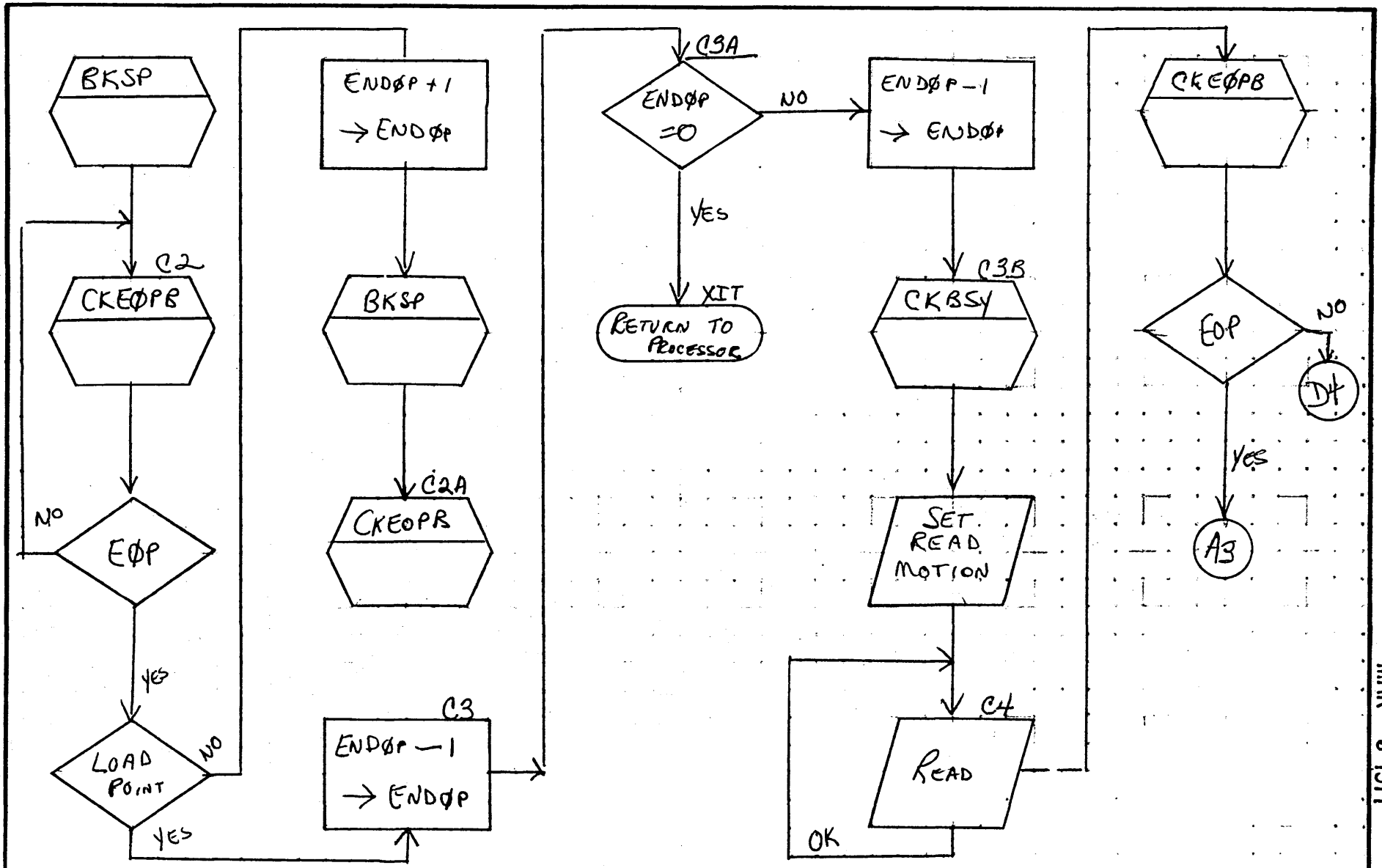
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	RECVTB	REV	APPROVED	DATE
	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
	BUFFERED	PAGE 20 OF 22	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

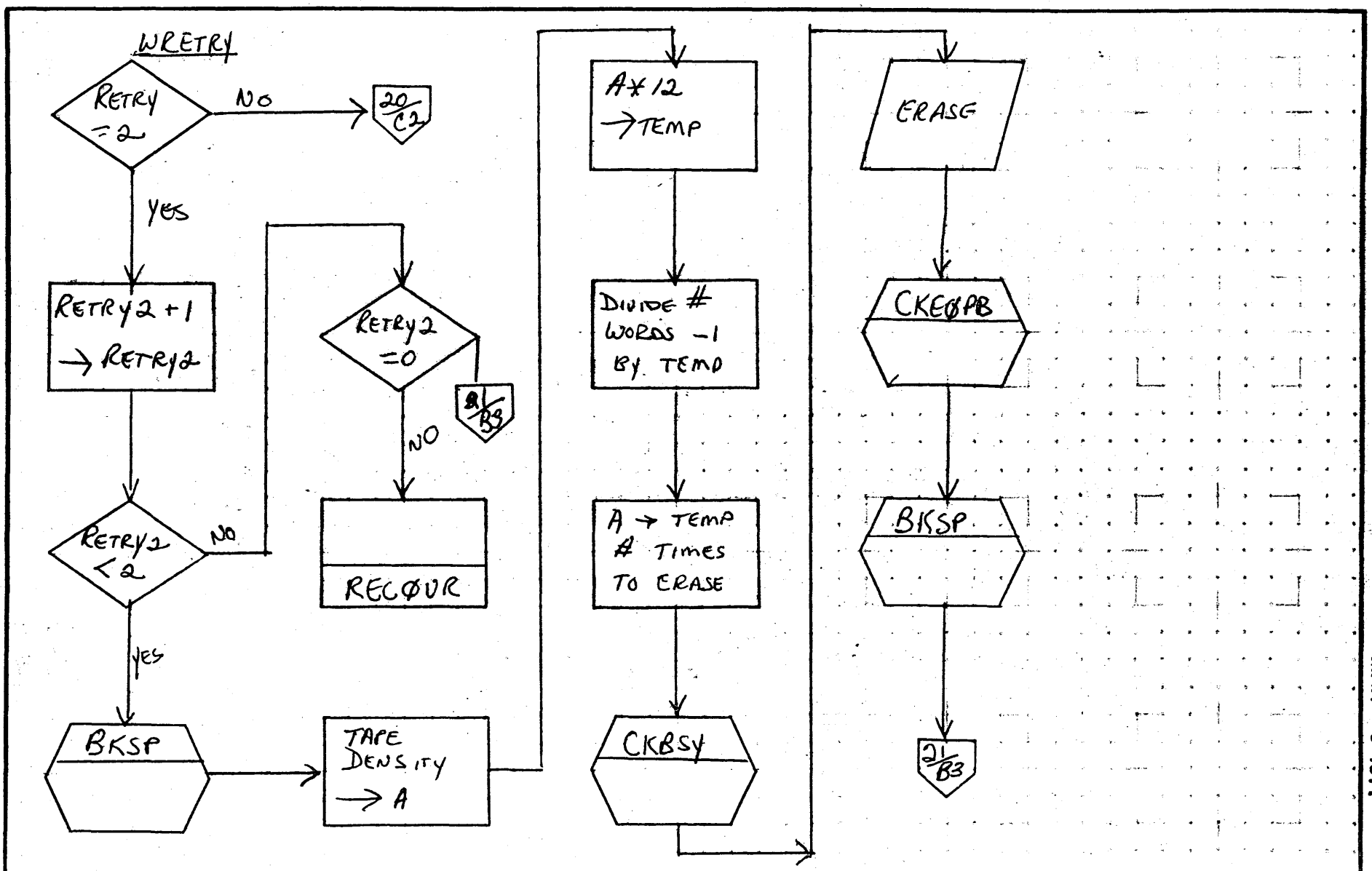
53.74



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	RECVTB	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1731/601 DRIVER		PROJECT MGR.			
FLOWCHART ☐	BUFFERED	PAGE 21 OF 22	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

53.75



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	RECVTB	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 22 of 22	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

53 of 76

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

54.0 PROGRAM NAME

1732-608/609 Magnetic Tape Driver (DR1732)

54.1 FUNCTION

DR1732 initiates, monitors and controls all operations on any combination of up to eight 608 and/or 609 magnetic tape drivers. These operations are the end result of monitor requests on magnetic tape drive logical units with a request code defined as follows:

1	READ
2	WRITE
4	FREAD
6	FWRITE
14	MOTION

DR1732 utilizes those features of the 1732 Controller which will promote maximum throughput. Non-stop mode of operations on reads, writes and tape motion requests are performed when appropriate.

54.2 ENTRY POINT NAMES

The following entry points are declared if the driver is core resident.

TAPINT	driver initiator
TAPCON	driver continuator
TPHANG	driver I/O hangup

There are no entry points declared if the driver is mass memory resident. The above entry points are then declared in the core resident control module, DBLDRV.

54.3 EXTERNALS and DESCRIPTION

ALTDEV	Alternate Device Handler
LOG	Log errors on Engineer File
MAKEQ	Set error bits in Q

When the driver is mass memory resident, the following external names are also declared:

BUFALC	Checks the ready/busy status of the 1706 Buffered Data Channel.
MAS300	Allow other drivers to use allocatable core if DR1732 is not busy.
ADRINT	Core location in DBLDRV that the driver stores the address of the Initiator.

ADRCON Core location in DBLDRV that the driver stores the address of the Continuator.

ADRHNG Core location in DBLDRV that the driver stores the address of the I/O Hangup routine.

54.4 ENTRY ENTERFACES

54.4.1 When a magnetic tape request is executed, DR1732 is entered via TAPINT. The driver is initialized, validity of the request checked, equipment availability is checked and reserved. The request is then initiated.

54.4.2 When interrupts are received from the 1732 magnetic tape controller, DR1732 is entered through TAPCON. The interrupts are checked to determine expectancy. If the interrupt was expected, it is acknowledged, if it was unexpected, the Ghost Interrupt message is sent to the standard comment device.

54.4.3 Upon I/O or motion control difficulty, DR1732 is entered at TPHANG. An error code zero is set and control is given to FAILED.

54.5 EXTERNAL INTERFACES

54.5.1 When difficulty has been encountered with a physical tape device, a call is made to ALTDEV {Alternate Device Handler} to determine whether an alternate device has been assigned.

54.5.2 When errors are encountered, control is passed to LOG to record the difficulty on the engineer file.

54.5.3 Error status is set in the upper three bits of Q via MAKEQ.

When the driver is mass memory resident, additional externals are declared as follows.

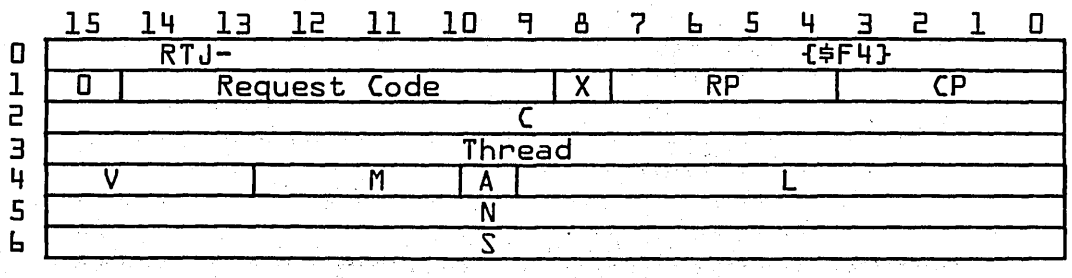
54.5.4 The BUFALC external routine checks the ready/busy status of the 1706 Buffered Data Channel and if not busy, a check is made for any stacked requests waiting for its use.

54.5.5 If DR1732 has completed, MAS300 is scheduled to allow other drivers to use its allocatable core.

54.5.6 During initialization of DR1732 the address of TAPINT, TAPCON and TPHANG are set into DBLDRV at core locations ADRINT, ADRCON and ADRHNG respectively.

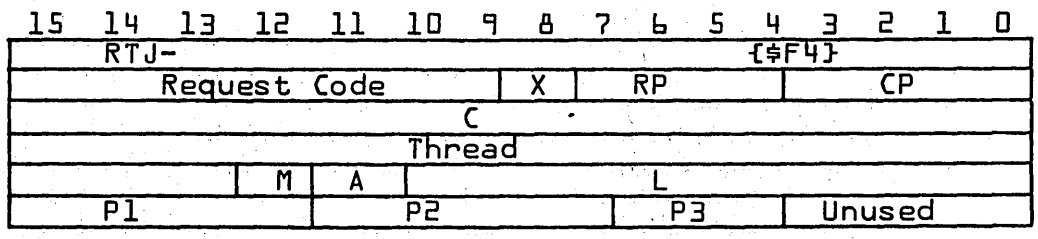
54.6 GENERAL PROGRAM INFORMATION

54.6.1 DR1732 is initiated by monitor calls which requests magnetic tape operations. The following is an example of a READ, WRITE, FREAD or FWRITE request.



Request Code	1, 2, 4, 6
RP	Request Priotity
CP	Completion Priority
X	Used to determine C, N and S
C	Completion Address
V	Error field
M	Mode, Binary or ASCII
A	Determines L
L	Logical Unit
N	Number of words to transfer
S	Starting address

54.6.2 The calling sequence for a tape motion request is defined as follows:



Request Code	14
RP	Request Priority
CP	Completion Priority
X	Used to determine actual completion address
C	Completion Address
M	Mode, Binary or ASCII
A	Determines L
L	Logical unit number
P1, P2, P3	Tape motion codes

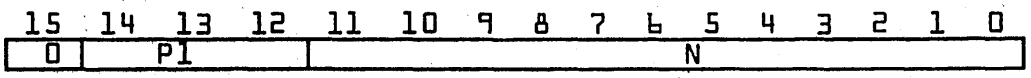
P codes are scanned from left to right and are defined as follows:

0 (zero)	Do nothing or request termination.
1	Backspace record
2	Write file mark
3	Rewind
4	Rewind and unload, terminates request
5	Skip file forward
6	Skip file backward
7	Advance record--an added software capability

The following binary values specify the DENSITY parameters:

3	Select 200 BPI
2	Select 556 BPI
1	Select 800 BPI
0	Do nothing

54.6.3 For repeated motion requests (up to 4095) and select density, the fifth word of the motion requests parameter string appears as follows:



P1	Motion or density code
N	Number of repeated requests

Interrupts are used on tape motion requests. The first zero parameter in P1-P3 or expiration of the repetition factor completes the request.

54.6.4 Appropriate entries must be made in LOCORE, LOG1A, LOG1, LOG2, MASKT, DGNTAB and PHYSTB. An example of the Physical Device Table (PHYSTB) for a 609 tape drive follows:

WORD	EXT	TAPINT, TAPCON, TPHANG	
0 TP0	NUM	$120X	X = Driver priority level
1	ADC	TAPINT	Initiator
2	ADC	TAPCON	Continuator
3	ADC	TPHANG	I/O Hangup
4	NUM	-0	Diagnostic Clock
5	NUM	0	Logical unit number
6	NUM	0	Parameter list pointer
7	NUM	X	X = Hardware address
8	NUM	X	X = Equipment class Equipment type Type of operations
9	NUM	0	Status word one
10	NUM	0	Core storage address
11	NUM	0	Last word address +1
12	NUM	0	Status word 2

13	NUM	X	X = Mode of operation Unit number
14	ADC	X	X = Address of next PHYSTB
15	NUM	0	Temporary storage
16	NUM	0	Temporary storage

54.6.5 Program dependant flags and values are as follows:

BUSY A flag when non-zero indicates the driver is currently processing a request.

LSTPT The same location as BUSY. The address of the Physical Device Table which is currently being processed.

ERRCNT An accumulator which contains the error count if any. It is used to determine which recovery procedure is to be used.

ALUCC Contains the address of the allocatable buffer when the buffer is present. Zero when not present.

54.6.6 DR1732 is coded as a MACRO skeleton. To parameterize the driver for a particular configuration, the user will prepare a card defining the MACRO parameters. This card is to be inserted in the source deck of DR1732 before the END card and the deck assembled. It should be pointed out that the assembly listing will provide only the machine code unless the M option is specified.

The MACRO call card appears as follows:

TPDRGN P1,P2,P3,P4,P5,P6,P7

Where the formal parameters P1 through P7 may have the following definitions.

P1 Defines the residency of the driver--CORE for core resident or MASS for mass memory resident. If P1 is equal to MASS, the core resident module, TAPCOR, must be loaded under the *L portion of the load and DR1732 loaded under the system name TAPMAS.

P2 Defines the method of data transfer--BUF for buffered transfers using a 1706 Buffered Data Channel, or UNBUF for unbuffered transfers using the AQ channel.

P3 Defines the type of tape drives which will be in the system. P3 will be 608, 609, or BOTH.

P4 Defines the type of Read/Write requests to be processed. P4 will be FORM when only formatted requests are to be processed, REG for regular requests only, or BOTH if formatted and regular requests are processed.

P5 Defines whether error recovery for parity errors will be attempted. If P5 is ERR, recovery is attempted. If NOERR is specified, the error bits are set and the request is completed.

P6 Defines the maximum tape record size for 608 units. If blank, 96 words will be assumed.

P7 Defines the priority level at which the driver is to operate, and should be in the range 5-14.

54.7 DR1732 GENERAL DESIGN

54.7.1 DR1732 having the capability of being a mass memory resident driver is coded in run anywhere code. When the driver does reside as a mass memory resident routine, sufficient allocatable core must be available in which the driver may reside or a hangup will occur. The driver, either core or mass memory resident makes a space request for its buffer.

54.7.2 The driver is not reentrant. When the initiator portion of the driver is entered, a checked is performed to determine whether a request is being processed (driver is busy) and if busy, an exit is taken to the DISPATCHER.

54.8 GENERAL PROGRAM LOGIC

54.8.1 When a request is made to perform a function on 608/609 magnetic tapes, the initiator portion of the driver is entered. The Physical Device Table (PHYSTB) address of the present PHYSTB is contained in the Q register. The priority level is set in all driver calls and if mass memory resident, the addresses of TAPINT, TAPCON and TPHANG are placed in the appropriate locations in DBLDRV. If the driver is busy an exit is made to the DISPATCHER. When the driver is not busy, the error count (ERRCNT) is initialized and unit rewind is determined. When the unit is rewinding, a check is made for the next PHYSTB, otherwise a check is made to determine if the unit has completed rewinding. If it has completed rewinding, the next motion code is picked up and an exit is made to MOTION+2. If still rewinding, FNR is entered

to find the next request. Upon return from FNR, a check is made to see if the 1706 Buffered Controller is busy only when the buffered version of the driver is being used. The request code is checked, if it is a motion request, control is given to MOTION, if not motion, control is given to XFER for data transfer. The data transfer is initiated by connecting to the designated unit, selecting the mode and type of operation. If it is a write binary operation on a 609 magnetic tape, the binary data is not unpacked. When binary data is written to a 608 type unit, all data is unpacked to two six bit characters per word. Upon returning from initiating the WRITE request, an exit is made to the DISPATCHER.

A READ request is initialized the same as a WRITE only there is no unpacking of data regardless of the type of unit. The PACK routine is entered after the first READ has been completed and the Continuator portion of the driver is entered. If any faults are discovered in the request, the appropriate code is picked up in the A register and control is given to the FAILED routine.

54.8.2 When the end of operation interrupt is received, the continuator {TAPCON} portion of the dtiver is entered with the address of the TPO labeled PHYSTB in Q. When the driver is not expecting an interrupt {no interrupts pending} a Ghost Interrupt {GI} message is sent to the standard comments device. Expected interrupts cause status to be taken on the functioning unit. If the unit was executing a motion request, an exit is taken to NXTMOT to check for and process the next motion request. When the request is not a motion request, parity is checked on the data transfer just completed. When parity is present an exit is made to the recovery {54.8.3} portion of the driver and recovery is attempted. When the operation completes without error, the type unit is determined, seven track or nine track. If the unit was a 608 {seven track} device, continue at 1.1 else update the first word address of the buffer. {1.1} It is determined if a read or write was last performed and if a read, a check is made for the file mark. If a write was being performed continue at 1.2. If the end of file is present, control is given to CMPRD where status bits are set in Q via MAKEQ and an exit is made via COMPRQ {complete request processor}. If the file mark was not present, and binary information was read, the data is packed into the users buffer. If ASCII data was transferred, conversion from BCD to ASCII is made. {1.2} The completion

bit is checked, if set, an exit is made to the complete request processor (COMPRQ) to perform the housekeeping for the driver. Upon return, control is given to FIND to get the next request. If the completion bit is not set, the status bits are set in Q and the preceding sequence follows via the complete request processor.

54.8.3 The recovery portion (RECVR) is entered when a parity error has been detected. A check is made for parity caused by an end of file mark. If end of file, an exit is taken to READCK to allow the completion request process previously described to be executed. If no end of file status is present, the following sequence is performed to try recovering from the parity error. The error is logged on to the engineer file via LOG. If a read was in operation when the error occurred, a backspace is done followed by a read and error check. If the error persists, the preceding sequence is performed three more times. If the error still exists three consecutive backspaces are performed checking for loadpoint after each backspace has completed. This allows the error portion of the tape to pass across the tape cleaner portion of the heads. Execute up to three reads depending upon load point and check for read error on last read. If the error remains, perform the preceding sequence four times prior to declaring the error irrecoverable and exiting to the FAILED routine. The FAILED routine sets error flags and makes a call to the alternate device handler (ALTDEV) and upon return the next PHYSTB is determined via NXTPT. The write parity errors follow the succeeding sequence. The write is performed three times detecting for the error after each. This procedure is tried three consecutive times and if the error persists, the preceding FAILED sequence is executed.

54.8.4 The I.O hangup (TPHANG) section is entered when the driver fails to reveive an interrupt from an initiated operation. The Q register contains the address of the physical device table that has failed to interrupt. The error code is set in the A register and control is given to the FAILED routine. The FAILED routine executes the same as previously explained.

54.9 SUBROUTINE LOGIC

In the following description, the labels used in the driver are placed to the left of the description to enhance an easy relationship directly to the listing and flowchart.

TAPINT Places the driver priority level into all requests made be driver and set the entry point adresses of TAPRINT, TAPCON and TPHANG in there respective locations in DBLDRV. The entry points are placed into DBLDRV only when the driver is mass memory resident. A check is made to determine if the driver is busy and if so, an exit is taken to the DISPATCHER or else control is given to label NOTBSY.

NOTBSY Zero the error counter, ERRCNT, and set the record of the last Physical Device Table, LSTPT, equal to the current table address.

TSTRWD Test the rewind bit (bit 6 of ESTAT1) to determine if the unit is presently rewinding. If rewinding, go to NXTPT or else go to NORWD.

NORWD If the motion bit (bit 5 of ESTAT1) is up, a rewind has just completed. Decrement the rewind counter, RW, update the motion control code word, ETEMP2, and go to MOTION+2.

FIND Go to Find Next Request routine. If there are no requests waiting on this unit, go to NXTPT or else save the current PHYSTB address in LSTPT and place the PHYSTB address into the parameter list of BUFALC. Put the absolute address of FAILED into the BUFALC parameter list. Give control to BUFALC to determine if the 1706 Buffered Data Channel is busy. If the 1706 is busy, go to FAILED routine or else extract the request code from the user parameter list and go to MOTION if a motion request or else go to XFER.

NXTPT Pick up the address of the next PHYSTB from ELINK of the present PHYSTB. If a new PHYSTB is not present, go to EXIT else check to determine whether an error is pending on the newly acquired PHYSTB. If an error is pending, do not remove the PHYSTB from the string until the error has been processed so go to NXTPT. If the PHYSTB is error free, go to TSTRWD.

EXIT Set the driver not busy and check if a rewind is in progress. When a rewind is in progress, the in core flag (ADRINT) is not cleared. The buffer, if present, is released and the buffer FWA is set to zero. The MAS300 routine residing

in DBLDRV is scheduled to allow other drivers the use of the 1706. BUFALC in DBLDRV is entered to determine the status of the 1706 and whether a request is on the BUFALC queue. If the 1706 rejects the status request, the exit is made to the DISPATCHER from BUFALC. Return is to P+3 of the BUFALC call when a request is found on the queue and the 1706 is not busy.

XFER The unit is selected, through UNTSEL, and bit 14 of ESTAT2 is tested to determine which type drive is connected. Go to M609 or M608, depending on type.

M609 Initialize the internal control addresses, FWA and LWA. Set the parity code to odd, set the motion index up for read or write, and go to IOXFER.

M608 Acquire allocatable core if needed and set up control addresses, FWA and LWA according to mode of transfer, ASC or binary. If this is a write request, repack the binary information for assembly mode, or convert the ASC code to external BCD, and go to IOXFER.

{buffered}
IOXFER Set the unit's parity through UNTSEL. If this is a write request, check for the presence of the write ring by RINGCK. The appropriate tape motion is started and the buffer transfer initiated. The diagnostic cleck, EDCLK, is set up and an exit is taken to Dispatcher to await the End of Operation interrupt.

{unbuffered}
IOXFER The parity code is set up and a write ring check made if this is a write request. The tape motion is then started. The data transfer of the record is then accomplished using the A@ channels. EDCLK is updated and an exit taken to the Dispatcher to await the EOP interrupt.

ASCBCD This is a subroutine which does the setup and interface with subroutine A2BCD, which performs the ASC to BCD conversions.

UNPK The unpacking of binary data is necessary for the assembly mode of the 608 magnetic tape drives. The format of unpacked data is as follows:

15	14 13 — 8	7	6 5 — 0
0	15 - 10_1	0	9 - 4_1
0	3-0., 15-14_2	0	13 - 8_2
0	7 - 2_2	0	1-0_2, 15-12_3
0	11 - 6_3	0	5 - 0_3
ETC.			

IN — A subroutine which performs hardware input instructions and monitors reject conditions.

UNTSEL — This subroutine combines the code in the A register with the select code, UNTMOD, performs the select function through the subroutine, OUT. The subroutine, STATUS, is also called.

STATUS — This subroutine reads both hardware status words, through the subroutine IN, and combines them into the Physical Device Table word ESTAT2.

TAPCON — This is the entry point for interrupts from the 1732 magnetic tape controller. The buffer is terminated, the diagnostic clock is reset and the interrupt is acknowledged. If no interrupts were expected, a monitor request is made to print the Ghost Interrupt message (GI MT) on the standard comment device.

EXPECT — The interrupt was expect so the diagnostic clock is reset and the STATUS subroutine is entered to status the device. The Ready Status bit is checked and if it is down, a failure code of 6 is passed to FAILED.

READY — If the motion control bit (Bit five of ESTAT1) is up, control is passed to NXTMOT. If not up, go to NOM01.

NOM01 — If the parity error status is set, control goes to RECVR otherwise go to NOPAR.

NOPAR — The device status is examined to determine the type of device being used. If the unit is not a 608, the first word address is updated before checking the type of I/O being performed. Control is given to READC if a read was performed or WRITEC if a write was performed.

READC — The EOF status is checked. If present, the EOF bit (Bit 14 of EREQST) is set and control goes to CMPRD. If not present, go to NOFILE.

NOFILE — If 608 device, go to CKMOD else go to WRITEC.

CKMOD — If the mode of operation is BCD, go to BCDASC or if the mode is Binary go to PACK.

BCDASC A subroutine which performs setup for and interfaces with the subroutine BCD2A, which performs the BCD to ASCII conversions.

WRITEC The completion bit and the FWA/LWA is checked for completion. If not complete, go to XFER else give control to COMPL.

COMPL If the function is either a write or motion request, a call is made to the Complete Request Processor (COMPRQ) prior to going to FIND. If the function was a read, MAKEQ is entered prior to going to COMPRQ.

PACK The binary data that was unpacked to assembly mode is repacked to full binary words.

RECVR When RECVR is entered, a check is made to determine if the EOF caused the parity. If the EOF caused the parity, the completion bit is set (bit 14 of word eight) and control is given to COMPL. If not EOF, the error is logged on the engineer file via LOG and the error counter, ERRCNT, is used as a jump index into the recovery routine table, RCVRTB, to enter the correct routine. ERRCNT is then tallied.

FALPAR Error code 3 is passed to FAILED to indicate a parity error.

RECVR The error counter, ERRCNT, is used as a jump index into the table, RCVRTB and ERRCNT is tallied.

FALPAR The error code 3 is passed to FAILED to indicate a parity error.

BCKRD The subroutine XCTMOT is called to backspace one record and control is passed to IOXFER.

BCK3RD This routine attempts to backspace three records, checking for load point after each backspace. Then the tape is advanced the number of records which has been backspaced and the transfer is attempted again at IOXFER.

BCKWT This routine backspaces one record by XCTMOT and then passes control to IOXFER.

ERASE This routine writes a file mark and then backspaces one record, which amounts to an erase tape function.

TPHANG This is the entry point for I/O hangups. An error code of zero is passed to FAILED.

FAILED If a failure occurred during a rewind, the rewind counter, RW, is ducremented and a call is made to MAKEQ to set the error bits prior to setting the device failed and error bits in control word ESTAT1. Then the error code and logical unit number are formed as the passed parameter in a scheduler call to the Alternate Device Handler (ALTDEV).

MOTION The requestor's motion code word is retrieved from the calling sequence and the subroutine XCTMOT is entered. Upon return, control is passed to COMPL.

XCTMOT The motion code is saved in location ETEMP2, the motion control bit is set in ESTAT1, and the unit is selected by the subroutine UNTSEL. The next motion code, bits 14-12 of ETEMP2, is used as a jump index into the table SUBMOT.

MOTSTR This routine entered by all of the motion processors. The proper motion function is initiated through OUT. If an end of operation interrupt was requested, an exit is taken to the Dispatcher. If EOP was not requested, the motion was either a rewind or a rewind and unload. If an unload was requested, DONEM is entered. If a rewind was requested, control is passed to NXTPT.

DONEM The motion code is zero or an unload function has just been executed. Density select is performed of requested, the motion control bit is reset, and a return through XCTMOT is taken.

WRFM The code is 2. A write ring present check is performed, proper parity determined, and control is passed to MOTSTR.

REW The code is 3. A load point check is performed by LPCHK and if not present, the rewind control bit (bit 6 of ESTAT1) is set and the low level program REWCK is scheduled at level 4. The subroutine BUSYCK is called and upon return control is passed to MOTSTR.

BKSP The code is 1. The subroutine, LPCHK, is called and control is passed to MOTSTR.

REWUN The code is 4. The control word, ETEMP2, is zeroed to terminate the motion request. The subroutine BUSYCK is called and control is passed to MOTSTR.

FLFR The code is 5. The proper parity is determined and control is passed to MOTSTR.

FLBK The code is 6. The subroutine, LPCHK, is called, proper parity is determined, and control is passed to MOTSTR.

NXTMOT This routine is entered after an EOP interrupt and when the motion bit is set in ESTAT1. If more iterations of the current motion are to be performed the iterations are tallied down and control is passed to DOMOT, which re-defines the motion index for MOTSTR. If no iterations remain,

the motion control word ETEMP2 is updated for the next motion and control is passed to DOMOT.

BUSYCK This subroutine checks the status of magnetic tape system for a busy condition and returns if the status is negative. If Busy is up, the low priority program BSYWAT is scheduled at level 4 and an exit taken to the Dispatcher.

BSYWAT When the busy status goes negative, the location NOBUSY is scheduled at the driver level, which restores the I-register with the address of the Physical Device Table which is currently being processed.

RINGCK This subroutine checks for the presence of a write ring on the selected unit. If not present, an error code of 1 is passed to FAILED.

LPCHK A check for load point is performed. A normal return is executed if a negative condition exists. If load point status is up, the motion interation counter is zeroed and control passed to NXTMOT.

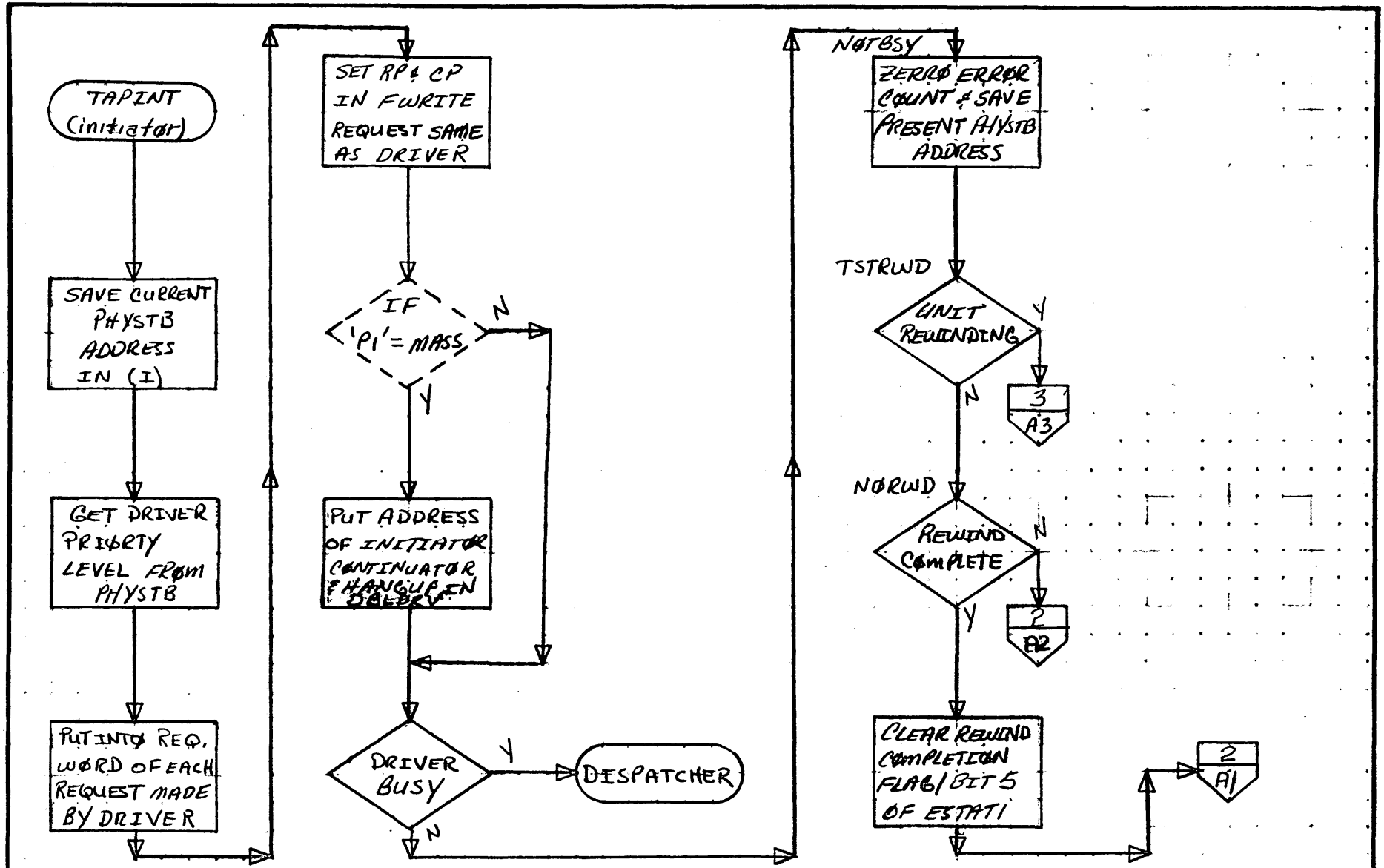
CKPRTY The proper parity is determined and selected for the file mark operations.

REWCK A rewind operation is in progress. This routine checks the status for the presence of load point and when it occurs, the rewind control bit is reset in ESTAT1 and the driver is scheduled. Upon and internal reject on status function, schedule up todrivers priority.

WAIT This routine monitors the controller active status at a low level for UNTSEL. When a not active condition occurs, the location CONSEL is scheduled at the priority of the driver.

A2BCD A2BCD converts ASCII codes (two ASCII characters per word, Bits 0-6, 8-14) to BCD codes (two BCD characters per word, Bits 0-5, 8-13). The original codes are replaced by the converted codes so the orginal ASCII buffer is modified to be a BCD buffer.

BCD2A BCD2A converts a buffer of BCD codes (two characters per work, 0-5, 8-13) to ASCII codes (two ASCII codes per word, Bits 0-6, 8-14). The original BCD buffer is modified to be an ASCII buffer.

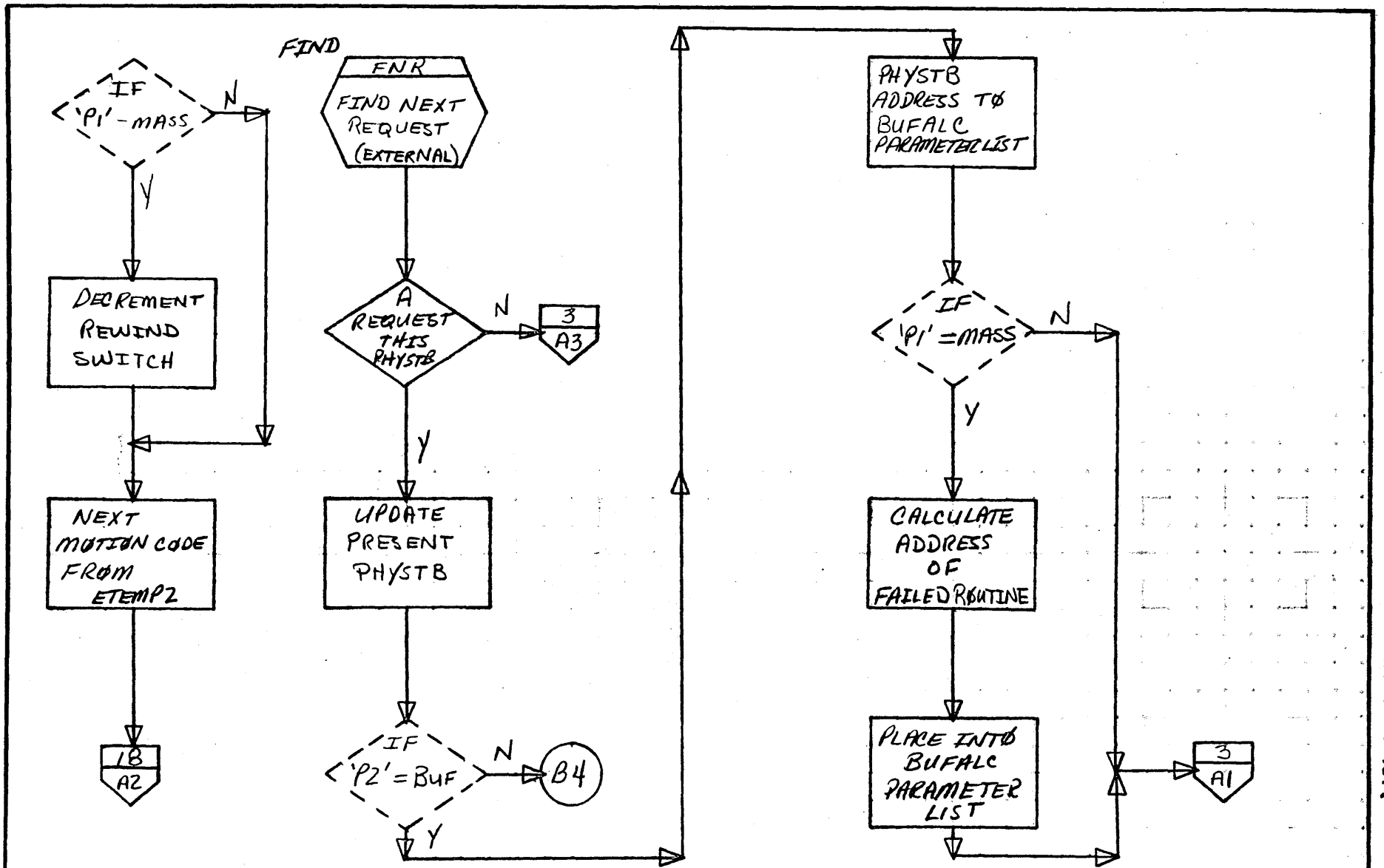


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1732/608-609		PROJECT MGR.			
FLOWCHART ☐	MAG. TAPE DRIVER	PAGE 1 OF 25	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



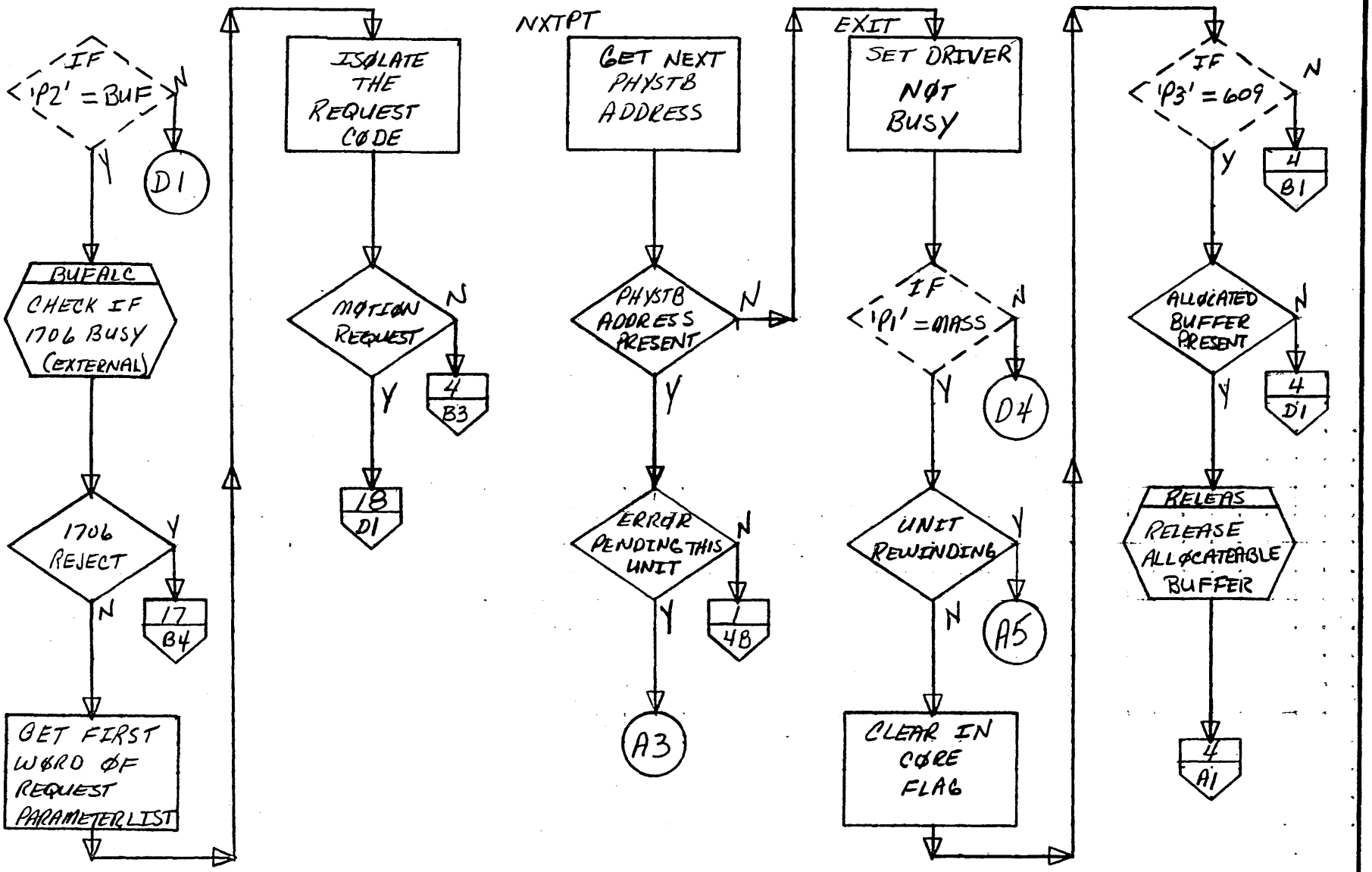
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1732/608-609 MAC.		PROJECT MGR.		
FLOWCHART ☐	TAPE DRIVER	PAGE 2 OF 25	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

54.16

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

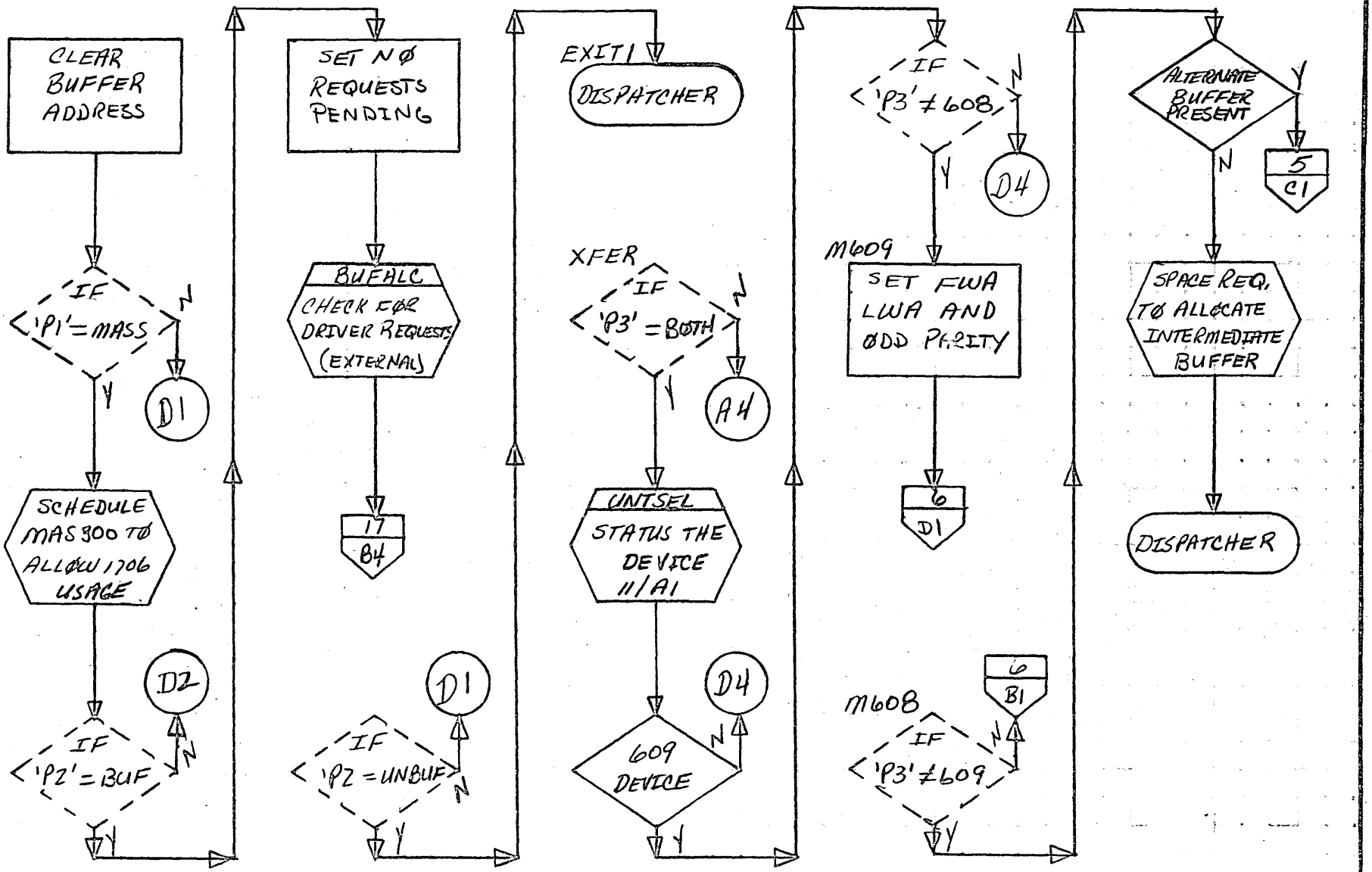


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: 1732/608-609 MAG. TAPE DRIVER PAGE 3 OF 25	PROJECT MGR.			
	NUMBER / ISSUE DATE	PROJECT NAME			
	DRAWN BY / DATE	TASK NO.			
		TASK NAME			

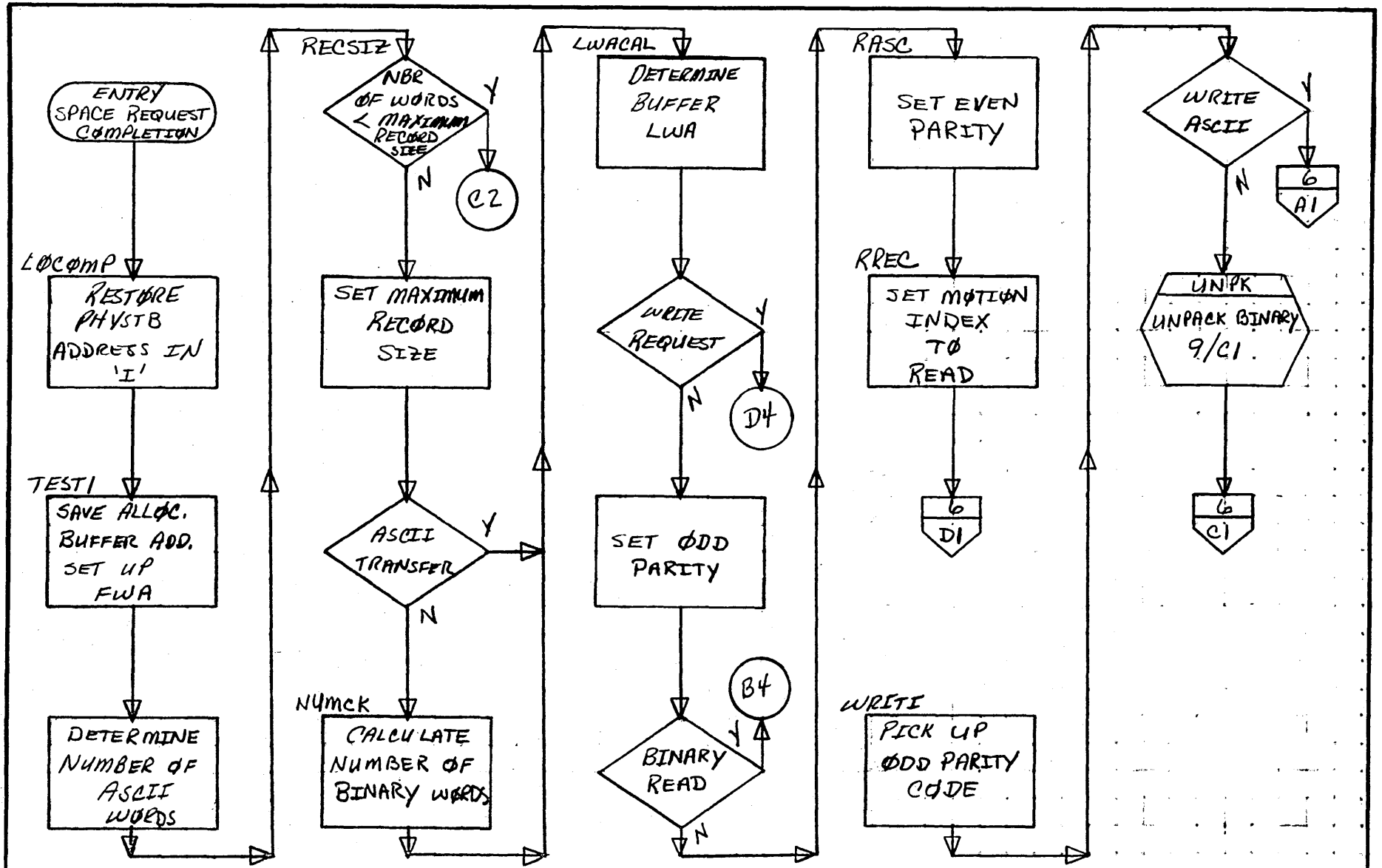
MAR 5 1971

54-17

A B C D



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS — MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: 1732/608-609 MAC.	PROJECT MGR.			
	TAPE DRIVER — PAGE 4 OF 25	PROJECT NAME			
	NUMBER — ISSUE DATE	TASK NO.			
	DRAWN BY — DATE	TASK NAME			



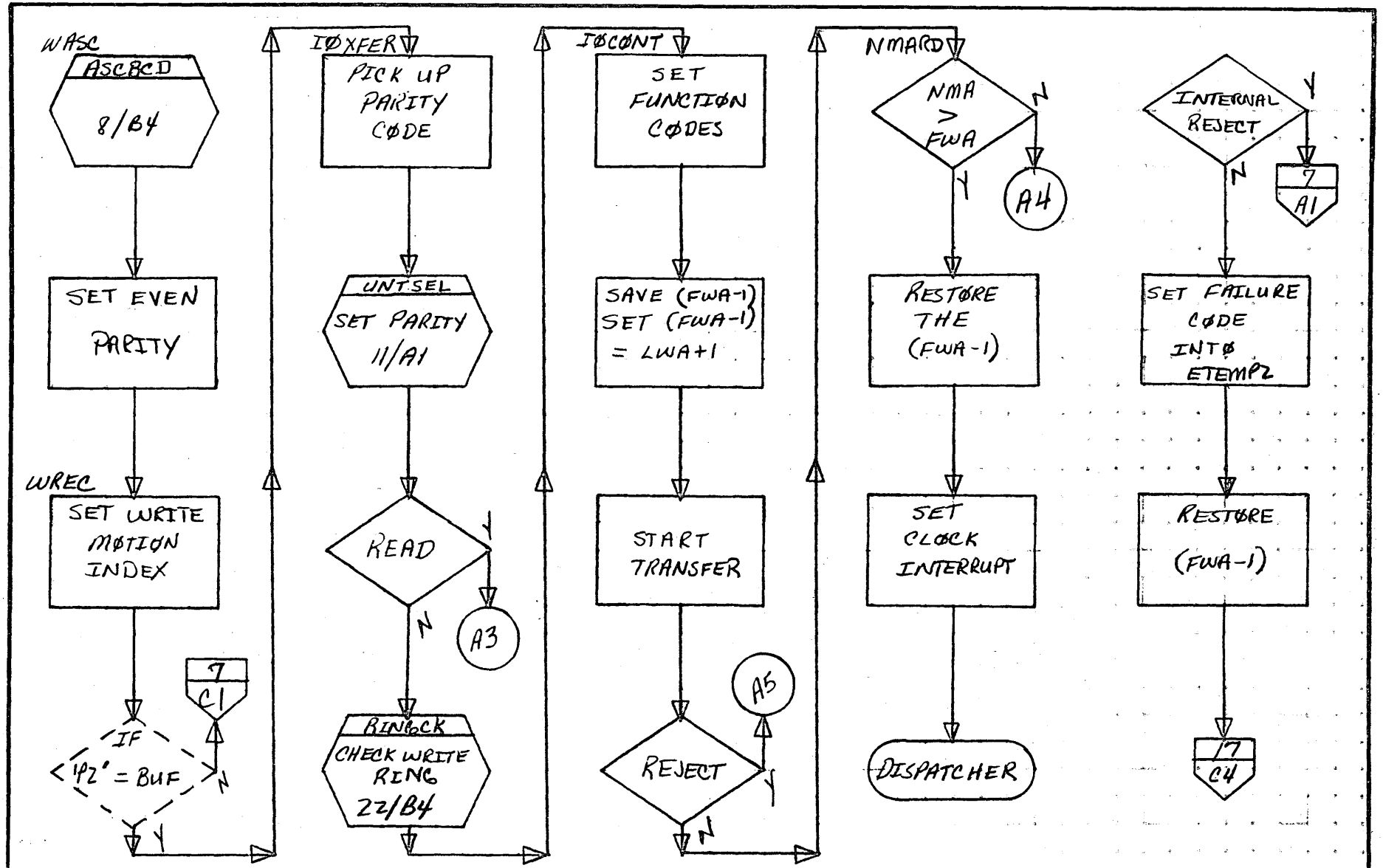
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAG. TAPE DRIVER	PAGE 5 OF 25	PROJECT MGR.			
	NUMBER	ISSUE DATE	PROJECT NAME			
	DRAWN BY	DATE	TASK NO.			
			TASK NAME			

AA1385 (FORMERLY CA127-1)

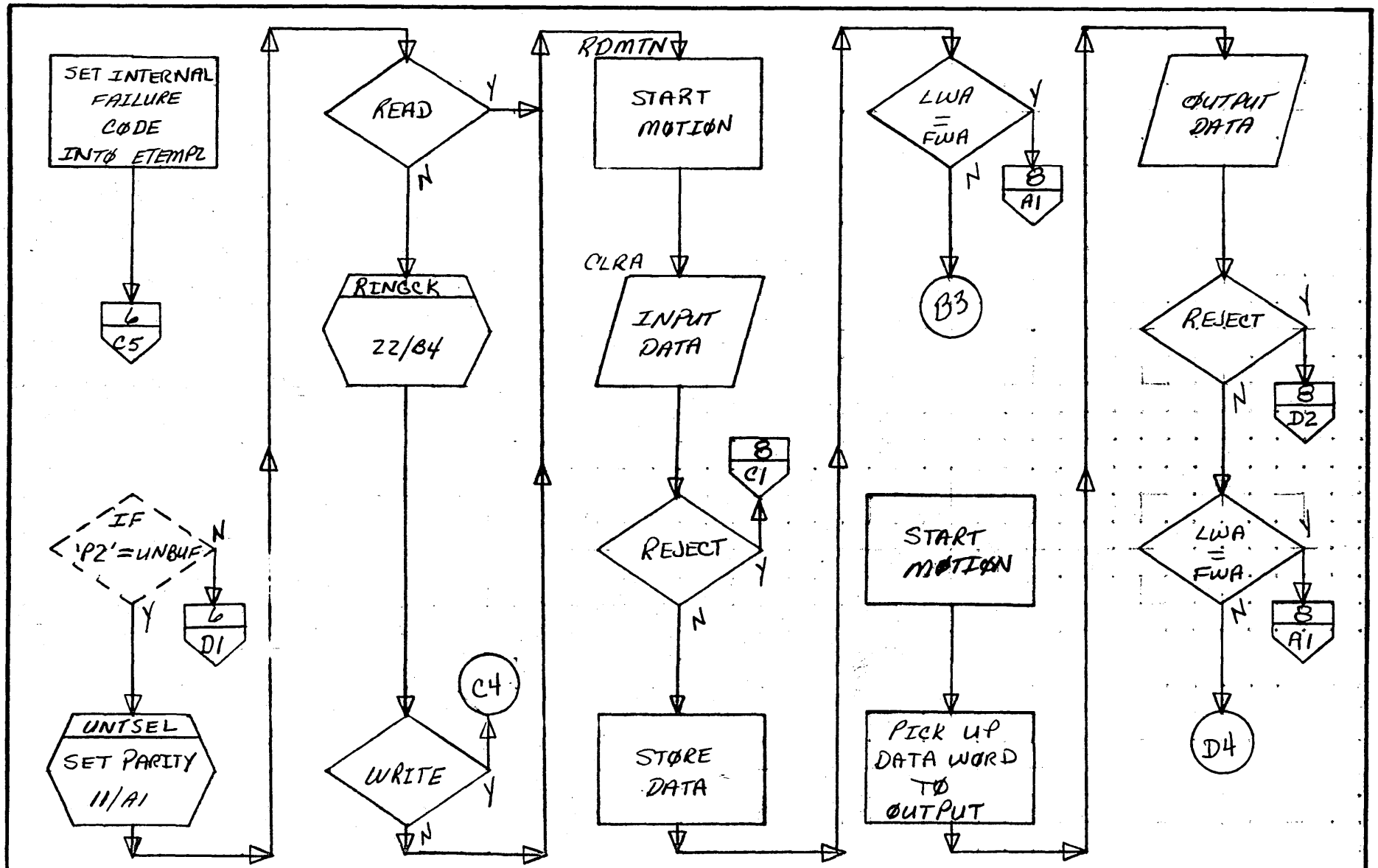
PRINTED IN U.S.A.

Mark 6 19/1

54.19



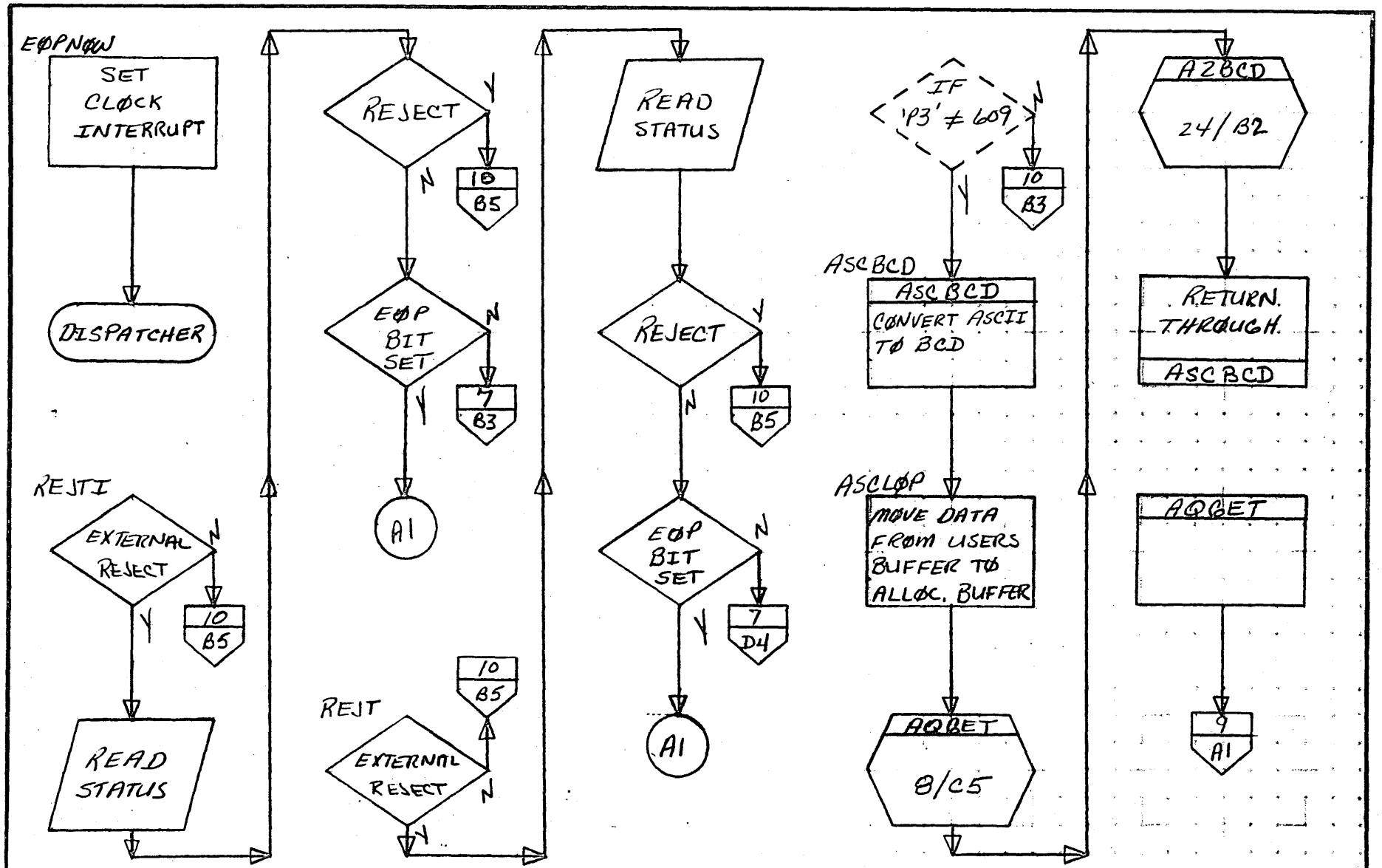
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAG.	PROJECT MGR.			
	TAPE DRIVER PAGE 6 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
FLOWCHART ☐	DOCUMENT TITLE 1732/608-609 MAG. TAPE DRIVER	PAGE 7 OF 25	PROJECT MGR.		
DECISION TABLE ☐			PROJECT NAME		
OTHER ☐	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

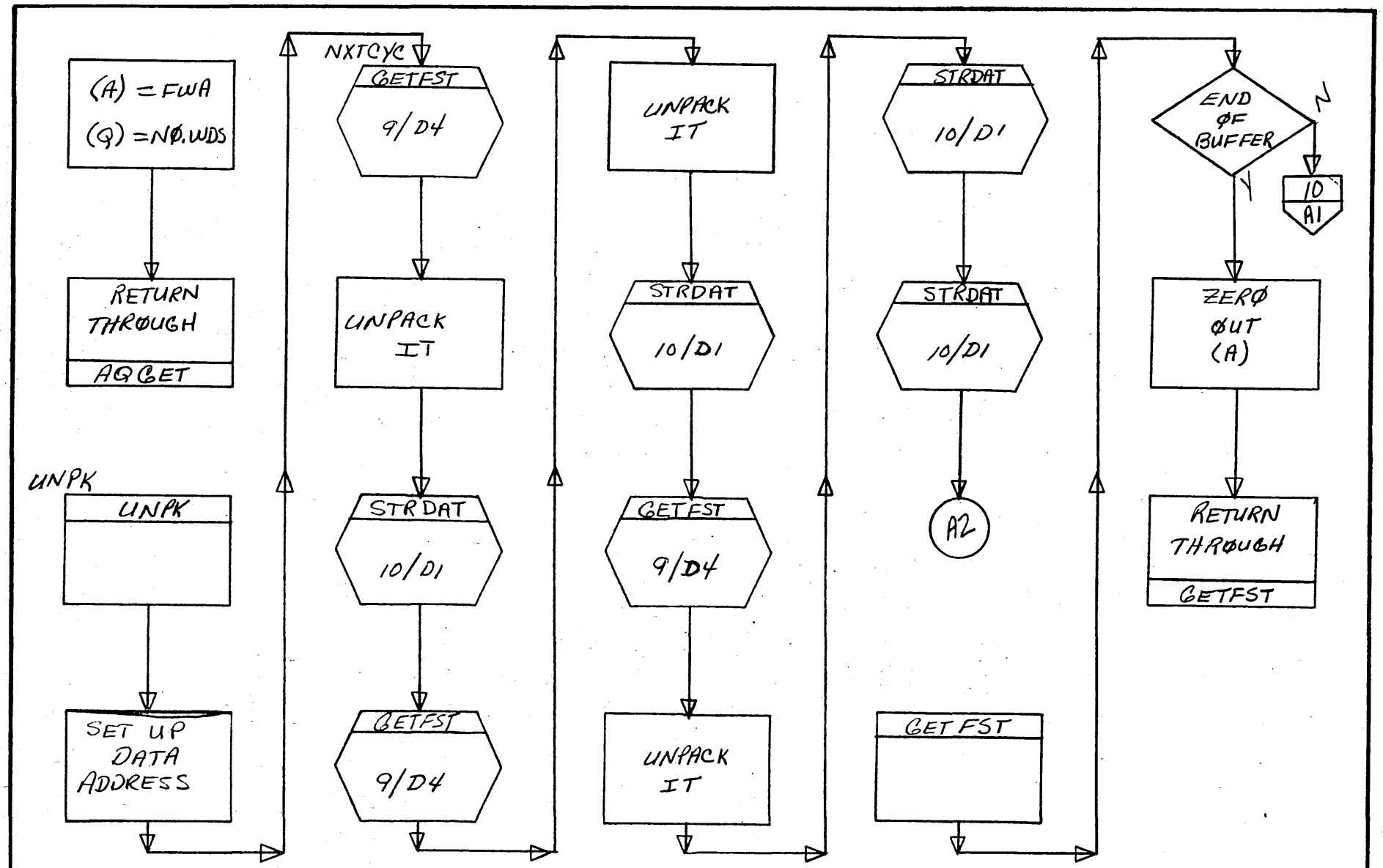
54.21



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAG. TAPE DRIVER PAGE 8 OF 25	PROJECT MGR.			
	NUMBER ISSUE DATE	PROJECT NAME			
	DRAWN BY DATE	TASK NO.			
		TASK NAME			

MAR 5 1971

54.22

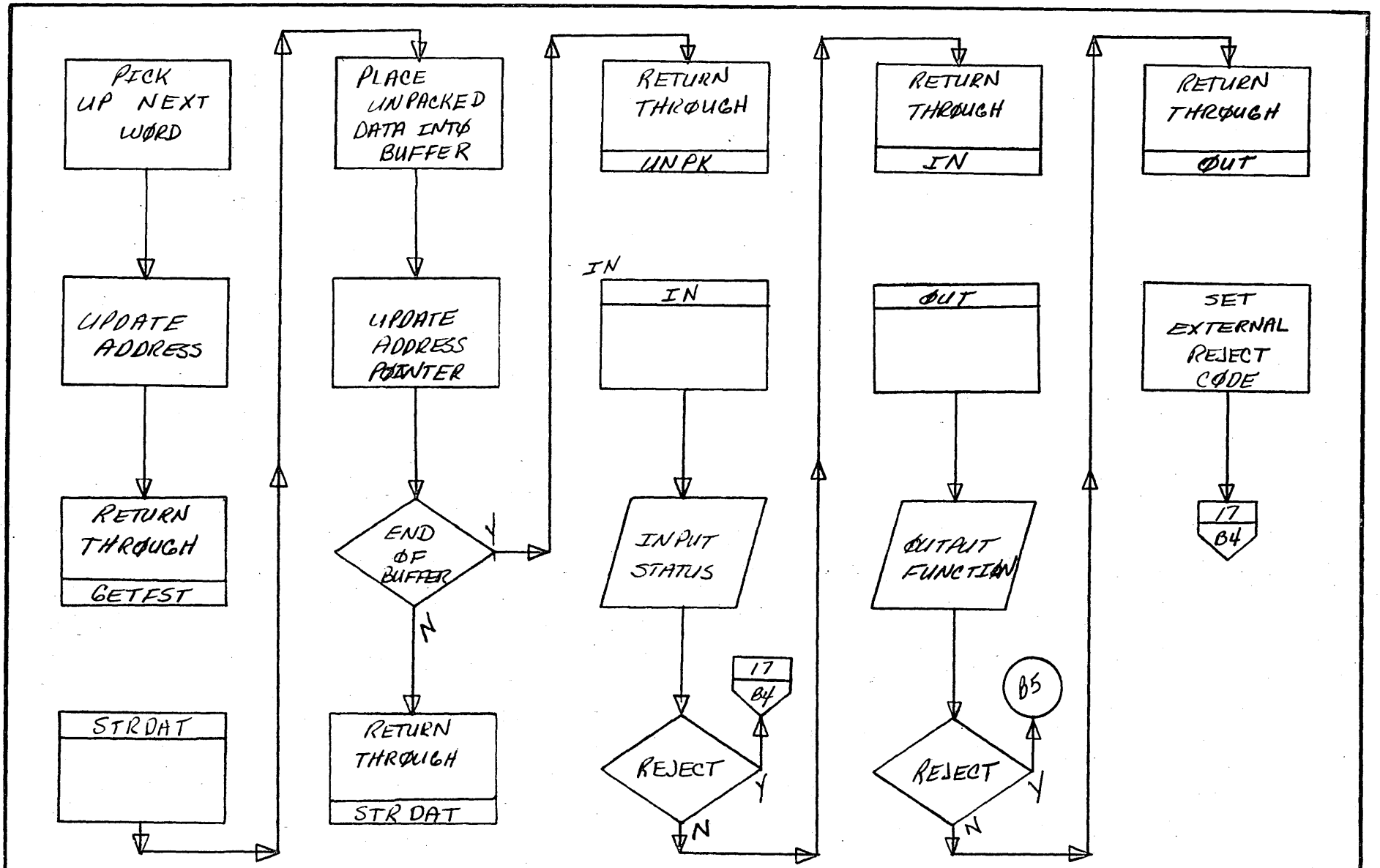


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAC.	PROJECT MGR.			
	TAPE DRIVER PAGE 9 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.23

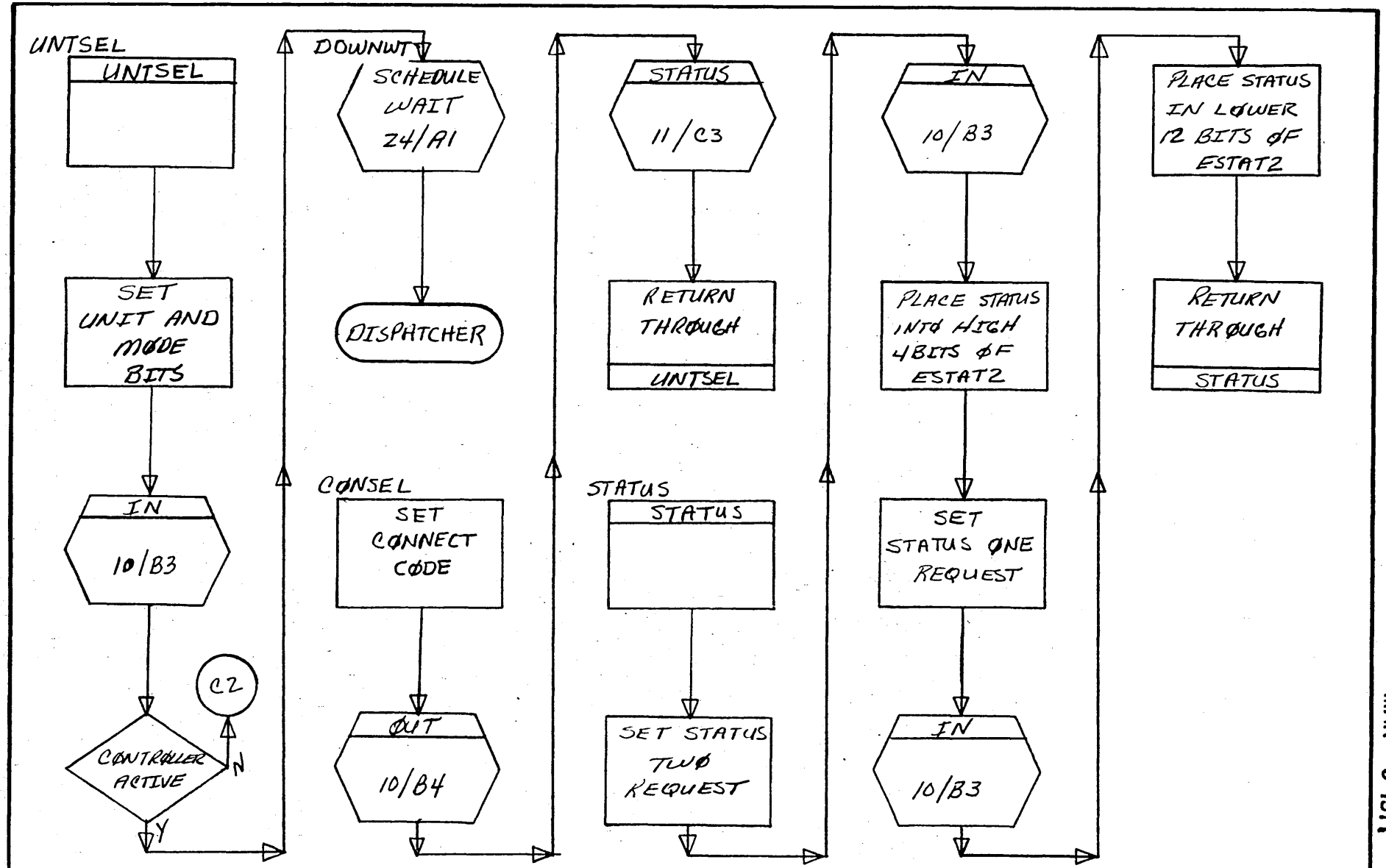


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1732/608-609 MAC.		PROJECT MGR.		
FLOWCHART ☐	TAPE DRIVER	PAGE 10 OF 25	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.24

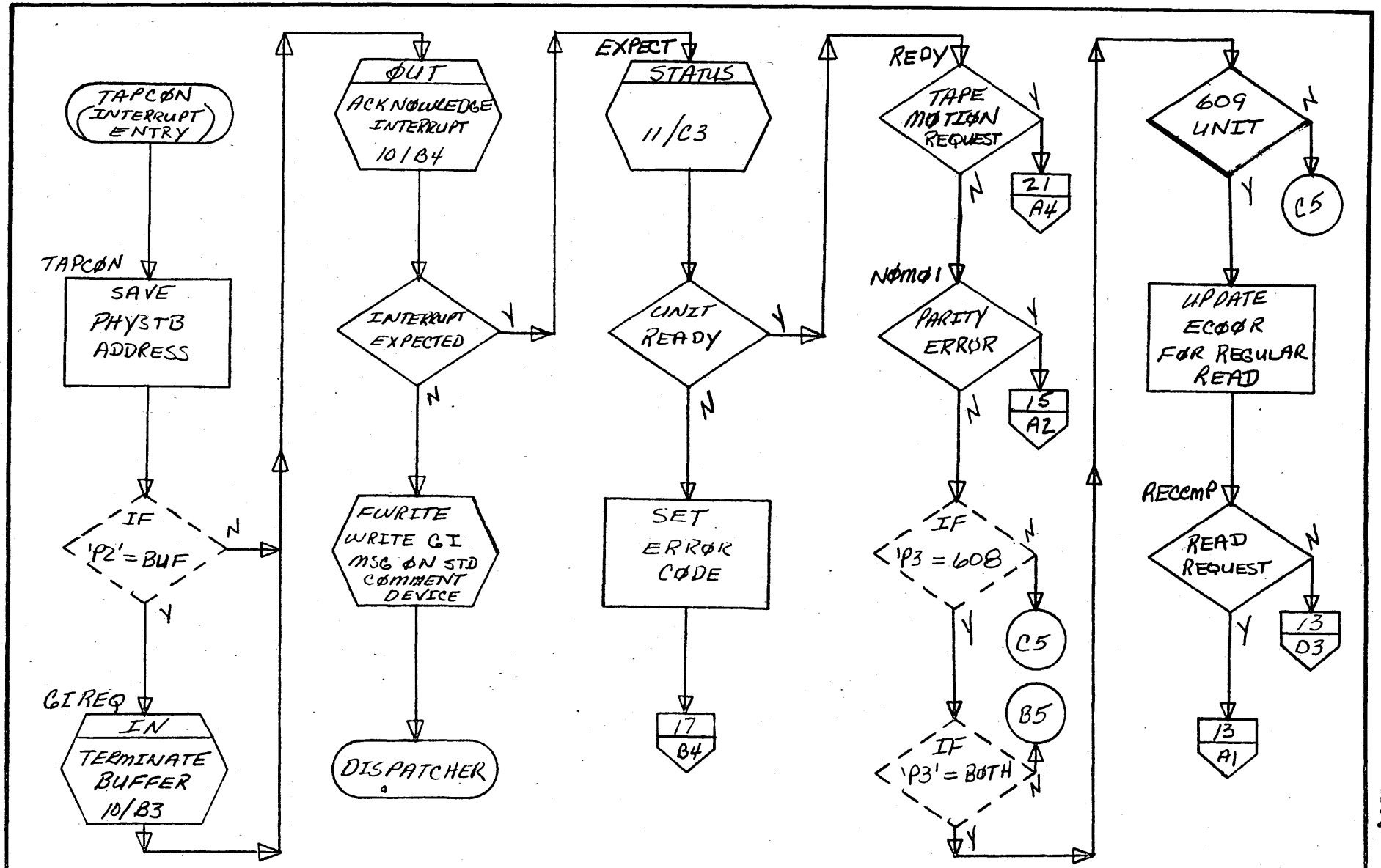


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE 1732/608-609 MAG. TAPE DRIVER PAGE 11 OF 25 NUMBER ISSUE DATE DRAWN BY DATE	PROJECT NO. PROJECT MGR. PROJECT NAME TASK NO. TASK NAME	REV	APPROVED	DATE

MAR 5 1971

54.25

AA1385 (FORMERLY CA127-1)

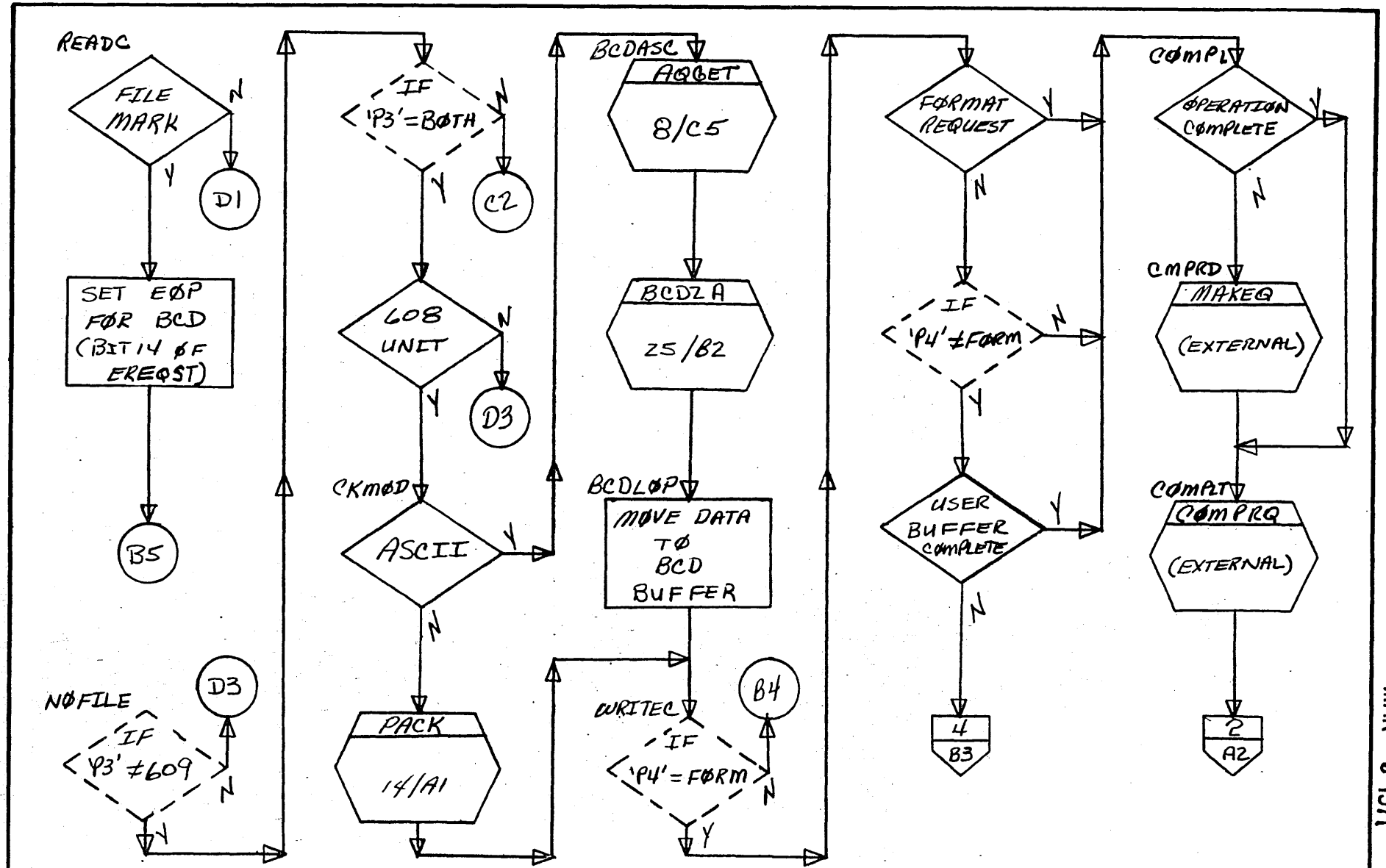


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAG.	PROJECT MGR.			
	TAPE DRIVER PAGE 12 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

54.26

AA1385 (FORMERLY CA127-1)

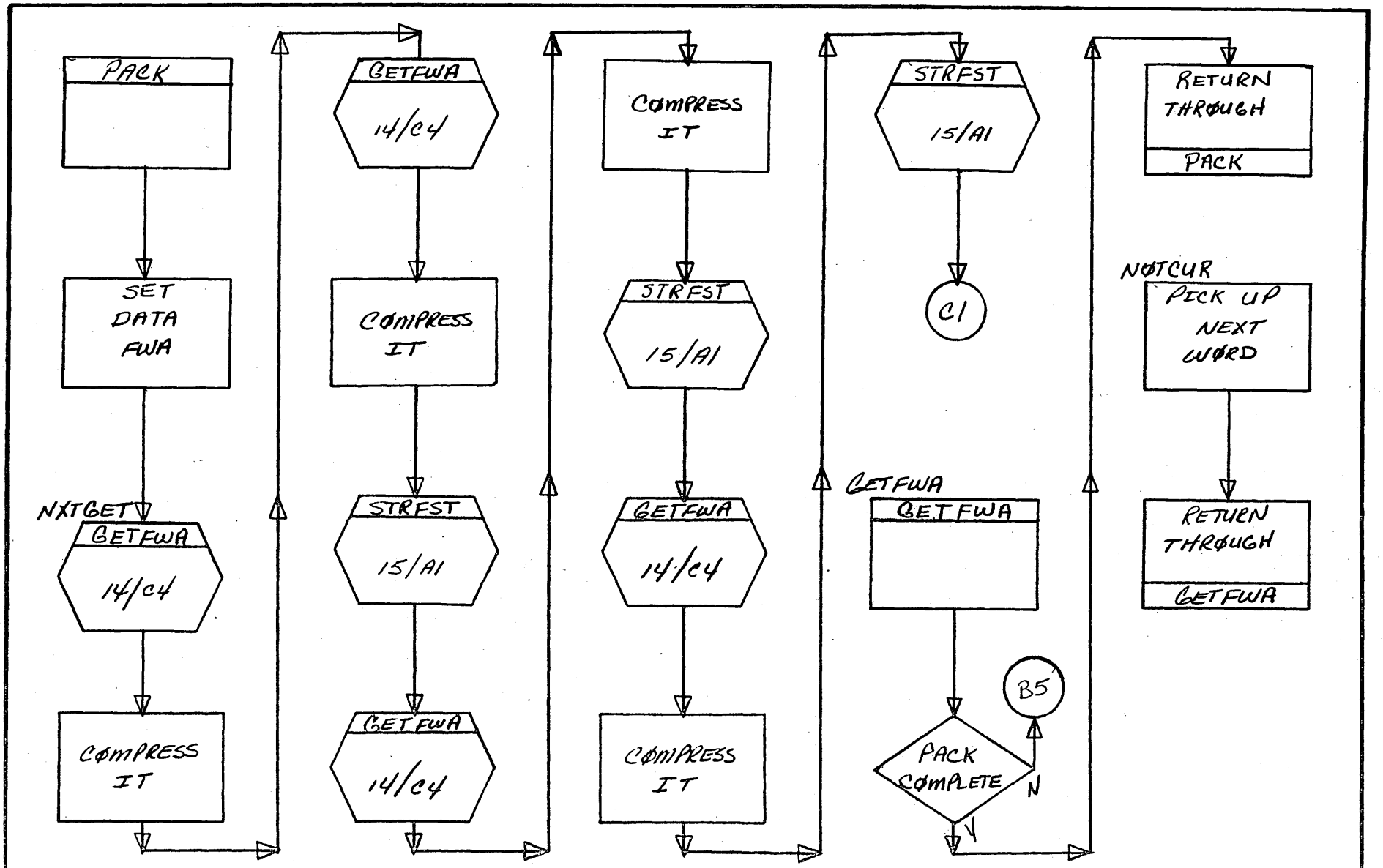


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1732/608-609 MAG.		PROJECT MGR.		
	TAPE DRIVER	PAGE 13 OF 25	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

54.27

AA1385 (FORMERLY CA127-1)

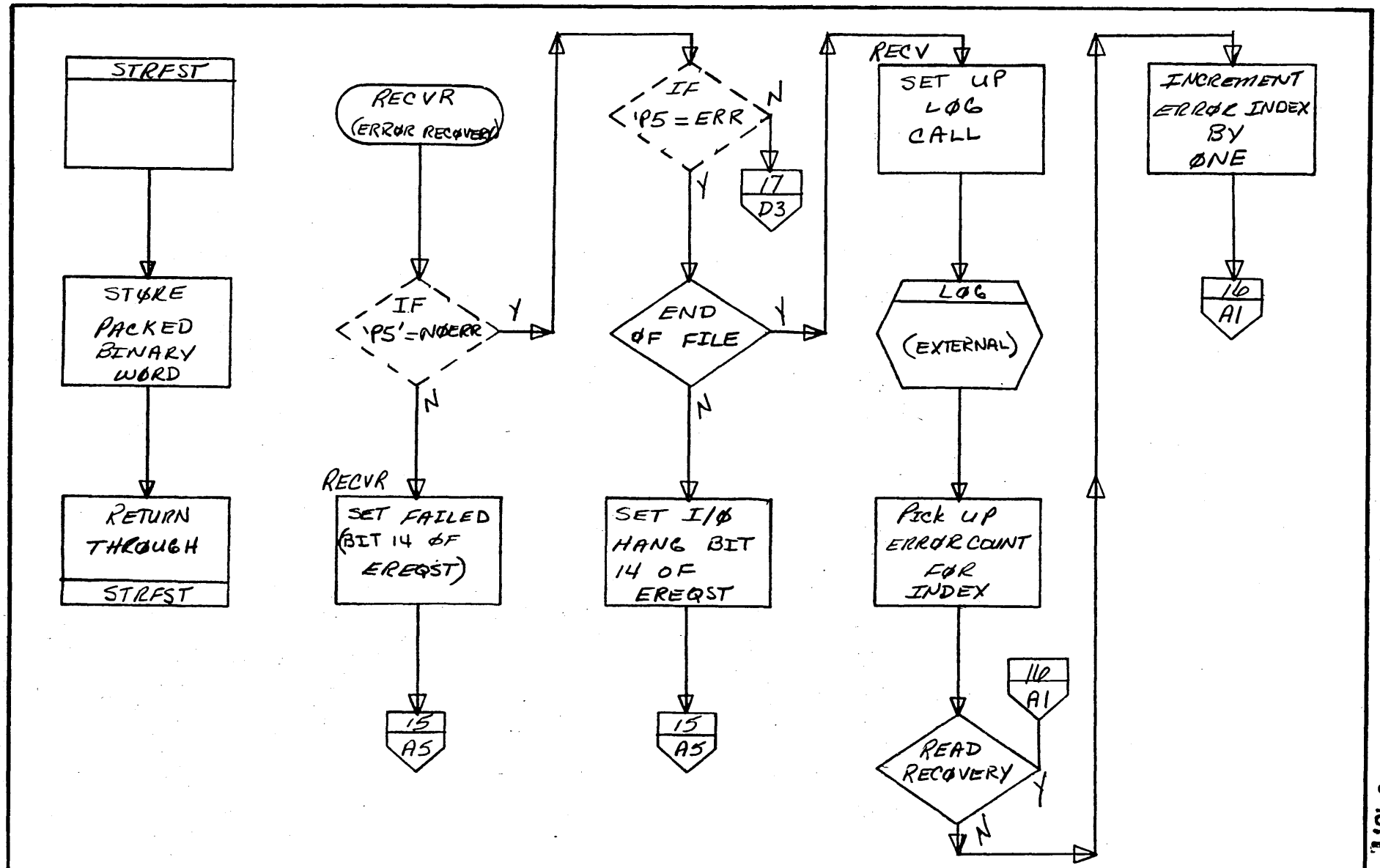


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 IMAGE.	PROJECT MGR.			
	TAPE DRIVER PAGE 14 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.28

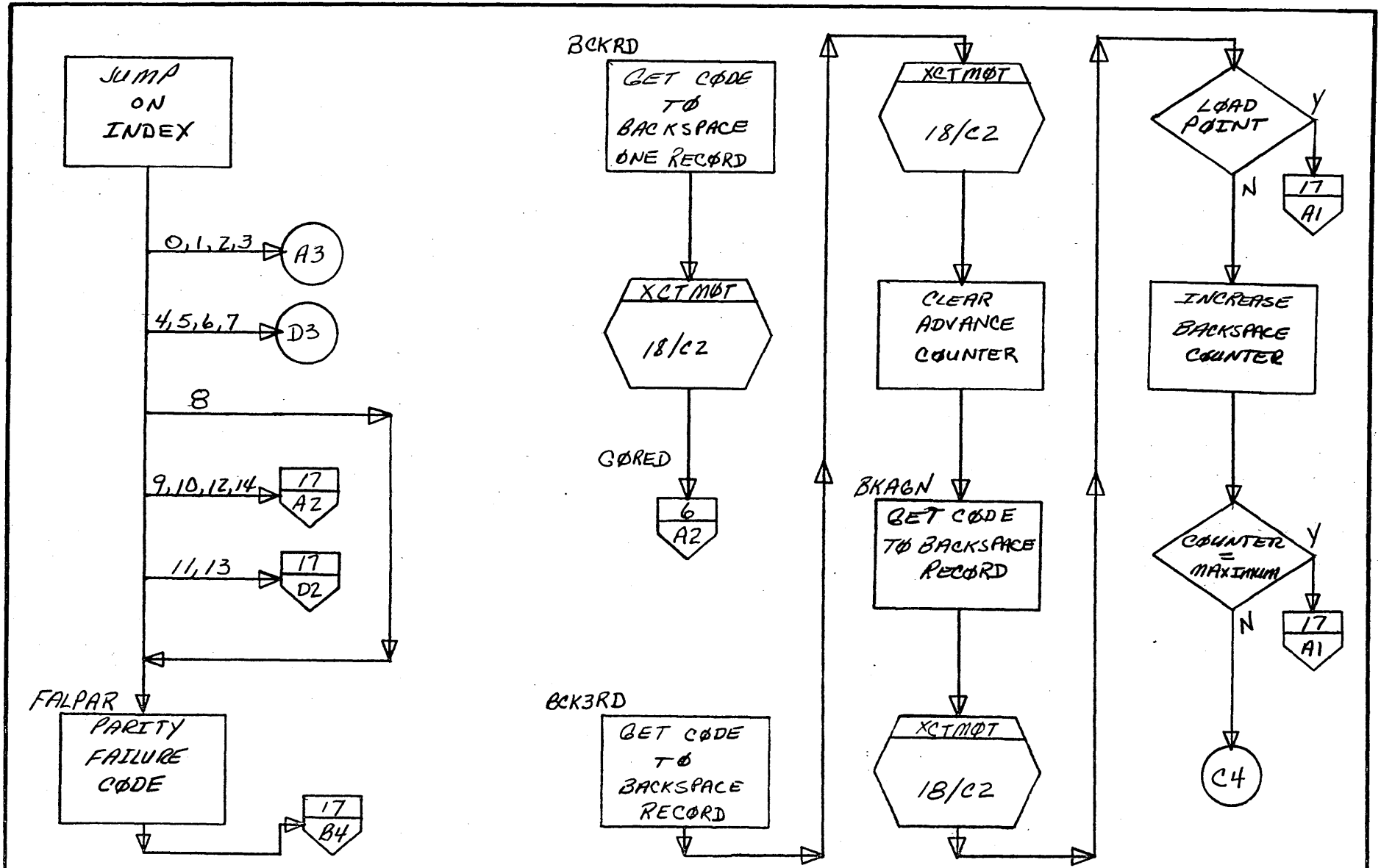


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAC.	PROJECT MGR.			
	TAPE DRIVER PAGE 15 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

54-29

AA1385 (FORMERLY CA127-1)

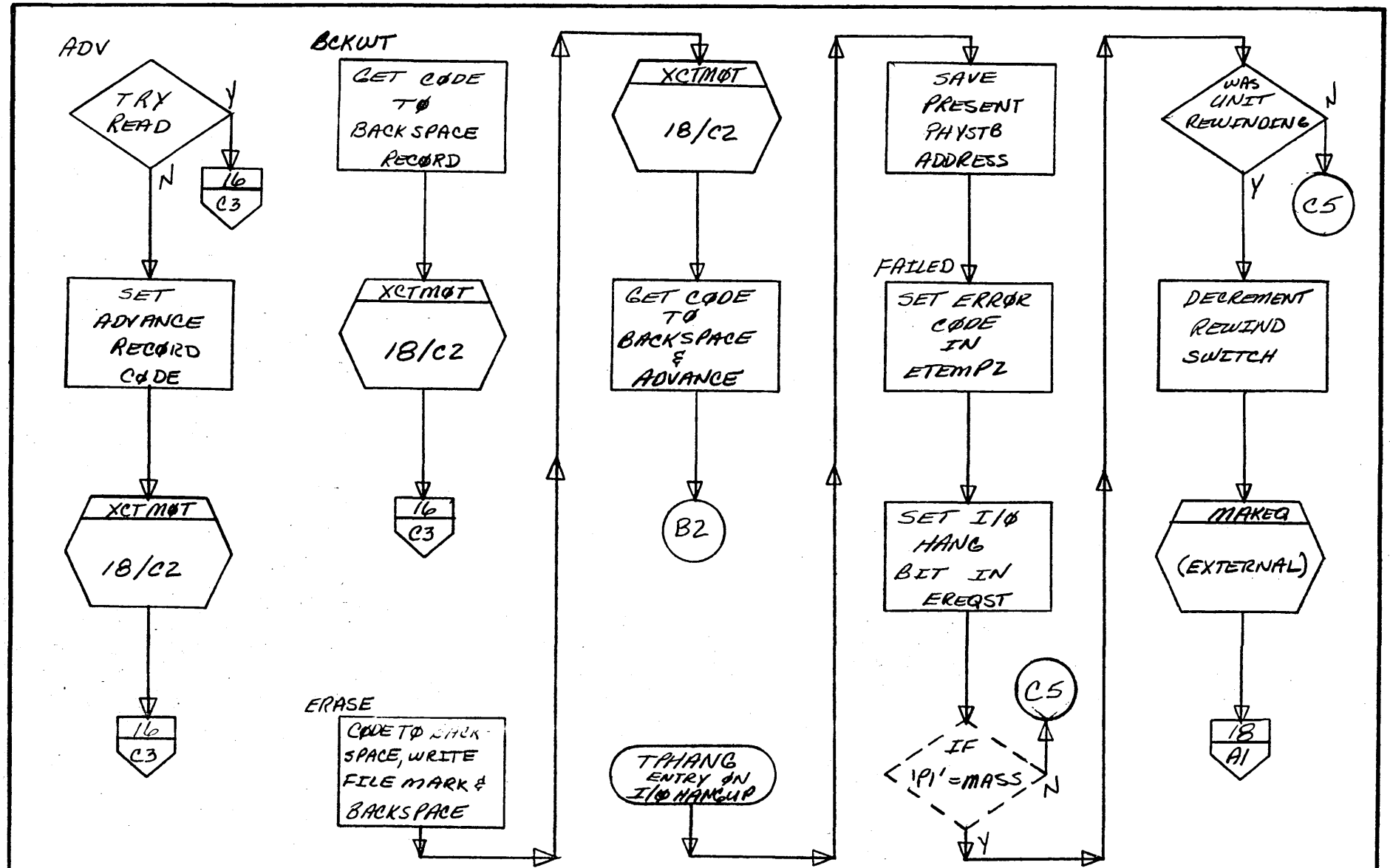


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1732/608-609 MAG. TAPE DRIVER	PAGE 16 OF 25	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

54.30

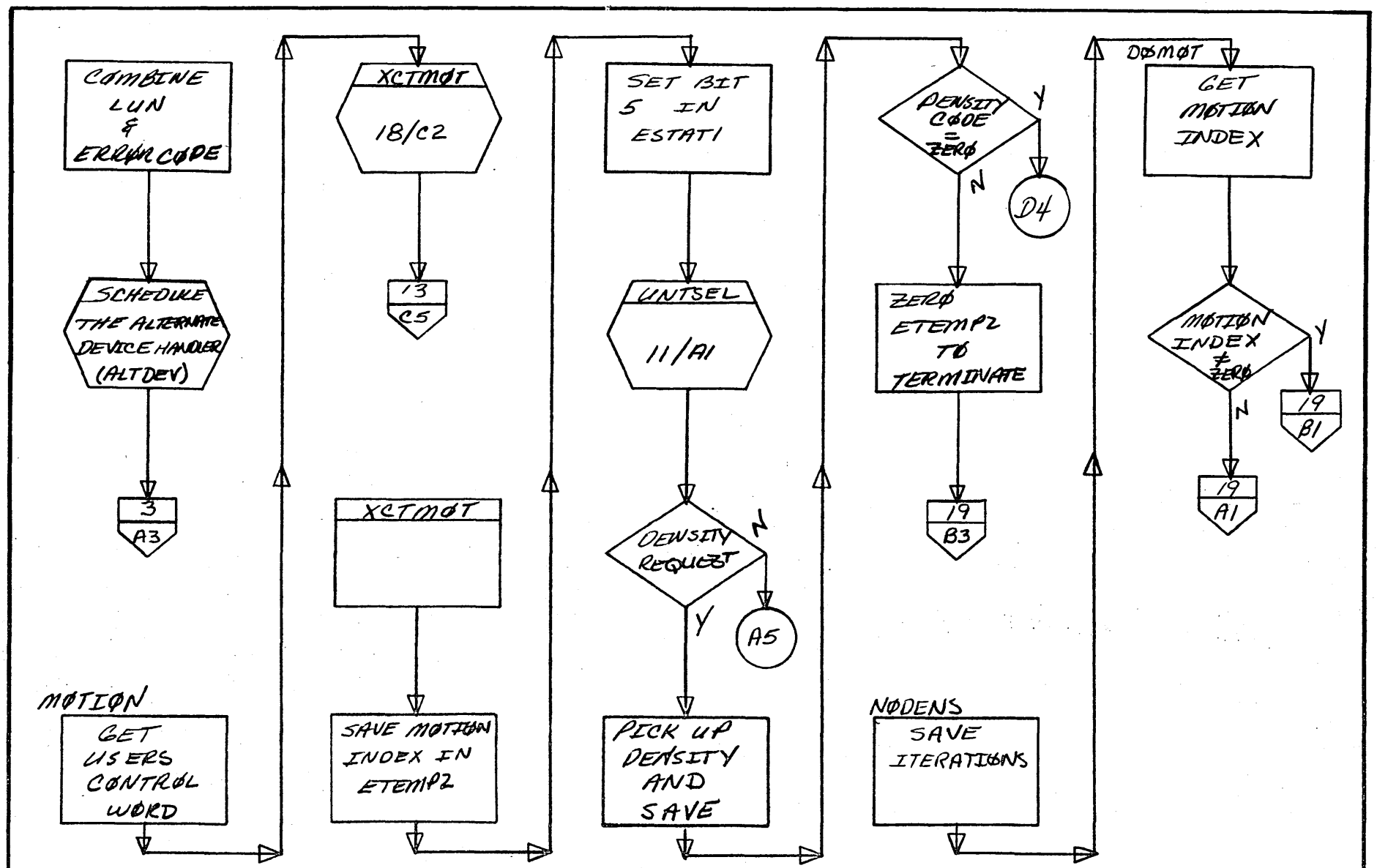
AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAG.	PROJECT MGR.			
	TAPE DRIVER PAGE 1 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

54.31

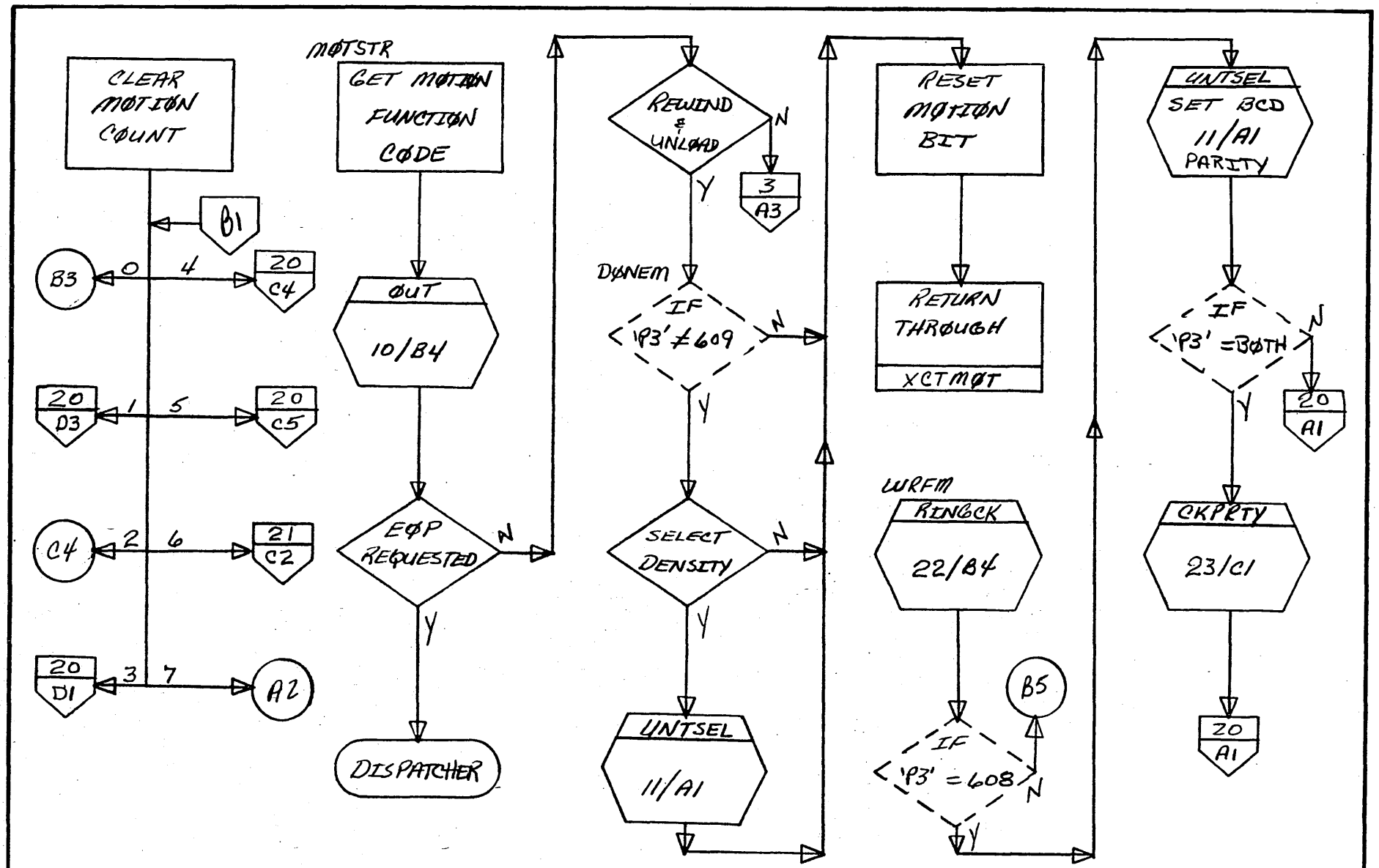


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1732/1608-609 MAG. TAPE DRIVER	PAGE 18 OF 25	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.32

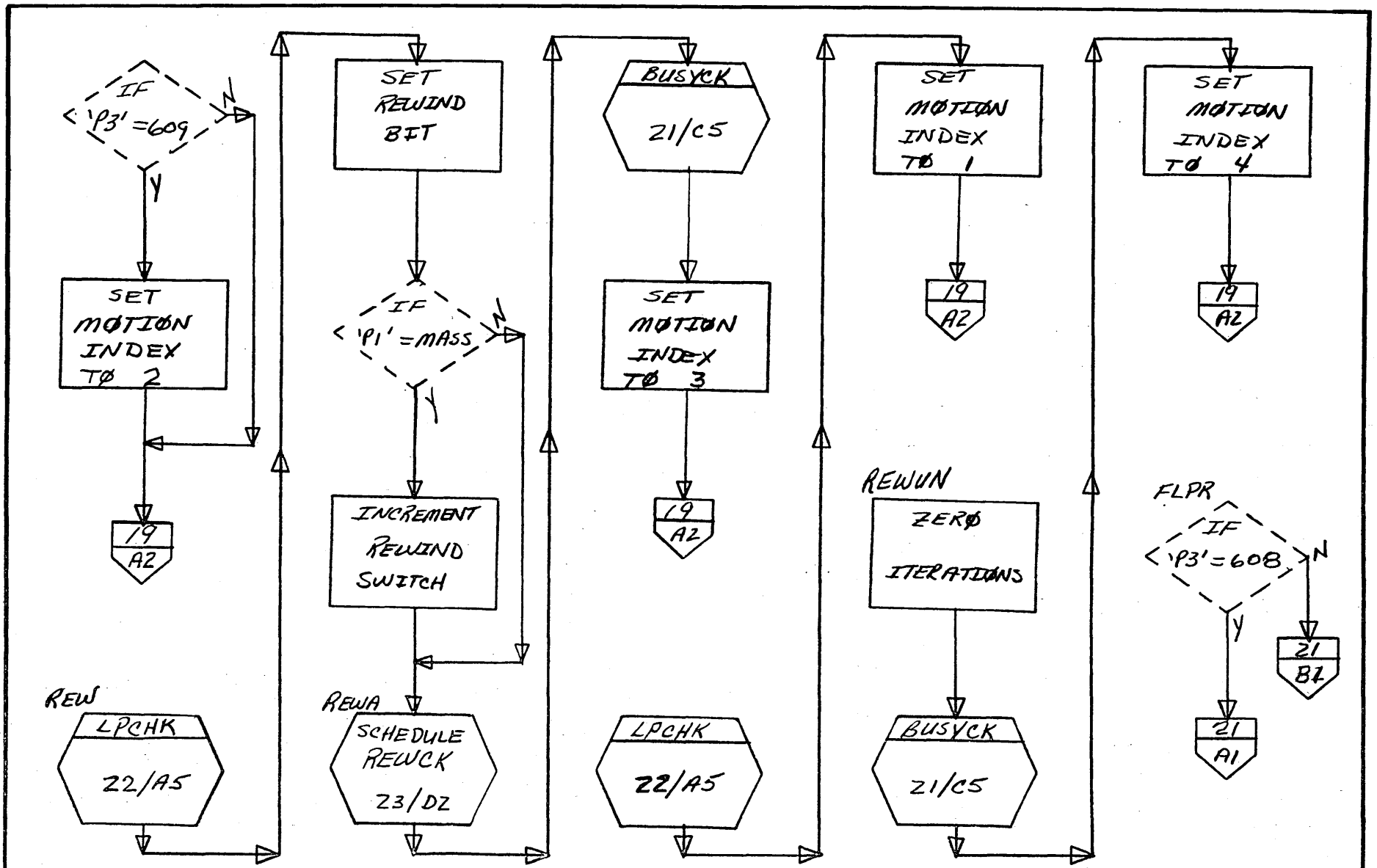


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS FMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1732/608-609 MAG. TAPE DRIVER	PAGE 19 OF 25	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971.

54.33

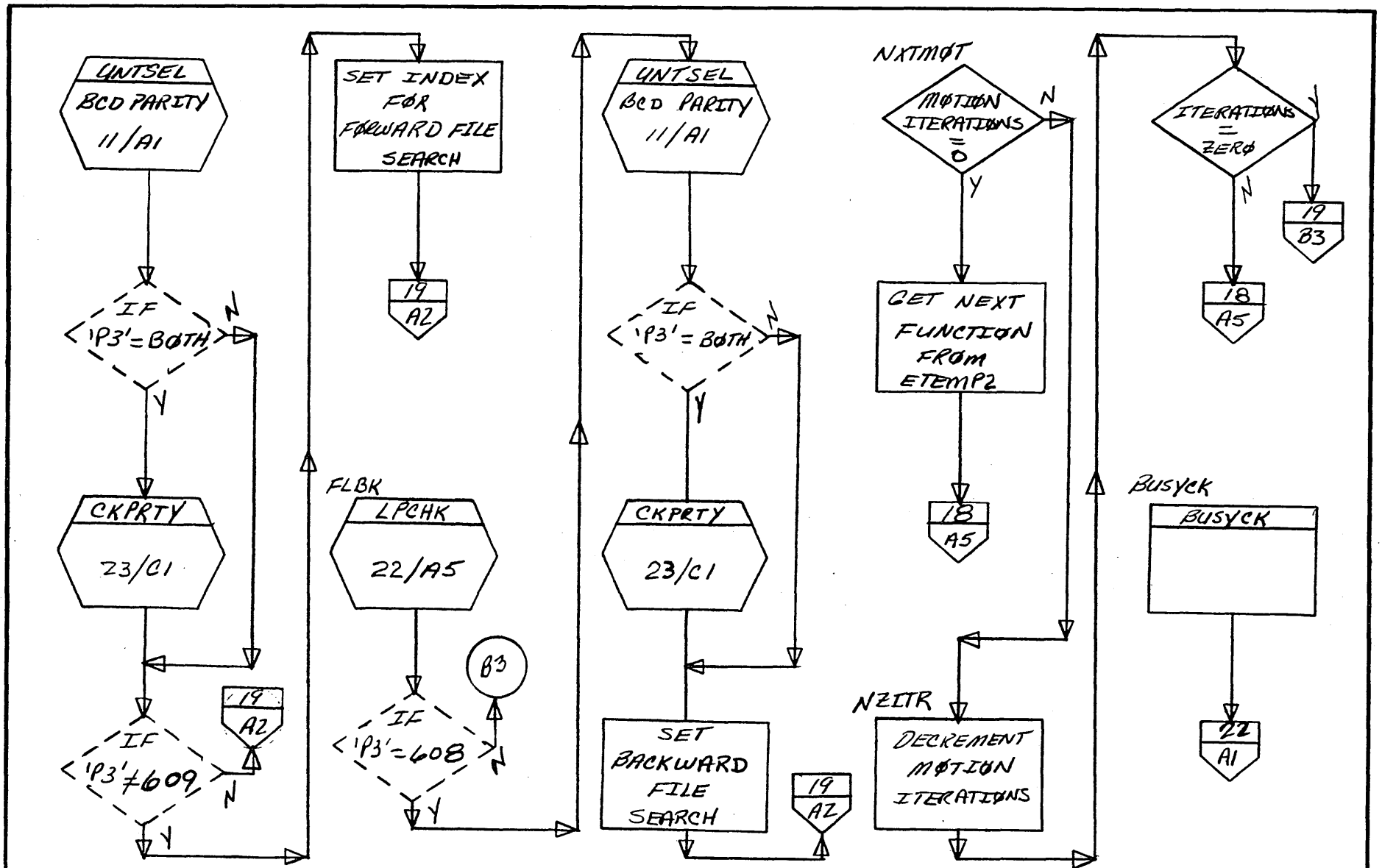


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAG.	PROJECT MGR.			
	TAPE DRIVER PAGE 20 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.34

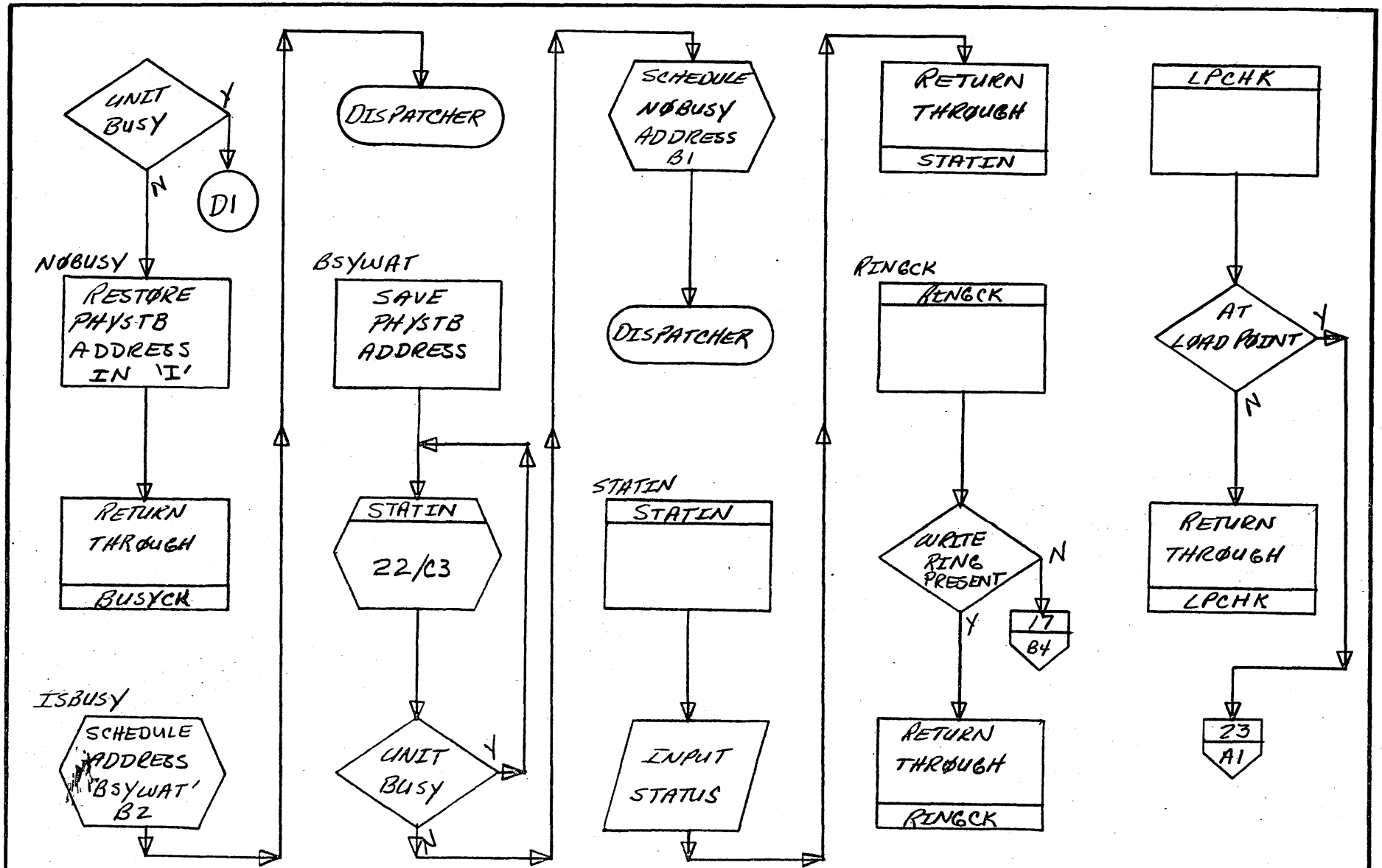


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE 1732/608-609 MAC.	PROJECT MGR.			
	TAPE DRIVER PAGE 21 OF 25	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.35

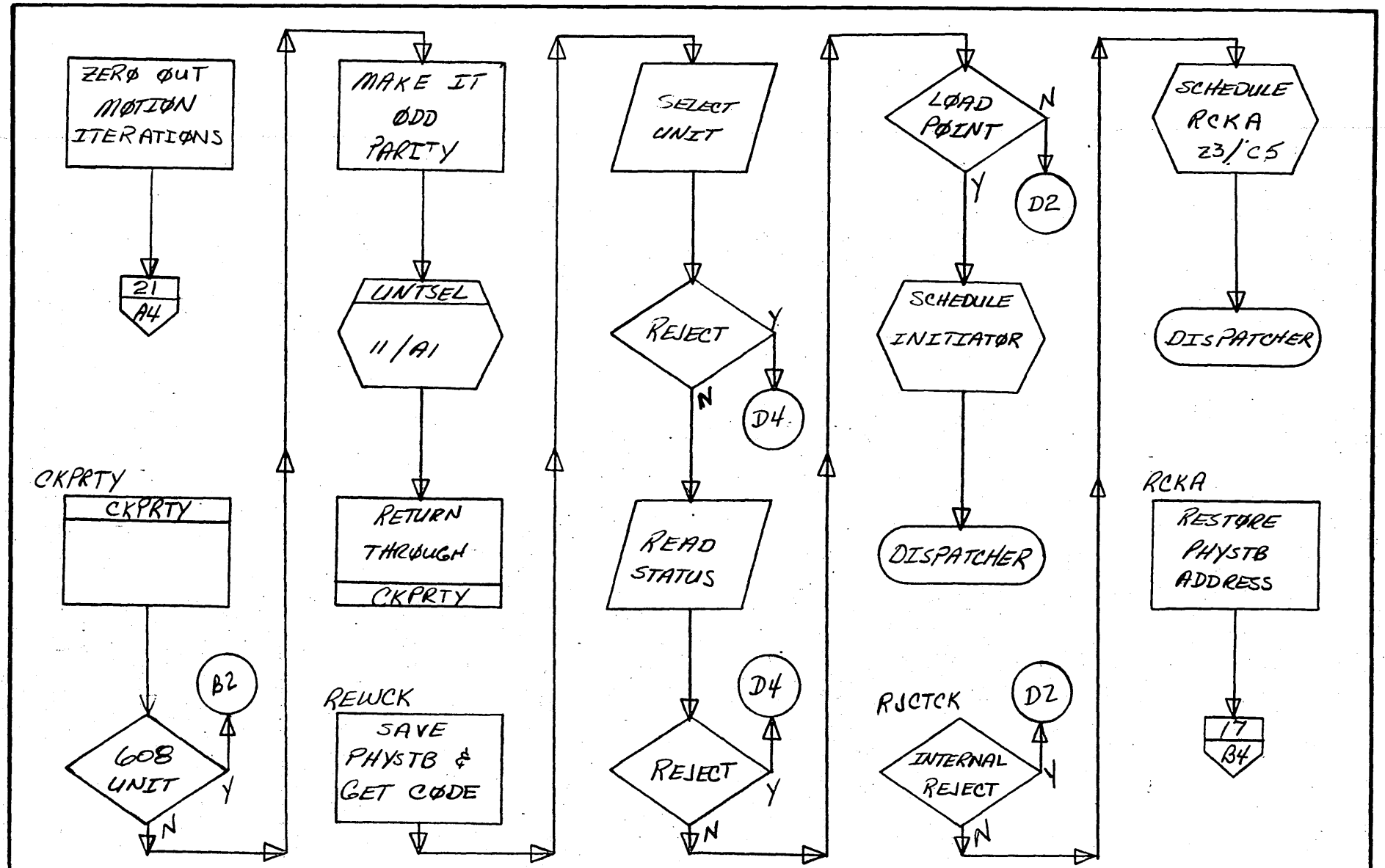


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE: 1732/608-609 IMAC.		PROJECT MGR.		
FLOWCHART ☐	TAPE DRIVER	PAGE 22 OF 25	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.36

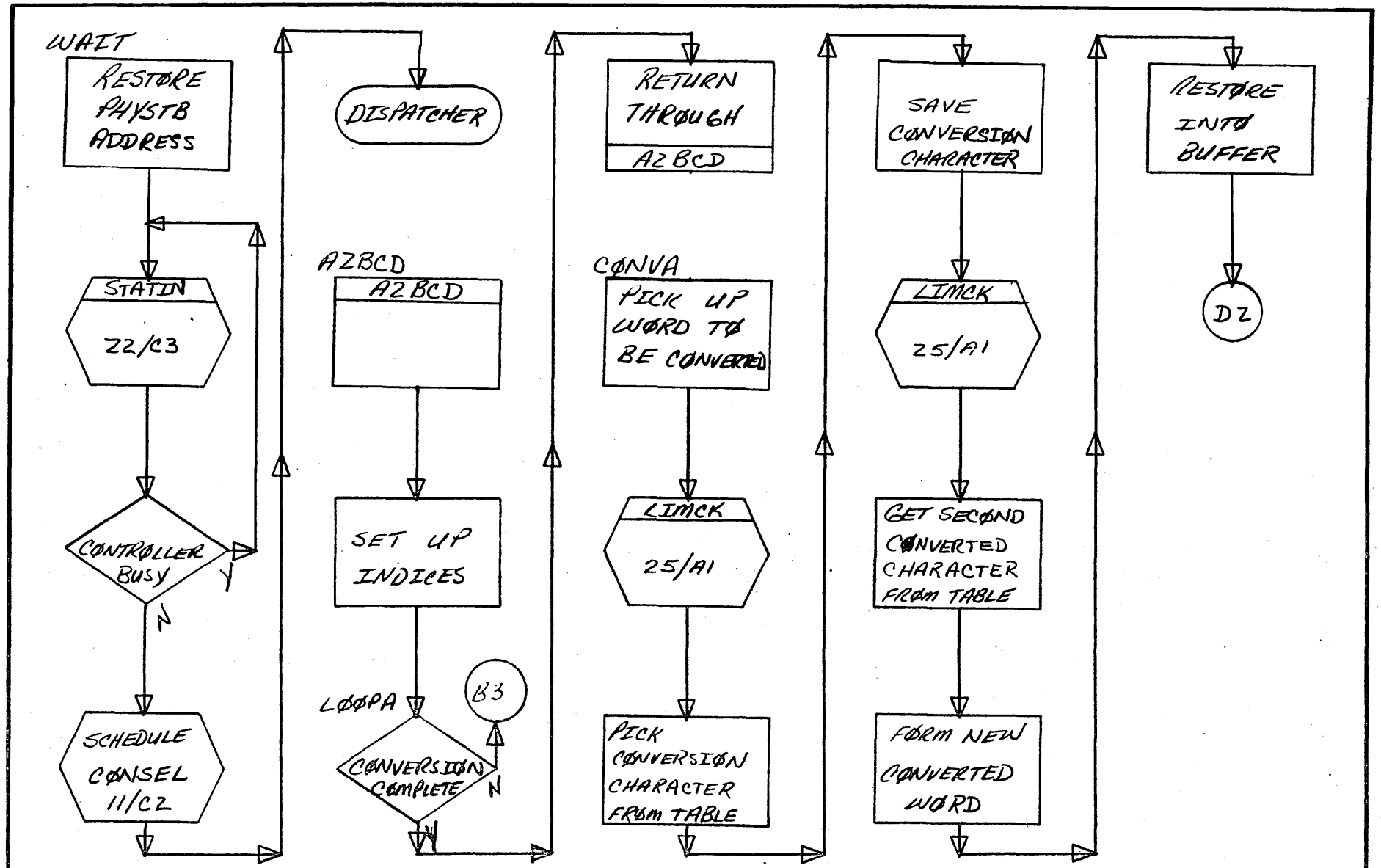


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1732/608-609 MAG. TAPE DRIVER	PAGE 23 OF 25	PROJECT MGR		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

AA1385 (FORMERLY CA127-1)

MAR 5 1971

54.37

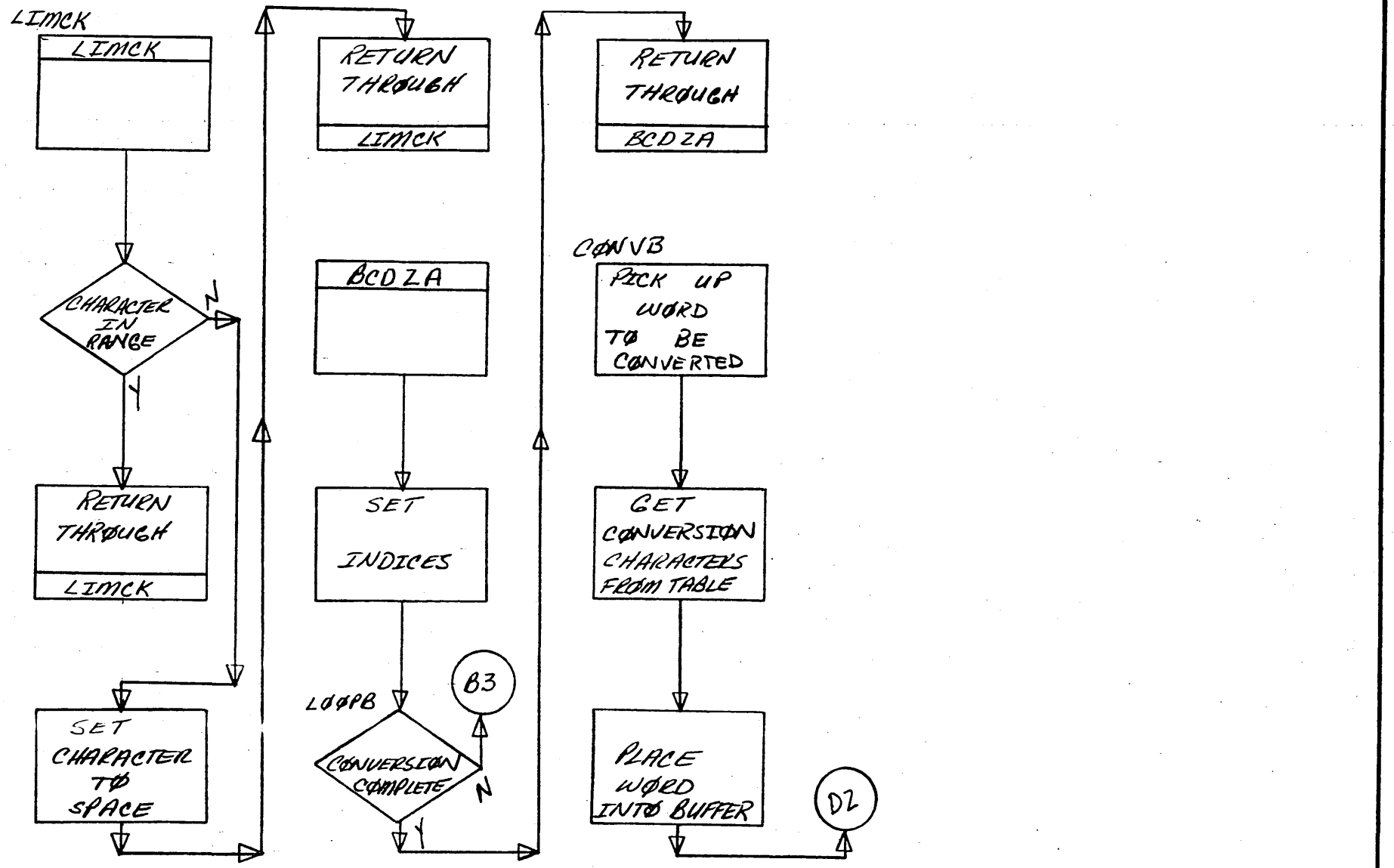


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1732/6608-6609 MAC		PROJECT MGR.		
FLOWCHART ☐	TAPE DRIVER	PAGE 24 OF 25	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971.

54.38

AA1385 (FORMERLY CA127-1)



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS I111S	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE 1732/608-609 MAG.		PROJECT MGR.		
	TAPE DRIVER	PAGE 25 OF 25	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

54-39

AA1385 (FORMERLY CA127-1)

55.0 1745-2 DISPLAY DRIVER

55.1 General Structure

55

There are three(3) modules in this driver, the Initiator, the Continuator, and the Diagnostic entry(DDINIT, DDCONT, DDDIAG). Actions are usually initiated in the Initiator and completed in the Continuator, but control is often passed back and forth between these modules during the operation of the driver.

Error Entry

The first portion of DDDIAG is entered only in the case of Timer Timeout. The second portion, which prepares for the jump to the Alternate Device Handler, is entered nearly every time there is any kind of a reject of an I/O instruction. The only exception is in the unbuffered version of the driver when the driver initiates a read and the display controller rejects because data is not ready the first try. (A Data Interrupt will indicate when data is ready.) There are no re-tries on rejects.

This driver is interrupt-driven. Actions are initiated by the driver and control is then given to the dispatcher until an interrupt indicates the action is complete.

Interrupts are inhibited only when calls are made to request or release Volatile Storage.

2. Requesting Space

Normally when space is required in core a request is made for allocatable core via a Space Request. The driver will never request more than what is needed for a full CRT(13 X 80 CRT can hold maximum of 1040_{10} characters, 20 X 50 CRT maximum of 1000_{10} characters), and in the case of small requests goes to Volatile because Allocatable is not always available.

This driver pays no attention to line size(either 80 or 50 characters). Data arrangement in lines is up to the user.

Allocatable core is requested when data is read in for an FREAD command.(buffered version only) The data read in may include STX and ETX characters which need to be stripped from the message. Then the edited data is passed to the user's buffer.

Allocatable core is also used to hold the user's pseudo-escape codes because the hardware will not handle pseudo-escape codes appropriately. Instead, the pseudo-escape codes in the user's buffer are replaced with Sync codes (which do nothing), until the message has been sent. Then the pseudo-escape codes are returned to the user's buffer allowing the user to send the same message again if he desires.

3. Pseudo-Escapes

Pseudo-escape codes are those which the driver will interpret and take the necessary action to perform but which are not acted upon solely by the hardware.

They are:

FUNCTION	ESCAPE	HEX VALUE
Clr/Reset	1B	0C
Turn on Alert	1B	07
Disallow Send Interrupts	1B	1A
No Cursor Movement	1B	16

4. Programmed Line Skip

The FWRITE instruction normally activates a Line Skip procedure before the user's data is sent to the CRT. This will cause the data to be displayed on a new line. If the user wishes to prevent this Line Skip, and he chooses to write no other escape functions(either pseudo or otherwise) the "No Cursor Movement" function will enable him to do so. Any other escape function at the beginning of the message will automatically inhibit the usual Line Skip.

The automatic Line Skip also has a couple Line Clear functions associated with it. (Line Clear will work only with a display that has an edit keyboard.) The automatic Line Skip also occurs preceding an FREAD and includes an STX code. Then data which the operator enters will be read beginning with the STX. If the operator chooses, the CRT may be cleared before entering, and if no STX is placed back on the CRT, the read operation, which begins after the operator depresses SEND, will begin at the upper left corner.

5. Priority

If a CRT is to be used as a comment device, it should be run at the same priority level as the standard comment device. If buffered it should run at a priority level equal to or lower than other devices on the 1706, and at the same priority as the 1706.

6. Status

A program which requests Status of the device will get Director Status one(1) from word twelve(12) of the Physical Device Table.

7. Status Switches

The setting of the Status Switches, which is obtained with Director Status Two(2), is passed to the user in his routine RD1745(Send Request Processor) in the Q Register along with the logical unit number of the station that interrupted. The logical unit number appears in the lower eight(8) bits, and the Status Switches appear in bits eight(8) to eleven(11).

8. Not Re-entrant

This driver normally expects to complete any operation which it has begun before allowing another request to be acted upon, even if a newer request is of a higher priority level. At the beginning of the driver the "op in progress" bit is examined to check for driver in operation. If the driver is busy with an FWRITE, WRITE or READ request on any of the logical units which are threaded together, control will be released to the Dispatcher. The request will be picked up after completion by going through FNR for each device on the Thread.

The only exception is if an FREAD request is in progress and has reached the point where the driver is waiting for the operator to enter information. This point is indicated by thc absence of bit six(6) in word nine(9) of the Physical Device Table. Bit six(6) present indicates the driver is processing an FREAD request. When bit six(6) is absent the driver will allow other requests for other logical units to be processed.

9.1 Request Flow

The general sequence of events for an FREAD request is as follows:

An Alert, Line Clear, Line Skip,Line Clear, and STX codes are first written on the display. In the 1706(Buffered) version of the driver, a completion interrupt from the 1706 will then be expected. In the unbuffered version, a data interrupt is expected after each word written or read. Then a Write Terminate instruction is executed to obtain a 1745-2 End of Operation interrupt to assure that the display is done handling the data. (When a Line Skip or New Line Code is sent as the last character in a message, a maximum of one(1) millisecond may elapse before the End of Operation interrupt occurs.) After clearing bit six(6) in word nine(9) and clearing Active on the display, the driver is exited to wait for operator data entry. When the Send interrupt occurs, indicating the operator has entered the data, the data is read in from the STX marker or from upper left corner of the CRT.

Completion of the read is indicated by the ETX causing a 1745-2 End of Operation interrupt or a 1706 End of Operation interrupt if the buffered version is being used. The 1706 interrupt will occur only if the number of words requested is equal to or less than the number of words entered. In the unbuffered version, the driver will read to ETX (End of Operation interrupt), or until the requested number of words has been read whichever is shorter.

The data is read in and edited so as to remove the STX(if any,) and the ETX codes. Then control is passed back to the user program or to additional requests if any are stacked. (In the buffered version the data is first read into an Allocatable or Volatile Storage area before being edited and moved to the user's area.) The edited data is followed by an $FF character or an $FFFF word if space is available.

9.2 READ request

The READ request is not preceded with any kind of a write but reads immediately from the current cursor position to ETX or buffer full, whichever is shorter. The data is passed directly to the user unedited.

9.3 FWRITE request

The data in an FWRITE request is examined for an escape code in the first character position. If one is present the next character is examined to see if it is one which the hardware will act upon, or if it is one defined as a pseudo-escape code upon which the driver will act. Pseudo-escape codes are processed by the driver, stored elsewhere, and replaced in the data stream with Sync codes. If there are no escape codes at the beginning of the data stream, a Line Clear, Line Skip, Line Clear series of codes is sent to the display. They are followed by a Write Terminate. Then the data stream is sent, followed by a Write Terminate. At the completion of the Write Terminate, the pseudo-escape codes are returned to their respective positions in the data stream, the area where they were stored is released, and the request is completed.

9.4 WRITE request

This request is much like the FWRITE request because data at the beginning of the data stream is examined for pseudo-escape codes and such codes are implemented if found. However, there is no provision for an automatic Line Skip as in the FWRITE request.

10. Flag Settings

When an interrupt occurs, the following values of DINFLG enable the driver to know what operation was completed:

Buffered version:

1	Write Terminate caused End of Op Interrupt on 1745-2
2	READ completed through 1706
4	Line Skip preceding FWRITE completed through 1706
8	Line Skip preceding FREAD completed
10	Completed transfer of data on FREAD through 1706
20	Completed transfer of data on WRITE or FWRITE

Unbuffered version:

2	READ data transfer completed
4	Line Skip preceding FWRITE completed
8	Line Skip preceding FREAD completed
10	FREAD data transfer completed
20	Completed data transfer WRITE or FWRITE
40	Completed reading one extra word on FREAD

The FREAD commands usually end up reading one word more than required due to the additional character positions taken up by ETX and STX if present. If STX is present at the beginning of the message, the STX character is stripped off and all other characters moved up one character position. The operator may remove the STX by clearing the CRT manually. Therefore, the driver checks first whether the STX is present.

11. Additional Flags

SENFLG 1 = Do not enable SEND interrupts
0 = Enable SEND interrupts

This flag is examined and cleared each time the driver nears completion; therefore, the user must request the pseudo-code "Disallow Send Interrupts," with each WRITE or FWRITE command if he wishes to continue disallowing Send Interrupts. Send is automatically allowed when the driver encounters an FREAD command.

RWFLG (in the unbuffered version) indicates whether the driver is in a read or write operation.

0 = Read
1 = Write

ALLFLG if set indicates the address at which allocated core begins.

MIFLG indicates whether manual Interrupt has been requested by operator.

12. Physical Device Table

	EXT	DDINIT, DDCONT, DDDIAG	
CRT451	NUM	$1299	Request for Scheduler
Word 1	ADC	DDINIT	Initiator Entry Address
2	ADC	DDCONT	Continuator Entry Address
3	ADC	DDDIAG	Diagnostic Entry Address
4	NUM	-1	Diagnostic Clock
5	NUM	0	Logical Unit
6	NUM	0	Parameter List Location
7	NUM	$1501(buffered) or $501 (unbuffered) Code to obtain status from display device	
8	NUM	$E6	Equipment Code
9	NUM	0	Error field and request code
10	NUM	0	First Word Address(FWA)
11	NUM	0	Last Word Address + 1 (LWA + 1)
12	NUM	0	Last Equipment Status of device(Director Status 1)
13	NUM	$40	Physical Station Number(Example shows Physical Station 1)
14	ADC	CRT452	Thread of next device(If only one device, should thread to itself--last device should always thread back to first device--if buffered, should thread to other devices on 1706 Buffered Data Channel.)
15	NUM	11	Station Logical Unit
16	NUM	0	Words 16 & 17 are temporary storage

12. Physical Device Table, concluded...

17	NUM	0	for the driver necessary when other stations have requests to be processed while one station is waiting for operator to enter data for an FREAD request. When FNR searches for other requests, words 5, 10, and 11 are destroyed. These must be replaced when the FREAD request is processed to completion after operator hits SEND.
18	NUM	$2000	Code to obtain Status on 1706 (Buffered driver only)

13. 1745-2 Hardware Function and Status Codes

Director Function 1

Bit 0	Clear Controller
1	Clear Interrupts and Enables
2	Enable Data Interrupt
3	Enable End of Operation Interrupt
7	Enable Station Interrupt
8	Clear Memory and Reset Marker
9	Alert
10	Reset Marker
12	Write Terminate

Director Function 2

Bit 4	Set Station Active
5	Clear Active
6-9	Station Address
10	Select Station
11	Deselect Station

Director Status 1

Bit 0	Ready
1	Busy
2	Interrupt
3	Data Ready
4	End of Operation
11	EOM
12	Station Active
13	Send Request
14	Message Pending

Director Status 2

Bit 0	Status Switch 1
1	Status Switch 2
2	Status Switch 3
3	Status Switch 4
6-9	Station Address

14. EPROC

Since EPROC is inadequate at present to handle interrupts from the 1706 or from multiple displays even if unbuffered, the user must provide his own interrupt handling capability. An example of such a routine which has been attached to SYSBUF is given in the Installation Manual.

In addition, the LOCORE Interrupt Trap region must be set up to indicate the user's interrupt handler instead of EPROC at the appropriate interrupt entry.

15. User's Send Request Processor

This routine is necessary because of the nature of the display Send Key. Since it is capable of interrupting at any time, the Send Request Processor should be set up to determine what to do with the interrupt. If there is a request for the driver when the SEND interrupt occurs, the driver will handle it, but if there is no request against the driver, control will be passed to the Send Request Processor, RD1745. In the routine RD1745 which must be installed in the Operating System at the same time as the driver, the user can specify whether a process program should be scheduled, an error message written to the Comment Device, or the interrupt should be ignored.

The following sample coding for RD1745 looks for Status Switch 3 set. If found, a process program from mass memory is scheduled. If not found an error message is scheduled for the Comment Device and exit made to the Dispatcher.

```
                NAM     RD1745
                ENT     RD1745
                EQU     FWRITE($C00),RP(16),CP(1)
                EQU     AMONI($F4),ADISP($EA)
                EQU     NOOOF(6)
                EXT     USERXY
STØRQ           NUM     0
STORQL          NUM     0
RD1745          NØP     0
                STQ*    STØRQ                   SAVE LUN + STATUS
                LLS     8
                QRS     8
                STQ*    STØRQL                  SAVE LUN ØNLY
                AND-    NOOOF                   MASK ØFF STATUS BITS IN A
                INA     -4                      CK FØR SWITCH 3 ØNLY SET(BIT 2)
                SAN     A-*-1                   IF ØTHER SWITCHES, WRITE ERRØR MES
                LDQ*    STØRQ
                RTJ-    (AMØNI)                 SCHEDULE USER PRØCESS TYPE PRG
                NUM     $1255                   WHICH RESIDES ØN MASS MEMØRY
                ADC     (USERXY)                NAME ØF USER PRØGRAM TØ BE
                                                CALLED FRØM DISK
                JMP-    (ADISP)                 EXIT TØ DISPATCHER
A               RTJ-    (AMØNI)                 WRITE ERRØR MESSAGE TØ CØMMENT DEVICE
                ADC     FWRITE+RP*5+CP*5
                NUM     0                       CØMPLETIØN ADDR
                NUM     0
                NUM     $8FC                    CØMMENT DEVICE

                NUM     17                      LENGTH ØF MESSAGE
                ADC     MES                     FWA ØF MESSAGE
                JMP-    (ADISP)
MES             ALF     17,INCØRRECT STATUS SWITCH SELECTIØN.
                END
```

16. Sample Process Program

The following code is an example of a process program residing on mass memory and called by RD1745. The first instruction $C8FE picks up the address in core where the program is placed when read in from mass memory. This program cannot have any P designations in the assembled listing. These can be avoided by coding such as LDA =XMES-USERX. Core occupied by this program must be released when the program is done.

This sample routine merely writes a message on the device that had a Send interrupt.

```
                NAM       USERX                  SAMPLE USER PRØCESS PRØGRAM
                ENT       USERX
USERX           NUM       $C8FE                  LDA WITH *-1 TØ FIND ØUT THE
*                                                ACTUAL BEGINNING ADDR ØF USERX
*                                                AFTER IT HAS BEEN LØADED FRØM DISK
                STQ*      STØRQ                  SAVE LUN NØ.+ STATUS SWITCH
*                                                SETTING--LUN IN LØWER 8 BITS--
*                                                STATUS SWITCHES IN BITS 8-11
                TRQ       A
                AND       =N$FF
                STA*      LUN
                LDA       =XMES-USERX
                ADD*      ADDR
                STA*      FWA
                NUM       $C00                   FWRITE
                NUM       0                      CØNTINUATIØN
                NUM       0                      THREAD
LUN             NUM       0                      LUN
                NUM       0                      MESSAGE LENGTH
FWA             NUM       0                      FIRST WORD ADDRESS OF MESSAGE
                LDA*      ADDR
                STA*      REL + 2
REL             RTJ-      ($F4)                  REQUEST TØ RELEASE CØRE
                NUM       $1801                  ØCCUPIED BY THIS PRØGRAM
                ADC       0                      JUST BEFØRE EXITING
                JMP-      ($EA)                  EXIT TØ DISPATCHER
ADDR            NUM       0                      BASE ADDRESS ØF PRØGRAM IN CØRE
STØRQ           NUM       0
MES             ALF       9, THIS IS A MESSAGE
                END       USERX
```

A process program such as the one above is included in the system at installation time. A *YM entry with its ordinal, the program itself, and a priority level *S statement make up the items to be included at install time. The following are sample entries:

```
*YM,USERXY,19
      ≀
*M   USERXY
      ≀
*S,19,3,M
```

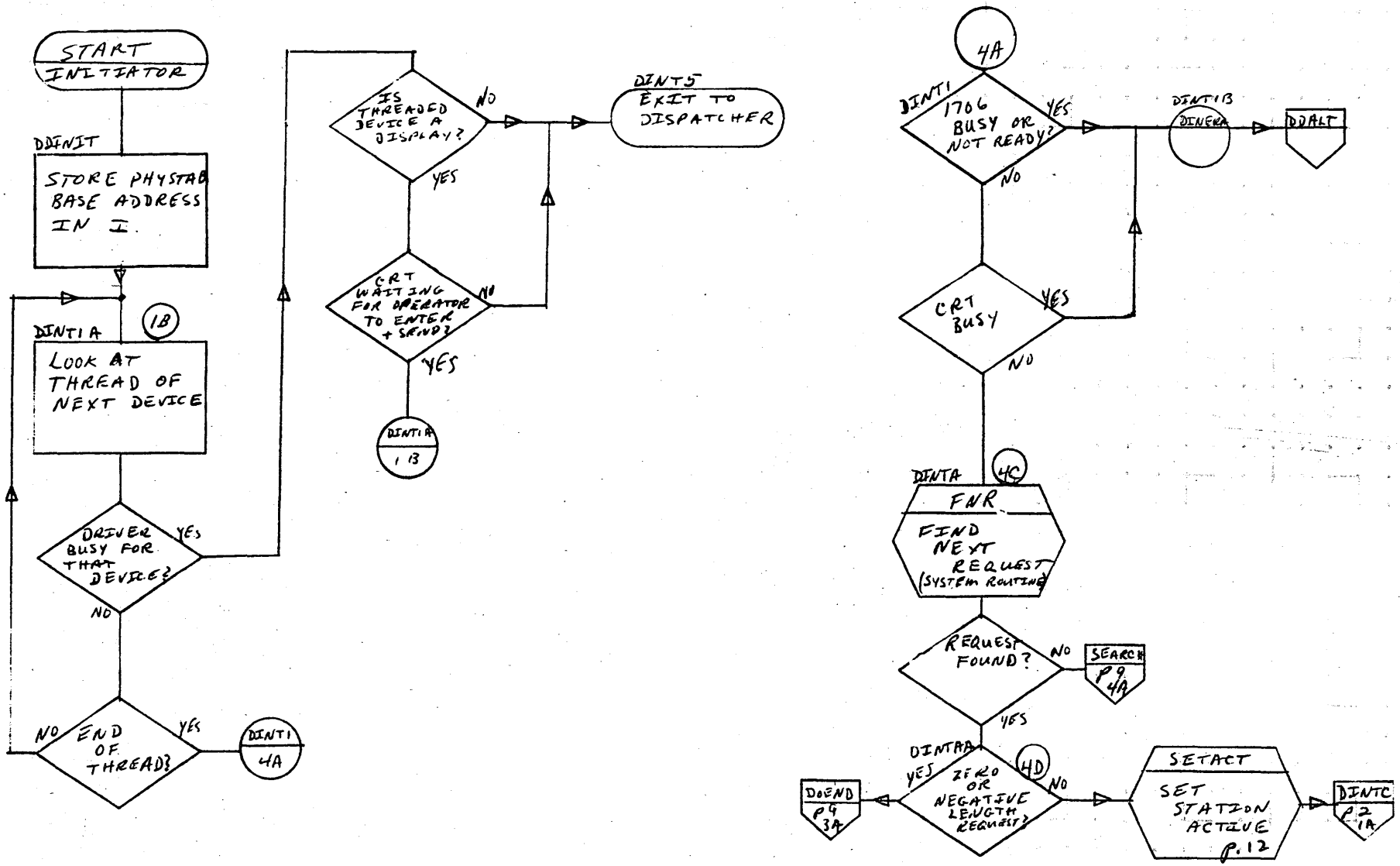
17. Clearing Interrupts

Normally the operator ought to manually Master Clear the 1745-2 Display Controller before putting it into operation to clear out SEND interrupts if any SEND keys have been pushed when the computer is not operating. However, the following code, when added to the SPACE module of the Operating System, will help to clear up this problem. It can be inserted following the REJ NØP 0 instruction. It should not be used if any devices other than CRT's are on the thread unless code is added to check the device type, which is found in word 8 of the Physical Device Table, bits 10 through 4. The device type for CRT's is E, and is found in bits 7-4. All display devices in the thread must be turned on before this coding will work properly.

```
********* CODE INSERTED IN SPACE MODULE TO CLEAR CONTROLLER
********* AT RESTART.
            EXT     CRT451
            LDQ     =XCRT451        NAME OF ANY CRT PHY DEVICE TABLE
            STQ*    SAVQ
            STQ*    STØRQ
LØØP        LDA-    13, Q
            AND     =N$03C0         SAVE STATIØN NUMBER BITS 6-9
            ADD     =N$0410         ADD SEL STATION 1 + SET ACT BITS
            LDQ-    7,Q
            INQ     1               DIRECTØR FUNCTIØN 2
            NØP     0
            ØUT     -1
            INQ     -2              SET UP DATA WRITE
            LDA     =N$1616         WRITE SYNCS TØ CLR INTERRUPTS IF ANY
            NØP     0
            ØUT     -1
            LDQ*    STØRQ
            LDA-    13,Q
            AND     =N$03C0
            ADD     =N$0020         CLEAR ACTIVE BIT
            LDQ-    7,Q
            INQ     1
            NØP     0
            ØUT     -1
            LDQ*    STØRQ
            LDA-    14,Q            THREAD WØRD
            STA*    STØRQ
            SUB*    SAVQ            CHECK FØR END ØF THREAD
            SAZ     4
            LDQ*    STØRQ
            JMP*    LØØP
SAVQ        NUM     0               TEMP STØRAGE
STØRQ       NUM     0               TEMP STØRAGE
```

1745-2 DISPLAY DRIVER (BUFFERED)

INITIATOR



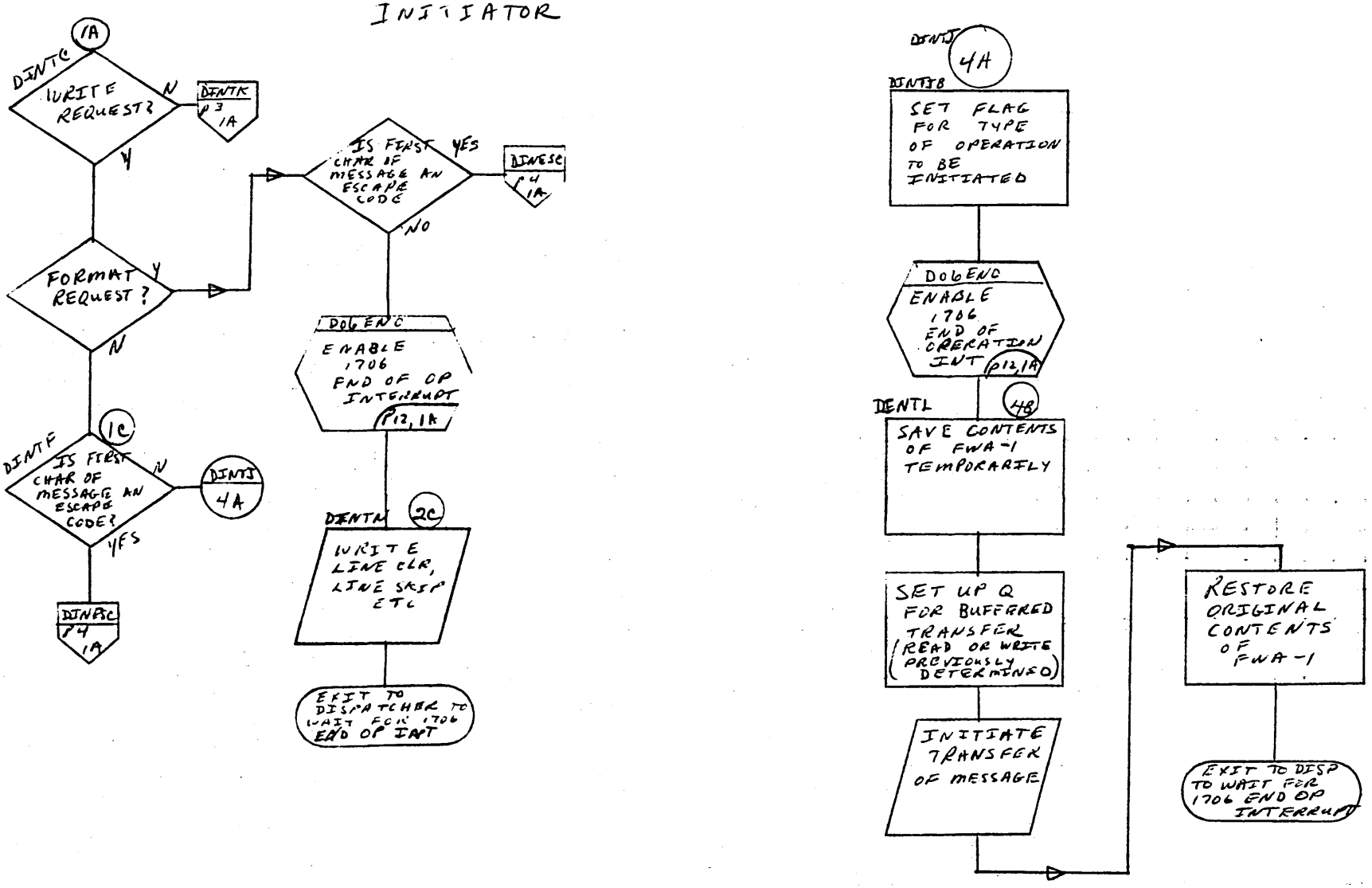
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE / OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

55.11

1745-2 DISPLAY DRIVER (BUFFERED)

INITIATOR



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 2 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

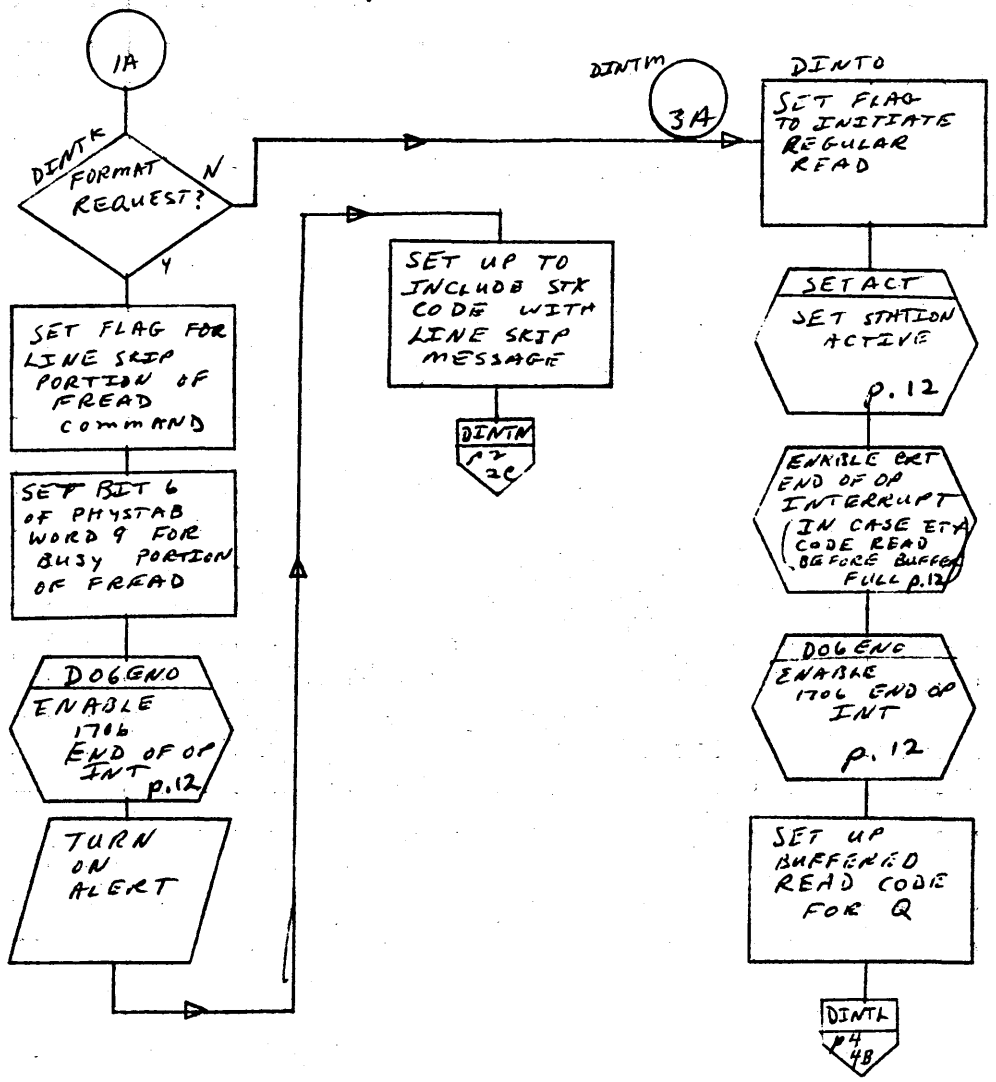
AA1385 (FORMERLY CA127-1)

MAR 5 1971

55.12

1745-2 DISPLAY DRIVER (BUFFERED)

INITIATOR



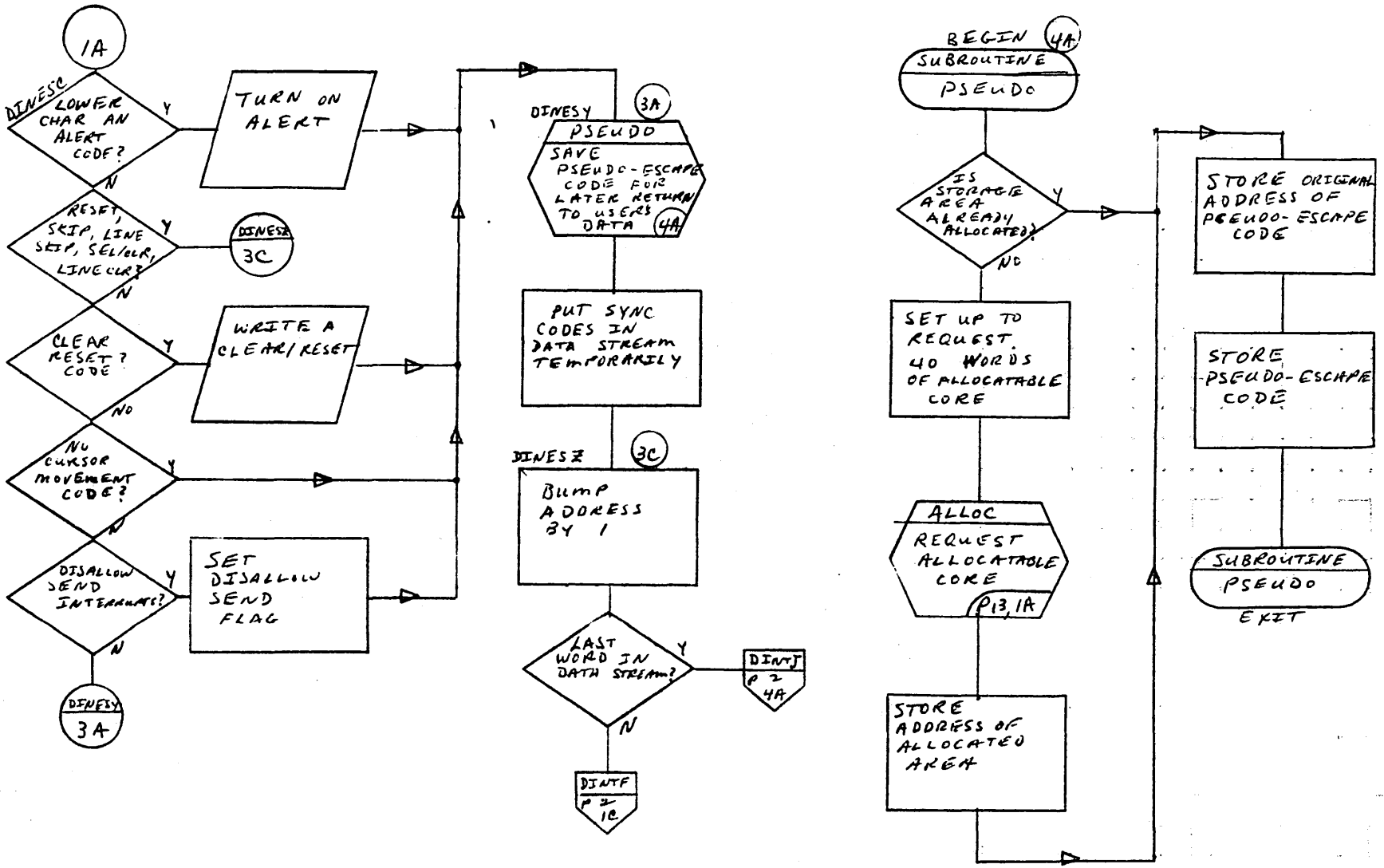
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 3 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

55.13

1745-2 DISPLAY DRIVER (BUFFERED)
INITIATOR

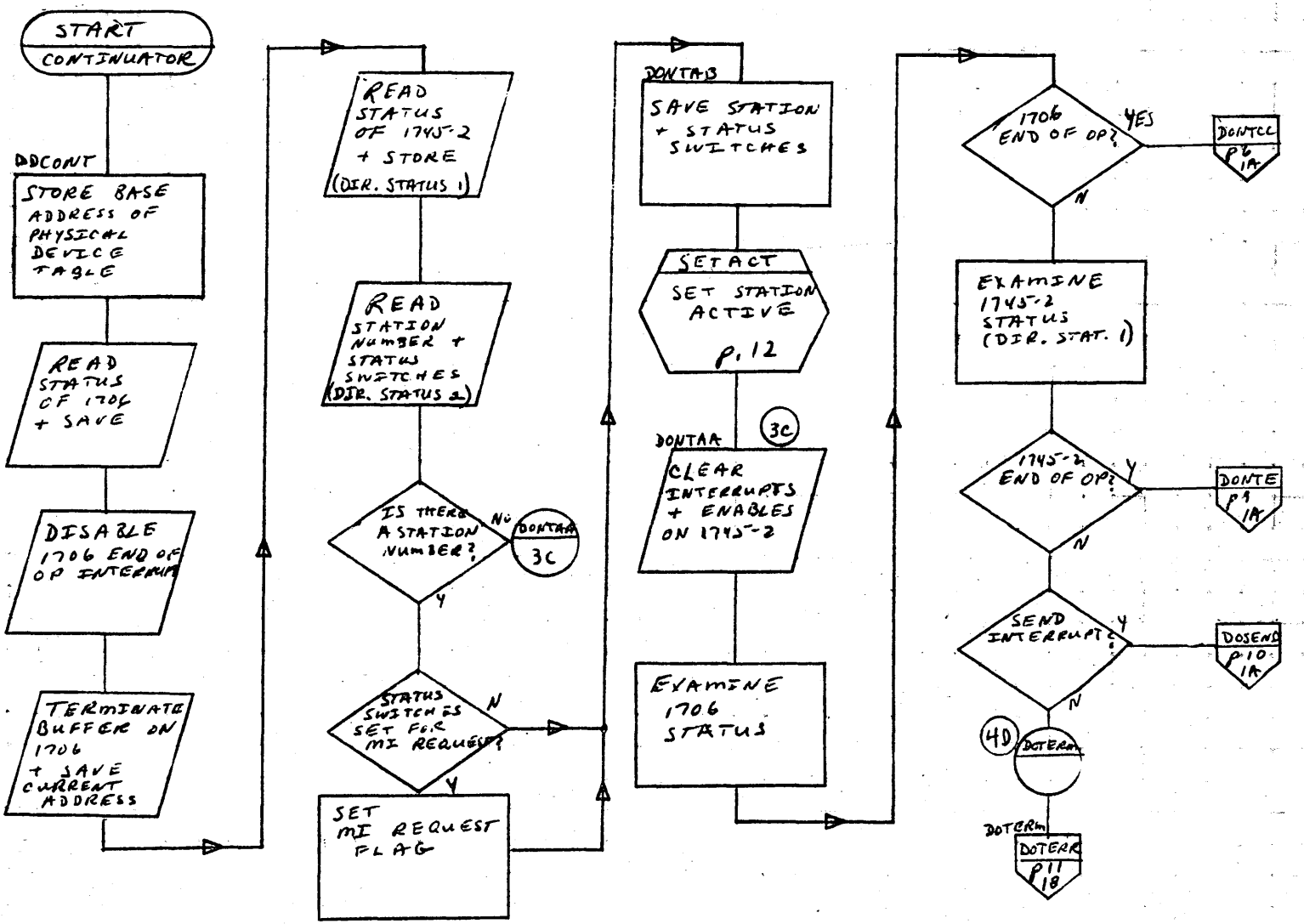


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 4 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55.14

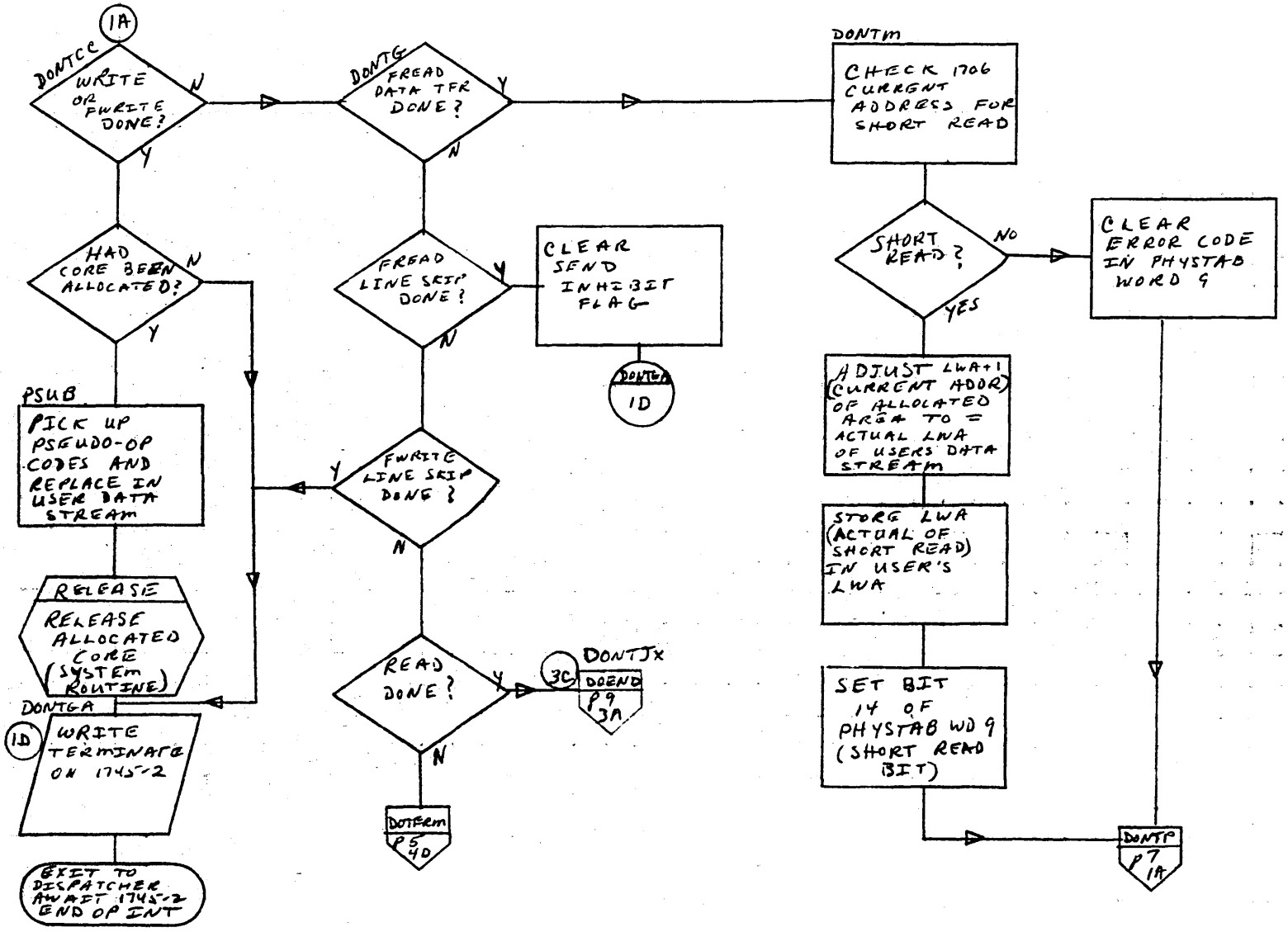
1745-2 DISPLAY DRIVER (BUFFERED)
CONTINUATOR



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 6 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

1745-2 DISPLA, DRIVER (BUFFERED)
CONTINUATOR



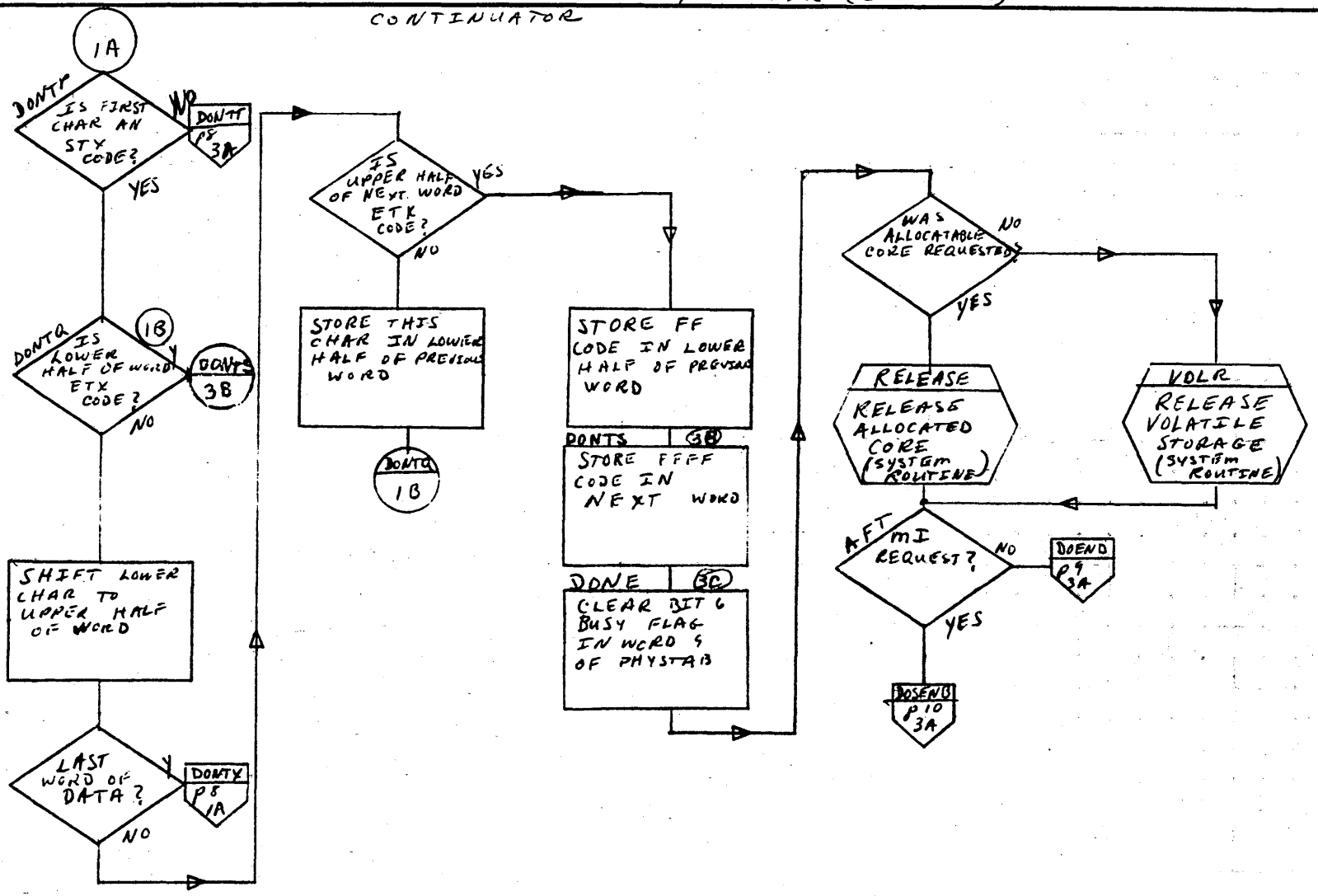
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 6 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55.19

1745-2 DISPLAY DRIVER (BUFFERED)

CONTINUATOR



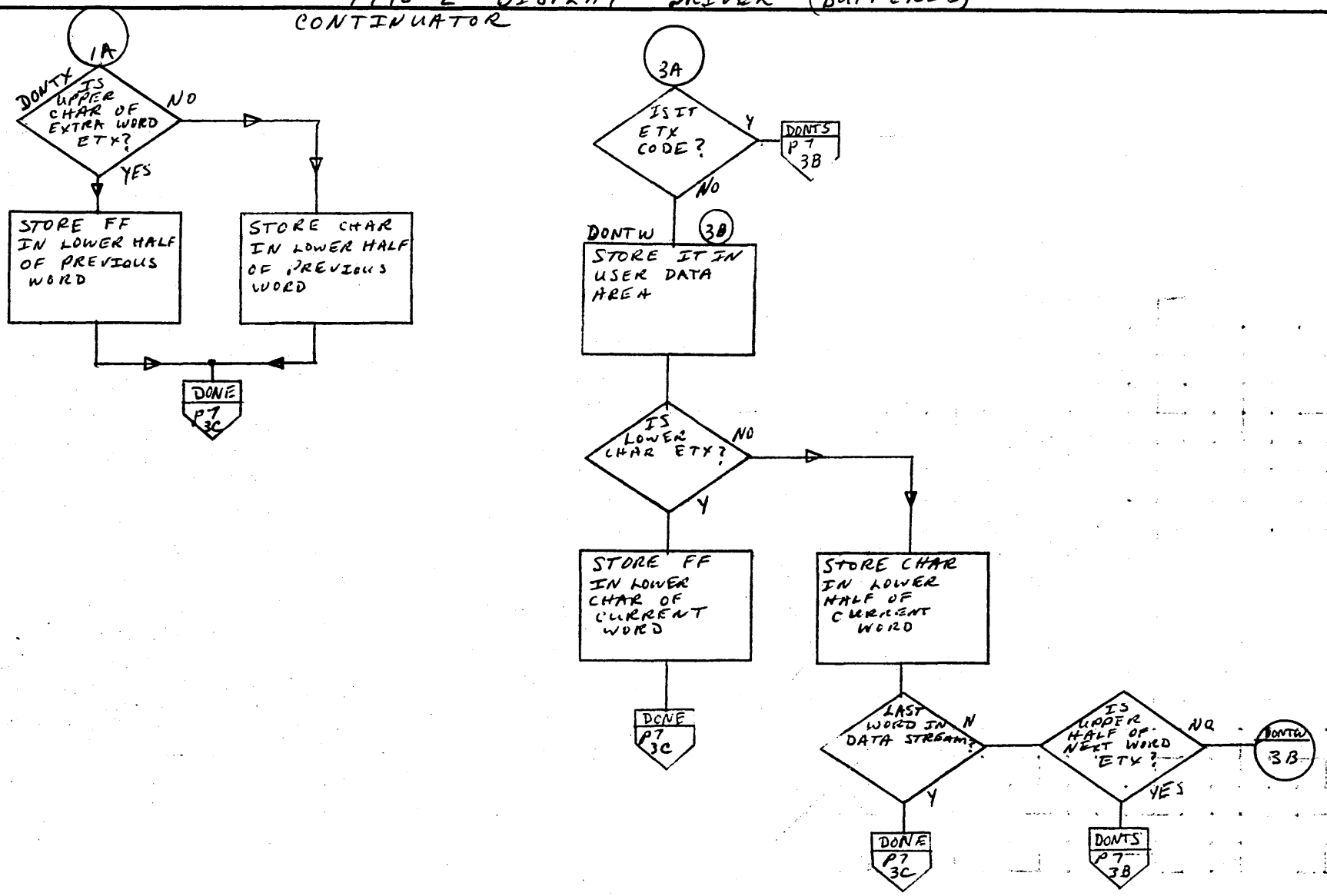
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 7 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55-17

AA1385 (FORMERLY CA127-1)

1745-2 DISPLAY DRIVER (BUFFERED)
CONTINUATOR

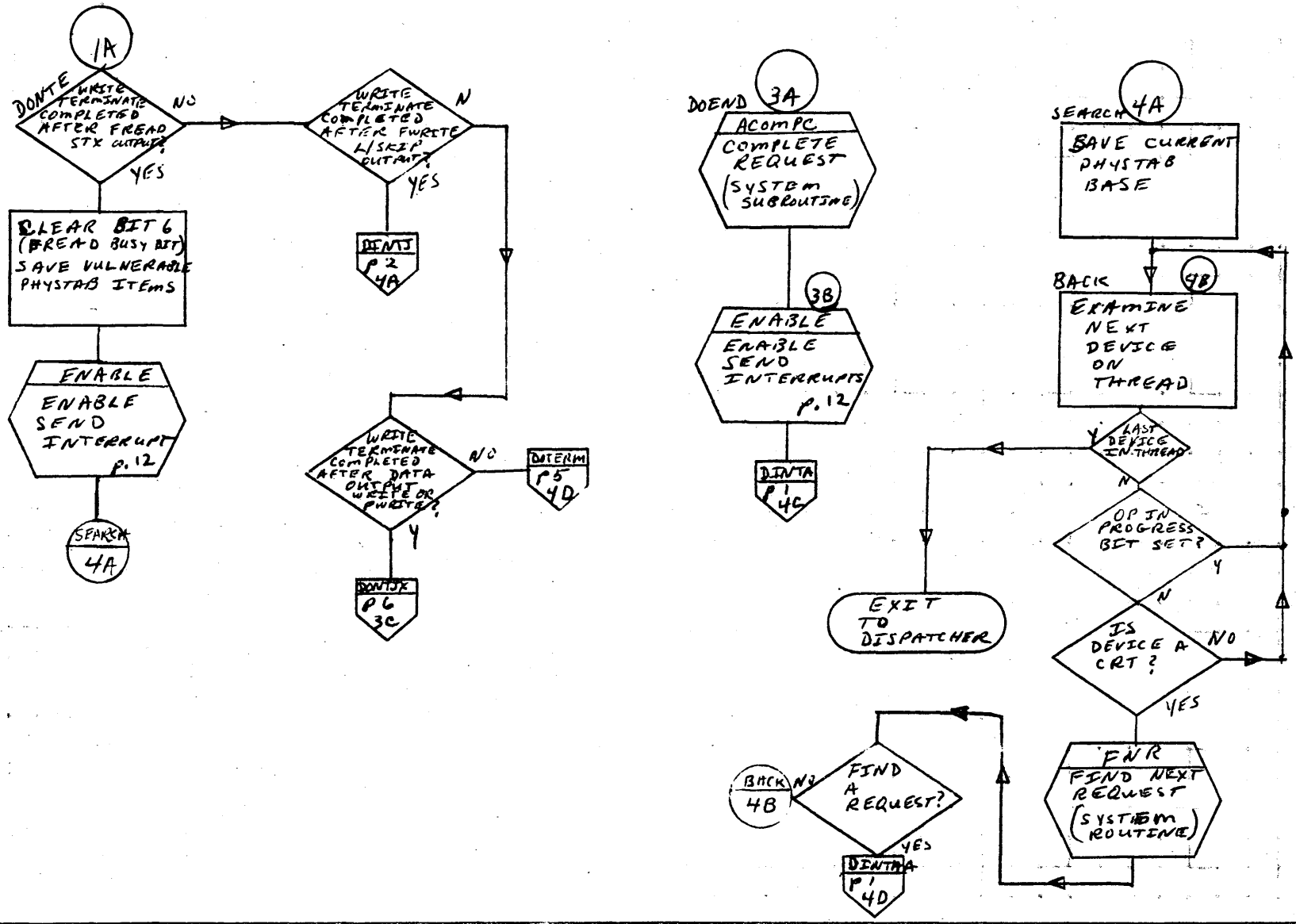


CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 6 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55-18

1745-2 DISPLA, DRIVER (BUFFERED)
CONTINUATOR



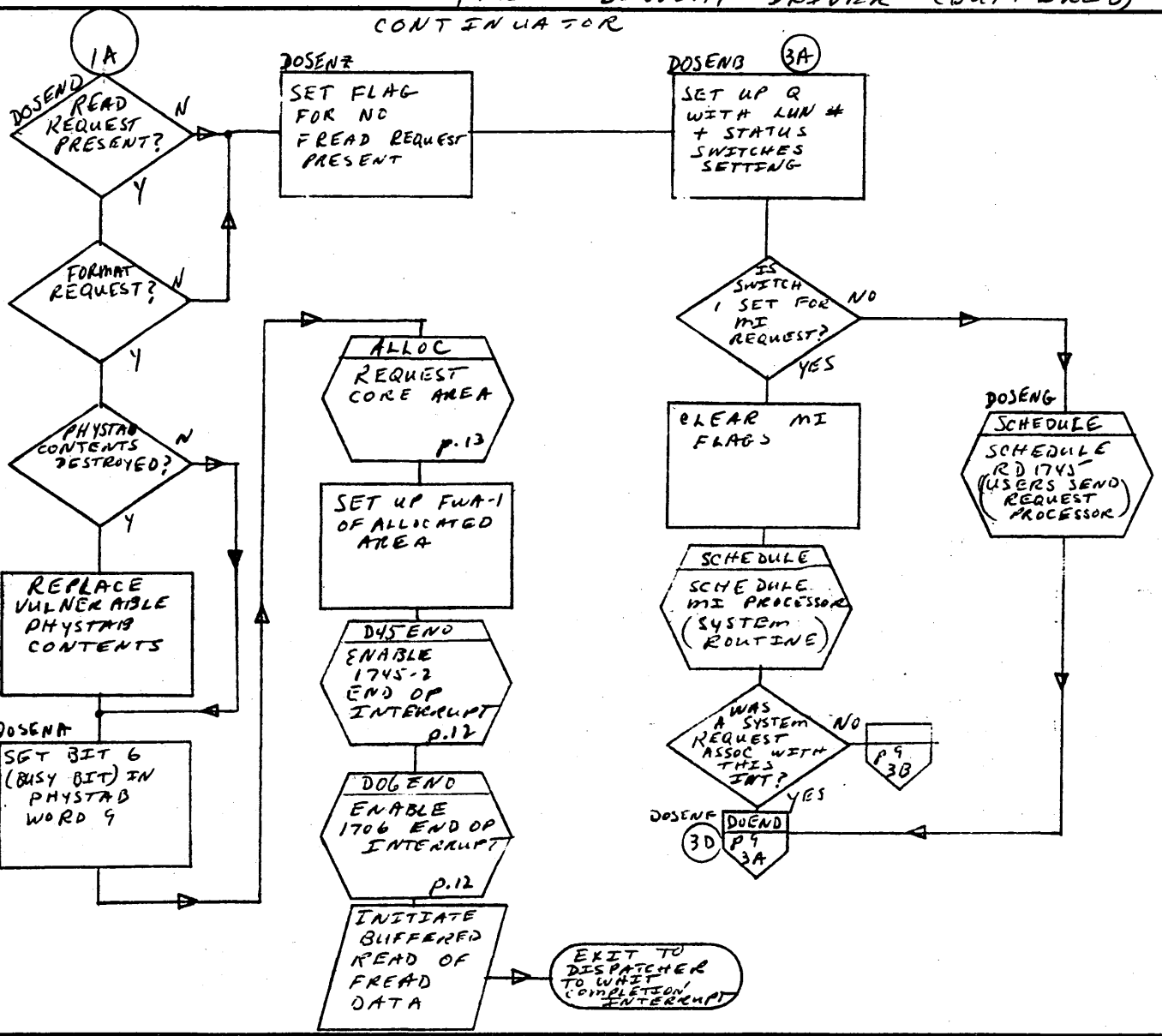
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☑ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 9 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55.29

1745-2 DISPLAY DRIVER (BUFFERED)

CONTINUATOR



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 10 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

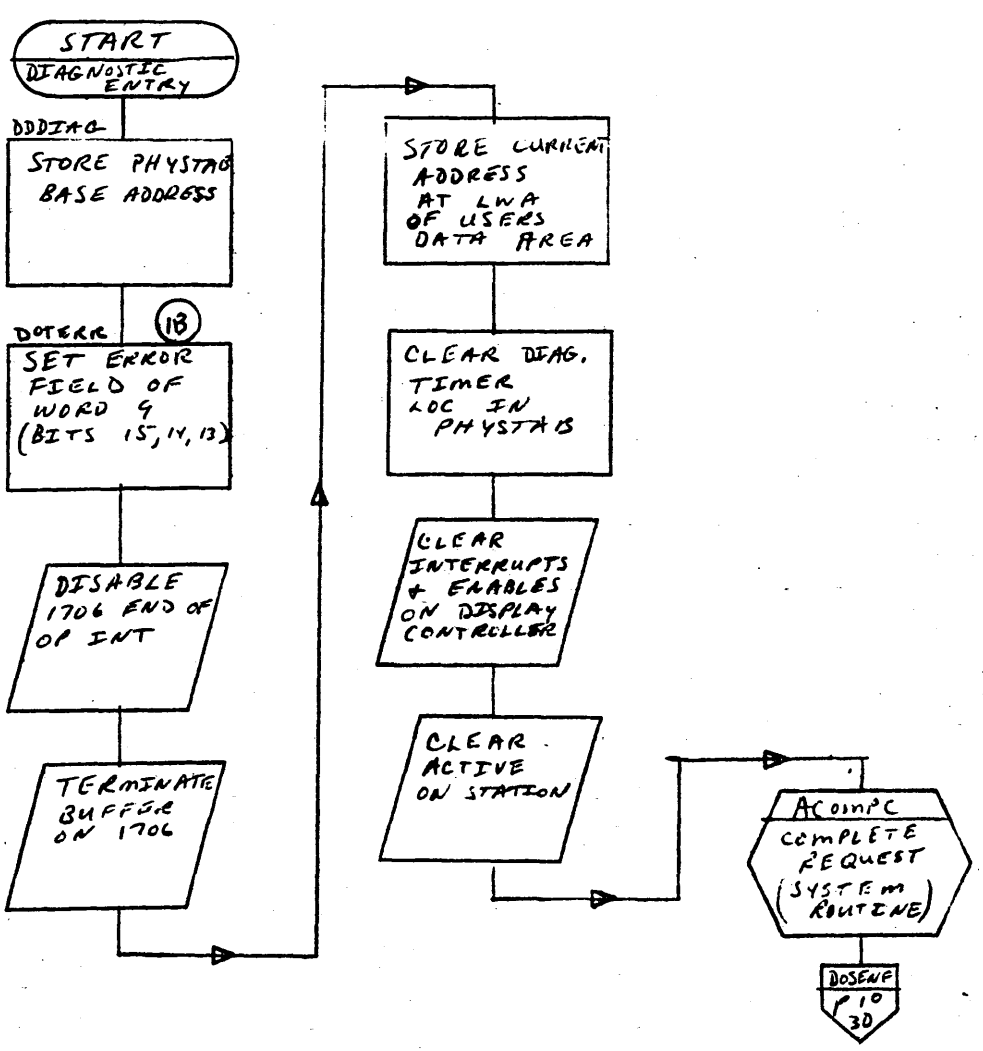
MAR 5 1971

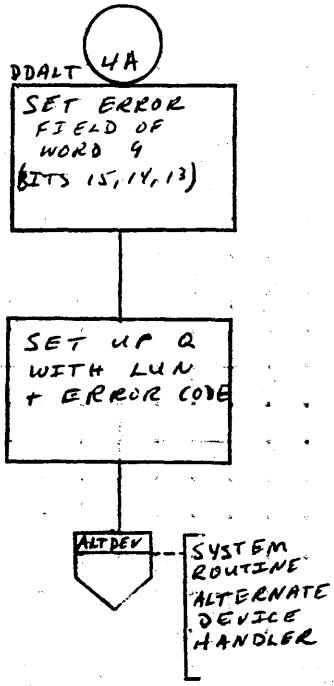
55-20

AA1385 (FORMERLY CA127-1)

1745.2 DISPLAY DRIVER (BUFFERED)

DIAGNOSTIC ENTRY





CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 11 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

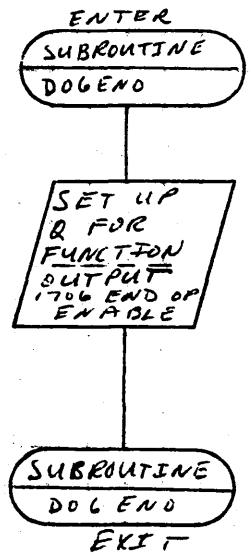
AA1385 (FORMERLY CA127-1)

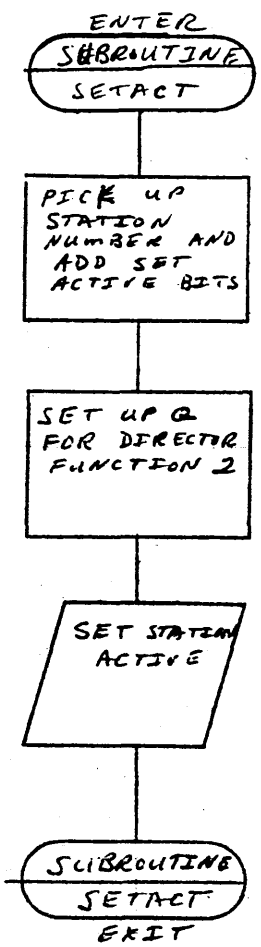
MAR 5 1971

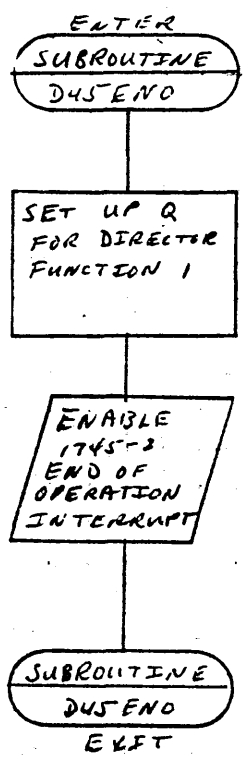
55.21

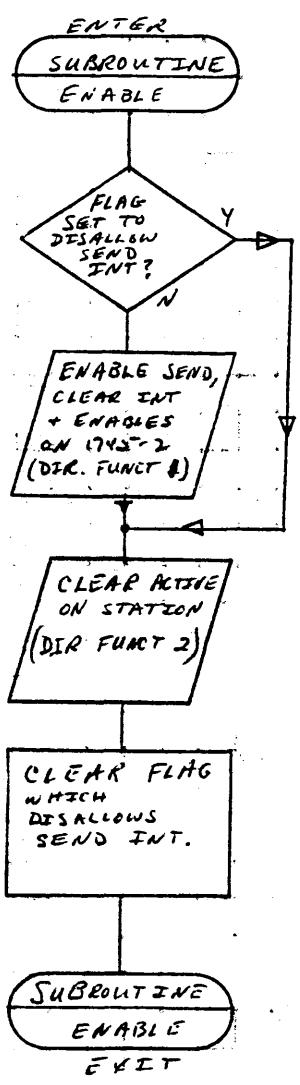
1745-2 DISPLAY DRIVER (BUFFERED)

SUBROUTINES







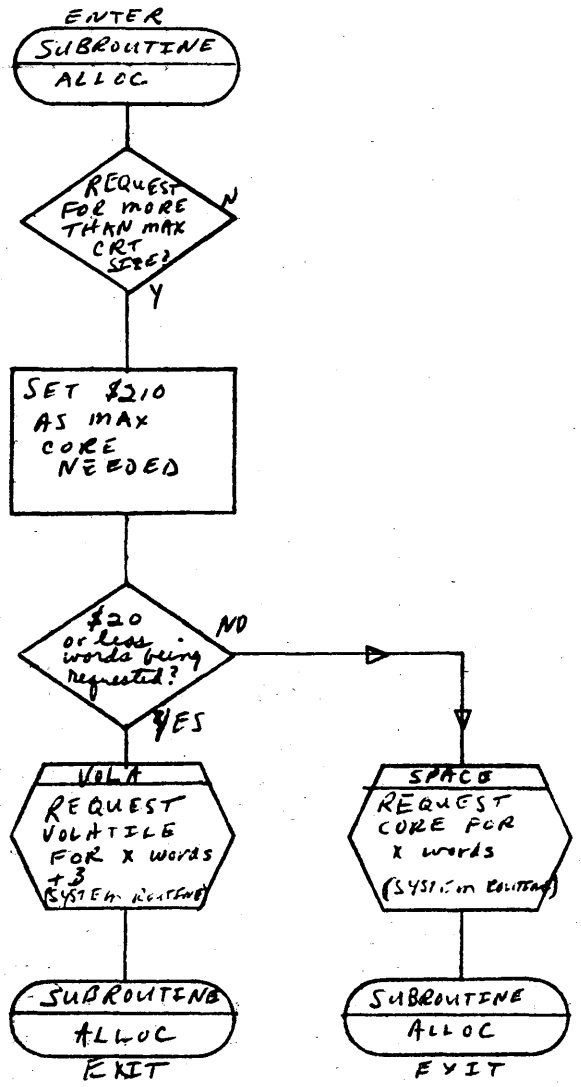


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒		PAGE 12 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

55.22

1745-2 DISPLAY DRIVER (BUFFERED)
SUBROUTINES



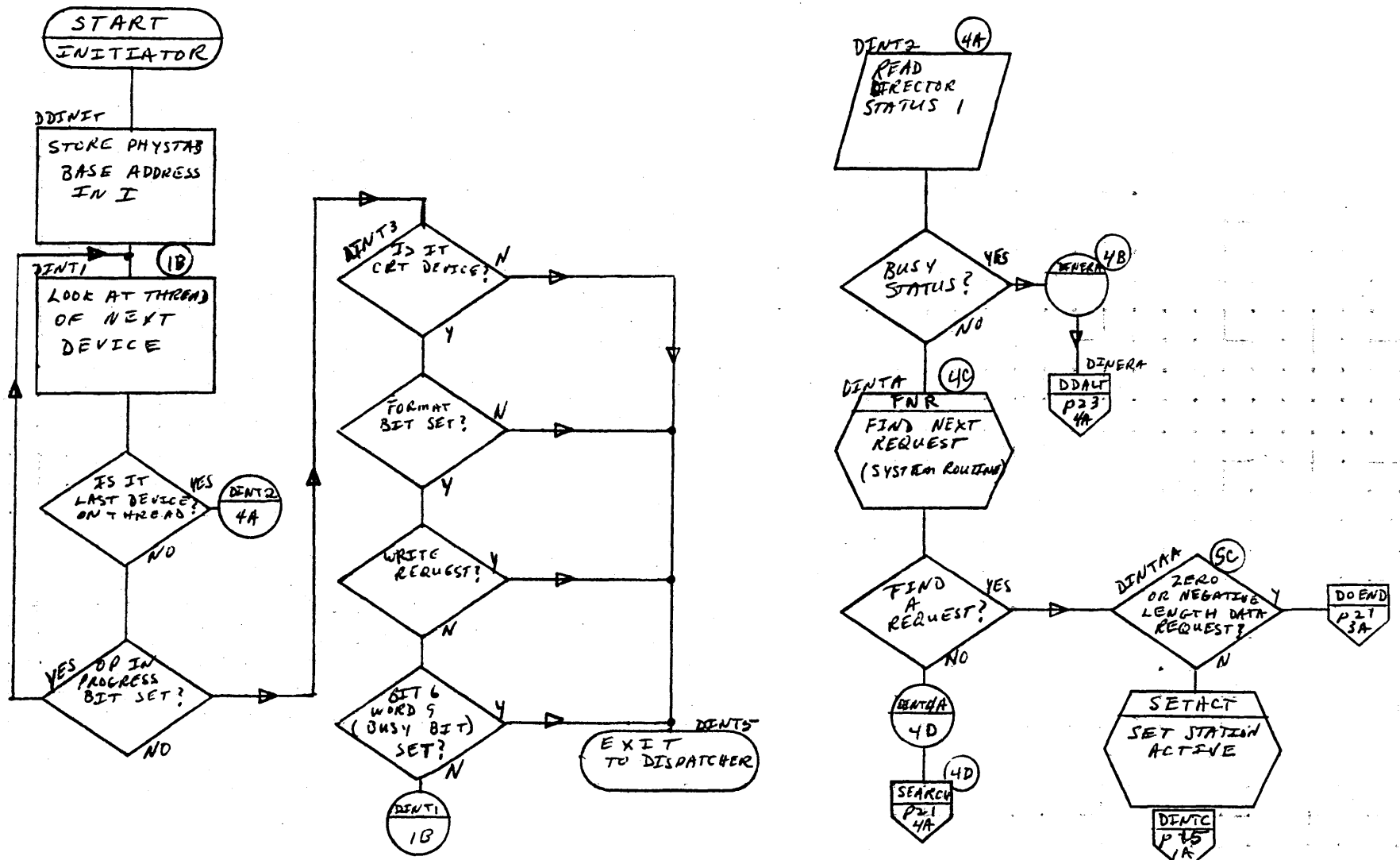
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 13 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

MAR 5 1971

55.23

1745-2 DISPLAY DRIVER (UNBUFFERED)
INITIATOR



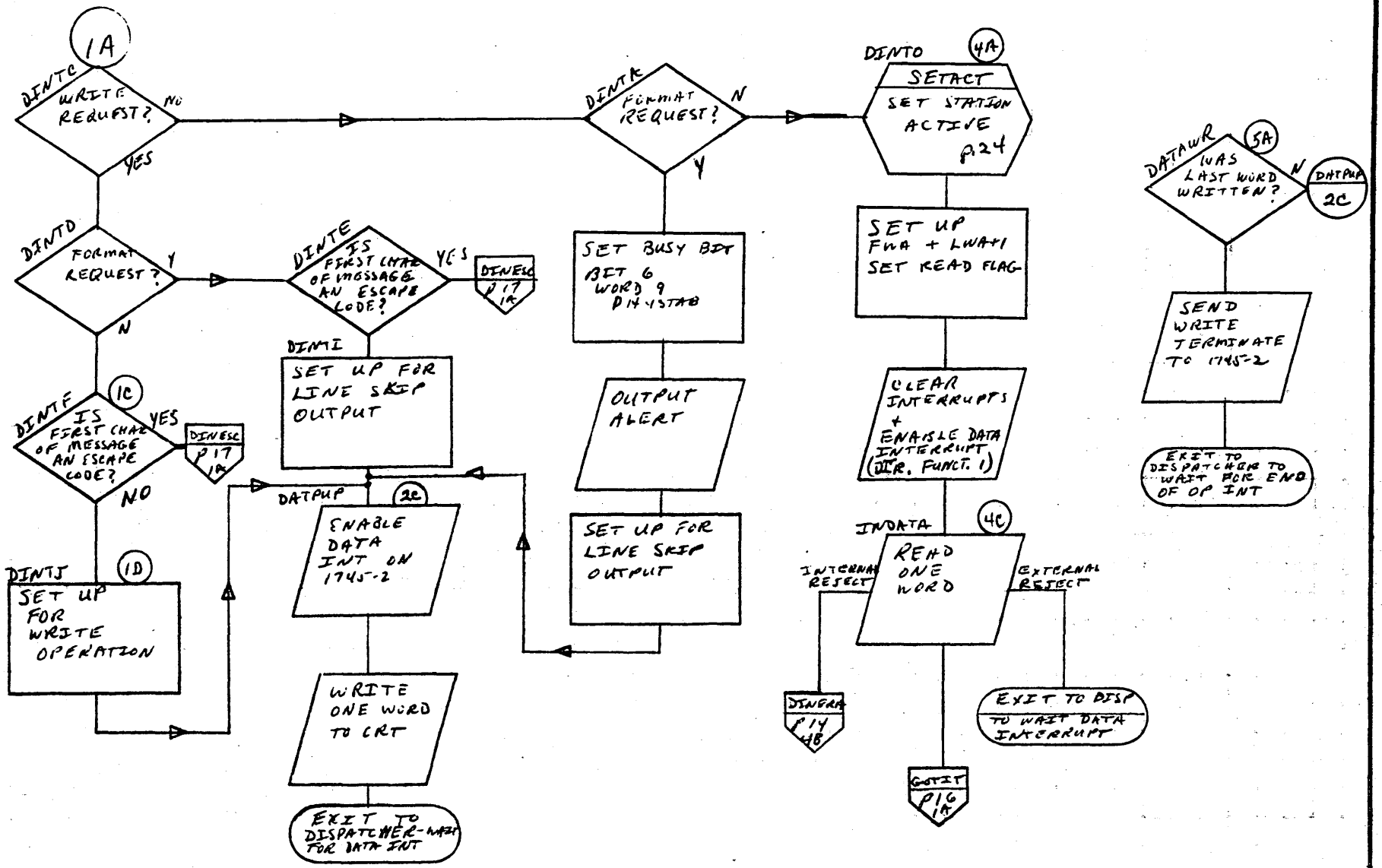
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 1 OF 4	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55.24

AA1385 (FORMERLY CA127-1)

1745-2 DISPLAY DR. ER (UNBUFFERED) INITIATOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 5 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

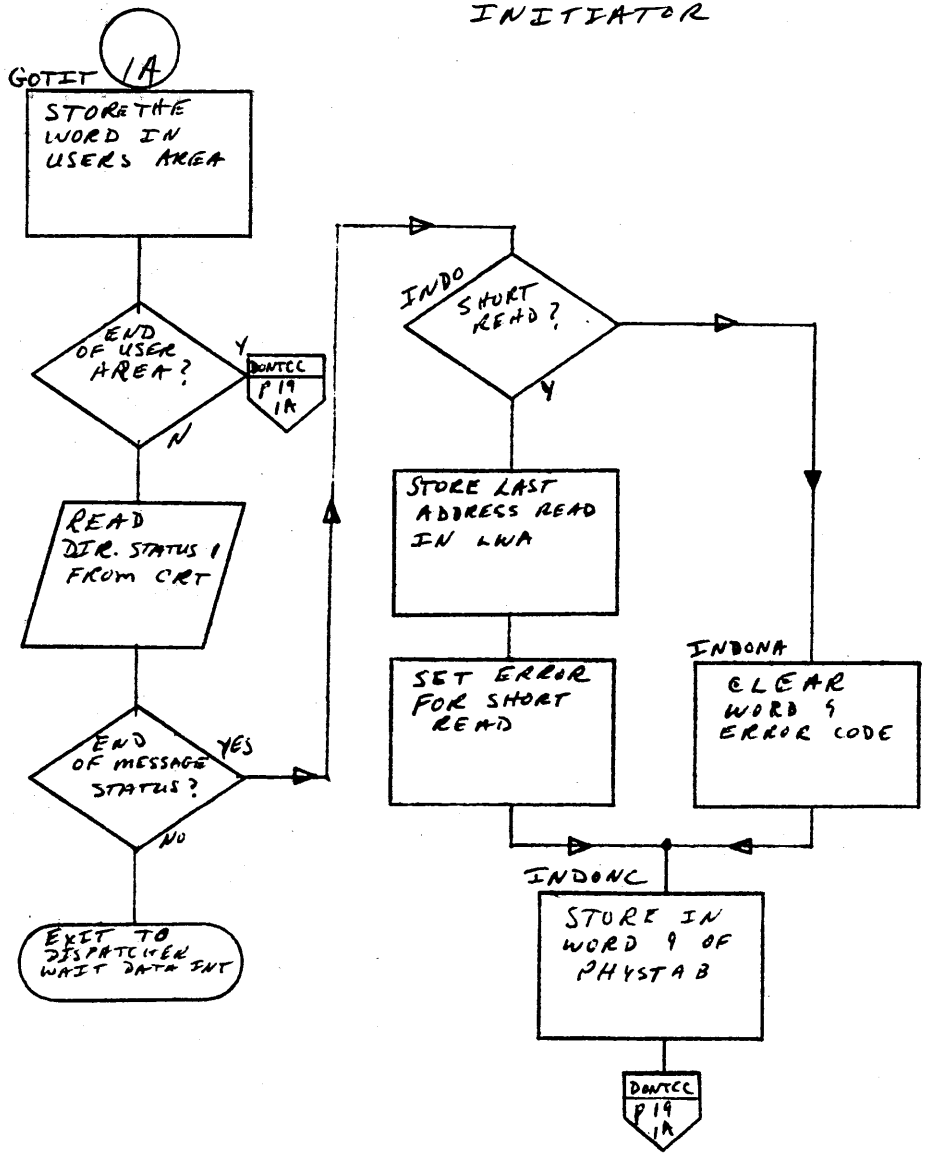
AA1385 (FORMERLY CA127-1)

MAR 6 1971

55-25

1745-2 DISPLAY DRIVER (UNBUFFERED)

INITIATOR

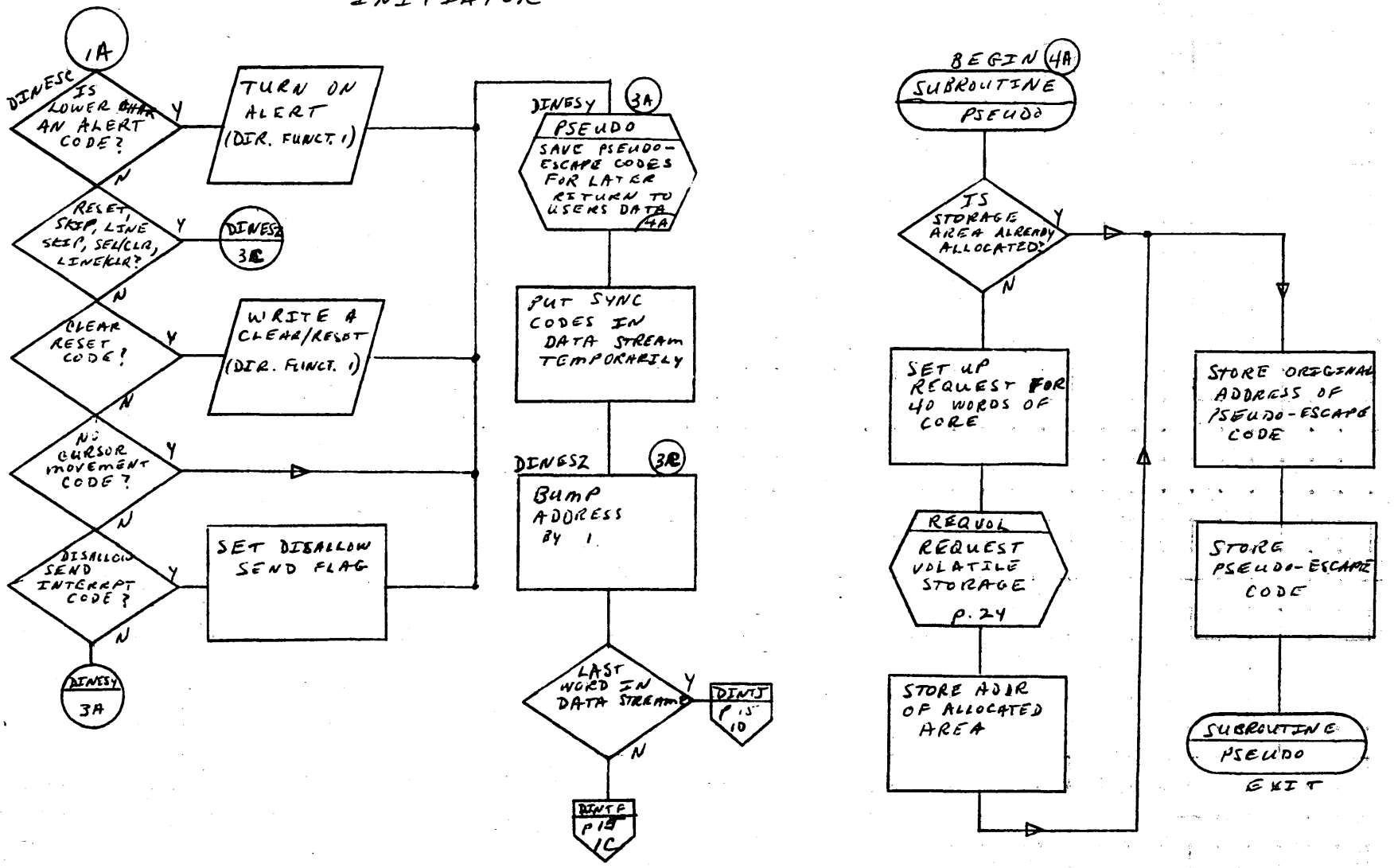


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 16 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

1745-2 DISPL / DRIVER (UNBUFFERED)

INITIATOR



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 17 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

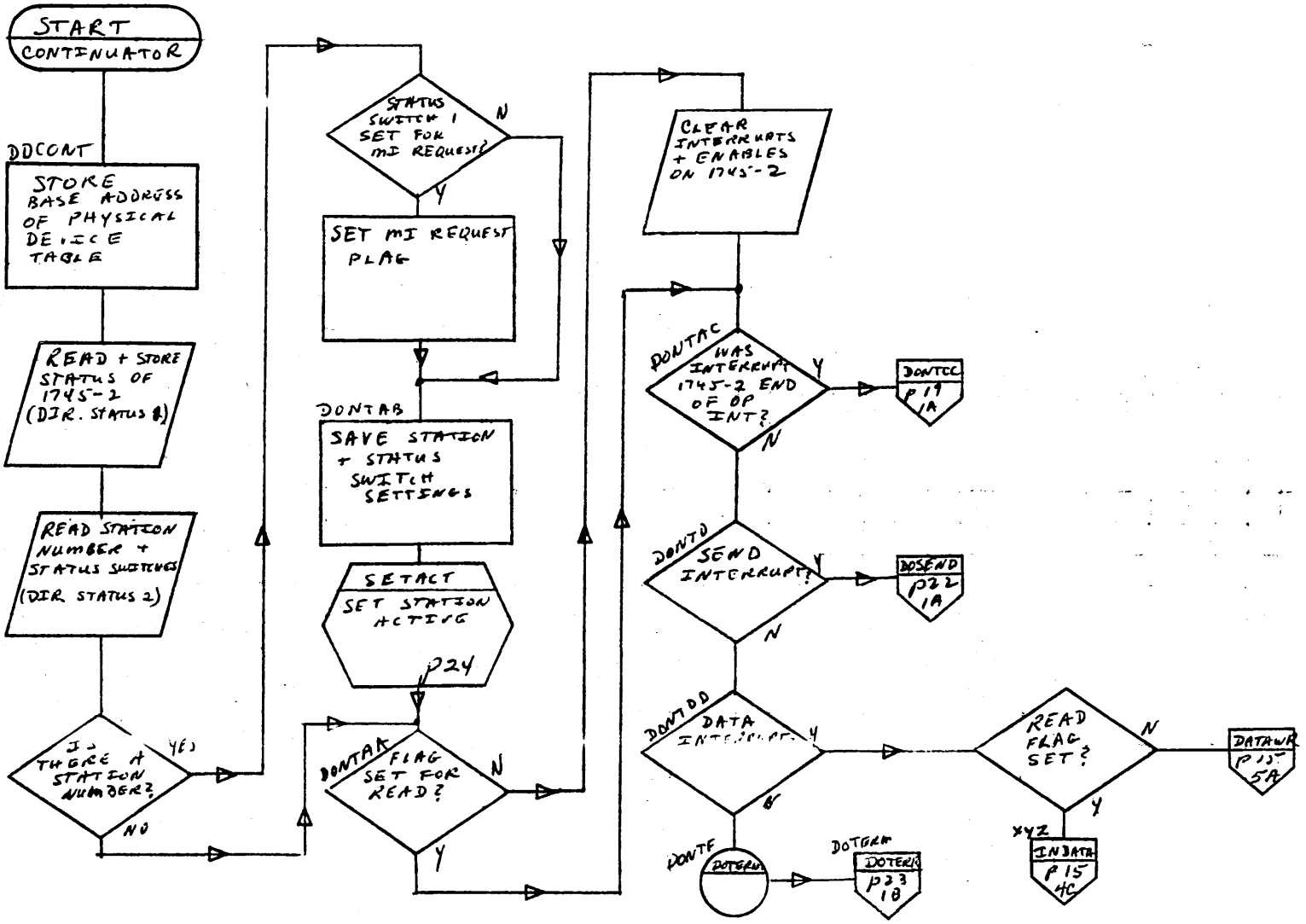
AA1385 (FORMERLY CA127-1)

MAR 5 1971

55.27

1745-2 DISPLAY DRIVER (UNBUFFERED)

CONTINUATOR



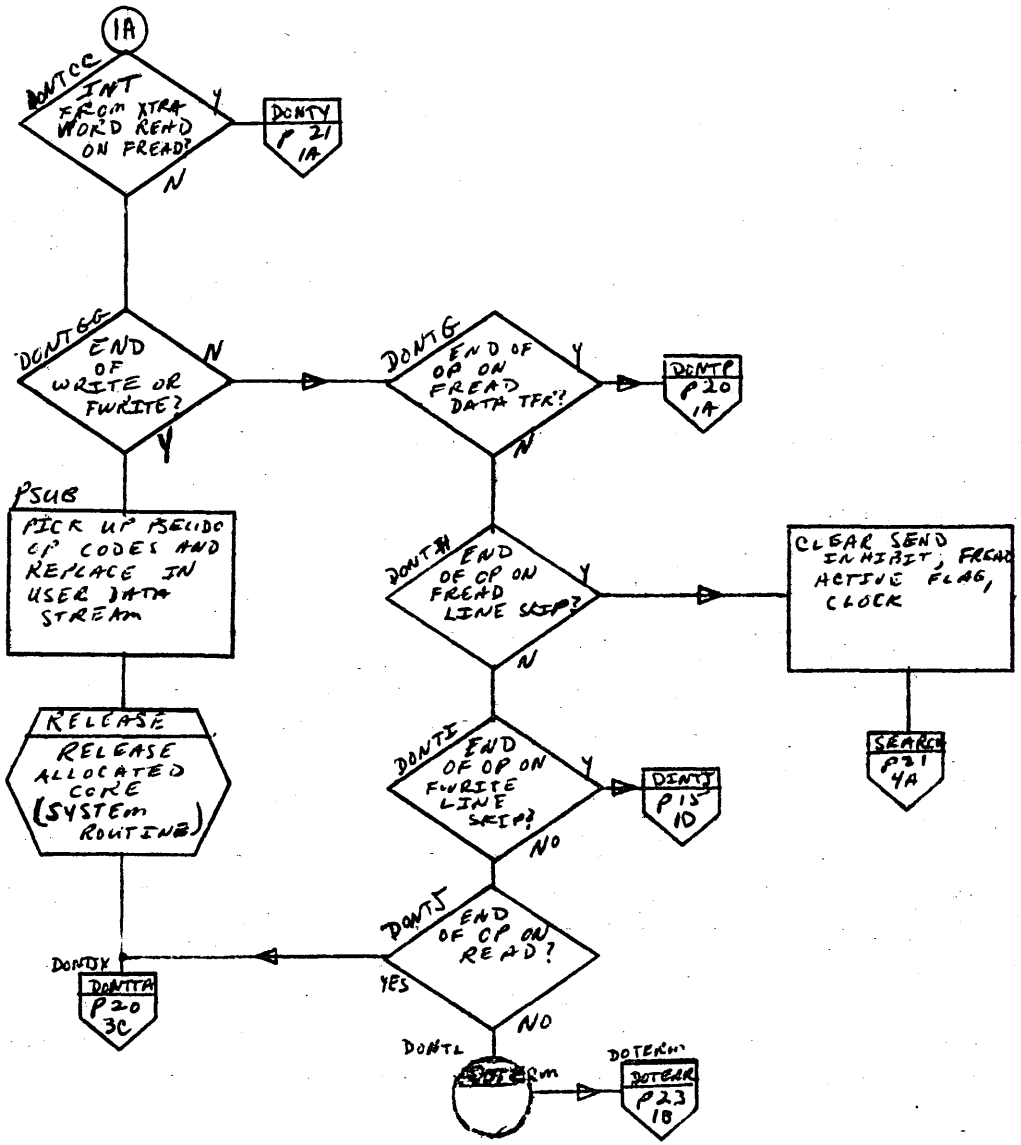
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 18 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55.28

AA1385 (FORMERLY CA127-1)

1745.2 DISPLAY DRIVER (UNBUFFERED)
CONTINUATOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 19 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

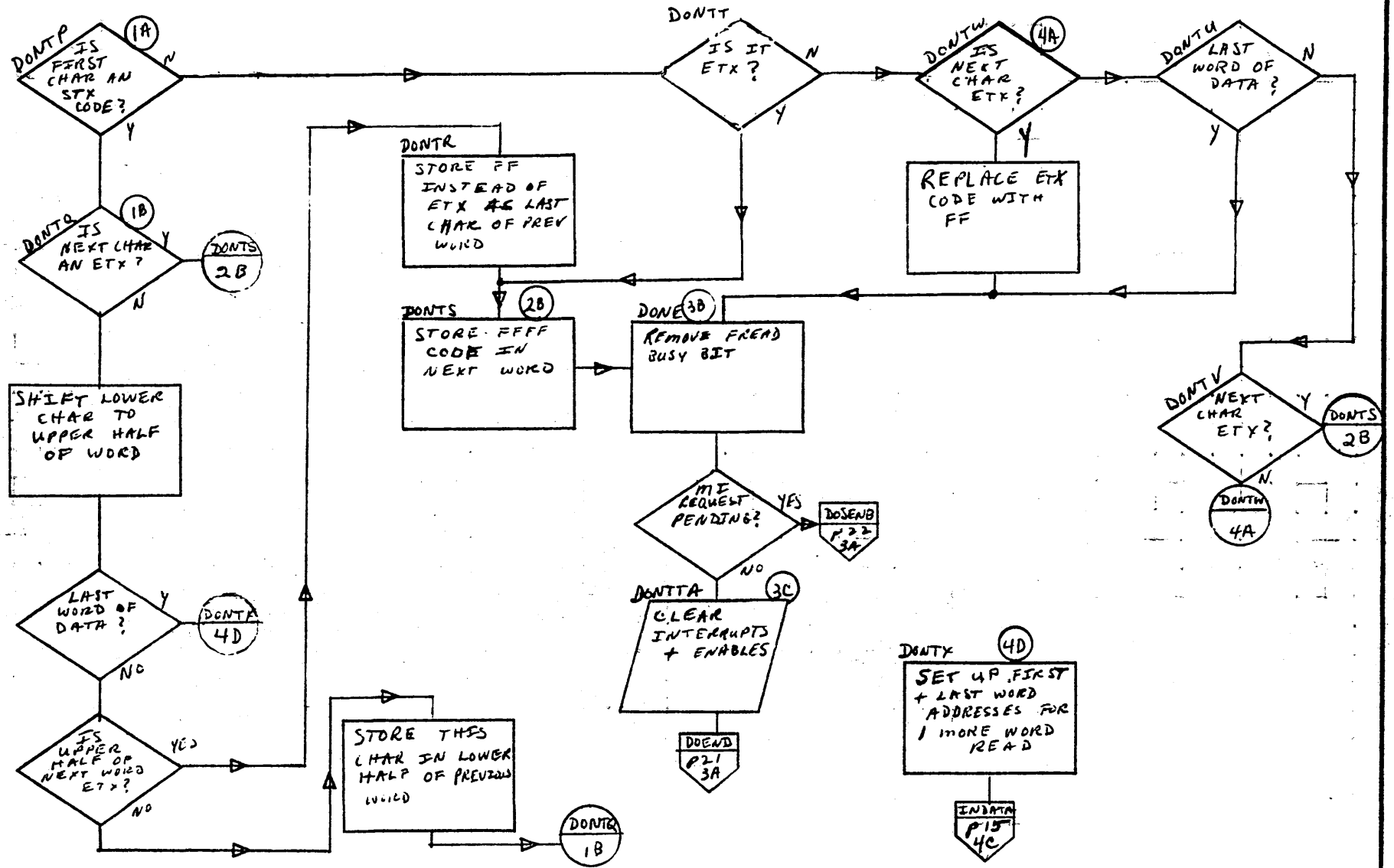
AA1385 (FORMERLY CA127-1)

MAR 5 1971

55.29

1745-2 DISPLAY DRIVER (UNBUFFERED)

CONTINUATOR



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 20 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

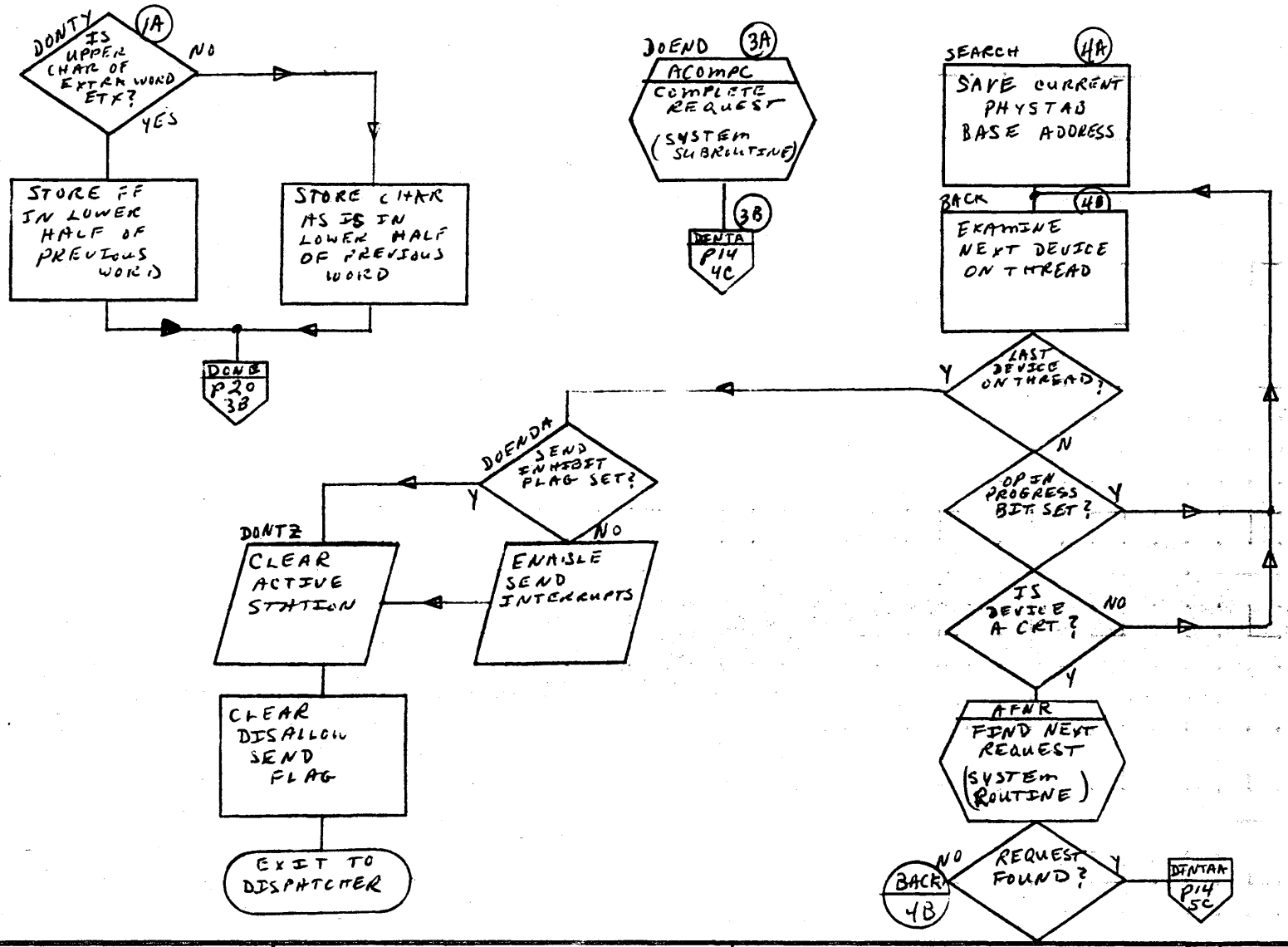
MAR 5 1971

55.30

AA1385 (FORMERLY CA127-1)

1745-2 DISPIA, DRIVER (UNBUFFERED)

CONTINUATOR



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 21 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

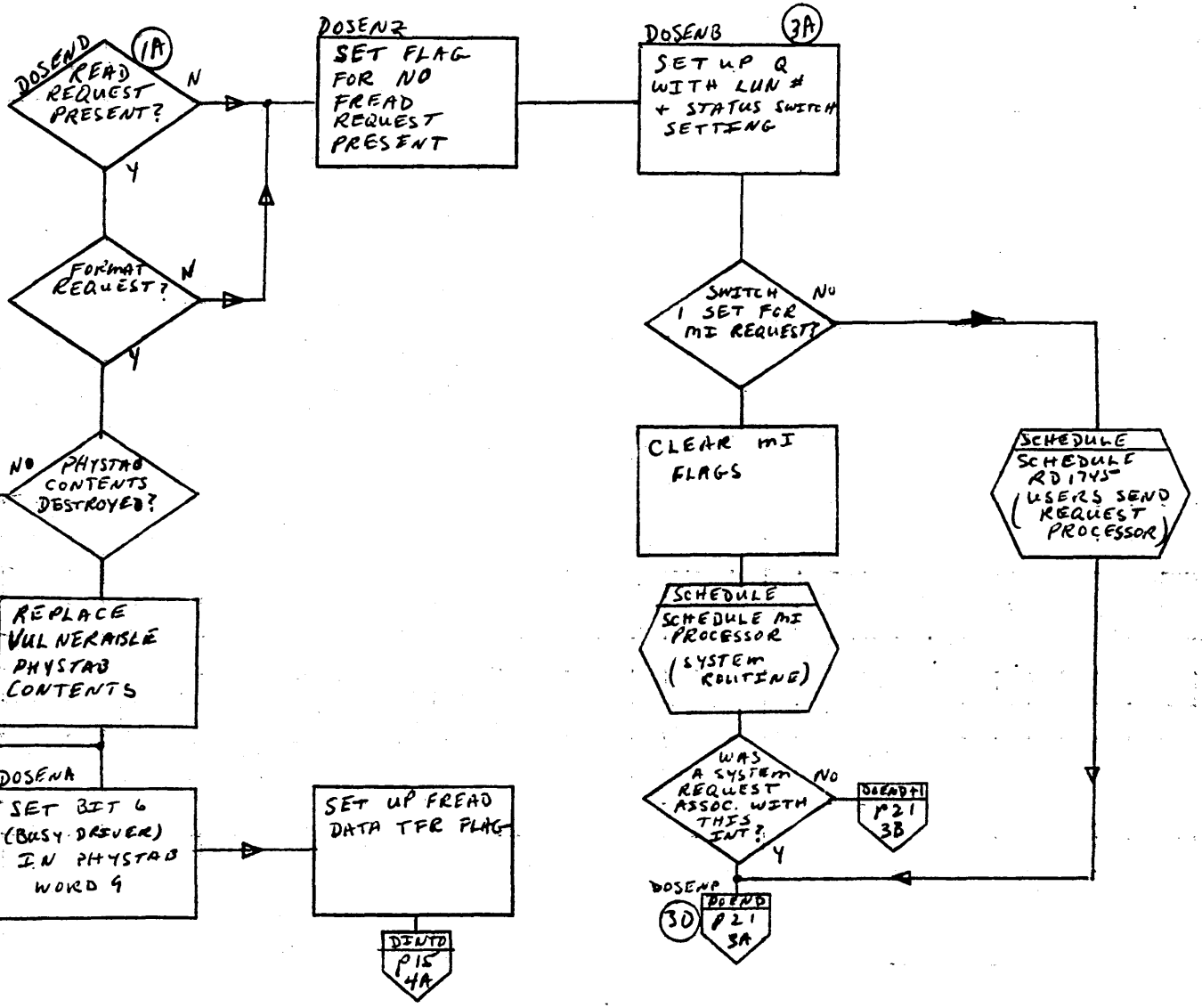
AA1385 (FORMERLY CA127-1)

MAR 5 1971

55.31

1745-2 DISPL. I DRIVER (UNBUFFERED)

CONTINUATOR



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 22 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

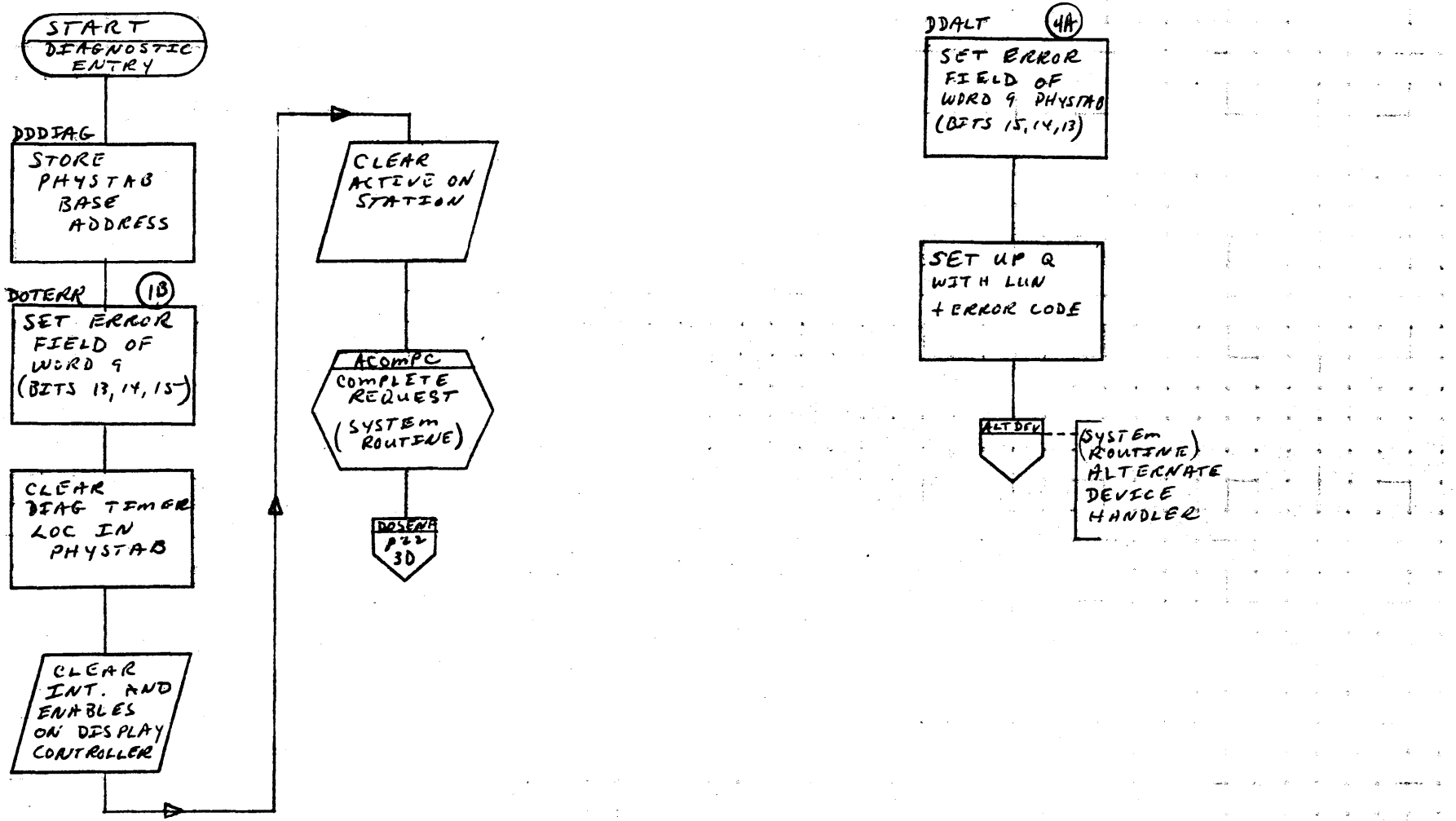
MAR 5 1971

55.32

AA1385 (FORMERLY CA127-1)

1745.2 DISPLAY DRIVER (UNBUFFERED)

DIAGNOSTIC ENTRY

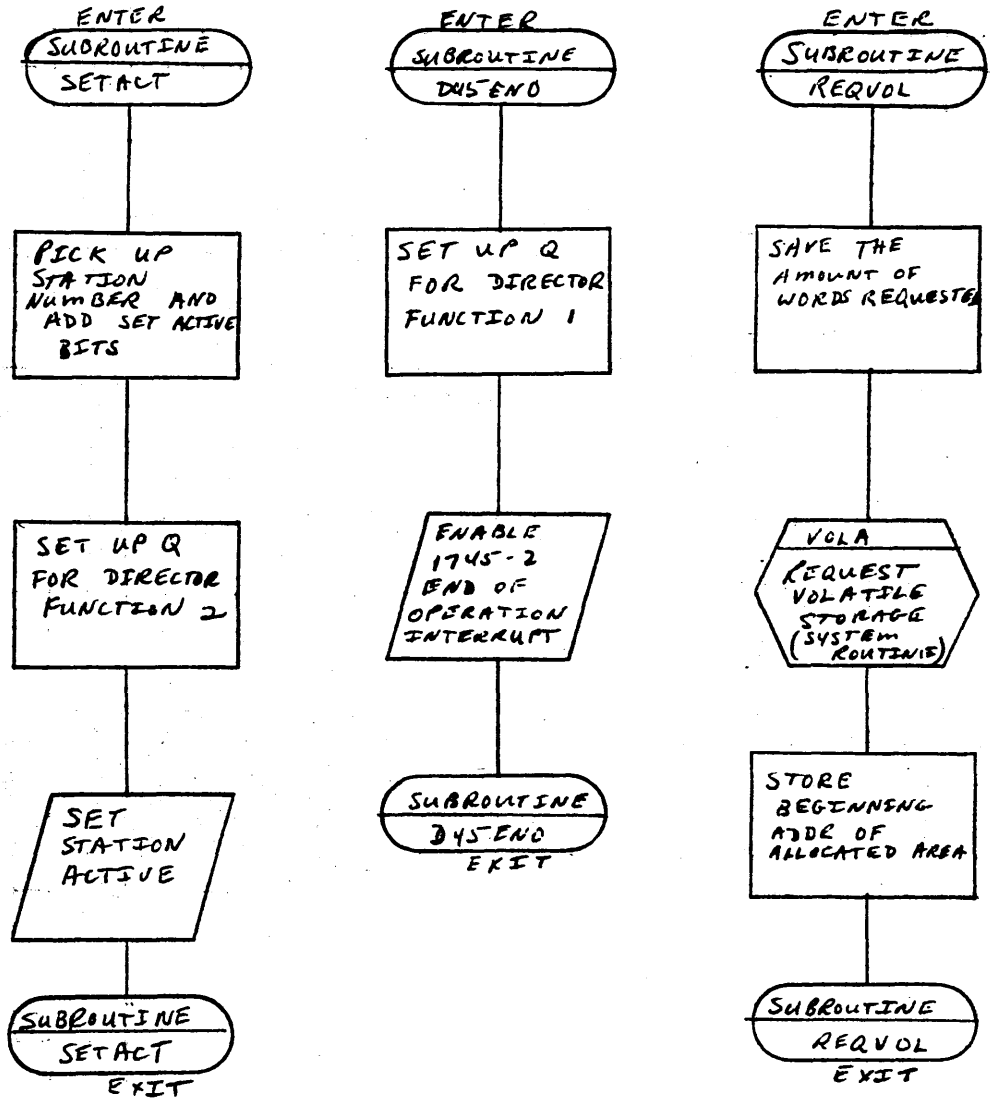


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
		PAGE 23 OF	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

55.33

1745-2 DISPLAY DRIVER (UNBUFFERED) SUBROUTINES



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 24 OF	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

55.34

AA1385 (FORMERLY CA127-1)

56.0 CYFT - Cosy Format

56.1 Function

CYFT is used to build a hollerith tape suitable for input to COSY. CYFT inserts the appropriate DCK/, HOL/ and END/ cards.

56.2 Entry Point Names

CYFT

56.3 Externals

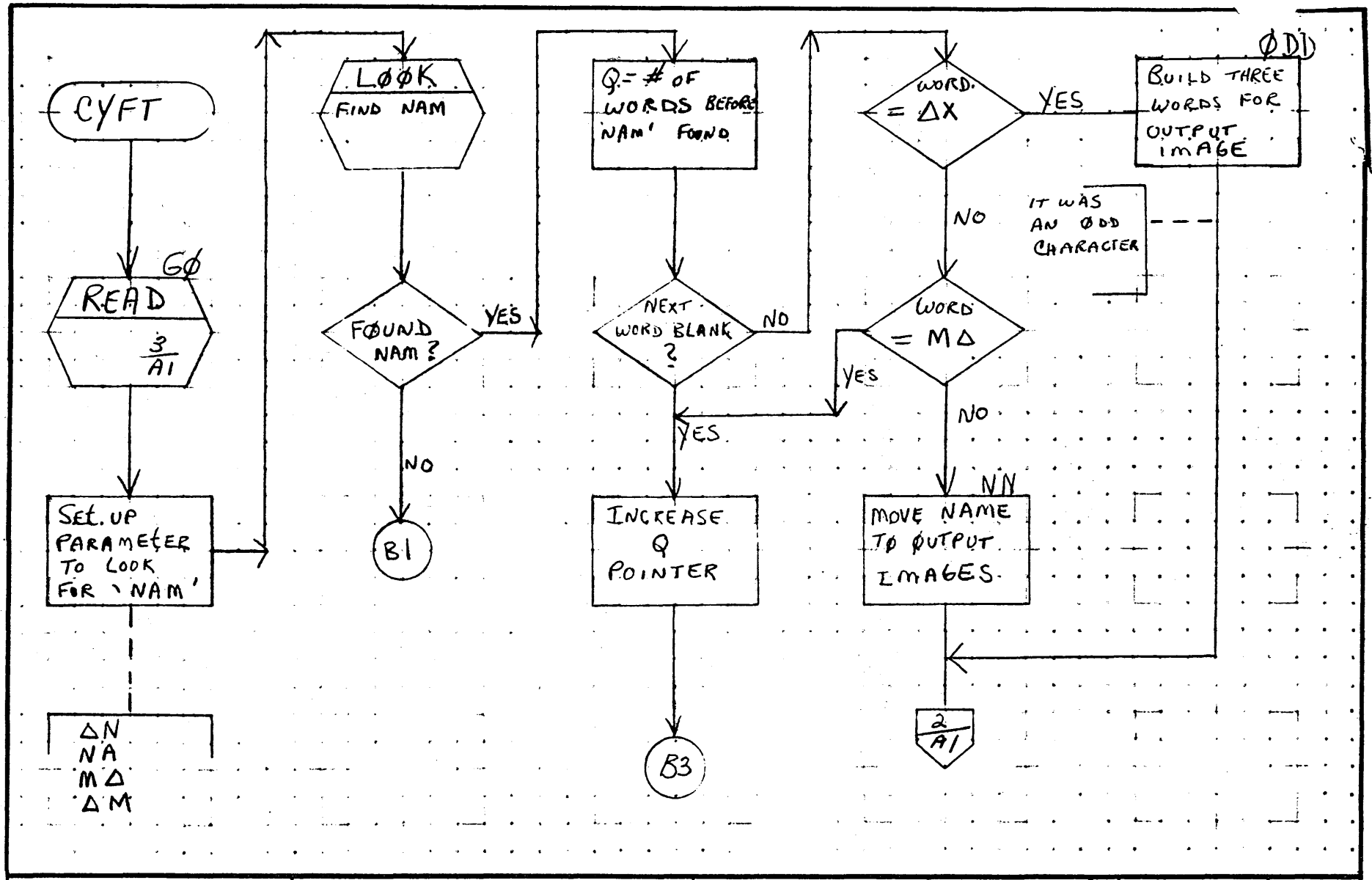
None

56.4 Entry Interfaces

CYFT is a program library program called by a ✖ entry point statement.

56.5 Exit Interfaces

None

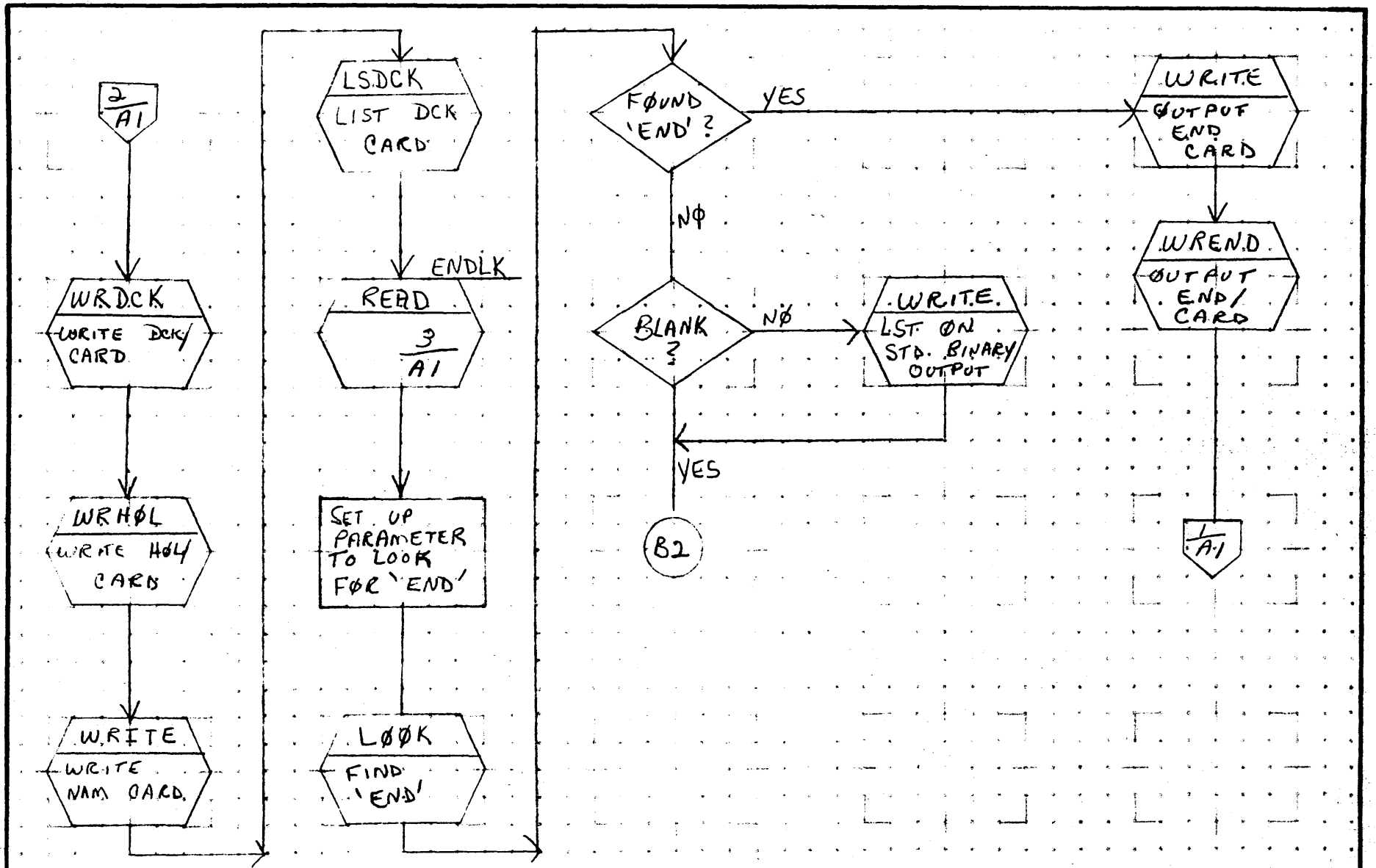


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CYFT		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



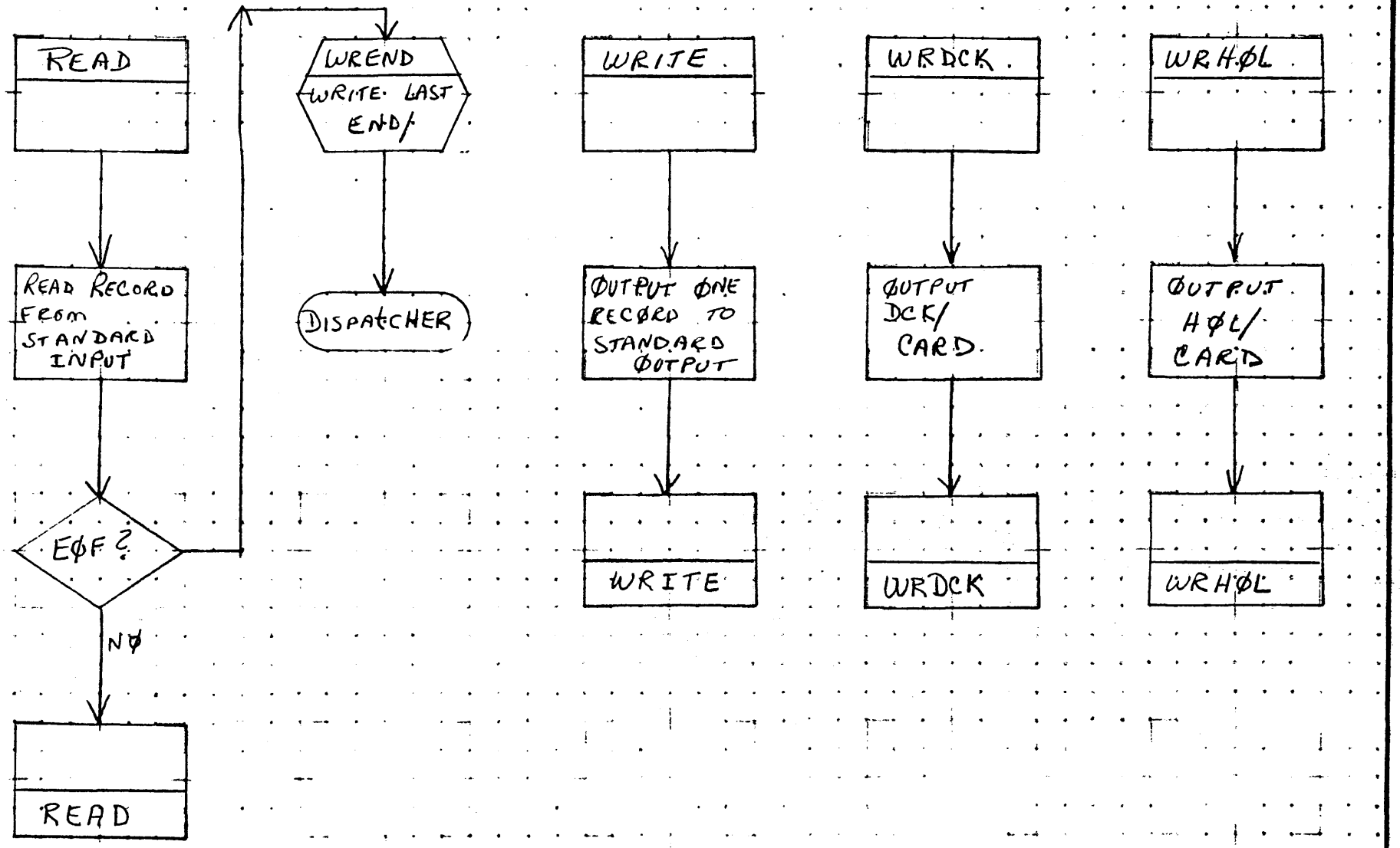
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CYFT		PROJECT MGR.		
	MSOS 3.0	PAGE 2 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

59.3

AA1385 (FORMERLY CA127-1)

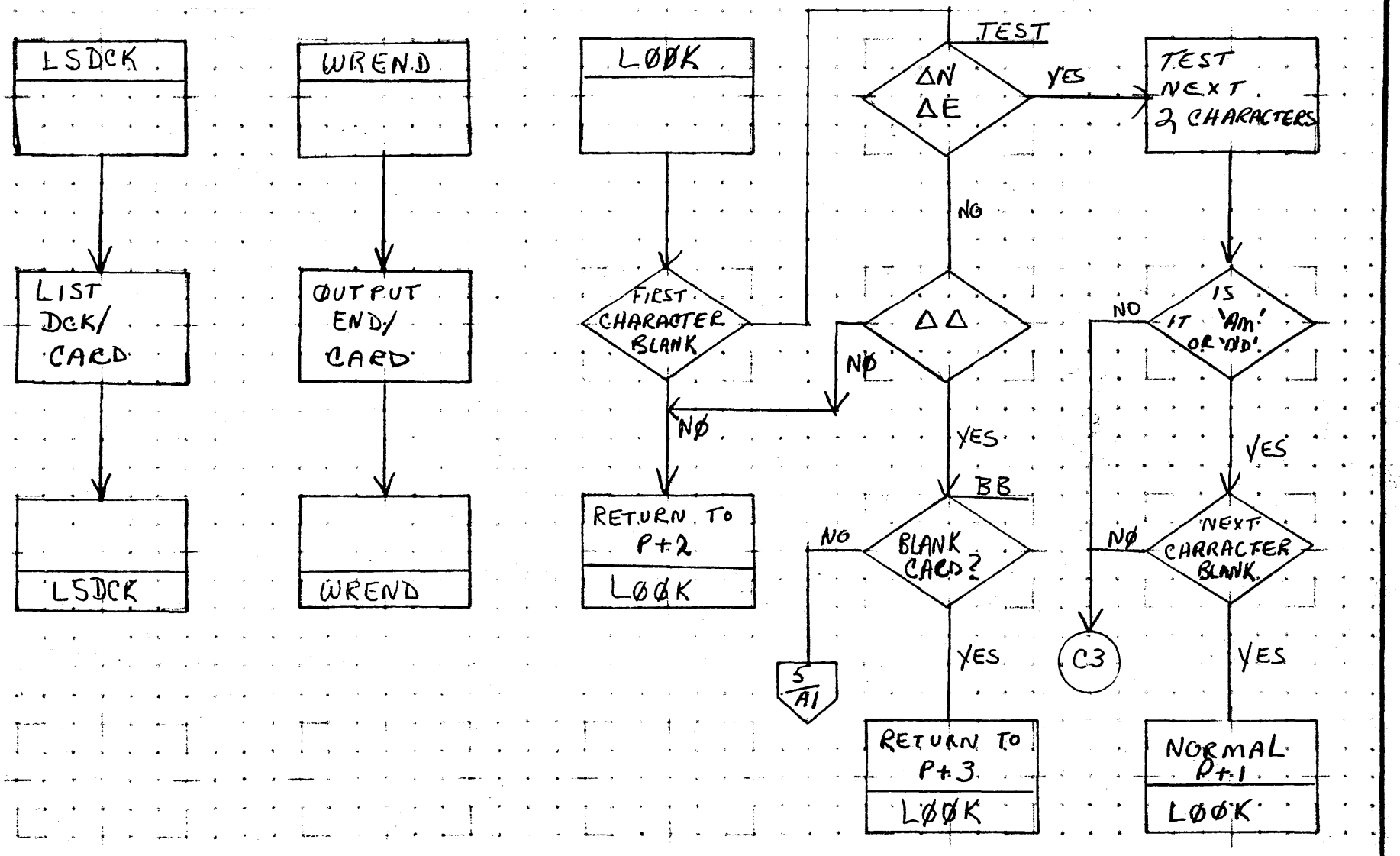
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CYFT		PROJECT MGR.		
	MSOS 3.0	PAGE 3 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

56.4



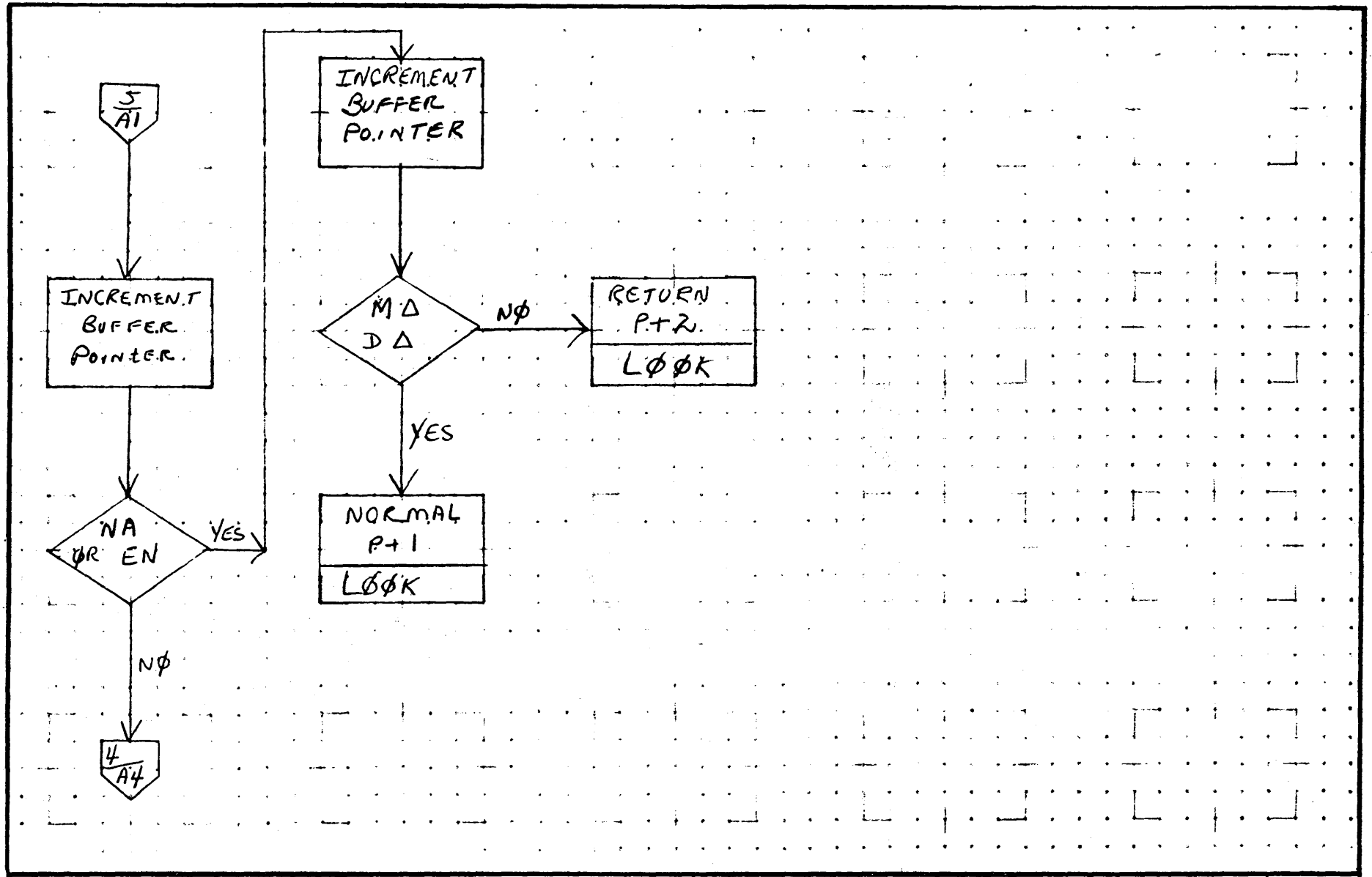
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE CYFT		PROJECT MGR.		
	MSOS 3.0	PAGE 4 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

56.5



CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE CYFT	PROJECT MGR.	
	MSØS 3.0 PAGE 5 OF 5	PROJECT NAME	
	NUMBER ISSUE DATE	TASK NO.	
	DRAWN BY DATE	TASK NAME	

MAR 5 1971

56.6

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

57.0 LCOSY

57.1 Function

LCOSY will list cosy deck names and punch deck cards on the assigned punch device. Input terminates with a cosy END/.

57.2 Entry Points

LCOSY

57.3 Externals

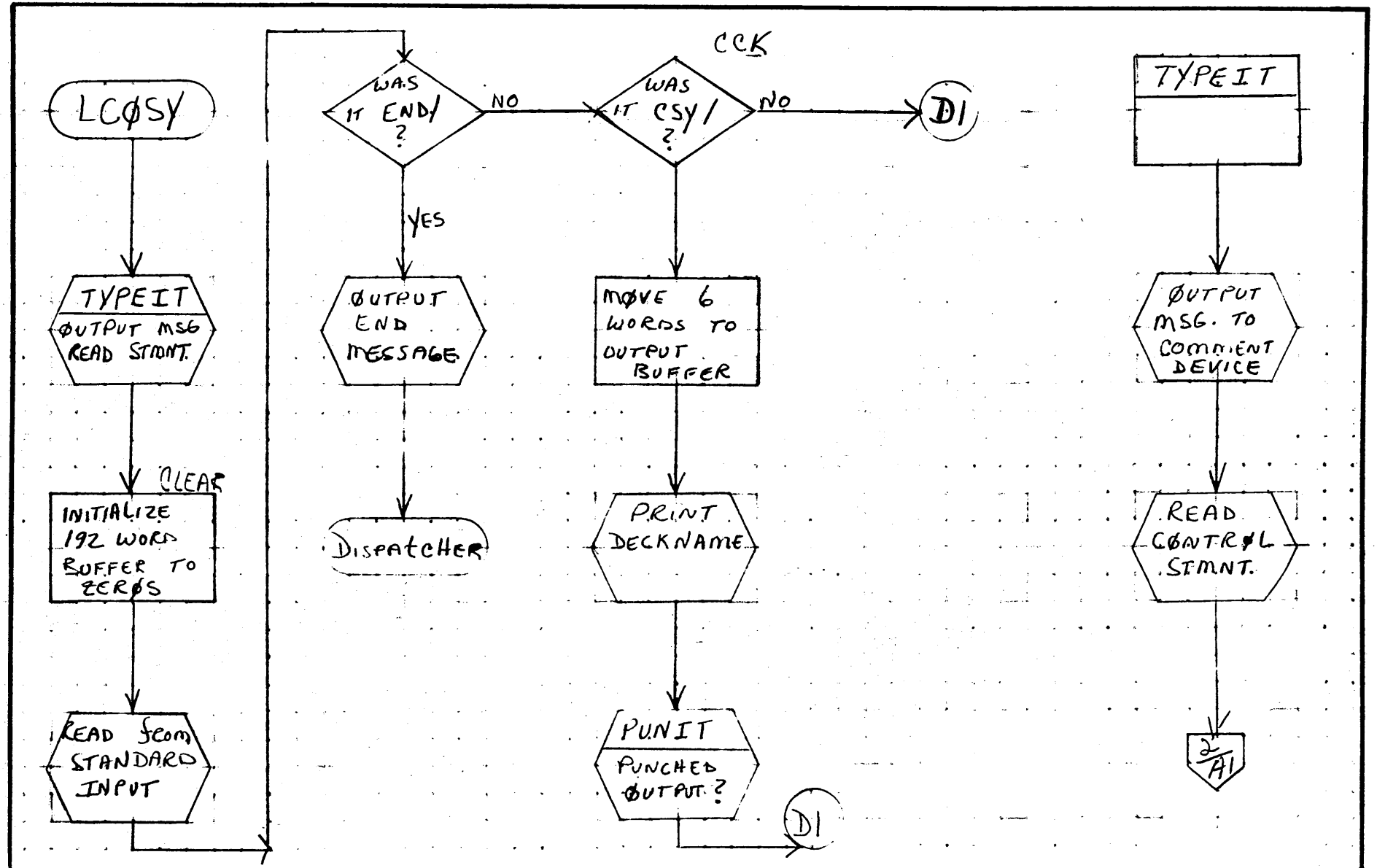
None

57.4 Entry Interfaces

None

57.5 Exit Interfaces

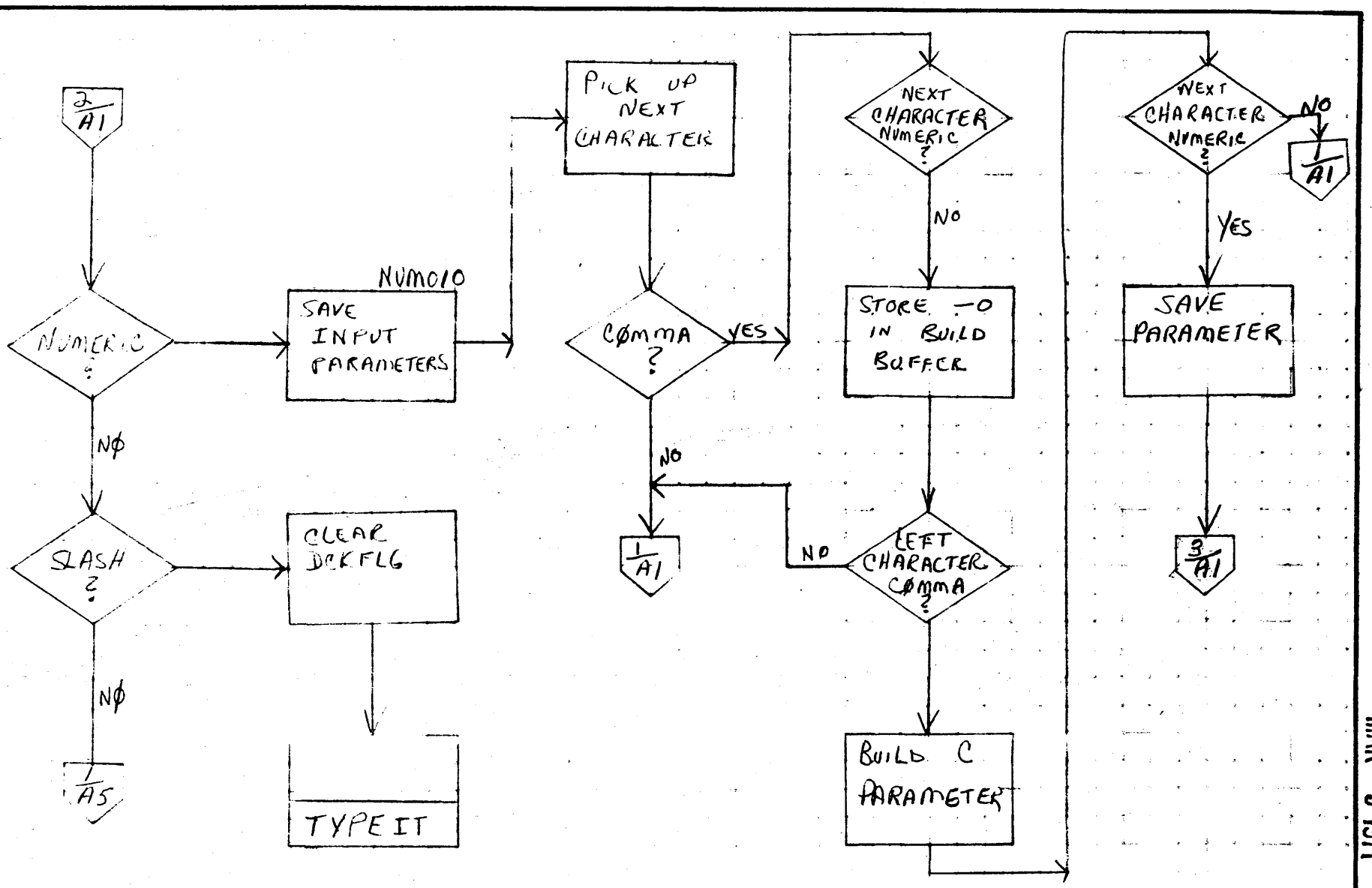
None



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: LCØSY		PROJECT MGR.			
	MSOS 3.0	PAGE 1 OF	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

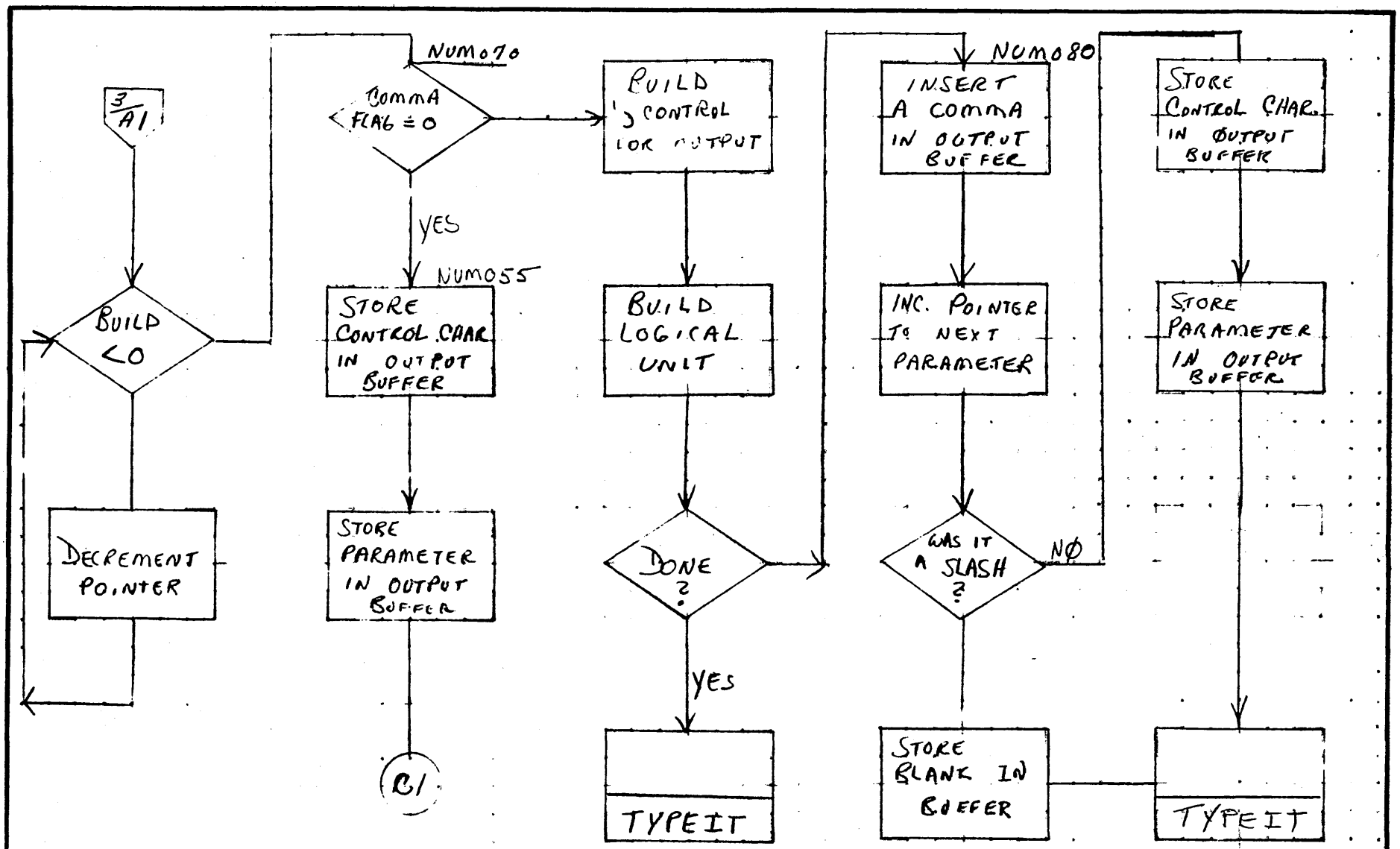
57.2



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ICQ1	PROJECT MGR.			
	1.2.3.0 PAGE 2 OF	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

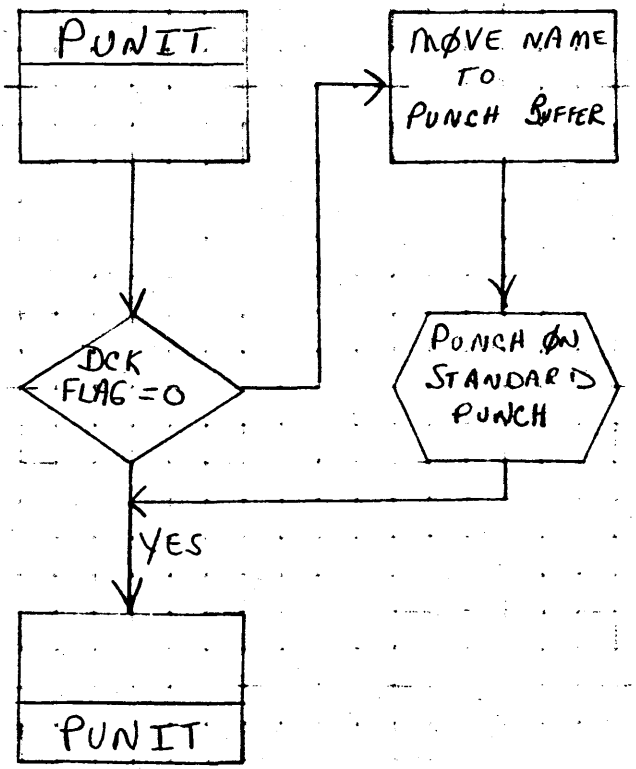
57.3



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LCØSY		PROJECT MGR.		
	MSOS 3.0	PAGE 3 OF	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

57.4



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LOGSY	PROJECT MGR.			
	MSOS 3.0 PAGE 4 OF	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

57.5

58.0 DTLP

58.1 Function

DTLP is a bootstrap program which provides a means of getting the DSKTAP file into core and execution.

58.2 Entry Points

DTLP

58.3 Externals

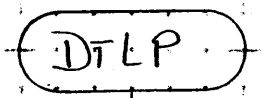
SI

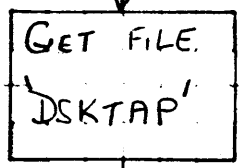
58.4 Entry Interfaces

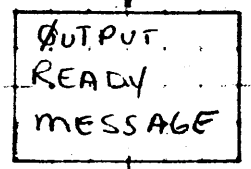
None

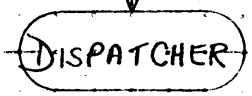
58.5 Exit Interfaces

None









CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: DTLP		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 1	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

59.0 Disk to tape Loading Program

59.1 Function

The function of DSKTAP is to save a mass storage operating system, which has been installed on a disk pack, on a magnetic tape. This magnetic tape can be dumped onto a disk pack via DSKTAP thus eliminating the need to re-install.

59.2 Entry Point Names

DSKTAP

59.3 Externals

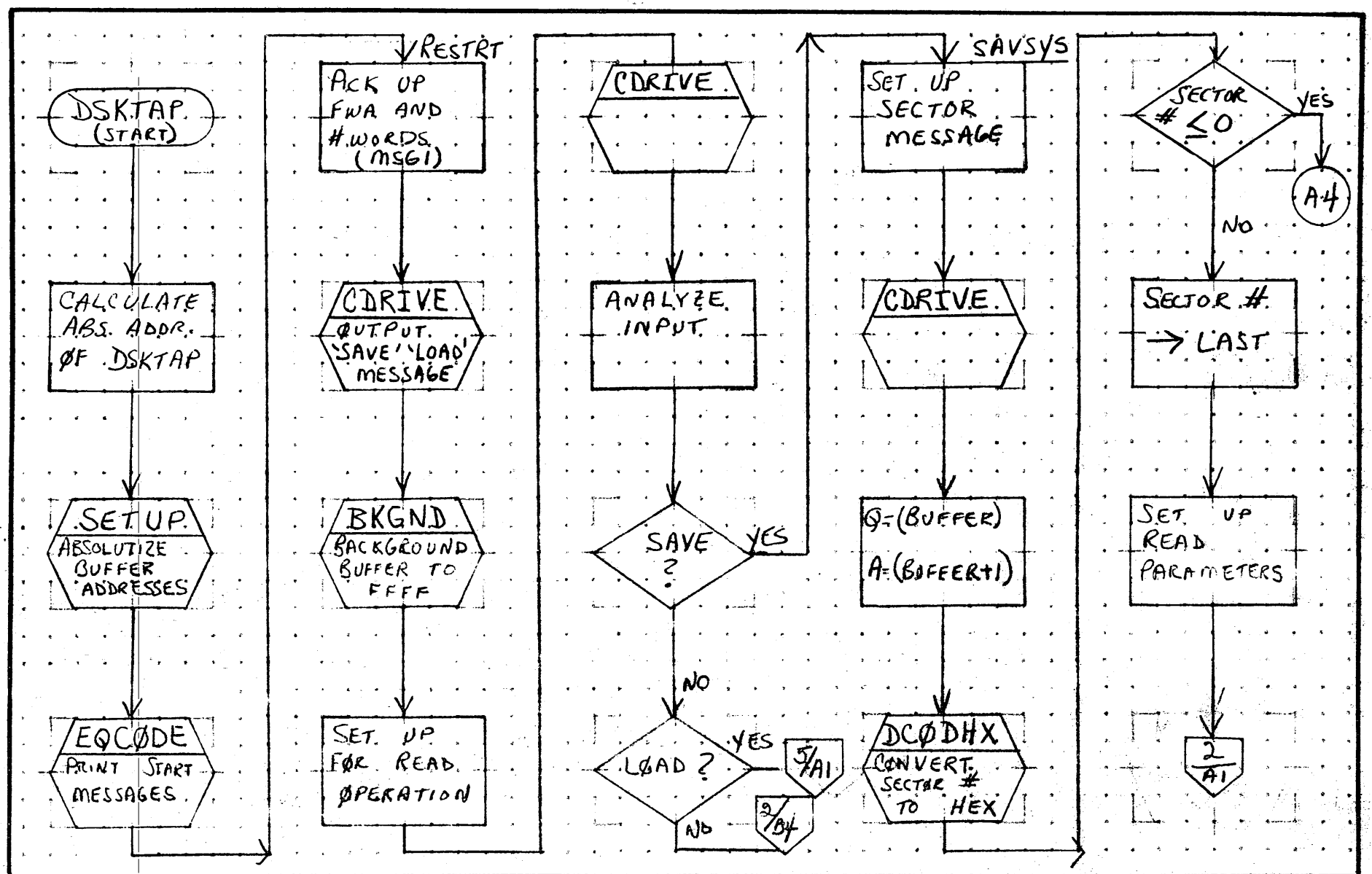
ENCDHX	
DCODHX	
EQCODE	
CDRIVE	Comment Driver
MDRIVE	Mass Storage Driver
MGDRIVE	Magnetic Tape Output
MGREAD	Magnetic Tape Input

59.4 Description

DSKTAP is a stand alone program. It has its own drivers for the 1738, 1731-1732, 1711. The DSKTAP program can be loaded in absolute form via a bootstrap loader, in relocatable form under MSOS, or called from the MSOS library.

The primary purpose of DSKTAP is to save a mass storage operating system, which has been installed on a disk pack, on a magnetic tape. The magnetic tape can be dumped unto a disk pack via DSKTAP.

Data is read from the disk in binary 1536 word blocks {1 track}, starting at sector zero to the sector specified, and copied onto magnetic tape. When data is transferred from magnetic tape to disk, 1536 word records are read from tape and copied to the disk, starting at sector zero.



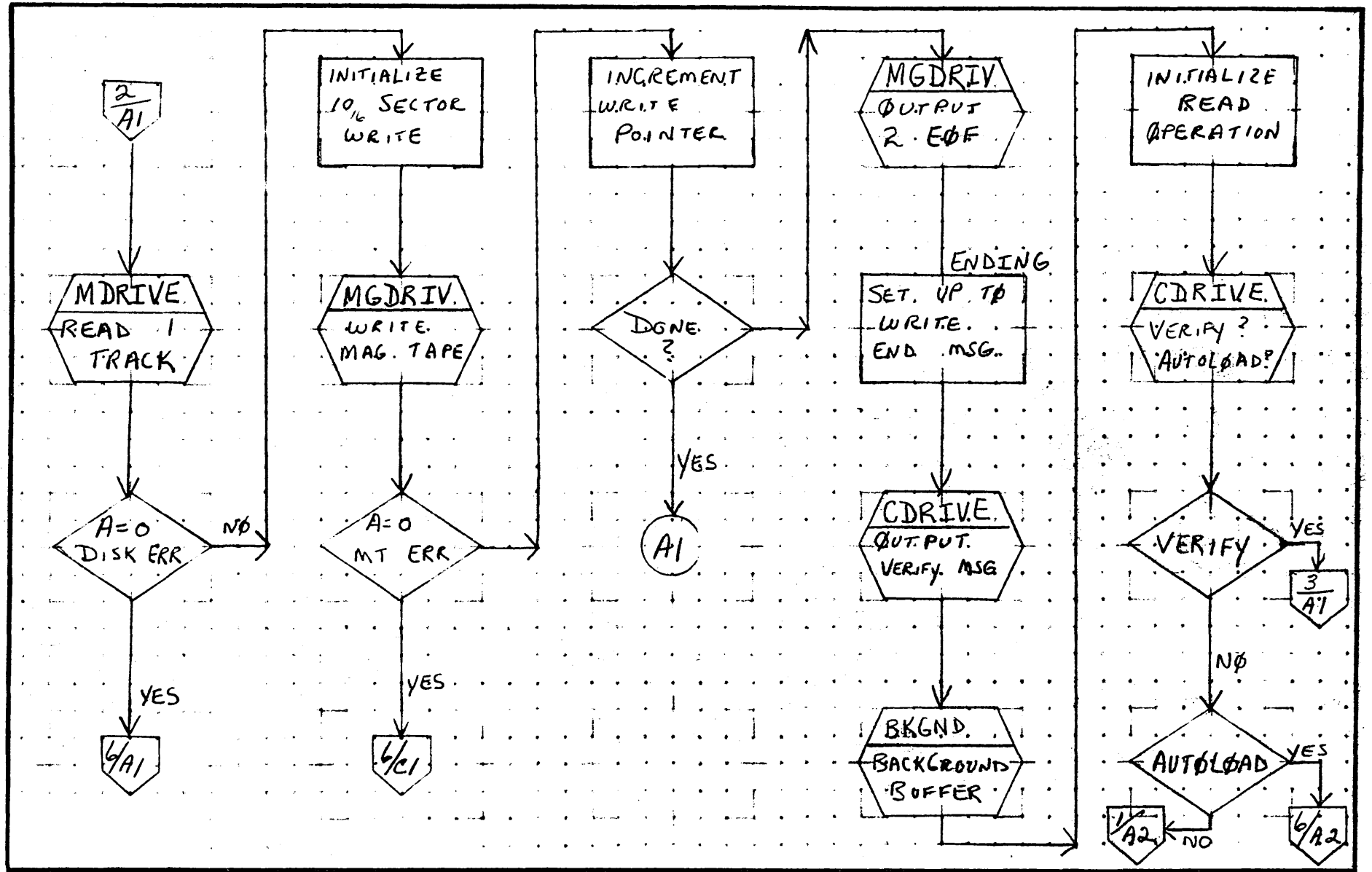
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

59.2

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



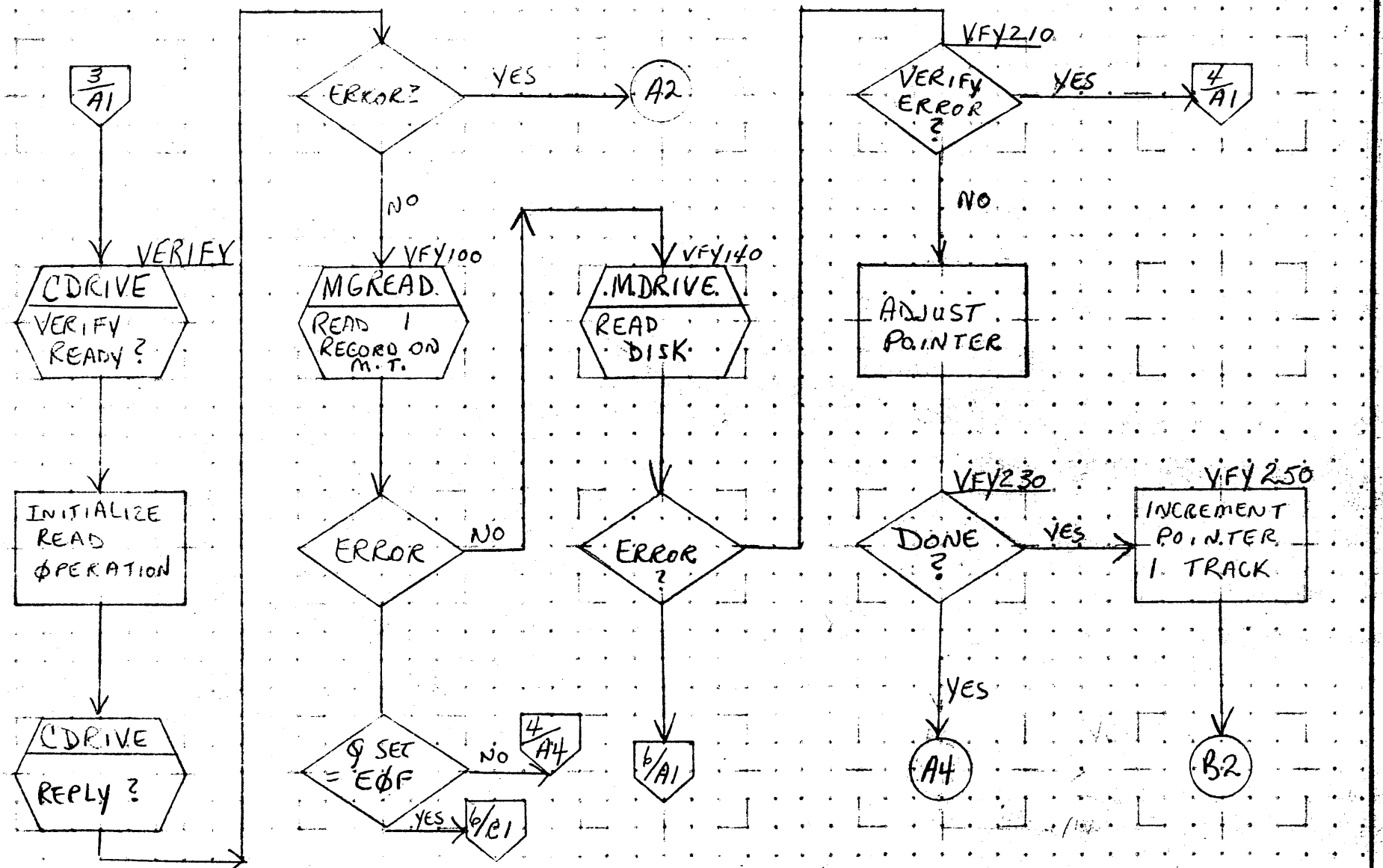
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DSKTAP	PROJECT MGR.			
	NOS 3.0 PAGE 2 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

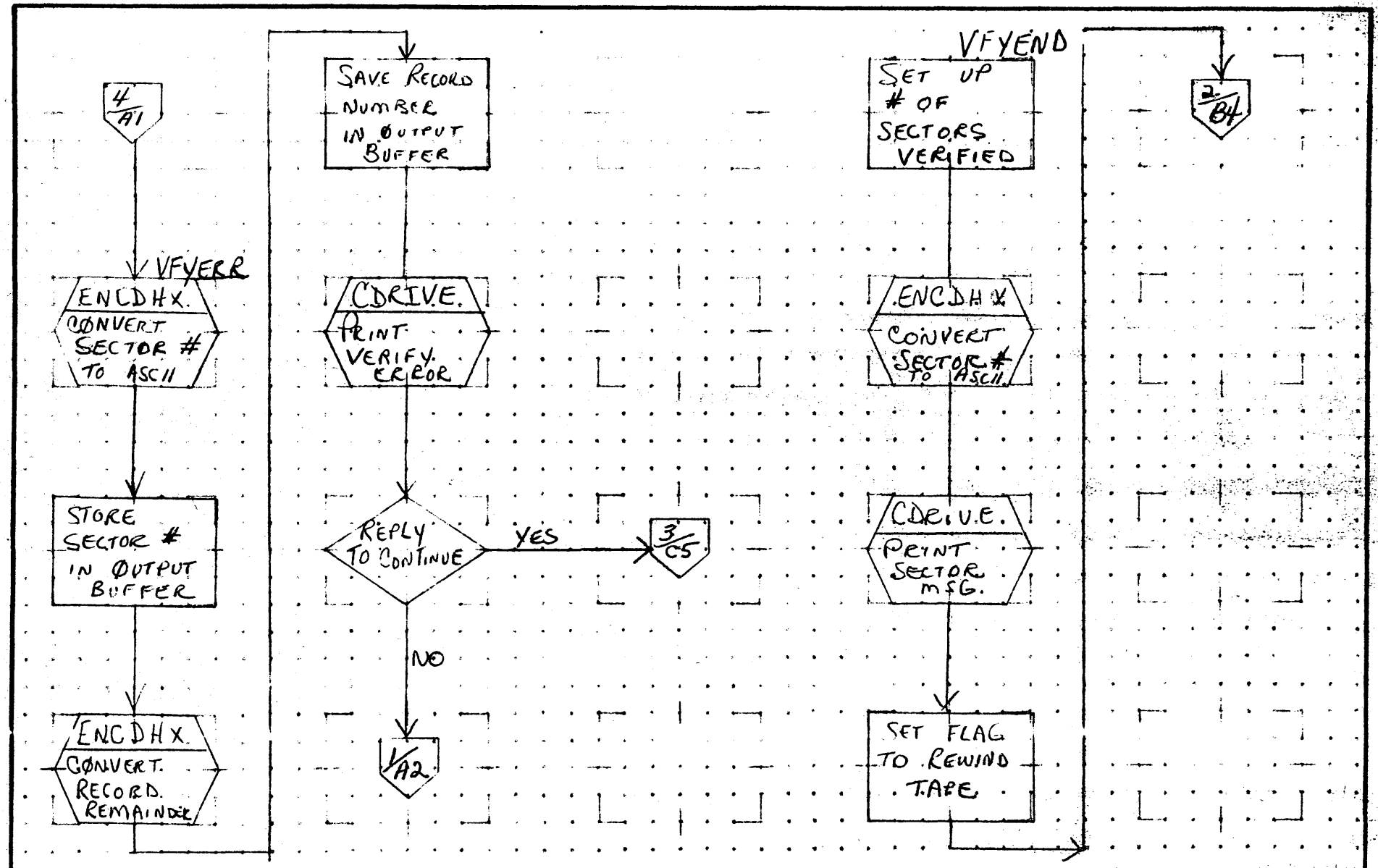
59.3



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT NO. 455 IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DSKTAP		PROJECT MGR.			
FLOWCHART ☐	MSØS 3.0	PAGE 3 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

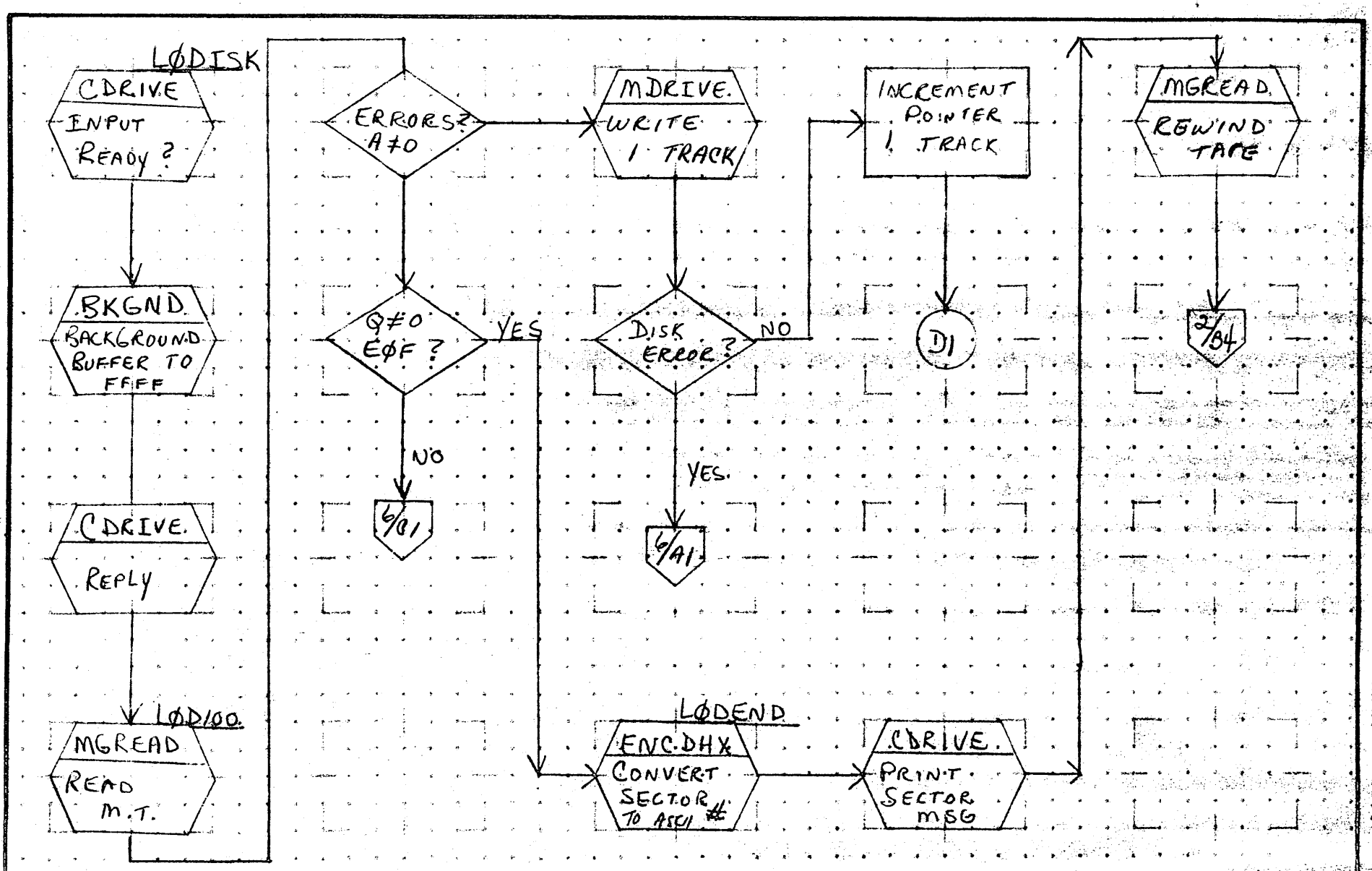
59.4



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DSKTAP		PROJECT MGR.			
	MSOS 3.0	PAGE 4 OF 6	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

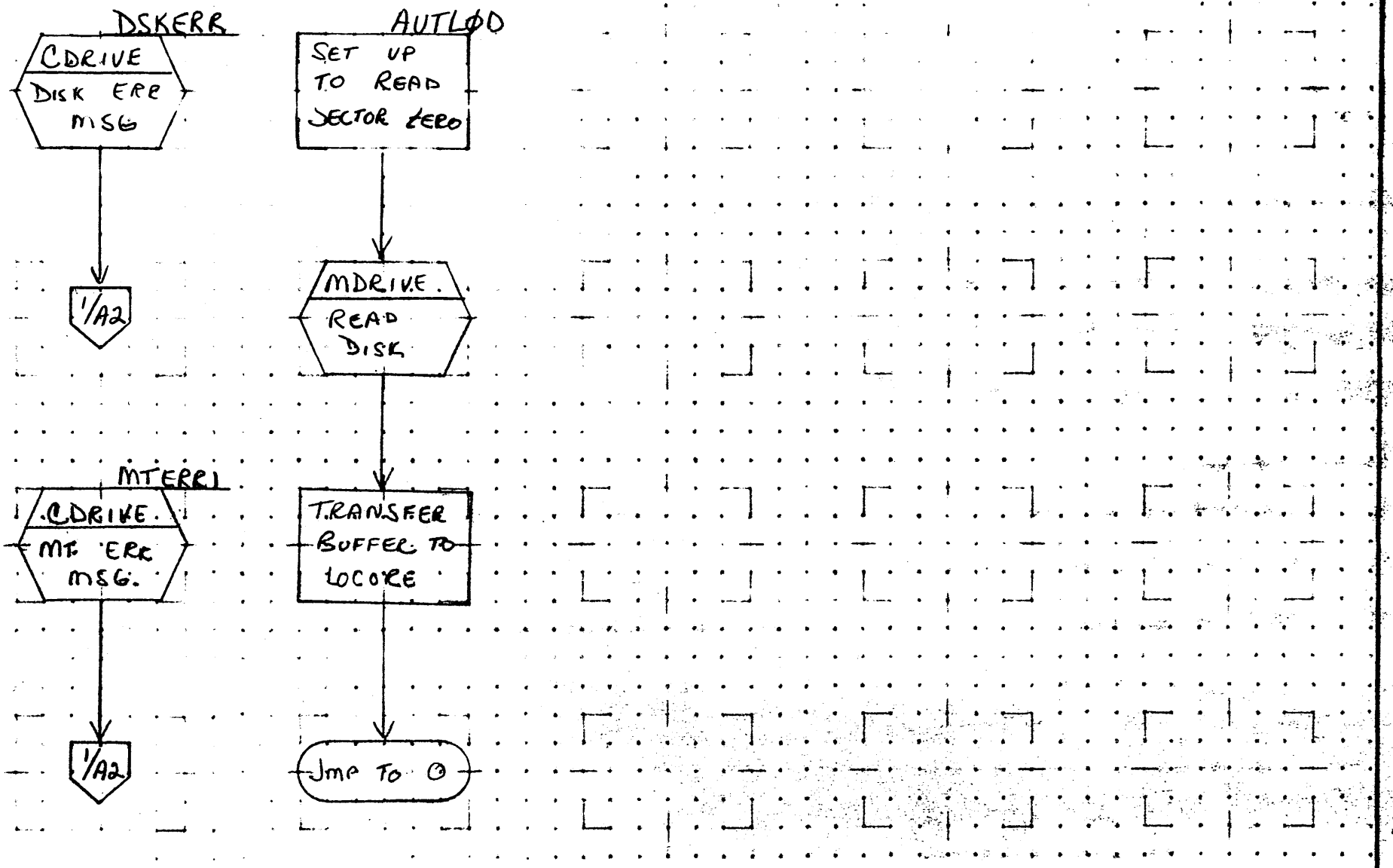
5h.5



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED / DATE
	DOCUMENT TITLE DSKTAP		PROJECT MGR.		
	MSOS 3.0	PAGE 5 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

59.6



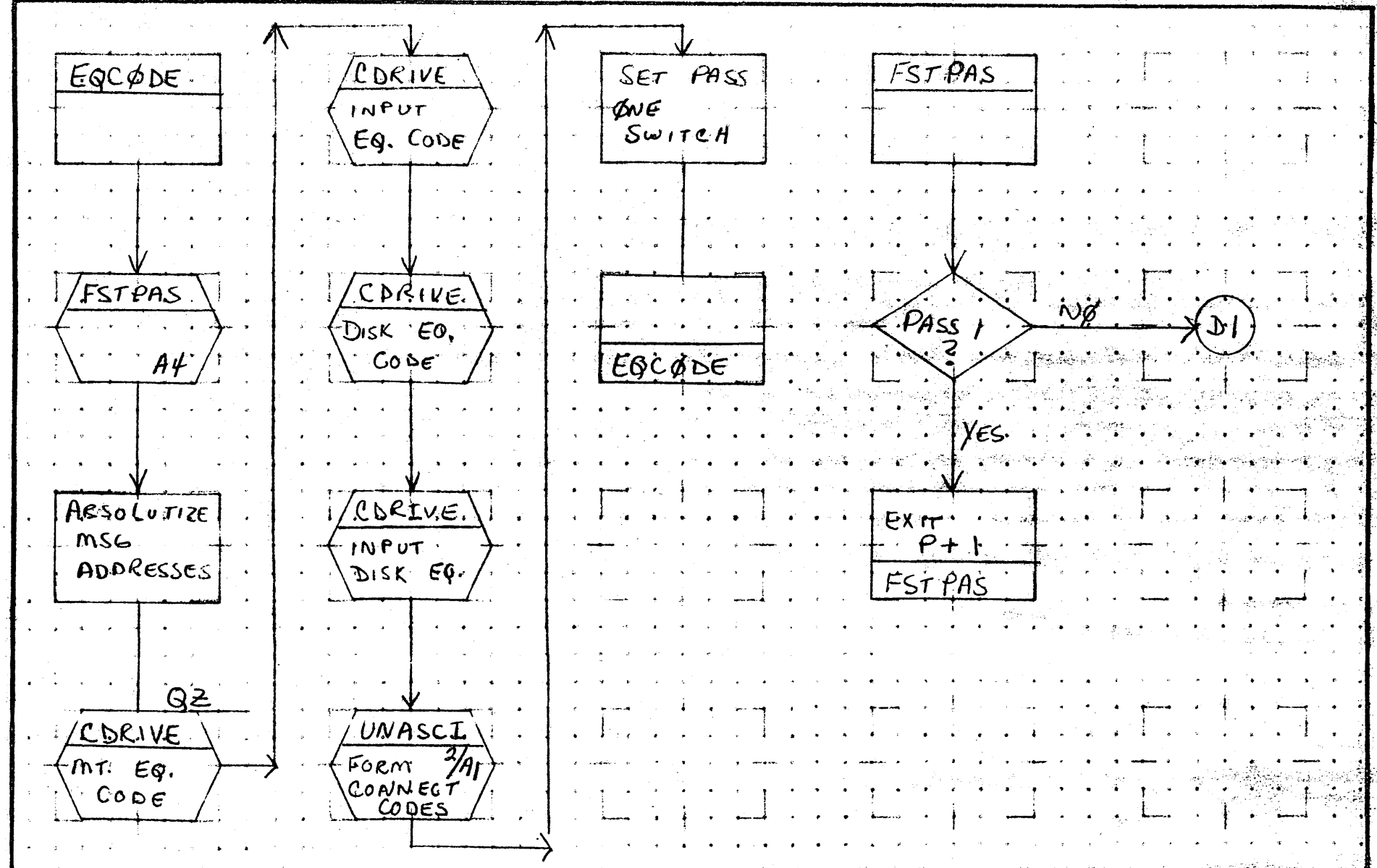
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DSKTAP	PROJECT MGR.			
	MSØS 3.0 PAGE 6 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

59.7

AA1385 (FORMERLY CA127-1)

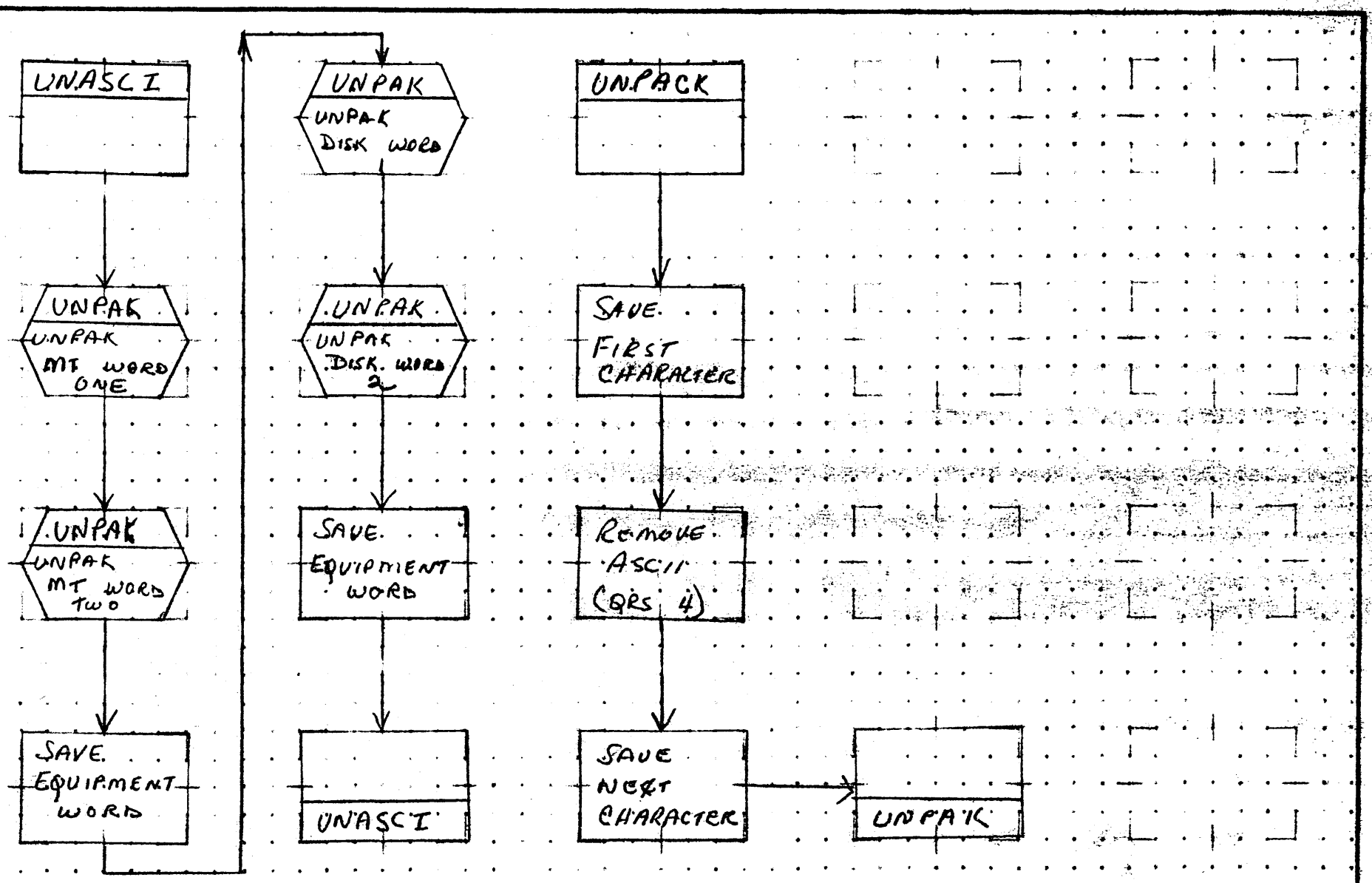
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP (EQCODE)		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

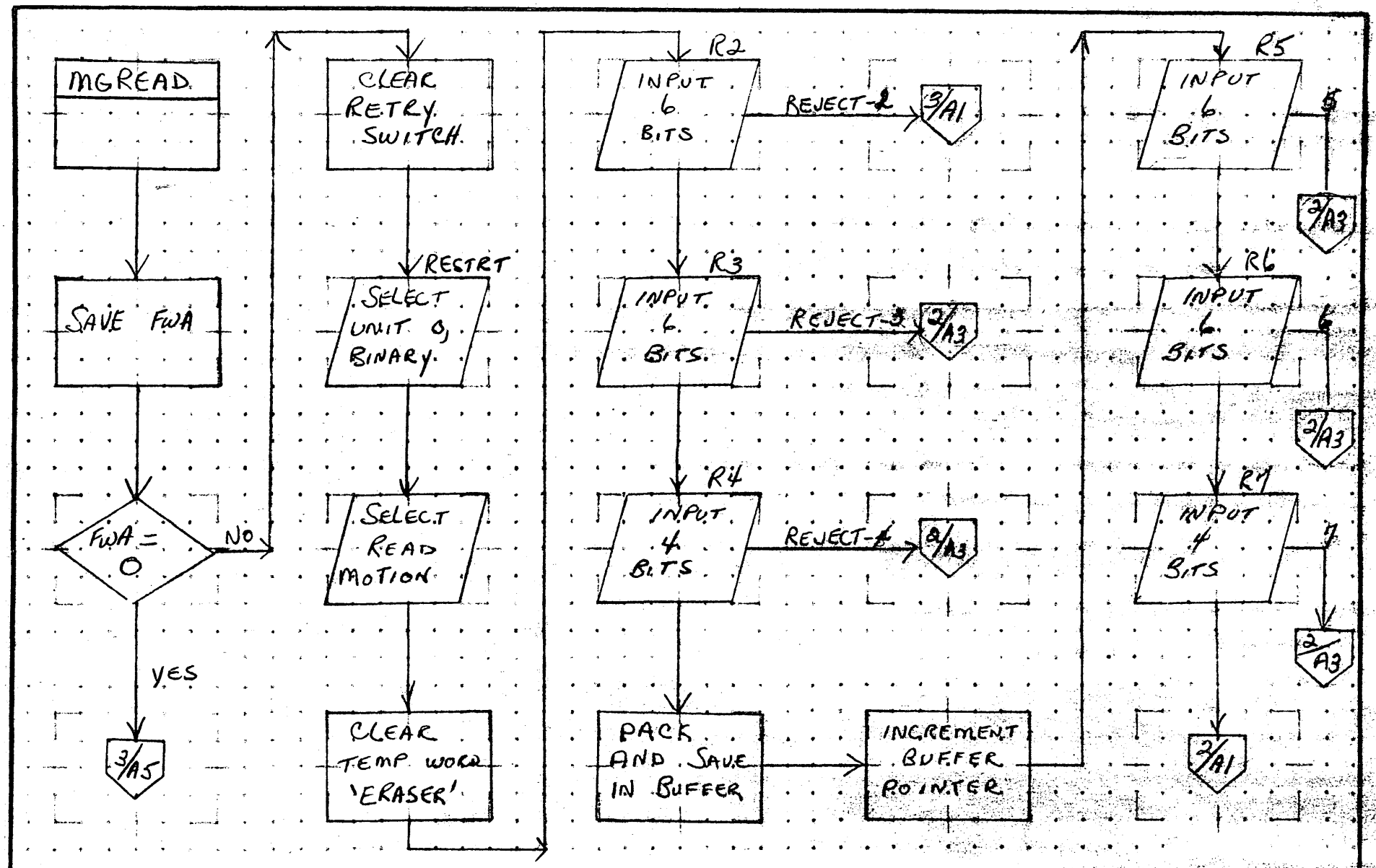
59.8



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP (EQCODE)		PROJECT MGR.		
	MSOS 3.0	PAGE 2 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

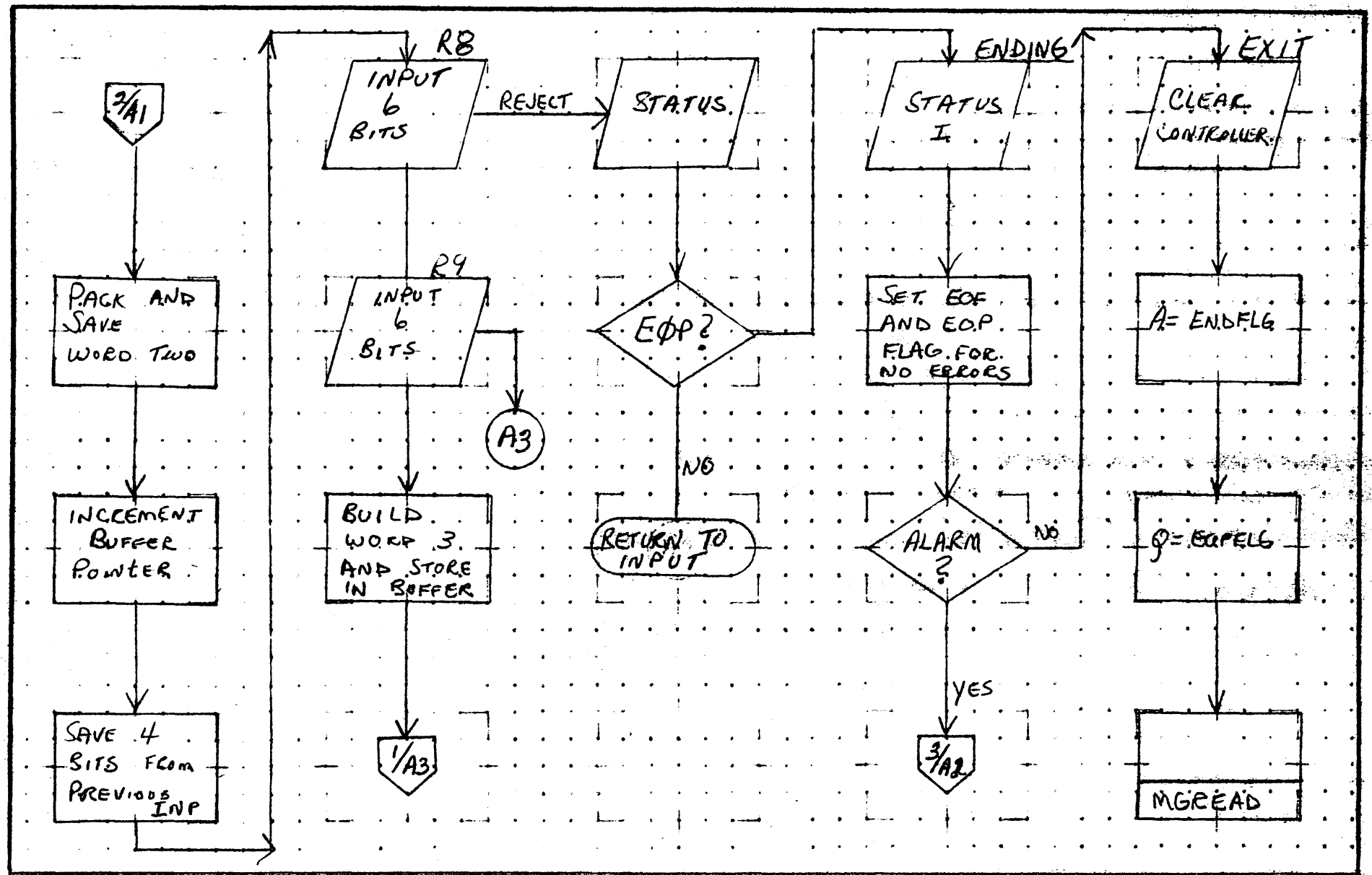
MAR 5 1971

59.9



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP(MTINP)		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 6 1974 5.10



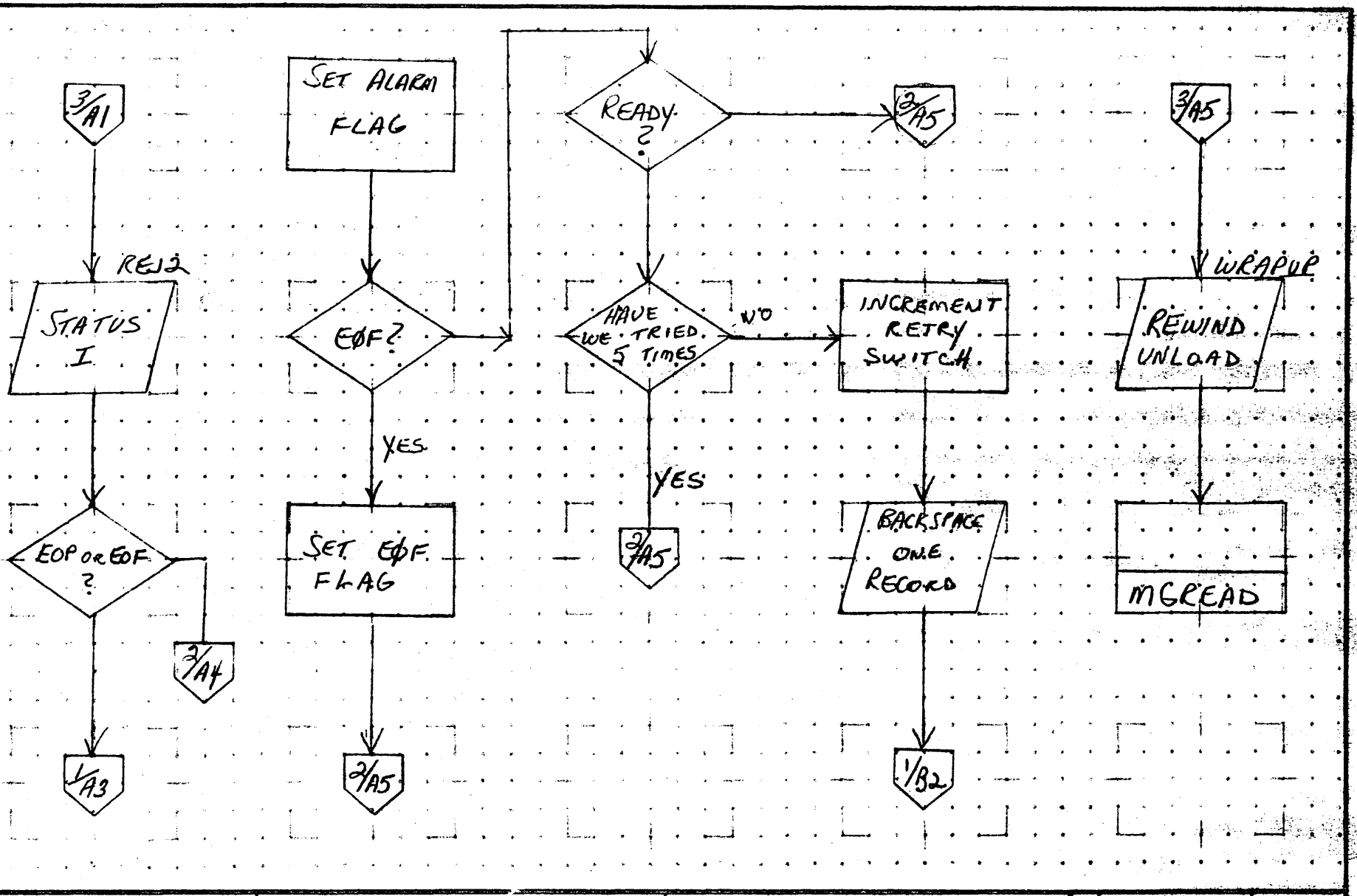
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DSKTAP (MTINP)		PROJECT MGR.			
	MSOS 3.0	PAGE 2 OF 3	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

59.11

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

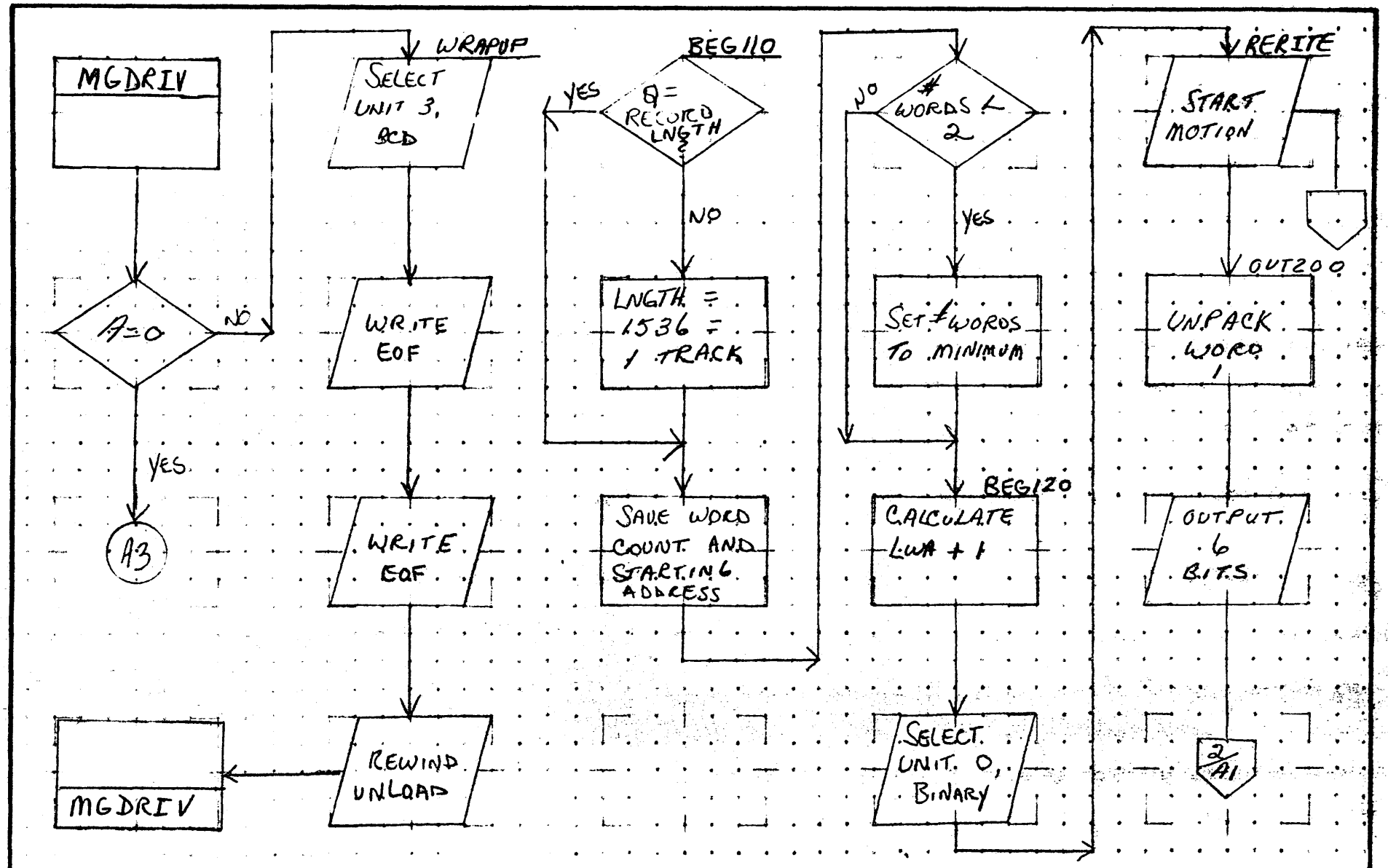


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 7700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DSKTAP (MTINP)	PROJECT MGR.			
	NOS 3.0 PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971 59.12

AA1385 (FORMERLY CA127-1)

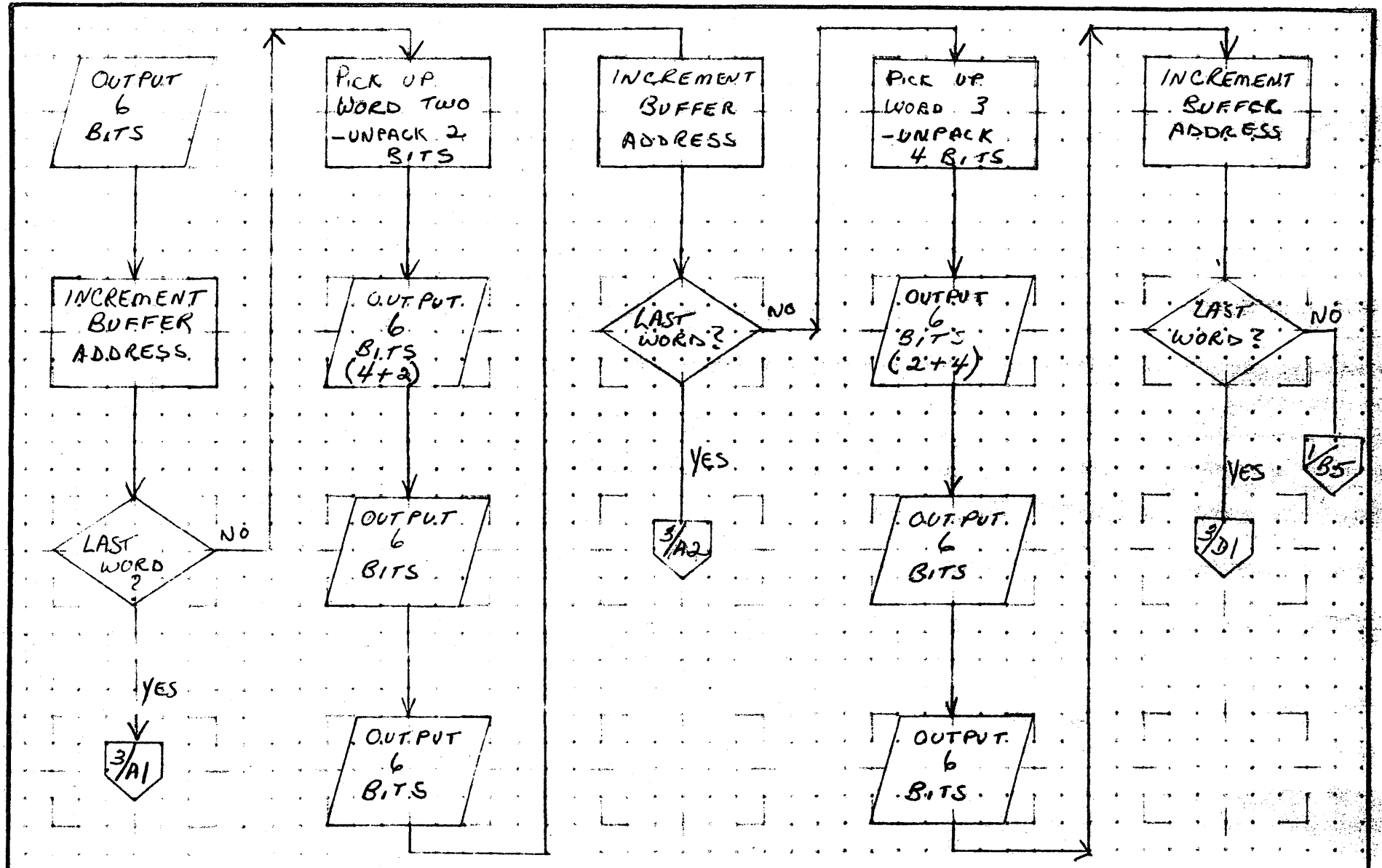
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP(MTOUT)		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

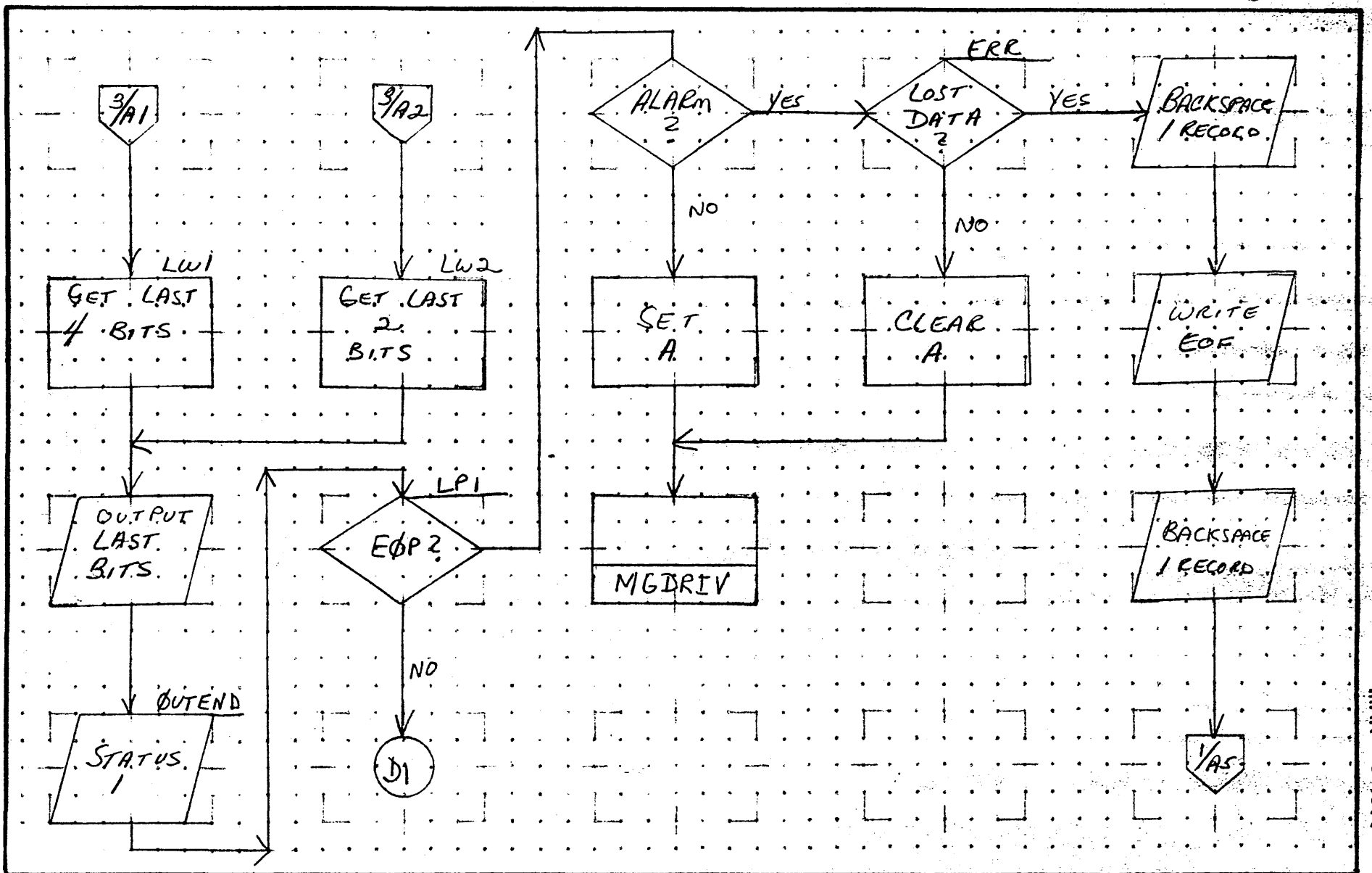
59.13



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP (MTOUT)		PROJECT MGR.		
	MSOS 3.0	PAGE 2 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 6 1971

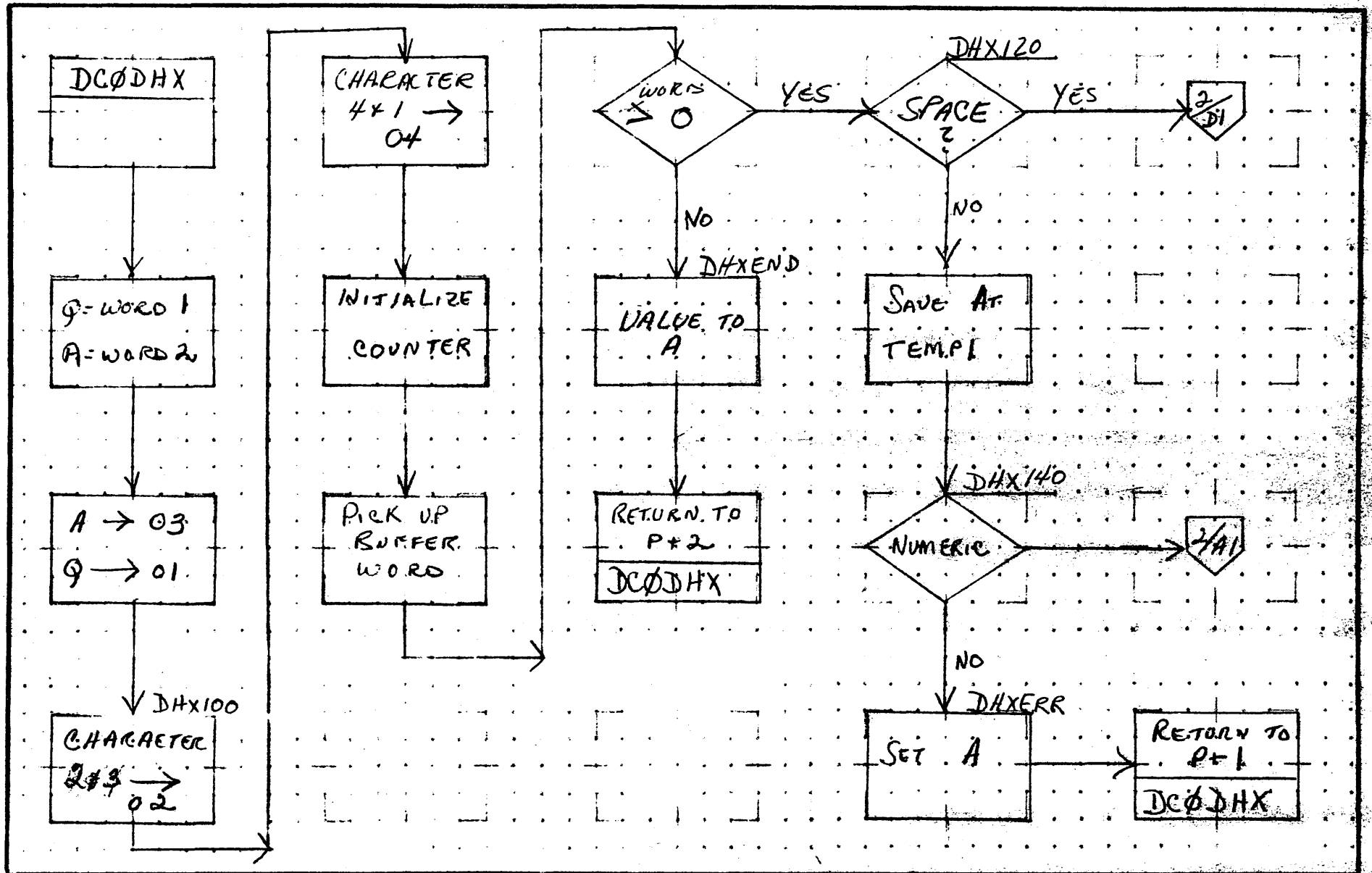
59.14



CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE DSKTAP (MTOUT)	PROJECT MGR.	
	MS/S 3.0 PAGE 3 OF 3	PROJECT NAME	
	NUMBER ISSUE DATE	TASK NO.	
	DRAWN BY DATE	TASK NAME	

MAR 5 1971

59.15



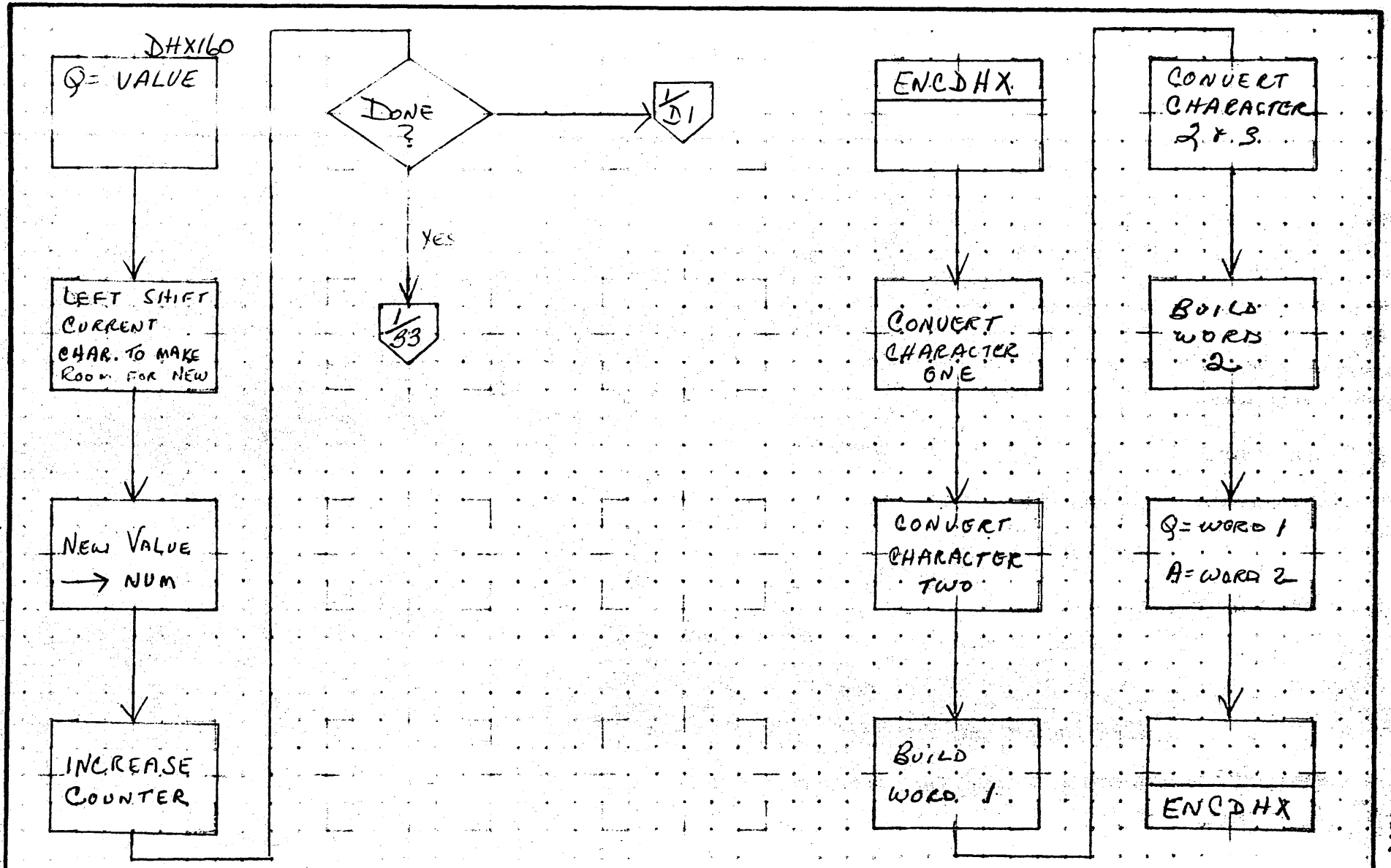
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP(DCØDHX)		PROJECT MGR.		
	MSOS 3.0	PAGE 1 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

59.16

AA1385 (FORMERLY CA127-1)

PRINTED IN



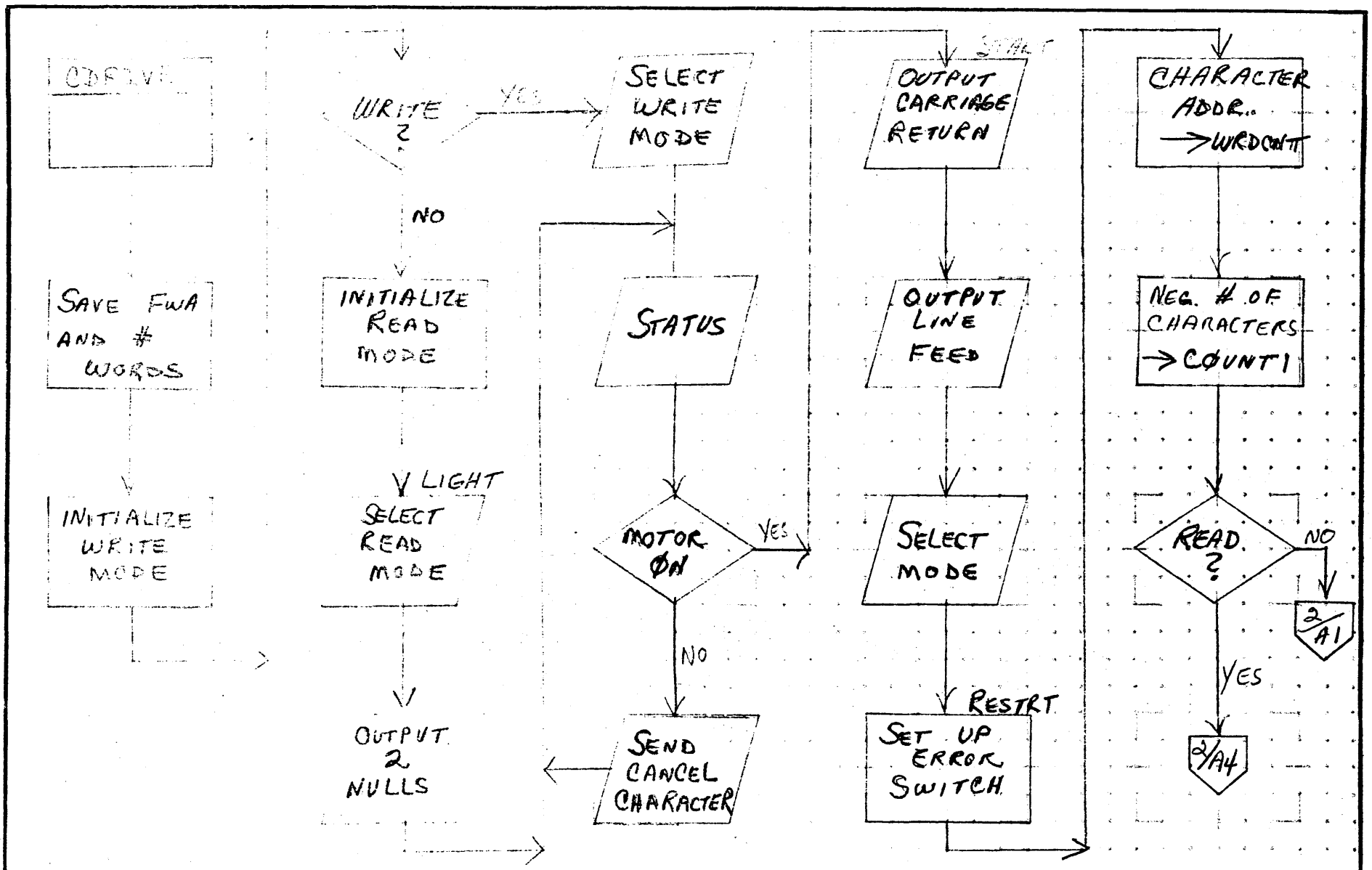
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP(DCØDHX)		PROJECT MGR.		
	MSOS 3.0	PAGE 2 OF 2	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

59.17.2

AA1385 (FORMERLY CA127-1)

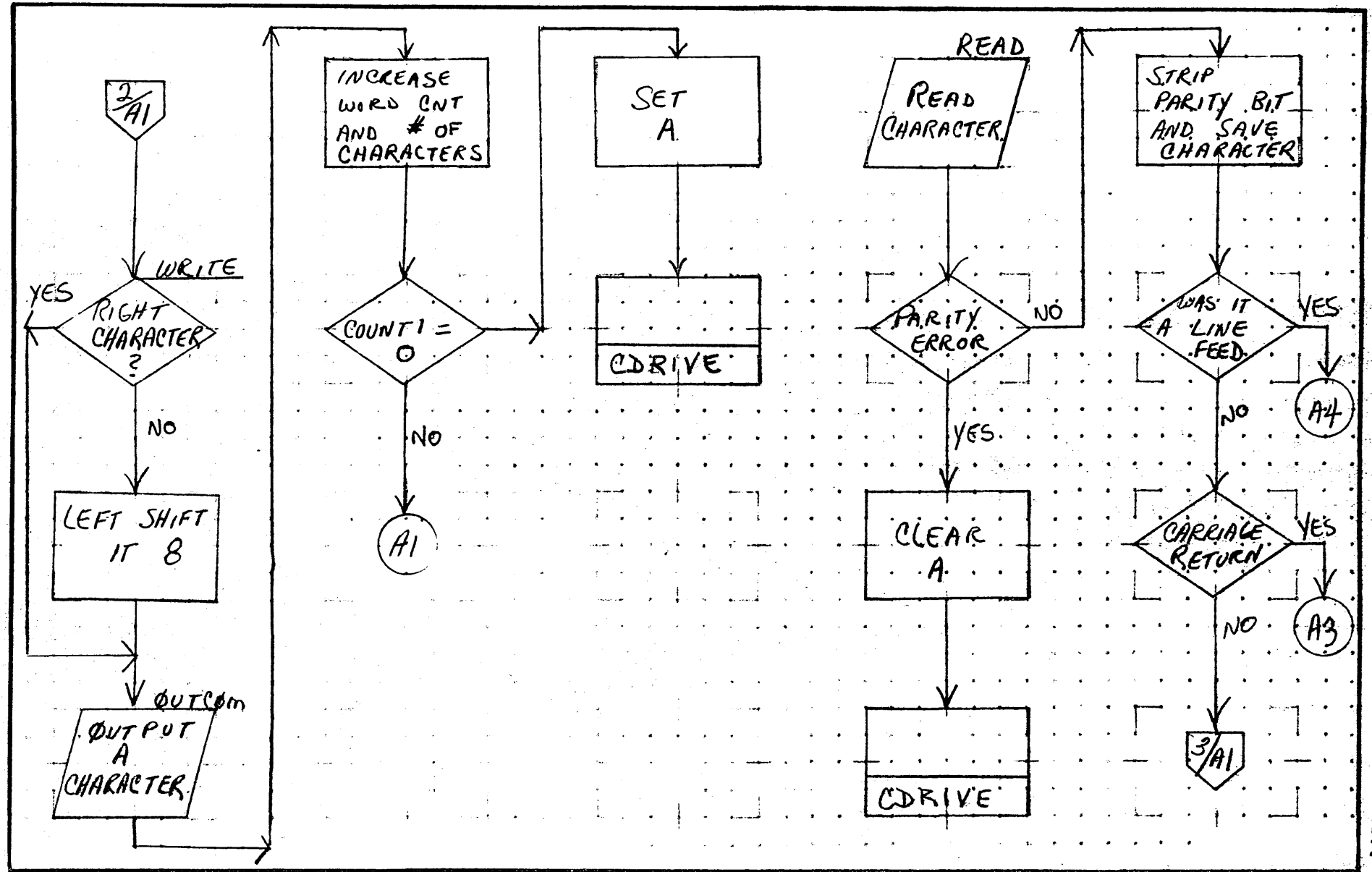
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS LMS	MACH. TYPE 7600	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DATAR (CDC 6-12)		PROJECT MGR.		
		PAGE 1 OF 3	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

59.18



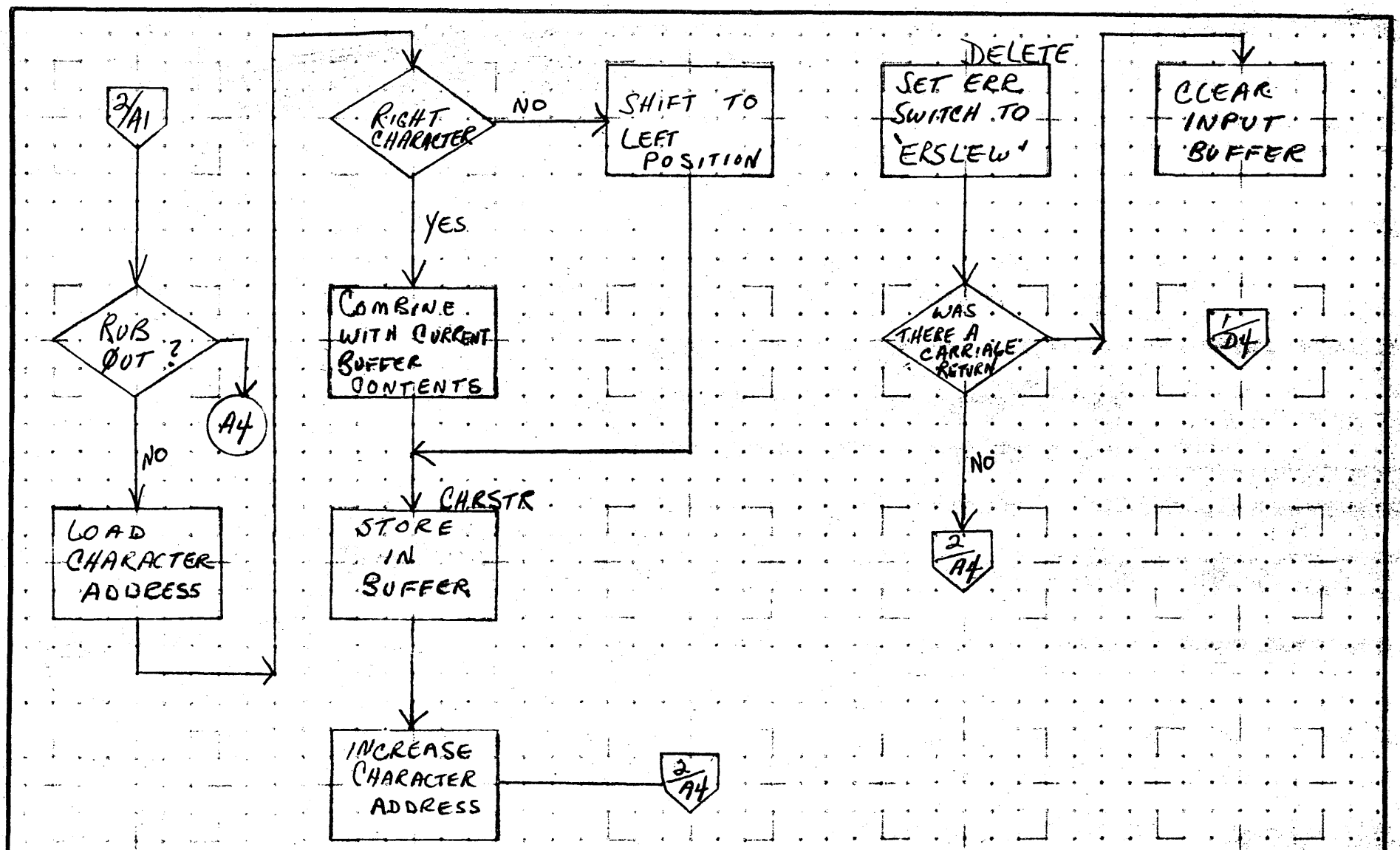
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE DSKTAP(CDRIVE)		PROJECT MGR.		
FLOWCHART ☐	MSOS 3.0	PAGE 2 OF 3	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

59.19

AA1385 (FORMERLY CA127-1)

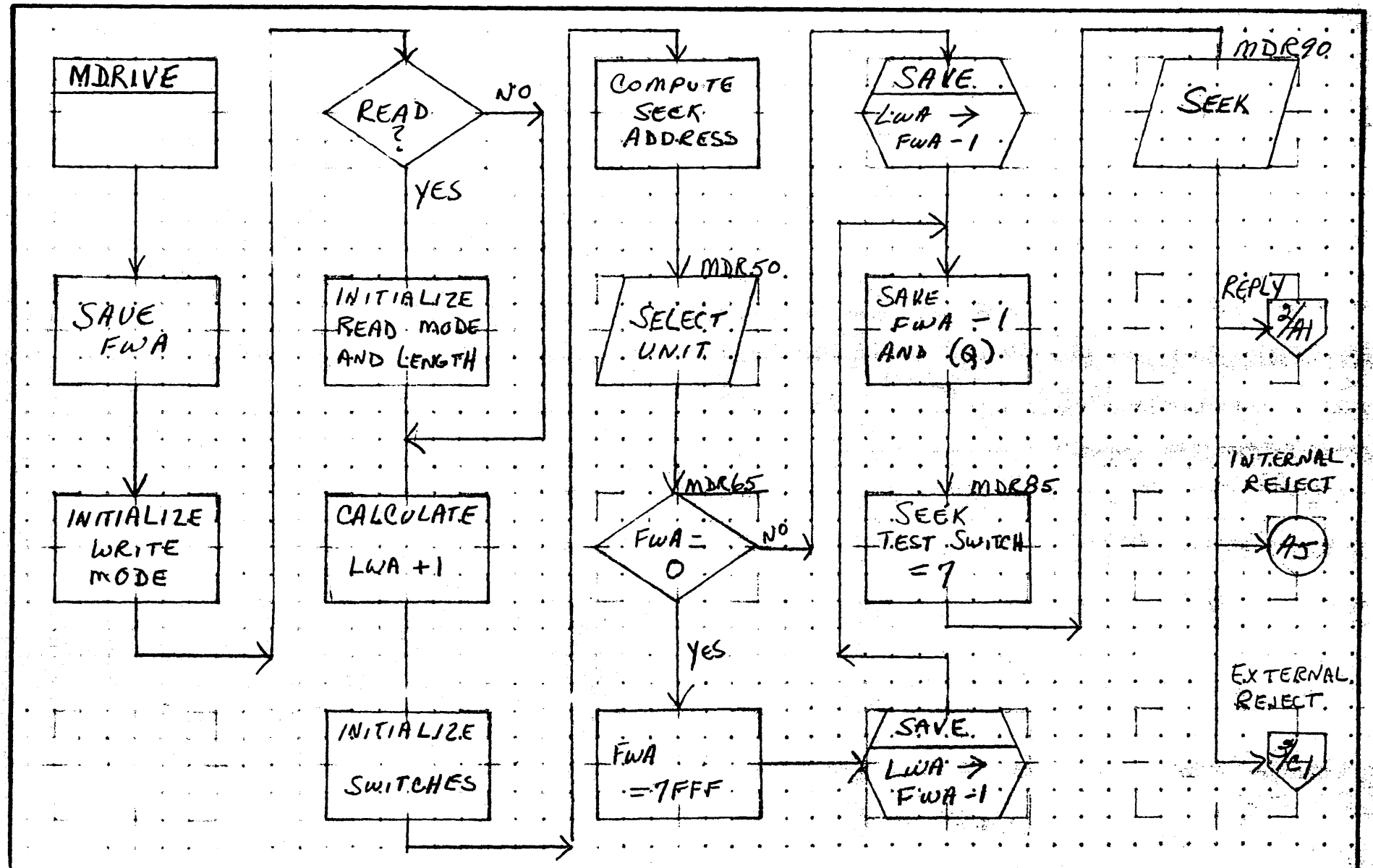
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DSKTAP (CDRIVE)	PROJECT MGR.			
	MSOS 3.0 PAGE 3 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

59.20



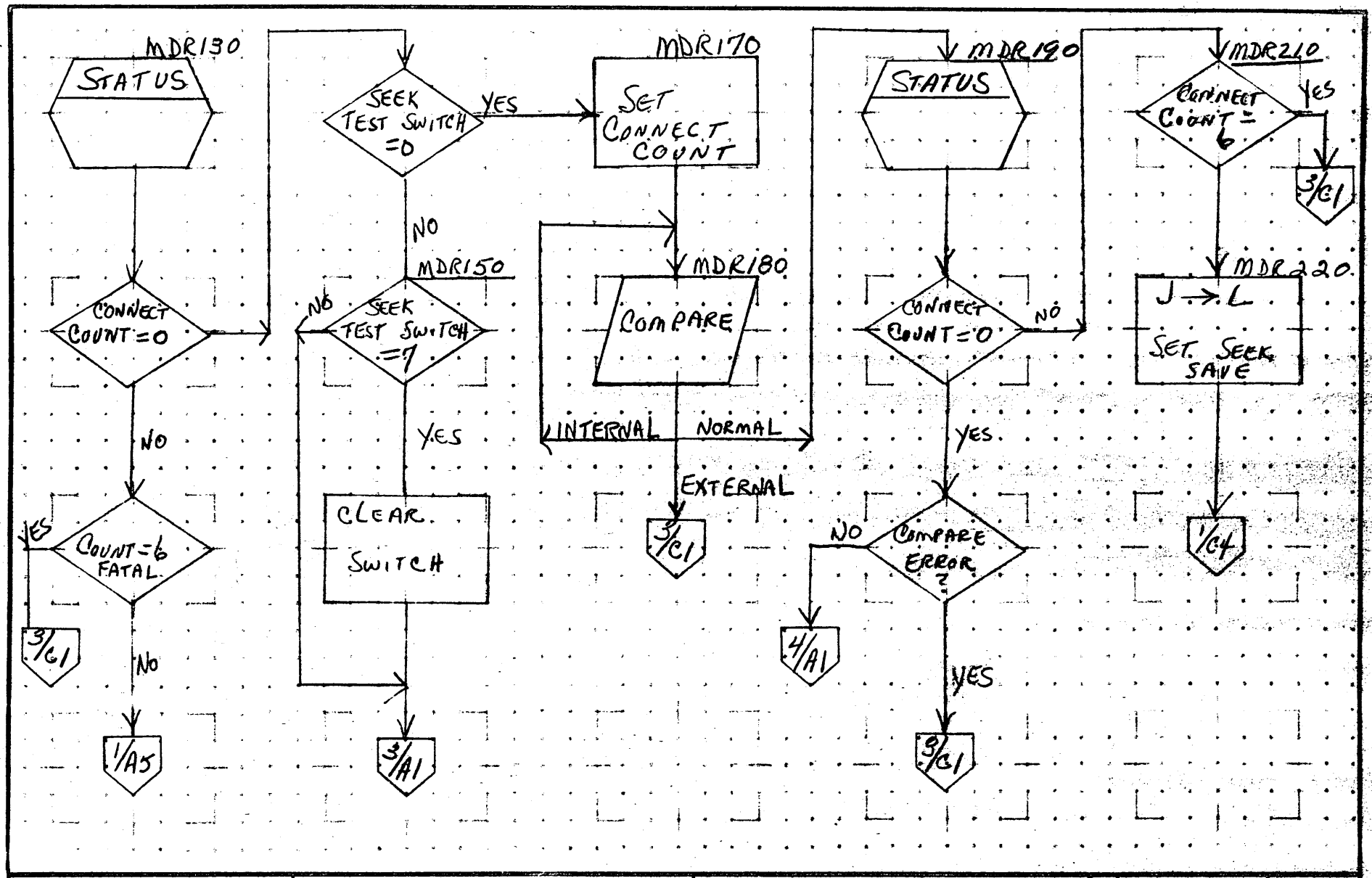
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DSKTAP(MDRIVE)		PROJECT MGR.			
	MSOS 3.0	PAGE 1 OF	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

59.21

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



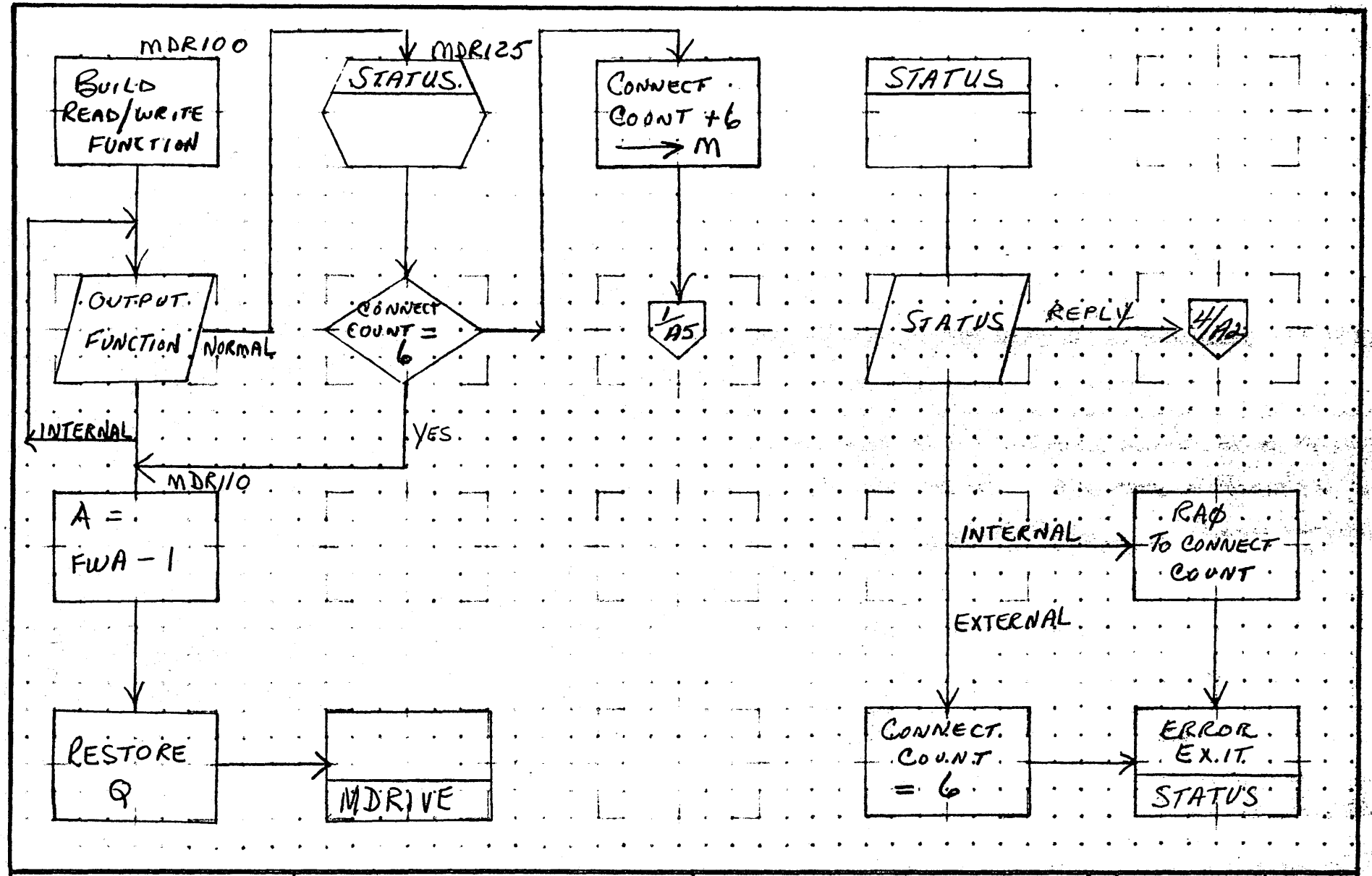
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE DSKTAP(MDRIVE)		PROJECT MGR.			
FLOWCHART ☐	MSOS 3.0	PAGE 2 OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

59.22

AA1385 ERLY CA127-1)

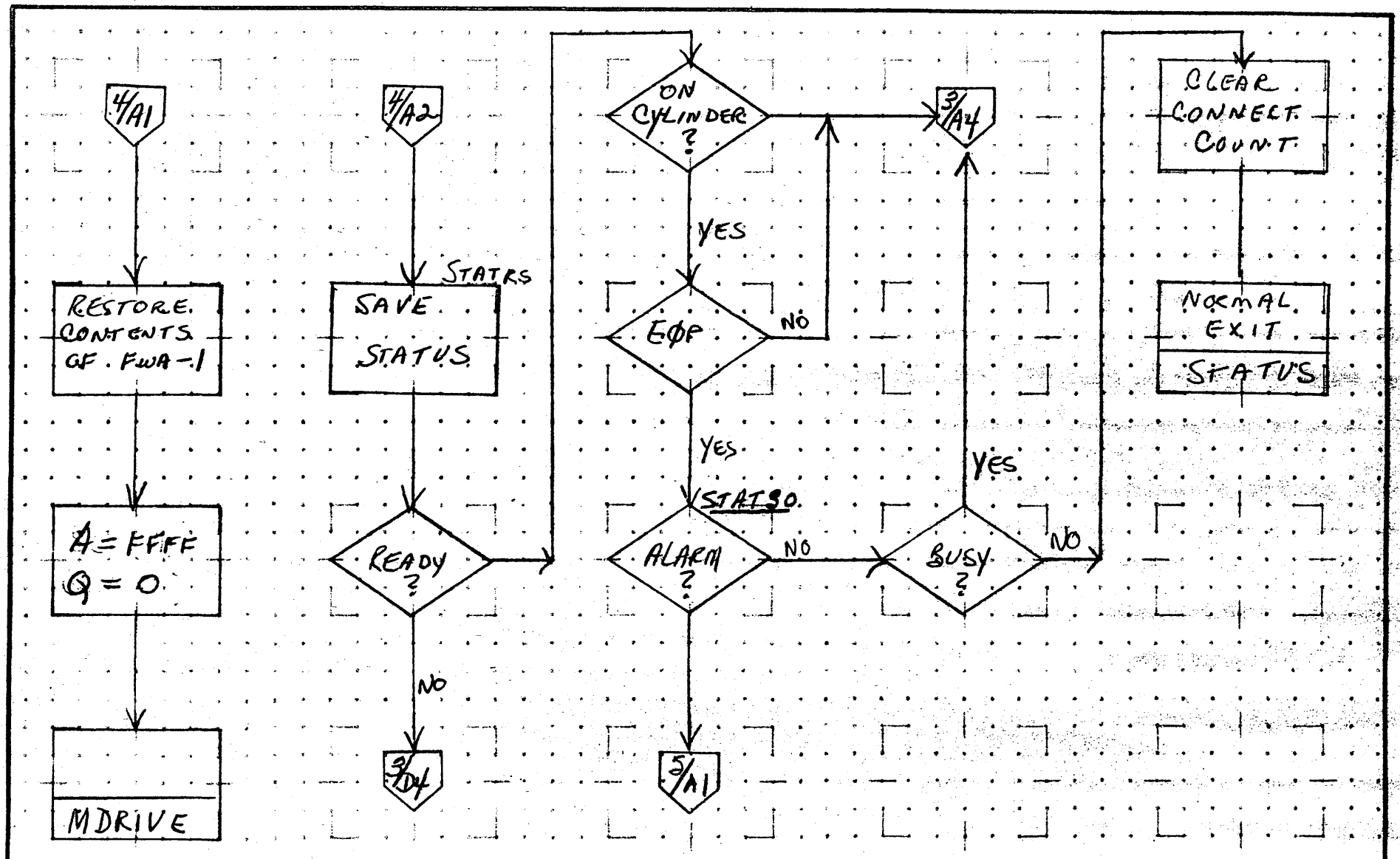
PRINTED IN U



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP(MDRIVE)		PROJECT MGR.		
	MSOS 3.0	PAGE 3 OF	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

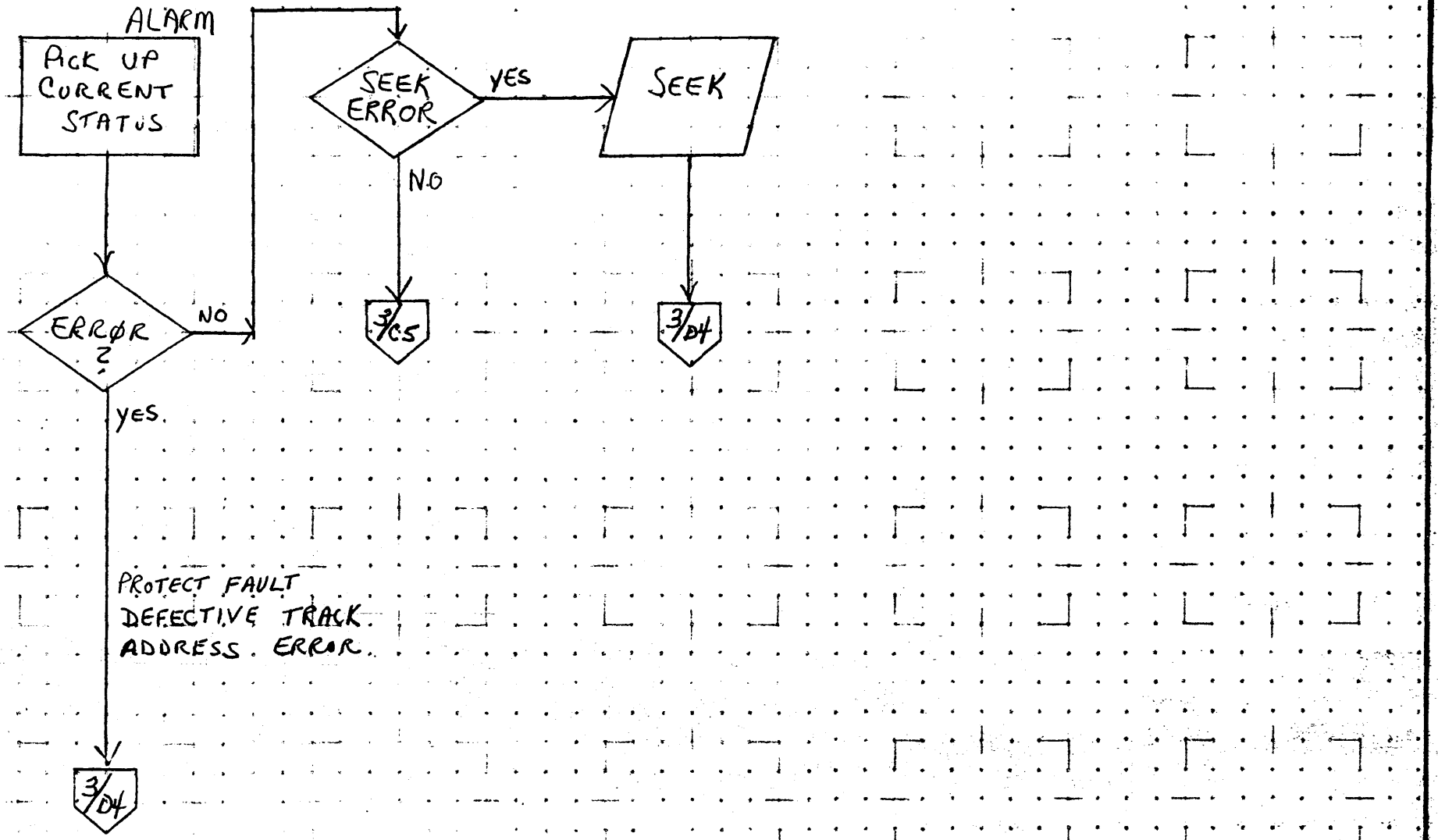
59.23



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: DSKTAP(MDRIVE)	PROJECT MGR.			
	MSOS 3.0 PAGE 4 OF	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

59.24



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE DSKTAP(MDRIVE)		PROJECT MGR.		
	MSOS 3.0	PAGE 5 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

60.0 SILP

60.1 Function

SILP is a bootstrap program used to load the System Initializer (SI) file.

60.2 Entry Points

SILP

60.3 Externals

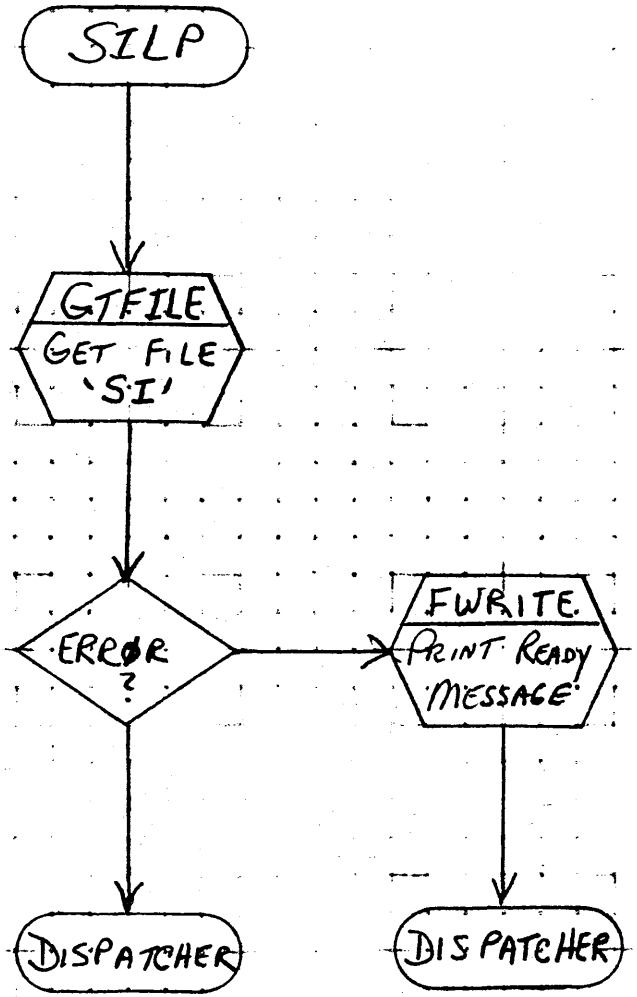
SI

60.4 Entry Interfaces

None

60.5 Exit Interfaces

None



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE SILP		PROJECT MGR.			
FLOWCHART ☐	MSOS 3.0	PAGE 1 OF 1	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

60.2

AA1385 (FORMERLY CA127-1)

61.0 UPDATE - Binary System Updates

61.1 Function

This program is used to update a System Binary Tape. It provides a means to replace programs with updated versions using only Mag Tape eliminating the intermediate card handling of the System Binary Deck.

61.2 Entry Point Names

UPDATE

61.3 Externals

None

61.4 Entry Interfaces

Scheduled by Job Processor

61.5 Exit Interfaces

Exit request

61.6 Program Equivalences

SCRLSB	FIRST SECTOR OF SCRATCH AREA
SCRLU	LOGICAL UNIT CELL OF SCRATCH
STDINP	LOGICAL UNIT CELL OF STD INPUT
LSTOUT	LOGICAL UNIT CELL OF LIST OUTPUT
OUTCOM	LOGICAL UNIT CELL OF COMMENT OUT
INPCOM	LOGICAL UNIT CELL OF COMMENT INP
BINOUT	LOGICAL UNIT CELL OF BINARY OUT

ER1	DISK ERROR (INRECOVERABLE)
ER2	STD INP READ ERROR
ER3	BINARY OUTPUT WRITE ERROR
ER4	MAG TAPE UNIT ZERO READ ERROR
ER5	NAM/XFR SEQ ERR (UPDATED PROG)
ER6	NAM/XFR SEQ ERR (OLD SYSTEM)
RC	REQUEST CODE CONSTANT
FREAD	FORMAT READ
FWRITE	FORMAT WRITE
STATUS	STATUS REQUEST
MOTION	MOTION REQUEST
WFM	(MOTION) WRITE FILE MARK
RWD	REWIND CODE
ONEBIT	
ZERO	
BLANKS	ASCII BLANKS
CBLKAD	CURRENT BLOCK ADDR OF DISK
BTHRAD	BACKWARD THREAD OF DISK

61.7 Description

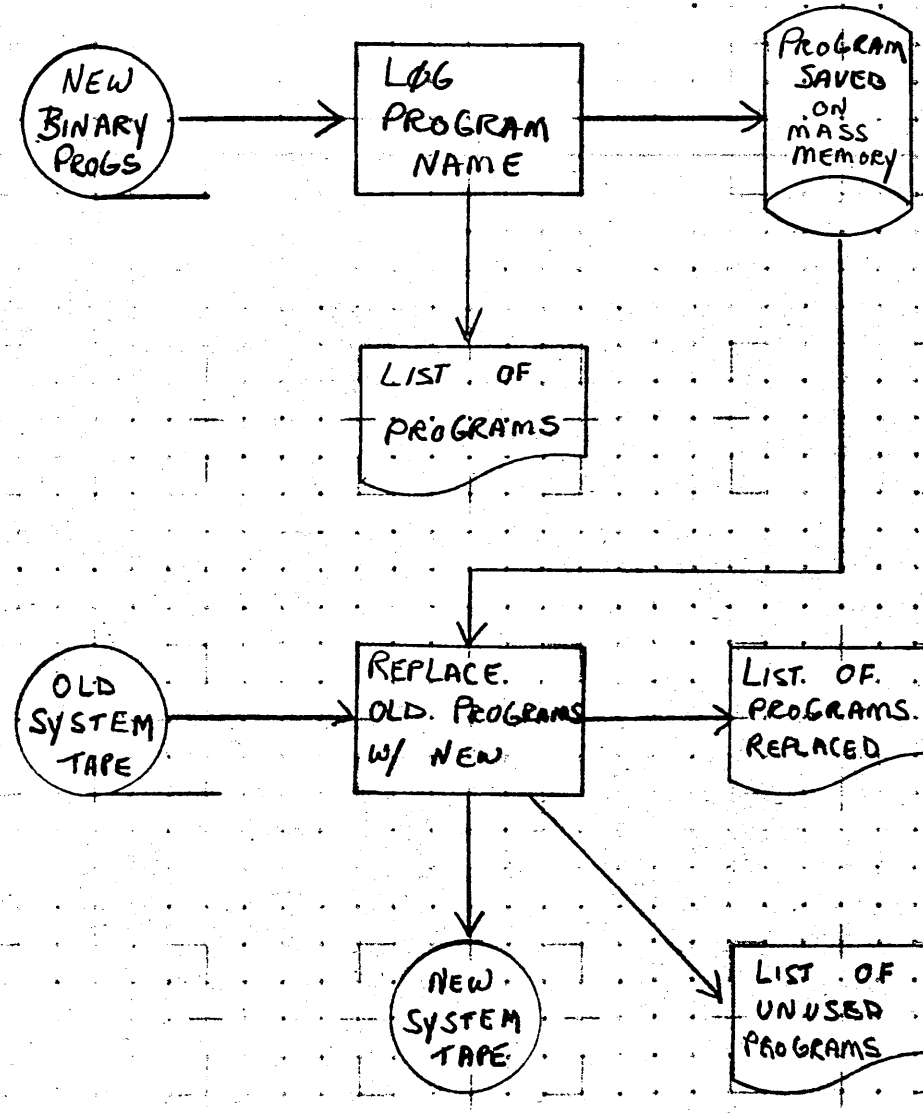
The assembled object binary tape of the update program/s is input via the Standard Input Tape Unit and saved in the scratch area of the disk. As each program NAM card is recognized, the name is logged and entered in a core resident table.

When input is complete, the operator is asked ▽READY?▽. At this point, the operator checks to see that the old system tape is ready on Mag Tape Unit Zero (0) and a scratch tape is ready on the Standard Binary Output Unit (New System Binary). When ready, depress Carriage Return. All ASCII records (System Initializer Control Cards) are transferred directly from the old system tape to the new system tape. When a NAM card is recognized, the program name is compared to those in the core table. If a match is not found, the old system program is transferred to the new tape. If a match is found, the program name is listed under ▽PROGRAMS REPLACED▽, and the updated version of that program is transferred from disk to the new tape. The old tape is searched forward to the XFR card of that program and processing continues with the next card. Duplicate programs, i.e. some NAM, will be replaced by their one updated version regardless of the number of times it appears on the old system tape.

When a file mark is read on the old system tape, all programs will have been transferred to or replaced on the new tape. A file mark will then be written on the new system tape.

The core table is now searched for names of programs not transferred. If found, they are listed under ▽PROGRAMS NOT USED▽.

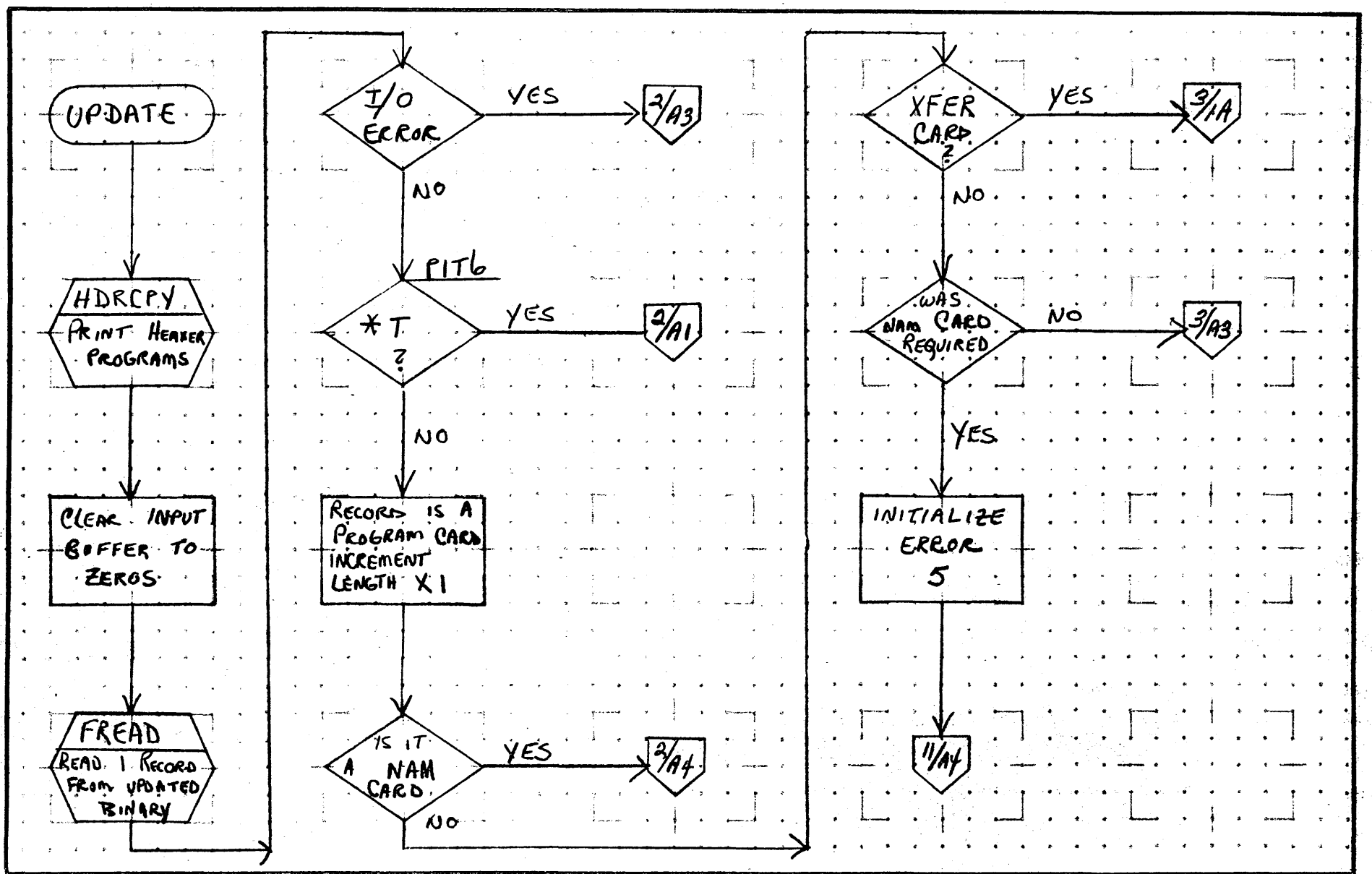
The operator is then informed ▽UPDATE FINISHED▽ and the program exits to the job processor.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE MSOS	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE UPDATE		PROJECT MGR.		
	MSOS 3.0	PAGE OF	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

61.4



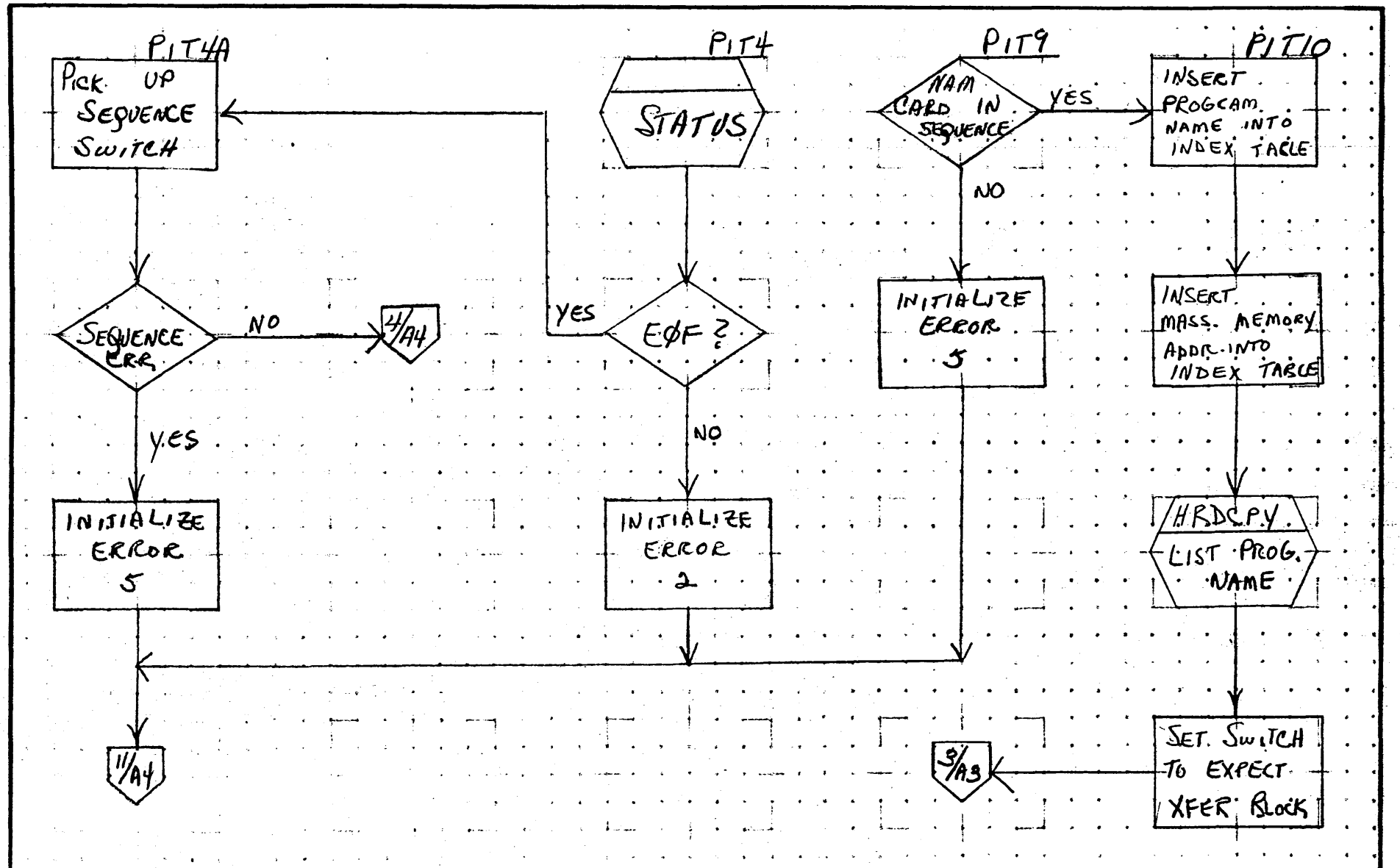
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE UPDATE	PROJECT MGR.			
	MSOS 3.0 PAGE 1 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

61.5

AA1385 (FORMERLY CA127-1)

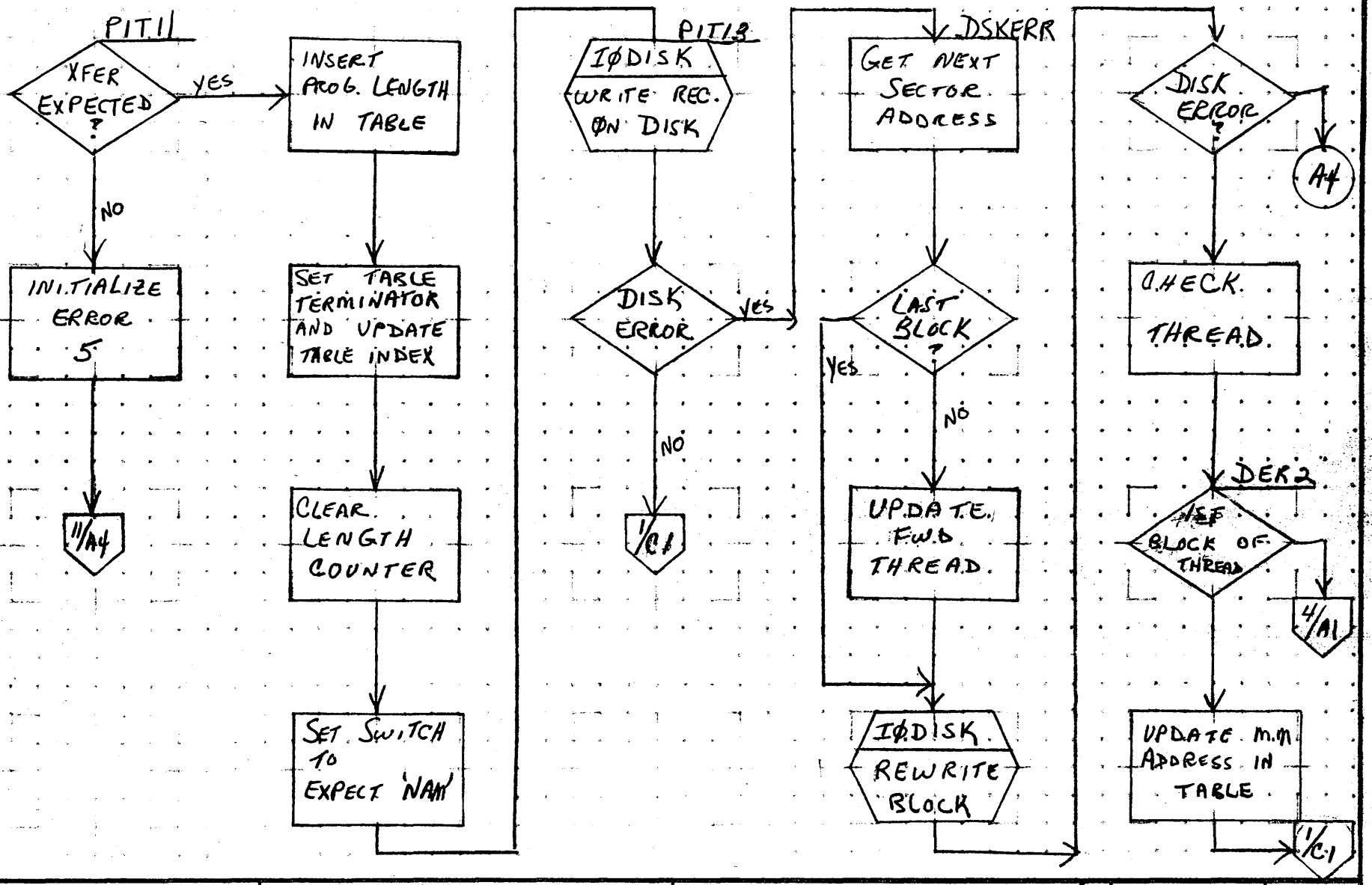
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE UPDATE		PROJECT MGR.		
	MSOS 3.0	PAGE 2 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

6.1.6

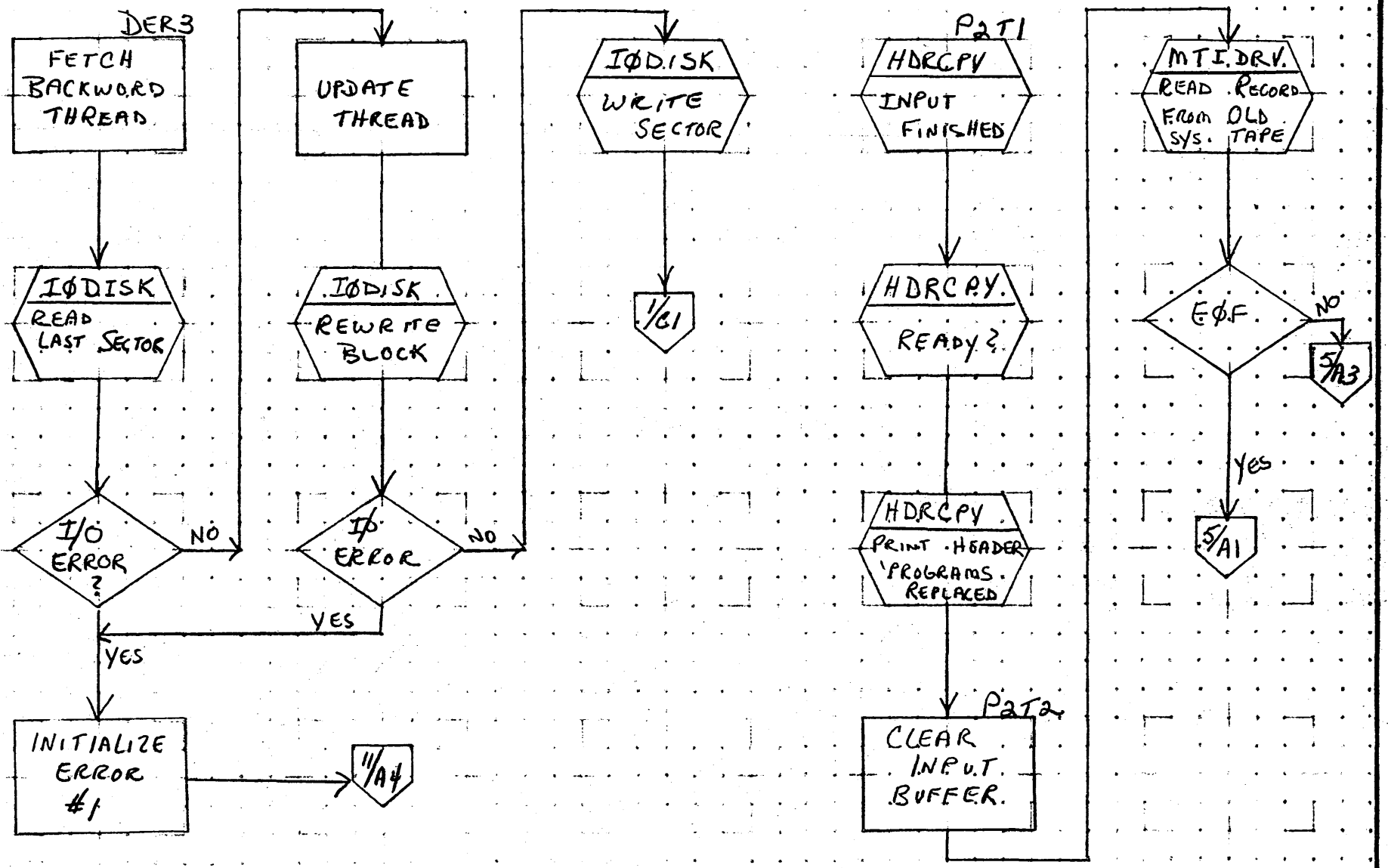


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE UPDATE		PROJECT MGR.		
	MSOS 3.0	PAGE 3 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

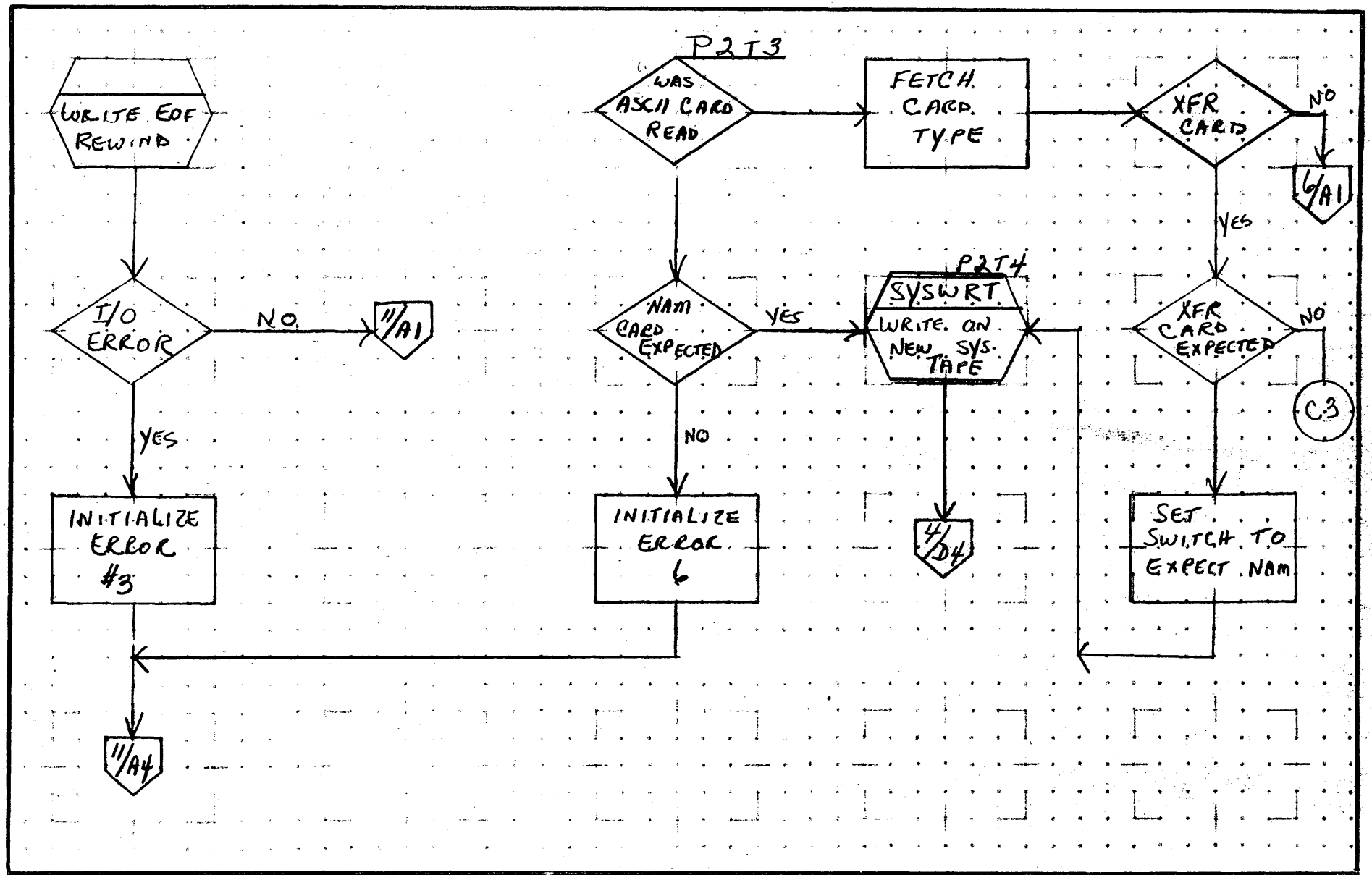
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE UPDATE		PROJECT MGR.		
	MSOS 3.0	PAGE 4 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

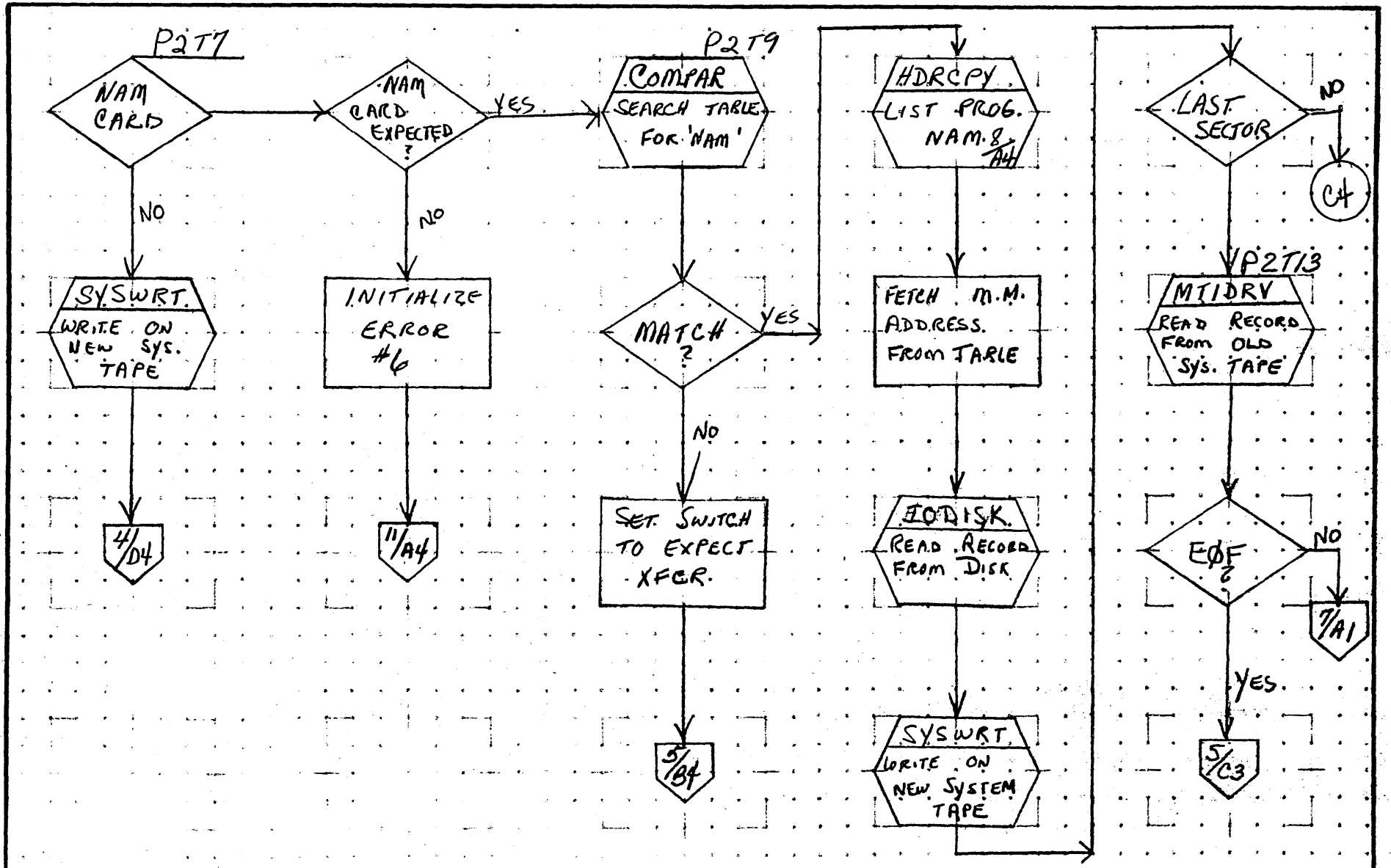
MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE UPDATE	PROJECT MGR.			
	MSOS 3.0 PAGE 5 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

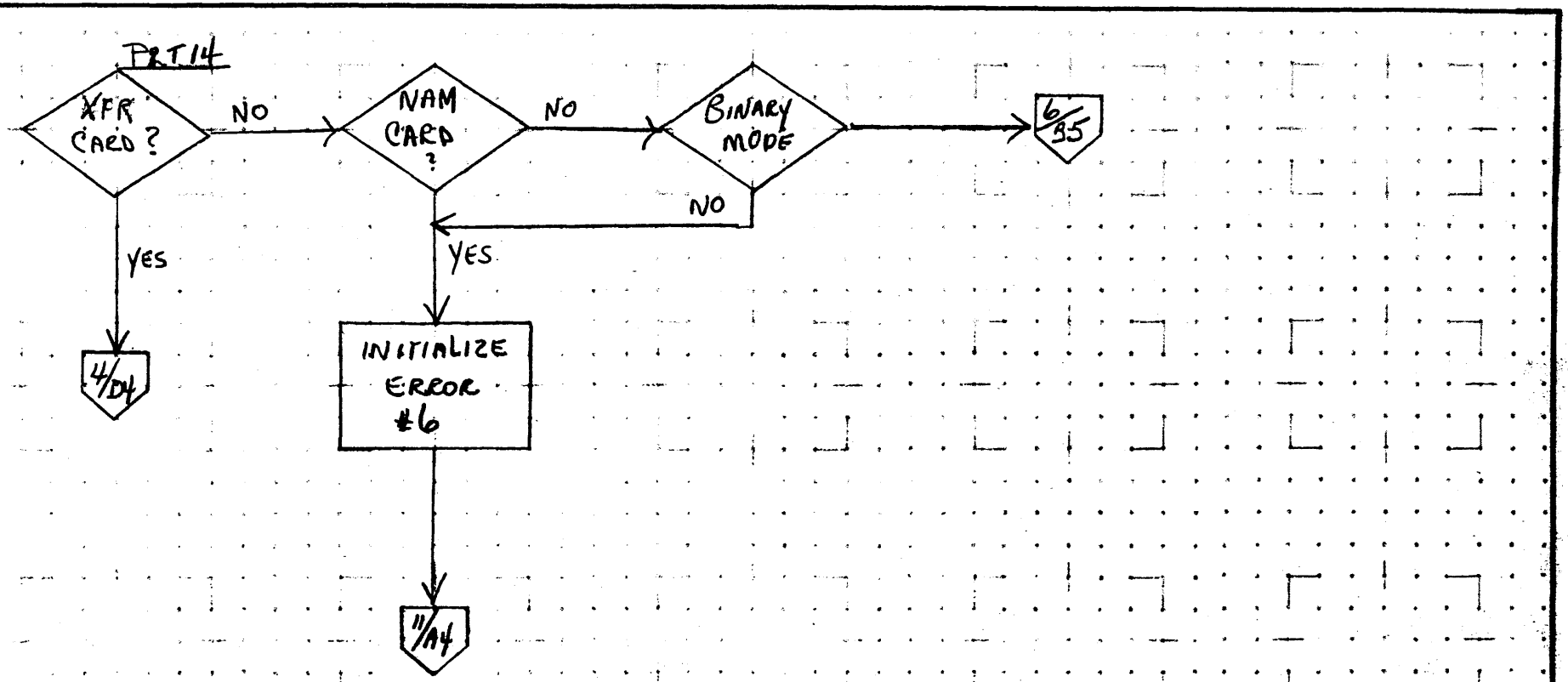
61.9



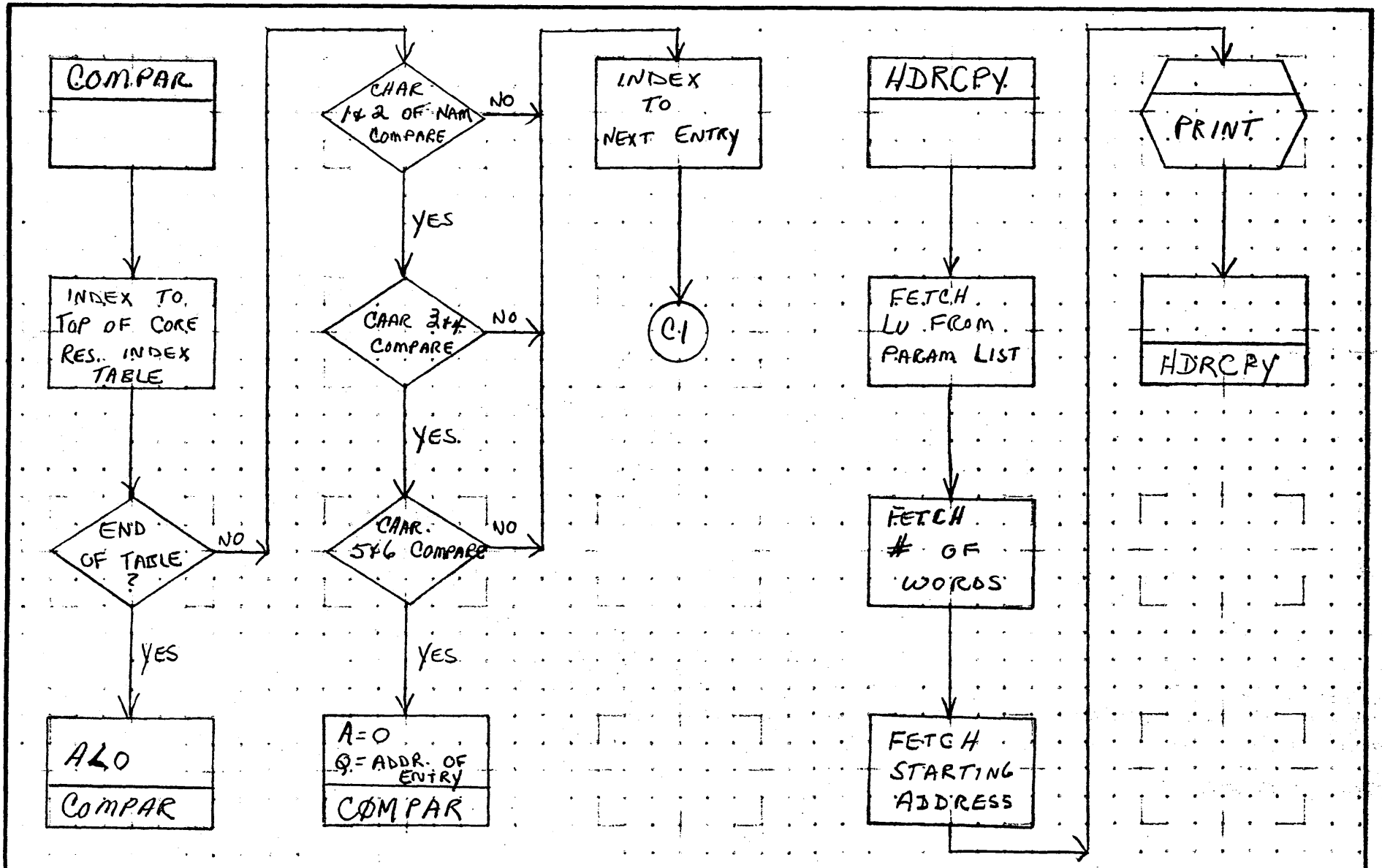
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE UPDATE		PROJECT MGR.			
FLOWCHART ☐	MSOS 3.0	PAGE 6 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

61.10



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE UPDATE	PROJECT MGR.			
	MSOS 3.0 PAGE 7 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

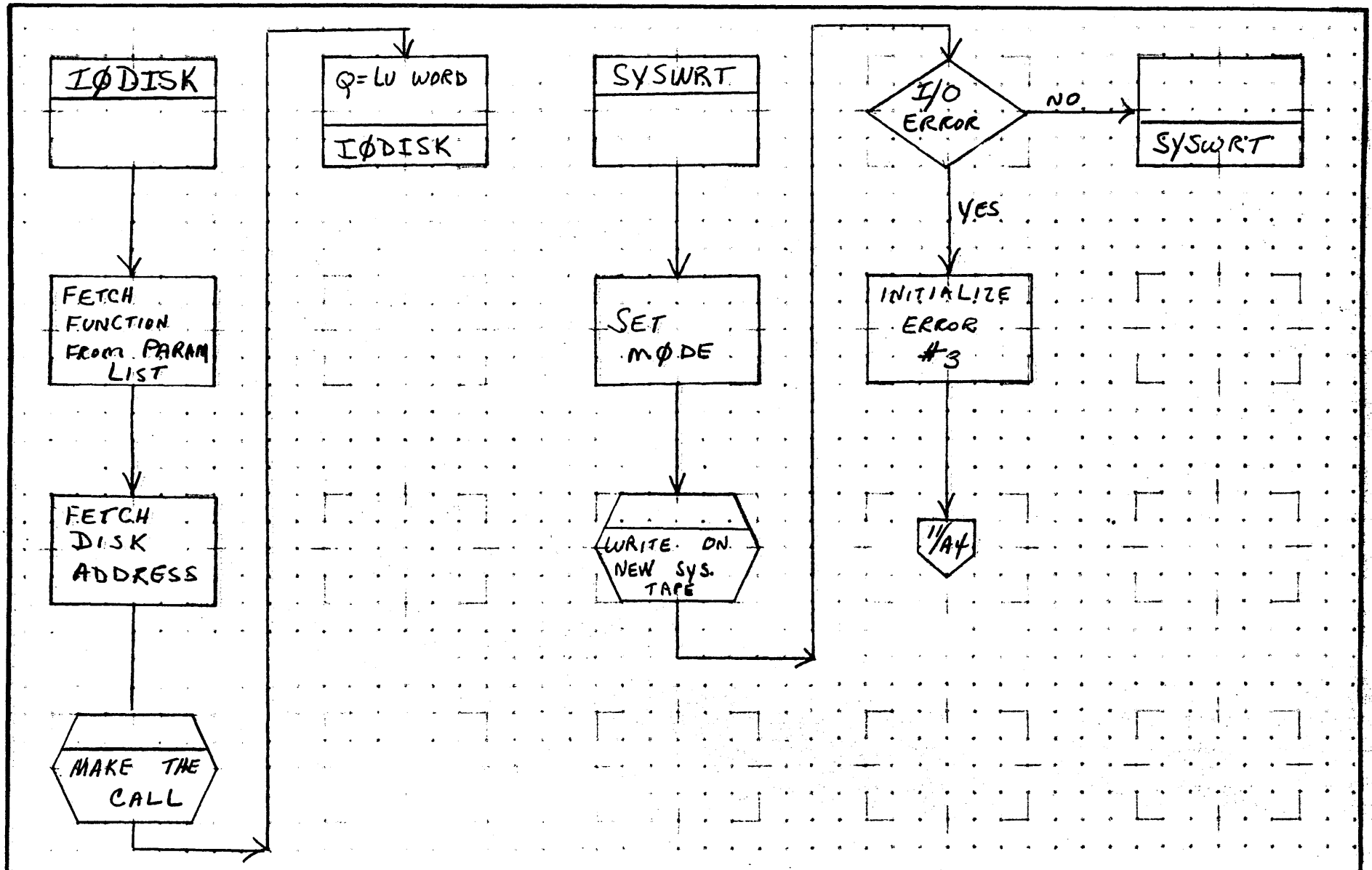


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE UPDATE		PROJECT MGR.			
FLOWCHART ☐	MSOS 3.0	PAGE 8 OF 11	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



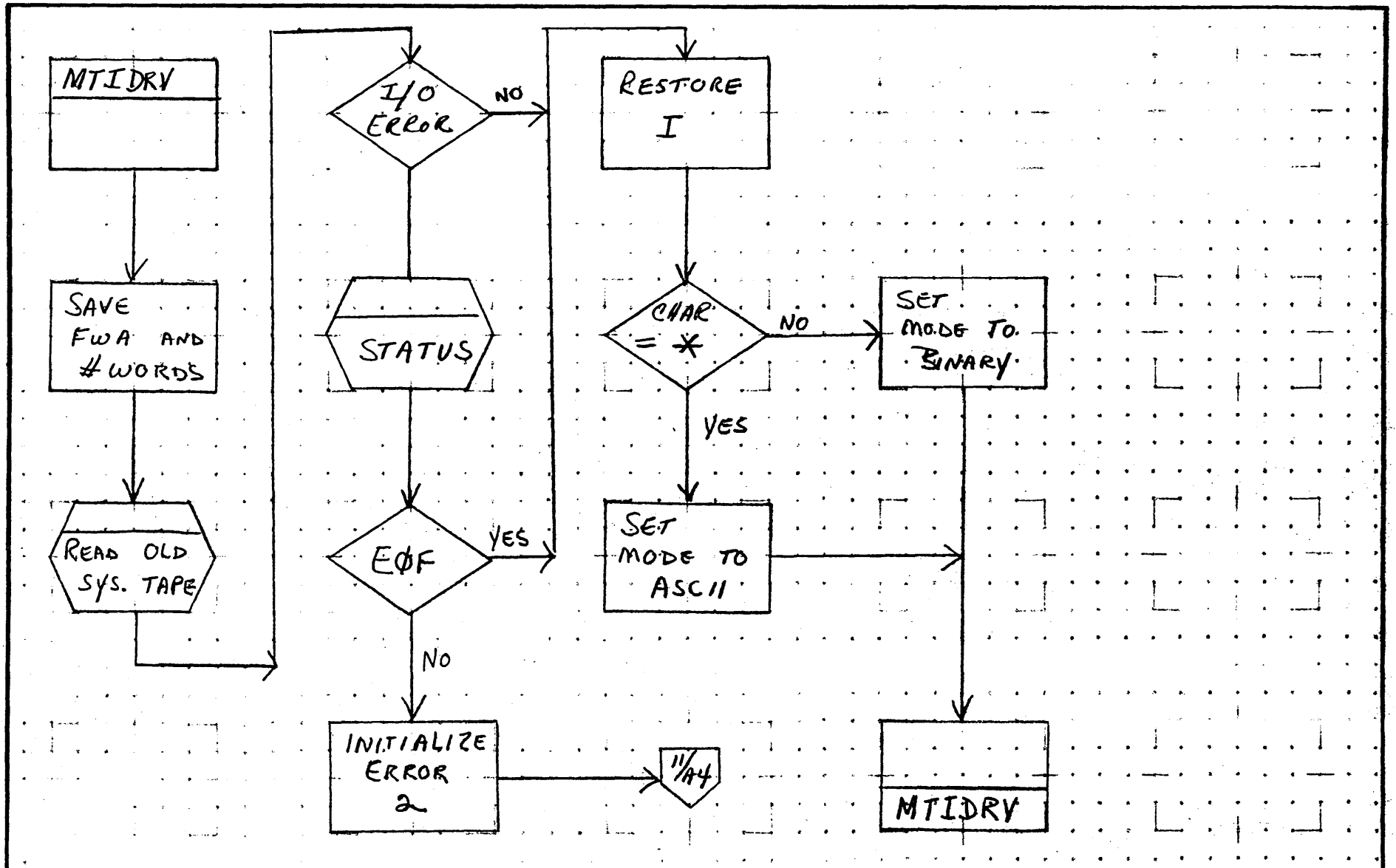
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE UPDATE		PROJECT MGR.		
FLOWCHART ☐	MSOS 3.0	PAGE 9 OF 11	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

61.13

AA1385 (FORMERLY CA127-1)

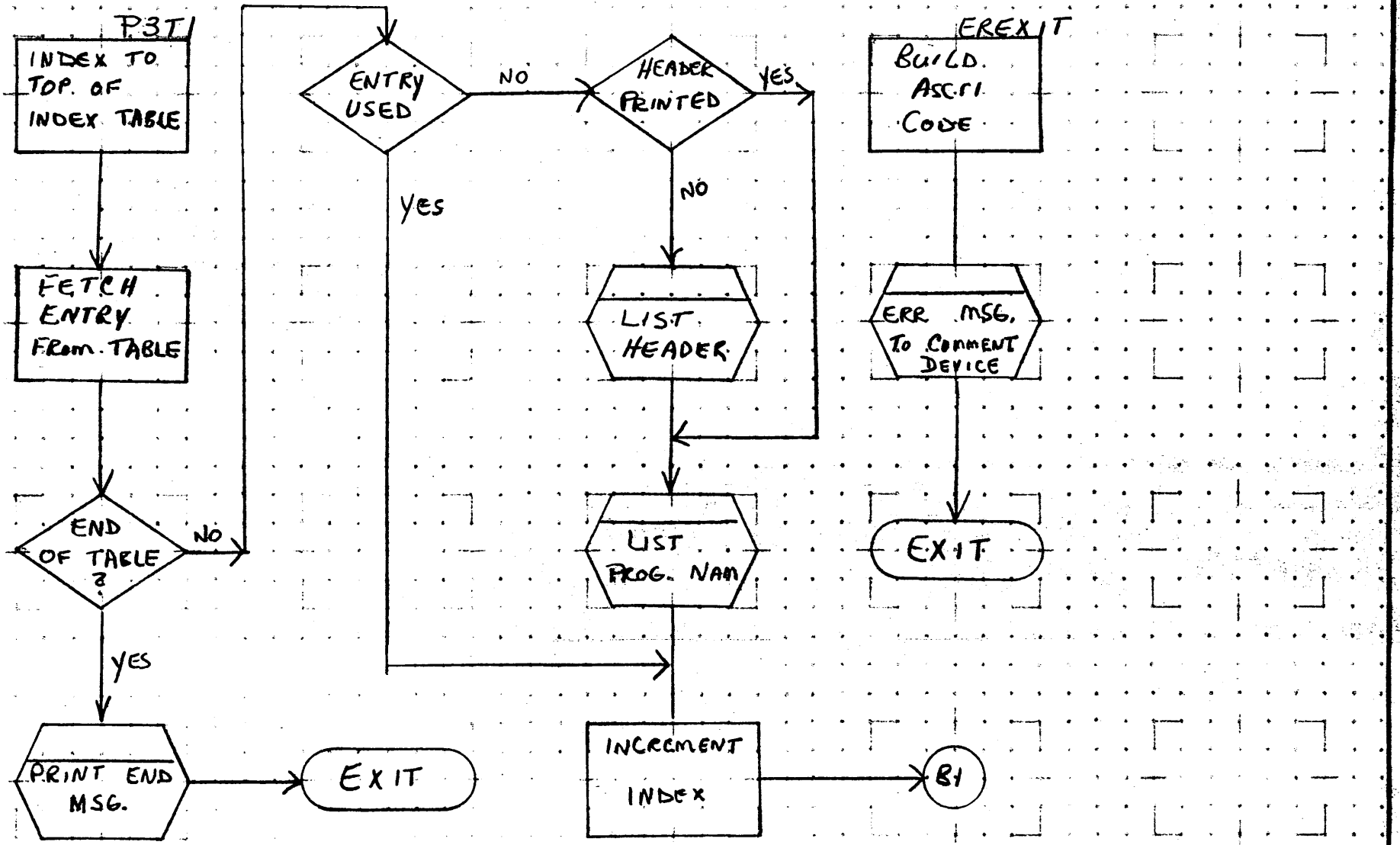
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS — MACH. TYPE: 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE: UPDATE	PROJECT MGR.	
	MSOS 3.0 — PAGE 10 OF 11	PROJECT NAME	
	NUMBER — ISSUE DATE	TASK NO.	
	DRAWN BY — DATE	TASK NAME	

MAR 5 1971

6.1.14



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE UPDATE		PROJECT MGR.		
	MSOS 3.0	PAGE 11 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

62.0 LISTR

62.1 Function

LISTR will list the name and program length of all the binary programs on a binary input device. Control statements are listed in sequence. A cumulative sum is also listed.

62.2 Entry Points

LISTR

62.3 Externals

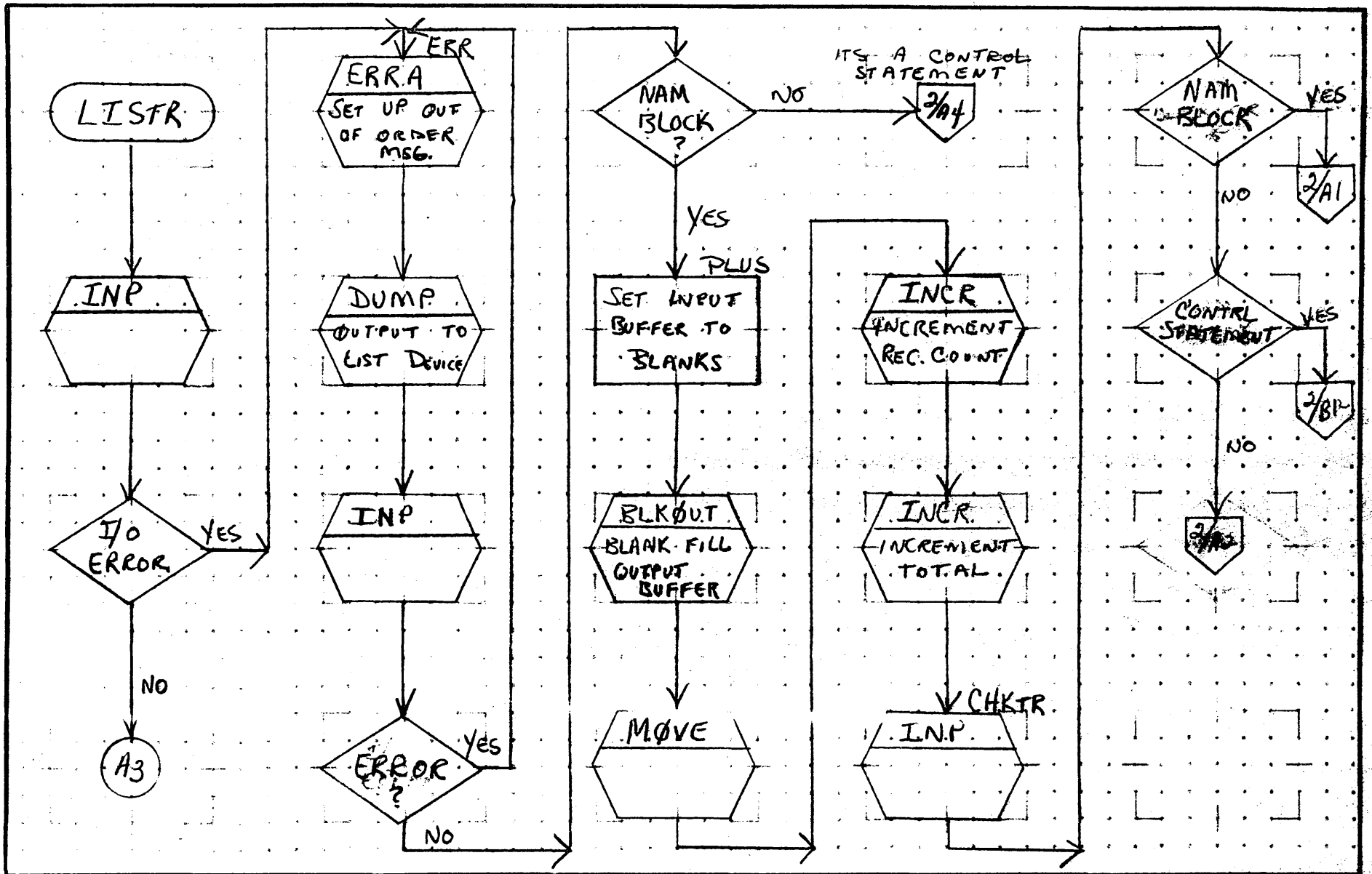
None

62.4 Entry Interfaces

None

62.5 Exit Interfaces

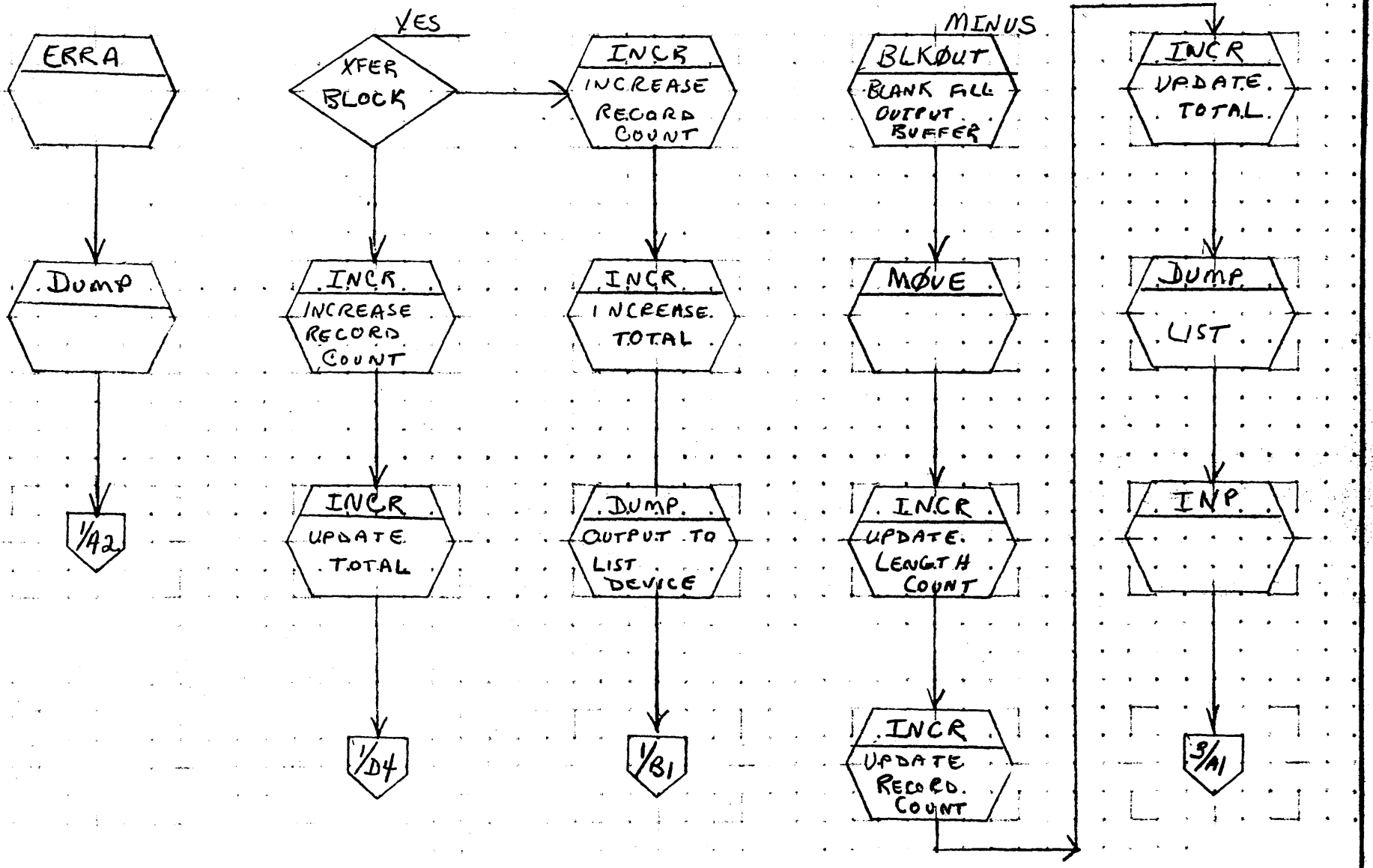
None



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: LISTR		PROJECT MGR.		
		PAGE 1 OF 6	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

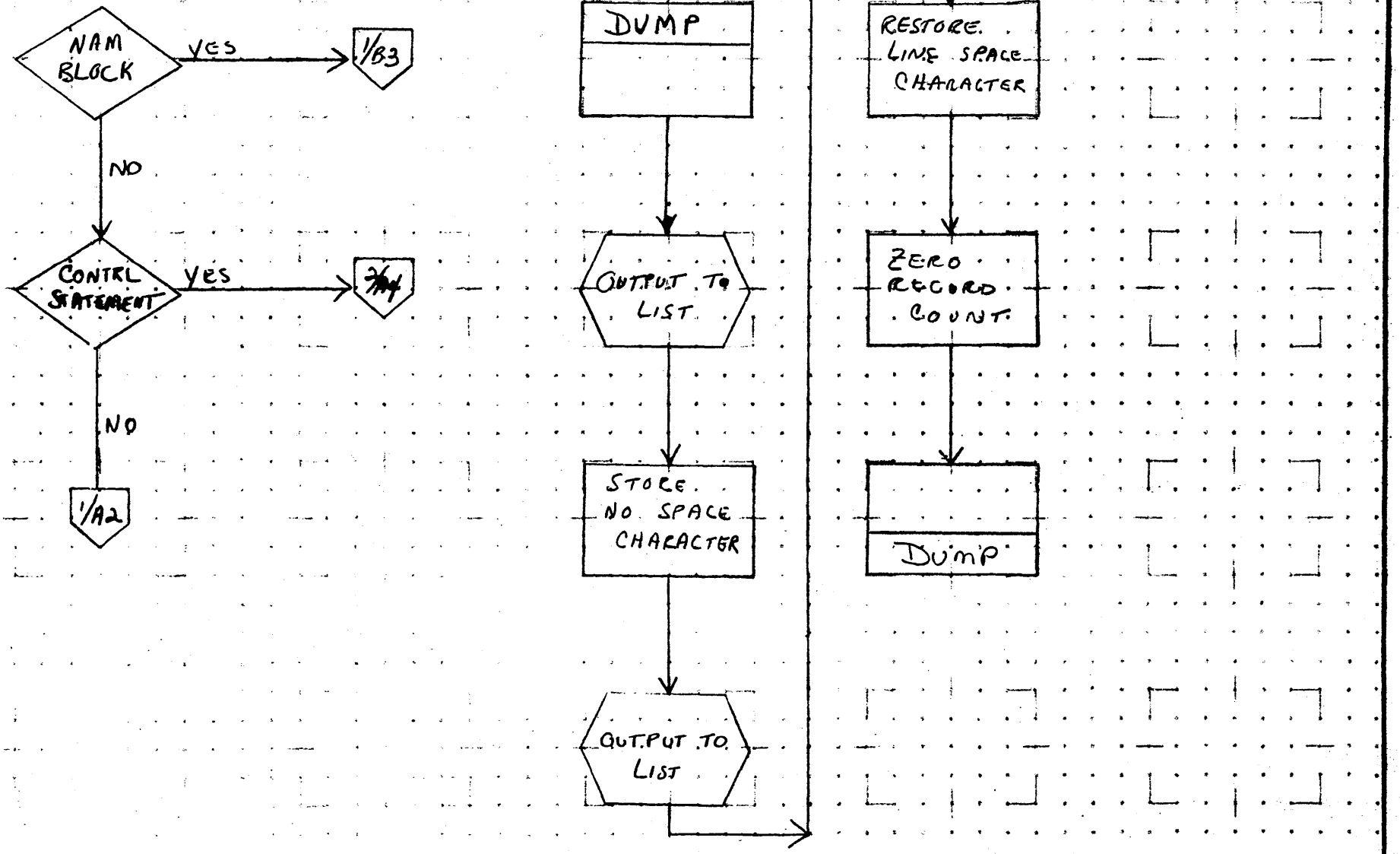
62.2



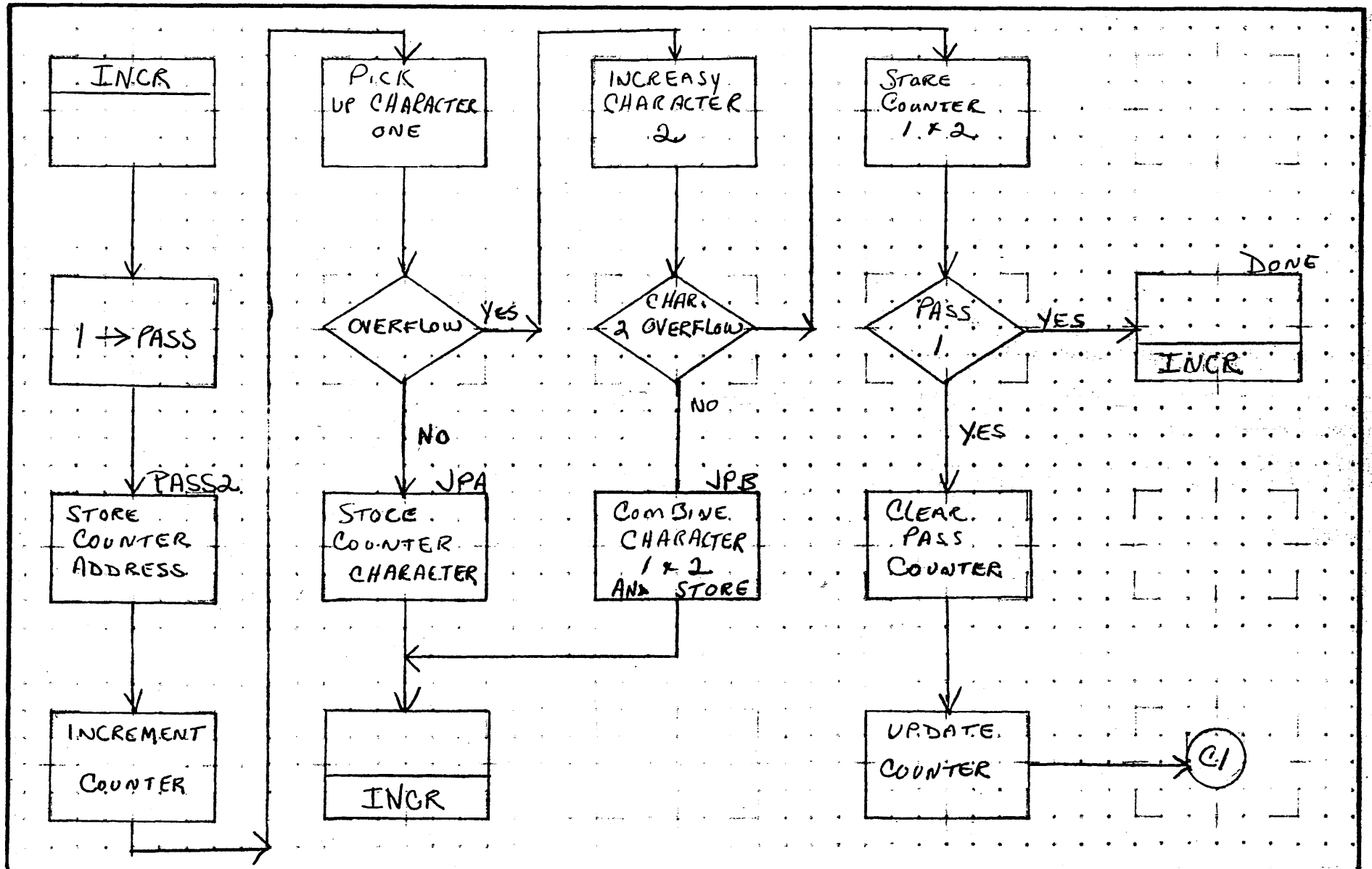
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE LISTR		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 6 1971

6.2.3



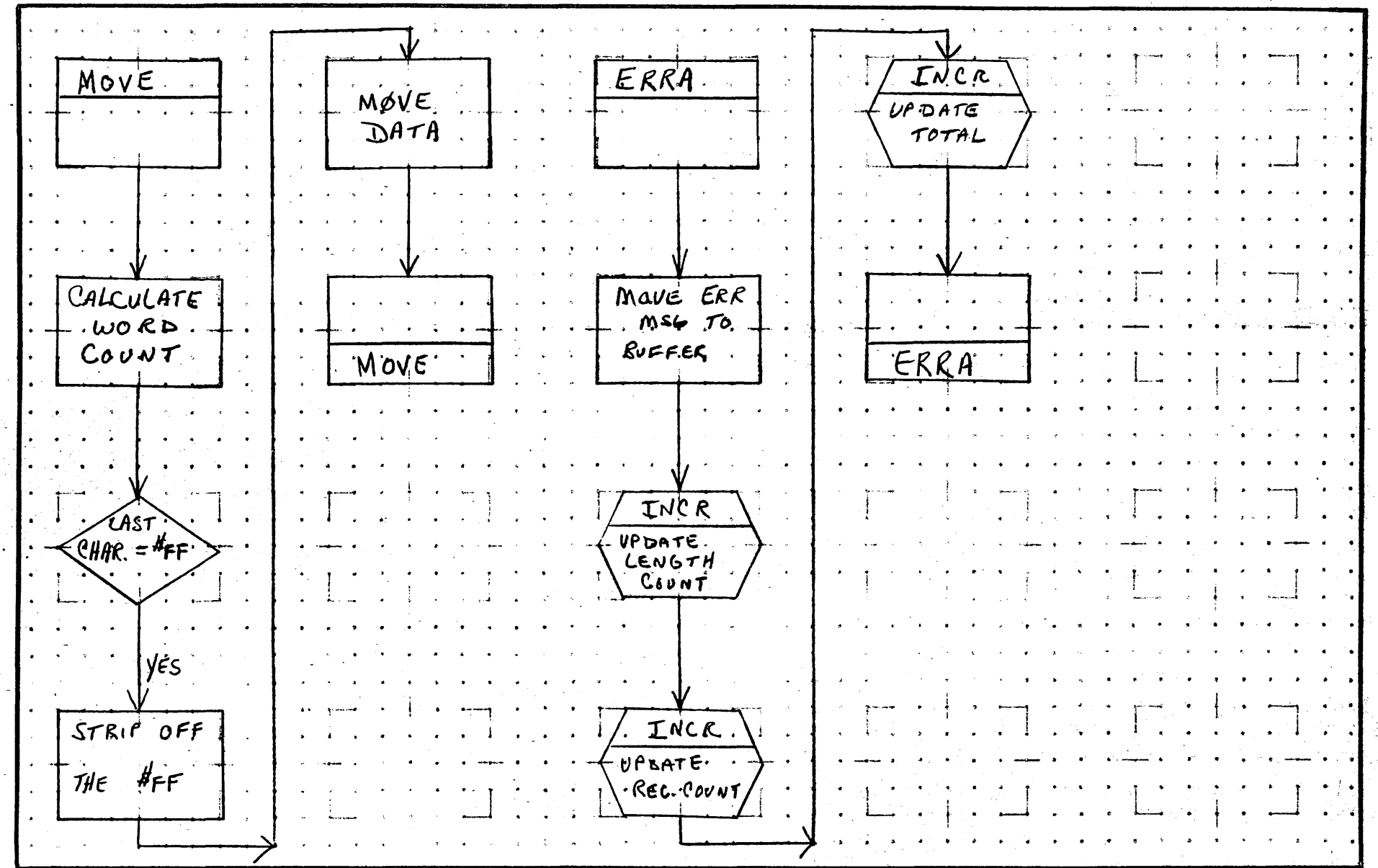
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE: LISTR		PROJECT MGR.		
FLOWCHART ☐		PAGE 3 OF 6	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LISTR	PROJECT MGR.			
	PAGE 4 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

92.5



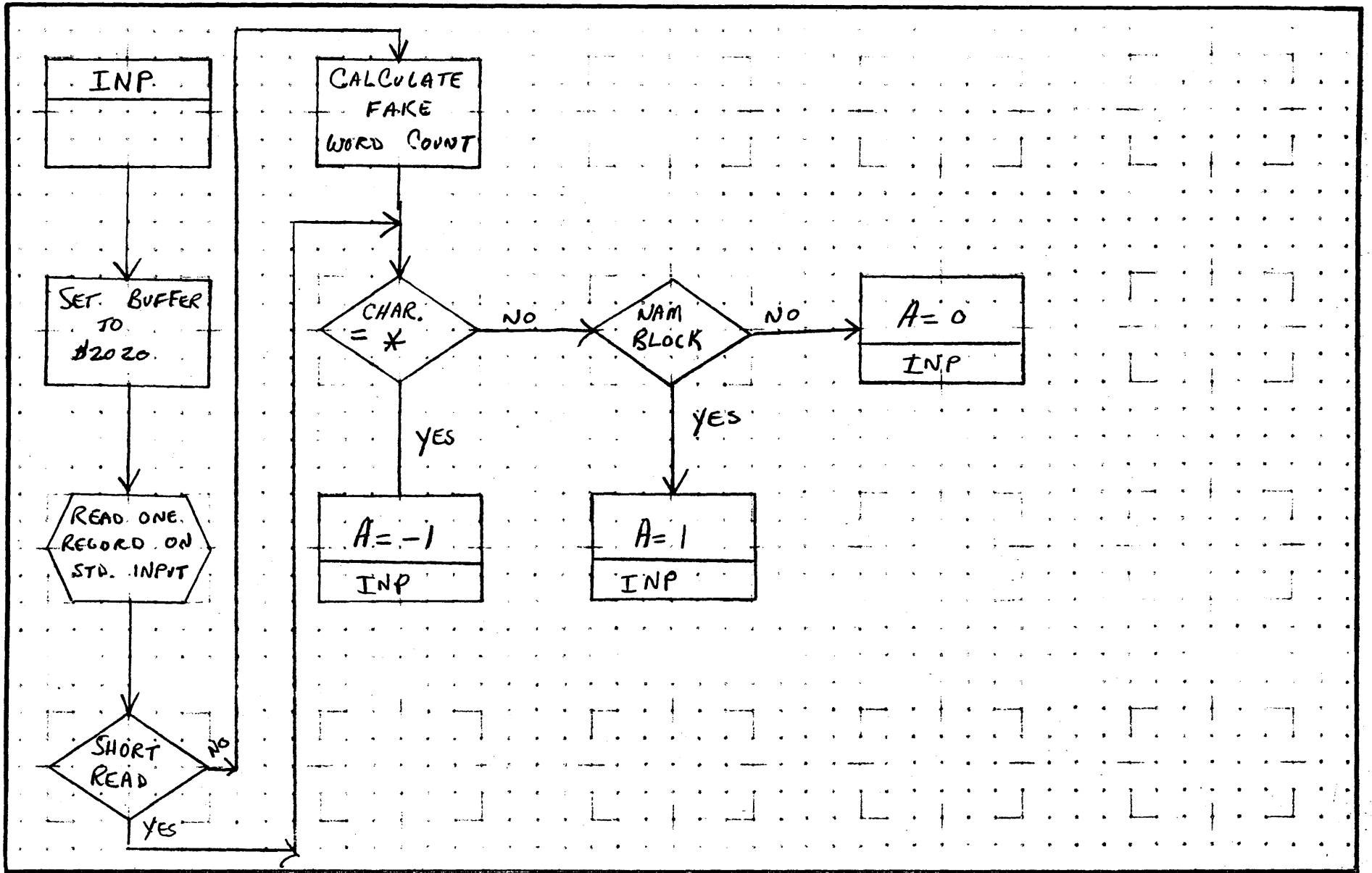
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LISTR	PROJECT MGR.			
	PAGE 5 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

42.0.0

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE LISTR		PROJECT MGR.			
FLOWCHART ☐		PAGE 6 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

62.7

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

63.0 OPSORT - Operand Sort Program

63.1 Function

The 1700 Operand Sort Program will provide a means of cross referencing 1700 assembly language operands. The program will operate on the 1700 MACRO Assemblers list output. It will read the list output from the standard input device (cards, paper tape, magnetic tape). Each record will be sorted by operand. An alphabetic listing of all operands will be output on the standard printer output device. The listing will include the source card number and operation code for each operand. If any macros are present in the program, they will be printed along with their location numbers prior to output of the sorted operands.

63.2 Entry Points

OPSORT

63.3 Externals

None

63.4 Entry Interfaces

None

63.5 Exit Interfaces

None

63.6 General Description

The following is a general flow chart of the Operand Sort Program.

PASS 1

1. Read a record from standard input device.
2. If the record is an error on list output (example **NN**** etc.) return to step 1.
3. If the record is the second record of a two word instruction (no card number) return to step 1.

4. If the first character of the record is an alpha-numeric character return to step 1 (this will occur when processing list tapes etc. Where it is necessary to advance to the NAM card of the next routine and therefore by-pass the label table of the previous routine.)
5. If the card is an END card, go to step 11.
6. Pick up the first character of the operand.
7. If the character is a left paren, pick up the second character.
8. Store the operand, card number and opcode into 1 of 41, 96 word "character buffers" based on the first (or second character of the operand. The 41 buffers are:

   10 buffers for numeric characters (0-9).
   26 buffers for alphanumeric character (A-Z).
   1 buffer for =.
   1 buffer for ¤.
   1 buffer for +.
   1 buffer for -.
   1 buffer for all other operands.

   Information is stored in the following format:

   ZZZZ YYYY XXXXXXXXXX
   where X is the operand, Y is the card number and Z is the instruction.
9. If the "character buffer" is full (9 records fill it), output the buffer on mass storage scratch. Save the sector number in the "sector table" and in the 91 word of the information written on the sector.
10. Return to step 1.
11. Output all partially filled "character buffers" on mass storage scratch, saving the sector number in the "sector table."
12. End of Pass 1. Go to Pass 2.

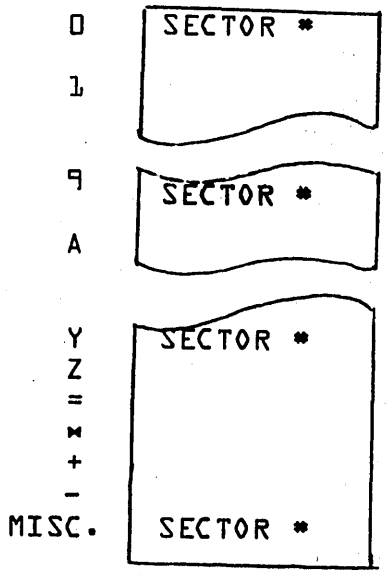
PASS 2

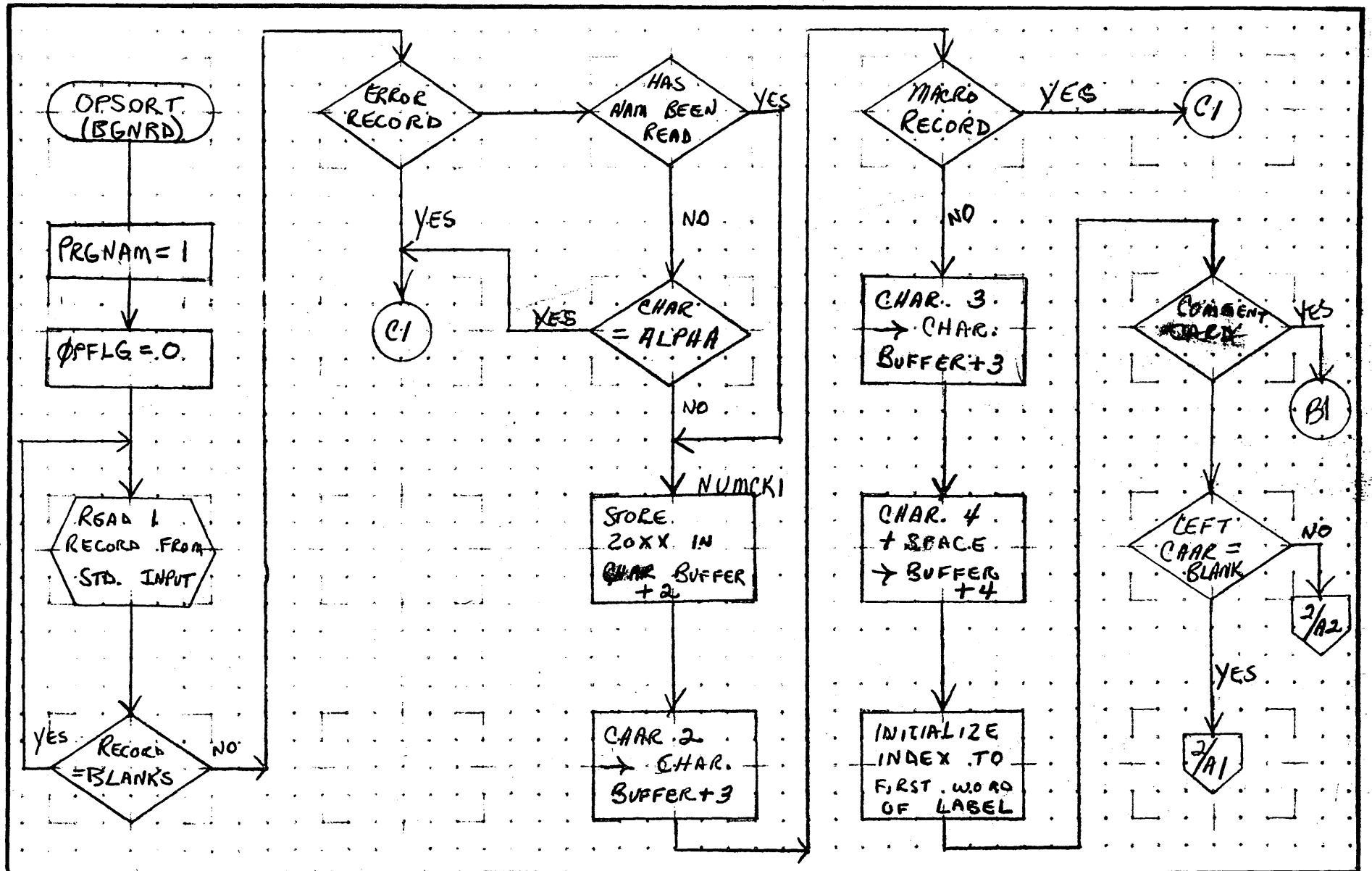
1. Using the "sector table" read all "0" character records in to memory overlaying Pass 1 of the program. If there are more than 400 operands of the nature ZZZZ YYYY XXXXXXXXXX beginning with the same character (0), a multi-pass load will be performed.

2. Sort the records based on operand and output on the standard printer output device.
3. Repeat steps 1 and 2 for the characters 1 through 9, A through Z, =, ×, +, -, and the miscellaneous characters.
4. End of Program.

Sector Table

A 41 word table containing the sector number of the last "character buffer" written on mass storage scratch for each key character the table is ordered as follows:

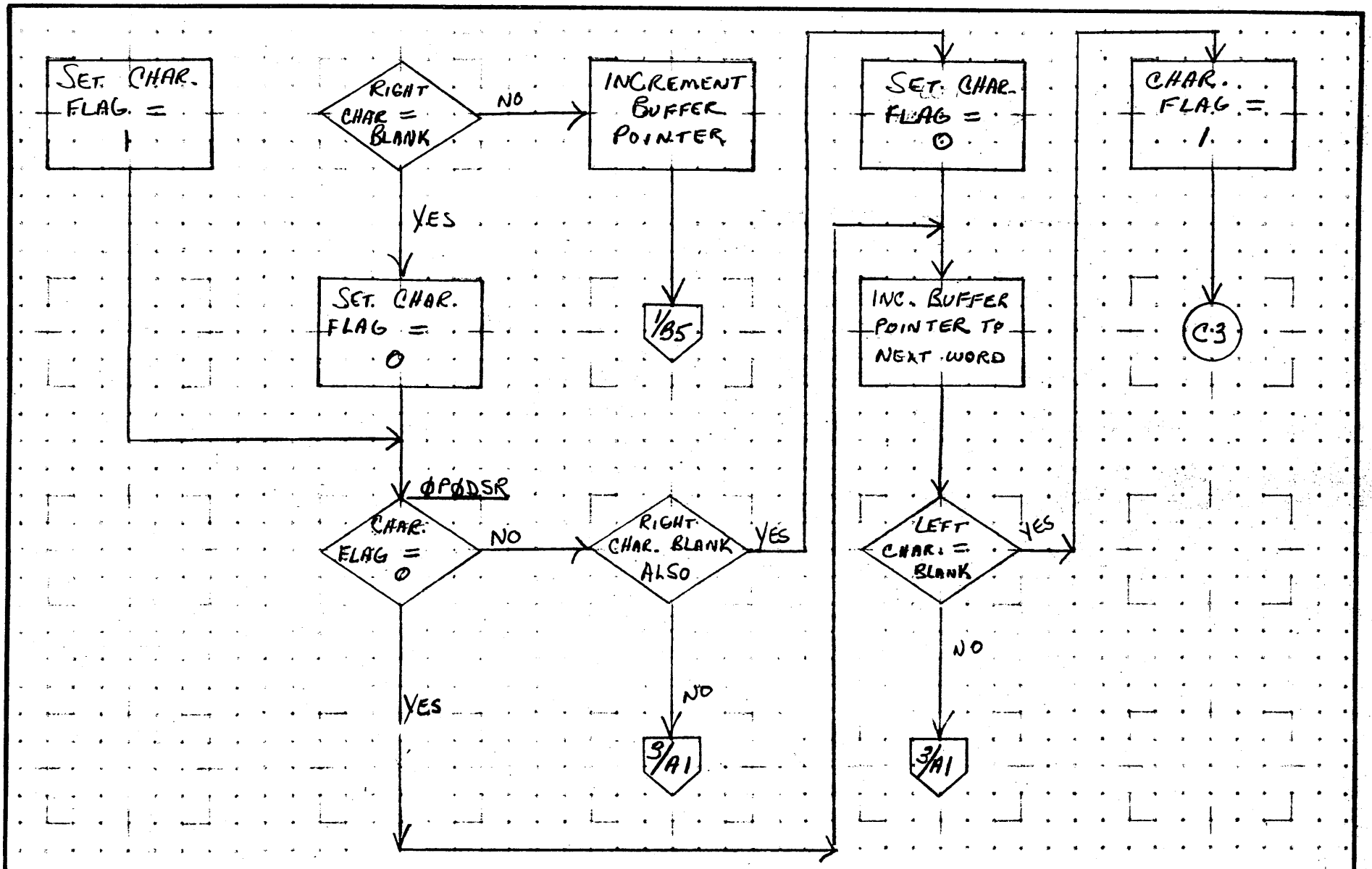




CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ØPSØRT		PROJECT MGR.			
FLOWCHART ☐		PAGE 1 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

L3.4



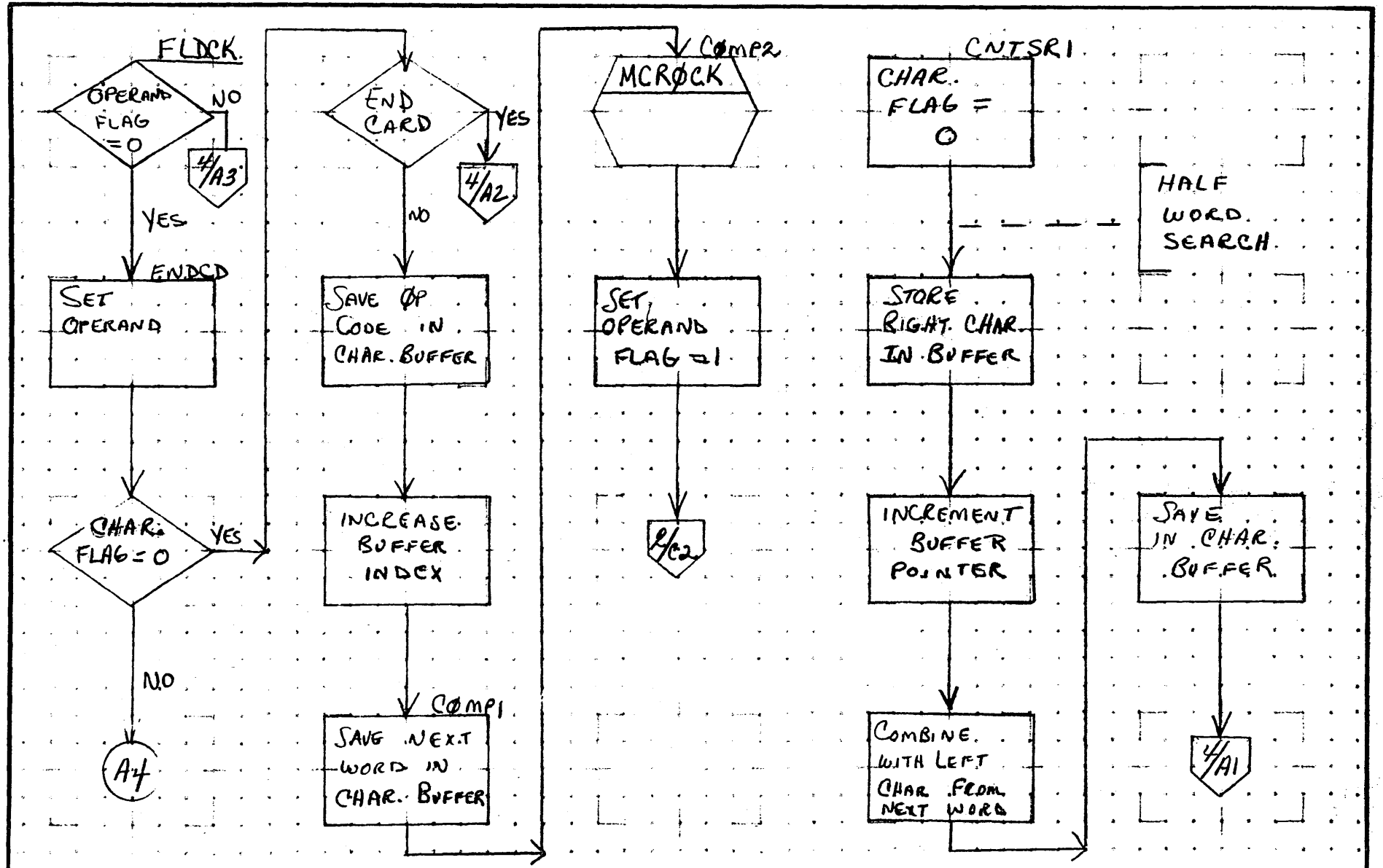
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ØPSØRT		PROJECT MGR.		
		PAGE 2 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

63.5

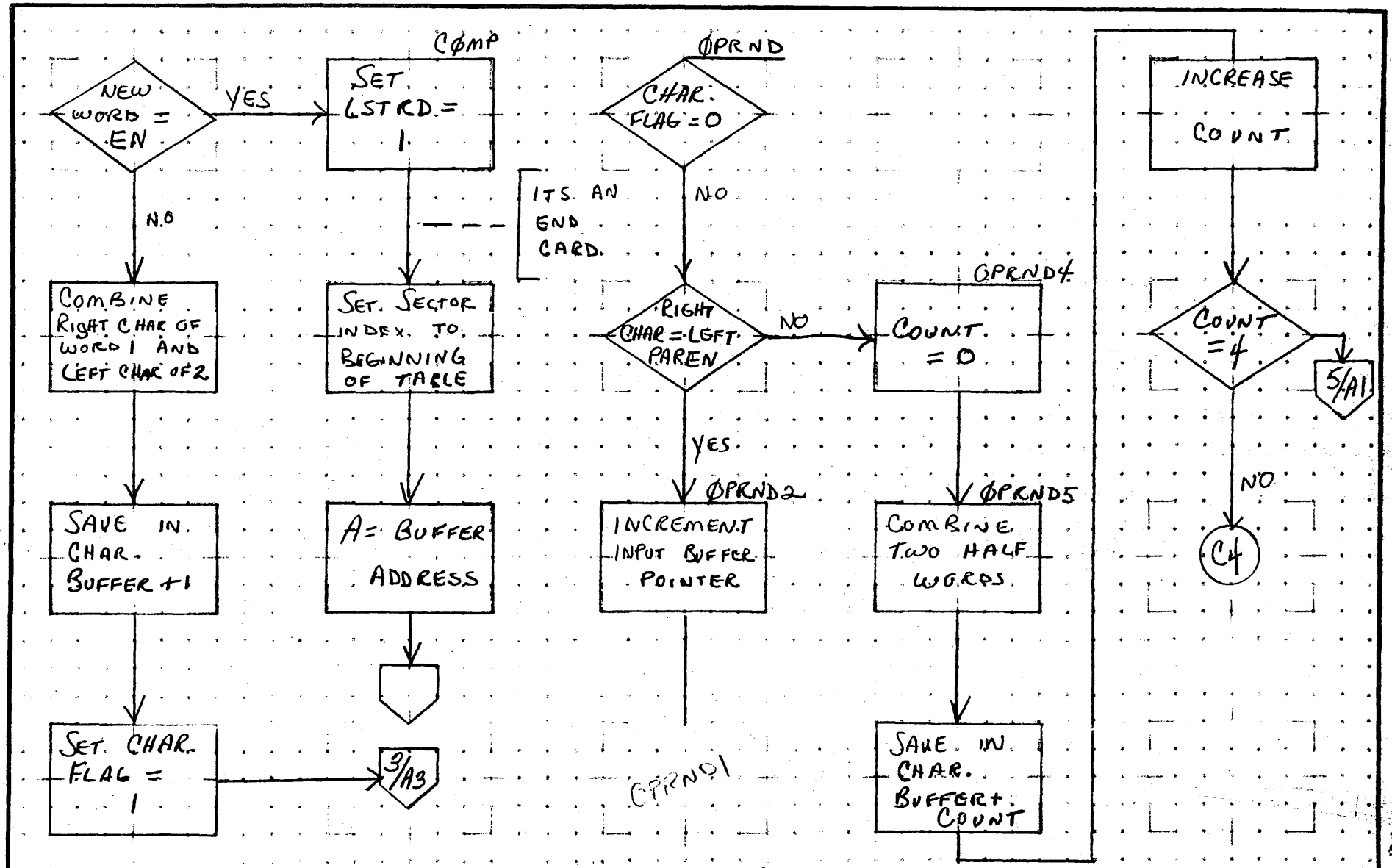


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ØPSØRT		PROJECT MGR.			
FLOWCHART ☐		PAGE 9 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



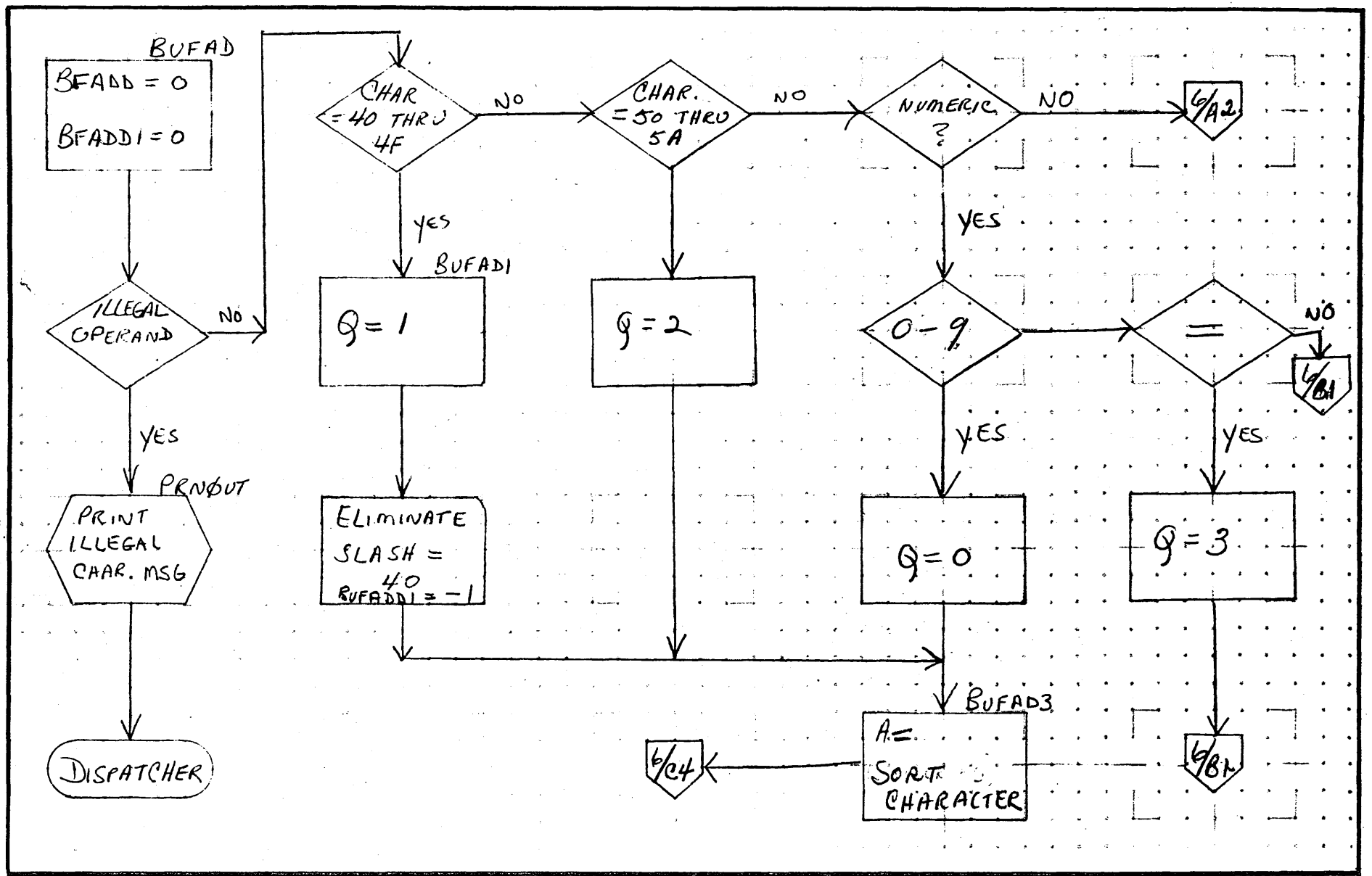
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: INS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: ØPSØRT		PROJECT MGR.		
		PAGE 4 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

63.7

AA1385 (FORMERLY CA127-1)

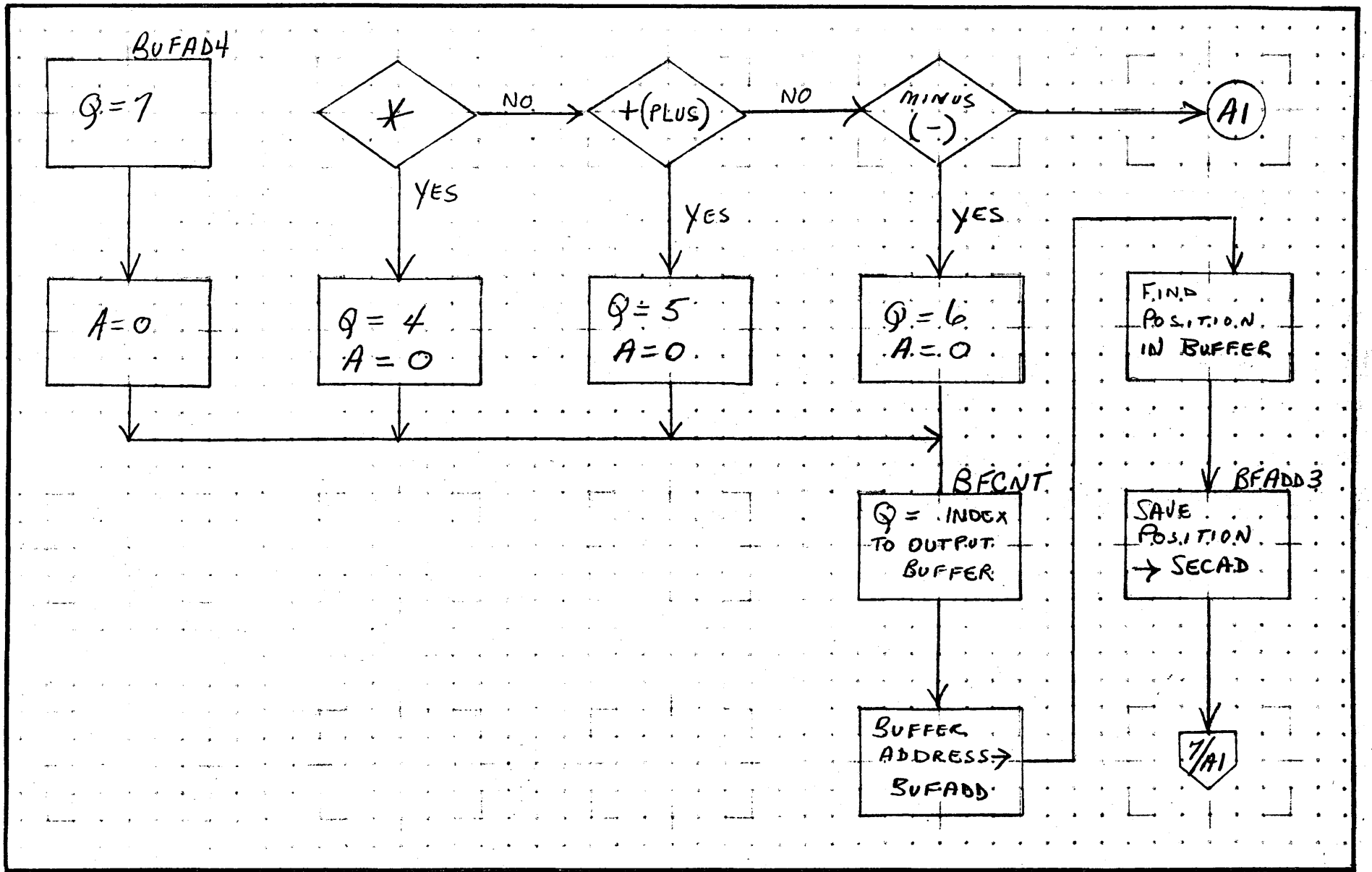
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE QPSORT		PROJECT MGR.			
FLOWCHART ☐		PAGE 5 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

5.3-8



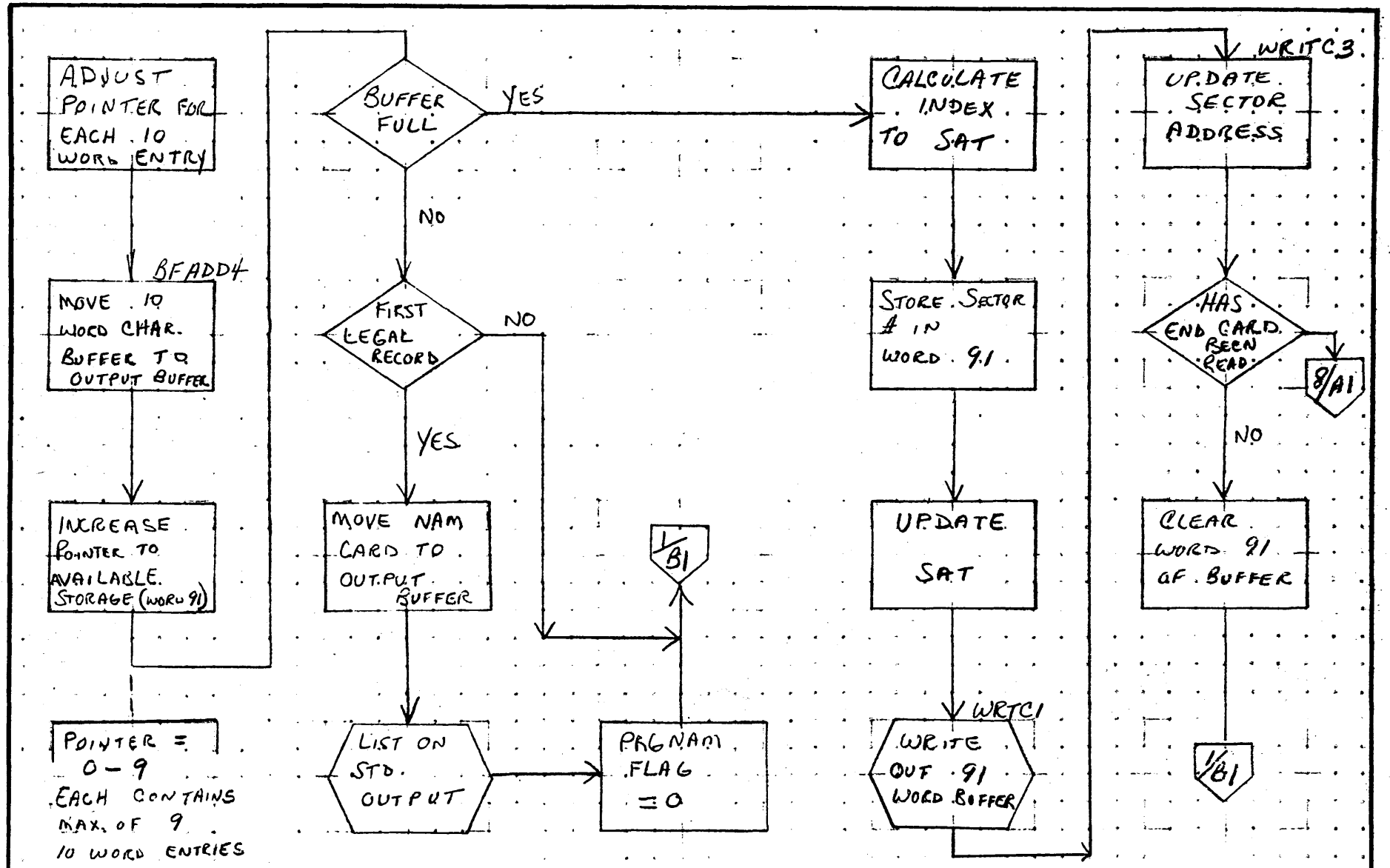
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE QPSQRT	PROJECT MGR.			
	PAGE 6 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 6 1971

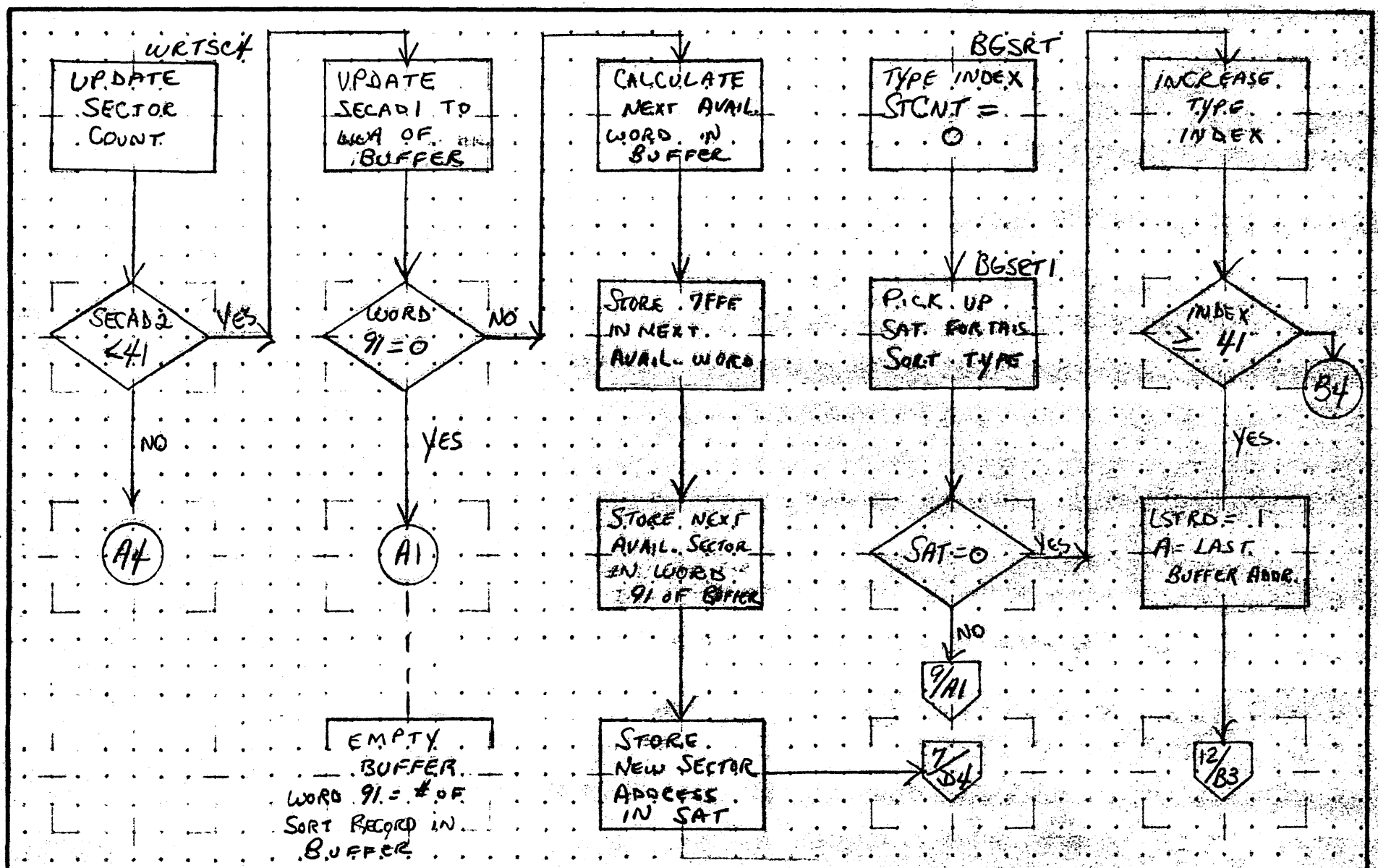
63.9



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ETSOR.		PROJECT MGR.			
FLOWCHART ☐		PAGE 7 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

63.10



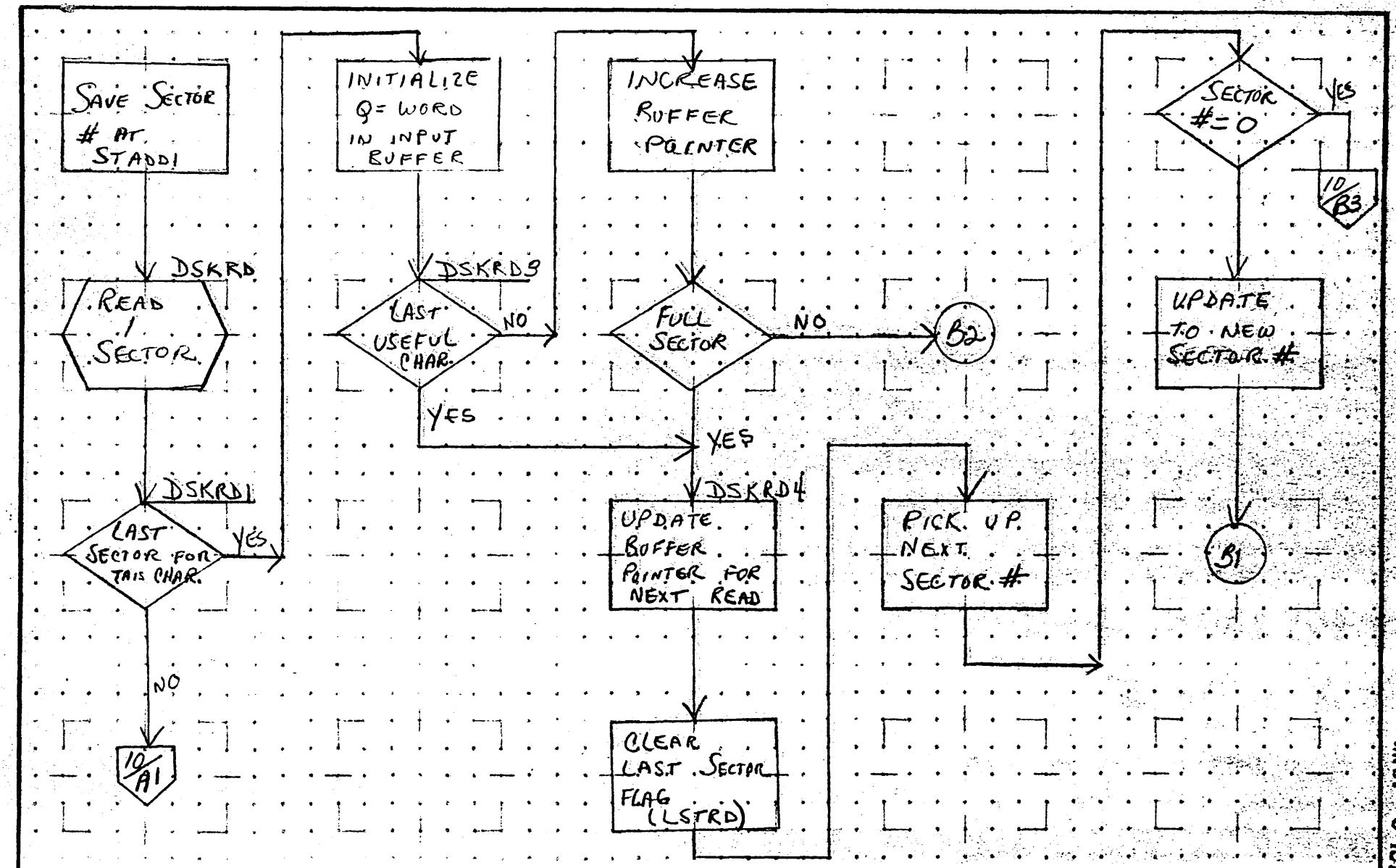
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ØPSØRT	PROJECT MGR.			
	PAGE 8 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

5.5.11

AA1385 (FORMERLY CA127-1)

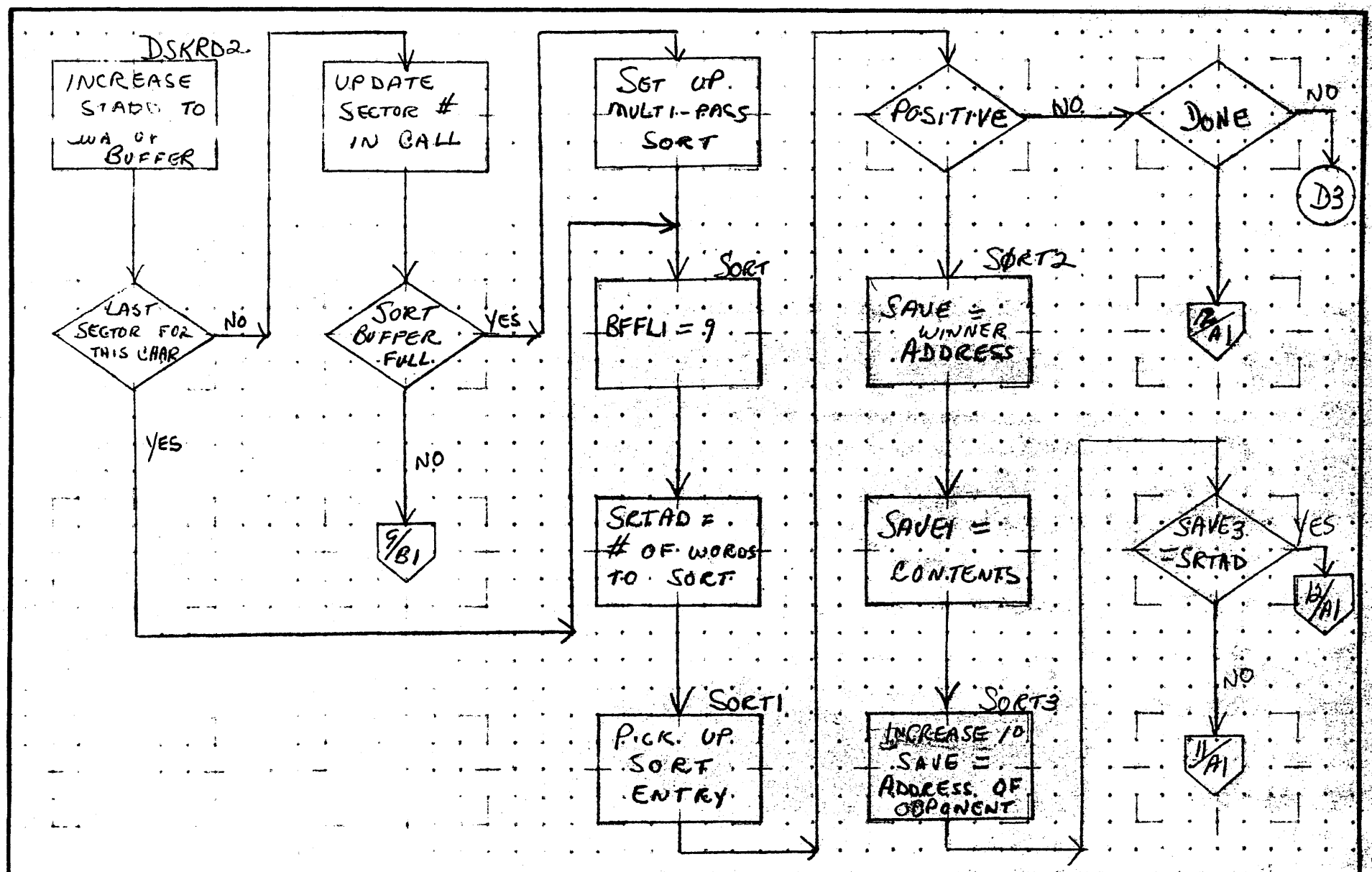
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ØPSØRT	PROJECT MGR.			
	PAGE 9 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

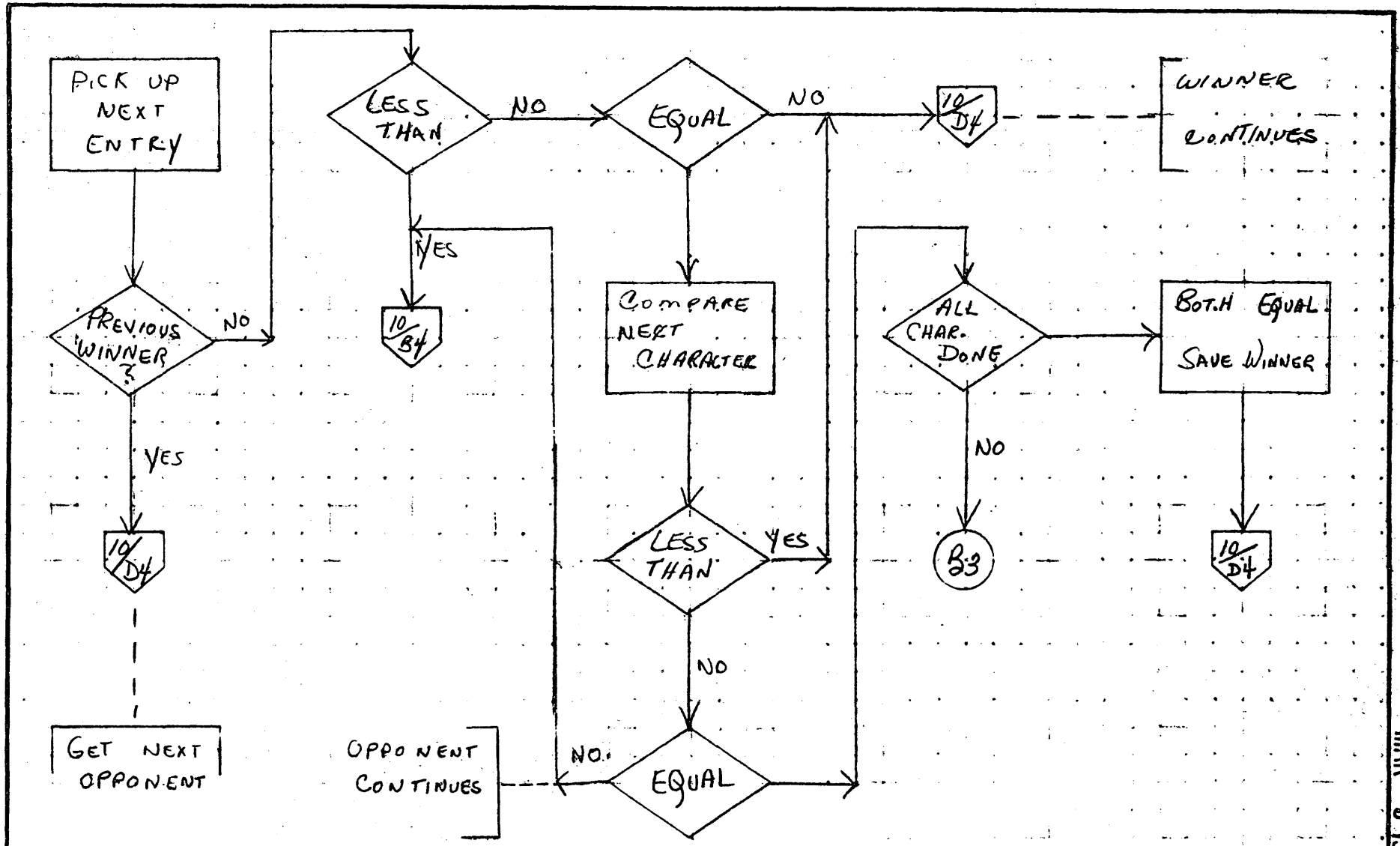
MAR 5 1971

63-12



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE OPSORT	PROJECT MGR.			
	PAGE 10 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

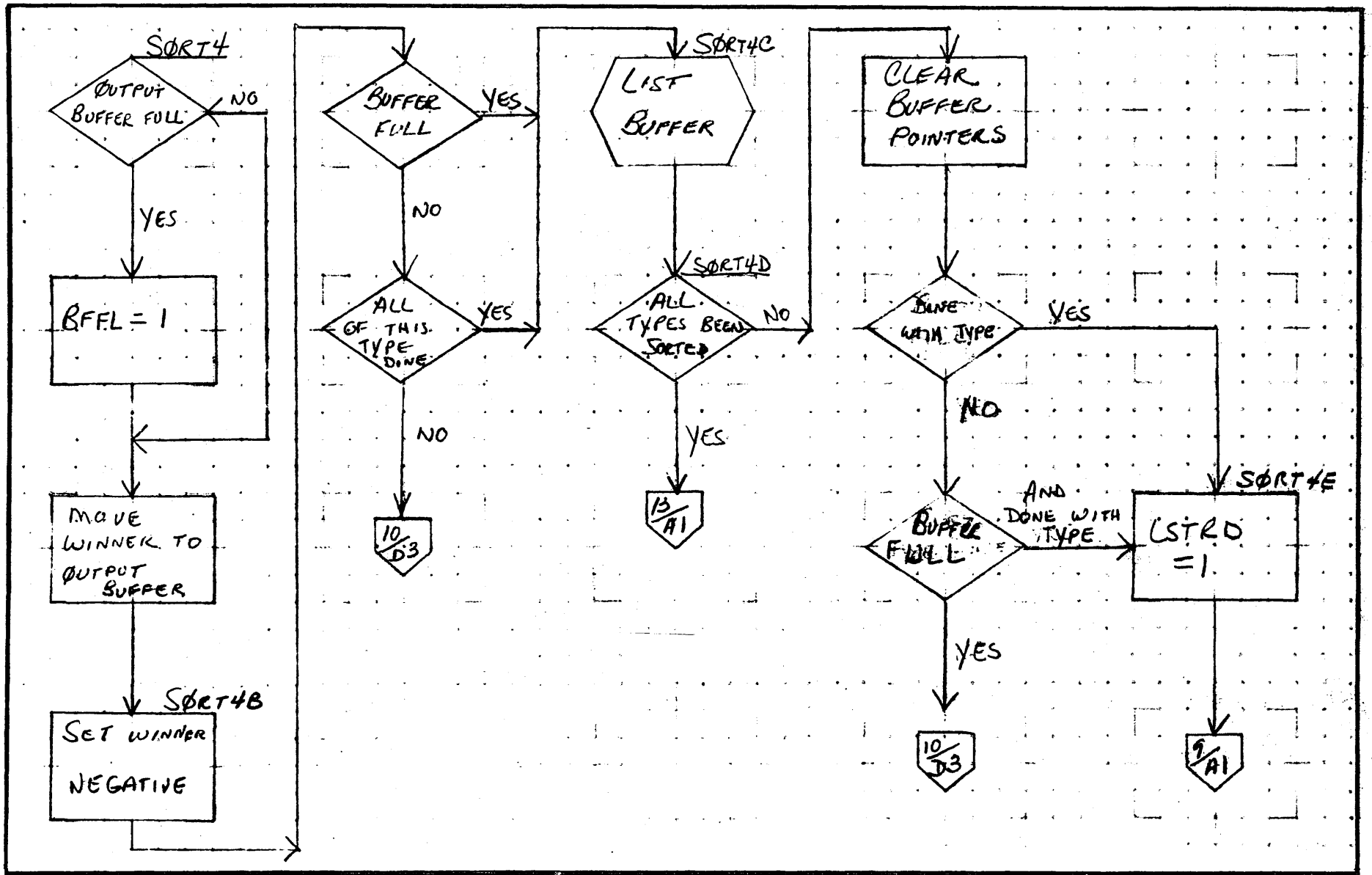
MAR 5 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IM	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ØPSØ12T		PROJECT MGR.		
		PAGE 11 OF 13	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

63.14



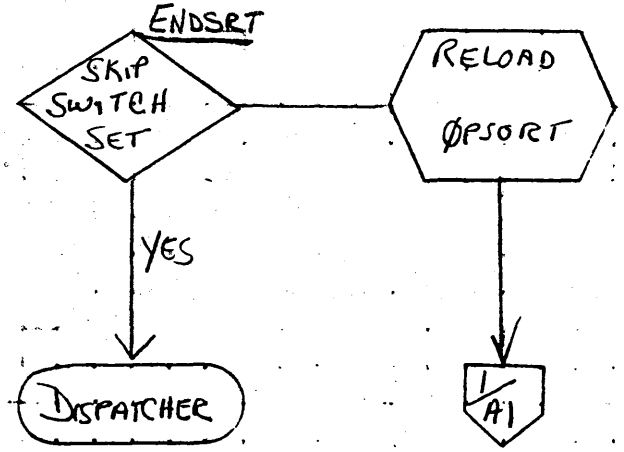
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ØFSØRT		PROJECT MGR.			
FLOWCHART ☐		PAGE 12 OF 13	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

63.1.5

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ØPSØR1	PROJECT MGR.			
	PAGE 13 OF 13	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

63.19

64.0 Fortran Monitor Runtime Package (FORTRA)

64.1 Function

The 1700 FORTRAN/Monitor Run-Time Package allows a FORTRAN written program to make requests of the 1700 Monitor, such as Read, Write, Schedule and Timer Requests. This capability allows FORTRAN programs to communicate with the Monitor through the FORTRAN subroutine Call Statement.

64.2 Entry Points

READ	Regular Read
WRITE	Regular Write
FREAD	Format Read
FWRITE	Format Write
SCHEDL	Scheduler Request
TIMER	Timer Request
DISP	Dispatcher Request
DISPAT	Dispatcher Request (alternative name)
LINK	Passed Parameter Request
CLOCK	Core Clock Request
INPINS	Input Through A,Q Channel Request
OUTINS	Output through A,Q, Channel Request
RELESE	Release Memory Request
ICONCT	Connect 1750 and Execute an Input Request
OCONCT	Connect 1750 and Execute an Output Request

64.3 Externals

None

64.4 Entry Interfaces

Refer to calling sequence, section 64.6.1.

64.5 Exit Interfaces

Refer to calling sequences, section 64.6.1

64.6 General Program Information

64.6.1 Calling Sequence

64.6.1.1 Read, Write, Format Read & Format Write

Call Name {Logical unit, buffer, length, completion location, flag-priority, temp.} where name is

a. READ — for read
b. WRITE — for write
c. FREAD — for read a format record. A format record's length is defined for each device as follows:

DEVICE	MODE	FORMAT RECORD
Card Reader	ASCII	80 columns or length of the buffer, whichever is less.
Paper Tape Reader	Binary	Word count is the complement of the first two frames of tape. Checksum is the last two frames.
Paper Tape Reader & Typewriter	ASCII	String of characters terminated by a carriage return.

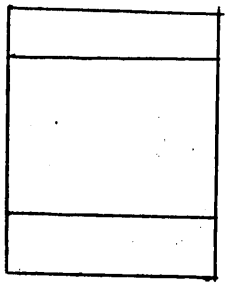
d. FWRITE — for Write a format record. A format record's length is defined as follows:

DEVICE	MODE	FORMAT RECORD
Card Punch	ASCII	80 columns or length of the buffer, whichever is less.
Paper Tape Punch	Binary	Word count N is complemented and punched as first two frames. Checksum is the last two frames.
Paper Tape Punch Typewriter	ASCII	A string of characters, terminated by a carriage return.

Buffer — An area of core which data is read into or written from.

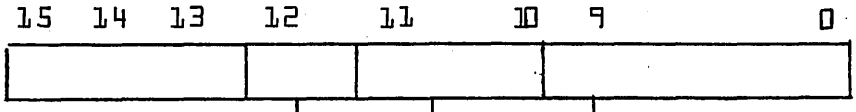
Length — The number of words in the Buffer.

If this is a mass memory logical unit, then length is the name of a three-word vector containing.



Mode-Logical Unit — The logical unit number of the device is in bits 9-0. The mode is bit 12. The logical units are detailed below.

LOGICAL UNIT NO.	MEANING	MODE
$18F9	Input medium	ASCII
$08F9	Input medium	Binary
$08FA	Output medium	Binary
$18FA	Output medium Printer	ASCII
$18FC	Output comment medium	ASCII
$18FD	Input comment medium	ASCII
$09C2	Mass memory logical unit	Binary only



A
0 Actual logical unit number
2 A core location containing the logical unit number.

LU

MODE — Used only on devices capable of both modes.

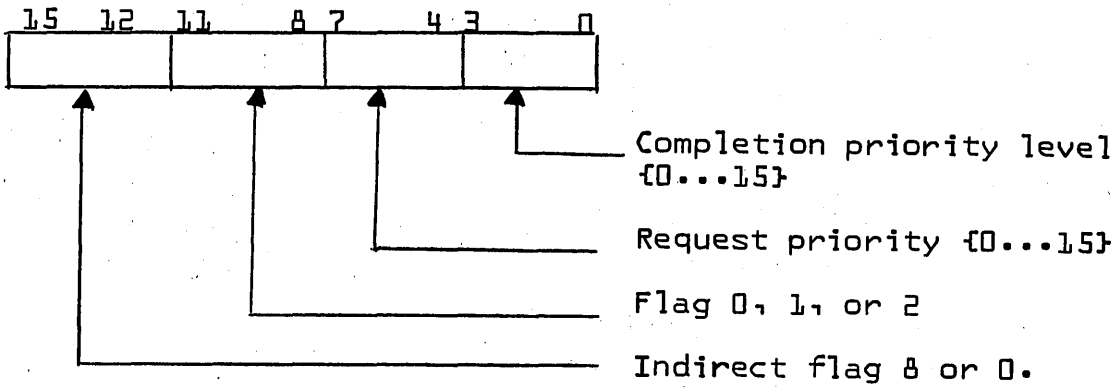
1 ASCII mode
0 Binary mode

Completion Location:

A location in a program to return control to when the I/O has finished. This may be a statement label in the same program {flag =0}, an index to the directory {flag-1}, or another program residing in Core {flag=2}.

Flag Priorities:

A 3-digit number {hexadecimal} containing the completion priority in bits 3-0, and the Request priority in bits 7-4, and a flag in bits 11-8 {refer to completion location for flag meaning}. If bit 15 is set, the actual buffer address can be found in the location specified by the calling sequence.



TEMP:

An 8-word area in which the Monitor Call will be generated.

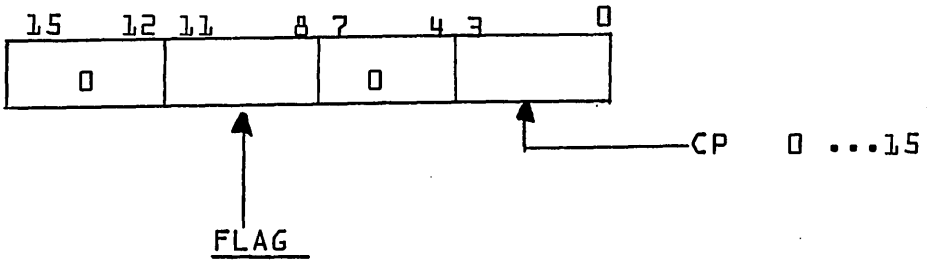
64.6.1.2 Scheduler Call:

CALL SCHEDL {L, flag-priority, parameter passed, temporary area}

L is the requested program to be scheduled at the priority CP {in the Flag-Priority word}.

Flag-Priority:

A packed word containing a flag in bits 11-8 and a completion priority CP in bits 3-0. {Bits 7-4 are not used}.



0	L is a statement label
1	L is an index to the directory
2	L is an external core resident main program

Parameter Passed:

A positive integer may be passed to the scheduled program. The scheduled program obtains the parameter by calling the integer function LINK (see Section 3.4.2.9).

Temporary Area:

A four-word temporary area in which the scheduler call is generated. After the scheduler call is complete, the temporary area is available for other use.

64.6.1.3 Timer Call:

CALL TIMER (L, Flag-Units-Priority, time interval, Temporary Area)

L is a program to give control at priority CP after the time interval has expired.

Flag-Units-Priority:

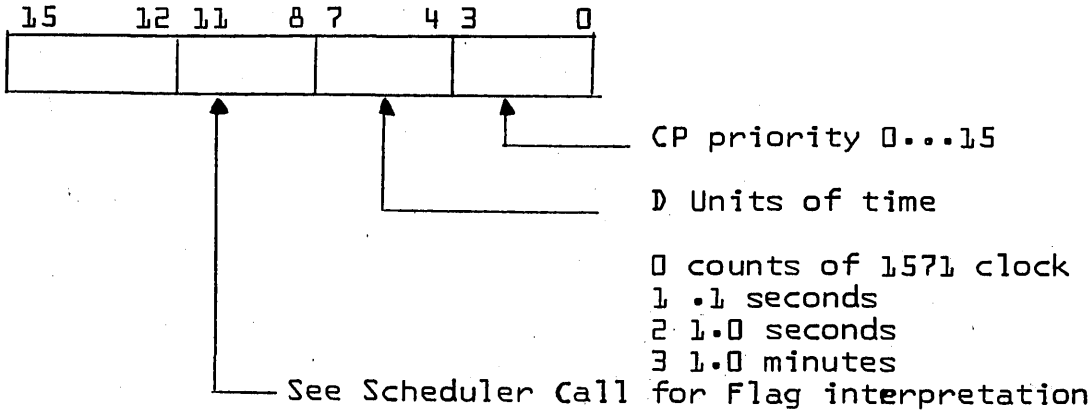
A packed word containing a flag in bits 11-8, a unit of time code in bits 7-4, and a completion priority, CP, in bits 3-0.

Time Interval:

The time interval to delay before scheduling the program L, at level CP.

Temporary Area:

A four-word temporary area used when generating the call. After the CALL has been executed, the temporary area is free for other use.



The core clock at the end of the time interval will be passed to the requested program as a parameter. To obtain this passed parameter, execute the following:

ITIME = LINK {0}

where link is a function.

64.6.1.4 DISPATCHER entry to the Monitor:

CALL DISPAT Gives control to the dispatcher. The next highest priority program will be started.

64.6.1.5 Output Commands via the 1705 A,Q Channel:

Call OUTINS where IOTAQ is a 3-word table.

IOUTAQ {1} Loaded into the Q register. Should contain Converter, Equipment, and Station codes or the channel addresses for a continuous command.

IOUTAQ {2} Loaded into the A register. Contents varies depending upon the device selected.

IOUTAQ {3} A flag word which contains the following information after the call.

0 No reject
1 Internal reject
2 External reject

64.6.1.6 Input Commands via the 1705 A,Q Channel:

CALL INPINS {IINAQ}

where IINAQ is a 3-word table.

IINAQ {1} Loaded into the Q register. Should contain converter, equipment, and station codes or the channel address for a continuous command.

IINAQ {2} After the call, contains the data or status obtained on input.

IINAQ {3} A flag word which contains the following information after the call:

0 No reject
1 Internal reject
2 External reject

64.6.1.7 Connect the 1750 and Execute an Input Command:

CALL ICONCT {IINAQ}

See 64.6.1.6 for the calling sequence interpretation.

64.6.1.8 Connect the 1750 and Execute an Output Command:

CALL OCONCT {IOTAQ}

See 64.6.1.5 for the calling sequence interpretation.

64.6.1.9 Obtaining the Passed Parameter:

A program started by a schedule call or timer call may obtain the parameter passed to it by calling the integer function LINK. The calling sequence is:

```
I = LINK (0)
```

This statement must be the first executable statement of the program. The completion location of a read/write call has an error code passed to it, which may be obtained by calling the integer function LINK.

```
I = LINK (0)
```

If I is positive, then no error occurred in the Read or Write call. If I is negative, then an error occurred.

64.6.1.10 Obtaining the Core Clock:

The integer function ICLOCK obtains the current value of the clock. For example:

```
I - ICLOCK (0)
```

I contains the current value of the core clock.

64.6.1.11 Release Allocated Memory Request:

```
CALL    RELESE (MAIN)
```

where MAIN is the main program name.

64.6.2 Reentrancy

The program may be either reentrant or non-reentrant.

Reentrant - volatile storage is used to hold temporarily perishable data.

Non-reentrant - an area within the program is used for temporary storage of parameters.

64.7 Program Description:

64.7.1 Read, Write, Scheduler, and Timer Monitor Calls:

Each Entry to the FORTRAN/Monitor Run Time Package performs an initialization sequence in which the volatile allocation routine is called and the return address is saved. The calling sequence parameters are then interpreted and the appropriate absolute monitor call is generated. The area in which the call is then executed using an indirect reference to the generated parameters.

The FORTRAN program may have the parameters located anywhere in core.

64.7.2 Miscellaneous Calls:

Input and Output Instruction Routines:

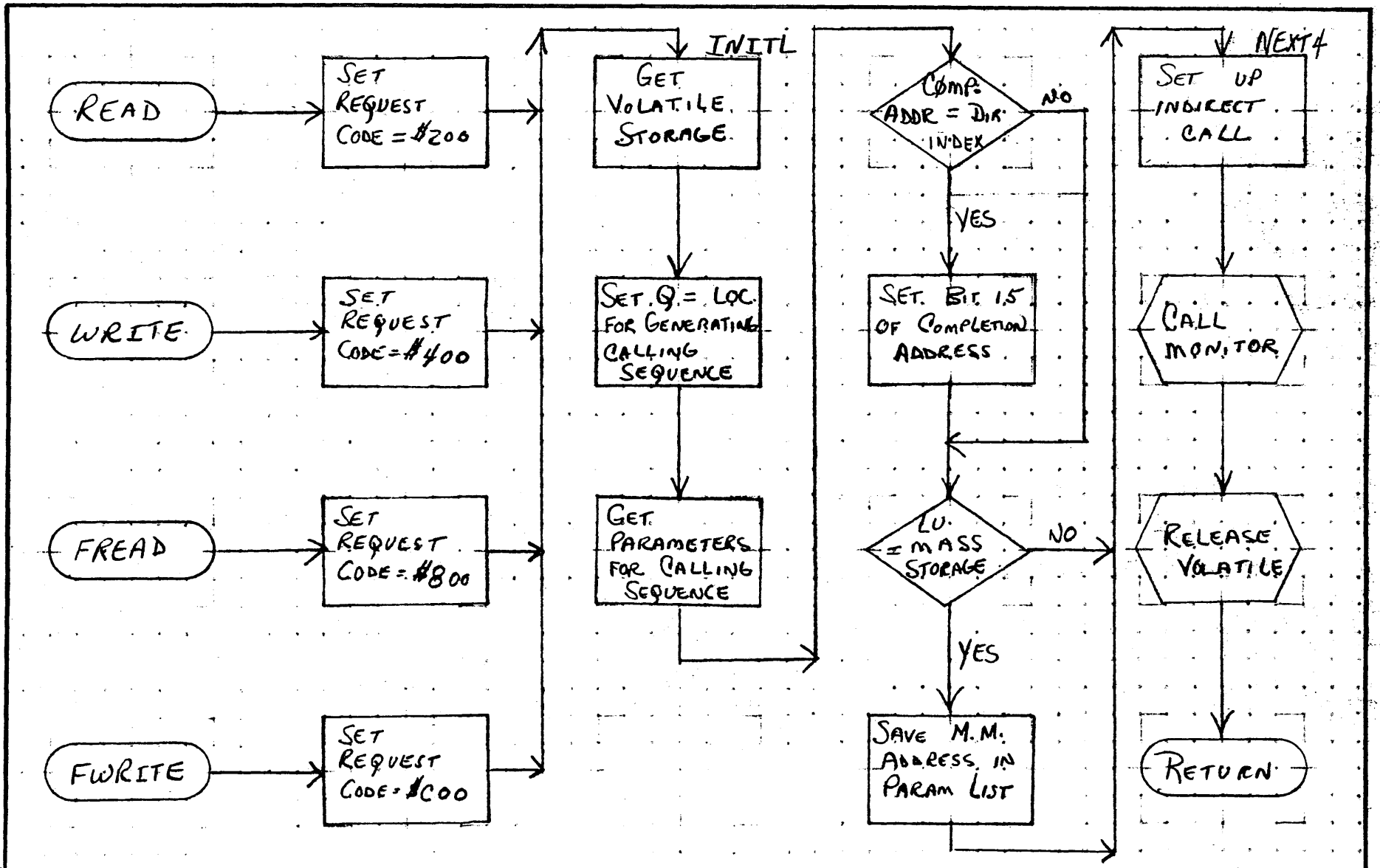
The Input and Output Instruction Routines allow the FORTRAN user to directly communicate with I/O devices tied to the 1700 computer via the 1705 A, Q channel. Upon entry, the address of the three-word table is generated, A and Q are loaded, and the appropriate instruction {input or output} is executed. The contents of the A register are then stored back in the table. If a reject occurred in the I/O instruction, the flag word is set {word 3} and control is returned to the user.

Obtaining the Passed Parameter:

The function LINK simply transfers the contents of the Q register, which the Monitor has loaded with the passed parameter, to the A register and returns control to the caller.

Obtaining the Core Clock:

The function ICLOCK simply loads the core clock, location $E8_{16}$ into the A register and returns control to the caller.



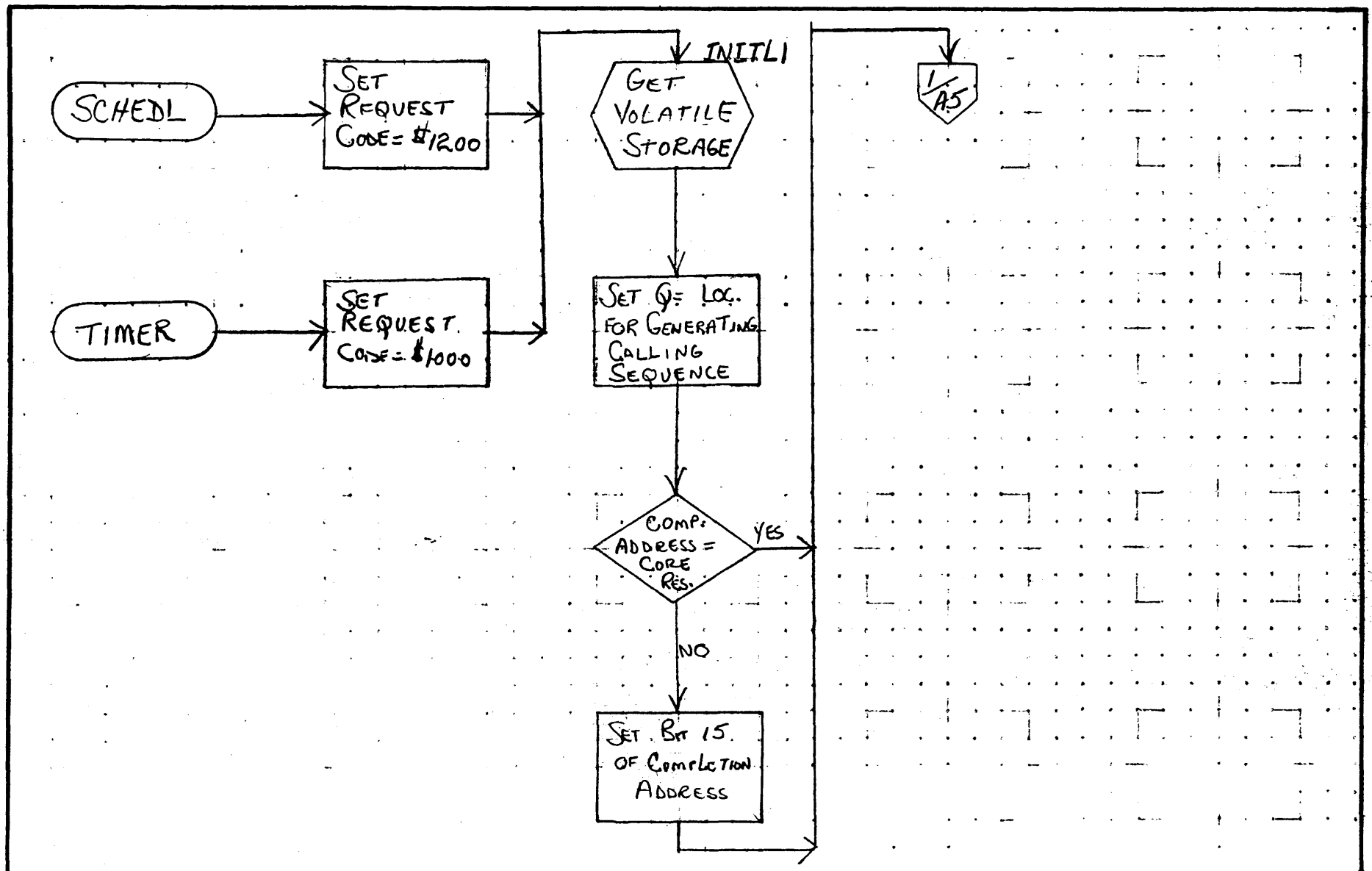
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE MONITOR INTERFACE		PROJECT MGR.			
FLOWCHART ☐	FORTRA	PAGE 1 OF 3	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

64.11

AA1385 (FORMERLY CA127-1)

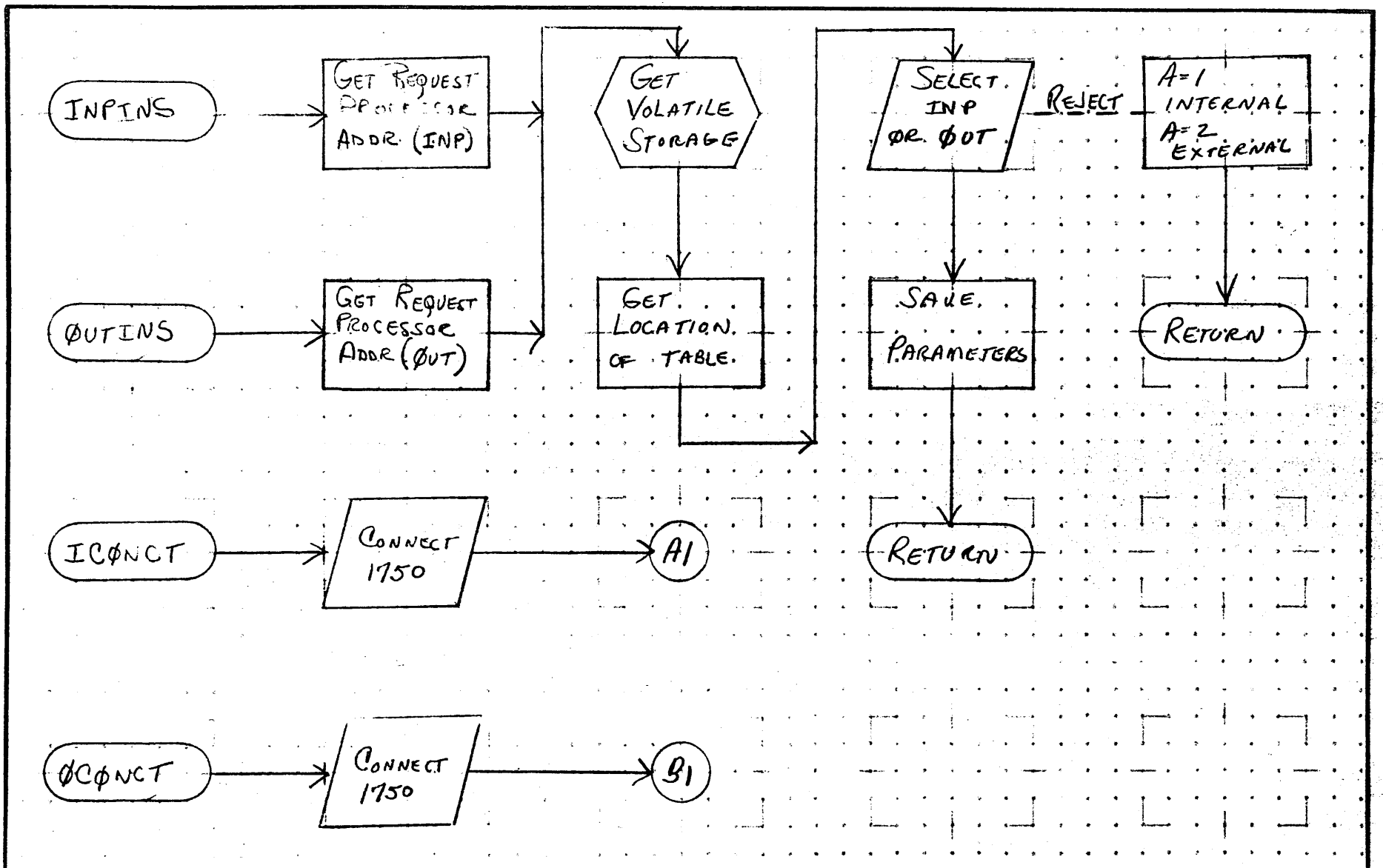
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MONITOR INTERFACE	PROJECT MGR.			
	FORTRAN PAGE 2 OF 3	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

64.12



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE MONITOR INTERFACE FORTRA	PAGE 3 OF 3	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971

64.13

65.0 Fortran Read/Write Statement Processor (Q8QIO)

65.1 Function

This program serves as an interface between the FORTRAN READ/WRITE statement, ENCODE/DECODE, and the 1700 Monitor Read/Write Request Processor. It will allow the FORTRAN programmer to use READ/WRITE statements as defined by FORTRAN. The resulting I/O processing will be through the reentrant ENCODE/DECODE package. The only changes resulting from this implementation are: 1) The user must supply a buffer in which the format processing will take place, 2) 18 temporary locations, which immediately precede the buffer, and will contain the calling sequences to the monitor for read/write processing and information for re-entrancy, 3) on input only one record/read statement may be executed, i.e., the FORMAT statement may specify only 80 columns of data for card input, 4) on output the record/write statement may be as long as the space in the buffer allows with the following limitations: If the programmer has not specified a new line (/) after 150 characters have been packed into the buffer, a carriage return is automatically inserted in the message, and will continue to be inserted every 150 characters until the FORMAT processing is complete.

65.2 Entry Points

ARGUO - Entry containing FWA of user's buffer (reentrant version only) which is saved by the scheduler.
Q8QINI - Entry to initializer format processing.
Q8QX - Entry to pick up the parameters in list of variables to be converted.
Q8QEND - Entry to signal the termination of format processing.
Q8QGET - Entry for ENCODE/DECODE to pass the user's parameter addresses for format processing.
SETBFR - Entry for user to specify the buffer address with temporary locations for generating the READ/WRITE calling sequence and the length of the buffer.
IOERR - Entry for obtaining ERROR flag immediately after executing a FORTRAN READ/WRITE statement.

65.3 External Symbols

FORMTR Entry point for Encode/Decode

Binary

65.4 Entry & Exit Interfaces

Before a READ/WRITE statement can be properly executed a CALL SETBFR (BUFFER,LENGTH) must be made at least once. Also, after any Dispatcher call the SETBFR call must be repeated, otherwise the buffer location has been destroyed. The user's Q, I registers will be saved and restored. After any ENCODE/DECODE call, and before any FORTRAN READ/WRITE statement, a call to SETBFR must be made. The scratch area contained in the user's buffer is as follows:

WORD 1. Last Word Address (LWA) of Buffer
2. Request Code for READ/WRITE
3. Completion Address
4. Thread
5. Logical Unit
6. Message Length
7. First Word Address (FWA) of Message
8. Unused
9. Unused
10. Q-Register of User
11. Return Address to User's Program
12. I-Register of User

WORD 13. READ/WRITE Flag (ICODE)
14. LIST Address
15. Total Number of Variables in LIST (MV)
16. ENCODE/DECODE Flag (DEFLAG)
17. FORTRAN FORMAT Flag
18. Error Flag
19. Starting Location of User's I/O Buffer

The format for the FORTRAN FORMAT FLAG is:

Bit 15 not used
14 1 = formatted
0 = unformatted
13 1 = actual LU
0 = address of LU
12 1 = format statement address
0 = address of format statement address
11 1 = list
0 = no list
10 1 = read
0 = not read
9 1 = write
0 = not write
8-0 not used

The general calling sequence generated by the FORTRAN compiler for a READ/WRITE statement is:

```
RTJ     Q8QINI
1. Flag word (see above).
2. I/O Request number (user's statement number).
3. LU or address of LU.
4. Format statement address, or address of format statement
   address.

RTJ     Q8QX
1. Address of LIST element.

RTJ     Q8QEND
```

65.5 General Program Information

65.5.1 Calling Sequence

65.5.1.1 Assign Buffer Location

CALL SETBFR (BUFFER,LENGTH) where the first 18 words of the buffer will contain the calling sequence for the I/O request and information for reentrancy. The remainder will contain the input/output message. LENGTH is the total length of the BUFFER which includes the 18 words needed by Q8QIO.

For input operation:

READ (LU,10) LIST
10 FORMAT (----) Where LU = the logical unit and LIST contains the elements to be input.

For output operation:

WRITE (LU,10) LIST
10 FORMAT (----) Where LU = the logical unit of LIST contains the elements for output.

For error flag:

IFLAG = IOERR(0) Where IFLAG will contain a -1 if errors occurred.

65.5.2 There are seven entry points in Q8QIO with each having a unique function. They are as follows:

1. Q8QINI picks up the location of the format, logical unit, and flag for READ/WRITE processing and returns if data is contained in LIST (as indicated by the flag word). If only a format (NO LIST) is to be processed ENCODE/DECODE is called. The format is transferred into the buffer, a write request of the monitor is made, followed by a call to the dispatcher. On completion of I/O, control is returned to the user's program.

2. Q8QX passes the current parameter address within the LIST to ENCODE/DECODE for formatting. For the first request the input record will be obtained for a READ statement before calling ENCODE/DECODE, or the record for a WRITE statement will be output when format processing is complete.

3. Q8QEND simply passes the last parameter address within LIST to ENCODE/DECODE for processing and signals the end of formatting.

4. Q8QGET is called by ENCODE/DECODE to signal that a new parameter is needed for continued format processing. If ENCODE/DECODE was called directly by the user Q8QGET updates the LIST address and returns to continue format processing.

5. SETBFR is an entry which is called by the user once to pass the buffer address and buffer length to the reentrant I/O processor. This address and length are retained by the I/O processor throughout execution of the current program unless the user calls the dispatcher or ENCODE/DECODE directly. If this occurs, the user must call SETBFR again before any I/O can be done.

6. IOERR is an entry for a FORTRAN function call to return an error flag (-1) if any errors occurred while processing the previous READ/WRITE statement.

7. ARGUD is an entry in the reentrant version containing the FWA of the user's buffer. When a FORTRAN priority level change is made, this location is saved by the SCHEDULER/DISPATCHER in the monitor.

65.5.3 Core Memory

Approximately 246 locations (reentrant version) and ENCODE/DECODE (1766). Approximately 194 locations (non-reentrant version) and ENCODE/DECODE (1681).

65.6 General Design Peculiorities

65.6.1 Reentrancy

This program is reentrant using the volatile storage required by Encode/Decode.

65.6.2 Limiations and Restrictions

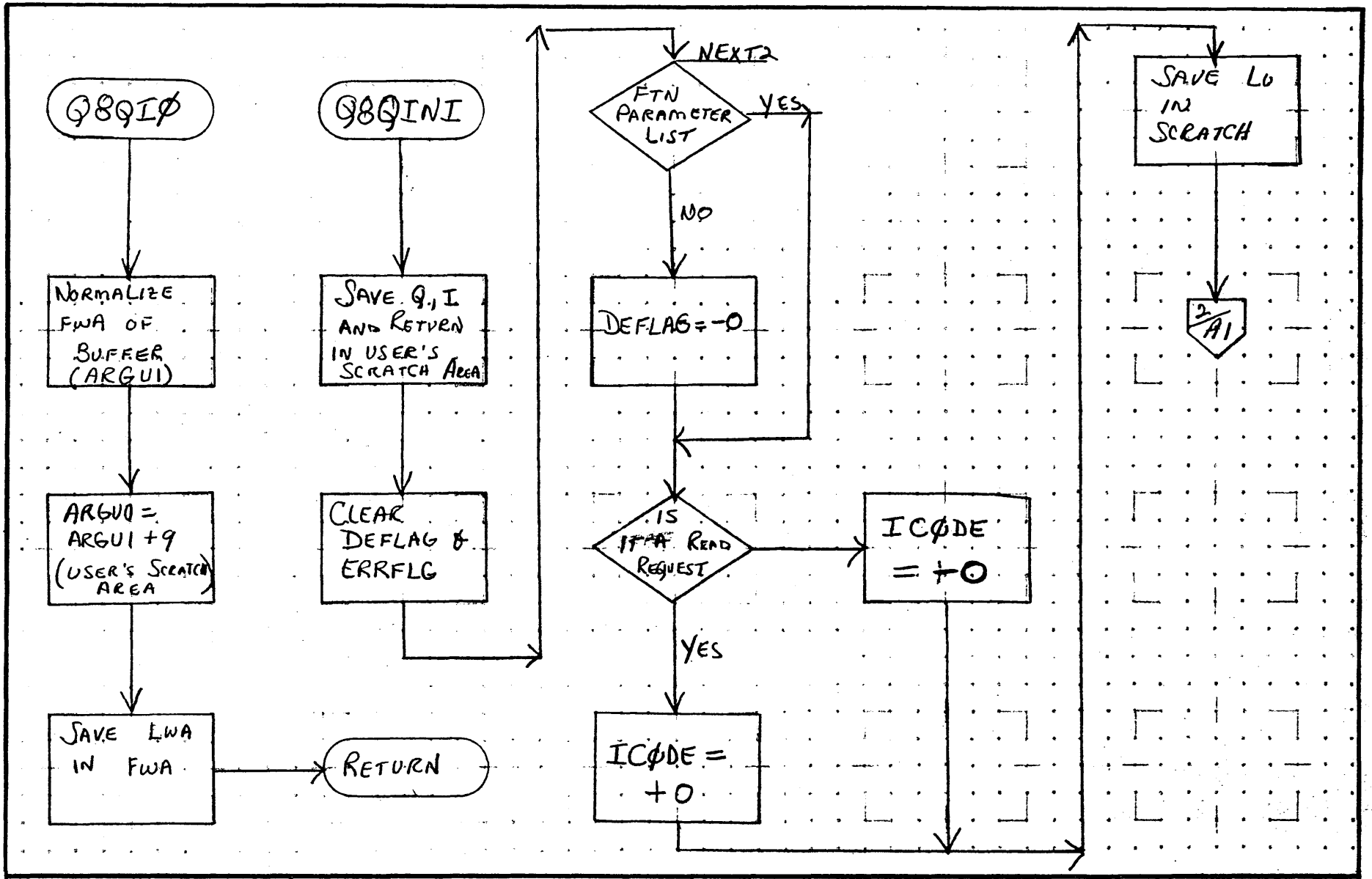
The FORTRAN programmer is no longer restricted in the use of implied DO statements in the LIST, but still is restricted to one level of repetition in the FORMAT statement (see ENCODE/DECODE) . A restriction is placed on input in that the FORMAT may designate only one input record (80 card columns)/record. On output the record

size will automatically be broken into 150 characters/line if the programmer has not designated a new line of output before the 150 characters. However, the buffer may be filled with as many records as the size of the buffer allows. When formatting is complete the buffer is output with the N-records contained within.

The format READ/WRITE are the only calling sequence implemented in Q8@I0; therefore, unformatted READ/WRITE must still be made through the FORTRAN/MONITOR RUN-TIME Package. Further, there is no implemenetation of mass memory READ/WRITE; these requests must be made through the RUN-TIME Package.

65.6.3 User Instructions

This program must be run in conjunction with the ENCODE/DECODE package

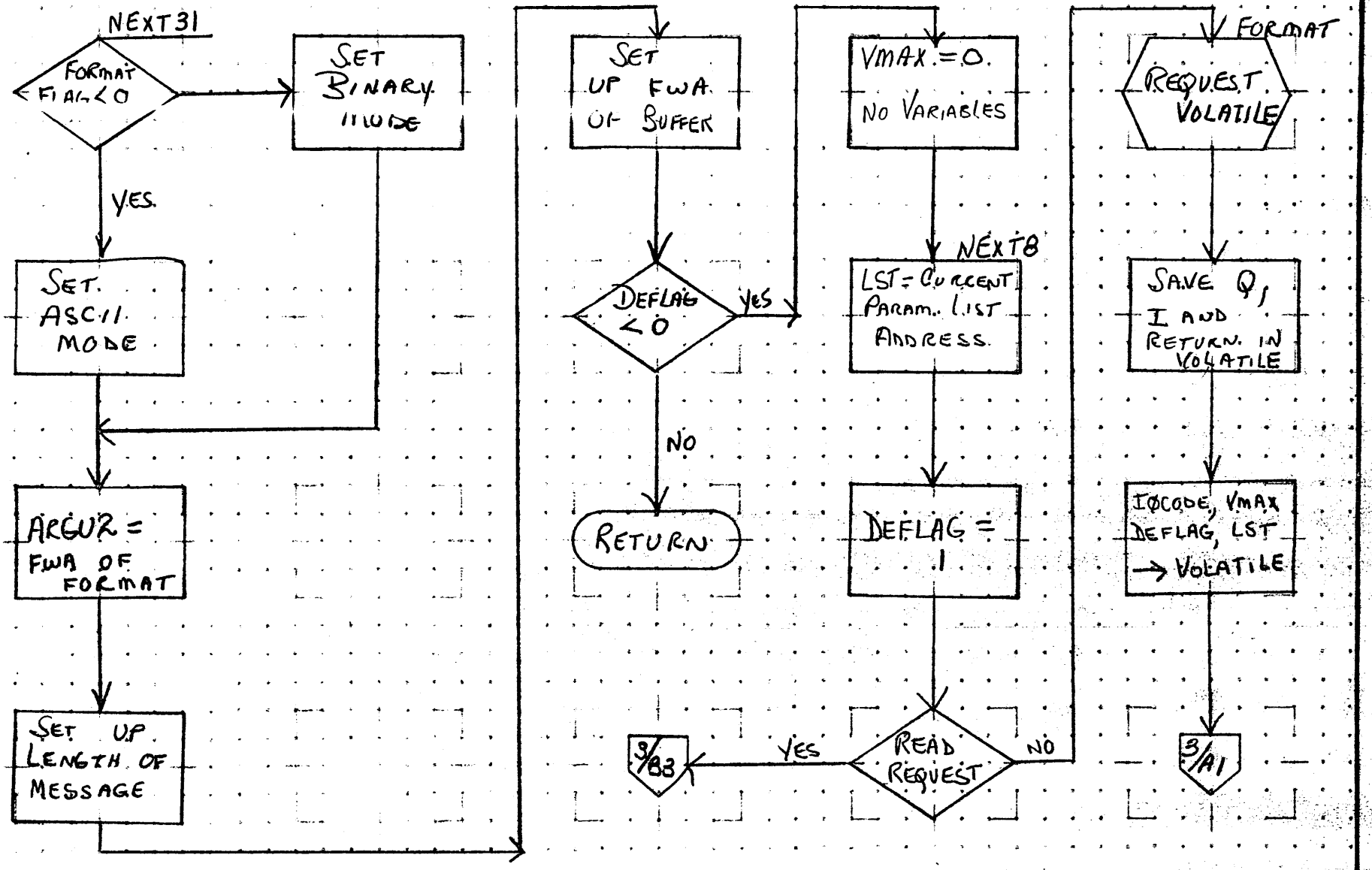


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE FTN READ/WRITE PROCESSOR		PROJECT MGR.			
FLOWCHART ☐	RE-ENTRANT	PAGE 1 OF 5	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

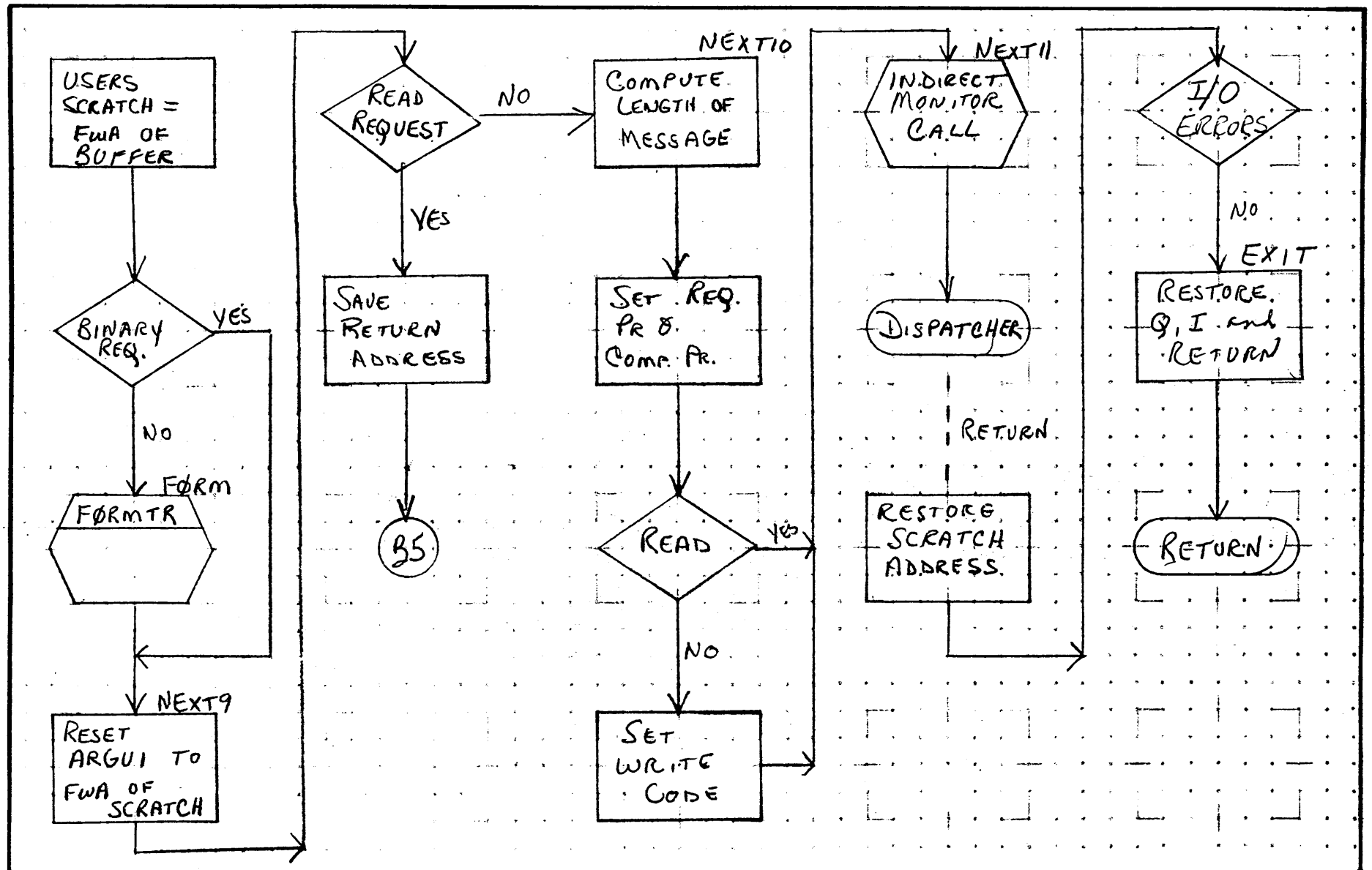


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE FTN R/W PROCESSOR		PROJECT MGR.		
	REENTRANT	PAGE 2 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



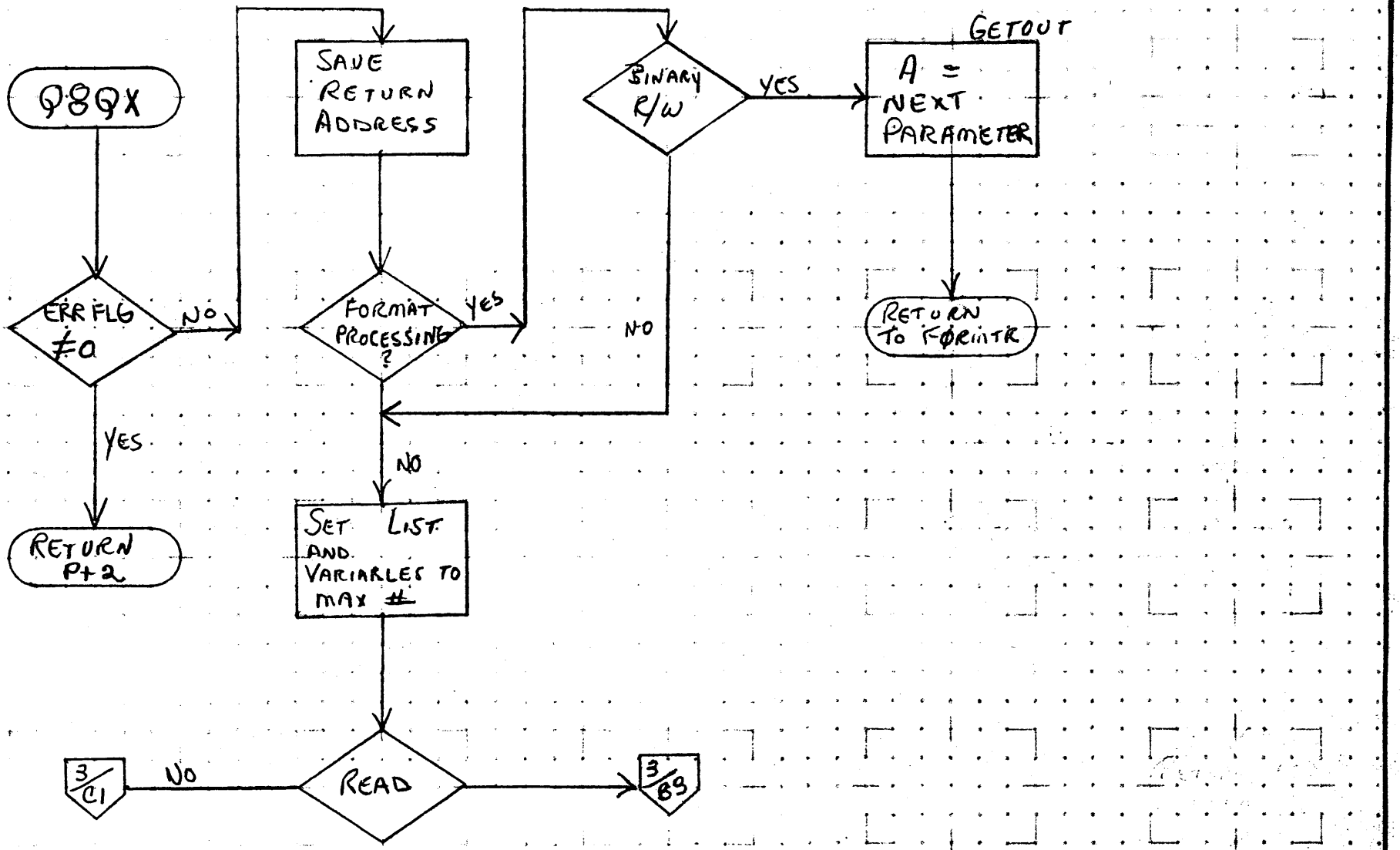
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE FTN R/W PROCESSOR		PROJECT MGR.		
FLOWCHART ☐	REENTRANT	PAGE 3 OF 5	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

4.5.9 /

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

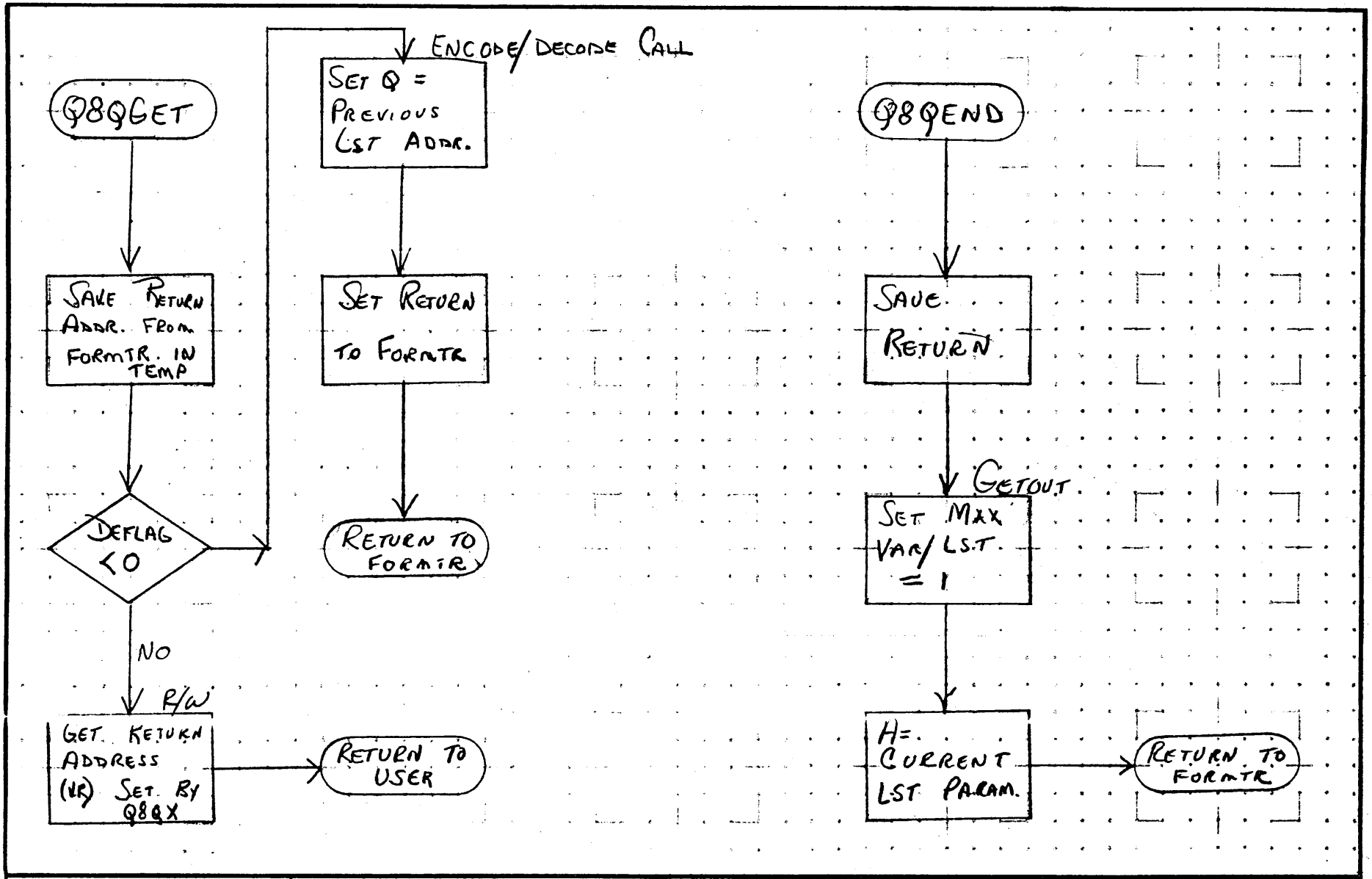


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE FTN R/W PROCESSOR		PROJECT MGR.		
	REENTRANT	PAGE 4 OF 5	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE FTN R/W PROCESSOR		PROJECT MGR.			
	REENTRANT	PAGE 5 OF 5	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

66.0 Fortran Encode/Decode

This document describes the Encode/Decode subroutines. Also subroutines which are a subset of Encode/Decode to be used by the FORTRAN programmer. This subset includes HEXASC, HEXDEC, ASCII, DECHEX, AFORM, RFORM, and FLOAT.

66.1 Function

These subroutines afford the FORTRAN programmer the ability to format a list of variables for input/output with a compiled format statement, or to format a single variable by calling the appropriate formatting subroutine The purpose of formatting one variable at a time (whenever possible) will save execution time needed for interpreting the format statement.

66.2 Entry Points

Entry Symbols

ENCODE An entry which converts a list of variables from a binary to external form according to a specified format for output.

DECODE An entry which converts a list of variables from an external to binary form according to a specified format for input.

HEXASC An entry which converts a hexadecimal integer into two words in ASCII format.

HEXDEC An entry which converts a hexadecimal integer into a three word decimal integer in ASCII format (maximum value 32767).

ASCII An entry which converts two words in ASCII format into a one word hexadecimal integer.

DECHEX An entry which converts three word decimal integer in ASCII format into a one word hexadecimal integer (Maximum value 32767).

AFORM — An entry which converts one word in ASCII format into two words each containing an ASCII character {left justified, blank filled}.

RFORM — An entry which converts one word in ASCII format into two words each containing an ASCII character {right justified, zero filled}.

FLOAT — An entry which converts a floating point variable into six words in ASCII code with the following format: ‡.XXXXXXE‡00.

66.3 Externals

FLOT — A FORTRAN floating point package used to handle floating point arithmetic.

66.4 Internal Symbols

IA — Contains the most significant bits of a floating point number. Also used as a temporary location.

IB — Contains the least significant bits of a floating point number. Also used as a temporary location.

ICH — Contains the current format conversion character.

IFIELD — Contains the current total field width specified in the format.

JFIELD — Contains the current number of decimal places to the right of the decimal point which is specified in the format.

FORMAT — A location containing the FWA of the user's FORMAT in the non-reentrant version of Encode/Decimal.

DEFLAG	A flag to indicate whether an Encode/Decode call or a Q8QIO call was made.
TEMP	Contains the return address to FORMTR [format processor] when a new parameter address is required for formatting.
NUMBR	Contains the current number of repeats on a format conversion character.
ITERAT	Contains the current number of repeats on a parenthesized expression.
IX	The ordinal to the current word in the format.
JX	The ordinal to the current character within the format.
IBX	The ordinal to the current word of the buffer.
JBX	The ordinal to the current character within the buffer.
IR	Scratch area for the conversion routines.
ICODE	Encode/Decode flag.
ISTART	Ordinal to the start of the parenthesized expression within the format.
JSTART	Ordinal to the starting character of the parenthesized expression within the format.
LIST	The address of the current variable to be converted.
MV	Total variables to be converted.
LPAREN	Total number of left paren encountered in the format.
MAXCH	Allowable number of characters packed into a buffer location.

ARGU1	Contains the FWA of the buffer.
ARGU2	Contains the FWA of the format.
ARGU3	Location for computing addresses.
ARGU4	Location for computing addresses.
BUFFER	A location containing the FWA of the user's buffer in the non-reentrant version of Encode/Decode.

66.5 General Program Information

66.5.1 Calling Sequence

66.5.1.1 Encode and Decode Call

CALL NAME {BUFFER, FORMAT, NO. OF VARIABLES, LIST}

BUFFER The starting address of a buffer for encoding into/decoding from. The buffer length must be at least one half the total number of characters specified in the format statement.

FORMAT The location of a format statement. This format may not contain more than one level of repetition for any given set of variables. The following character set will be interpreted as:

Ew.d for output formatting of a floating point quantity with an exponent. ▽w▽ specifies the total field width and ▽d▽ specifies the total number of significant digits {maximum of 6 significant digits}.

Fw.d for both input and output formatting of a floating point quantity. ▽w▽ specifies the total field width and ▽d▽ specifies the total number of significant digits to the right of the decimal point $\{10^{-5}$ to $10^{+5}-1\}$.

Iw.d for output formatting of a decimal integer. ▽w▽ specifies the total field width and ▽d▽ specifies the scalling factor $\{10^{-d}\}$. The magnitude of the integer: $8^5-1 \geq I$.

Iw	for both input and output formatting of a decimal integer. ▽w▽ specifies the total field width. The magnitude of the integer: {$2^{-15}-1$ to $2^{15}-1$}.
$w	for input/output of hexadecimal integers. ▽w▽ specifies the total field width. The magnitude of the hex integer $7FFF.
Aw	for input/output of alphanumeric data. The character will be left justified blank fill. ▽w▽ specifies the field width which must be ≤2.
Rw	for input/output of alphanumeric data. The character will be right justified zero fill. ▽w▽ specifies the field width which must be =1.
wH	for headings and labeling {ASCII code}.
/	beginning new record {single line space}.
1	as the first character in a format statement the interpretation will be ▽Top of Form▽.
0{zero}	as the first character in a format statement the interpretation will be double line space.
No. of variables	number of variables contained in the list for formatting.
List	a list of variables to be converted {cannot contain an implied DO statement}.

NOTE: If an odd number of characters has been packed into the buffer, a null {00} will be stored as the remaining character.

A special calling sequence is:

CALL ENCODE/DECODE{IBUF,IFORM,0,0}

The format must contain nH specifications only. For ENCODE ▽n▽ ASCII characters will be transferred from the format to the buffer. For DECODE ▽n▽ ASCII characters will be transferred from the buffer to the format.

66.5.1.2 HEXASC and HEXDEC Call

CALL NAME {VARIABLE,BUFFER}

VARIABLE the location of a hexadecimal integer to be converted to ASCII format.

BUFFER the location of a two word buffer to contain the converted integer in hexadecimal form, or the location of a three word buffer to contain the converted integer in decimal form. The format for the decimal integer will be a maximum of 5 digits preceded by a ± sign Right-justified blank fill.

66.5.1.3 ASCII and DECHEX Call

CALL NAME {BUFFER,VARIABLE}

BUFFER the location of a two word buffer which contains a hexadecimal integer in ASCII format or the location of a three word buffer which contains a signed decimal integer in ASCII format.

VARIABLE the location which will contain the integer converted to hexadecimal form.

66.5.1.4 AFORM and RFORM Call

CALL NAME {VARIABLE,BUFFER}

VARIABLE the location containing two ASCII characters.

BUFFER the location of a two word buffer which will contain an ASCII character per word.

66.5.1.5 FLOAT Call

CALL FLOAT {VARIABLE,BUFFER}

VARIABLE the location of a floating point variable.

BUFFER the location of a six word buffer.

66.6 Program Description

1. The entry to the non-reentrant ENCODE/DECODE package performs an initialization sequence in which the users calling parameters are absolutized and stored in a calling sequence for the subroutine FORMTR. FORMTR is the main program which interprets the format specifications and calls the appropriate conversion routine. The conversion routines do the computations from/to binary representation and pack/unpack the ASCII characters according to the format specifications. The subroutine HEXASC,HEXDEC,ASCII,DECHEX,AFORM,RFORM, and FLOAT call the appropriate conversion routines directly.

2. The entry to the reentrant ENCODE/DECODE package performs an initialization sequence in which the volatile allocation routine is called and the return address is saved.

   The procedure is then the same as in the non-reentrant package. The subroutine HEXASC,HEXDEC,ASCII,DECHEX, AFORM,RFORM, and FLOAT are also reentrant.

The ENCODE/DECODE package is to be used in conjunction with the FORTRAN/MONITOR RUN-TIME PACKAGE which output/input a formatted record.

66.7 Relocatability

The reentrant version is loader relocatable; the non-reentrant version is loader relocatable run-anywhere.

66.8 Reentrancy

There are two versions of the ENCODE/DECODE package:

1. A non-reentrant ENCODE/DECODE package written in FORTRAN and may be compiled as a loader relocatable package or a loader relocatable run-anywhere package.

2. A reentrant ENCODE/DECODE package written in assembly language which is loader relocatable {not run-anywhere} code.

66.9 Core Requirements

66.9.1 Non-Reentrant ENCODE/DECODE

Core Program	1644	
Mass Memory	none	
Non-Reentrant FORMATTING SUBROUTINES		
Core Program	219	
Mass Memory	none	
Tables {Data}	37	
	1900	Total

66.9.2 Reentrant ENCODE/DECODE

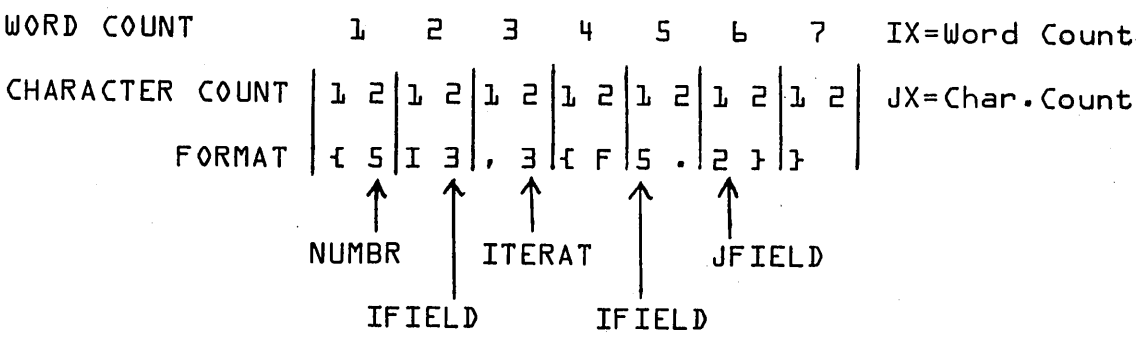
Core Program	1732	
Mass Memory	none	
Reentrant FORMATTING ROUTINES		
Core Program	212	
Mass Memory	none	
Volatile Storage	29	
	1973	Total

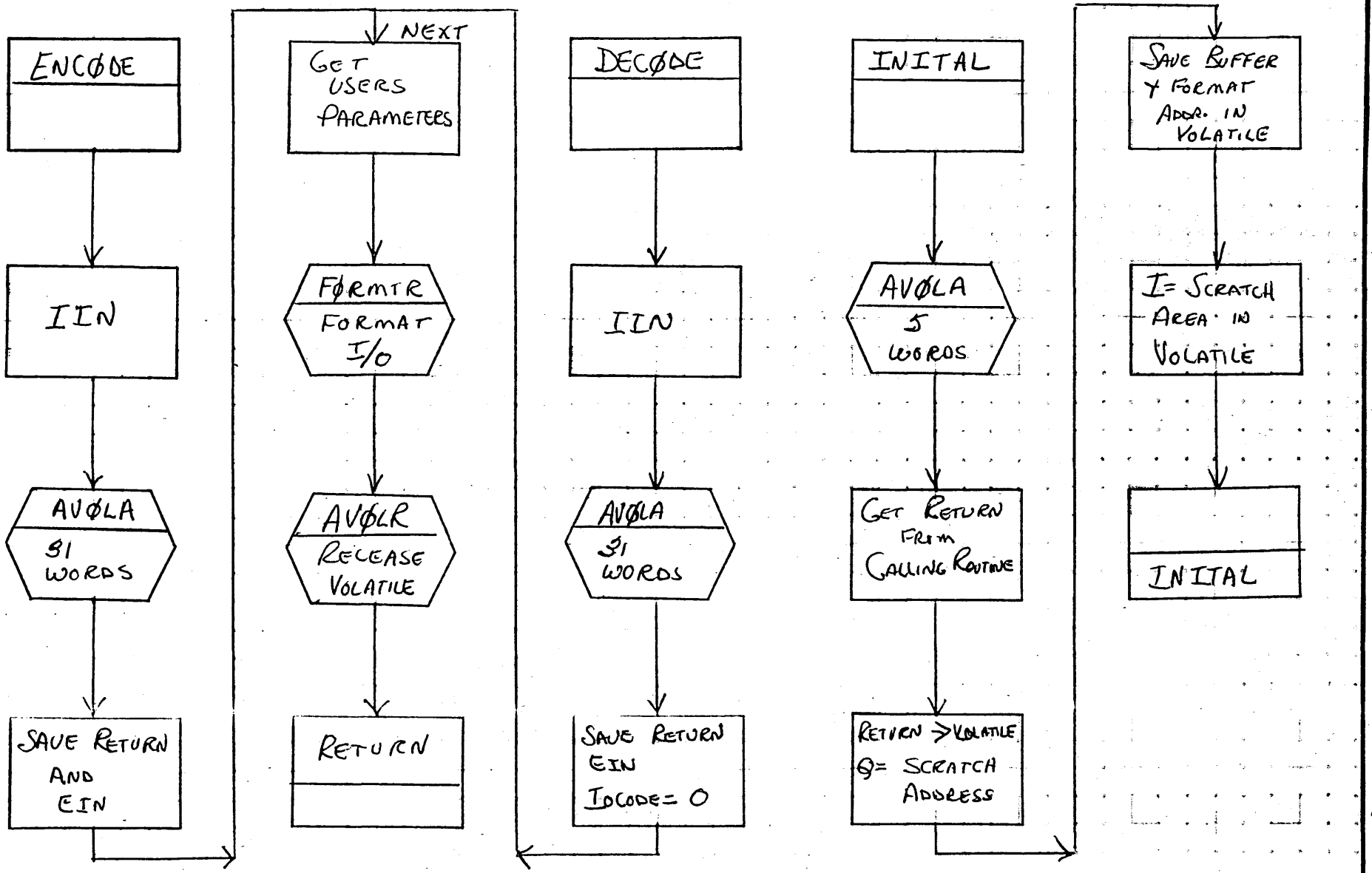
66.10 Reentrant ENCODE/DECODE

All subprograms of the reentrant ENCODE/DECODE must have the following instructions:

```
          ENTRY     NAME
            .
            .
            .
NAME        0         0
          INN         0
          RTJ       INITAL
            .
            .
            .
            .
          JMP       RSTORE
          END
```

FORMAT (513, 3(F5.2))





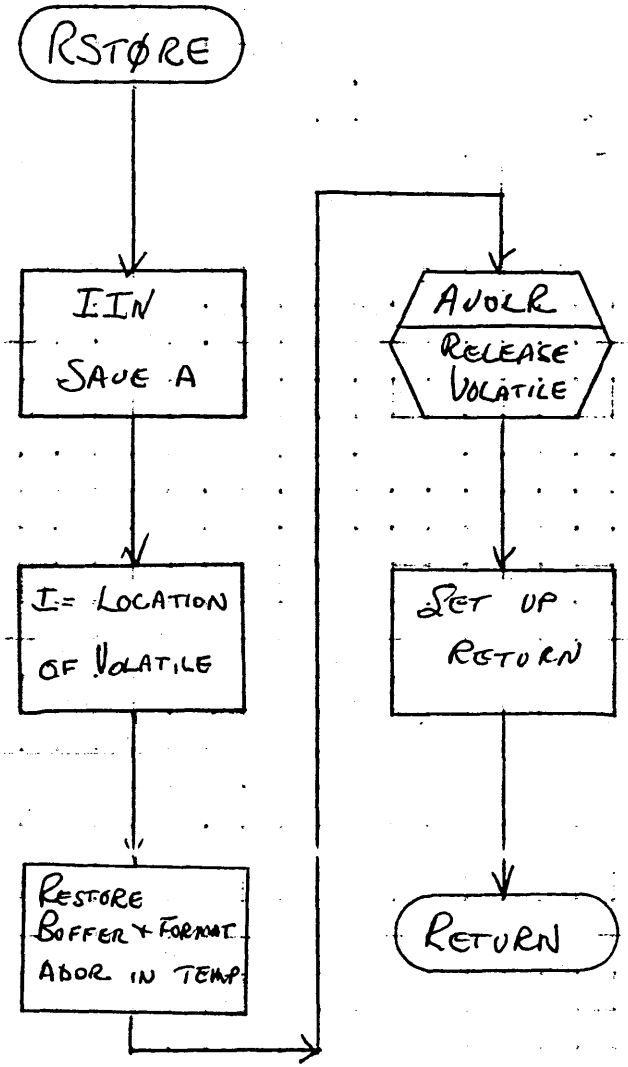
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	IØCODE	PAGE 1 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

64.11

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



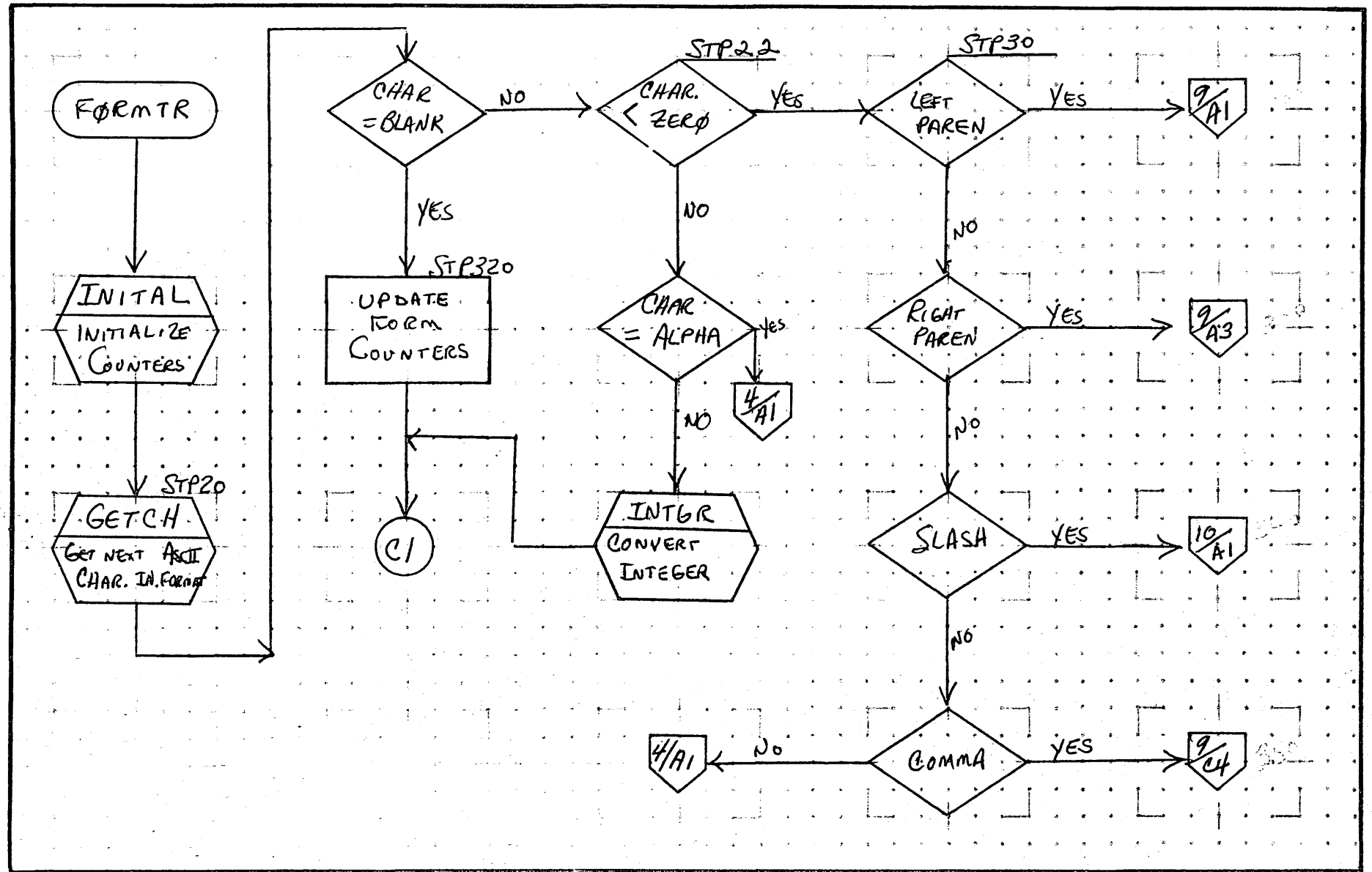
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	RSTØRE PAGE 2 OF 95	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

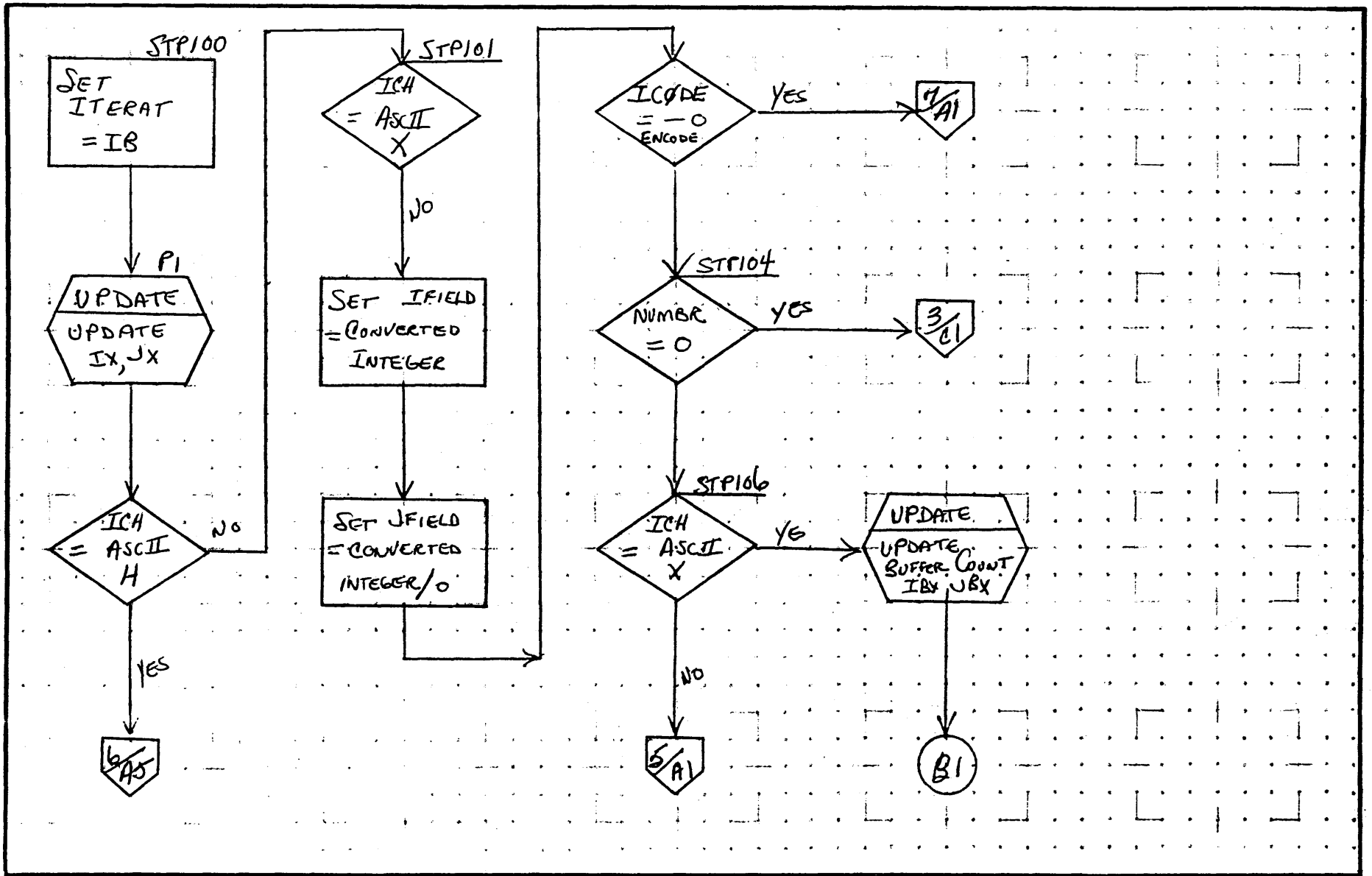
69.12

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	FORMTR	PAGE 3 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



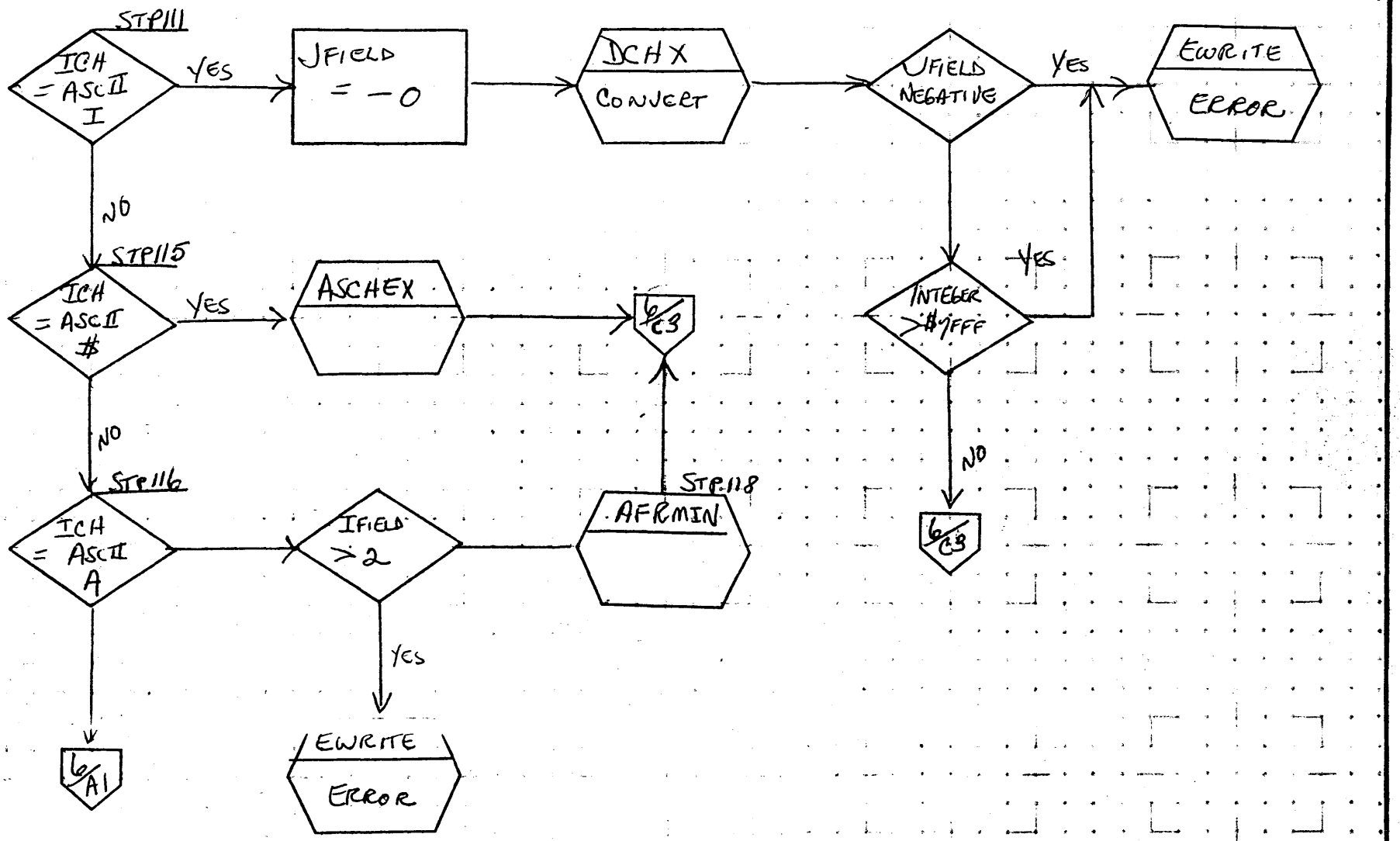
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	FORMTR PAGE 4 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

66.14



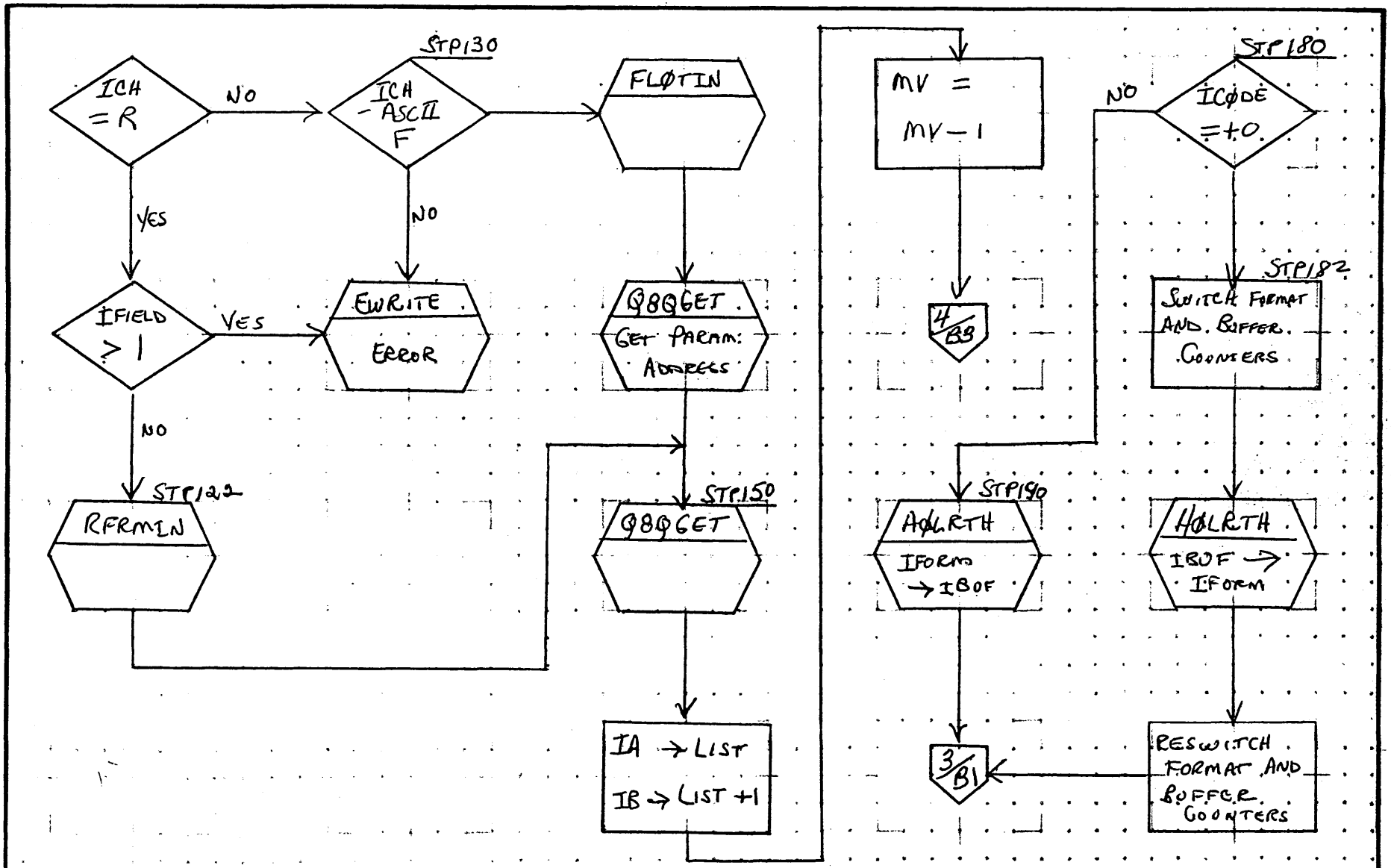
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	FORMTR PAGE 5 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

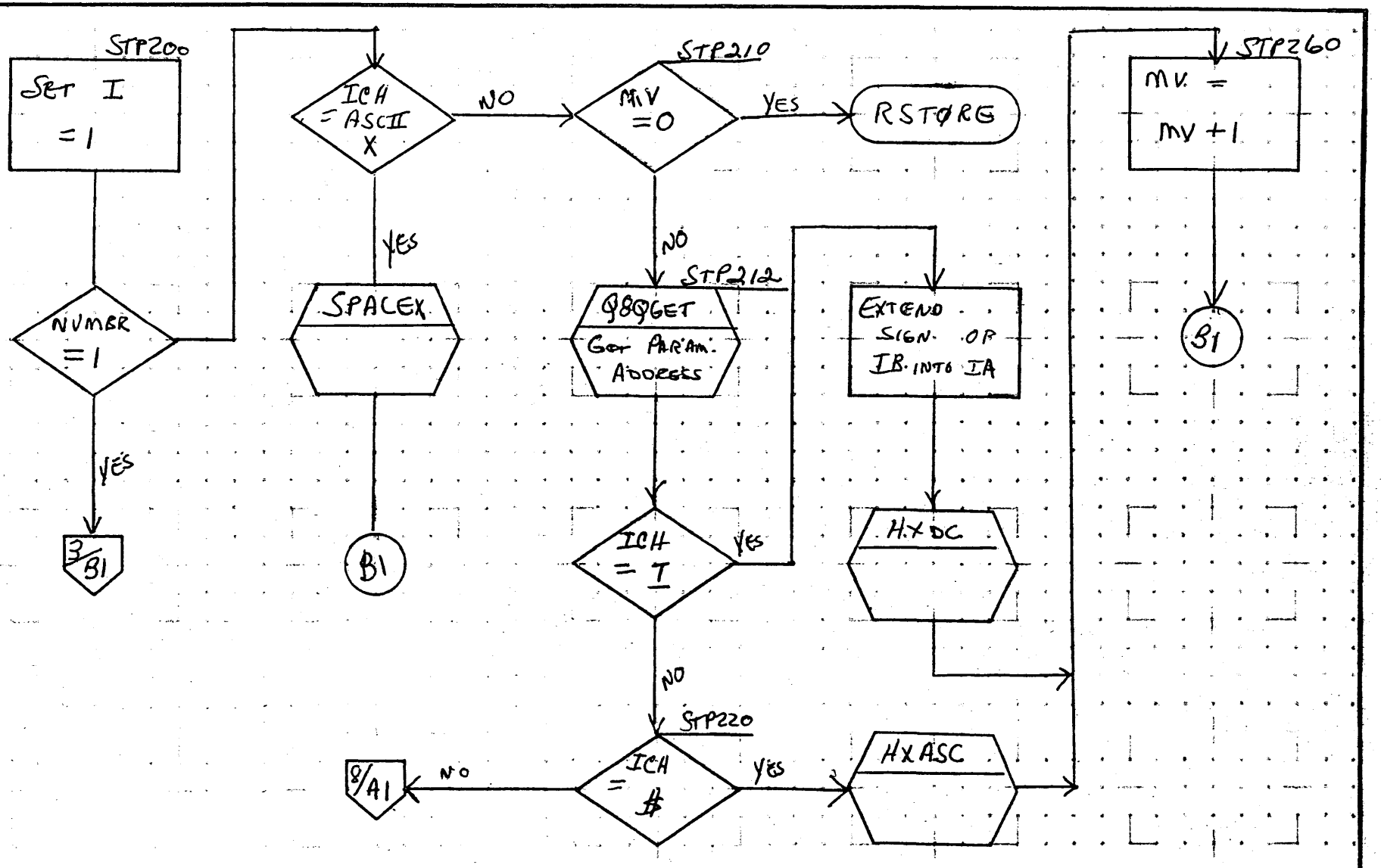
MAR 5 1971

66.15



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 17c C	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/ DECODE FORMTR	PAGE 6 OF 35	PROJECT MGR.		
	NUMBER	ISSUE DATE	PROJECT NAME		
	DRAWN BY	DATE	TASK NO.		
			TASK NAME		

MAR 5 1971



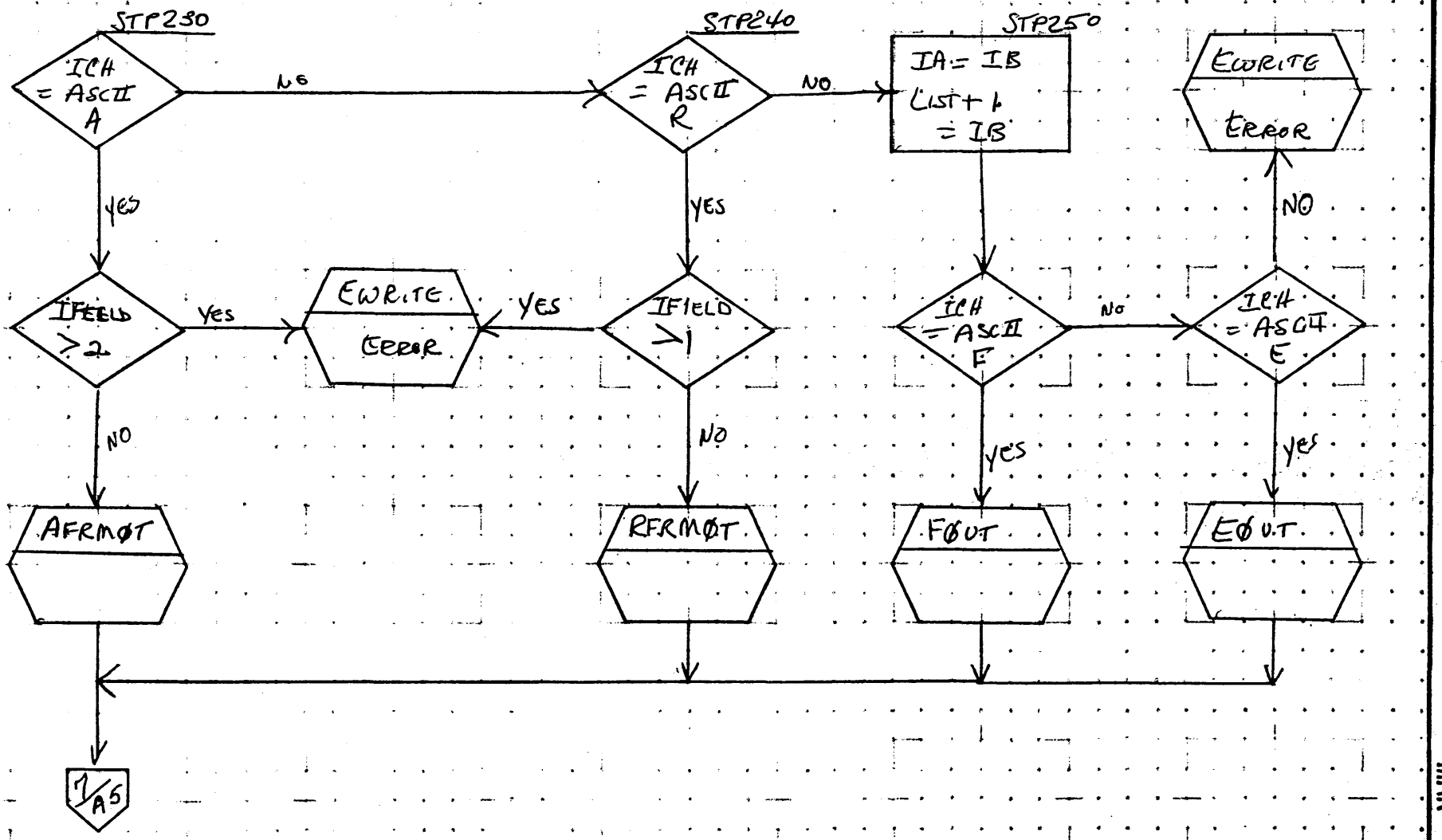
CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS Ims MACH. TYPE 1700	PROJECT NO.	REV / APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.	
	FORMTR PAGE 7 OF 35	PROJECT NAME	
	NUMBER ISSUE DATE	TASK NO.	
	DRAWN BY DATE	TASK NAME	

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

99.17

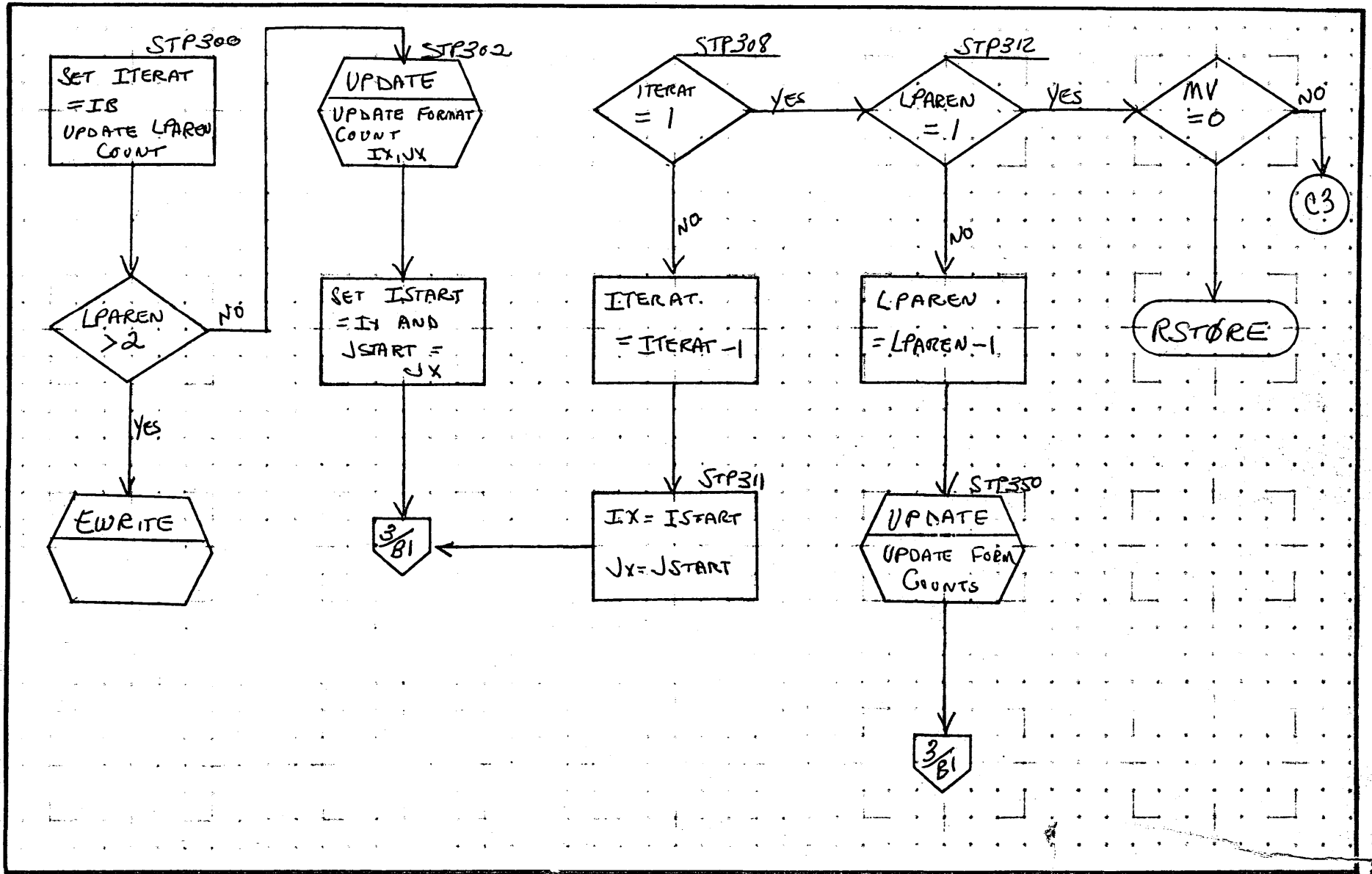


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	FORMTR PAGE 6 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

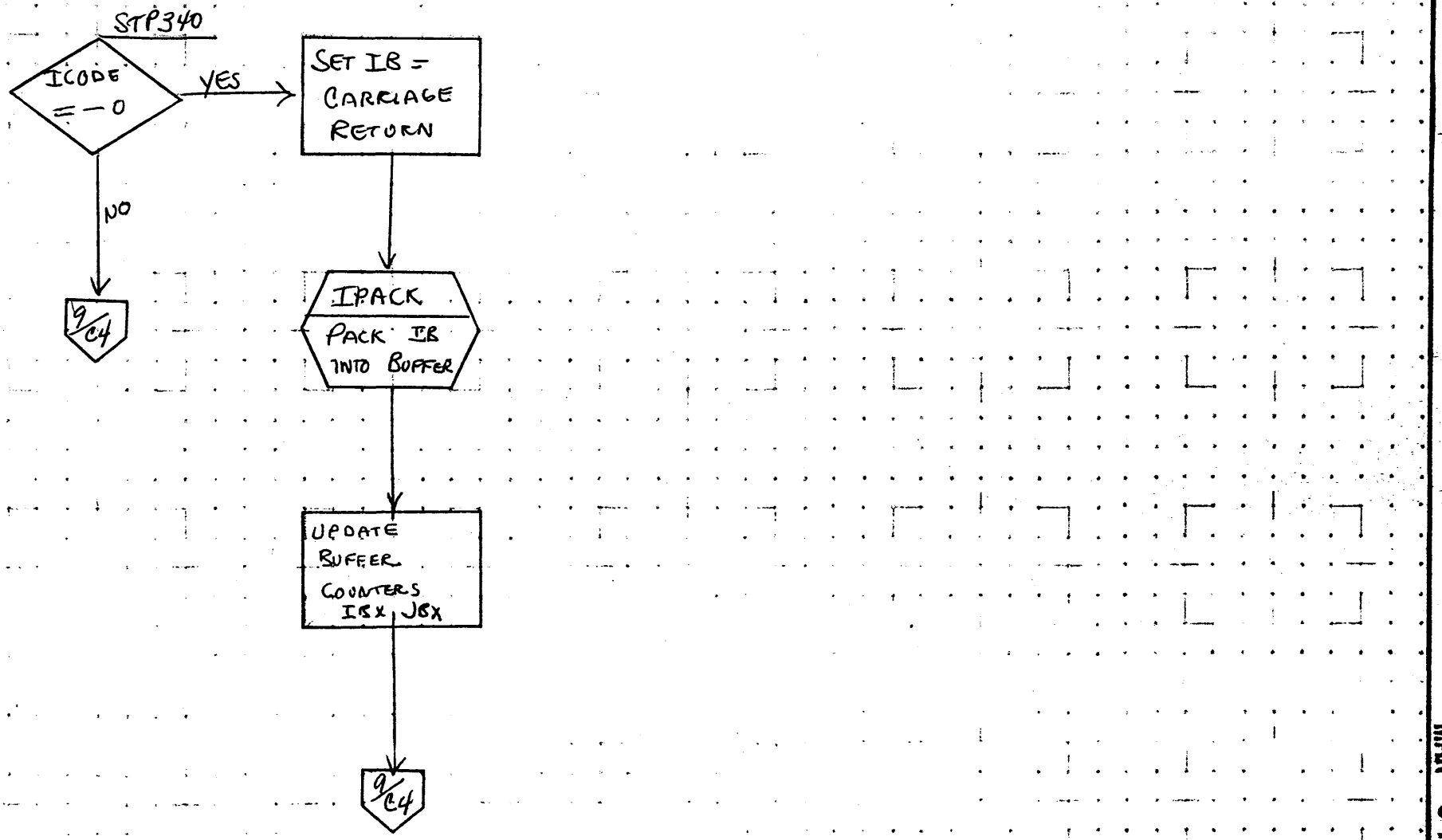
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	FORMTR PAGE 9 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971 66-19



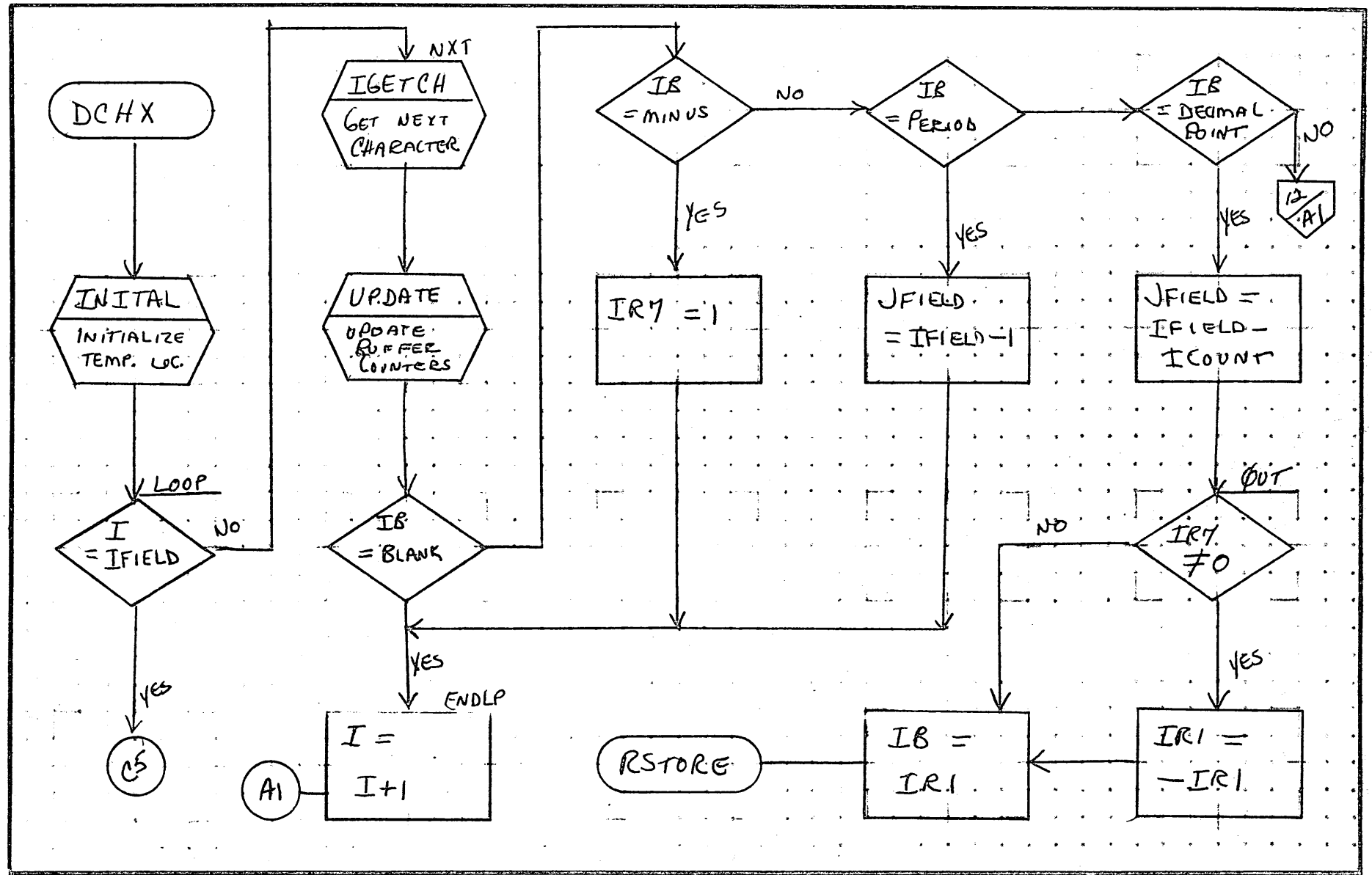
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
FLOWCHART ☐	FORMTR	PAGE 16 OF 85	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

66.20

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



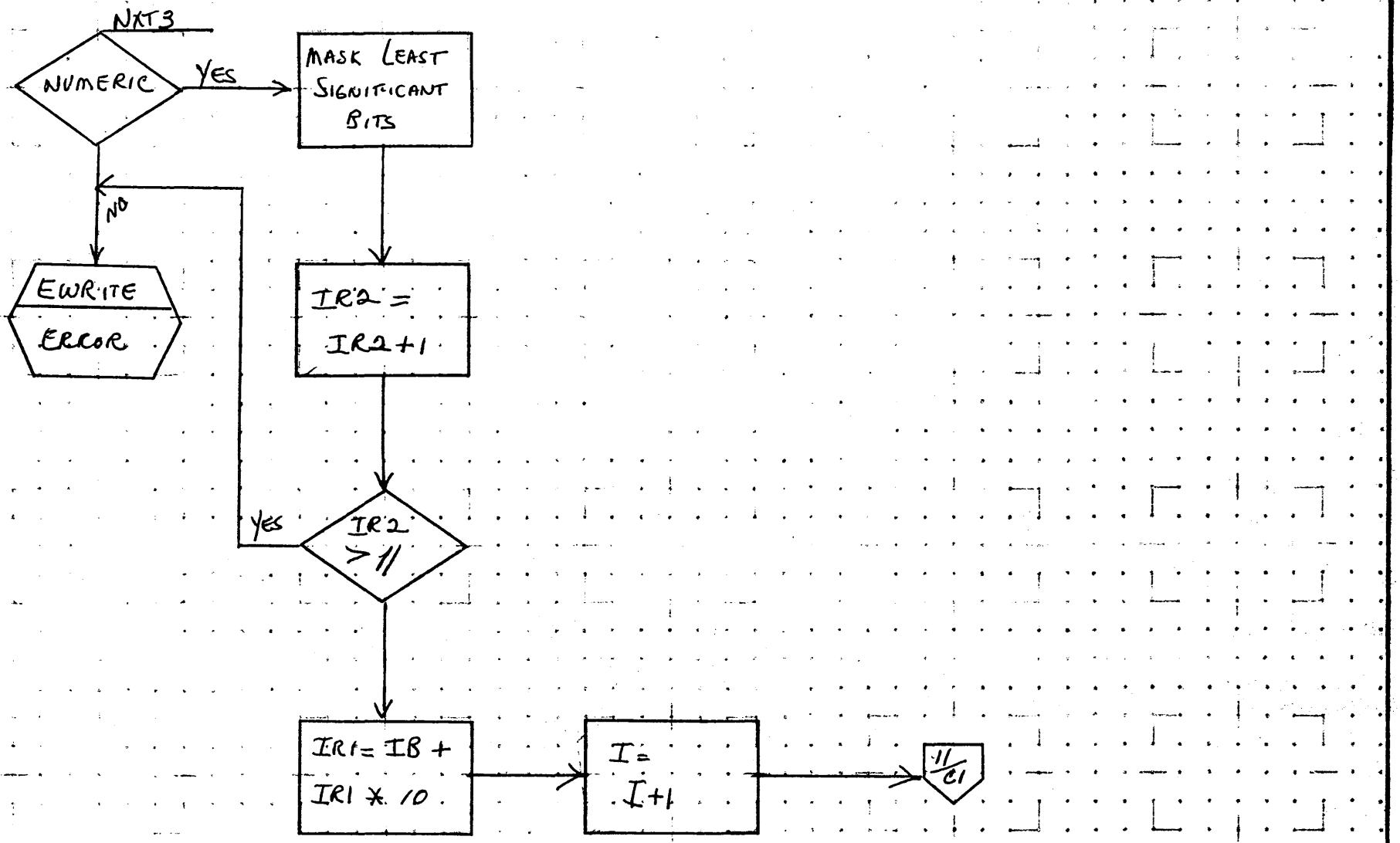
CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV. / APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.	
	DCHX PAGE 11 OF 35	PROJECT NAME	
	NUMBER ISSUE DATE	TASK NO.	
	DRAWN BY DATE	TASK NAME	

MAR 5 1971

69.21

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



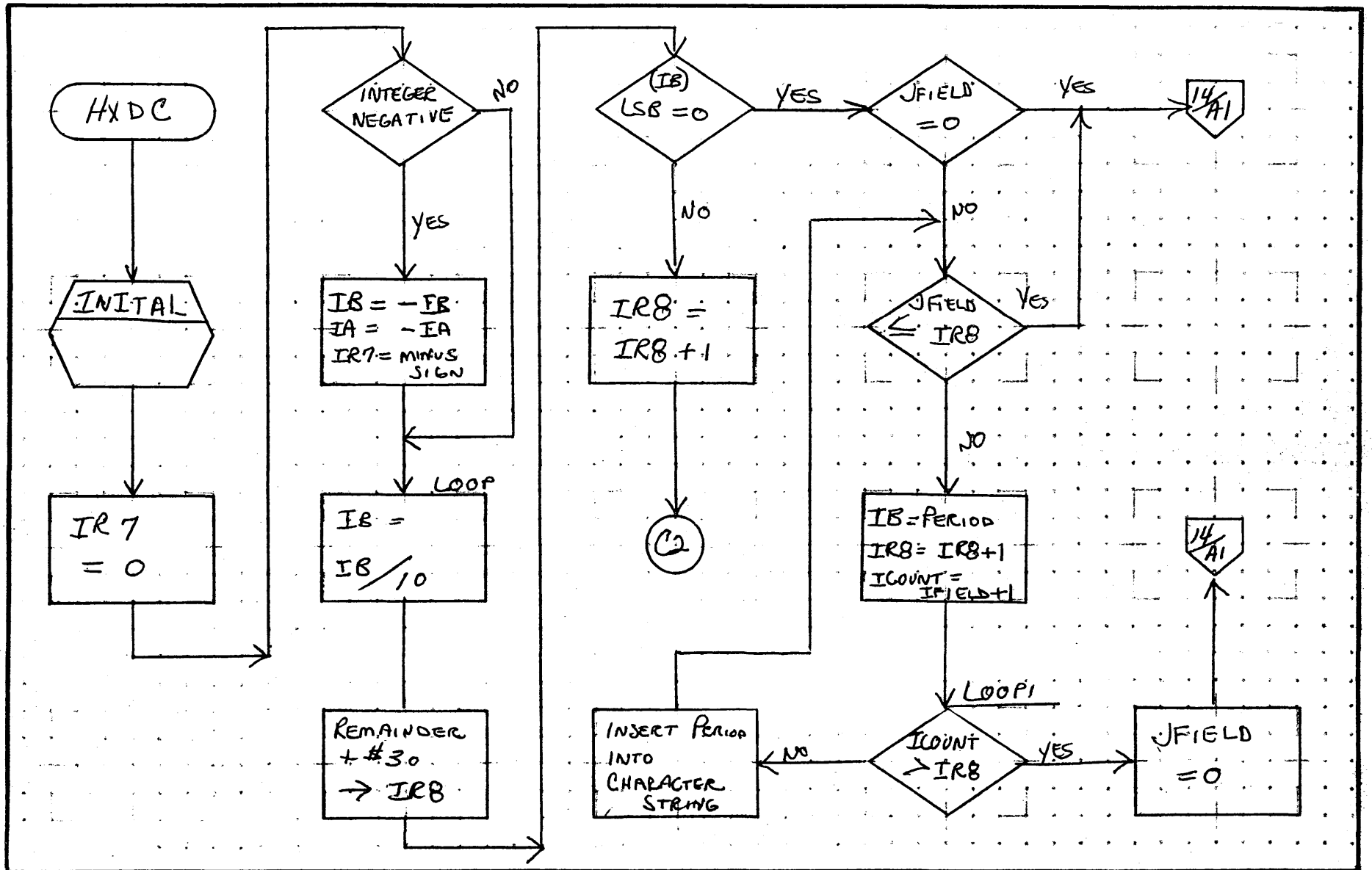
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: INS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE: ENCODE/DECODE		PROJECT MGR.		
	DCHX	PAGE 12 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

66.22



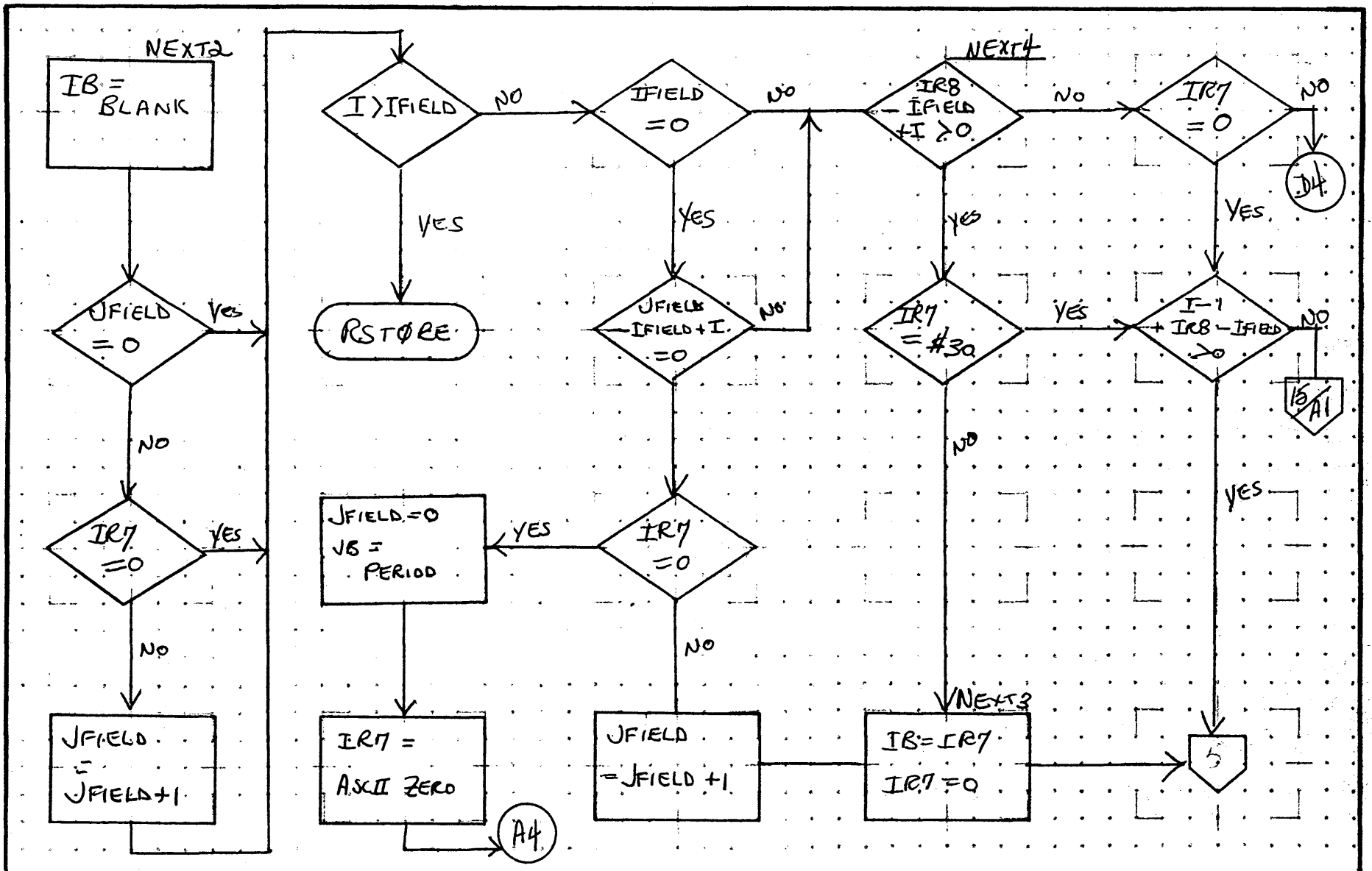
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	HXDC	PAGE 13 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

09:23



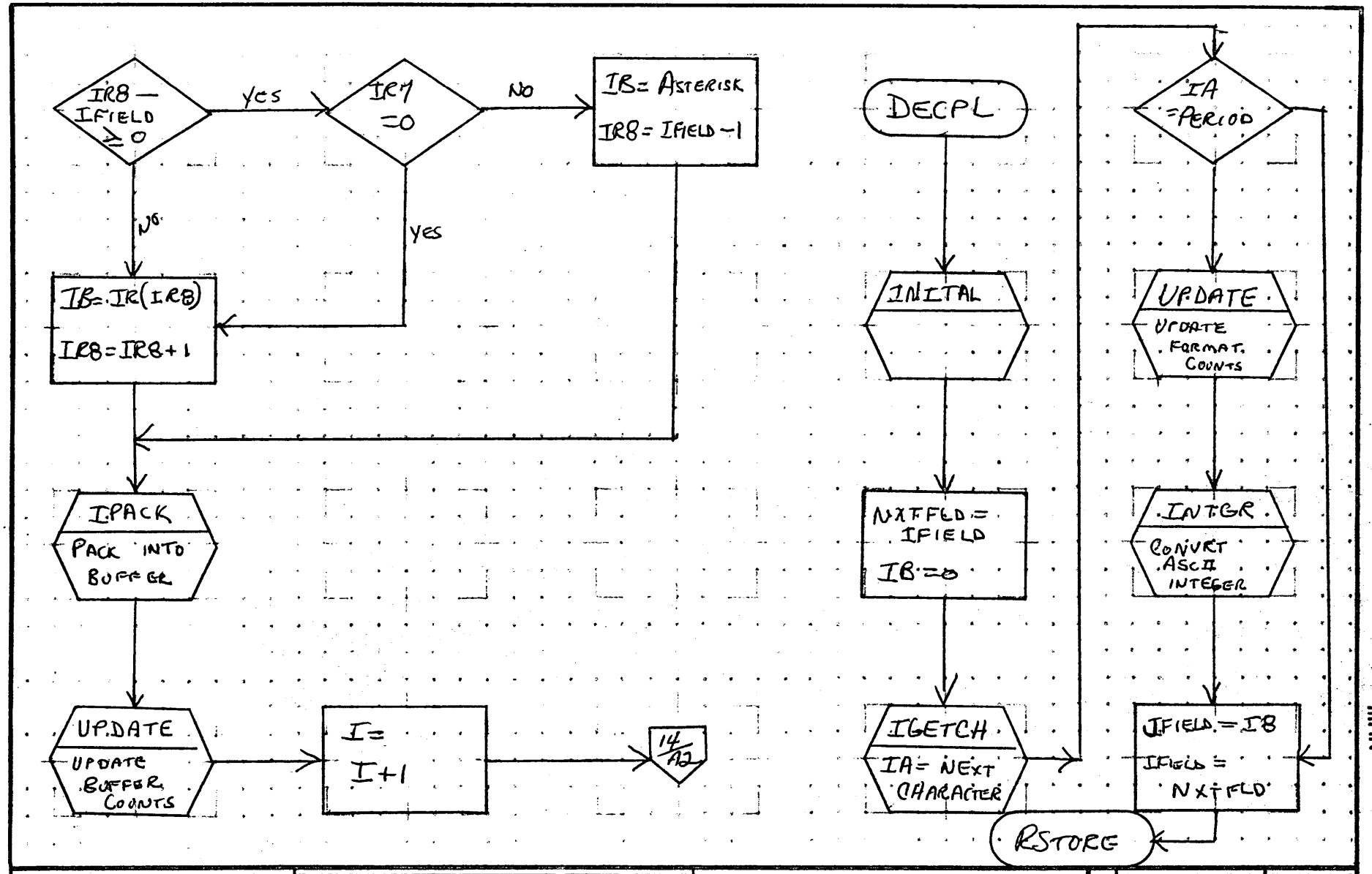
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	HXDC	PAGE 14 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

66-24



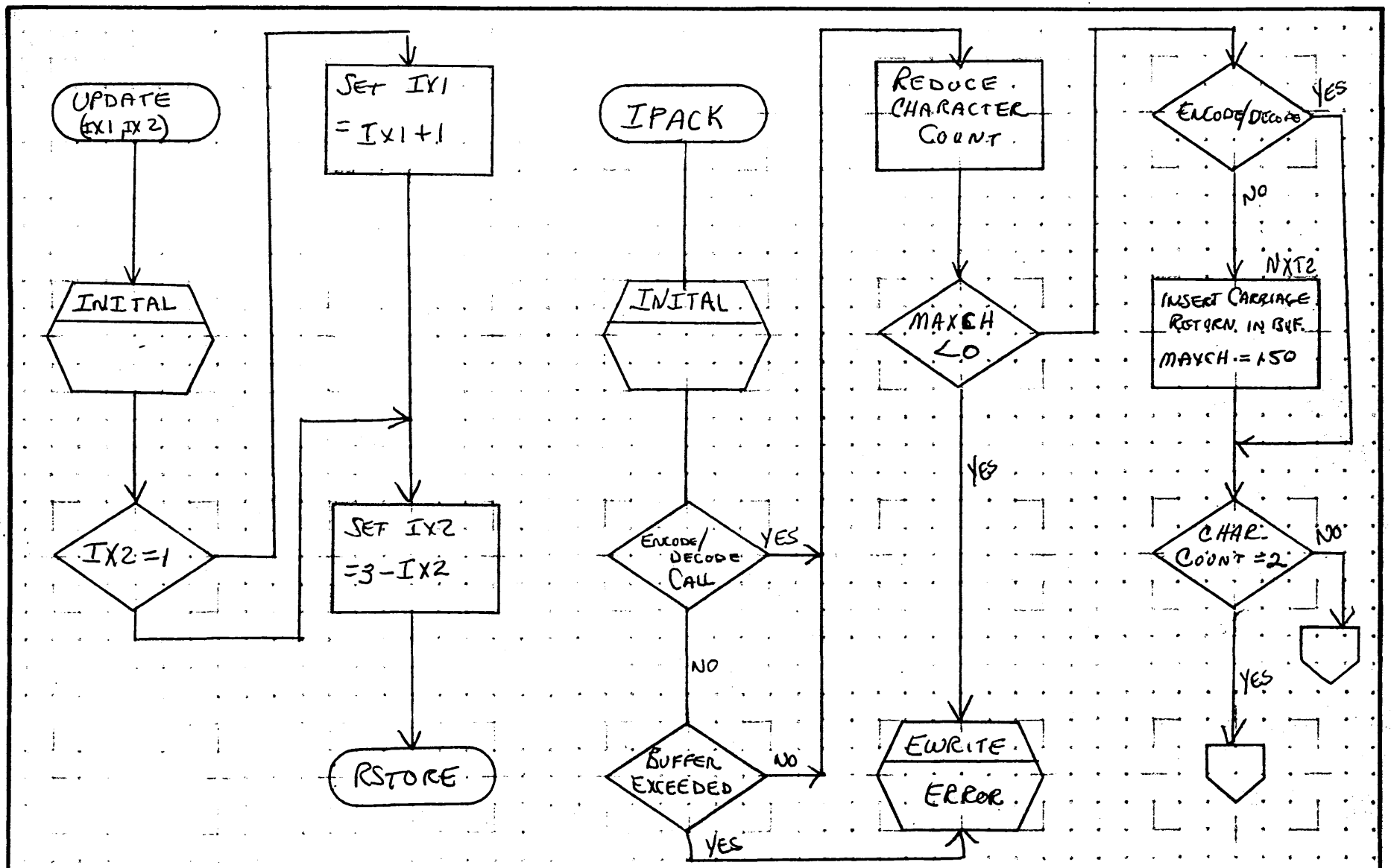
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.			
	HXDC - DCPL	PAGE 15 OF 35	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

66.25

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



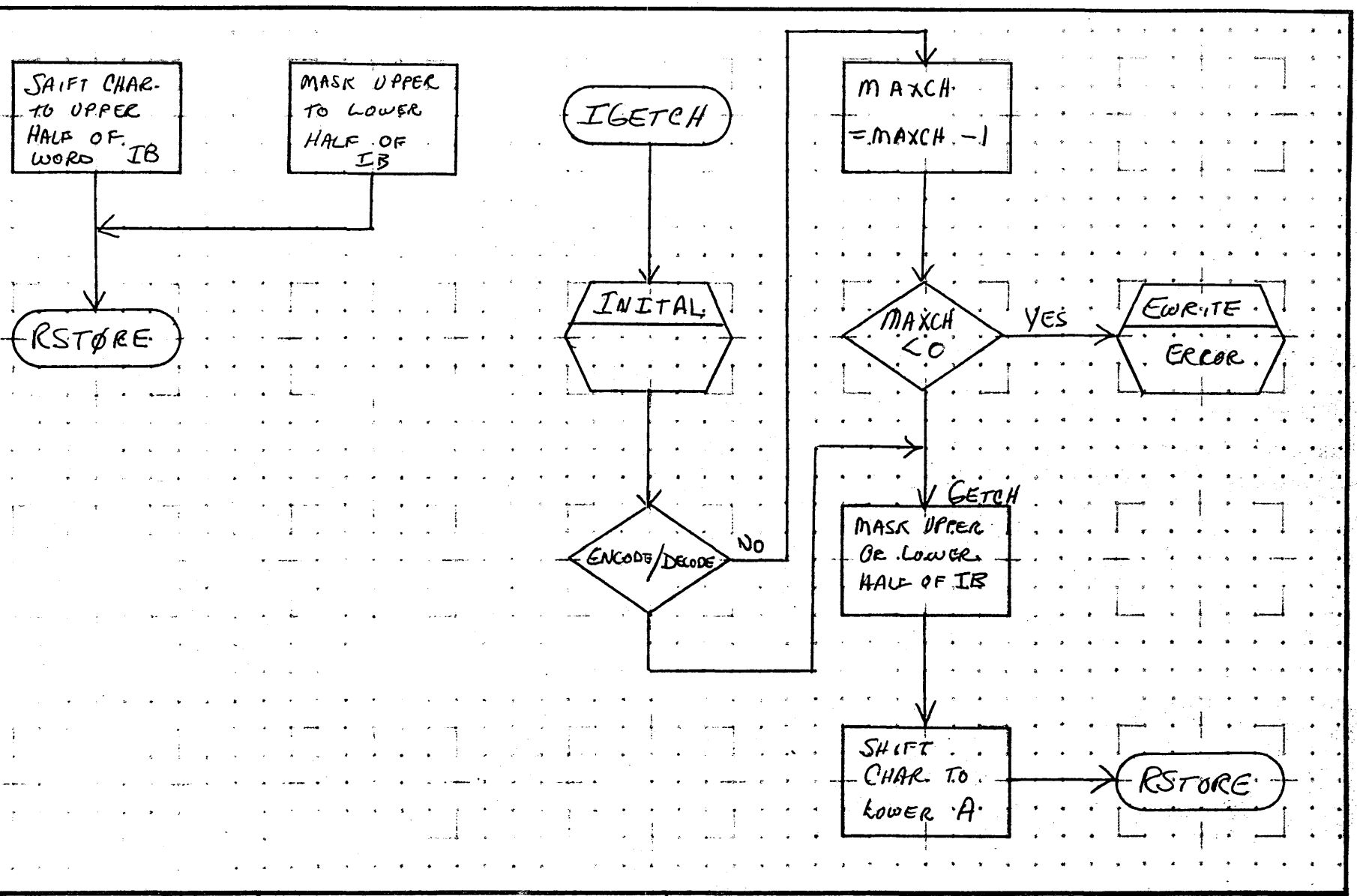
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/ DECODE		PROJECT MGR.		
	UPDATE – IPACK	PAGE 16 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

69.29

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



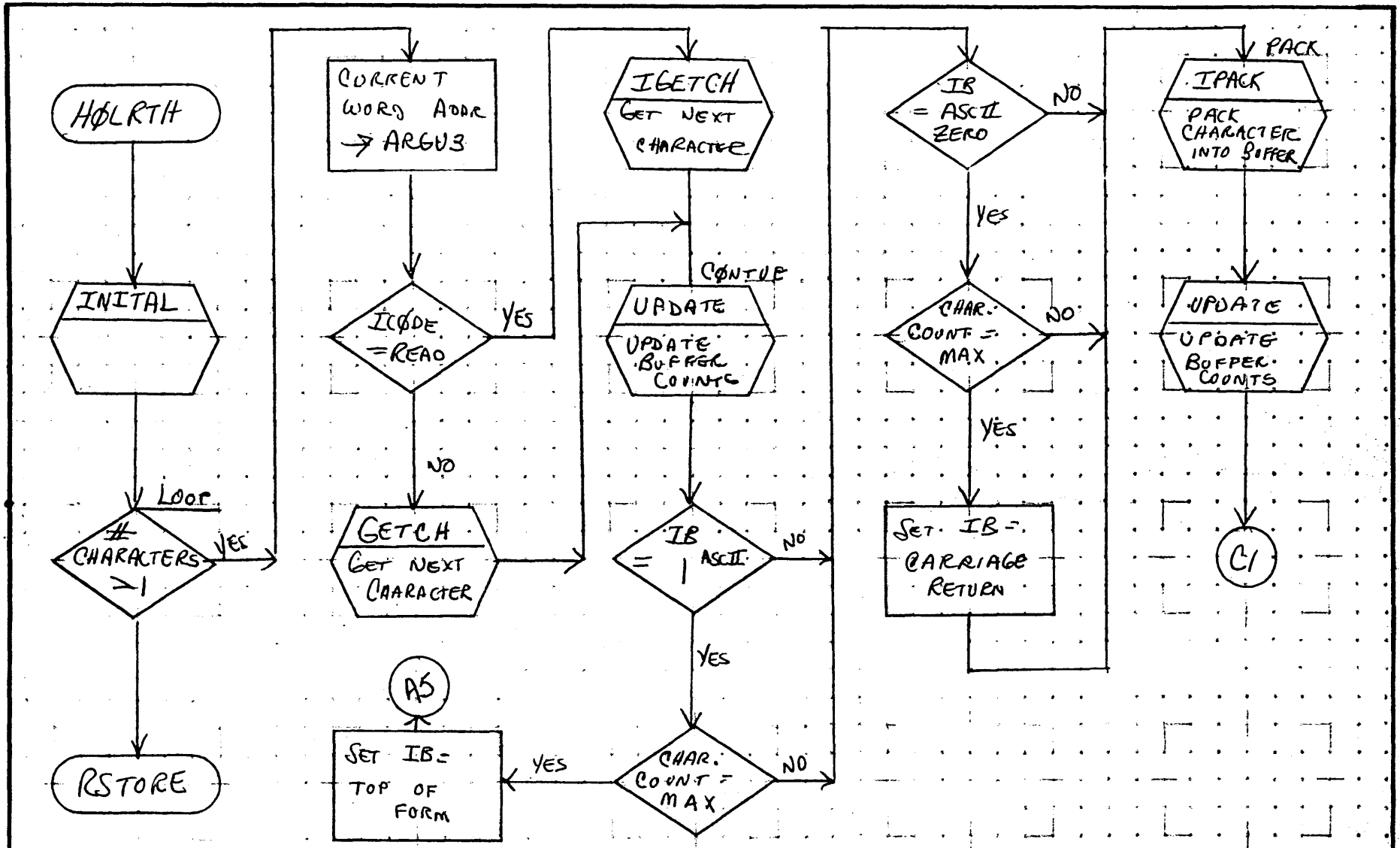
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS Ims	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	IGETCH	PAGE 17 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

66.27

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



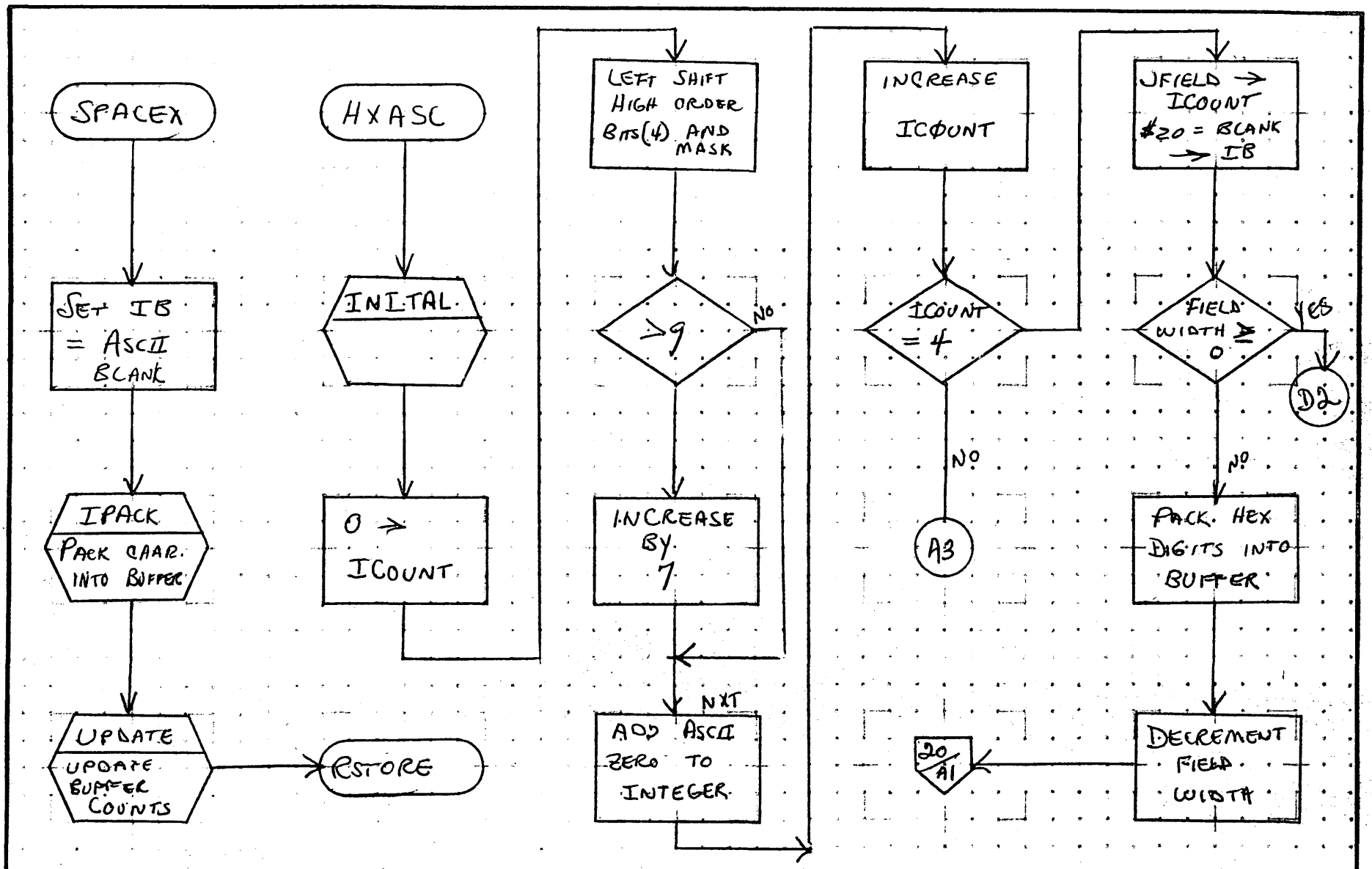
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 7700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
FLOWCHART ☐	HØLRTH	PAGE 18 OF 35	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

66.28

AA1385 (FORMERLY CA127-1)

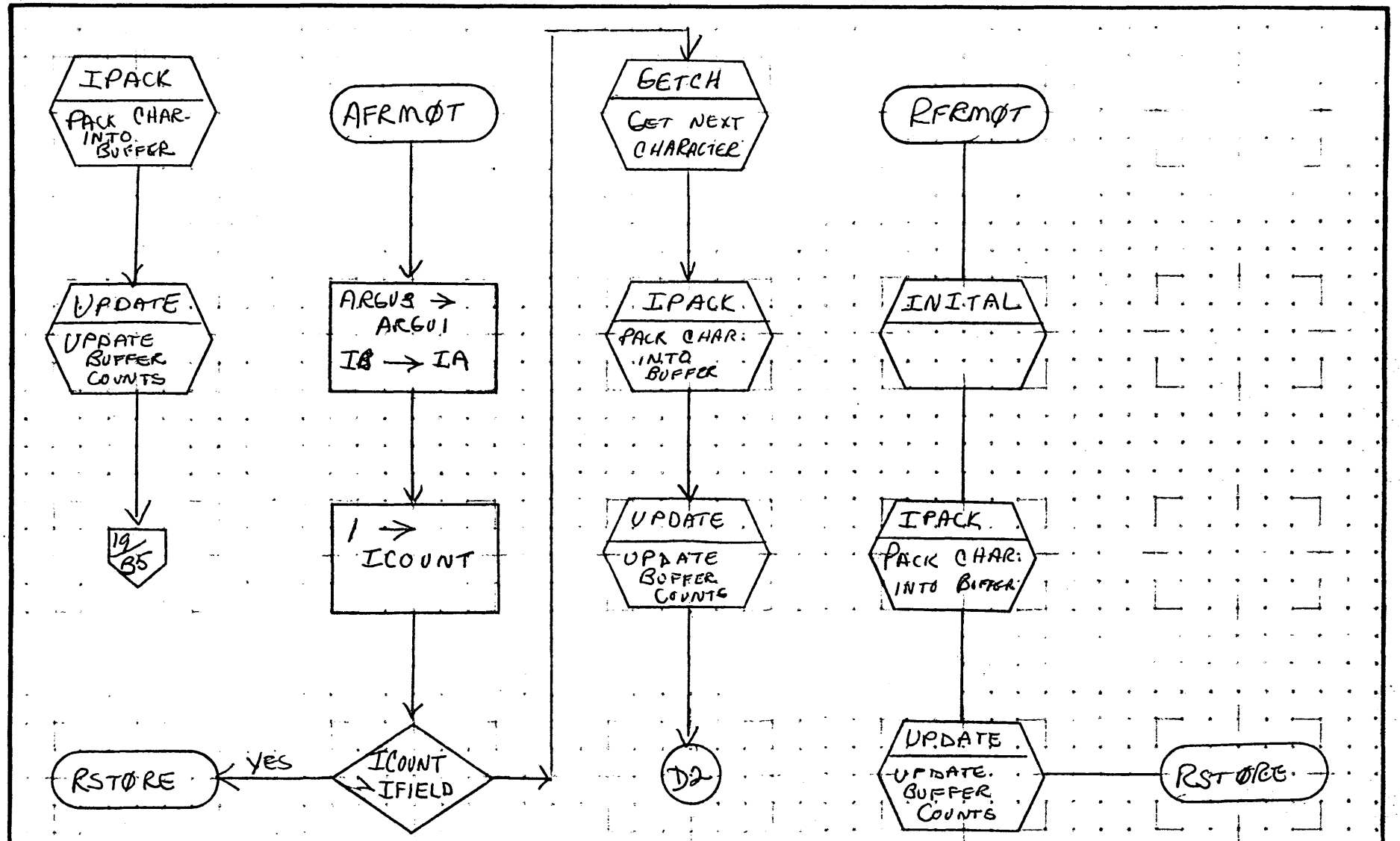
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	SPACEX - HXASC	PAGE 19 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

bb.29



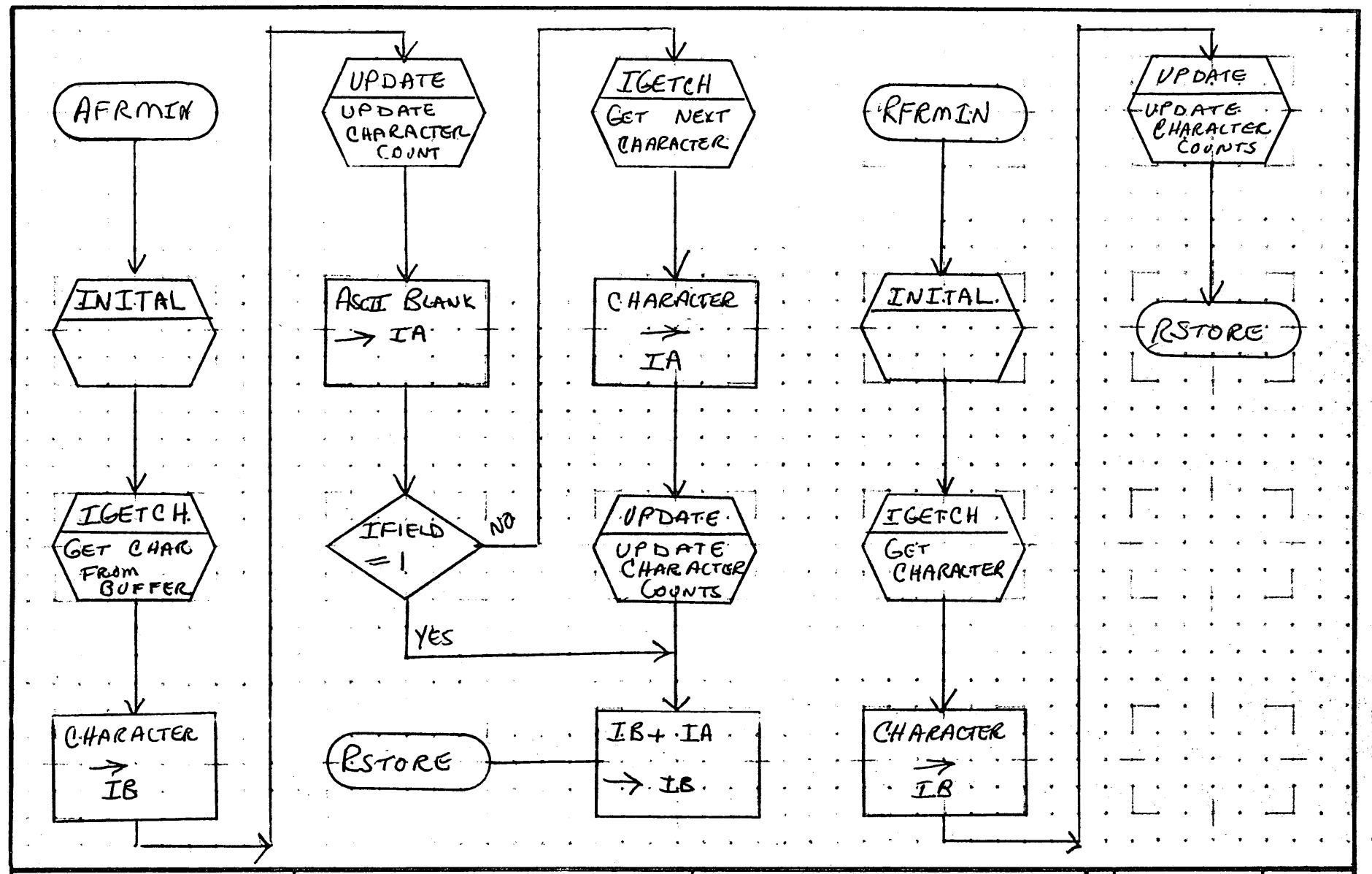
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
FLOWCHART ☐	HYASC	PAGE 20 OF 35	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

66.30

AA1385 (FORMERLY CA127-1)

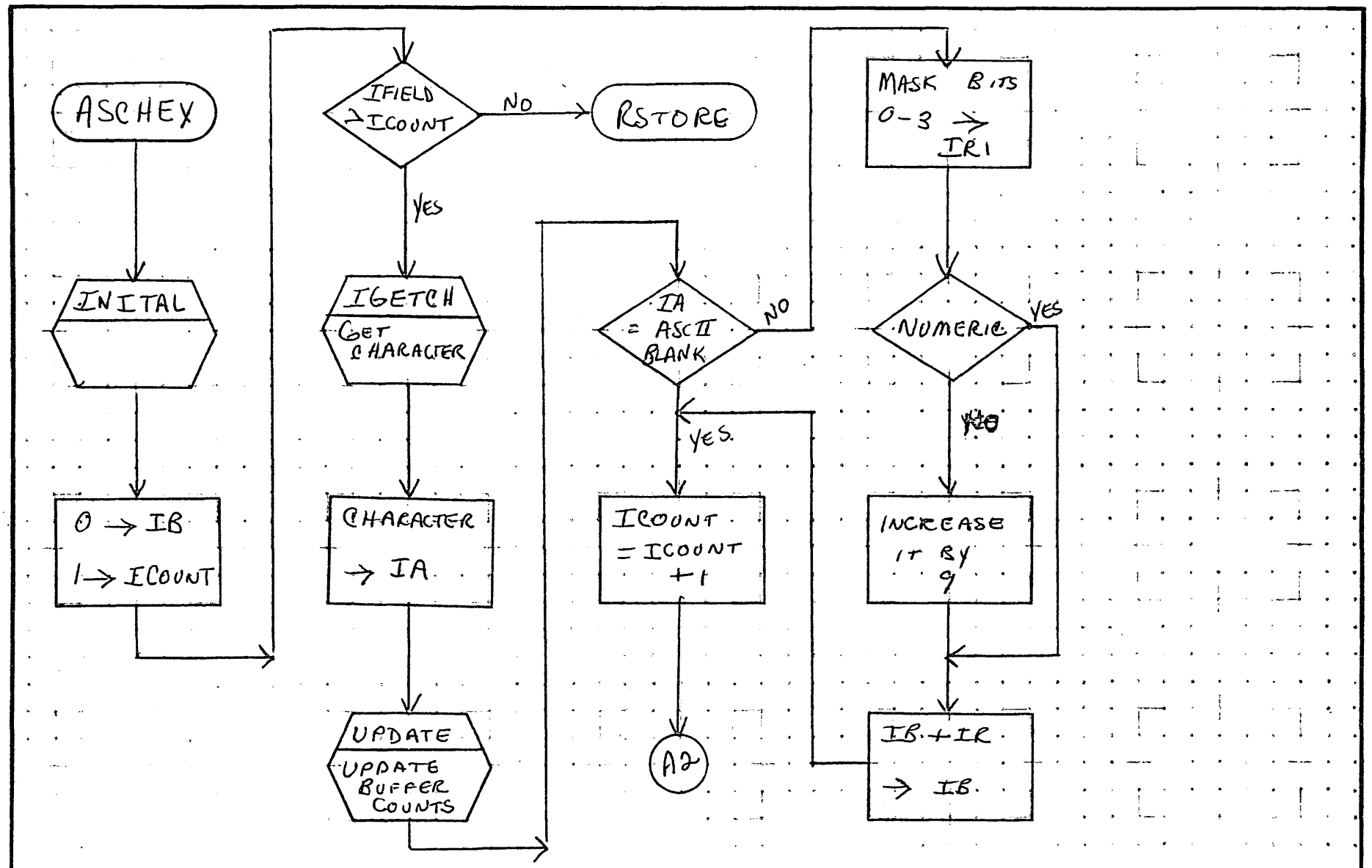
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	AFRMIN - RFRMIN PAGE 21 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

60.31.1



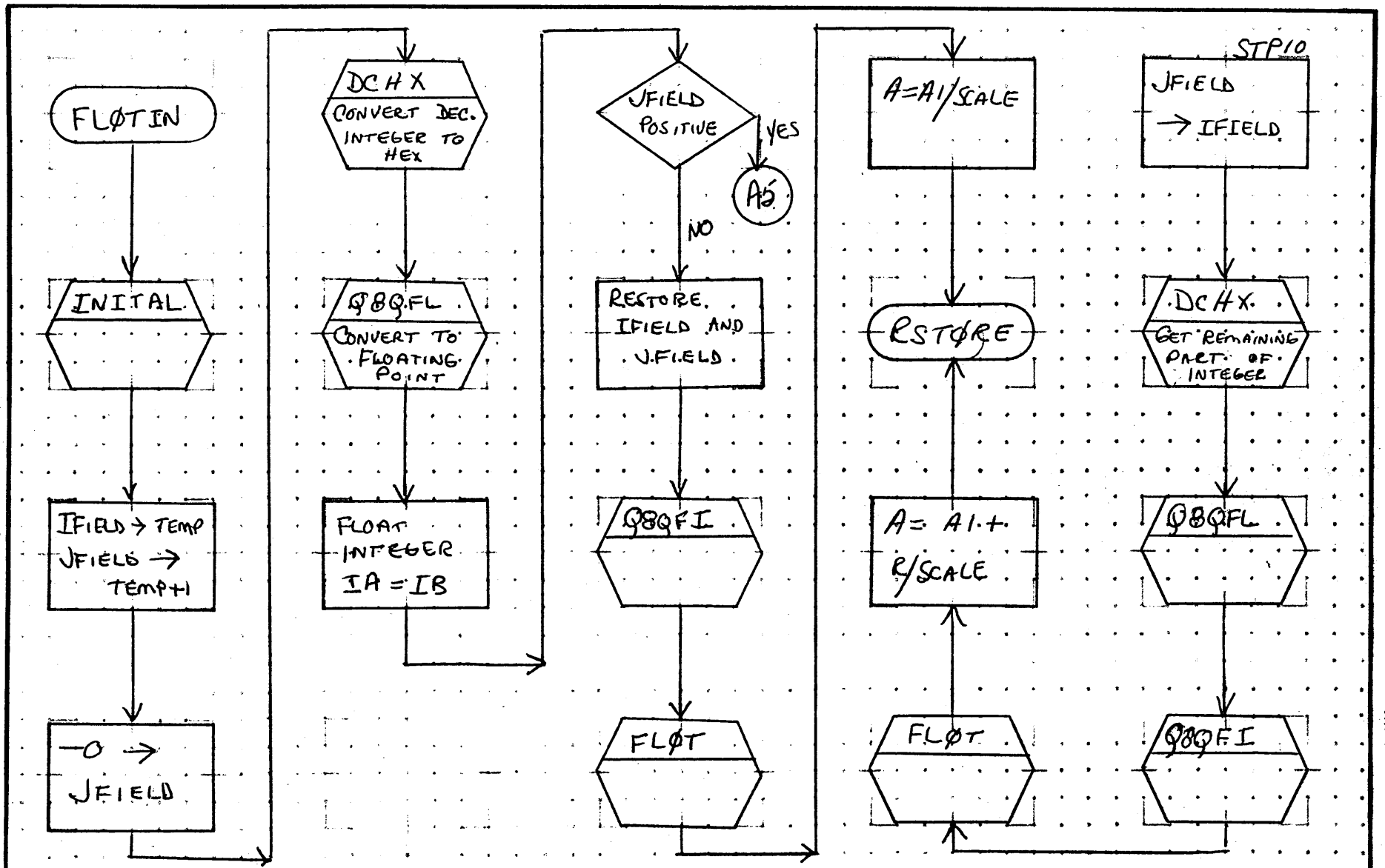
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ENCODE/ DECODE		PROJECT MGR.			
FLOWCHART ☐		PAGE 22 OF 35	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

64-32

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

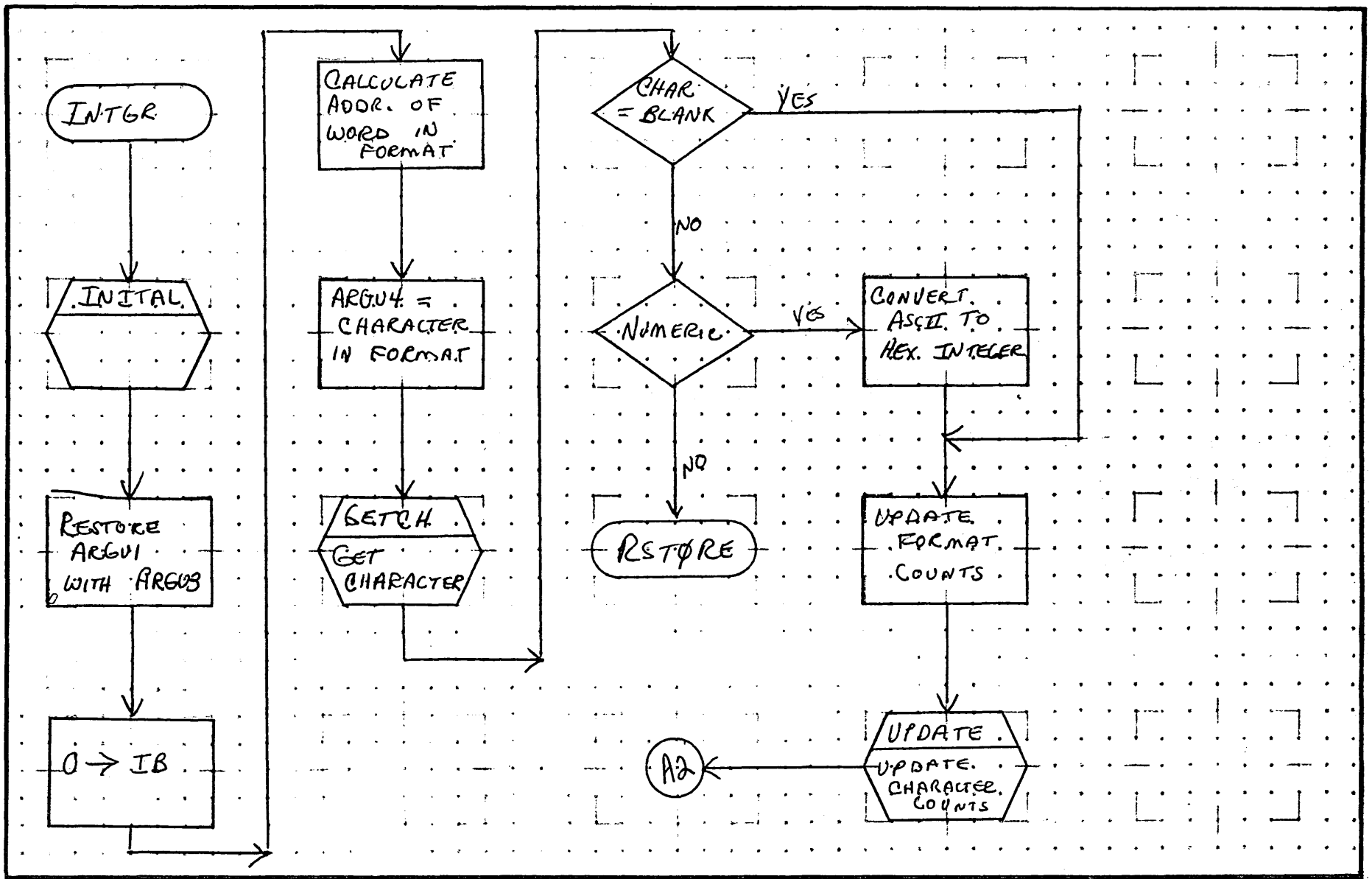


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.			
FLOWCHART ☐		PAGE 23 OF 35	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



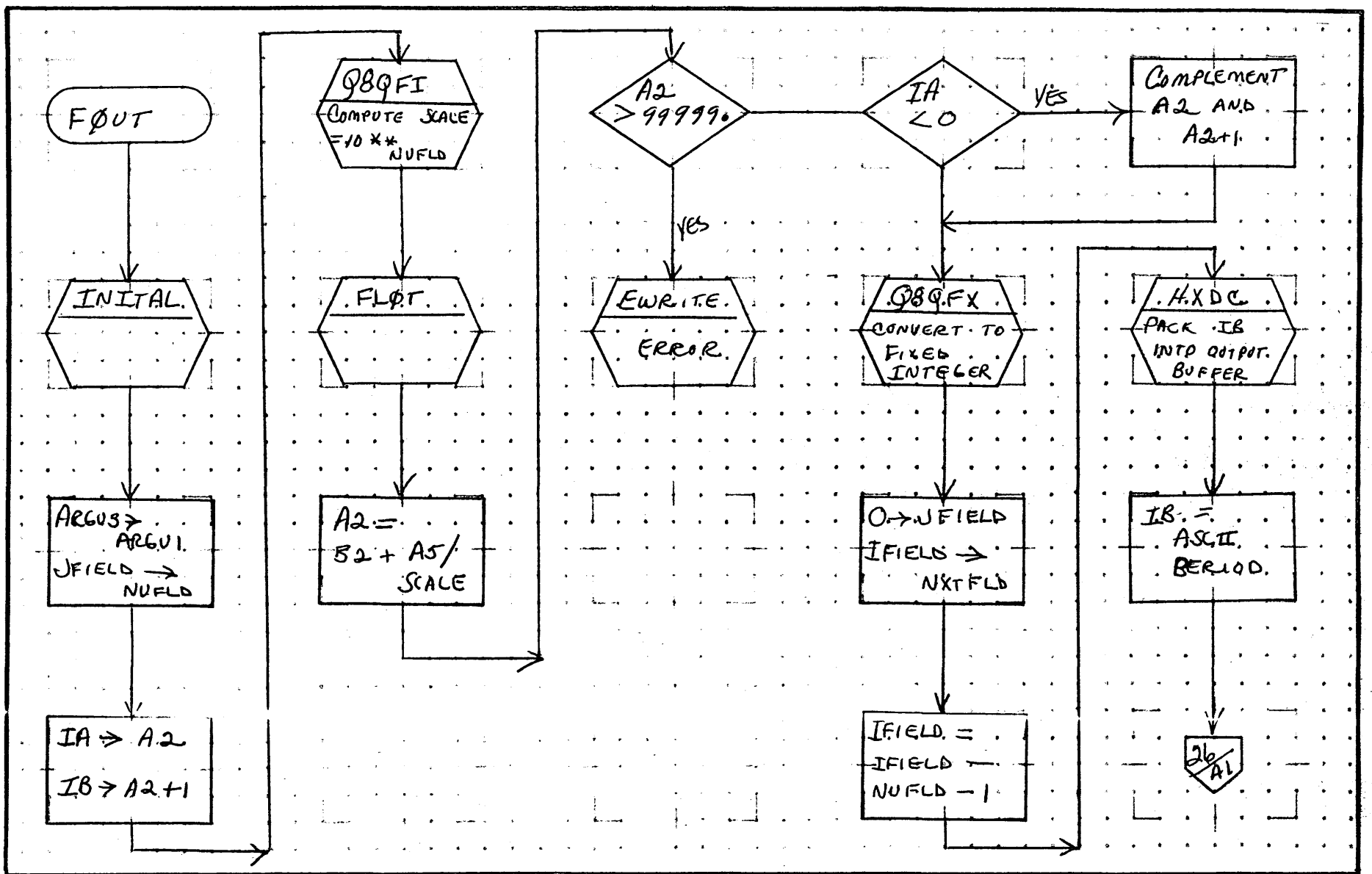
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	INTGR	PAGE 24 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

60.34

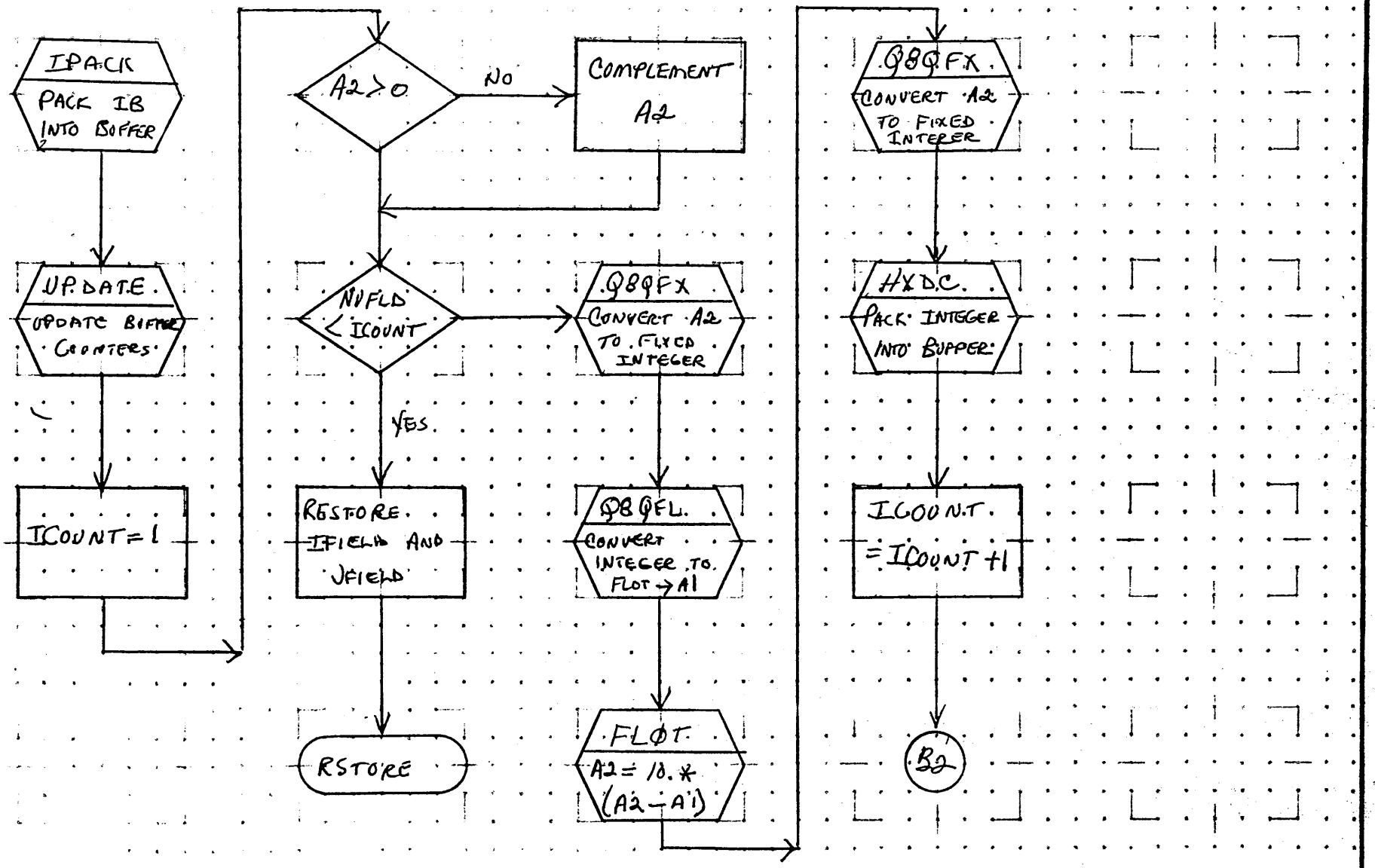
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	FOUT	PAGE 25 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971 66.35



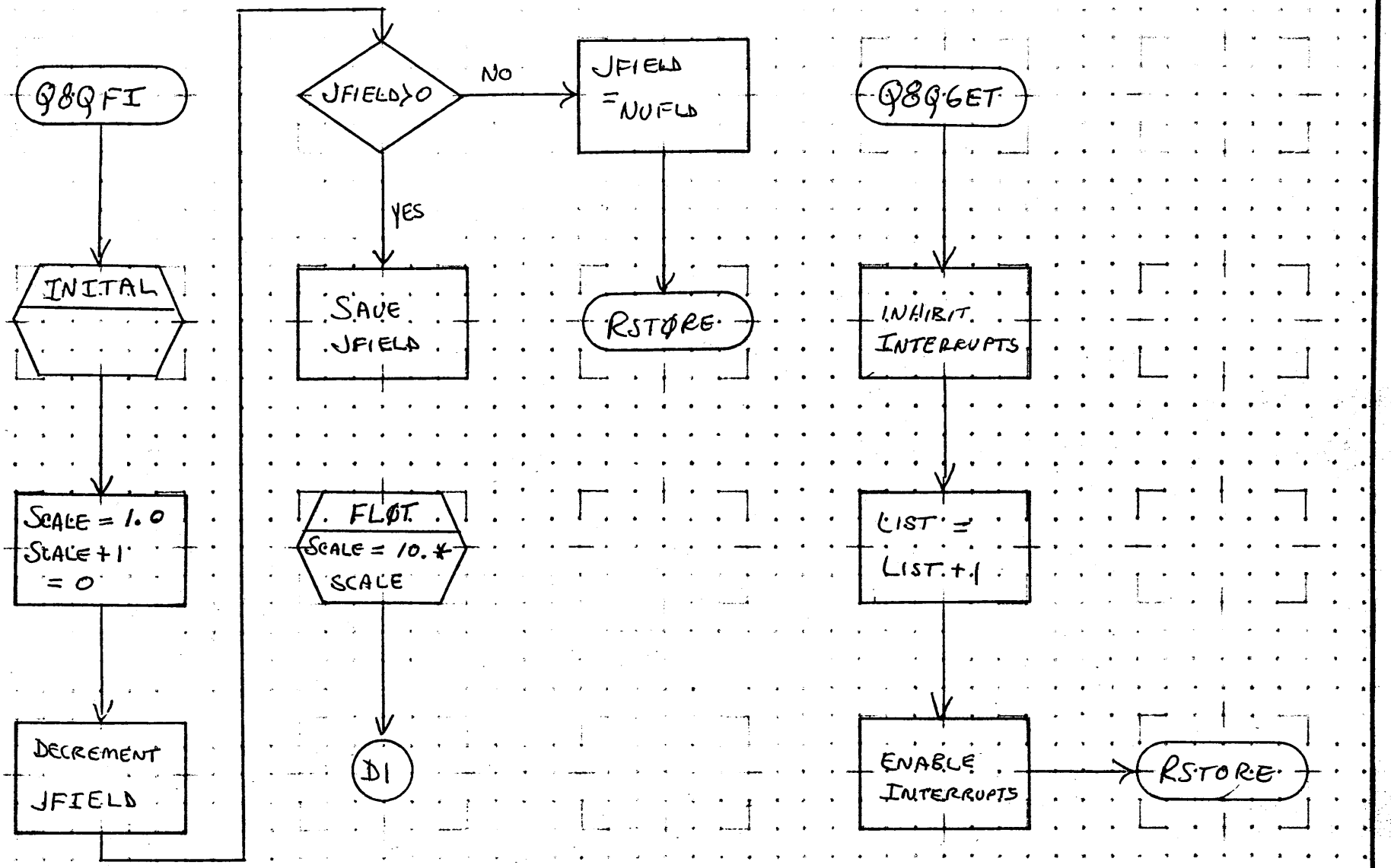
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	FLOT	PAGE 26 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

60.36



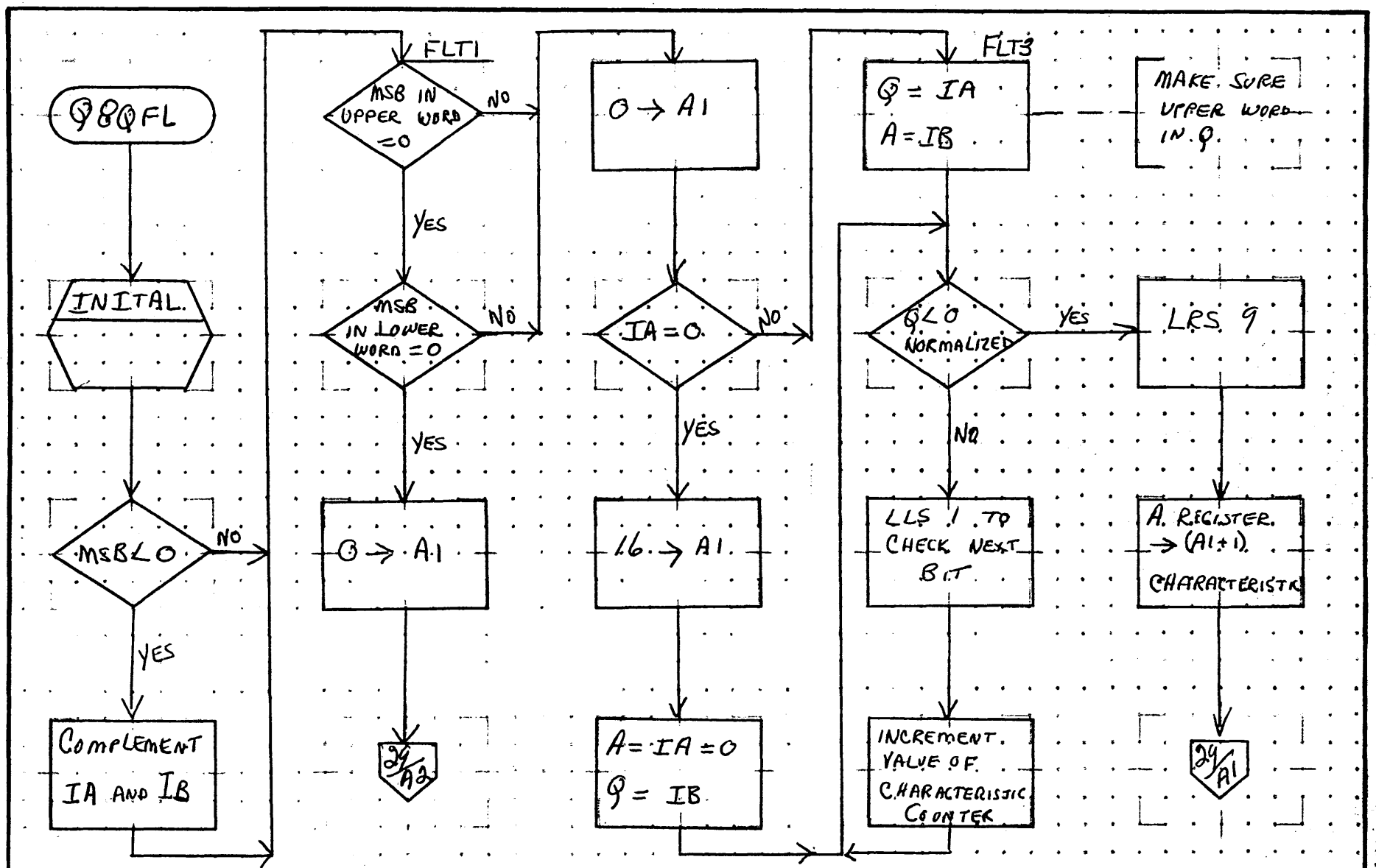
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	Q8QFI – Q8Q6ET	PAGE 27 OF 35	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

60.37

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



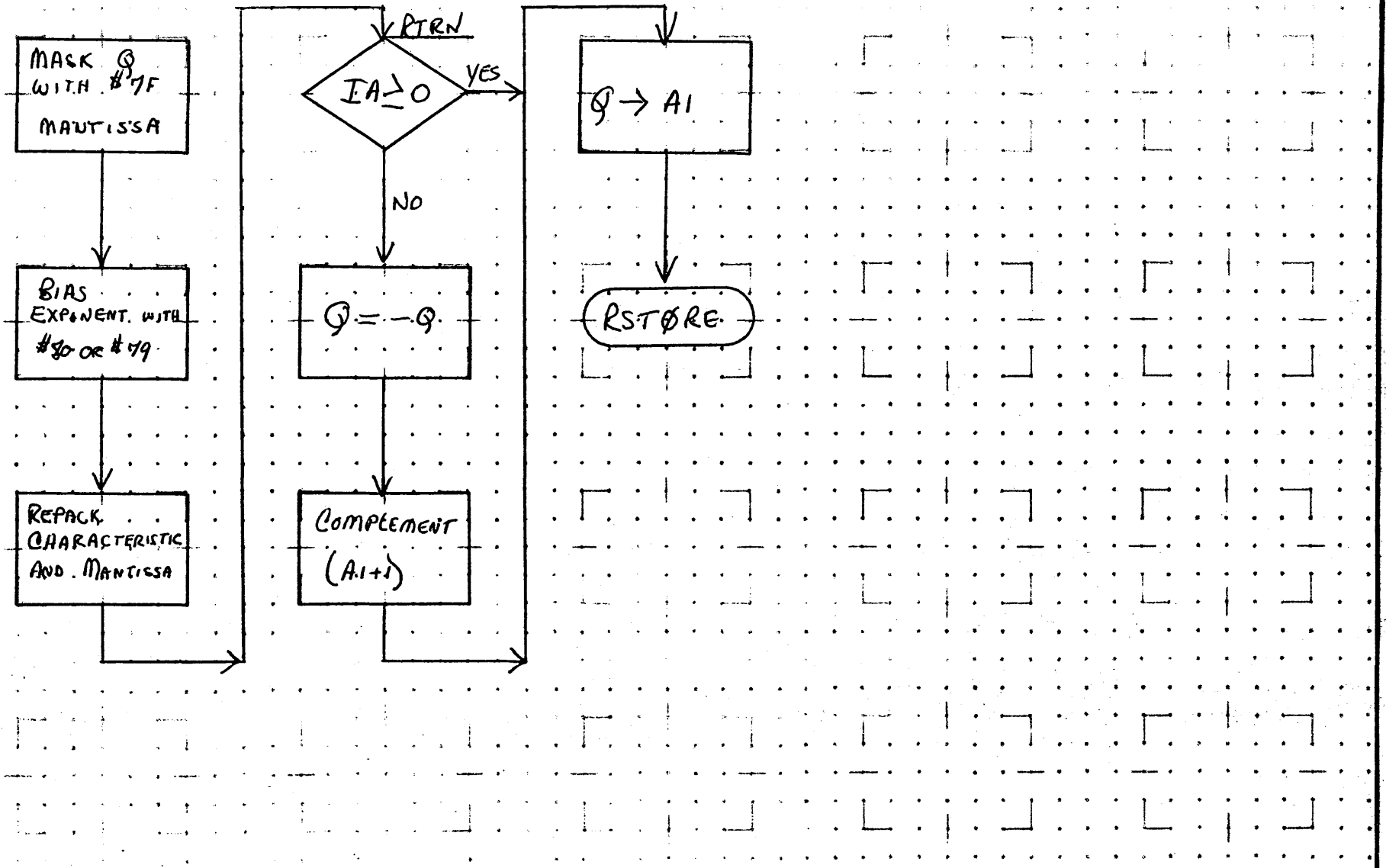
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☐	PAGE 28 OF 35	PROJECT NAME			
DECISION TABLE ☐	NUMBER / ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY / DATE	TASK NAME			

MAR 5 1971

60-36

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



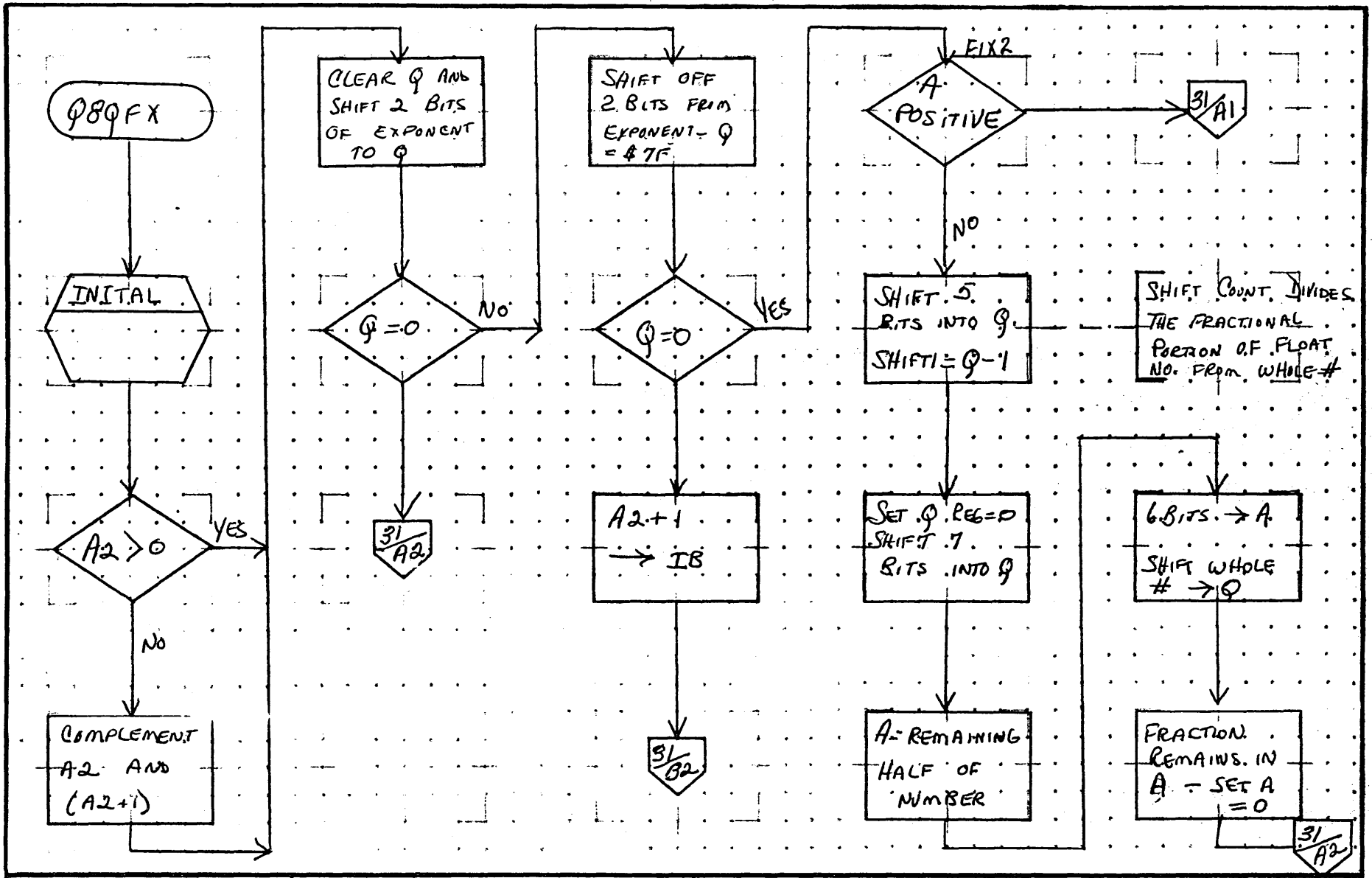
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE	PROJECT MGR.			
	PAGE 29 OF 35	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

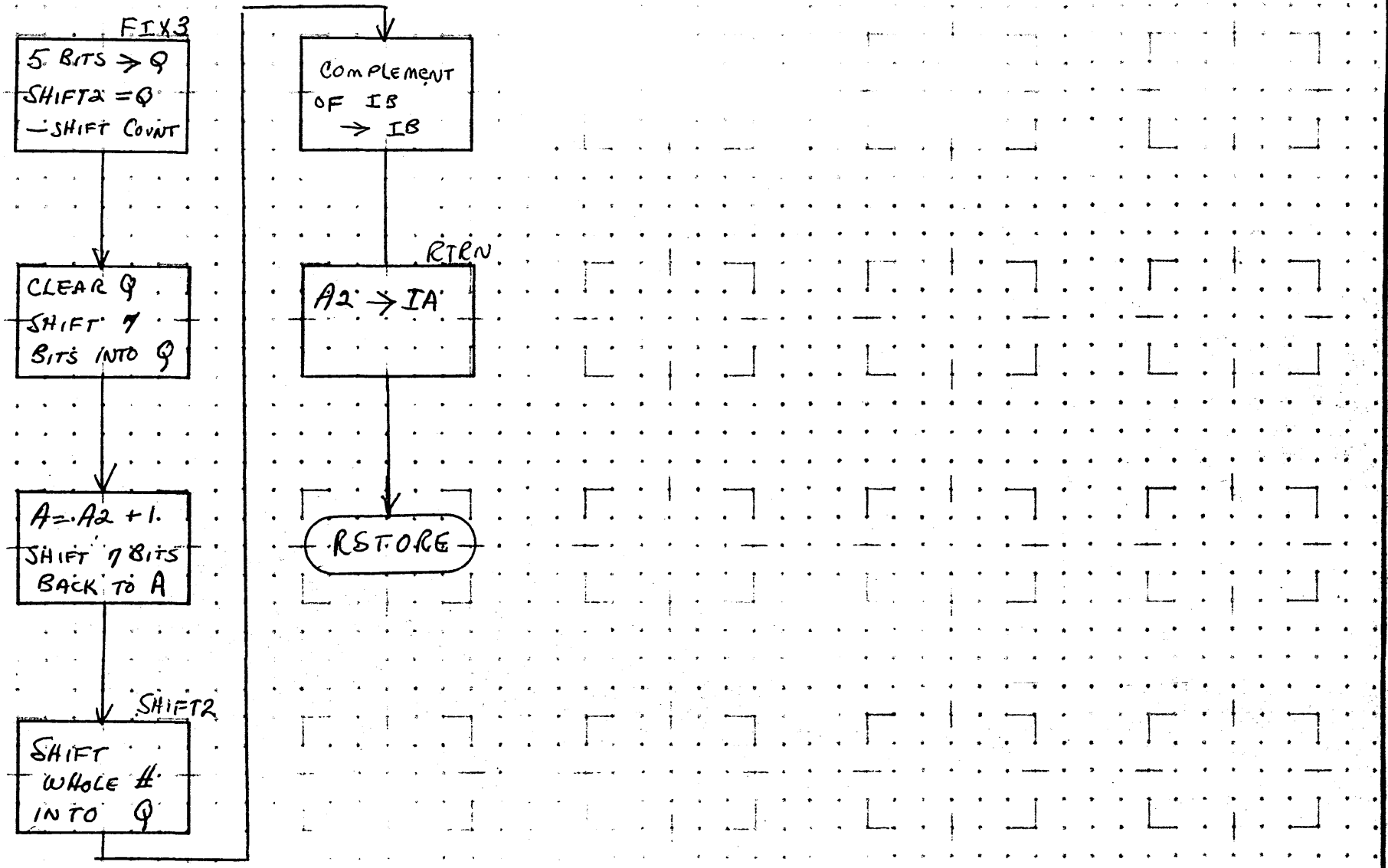
66.39



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	Q8QFX	PAGE 30 OF 95	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

60.40



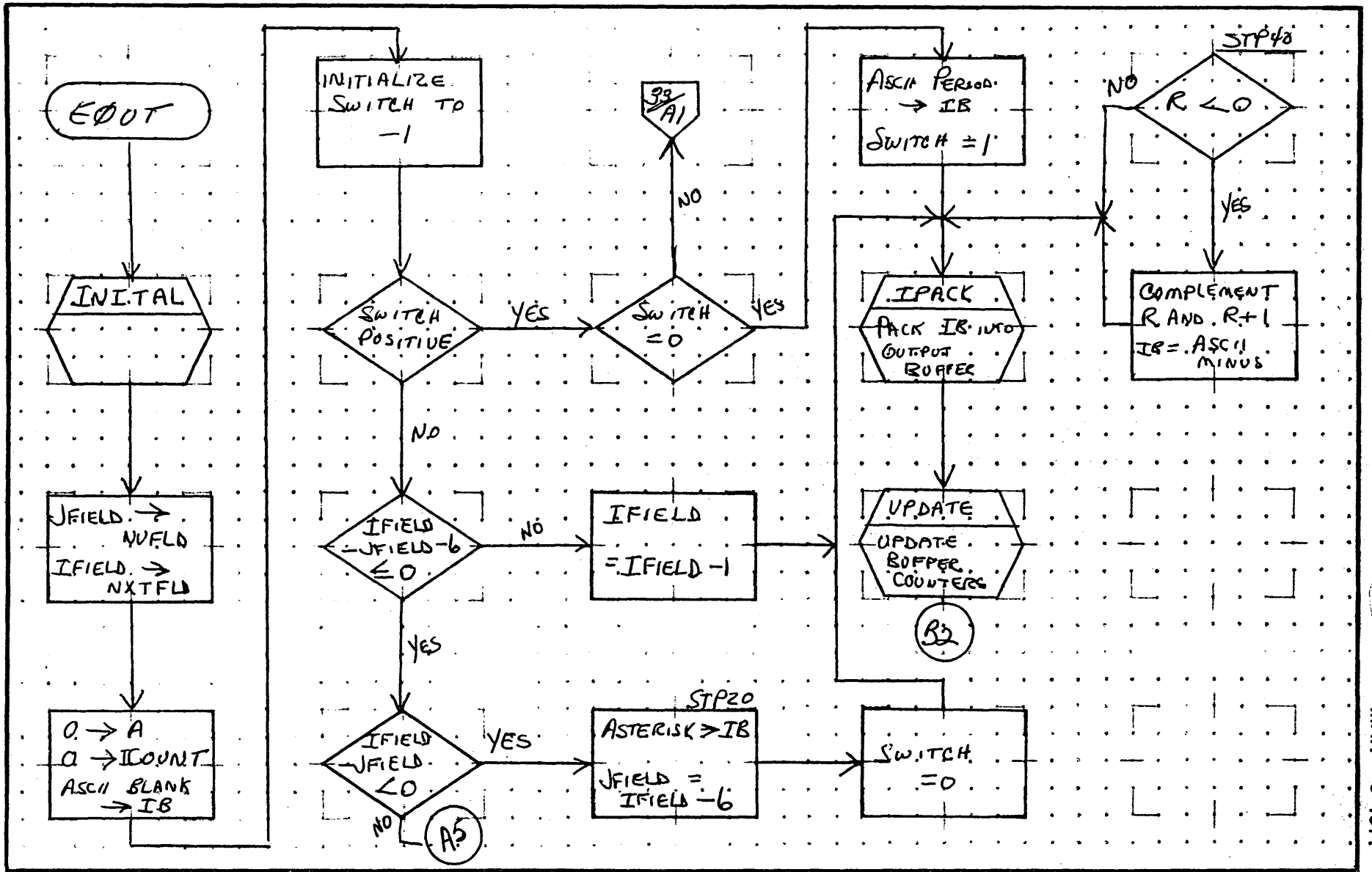
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	PAGE 31 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

64.41

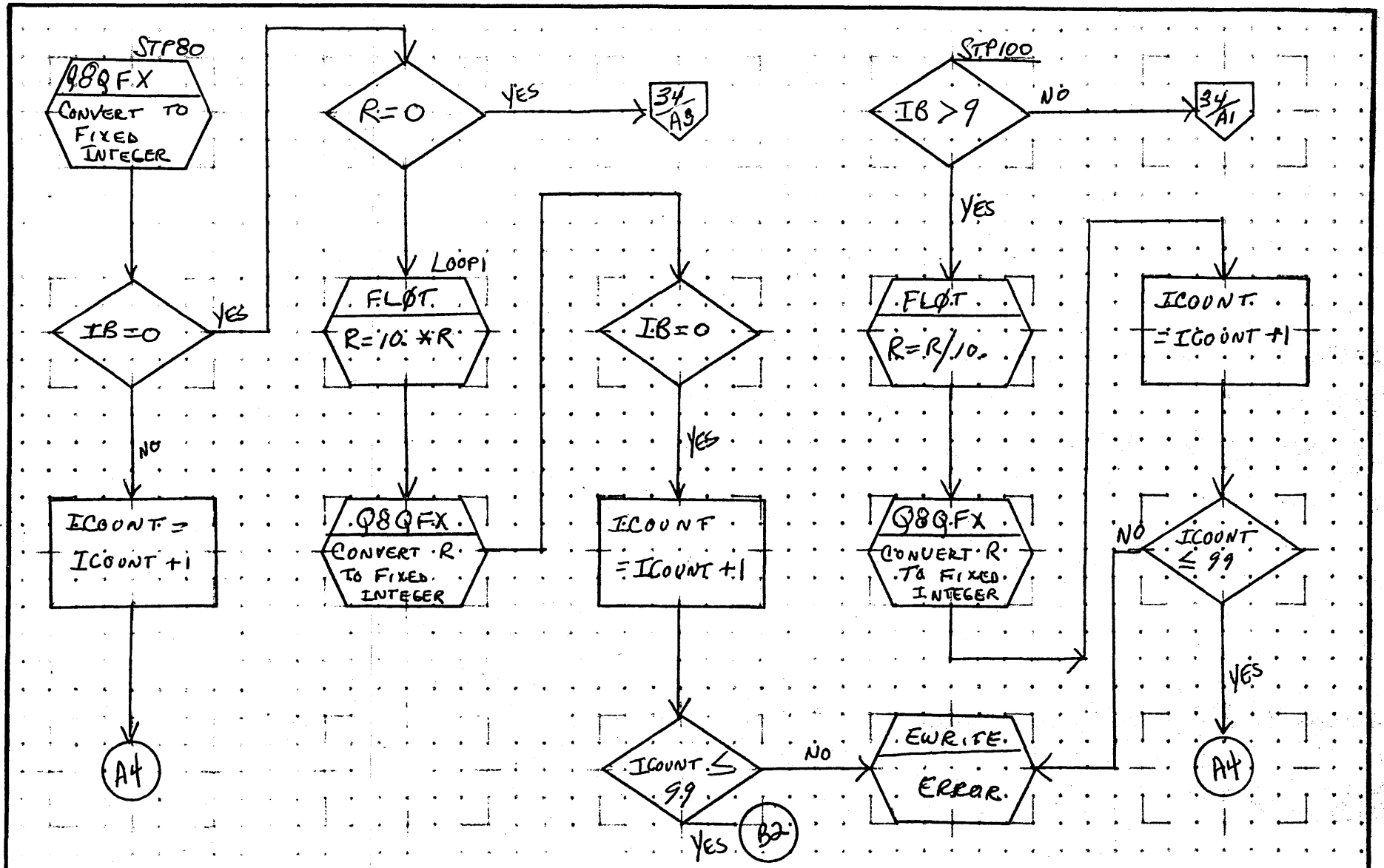
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

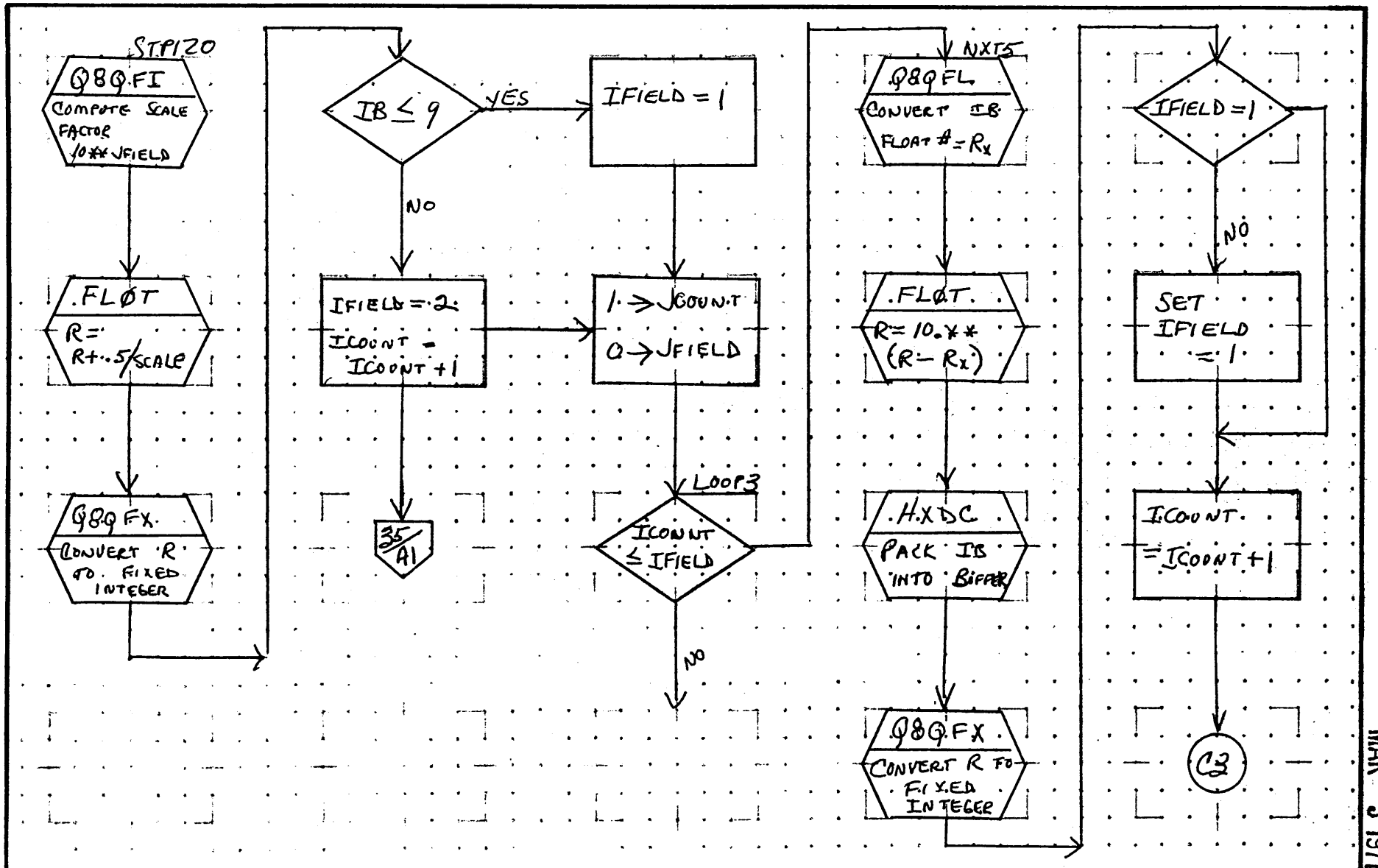


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/ DECODE		PROJECT MGR.		
	EØUT	PAGE 82 OF 325	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971 66.42



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	EDOT PAGE 33 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			



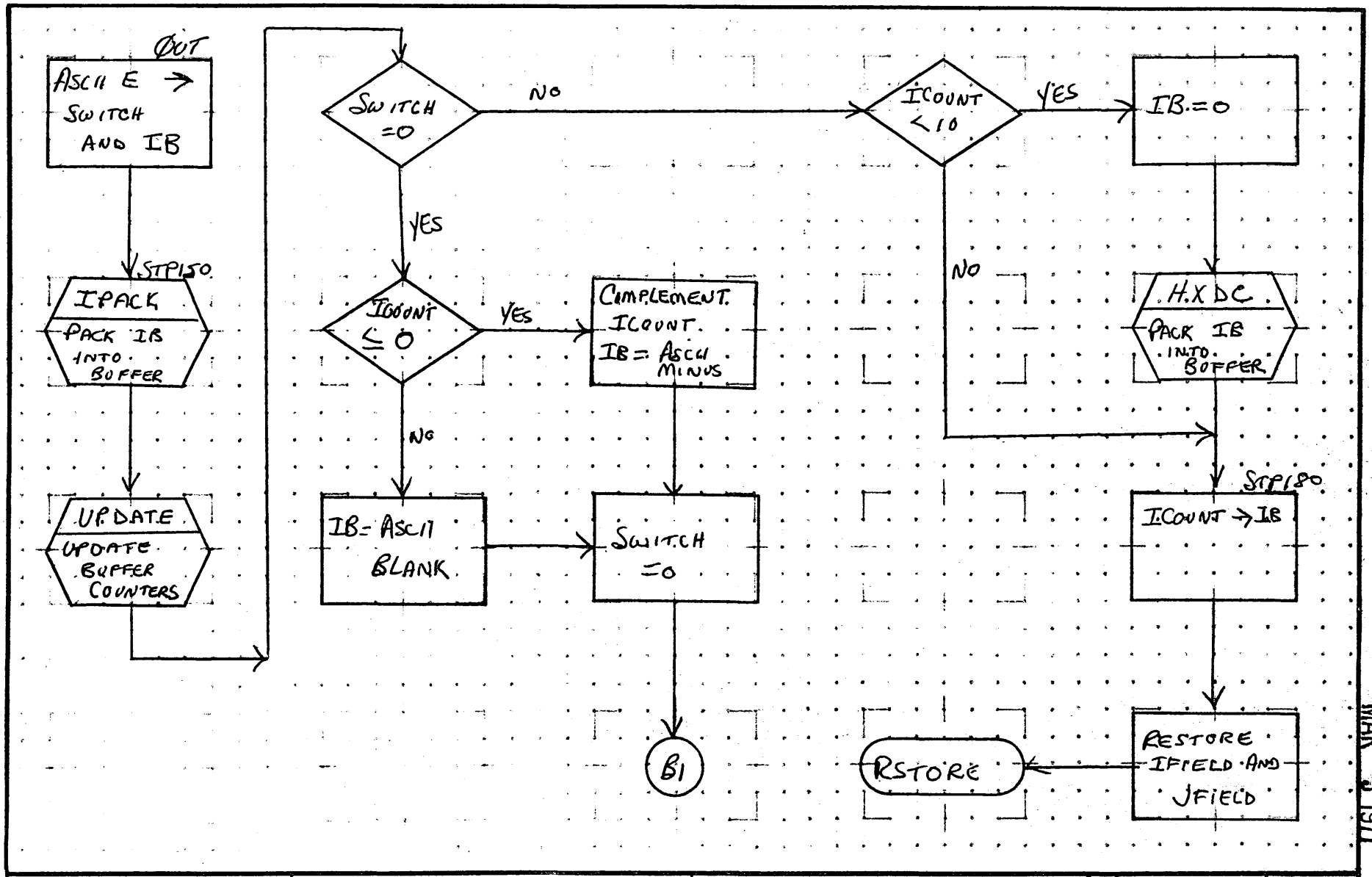
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE ENCODE/DECODE		PROJECT MGR.		
	EOUT	PAGE 34 OF 55	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

MAR 5 1971

66.44

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENCODE/DECODE	PROJECT MGR.			
	EOUT PAGE 35 OF 35	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

MAR 5 1971

69.45

67.0 Output Message Buffering Package

67.1 Function

The Output Message Buffering Package completes user message output by writing the message into a buffer area of core or mass memory. The message is subsequently read back from the buffer to a core area {character buffer} from which actual message output to a physical device is made by a monitor FWRITE request. A buffer physical equipment table is required for each buffer and a corresponding logical unit is assigned.

User requests are made by making FWRITE monitor calls that address a buffer input logical unit. The corresponding buffer table includes the buffer output logical unit number to be used for actual output and the request priority for actual output. The buffer table also defined the available buffer area in core or on mass memory and the size of the character buffer.

67.2 Entry Points

BUFDRI	Buffer Driver Initiator	{Input Section}
BUFDRC	Buffer Driver Continuator	{Input Section}
BWRITC	Transfer to Buffer Completion	{Output Section}
BREADC	Transfer from Buffer Completion	{Output Section}
BOUTPC	Actual Output Completion	{Output Section}

67.3 Externals

ALTDEV	Alternate Device Handler

67.4 General Program Information

67.4.1 Definition of Terms

BUFFER - BUFFER AREA: The area of core or mass memory reserved for storing messages prior to actual output.

BUFFER INPUT LOGICAL UNIT: The Logical Unit used to request buffered message output.

BUFFER OUTPUT LOGICAL UNIT: The Logical Unit used by the Message Buffering Package for actual output of the message.

BUFFER TABLE - BUFFER PHYSICAL DEVICE TABLE - BUFFER PHYSICAL EQUIPMENT TABLE: The equipment table corresponding to the Buffer Input Logical Unit. Includes all parameters for definition and use of the buffer.

CHARACTER BUFFER: The core area reserved for holding the message or portions of the message dump actual output.

MESSAGE: The data specified for output by a user request.

67.4.2 Internal Symbols

BFLEVL	Priority Level of the Buffer Package
BFFMLU	Logical Unit for the Mass Memory Device

These symbols must be defined by EQUs when the BUFFER macro is used. They are normally assigned as follows:

```
EQU BFLEVL {10}
EQU BFMMLU{$8C2}
```

67.4.3 Buffer Macro

BUFFER F, L , H, LU, RP, N

F = Start Address of Buffer {LSB}

L = End +1 Address of Buffer

H = Most Significant Bits of Buffer Address {MSB}
{Blank for a Core Buffer}

LU = Logical Unit for actual output

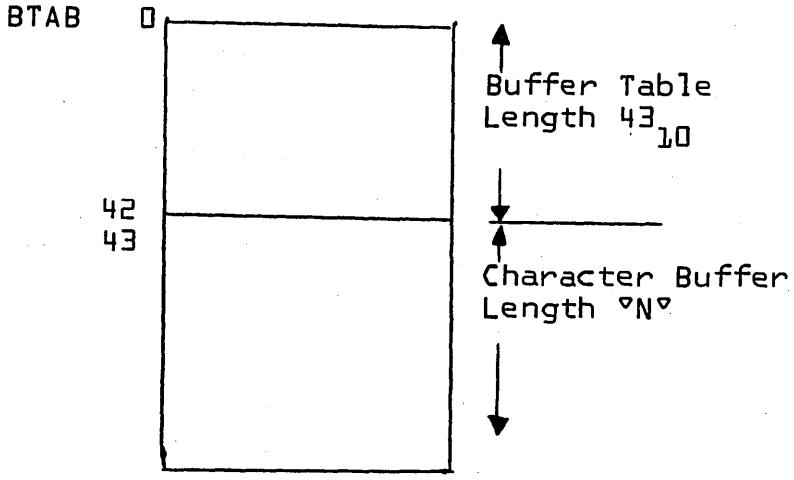
RP = Request Priority for Buffer Output on this lu

N = Size of Character Buffer for actual output

This macro generates the Buffer Physical Device Table and the Character Buffer area. The character buffer will follow the last word of the Buffer Physical Device Table. The address of the Buffer Physical Device Table must be added to the L0G1A Logical Unit Table and its relative position in that table determines the logical unit number assigned to the Buffer.

The corresponding L0G1 and L0G2 entries are 0 and $FFFF_{16}$ respectively.

Data generated by BUFFER macro:



▽N▽ is the length of the character buffer as specified in the macro call.

▽BTAB▽ is the address of the Buffer Table that must be put in the L0G1A logical unit table.

67.4.4 Buffer Table

The buffer table is the physical equipment table for the buffer. It is extended beyond the standard 13 words common to all drivers. The following diagram shows the assignment of symbolic names and the initial setup of the buffer table.

F,L,H,LU,RP,N are as defined for the BUFFER macro
LV = BFLEVL = Priority Level {=9}
BFMMLU = Mass Mem. Logical Unit {=$8C2}
CHBUFF is the Character Buffer Address {=BTAB+43}

Symbolic Addresses

0	ELVL
1	EDIN
2	EDCN
3	EDPGM
4	EDCLK
5	ELU
6	EPTR
7	EWES
8	EREQST
9	ESTAT1
10	ECCOR
1	ELSTWD
2	ESTAT2
3	TIMER
4	LOCB
5	ENDB
6	FIRST
7	LAST
8	DPLO
9	1
20	2
1	3
2	DLEG
3	DART
4	DTRACK
5	STOR
6	CONTRL
7	DOUTO
8	1
9	2
30	3
1	OUTLNG
2	DADR
3	OUTTK
4	READ
5	SKELNG
6	OUTPO
7	1
8	2
9	3
40	4
1	ACHAR
2	LCHAR

CONTROL DATA CORPORATION
Arden Hills Development DIVISION

DOCUMENT CLASS IMS PAGE NO. 67.5.1
PRODUCT NAME 1700 OPERATING SYSTEM
PRODUCT MODEL NO. E006*3.0 MACHINE SERIES 1700

		Initial Setup
Standard Physical Equipment Table	0	$1200+LV
Initiator Entry	1	BUFDRI
Continuator Entry	2	BUFDRC
Diagnostic Entry	3	BUFDRC
Diagnostic Clock	4	-1
Logical Unit Assigned	5	0
Adr. of Request	6	0
Hardware Address	7	0
Type Code	8	$A4
Status Word 1	9	0
Start Code Address	10	0
End Core Address +1	1	0
Status Word 2	2	0
No of attempts	3	0
Buffer Start	4	F
Buffer End +1	5	L
Temp Buffer Start	6	F
Temp Buffer End +1	7	L
Mass Memory WRITE	8	$04F0+LV
Completion Address	9	BWRITC
Thread	20	0
Logical Unit	1	BFMMLU
Length	2	0
Core Address	3	0
MSB of M.M. Address	4	H
Buffer Store Pointer	5	F
Control Word	6	0
Mass Memory READ	7	$200+16*LV+LV
Completion Address	8	BREADC
Thread	9	0
Logical Unit	30	BFMMLU
Length	1	0
Core Address	2	CHBUFF
MSB of M.M. Address	3	H
Buffer Read Pointer	4	F
Control Word	5	0
Character Output FWRITE	6	$C00+16*RP+LV
Completion Address	7	BOUTPC
Thread	8	0
Output Logical Unit	9	LU
Length	40	0
Address of Char. Buffer	1	CHBUFF
Length of Char. Buffer	2	N

BUFFER OUTPUT PHYSICAL DEVICE TABLE

67.5 Description

The buffer table includes the initiator continuator and diagnostic time-out entries to the buffer package in words 1 - 3. Therefore, a request for buffered output will cause the input section of the buffer package to be scheduled unless it is busy. If it is busy, requests are threaded in the normal manner.

The input section is concerned with transfer of messages into the buffer. This section operates just like the driver for a physical device. The buffer area correspond to the physical devices to be driven. FNR (find next request subroutine) is used to pick up the request parameters. If no requests remain then the program exits to the dispatcher, otherwise, an attempt is made to write the message into the buffer. The I register holds the buffer table address.

Tests are first made to determine that the length of the message +1 is small enough to fit into the available area of the buffer. The next location to store into is saved in the STOR pointer in the buffer table. If the end of the buffer is reached, a temporary end location, LAST, is set equal to STOR then STOR is set back to the start of the buffer. Insufficient space is available if the length of the message +1 + STOR is equal to or beyond the current location for reading from the buffer, READ. When this happens, five successive tries will be made at 1 second intervals via the diagnostic time-out entry. (This feature is unavailable if the 1573 timer is not included in the system). If space is still unavailable after five seconds, the error code is set in Q and control passes to the alternate device handler.

One extra word is added to the message to save the length of the message in the buffer. This word CONTRL, is written in the buffer first, followed by the actual message. The buffer STOR pointer is updated on successful completion. The monitor request parameters for the mass memory transfer form part of the buffer table. Core buffering does not require the monitor request but the high order bits of the "mass memory" address are set to 8000_{16} to denote a core buffer.

On completion of buffering, COMPRQ (complete request subroutine) is executed to complete the user request. The output section is then scheduled unless output is already in progress for the buffer. Control then passes back to the initial entry to search for further requests.

Entry to the output section is made with the Q register set equal to the buffer table address. This address is saved in the I register. The output section transfers the message out of the buffer to the character buffer and makes an FWRITE request for actual output to the physical device. All the parameters for the mass memory transfer and the FWRITE output are held in the buffer table.

The current location to read from the buffer is held in location READ in the buffer table. A transfer of length equal to the character buffer length is made starting with this location. For core buffers this is a simple core to core transfer. Otherwise, a monitor READ request is used to transfer data from the mass memory device. On successful completion, the READ pointer is updated by the length of the transfer. New messages are recognized by the contents of SKELNG, being reset from the first word of the data just transferred from the buffer.

The remaining length is then calculated by subtracting the length just transferred from SKELNG. If SKELNG is zero the message is complete. If SKELNG is negative too much data was transferred and the READ pointer and the length for actual output must be adjusted to the actual message length. If SKELNG is greater than zero, the message is incomplete and further transfers will be required. When the message is complete, the temporary start of buffer pointer FIRST is set to READ. If FIRST is now equal to LAST, FIRST and READ are reset to the actual start and LAST to the actual end of the buffer.

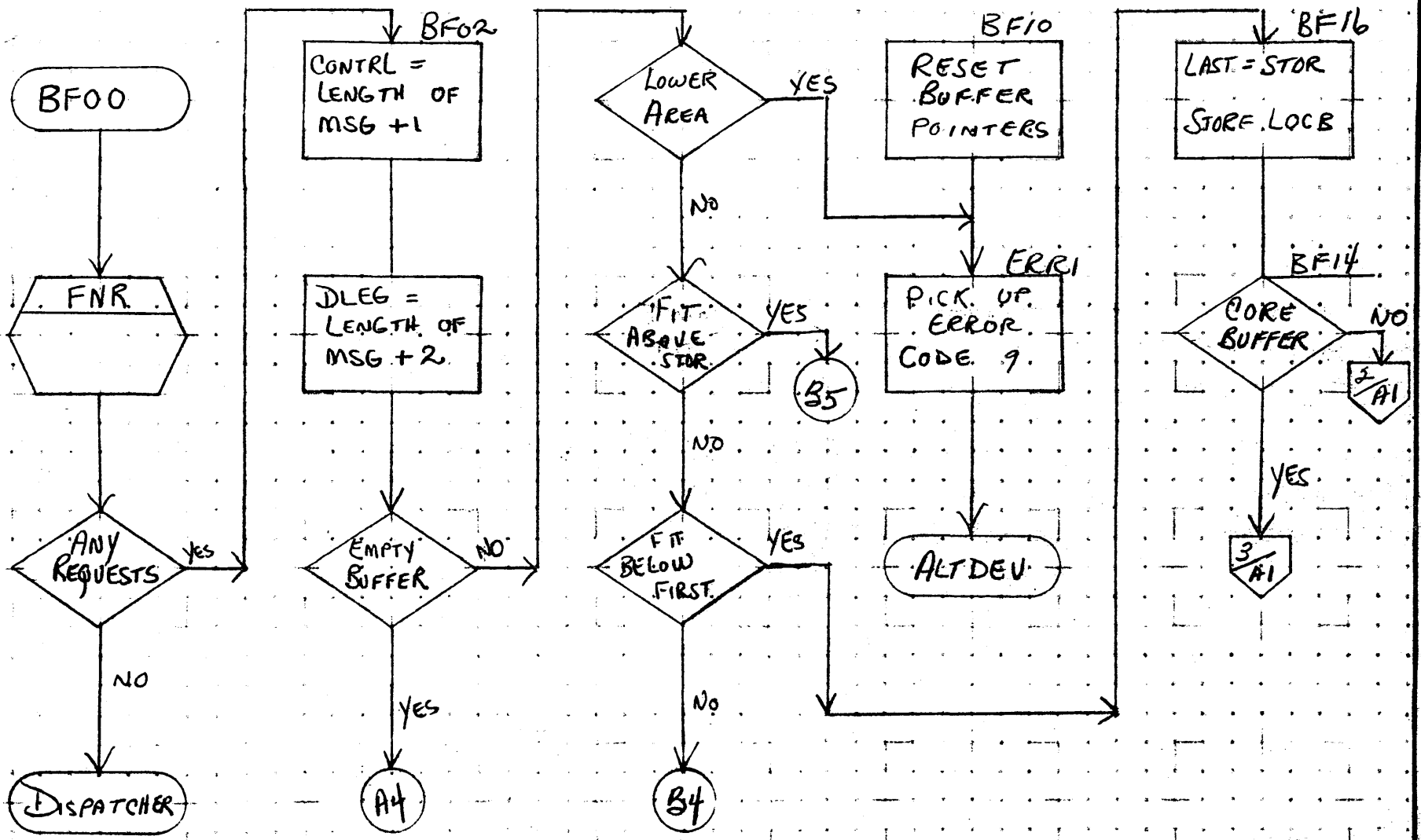
Actual output of the data in the character buffer is then initiated via a FWRITE monitor request to the buffer output logical unit specified in the buffer table. On completion of actual output control returns to the start of the output section where a test is made to determine if the buffer is empty, i.e., no messages remain. If so, the program exits, otherwise output continues.

Note that long messages will be split into separate FWRITE requests each of length equal to the length of the character buffer, (except that the last request may be shorter). This may result in undesired line feed control action in some cases. The user may prefer to change the monitor request for actual output to a WRITE in such cases (Word OUTPB).

Transfers to and from mass memory buffers are made using monitor READ/WRITE requests with word addresses in the same format as for the 1751 drum driver. Two WRITE requests are made to transfer data to the buffer, one for the length control word, the other for the message. The number of READ requests for transfer from the buffer is equal to the length of the message divided by the length of the character buffer.

67.6 Special Requirements

The area of core or mass memory specified as the buffer area must be reserved for exclusive use of the buffer package at System Initialization. A BSS block may be provided in the system tables for core buffering. For mass memory buffers the *M,hhhh,s Initializer statement may be used.

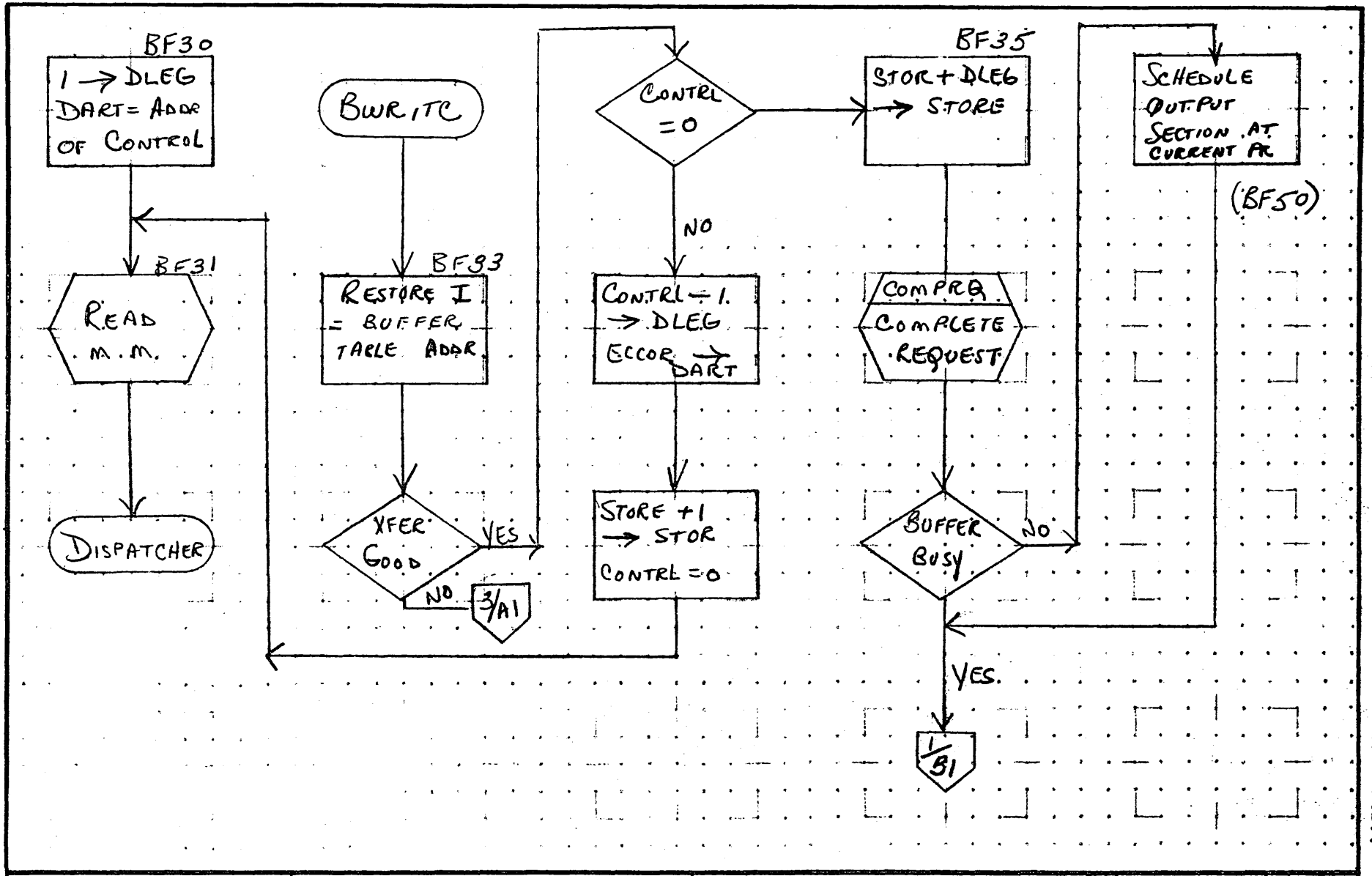


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: IMS — MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: BUFFER PACKAGE	PROJECT MGR.			
	PAGE 1 OF 6	PROJECT NAME			
	NUMBER — ISSUE DATE	TASK NO.			
	DRAWN BY — DATE	TASK NAME			

MAR 5 1971 67.9

AA1385 (FORMERLY CA127-1)

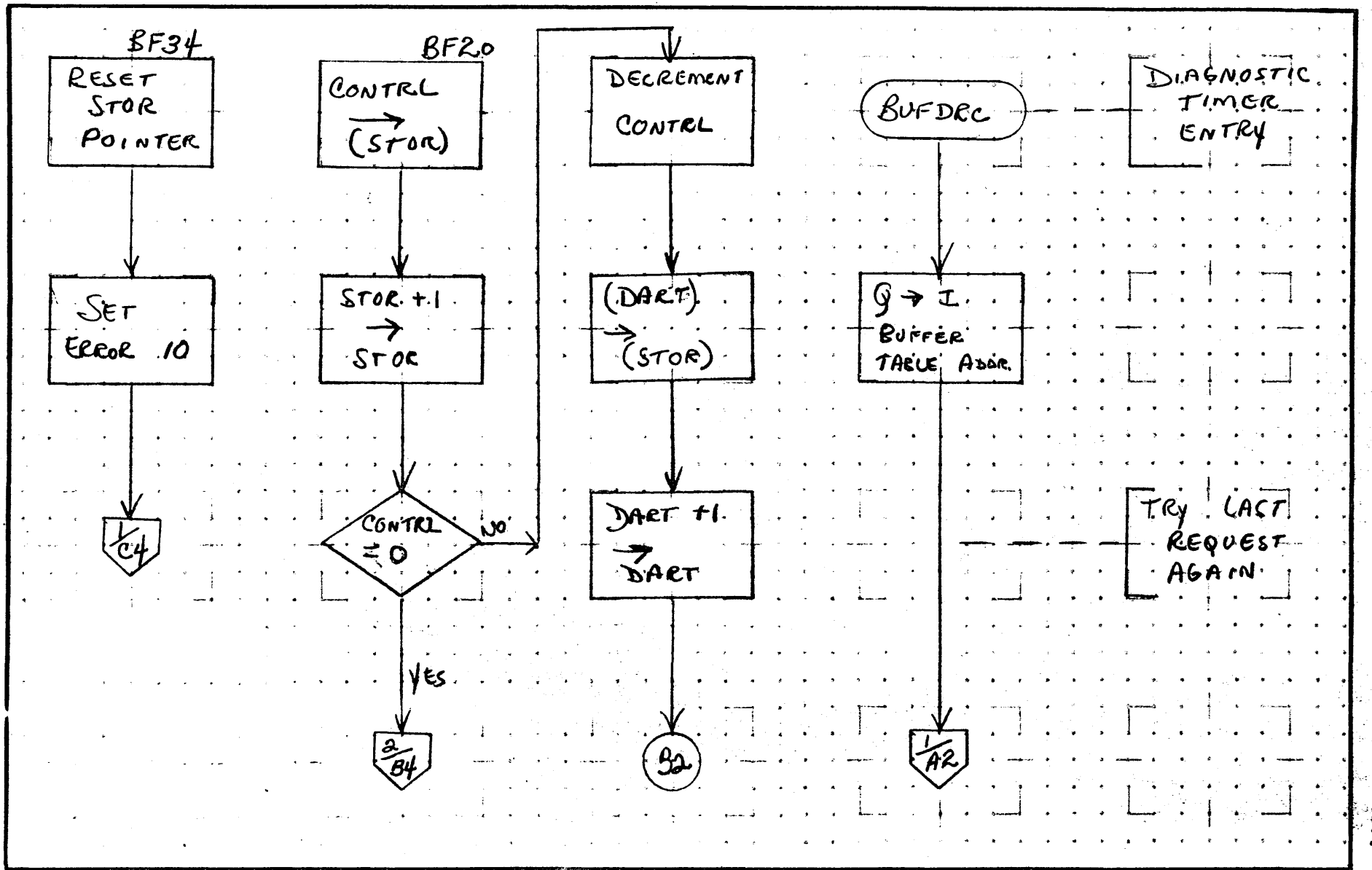
PRINTED IN U.S.A.



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE BUFFER PACKAGE		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

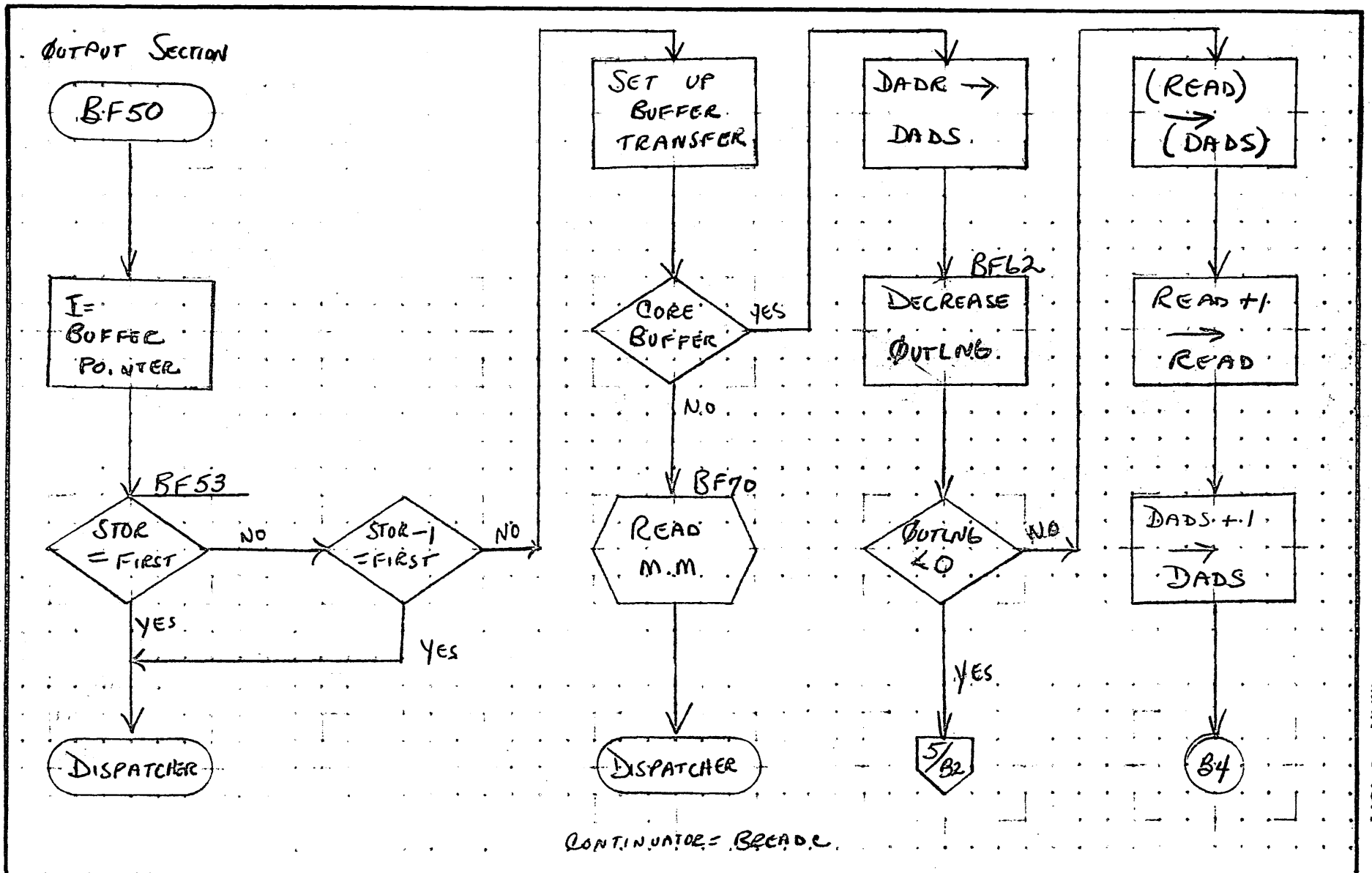
67.10



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: BUFFER PACKAGE		PROJECT MGR.			
FLOWCHART ☐		PAGE 3 OF 6	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

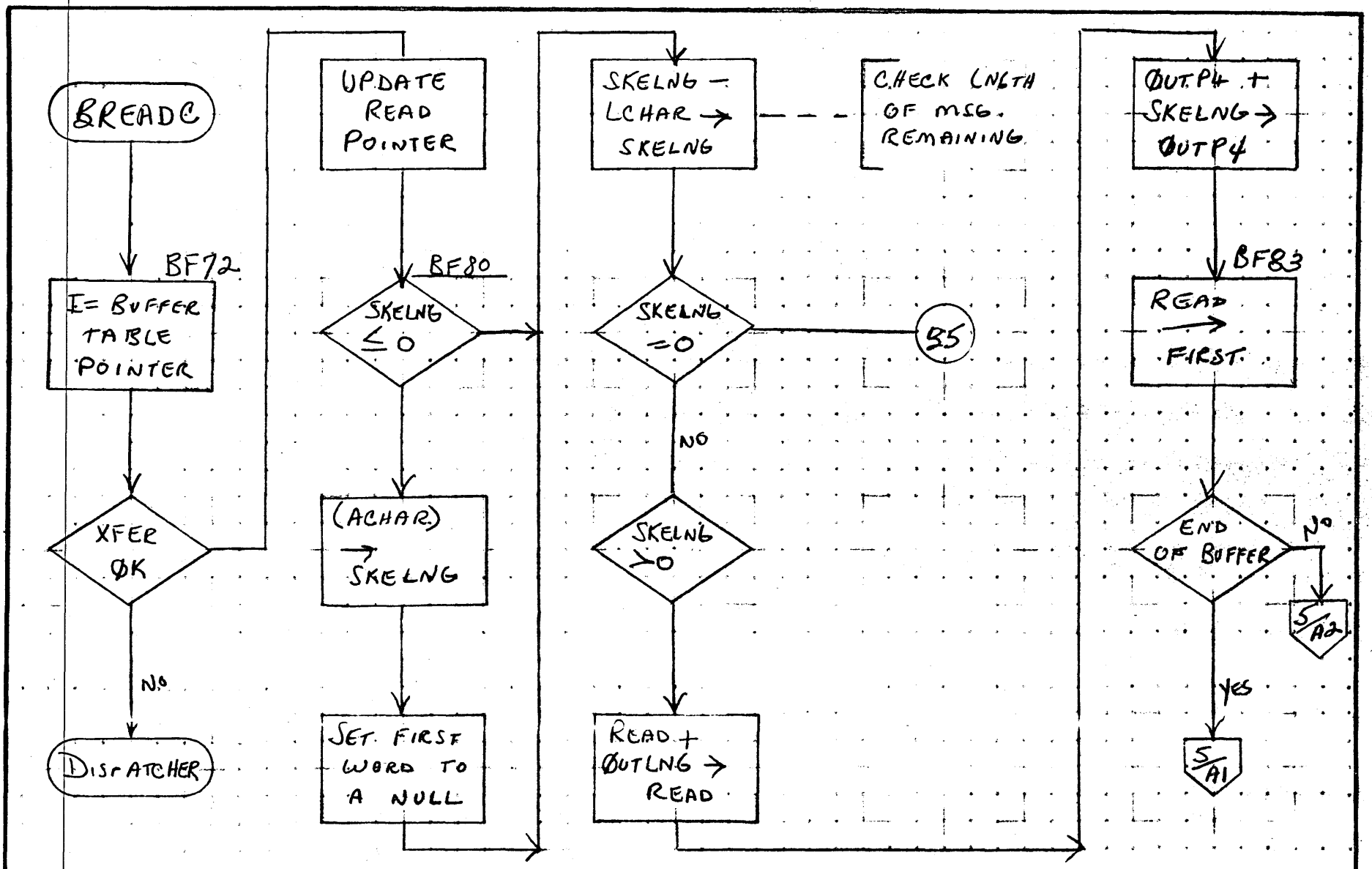
67.11



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE: BUFFER PACKAGE		PROJECT MGR.			
	PAGE 4 OF 6		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

MAR 5 1971

67.12



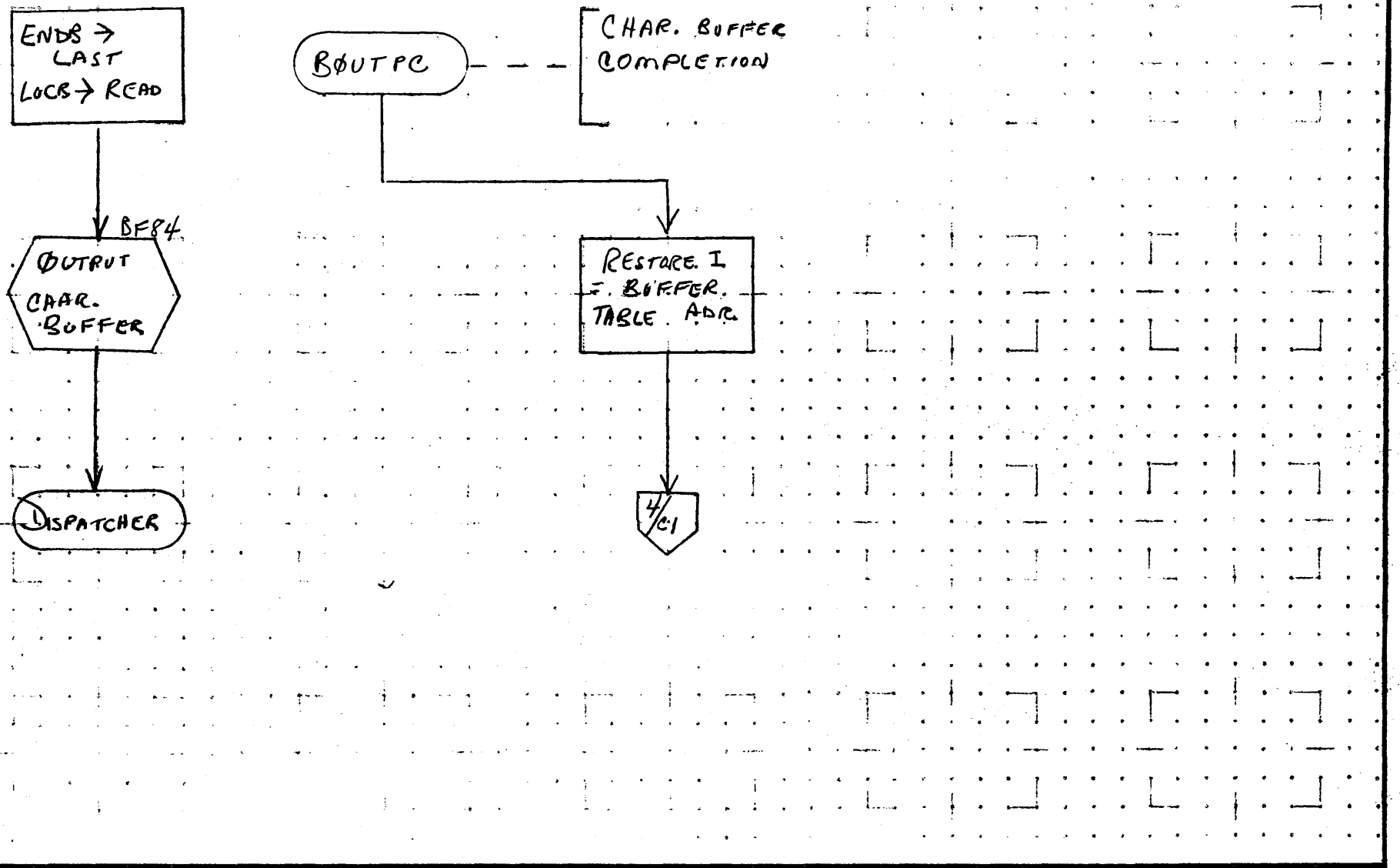
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BUFFER PACKAGE		PROJECT MGR.			
		PAGE 5 OF 6	PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

42 13



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BUFFER PACKAGE	PROJECT MGR.			
	PAGE 6 OF 6	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

MAR 5 1971

67.14