MSOS 65K Special Software

Addendum to MSOS 3.0

Internal Maintenance Specifications

TABLE OF CONTENTS

PAGE NO.

SECTION A - MODIFICATION TO EXISTING ROUTINE

1.0 MONITOR CHANGE 01

1.1 Routine - LOCORE 01
1.2 Routine - SYSBUF 01
1.3 Routine - NCMPRQ 01
1.4 Routine - NFNR 02
1.5 Routine - ADEV 02
1.6 Routine - RDISP 02
1.7 Routine - DRCORE 03
1.8 Routine - PARAME 04
1.9 Routine - COMMON 04
1.10 Routine - N1PROC 04
1.11 Routine - NMONI 04
1.12 Routine - RW 05
1.13 Routine - SPACE 05

2.0 DRIVERS 07

2.1 Routine - DR1728 07
2.2 Routine - DISKWD 07
2.3 Routine - DR1732 07
2.4 Routine - TAPE 07
2.5 Routine - DBLDRV 08

3.0 JOB PROCESSOR - 65K 09

3.1 JOBENT 09
3.2 T11 09
3.3 T7 10
3.4 T5 10
3.5 T3 10
3.6 JOBPRO 10
3.7 PROTEC 11
3.8 JBKILL 11
3.9 JPLOAD 12
3.10 JPCHGE 12
3.11 ASCHEX 12
3.12 T13 12
3.13 MIPRO 12
3.14 RESTOR 12

PAGE NO.

4.0 DEBUGGING AID ROUTINES 13

4.1 ODEBUG 13
4.2 Module - RDMPCR Module of Recovery Package 13

5.0 LIBEDT 14

5.1 Reference 35.6.2 of 3.0 IMS 14
5.2 Reference 35.6.3 14
5.3 Reference 35.6.4 14
5.4 Reference 35.5.5.2 14
5.5 Reference 35.8.3.9 15
5.6 Reference 35.8.3.10 17
5.7 Reference 35.9.3.8 Group No. 9 - AINLOC 19

6.0 RE-ENTRANT FORTRAN PACKAGE 44

6.1 FORTRAN Package 44
6.2 Table of Modifications by Routine 45

SECTION B - NEW ROUTINES FOR 65K MSOS

1.0.0 PART 1 - RELOCATING LOADER 49

1.0.1 Storage of the Loader 49
1.0.2 Loading of the Loader 49
1.0.3 Operation of the Loader 49
1.0.4 Types of Loader Operations 50
1.0.5 Relocatable Binary Load Operation 50
1.0.6 Load-and-Go Input 51
1.0.7 Set Program Entry Points 51
1.0.8 Patch Program Entry Points 51
1.0.9 Patch to PART 1 Core Resident Entry Points {CREPL} 52

1.1.0 BUILD TABLE OF PART 1 CORE RESIDENT ENTRY POINTS 52

1.1.1 Patch to Core Resident 52
1.1.2 Memory Map Operation 52
1.1.3 Print Unpatched Externals 53
1.1.4 Available Core for Pages 53
1.1.5 Input to the Loader 53
1.1.6 Exit from the Loader 54
1.1.7 Operation of Loader 55
1.1.8 Loader Table 56

		PAGE NO.
1.2.0	LOAD1 ROUTINE - INITIALIZATION	59
1.2.1	CONBAS	59
1.2.2	Constant Table	60
1.2.3	Accessing Locations in the Constant Table	60
1.2.4	Table of Contents of Constant Table	61
1.2.5	Exit from LOAD1	66
1.3.0	BRNCH1	67
1.3.1	Constant Table Storage Referenced by BRNCH1	67
1.3.2	The Usage of Constants Storage	68
1.3.3	Communication Region Constants Used by BRNCH1	70
1.3.4	Entrance to BRNCH1	70
1.3.5	RDLOAD - Relocatable Binary Loading	71
1.3.6	Exit from RBLOAD	72
1.3.7	MAPS	72
1.3.8	Exit from MAPS	73
1.3.9	Subroutines Used by and External to BRNCH1	73
1.3.10	I/O for Loader Operations	77
1.4.0	LIDRV1	79
1.4.1	Constant Table Storage Referenced by LIDRV1	79
1.4.2	Communication Region Constants Used by LIDRV1	79
1.4.3	Entry to LIDRV1	79
1.4.4	LIDRV1 I/O Request	80
1.4.5	Exit from LIDRV1	81
1.5.0	LCDRV1	82
1.5.1	Entry to LCDRV1	82
1.5.2	LCDRV1 I/O Request	82
1.5.3	Exit from LCDRV1	83
1.6.0	LMDRV1	84
1.6.1	Name Used in Documentation and Storage Position in Constant Table	84
1.6.2	Communication Region Constants Used by the LMDRV1 Routine	84
1.6.3	Entry to LMDRV1	84
1.6.4	LMDRV1 I/O Requests	85
1.6.5	Exit from LMDRV1	85

		PAGE NO.
1.7.0	LLDRV1	86
1.7.1	Entry to LLDRV1	86
1.7.2	LLDRV1 I/O Requests	86
1.7.3	Exit from LLDRV1	87
1.7.4	Subroutines Accessed Via Constant Table	87
1.8.0	ADJOF1	88
1.8.1	Constant Table Storage Referenced by ADJOF1	88
1.8.2	Entry to ADJOF1	88
1.8.3	Address Arithmetic	88
1.8.4	Address Arithmetic for Positive Address Relocation	89
1.8.5	Exit from ADJOF1	89
1.9.0	CNVRT1	90
1.9.1	Constant Table Storage Referenced by CNVRT1	90
1.9.2	Entry to CNVRT1	90
1.9.3	CNVRT1 Operation	90
1.9.4	Exit from CNVRT1	91
1.9.5	Subroutines Used by and Internal to CNVRT1	91
1.10.0	LSTOT1	92
1.10.1	PRINT3	92
1.10.2	PRINT2	93
1.10.3	PRINT4	94
1.10.4	PRINT5	95
1.10.5	PRINT6	97
1.10.6	STOP Routine	97
1.11.0	LINK11	98
1.11.1	Constant Table Storage Referenced by LINK11	98
1.11.2	Constant Table Storage is Used as Follows	98
1.11.3	Communication Region Constants Used by LINK11	99
1.11.4	Entrance to LINK11	99
1.11.5	Patching	99
1.11.6	Internal Subroutines Used by LINK11	100
1.11.7	Exit from LINK11	100
1.12.0	LOADR1	101
1.12.1	Constant Table Storage Referenced by the LOADR1 Routine	101

			PAGE NO.
	1.12.2	The Constants are Used as Follows	101
	1.12.3	Communication Region Constants Used by LOADR1	102
	1.12.4	Entry to the LOADR1 Routine	102
	1.12.5	Reading Input Records	102
	1.12.6	Reading Relocatable Binary Input Blocks	103
	1.12.7	Redaing ASCII Records from the Input Device	104
	1.12.8	Types of ASCII Records	105
	1.12.9	Branching to Process an Input Block	105
	1.12.10	SW1A	105
	1.12.11	SW1B	105
	1.12.12	SW1C	106
	1.12.13	SW1F	106
	1.12.14	Exit from the LOADR1 Routine	106
	1.12.15	Subroutines Used by and External to the LOADR1 Routine	106
1.13.0	NAMPR1		108
	1.13.1	Constant Table Storage Referenced by the NAMPR1 Routine	108
	1.13.2	The Constants are Used as Follows	108
	1.13.3	Entry to the NAMPR1 Routine	110
	1.13.4	Processing a NAM Block	111
	1.13.5	Reserving Common Storage	111
	1.13.6	Reserving Data Storage	112
	1.13.7	Reserving Command Sequence Storage NAMPR2	113
	1.13.8	NAMPR3 - Print Program Name and Execution Time Relocation Base	114
	1.13.9	OVFTST - Test for Core Overflow Condition	115
	1.13.10	Exit From NAMPR1	115
	1.13.11	Subroutines Used by and External to NAMPR1	115
	1.13.12	Subroutines Used by and Internal to NAMPR1	115
1.14.0	RBDBZ1		116
	1.14.1	Constant Table Storage Referenced by RBDBZ1	116
	1.14.2	The Constants are Used in the Following Way	116
	1.14.3	Entry to RBDBZ1	118
	1.14.4	RBDPR0	119
	1.14.5	Initialization for RBD Block Processing at RBDPR0	119
	1.14.6	Starting Address for Command Sequence Storage	119

		PAGE NO.
1.14.7	RBDPR1 - Command Sequence Words for RBD Blocks	120
1.14.8	BZSPR0	122
1.14.9	Initialization for BZS Block Processing	122
1.14.10	BZS Entries	123
1.14.11	Relocation for Starting Address	123
1.14.12	Zero Storage in BZS Block	124
1.14.13	Exit from RBDBZ1	125
1.14.14	Subroutines Used by and Internal to RBDPR0 and BZSPR0	125
1.14.15	NXTWRD	125
1.14.16	ADJUST	126
1.14.17	STRLNK	128
1.14.18	LINK TABLE	128
1.14.19	Subroutines Used by and External to RBDPR0 and BZSPR0	129
1.15.0 XFRPR1		130
1.15.1	Constant Table Storage Referenced by XFRPR1	130
1.15.2	The Constants are Used as Follows	130
1.15.3	Entry to XFRPR1	130
1.15.4	Recording Transfer Name	131
1.15.5	Recording Relocation Base for Next Program	131
1.15.6	Exit from XFRPR1	131
1.16.0 ENTEX1		133
1.16.1	Constant Table Storage Referenced by ENTEX1	133
1.16.2	The Constants are Used in the Following Way	133
1.16.3	Entry to ENTEX1	134
1.16.4	ENTPR0	134
1.16.5	ENTPR1 - ENT Block Entries	134
1.16.6	Searching the Loader Table for Matching Name	135
1.16.7	Matching Name is Entry Point	135
1.16.8	EXTPR0	135
1.16.9	EXTPR1 - EXT Block Entries	135
1.16.10	Exit from ENTEXT	136
1.16.11	Subroutines Used by and Internal to ENTPR0 and EXTPR0	136
1.16.12	NXTNAM	136
1.16.13	Subroutines Used by and External to ENTPR0 and EXTPR0	137

			PAGE NO.
1.17.0	STBASE		138
	1.17.1	Names Used in Documentation and Storage Position in Constant Table	138
	1.17.2	The Constants are Used as Follows	139
	1.17.3	Entry to STBASE	140
	1.17.4	Constants Set by STBASE are Set in the Following Manner	140
	1.17.5	Exit from STBASE	142
1.18.0	LNKENT		143
	1.18.1	Names Used in Documentation and Storage Position in Constant Table	143
	1.18.2	The Constants are Used in the Following Way	143
	1.18.3	Exit From LNKENT	144
1.19.0	LNKCRL		145
	1.19.1	Names Used in Documentation and Storage Position in Constant Table	145
	1.19.2	The Constants are Used in the Following Way	145
	1.19.3	External Subprograms Used by LNKCRL	146
	1.19.4	Internal Subroutines Used by LNKCRL	146
	1.19.5	Exit from LNKCRL	146
1.20.0	BLDCRL		147
	1.20.1	Names Used in Documentation and Storage Position in Constant Table	147
	1.20.2	Constant Storage is Used as Follows	147
	1.20.3	Entry to BLDCRL	148
	1.20.4	External Subprograms Used by BLDCRL	148
	1.20.5	Subroutines Used Internal to BLDCRL	149
	1.20.6	Exit Procedure	149
1.21.0	PATCH		150
	1.21.1	Names Used in Documentation and Storage Position in Constant Table	150
	1.21.2	Communications Region Used	151
	1.21.3	Usage of Constant Table is as Follows	151
	1.21.4	Subprograms Used by PATCH	151
	1.21.5	PATCH is Entered	152

		PAGE NO.
1.22.0	TBSCH1	153
1.22.1	Constant Table Storage Referenced by TBSCH1	153
1.22.2	Constant Storage Usage	153
1.22.3	Communication Region Constants Used by TBSCH1	153
1.22.4	Entry to TBSCH1	154
1.22.5	Searching Operation	154
1.22.6	Exit from TBSCH1	155
1.22.7	Subprograms External to TABSCH	155
1.23.0	HASH	156
1.23.1	Entry Point Name: HASH	158
1.23.2	Constant Table Storage Referenced by HASH	158
1.23.3	Communications Region Used	158
1.23.4	Constant Storage Usage	158
1.23.5	Exit from HASH	158
1.24.0	TBSTR1	159
1.24.1	Constant Table Storage Referenced by TBSTR1	159
1.24.2	Constant Table Storage is Used as Follows	159
1.24.3	Entry to TBSTR1	160
1.24.4	Table Store Operation	160
1.24.5	Internal Subroutines Used by TBSTR1	161
1.24.6	Exit from TBSTR1	162
1.25.0	PAGE	163
1.25.1	Calling Sequences	163
1.25.2	Entry Points	163
1.25.3	PAGE in Core	163
1.25.4	MDRIV	164
1.25.5	Names Used in Documentation and Storage Position in Constant Table	164
1.25.6	Communications Region Used	164
1.25.7	Constant Storage Usage	164
1.25.8	Entry to PAGE	165
1.25.9	Exit from this Routine	165
1.25.10	Other Constants and Flags Used by the PAGE Program	165
1.25.11	Possible Errors in PAGE	166
1.26.0	PART 1 Loader Flowchart Index	167

		PAGE NO.
2.0	PARTITIONED CORE ALLOCATOR - PRTCDR	251
2.1	GENERAL DESCRIPTION	251
2.2	ENTRY POINTS	251
2.3	EXTERNALS	251
2.4	DETAILED DESCRIPTION	252
2.4.1	Partition Allocation Request Entry	252
2.4.2	Partition Allocation Section	253
2.4.3	Partition Release Request Entry	253
2.4.4	Request Parameter Lists	253
2.4.5	Volatile Contents	254
3.0	PART 1 INDIRECT REQUEST PROCESSOR - T1B	260
3.1	GENERAL DESCRIPTION	260
3.2	ENTRY POINT	260
3.3	EXTERNALS	260
3.4	DETAIL DESCRIPTION	260

SECTION A

Modifications to Existing Routines

1.0 Monitor Changes

1.1 Routine - LOCORE

Modification - Locations 47_{16} through 50_{16}.

Reason - Locations 47_{16} through 50_{16} are no longer available to the user. More LOCORE locations were needed for the Part 1 Re-entrant FORTRAN math subroutines.

1.2 Routine - SYSBUF

Modifications - 1. Volatile Storage Block
2. MMDIAG Routine modified to release Part 1 core when a mass memory error occurs.
3. Partitioned Core tables

Reasons - 1. The volatile storage block was increased by 2 words per priority level because MONI requests 9 words instead of 8. The 9th word is used to indicate direct or indirect request. It was also increased by 10 words per FORTRAN priority level.

2. If Part 1 core is not released when a Mass Memory error occurs it could end up never being released. The routine now checks to see if allocatable core or Part 1 core is reserved and calls the appropriate routine to release the reserved core.

3. The Partitioned Core tables have been put in SYSBUF for use by the Partitioned Core Allocator and LIBEDT.

1.3 Routine - NCMPRQ

Modifications - 1. Mask for Request Code
2. 16 Bit Address

Reasons - 1. The request code will occupy bits 9-13 instead of bits 9-14. The mask for the request code was modified to account for this.

2. Because request could originate in Part 1 it was necessary to allow a 16 bit address and to make an indirect request with a 16 bit address.

1.4 Routine - NFNR

Modification - 1. Mask for Request Code
2. 16 Bit Address

Reason - 1. The routine was modified to mask for the request code as bits 9-13 instead of 9-14. Bit 14 now indicates a Part 1 request.

2. To account for 16 bit addresses the routine checks for Part 1 type requests. If Part 1 requests the starting address and number of words are obtained unmodified from the request. Part 0 requests are processed as before.

1.5 Routine - ADEV

Modifications - 16 Bit Address

Reason - Because request could have errors and go to ADEV for operator intervention the routine was modified to handle the 16 bit addresses possible. The Part 1 indirect request was used to accomplish this.

1.6 Routine - RDISP

Modifications - 1. Overflow condition with priority level
2. 16 Bit Address
3. 9th Word of Volatile
4. Part 1 Core Allocator
5. Mask for Request Code
6. Part 1 Directory schedule request
7. Additional re-entrant FORTRAN locations to to save

Reasons - 1. Because the interrupt trap must contain a 16 bit address it was necessary to save the overflow condition along with the priority level in the interrupt stack. This was accounted for when checking the priority of the interrupt stack against the priority of the scheduler stack. The overflow condition was restored before doing an EXI when the highest priority program was on the interrupt stack.

2. The routine was set up to handle 16 bit addressing throughout. This will allow request from the upper bank to be processed.

3. The 9th word of Volatile on a schedule request is tested to determine if the request was indirect or not. This word is set by MONI.

4. If the request was a system directory request to Part 1 a jump is made to the Part 1 core allocator. Bit 14 of the request word is used to determine which core allocator to use.

5. Whenever the request code is tested the mask is set to test bits 9-13 instead of 9-14. The 14th bit now indicates which type addressing to use, 15 bit or 16 bit.

6. Part 1 16 bit relocating loader is unable to load the 1st parameter of a normal directory schedule request so request code 18 has been set aside for scheduling mass memory resident programs from Part 1.

7. In order to remove all indirect parameters from calls to the floating point arithmetic package within re-entrant FORTRAN, it was necessary to add a few more locations to those saved by the scheduler for FORTRAN levels.

1.7 Routine - DRCORE

Modifications - 1. Eliminated Multi-Level Indirect Addressing
2. 9th Word of Volatile
3. Part 1 Release Request
4. Address to be released in RELEAS request is picked up from location 6 of Volatile.

Reasons - 1. Because multi-level indirect addressing will not be allowed in the 65K 1700 it was necessary to change that coding which used it. The coding was modified in the area where request priorities are set and where the wait thread is created.

2. The 9th word of volatile is now tested to determine if a release request is indirect.

3. If a release request has bit 14 set to indicate a Part 1 type request this routine now goes to the Part 1 release processor to release Part 1 partitions.

4. Before enabling interrupts MONI stores word 1 of a request parameter list in location 6 of Volatile. Picking up that parameter rather than going to the request parameter list decreases the chance of a user damaging the system by re-entering code which contains a release request.

1.8 Routine - PARAME

Modification - Guarrantee proper masking of result of 15 bit arithmetic.

Reason - Because of hardware wrap around some incorrectly masked addresses were never noticed on the 32K masking, but in 65K they put parameters in the wrong bank.

1.9 Routine - COMMON

Modification - Save overflow condition with priority level

Reason - Because the 65K 1700 will require all 16 bits in the interrupt trap address it is necessary to save the overflow condition under software control. The priority level word of the interrupt stack was selected for the saving of the condition. Bit 15 of the priority level in the interrupt stack will now indicate the overflow condition, 1 overflow, 0 no-overflow.

1.10 Routine - NIPROC

Modification - Save and restore overflow condition if power fail interrupt

Reason - The overflow must be saved on the 65K 1700 because the interrupt trap requires a 16 bit address word. The overflow condition must be saved when the Power Fail Interrupt occurs. The condition of the overflow must then be restored before doing an EXI after power has come back up.

1.11 Routine - NMONI

Modifications - 1. 9 Words of Volatile
2. Request Code
3. Word 6 of Volatile

Reasons - 1. Since it is now possible to have 16 bit address the indirect condition cannot be saved as bit 15 of word 0 of the request. Nine words of Volatile are now requested and the 9th word is used to indicate direct or indirect, Set-indirect, clear-direct.

2. The request code is now contained as bits 9-13 instead of 9-14 as before. Bit 14 is now used to indicate a Part 1 type request. Previous request are now Part 0 type and are unaffected by the 65K modifications.

3. In order to make the Part 1 indirect request re-entrant the first parameter of the request after the request code is saved in word 6 of Volatile before interrupts are enabled.

1.12 Routine - RW

Modifications - 1. 9th Word of Volatile
2. Part 1 type request

Reasons - 1. The 9th word of volatile is now tested to determine if the request was direct or indirect. This word is set by MONI.

2. Part 1 type requests do not use the x indicator and are never indirect, therefore, if Bit 14 is set and the request is a mass memory request the mass storage address immediately follows the request.

Part 1 type request also may have 16 bit request addresses therefore the schedule of the completion address must be a Part 1 type indirect.

1.13 Routine - SPACE

Modifications - 1. 9th word of Volatile
2. Part 1 Core request
3. Limits of initial load of PROTEC
4. JMP to FFFF rather than 7FFF in words 0 & 1 16 16
5. Set mode switch
6. Read in Part 1 core resident on autoload.

Reasons - 1. The indirect/direct condition is saved as the 9th word of volatile therefore that word must be tested to determine if direct or indirect.

2. Because the allocation of Part 1 core is different than allocatable core it is necessary to test the request code to determine if the request is a Part 1 Core Request (T17). If the request is for Part 1 Core a jump is made to the new core allocator.

3. PROTEC was lengthened and so the area initially read into core needed to be expanded.

4. If jump to zero occurs and DEBUG is not patched it will jump to $FFFF_{16}$ rather than $7FFF_{16}$.

5. A location with entry point MODB5K is set to 1 if mode is 65K and 0 if mode is 32K.

6. When an autoload occurs it is necessary to read in any core resident routines into Part 1 as well.

2.0 Drivers

2.1 Routine - DR1728

Modification - Mask For Request Code

Reason - The mask for the request code was changed to be bits 9-13 instead of bits 9-14. Bit 14 currently indicates a Part 1 type request.

2.2 Routine - DISKWD

Modification - 1. 16 Bit Addresses
2. Part 1 Type Requests

Reasons - 1. When checking to determine if FWA is greater than LWA the previous method would not have sufficed because of 16 bit addresses.

2. When the disk driver absolutizes the completion addresses and the starting, a check was added to account for Part 1 type requests. Part 1 type requests use the completion address as a 16 bit address as contained in the request. The starting address is always the 6th word of the request and the disk address immediately follows the request.

2.3 Routine - DR1732

Modification - Mask for Request Code

Reason - Because the request code is now bits 9-13 the mask was changed to account for this.

2.4 Routine - TAPE

Modification - 9th word of Volatile

Reason - Since indirect request is now indicated by word 8 of Volatile the T14 request processor has been altered to check this word of Volatile.

2.5 Routine - DBLDRV

Modification - 16 Bit Addressing

Reason - Bit 15 of the address of a driver's physical device table is used as a flag. Since this address will always be in bank 0 this is acceptable but code had to be added to mask the bit off when the address is used.

3.0 Job Processor - 65K

Job Processor Modules

JOBENT
T11
T7
T5
T3
JOBPRO
PROTEC
JBKILL
JPLOAD
JPCHGE
ASCHEX
T13
MIPRO
RESTOR

All modifications were made for the following core configuration:

1. The Job Processor will operate in BANK 0 in protected core

2. Unprotected core will reside wholly within BANK 0

All modifications are based on MSOS 3.0 released version.

3.1 JOBENT

There are no changes for indirect addressing to convert this program for the 65K 1700.

3.2 T11

No changes for indirect addressing to convert this program for the 65K 1700.

This program was modified to use word 8 of volatile for the indirect check {instead of the A register}.

Modification was also made to prohibit the locations passed in either A or Q from referencing the upper bank.

3.3 T7

There was a change to remove indirect addressing for the 65K 1700.

This change was made at COSY Card numbers 45, 52, 53, 58, 169, 170.

This program was modified to use word 8 of volatile for the indirect check (instead of the A register). This change was made at COSY Card number 48.

3.4 T5

The mode flag is tested and if mode is 65K return is not masked to 15 bits. COSY Card numbers 29, 48 and 55 were involved.

3.5 T3

This program was modified to use word 8 of volatile for the indirect check (instead of the A register).

This change was made at COSY Card number 42.

3.6 JOBPRO

The following list of COSY Card numbers indicate the locations where changes were necessary to remove indirect addressing for the 65K 1700.

COSY Card Numbers

62
64
65, 66
68
129
154
165
168
248
251
273

COSY Card Numbers (continued)

279
288
309
316
327, 328
483, 484

3.7 PROTEC

Modifications - 1. Size of JBKILL overlay
2. Save overflow on priority level
3. Remove multilevel indirect
4. Allow Part 1 requests
5. Issue diagnostic if D and X bits both set or bit 15 on C, S or N when D bit is set.

Reasons - 1. JBKILL has been expanded therefore more room is necessary for it.

2. Mode must always be checked when return and priority level are saved and restored to see whether overflow is to be saved on return address or on priority level.

3. Any two word absolute indirect addressing had to be removed.

4. Requests with the D bit set and the Part 1 indirect request must be allowed from unprotected core.

5. As long as Part 1 requests are allowed they must be legal.

3.8 JBKILL

There are several changes to this module to remove indirect addressing for the 65K 1700.

These changes are at COSY Card numbers 138, 139 and 142.

Mode had to be tested to see if overflow was saved on return or on priority level.

These changes are at COSY Cards 46, 89, 90 and 99.

3.9 JPLOAD

To remove indirect addressing for the 65K 1700 modifications were made at the following COSY Cards.

COSY Card Numbers

67
68
179, 180
255, 256
343, 344

3.10 JPCHGE

Modifications to remove indirect addressing for the 65K 1700 were made at COSY Card numbers 74, 75 in this module.

3.11 ASCHEX

There were no changes necessary to this module to operate it in the 65K 1700.

3.12 T13

Modifications were made at:

COSY Card number 26 and 44 for the program to check word 8 of volatile rather than the A register for the indirect indicator.

COSY Card numbers 307, 308 and 331 to remove indirect addressing.

COSY Card numbers 112, 115 to correctly mask 15 bit address.

3.13 MIPRO

There were no changes necessary in this program for operation in the 65K 1700. However, entry to the system test package was put in.

3.14 RESTOR

Modifications were necessary in this program to remove indirect addressing for the 65K 1700.
They were made at COSY Card numbers 89 and 90.

4.0 Debugging Aid Routines

4.1 ODEBUG

Modifications were necessary to the following:

LHX - It was necessary to modify the calculation of relative entries, i. e. entries with ¤ following the /.

DPC - The test for end location greater than beginning location was altered to take 16 bit address correctly.

SCN - The test for search finished had to be altered to take 16 bit address correctly.

SET - The test for finished had to be altered to take 16 bit address correctly.

MBC - The check to see which end of block to start with and the test for finished both must take 16 bit addresses into account.

SPE - Starts with $FFFF_{16}$ rather than $7FFF_{16}$.

CPP and
SPP - The test for done must handle 16 bit address compare correctly.

4.2 Module - RDMPCR module of Recovery Package

The legality check for beginning location before ending location was changed to accomodate 16 bit address calculation.

5.0 LIBEDT

Modification - Two new modules were added to process the loading of core-resident and mass storage resident routines in Part 1. Reference section 35, additions occur as indicated by section numbers above each addition.

5.1 Reference 35.6.2 of 3.0 IMS

GROUP NO.	SUBR M.S. ADDRESS	RELOC FACTOR	SUBROUTINE INCREMENT	SUBROUTINE NAME
9	AINRBL-LBDT	AINLOC	0	AINRBL
			AINLOD	LODR
			MOVMM	MVMM
10	AINS46-LBDT	ST4@6B	0	AINS46

5.2 Reference 35.6.3

CONTROL STATEMENT	INTERPRETATION CODE	ENTRY POINT NAME
¤A	13	AINSN
¤Y	14	YINSN

5.3 Reference 35.6.4

L19 No DATBAS entry point in system. Error occurs when user requests use of system data and there is no system data defined.

L20 Irrecoverable error from the loader.

L21 Attempt to load upper bank core resident more than once after a system installation.

5.4 Reference 35.6.5.2

IDWC, NOWLR, REMOV, REMFIL, UPDATE, and POINT are used in the ¤A and ¤Y statements. They are equated to labels.

PARAD = PABA = REMOV

PAND = REMFIL

DBASE = UPDATE

PACOM = POINT

IOWC = CPLAD

NOWLR = CPLLN

5.5 Reference 35.8.3.9

System Library Update for Part 1 Programs - AINSN

The AINSN program loads and absolutizes mass resident programs which will run in partitioned core. The statement which calls this program is

¤A,ORD,S,N,D,C,L,M

The parameter ORD is validated to be numeric. All other parameters must be alpha. Statement format failure causes the error L06 to appear on the output comment medium and the statement processing to be aborted.

If the ordinal is numeric the index into the System Directory is calculated and saved.

Next the numbers of the partitions requested are checked to insure that they are not out of range. The base address of the starting partition and the last word address plus one of the end partition are set up for the loader.

If the D parameter is not set a zero is set in for the data base of system data. If the D parameter is set the Core resident entry point table is searched for the entry point DATBAS, which is the base of system data.

If DATBAS is not found the error L19 is given and the ¤A processor is exited.

If DATBAS is found its address is set up for the loader. Next the C parameter is checked. If no C parameter is present a zero is set for the base of Common.

If the C parameter is present and there is no system Common the error L13 is given and the ¤A processor is exited.

If the C parameter is present and there is system common the base of system common is set up for the loader.

An indicator is set for the L parameter.

The setbases function is sent to the loader.

Then the loading of the program starts.

First a relocatable load function is sent to the loader.

This functions loads binaries from standard input until the termination signal is found.

Then the program checks the L indicator. If set it requests the operator to indicate the new input media. The user responds with the new input media, and loading continues until the user replies with the terminate statement.

When the user gives the terminate signal, or if there is no L parameter the program proceeds:

first among the programs,

if no unpatched externals it goes to print memory map else it links to the CREPL table

if no unpatched externals it goes to print memory map else it links to the CREP table

if no unpatched externals it goes to print memory map else it prints the unpatched externals {the loader after it is through linking checks for unpatched externals if it finds any it interrogates the user whether to continue see section 1 for complete description}.

The memory map is printed if the M parameter was found. Else it continues.

The loading is completed and the ×A processor proceeds to get the loaded program into its location on mass storage {it is loaded onto scratch mass storage} and modify the system to reflect this new information.

First the entry for the program in the System Directory is set so that no one can use it while it is being modified.

Next the loaded program is moved into its proper location on mass storage. If it is smaller than the program currently in

the entry it is moved into the location of the program currently in the entry. If it is larger, it is moved to the defined scratch area and the beginning of the scratch area is redefined. The system directory is updated. The completion address is the beginning location at which the program was loaded. The D bit is set in the request word. Location on mass storage is corrected. The directory thread is cleared. Then the system directory image is cleared. The sectors where the old program resided are released in the sector availability table. The sectors where the new program resides are made unavailable in the sector availability table.

The ¤A processor is exited.

5.6 Reference 35.8.3.10

Core Resident Part 1 Program Initializer - YINSN

In the 65K system programs which reside permanently in core in the upper part, Part 1, are loaded post initialization using the library editing program via the ¤Y processor.

The ¤Y processor - YINSN - loads and absolutizes the programs which are to be permanently resident in Part 1 of core. The statement which calls this program is

¤Y,S,D,C,L,M

The S parameter must be numeric.

Other parameters are validated to be alpha. Statement format failure causes the error L06 to appear on the output comment medium and the statement processing to be aborted. The program gets the S parameter.

If the S parameter is not present the L06 error occurs and the ¤Y processor is exited, else it must be the last defined partition. If not the L06 error is given and the processing is aborted.

If it is the last defined partition the starting address is set up for the loader. The last word address plus one of the partition is set up for the loader.

Next the program gets the D parameter.

If the D parameter is not set the data base of system data is set to zero. If the D parameter is set the core resident entry point table is searched for the entry point DATBAS which is the base of system data.

If DATBAS is not found the error L19 is given and the processing aborted.

If DATBAS is found its address is set up for the loader.

Next the C parameter is checked. If no C parameter is present the base of system Common is set to zero.

If the C parameter is present and there is no system Common the error L13 is given and the statement processing is aborted.

If the C parameter is present and there is system Common the base of system Common is set up for the loader.

An indicator is set for the L parameter.

The setbases function is sent to the loader.

Then the program loading starts.

First a relocatable load function is sent to the loader. This function loads binaries from standard input until the termination signal is found.

Then the program checks the L indicator. If set it requests the operator to indicate the new input media, and loading continues from the media indicated. This process continues until the user replies with the terminate statement.

When the user gives the terminate signel, or if there is no L parameter the program proceeds. It functions the loader to link entry points first, among the programs

if no unpatched externals it goes to print memory map; else, it links to the CREP1 table

if no unpatched externals it goes to print memory map; else, it links to the CREP table

if no unpatched externals it goes to print memory map; else, it prints the unpatched externals {after the loader is through linking it checks for unpatched externale. If it finds any it interrogates the user whether to continue. See section B-1 for complete description.}

The memory map is printed if the M parameter was found. Else it continues.

Next the CREP1 table is moved to permanent mass storage. If a CREP 1 table is already on mass storage an error 21 is generated. If not the program finds a string of sectors long enough to hold the CREP1 table and moves it to permenent mass storage. It updates the CREP1 pointer {SCREP1} on the Core image and the start of scratch if required.

Then the absolutized program is loaded into core at the partition indicated. This is also written to permanent storage. After it is written to permanent mass storage the autoload program which reads this part of the system into the partition at autoload time is set up to contain the correct data as to location in core, length in words, and location on mass storage. These words are also updated on the Core image on mass storage.

The absolutized program may be written out only once, after a system installation. Subsequent attempts will cause an error 21 to be printed and the program aborted.

5.7 Reference 35.9.3.8 Group No. 9 - AINL0C

AINL0C - Subroutine 9

AINRBE - sets up relocatable load function and calls the loading subroutine.

L0DR - loading subroutine. This routine loads in the loader if not in core and then functions it with the command passed it from the calling program.

On entry to loader A = loader function code on return from loader

A = 0 function completed

= -0 irrecoverable error all cases

Q = negative unpatched externals not negative - no unpatched externals

I = pointer to a 2 word parameter pack

2 word parameter pack

Word 0 address of loaded program
Word 1 length of loaded program

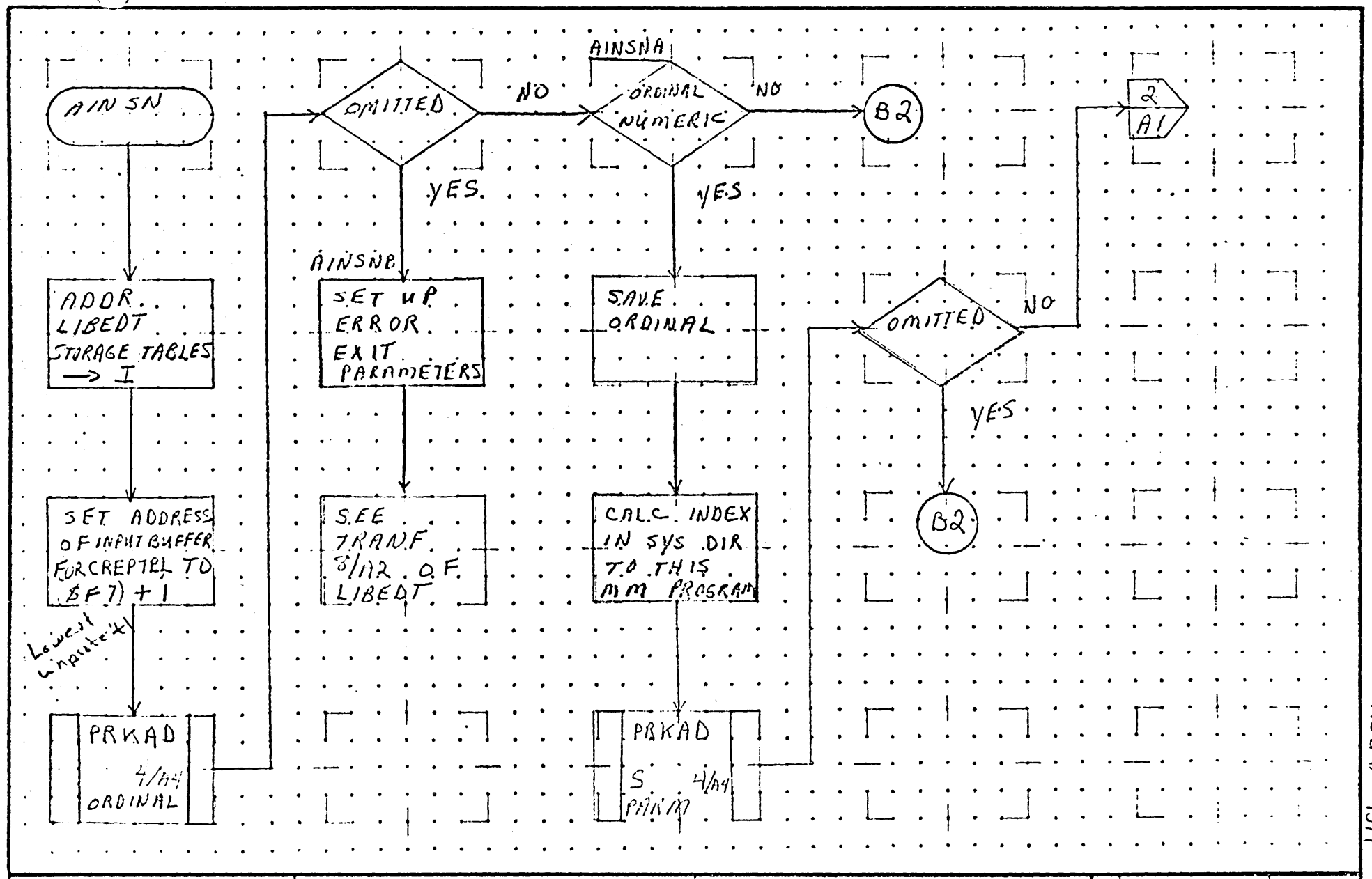
These parameters are used to get the length and location on mass storage of the CREPL table, and of the absolutized program.

On return from completing the load function this routine interrogates the register. If the A register is -0 it terminates the load with an error 20. If not the parameters are saved if the function was load or build CREPL table in LIBEDT Common area, and the Q register is passed back to the calling program.

MOVMM - this subroutine performs a mass storage to mass storage transfer. It uses unprotected core for its buffer and presumes it has the entire unprotected area available for a buffer.

Group No. 10 - ST4Q6B
ST4Q6B Subroutine 10

This subroutine is called by the ¤A processor and is used to update the system directory for the ordinal in question on the core image.



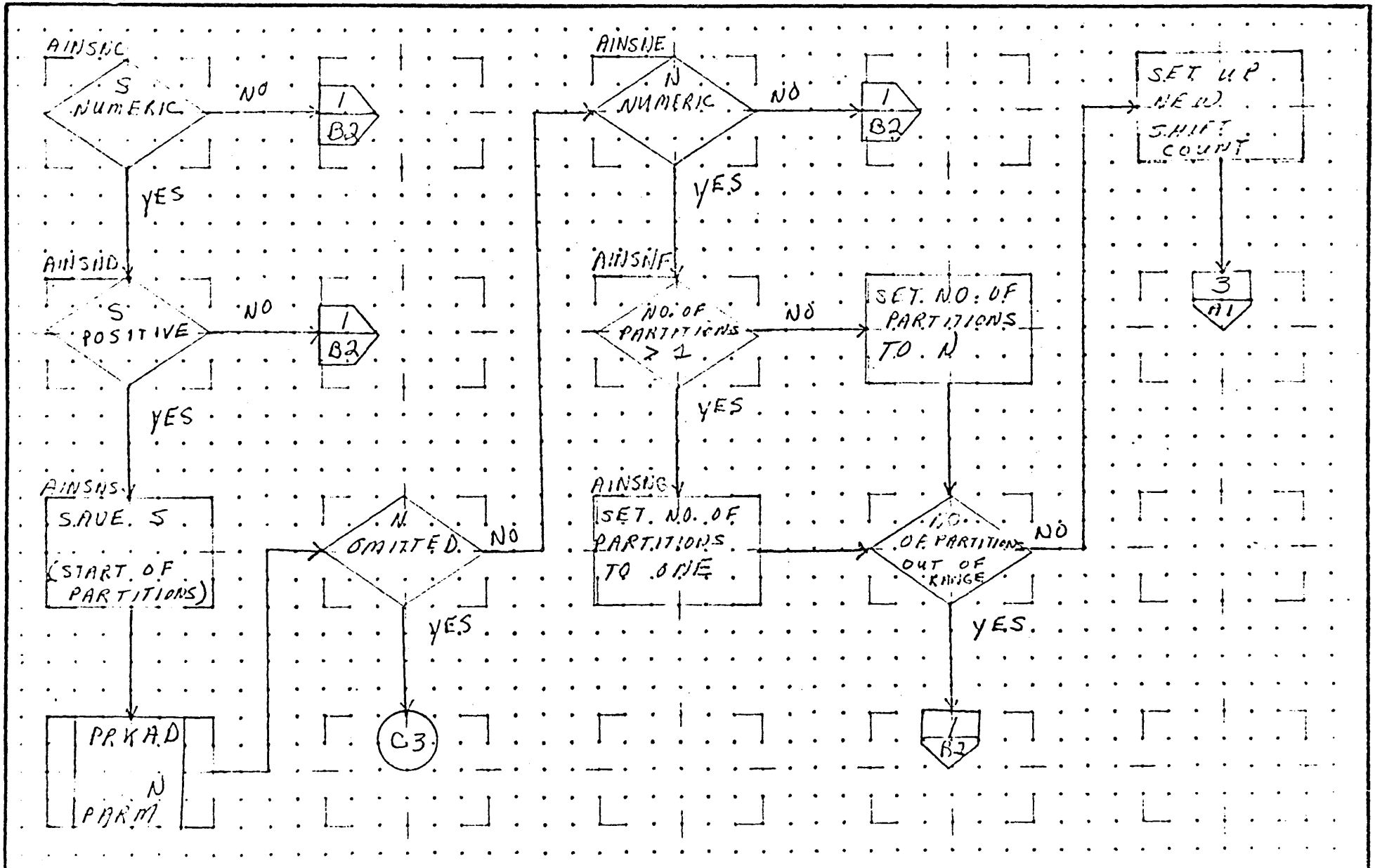
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE LIBEDT		PROJECT MGR.			
FLOWCHART ☒		PAGE 1 OF 23	PROJECT NAME 1700 MSOS			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY C.D.C.	DATE 7 [illegible] 71	TASK NAME LIBEDT			

AUG 9 1971

60021

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

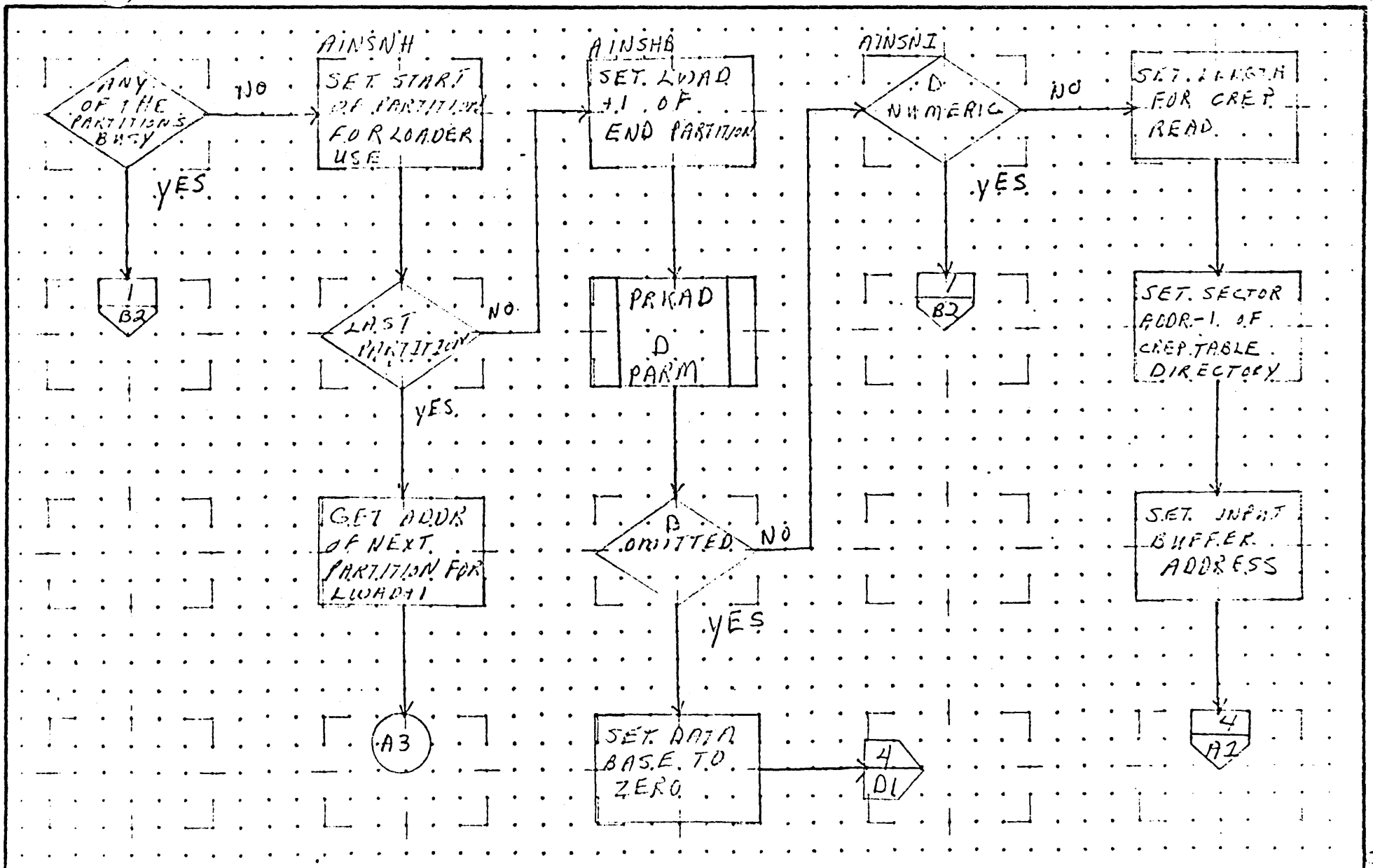


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
	PAGE 2 OF 23	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C.D.C. DATE 7/14/71	TASK NAME LIBEDT			

AUG 9 1971

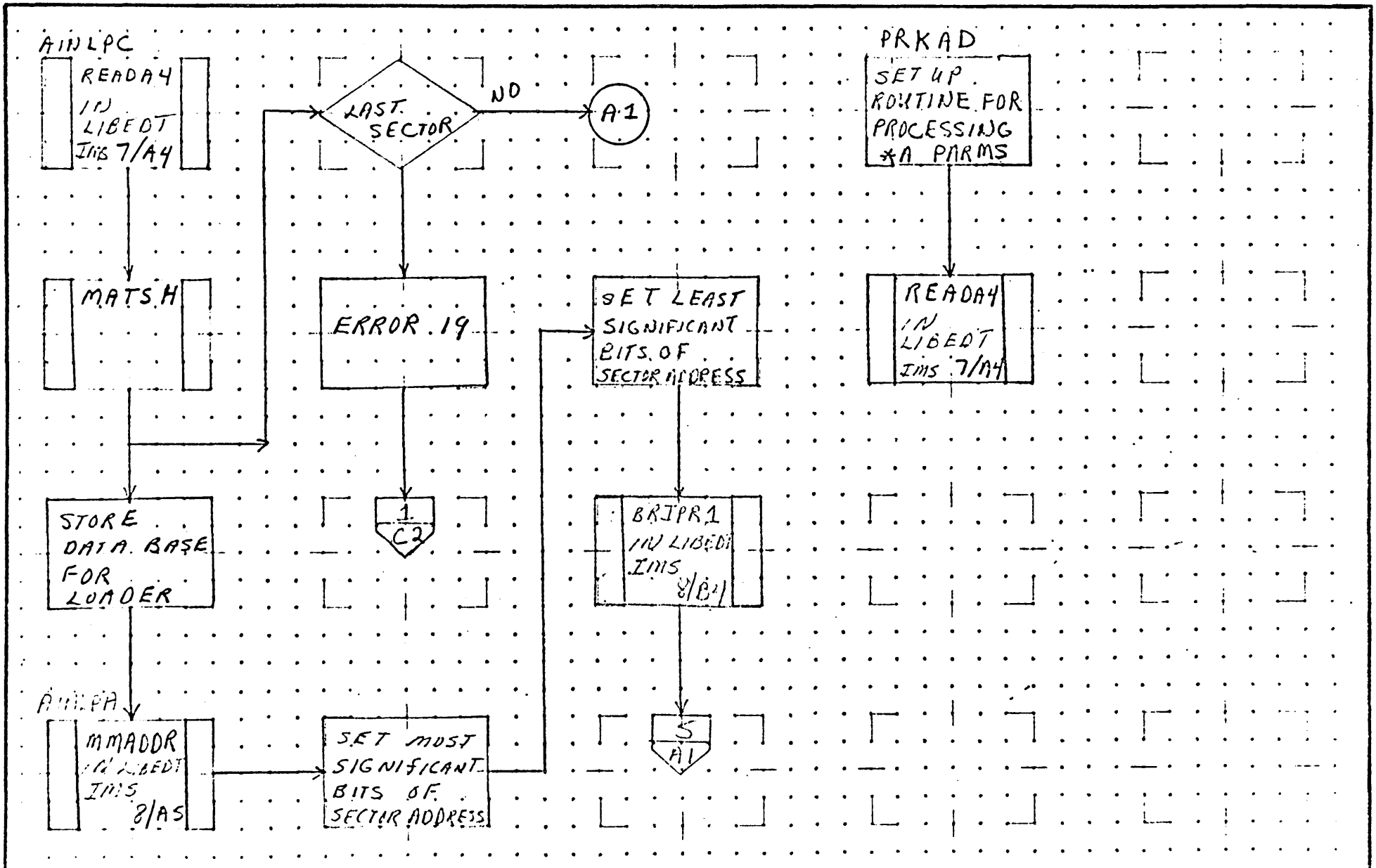
AA1385 (FORMERLY CA127-1)

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 LIBEDT	PROJECT MGR.			
	PAGE 3 OF 23	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY A.D.C. DATE 8/6/71	TASK NAME LIBEDT			

AUG 6 1971

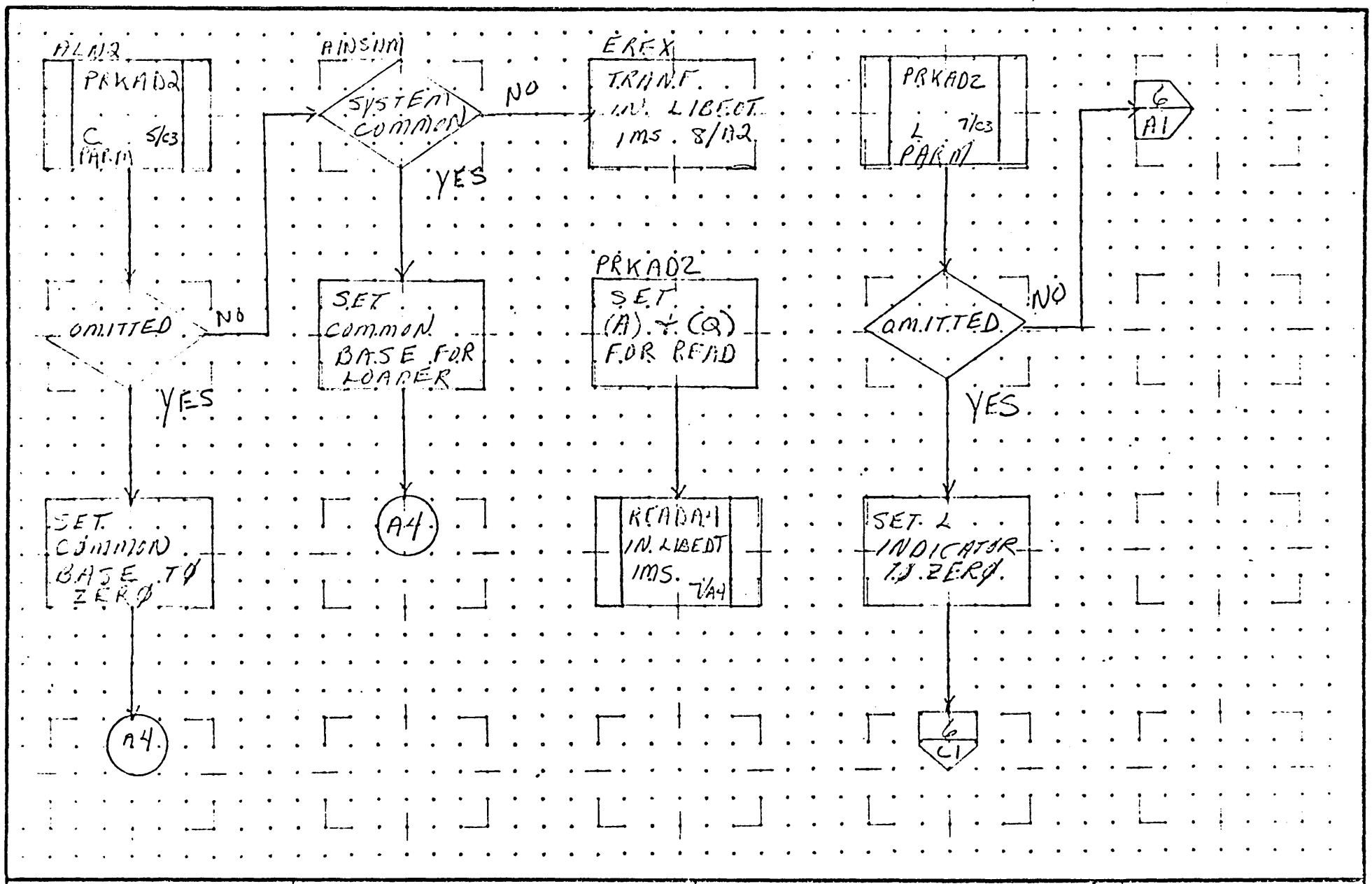


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
	PAGE 4 OF 23	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 7/14/71	TASK NAME LIB11			

AUG 9 1971

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



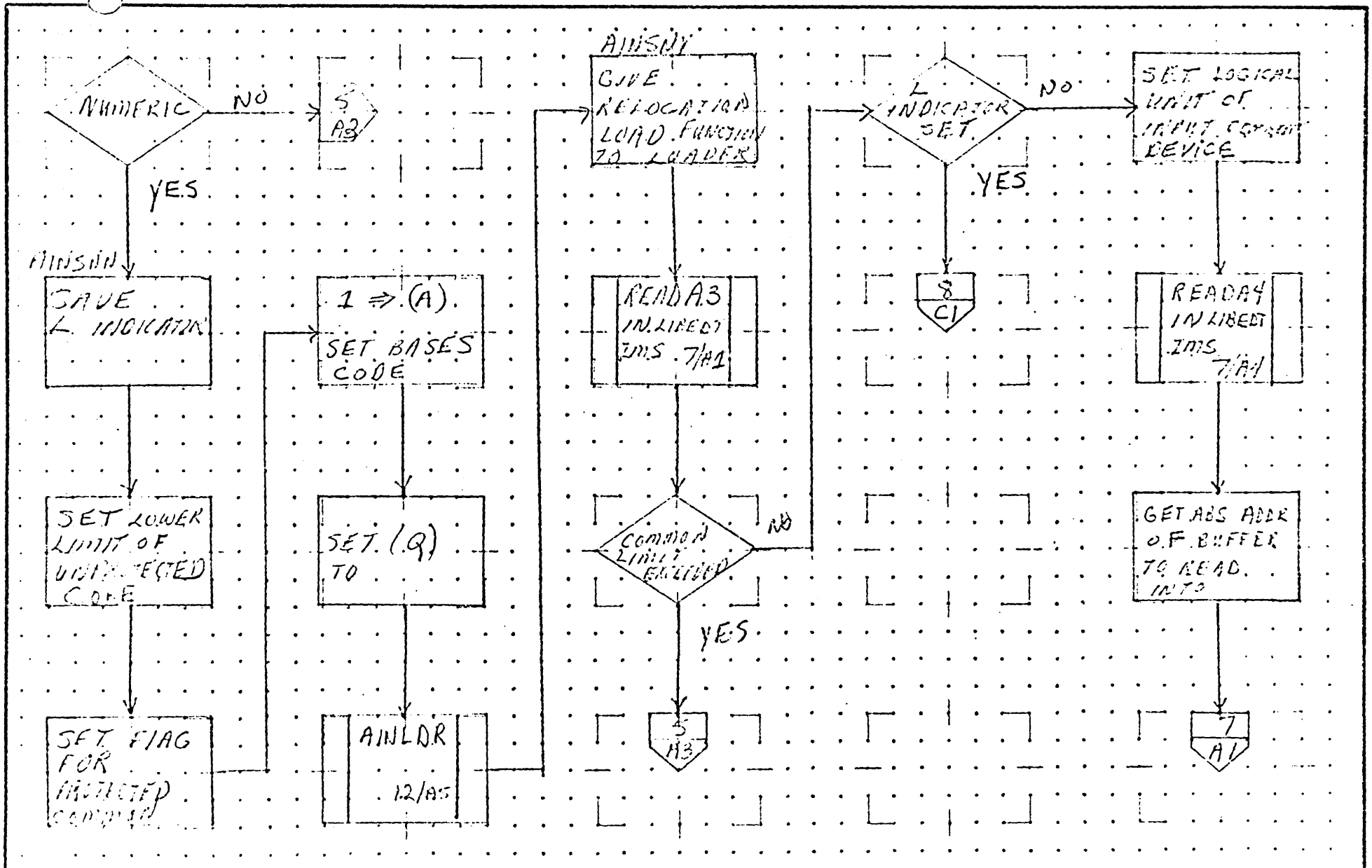
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
FLOWCHART ☒	PAGE 5 OF 22	PROJECT NAME 1700 MSOS			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY CT DATE 7/20/71	TASK NAME LIBEDT			

AUG 9 1971

60000525

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

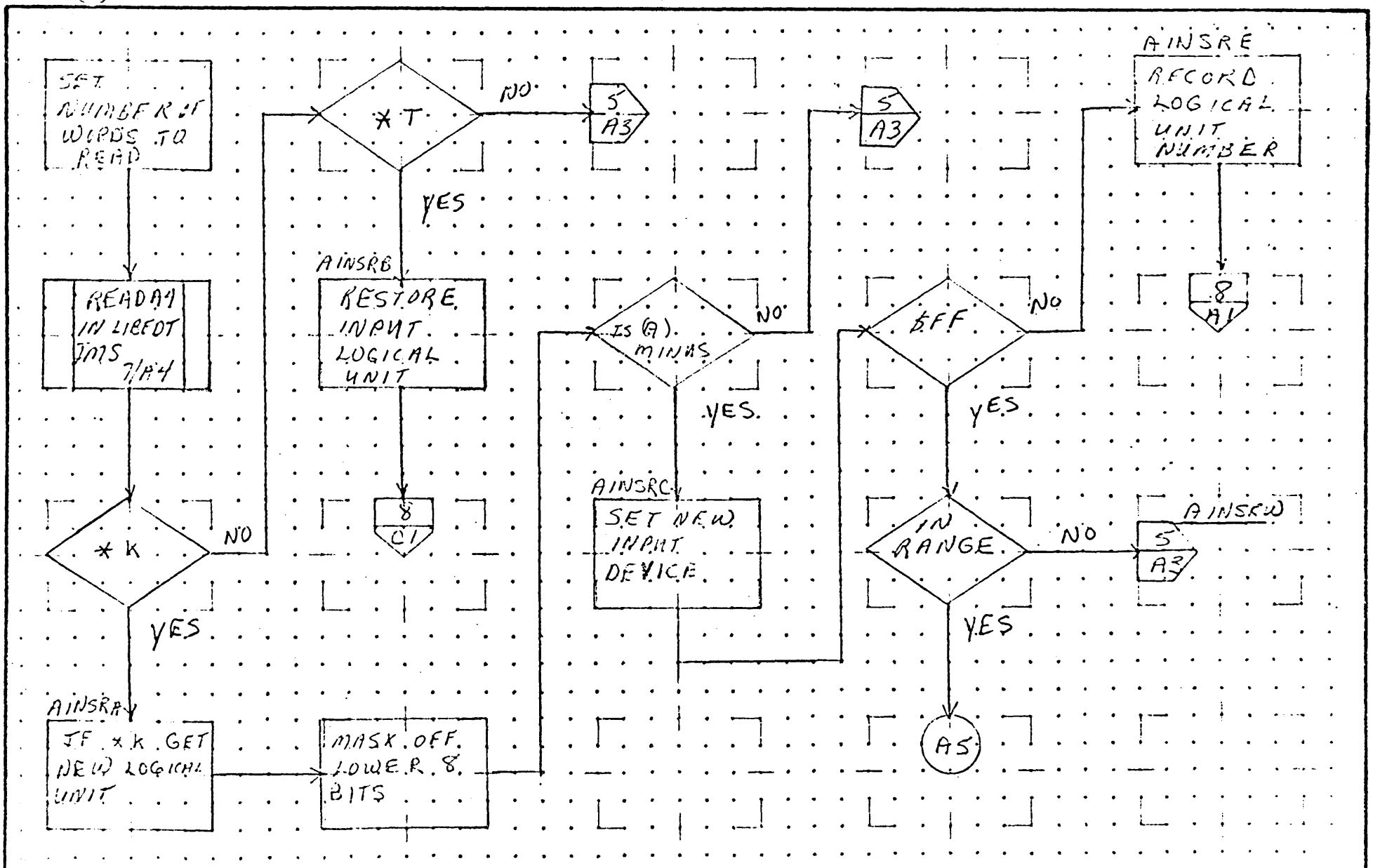


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 17[illegible]	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE [illegible]		PROJECT MGR.			
FLOWCHART ☒		PAGE 10 OF 32	PROJECT NAME [illegible]			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY [illegible]	DATE [illegible]	TASK NAME [illegible]			

AA1385 (FORMERLY CA127-1)

AUG 9 1971

960000



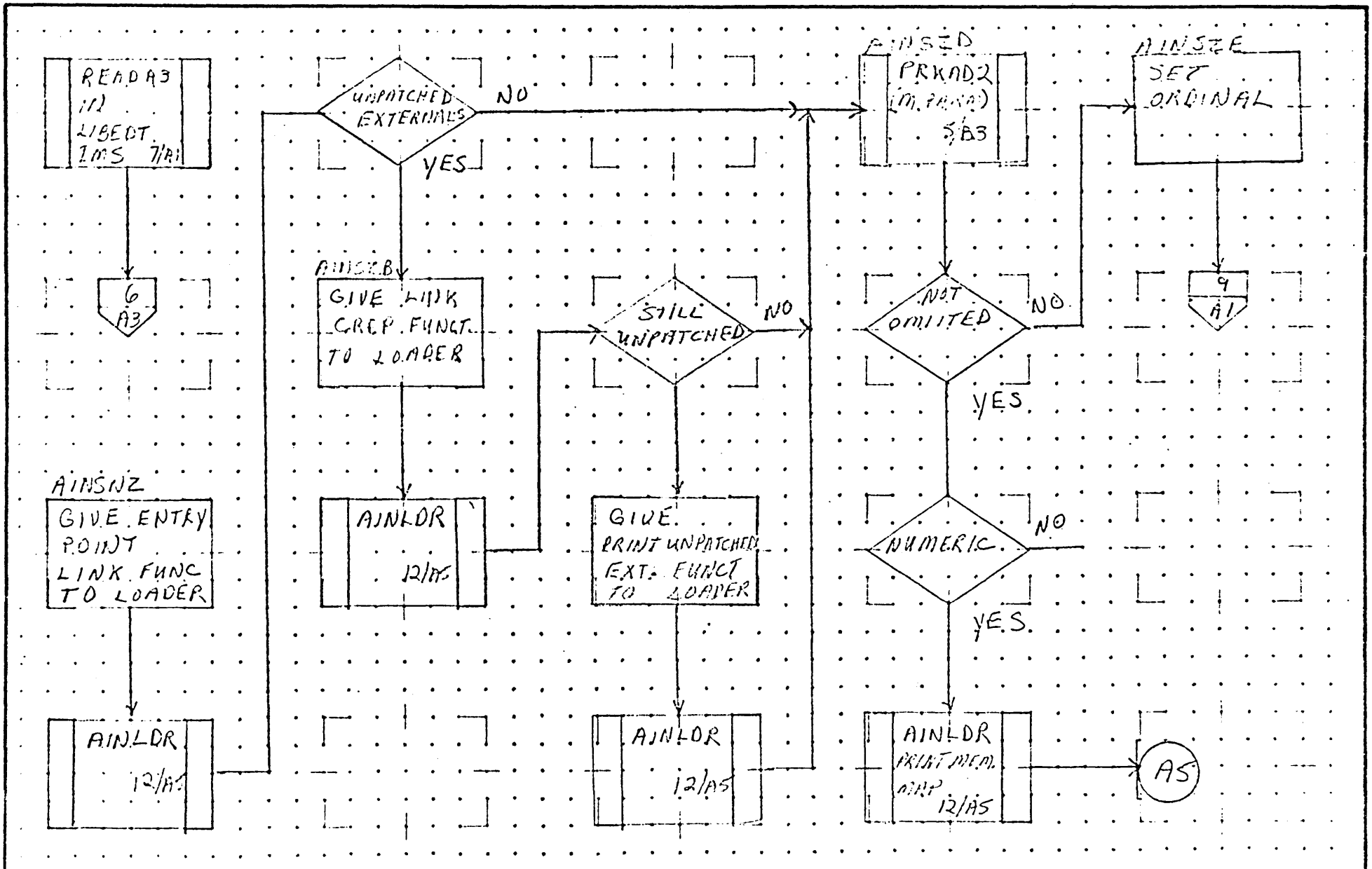
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE	PROJECT MGR.			
FLOWCHART ☒	PAGE 7 OF 2[illegible]	PROJECT NAME			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

AUG 5 1971

22001

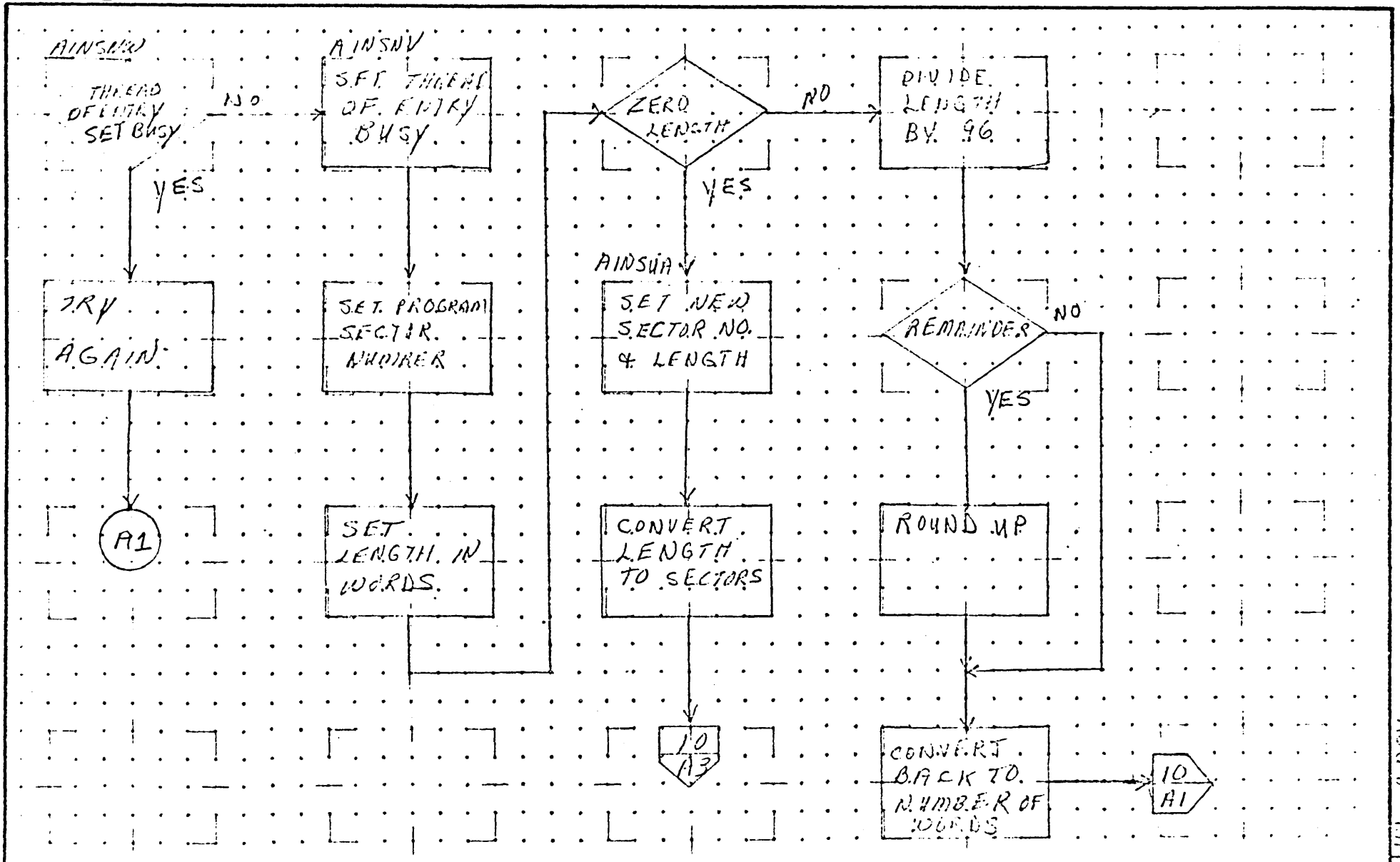


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700 DOCUMENT TITLE LIPFM PAGE 3 OF 4 NUMBER ISSUE DATE DRAWN BY G.D.G. DATE	PROJECT NO. PROJECT MGR. PROJECT NAME 1700 MSOS TASK NO. TASK NAME LIPFM	REV	APPROVED	DATE

AUG. 9, 1971 00028

AA1385 (FORMERLY CA127-1)

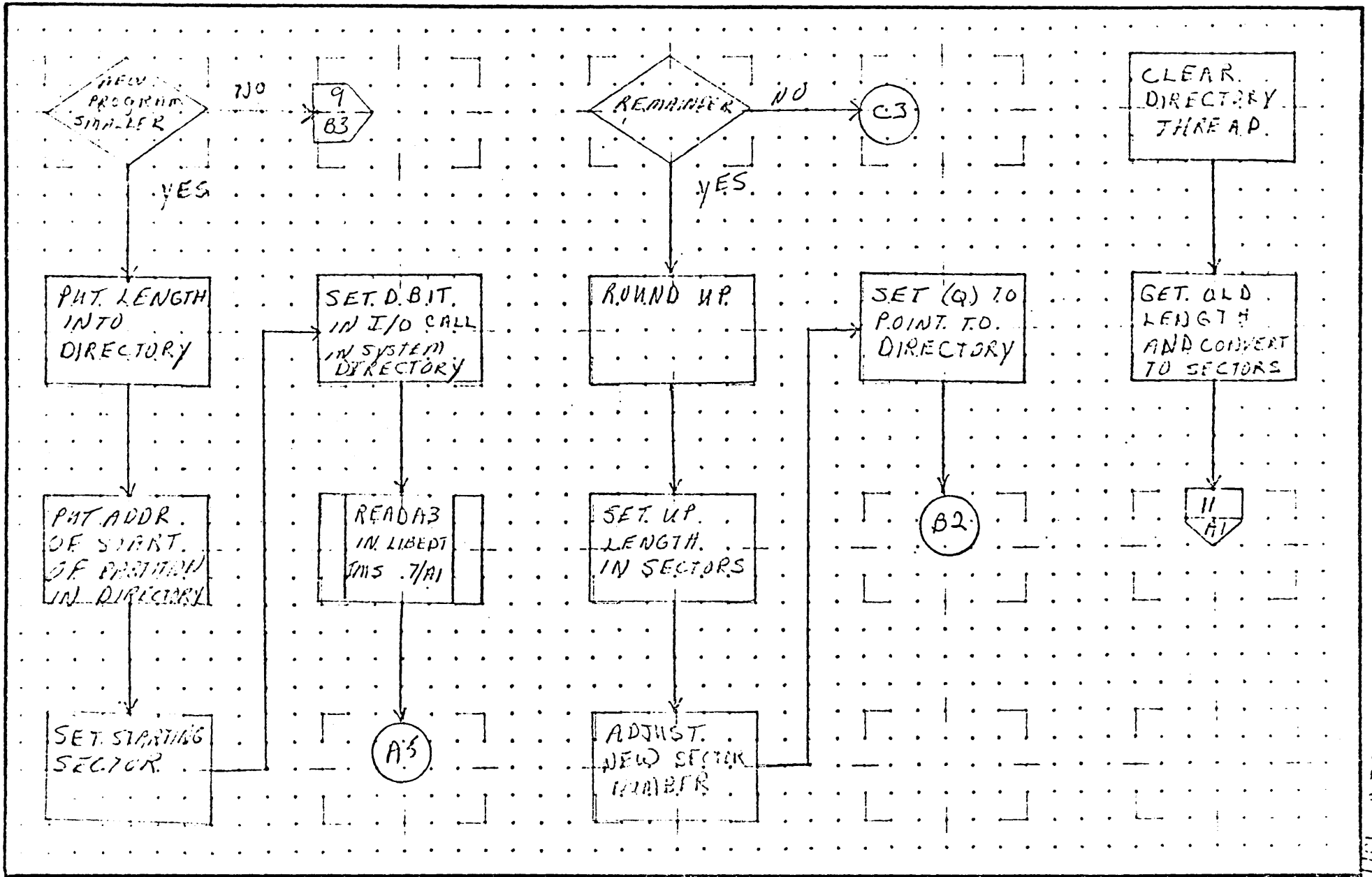
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JOBS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBRARY	PROJECT MGR.			
	PAGE 9 OF [illegible]	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C. L. C. DATE [illegible]	TASK NAME LIBR			

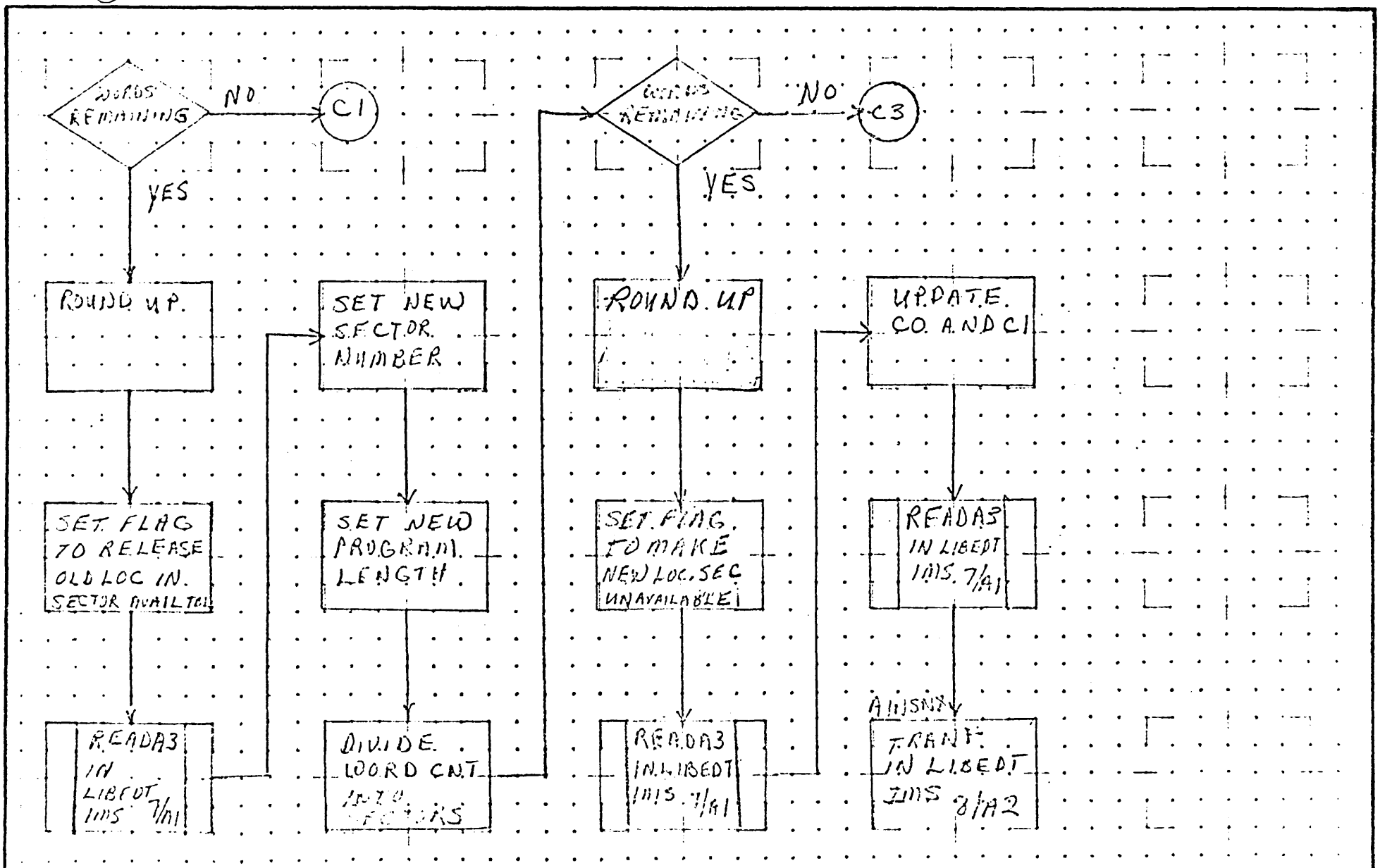
60000 1971 AUG 9

AA1385 (FORMERLY CA127-1)

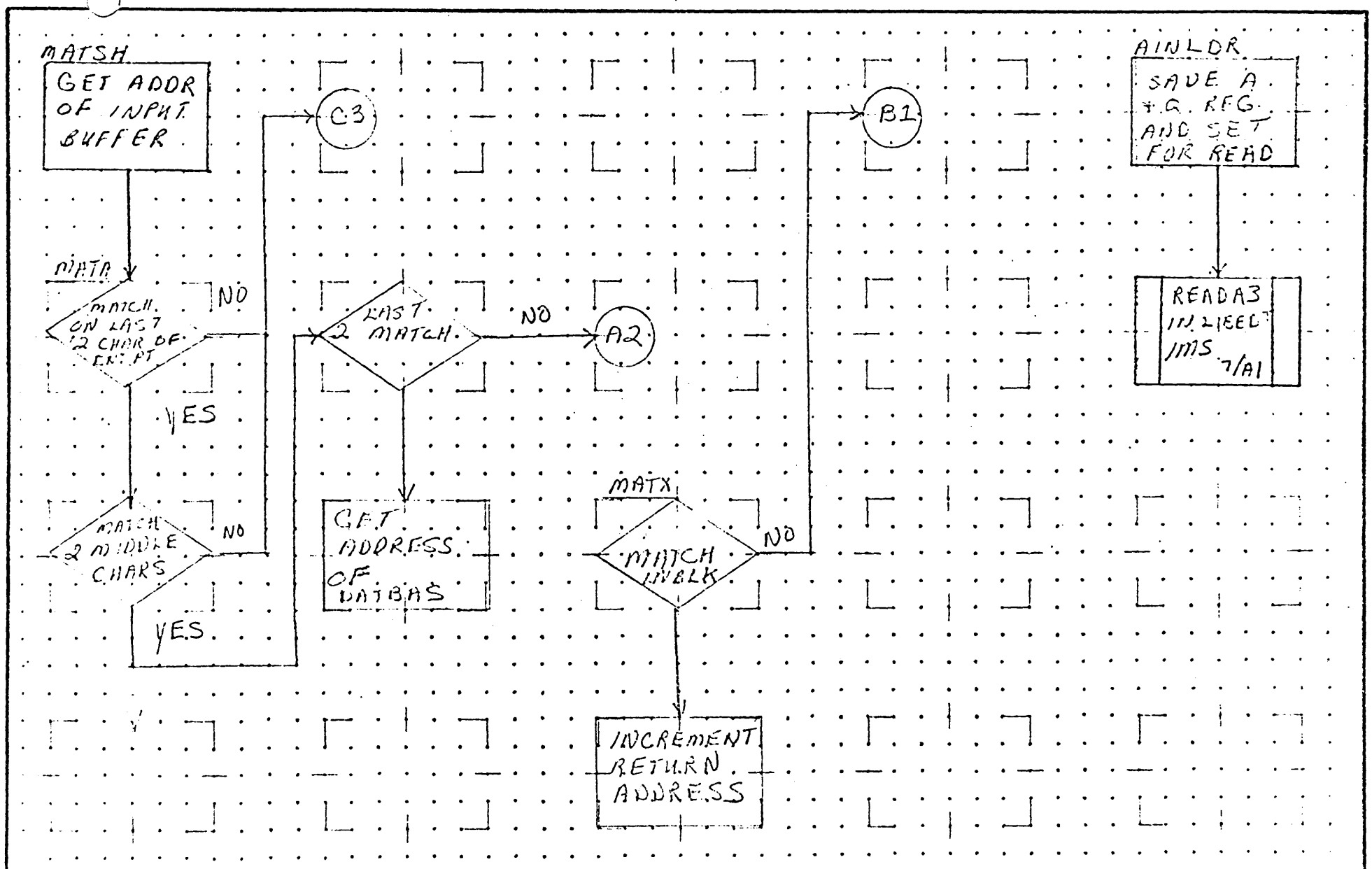


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT		PROJECT MGR.			
		PAGE 10 OF 23	PROJECT NAME [illegible] MSOS			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY C.D.C.	DATE [illegible]	TASK NAME LIBEDT			

AUG 9 1971 000:30



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	PROJECT MGR.			
	PAGE 11 OF	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE [illegible]	TASK NAME LIBEDT			

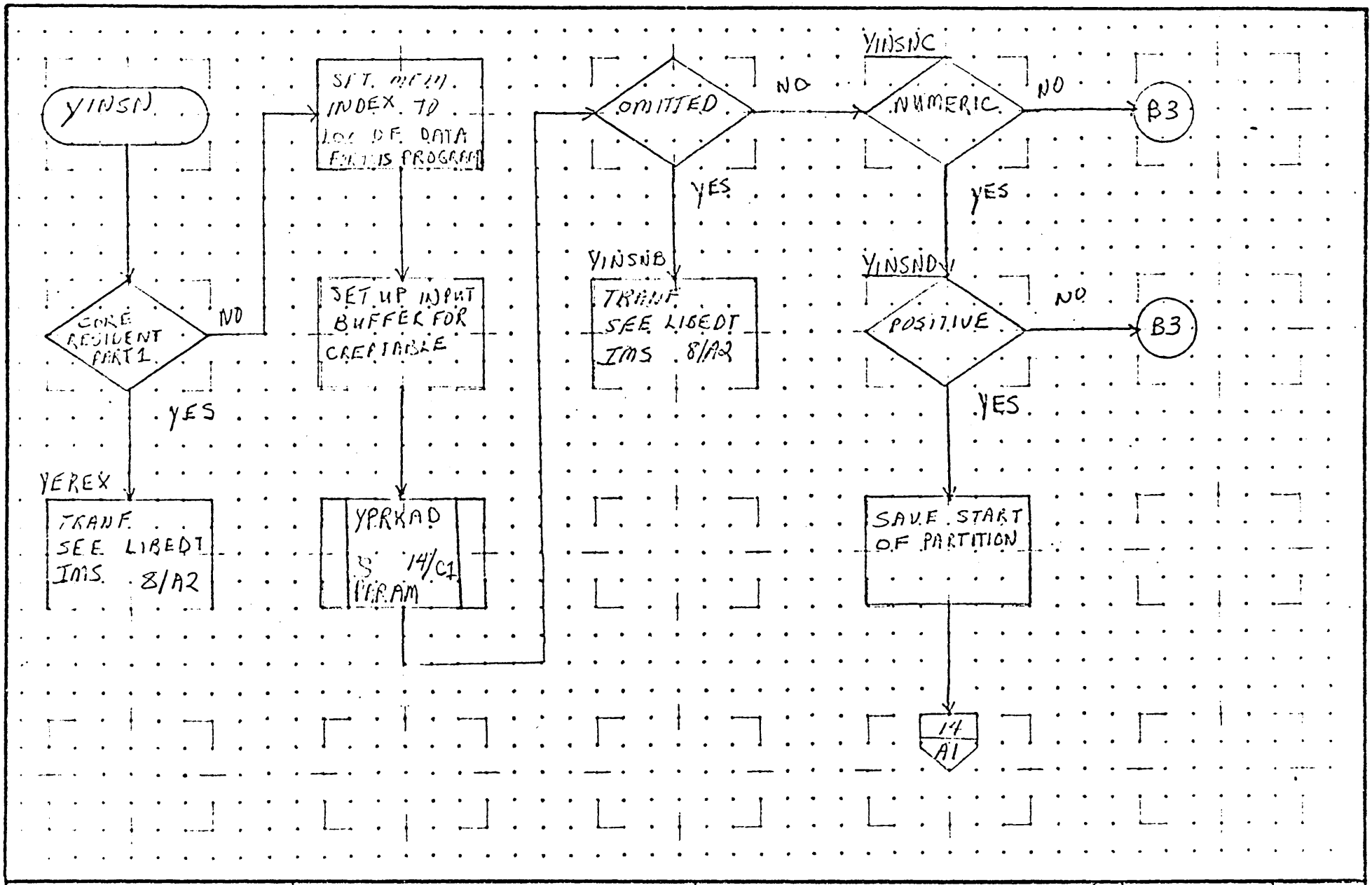


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LTRED1		PROJECT MGR.		
	PAGE 1 OF		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME LTRED1		

AA1385 (FORMERLY CA127-1)

AUG 9 1971

100322



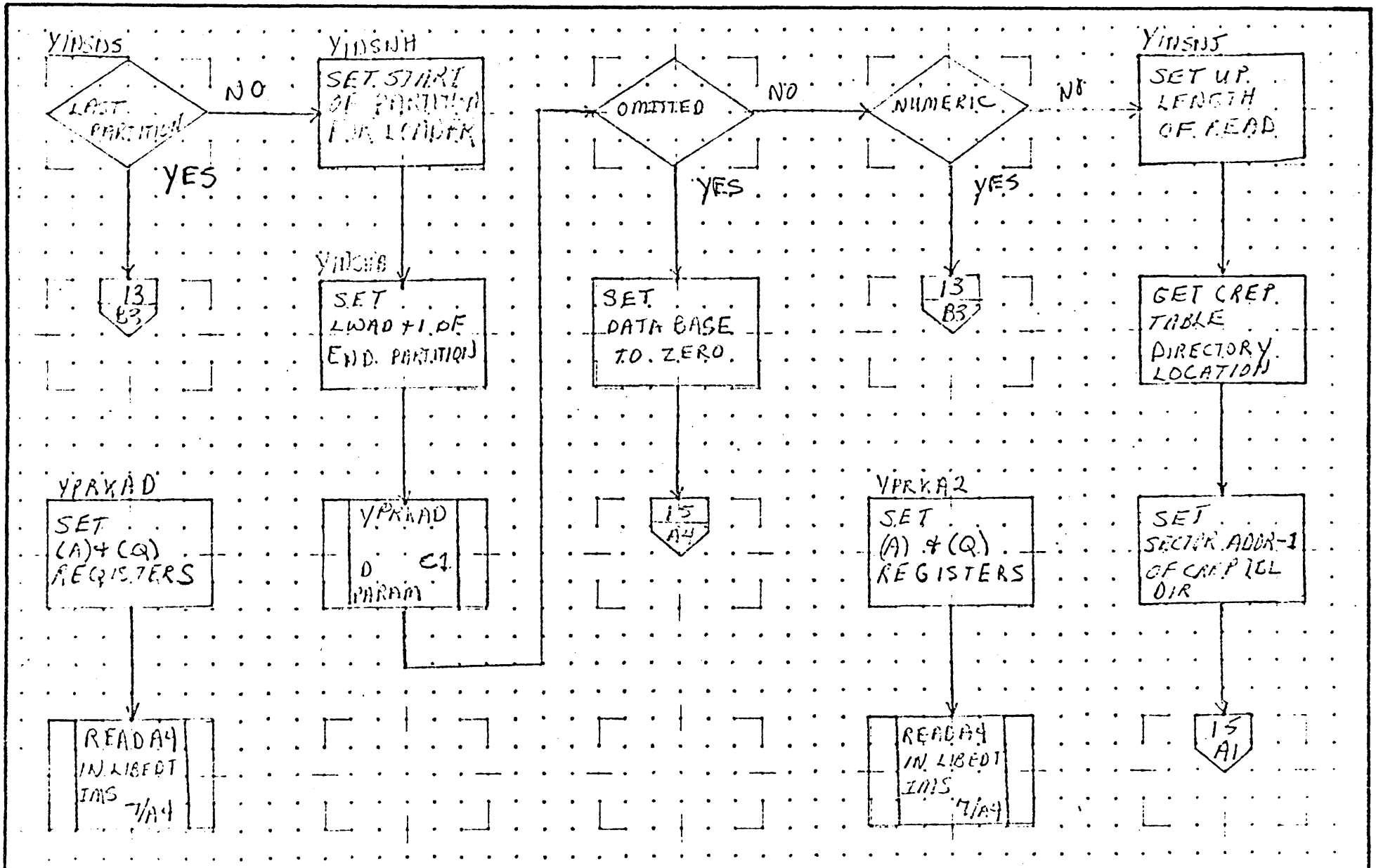
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
	PAGE 13 OF [illegible]	PROJECT NAME [illegible]			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY G D C DATE [illegible]	TASK NAME LIBEDT			

AA1388 (FORMERLY CA127-1)

PRINTED IN U.S.A.

AUG 5 1971

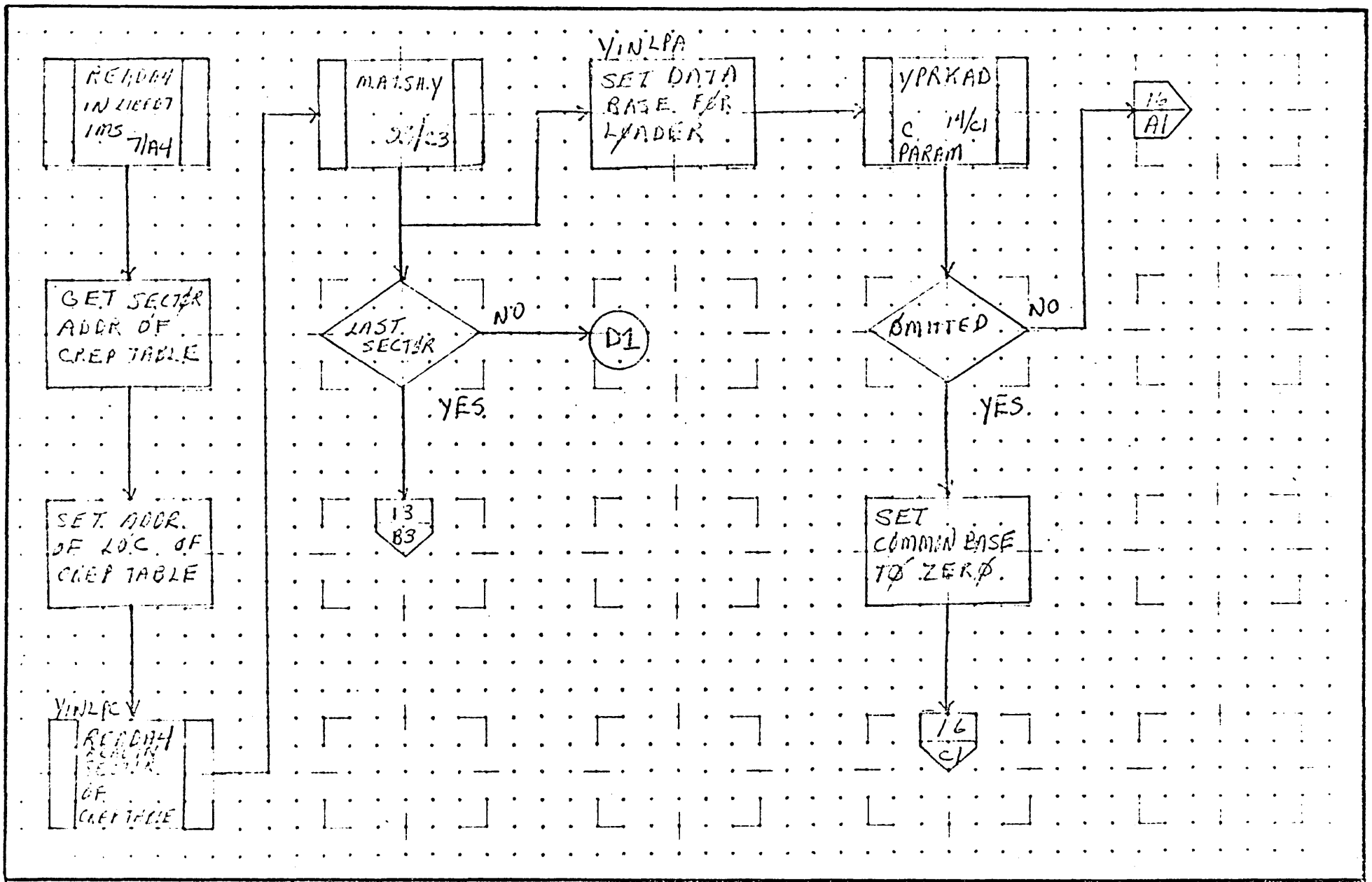
100033



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT		PROJECT MGR.			
	PAGE 14 OF 23		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE 7/14/71	TASK NAME LIBE1-7			

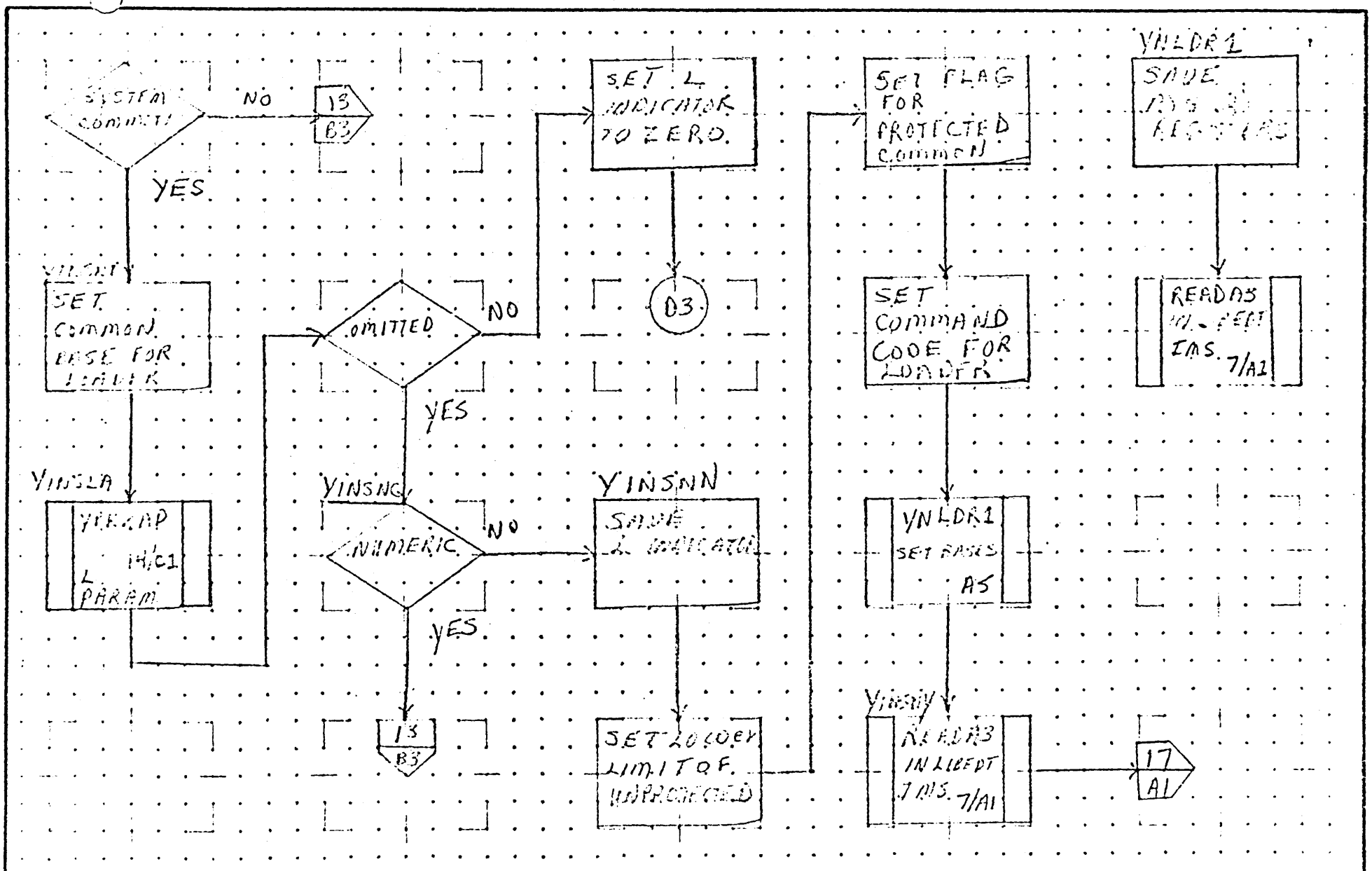
AUG 9 1971

000034



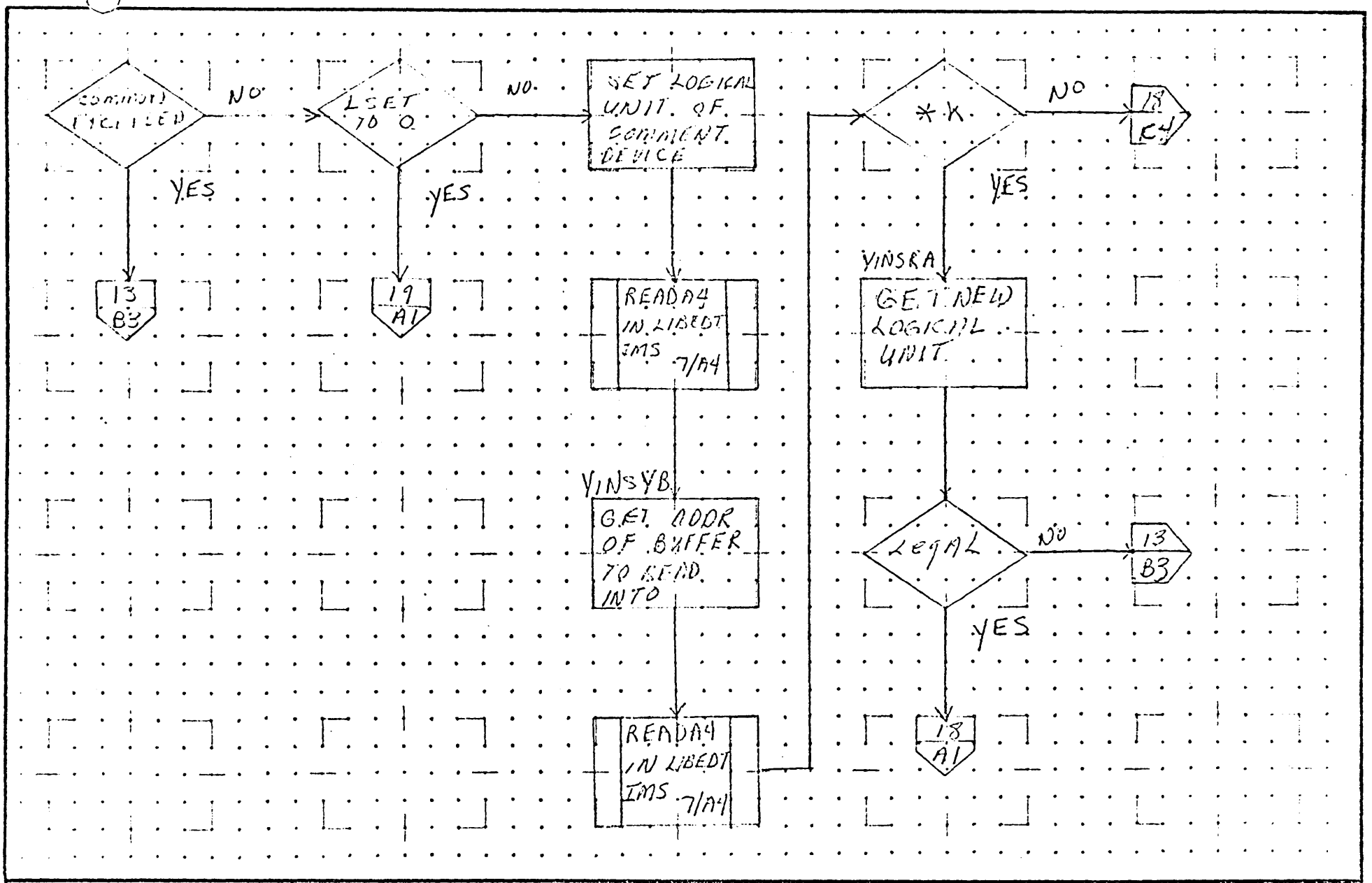
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIMIT	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY GHC DATE	TASK NAME LIMIT			

100035



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IAS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
	PAGE 11 OF 32	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY B.J.G. DATE 7/1/71	TASK NAME			

AA1385 (FORMERLY CA127-1)



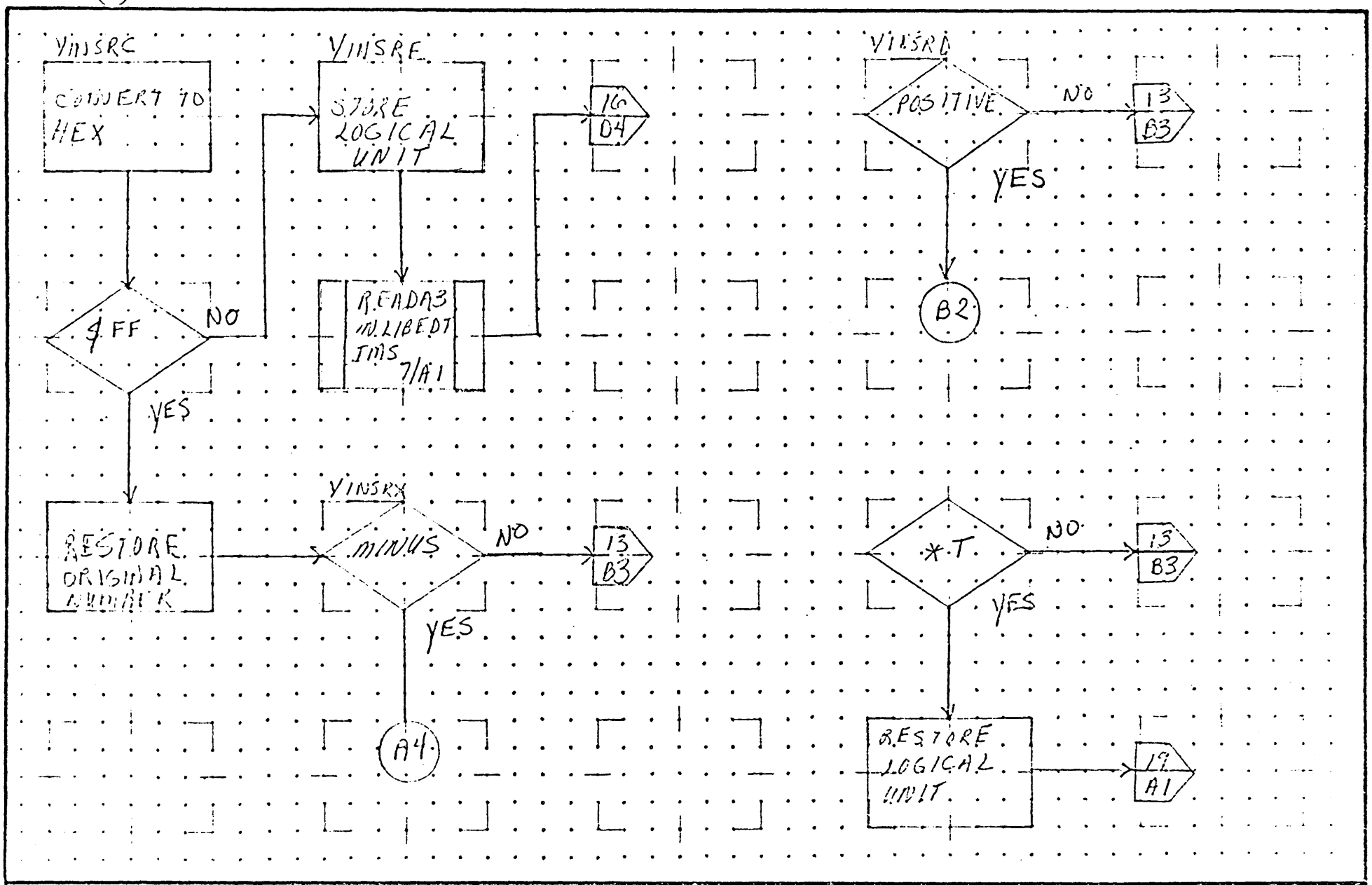
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 17...	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
	PAGE 17 OF 37	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY CDC DATE 10/68	TASK NAME LIBEDT			

AUG 9 1971

0000337

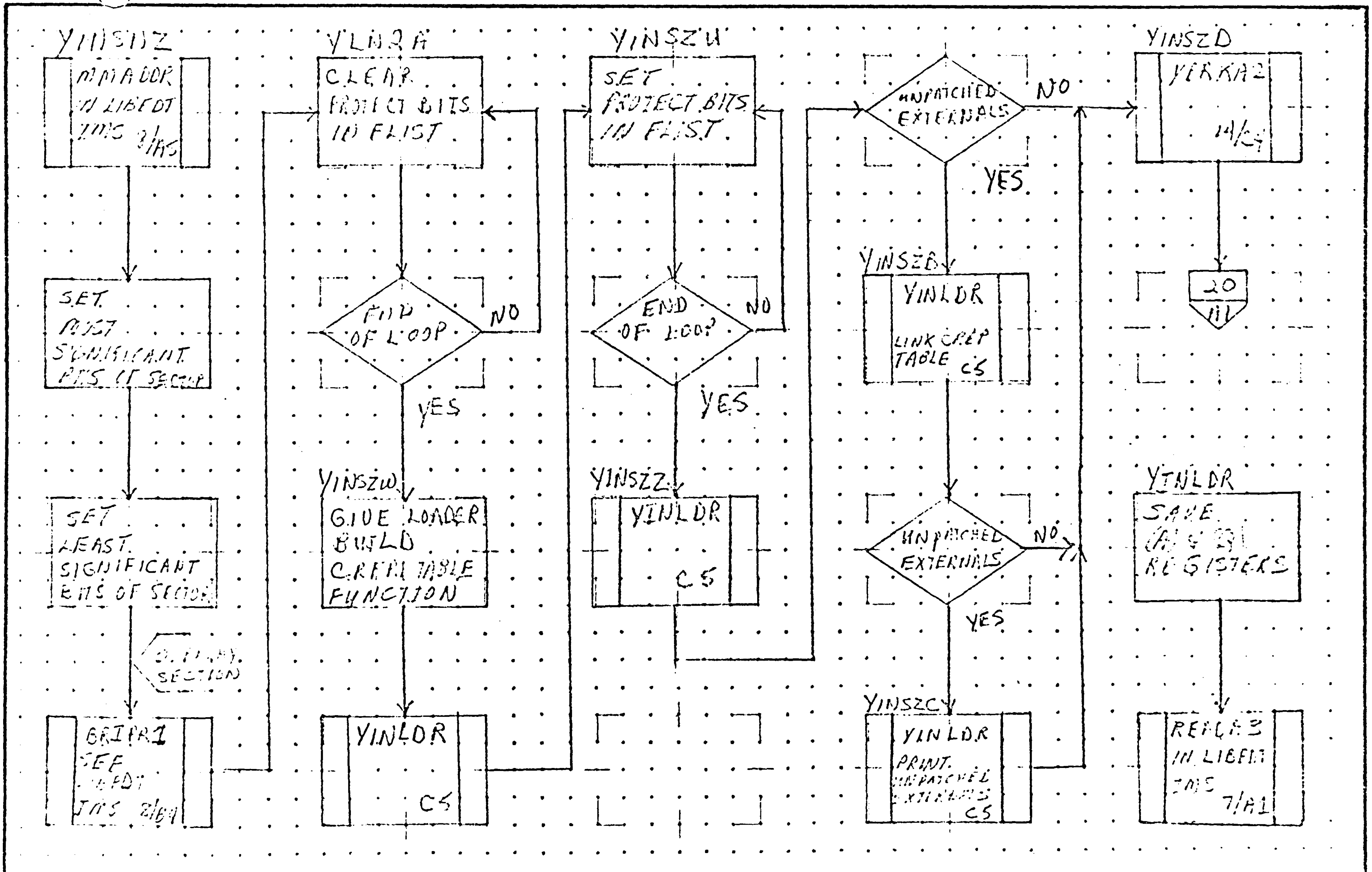
AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JNIS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBEDT	PROJECT MGR.			
	PAGE 1 OF 23	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY J.L.C. DATE 8/9/71	TASK NAME LIBEDT			

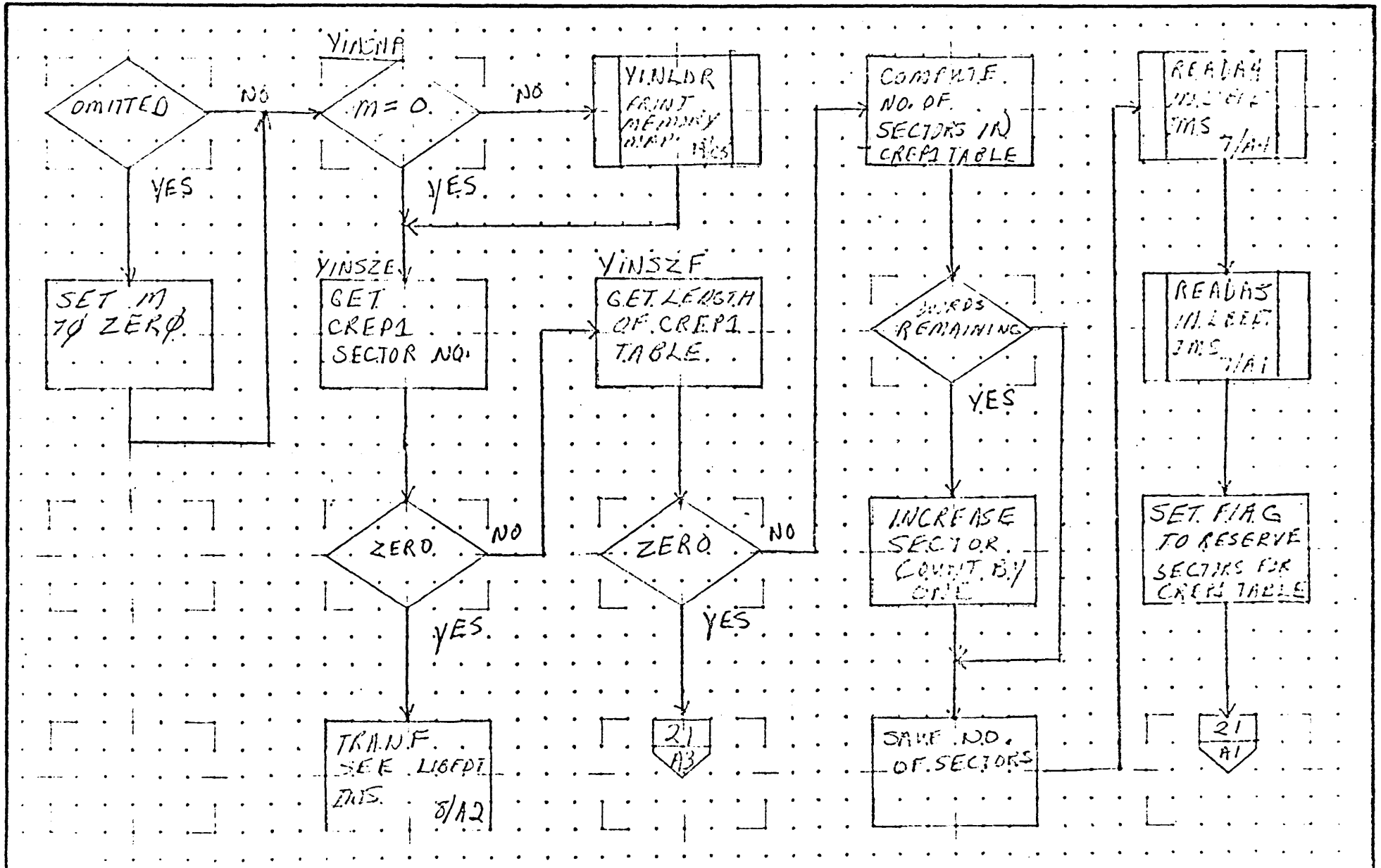
AUG 9 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS JMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE LIBEDT		PROJECT MGR.			
FLOWCHART ☒		PAGE 19 OF 23	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY CDC	DATE 7/14/71	TASK NAME LIBEDT			

AUG 9 1971

000339



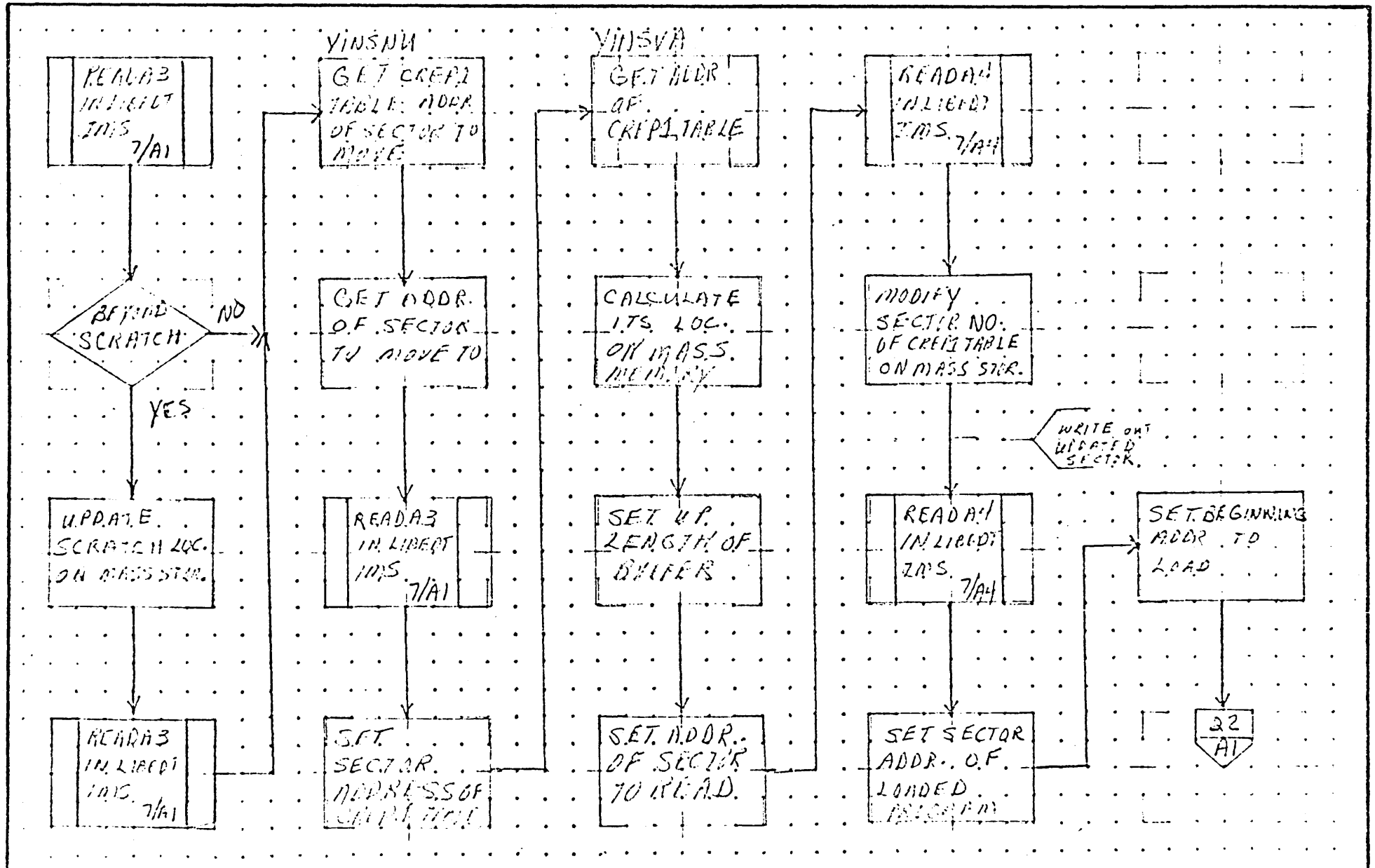
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBFDT	PROJECT MGR.			
	PAGE 10 OF 27	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C.D.C. DATE 7/14/71	TASK NAME			

AUG 9 1971

000040

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

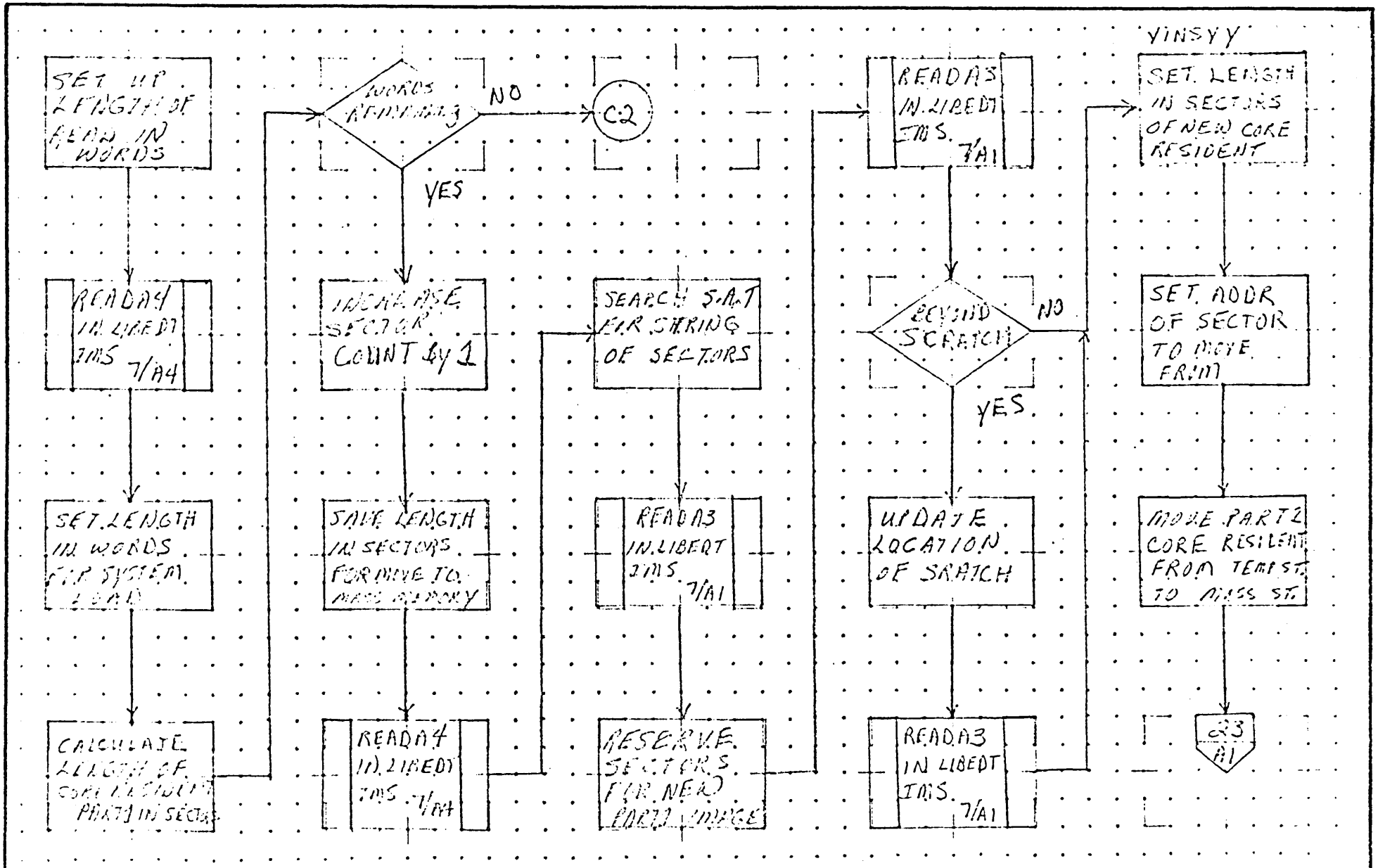


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IAS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE LIBRT		PROJECT MGR.			
FLOWCHART ☒		PAGE 21 OF 23	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY C.D.C.	DATE 7/19/71	TASK NAME			

AUG 9 1971

AA1385 (FORMERLY CA127-1)

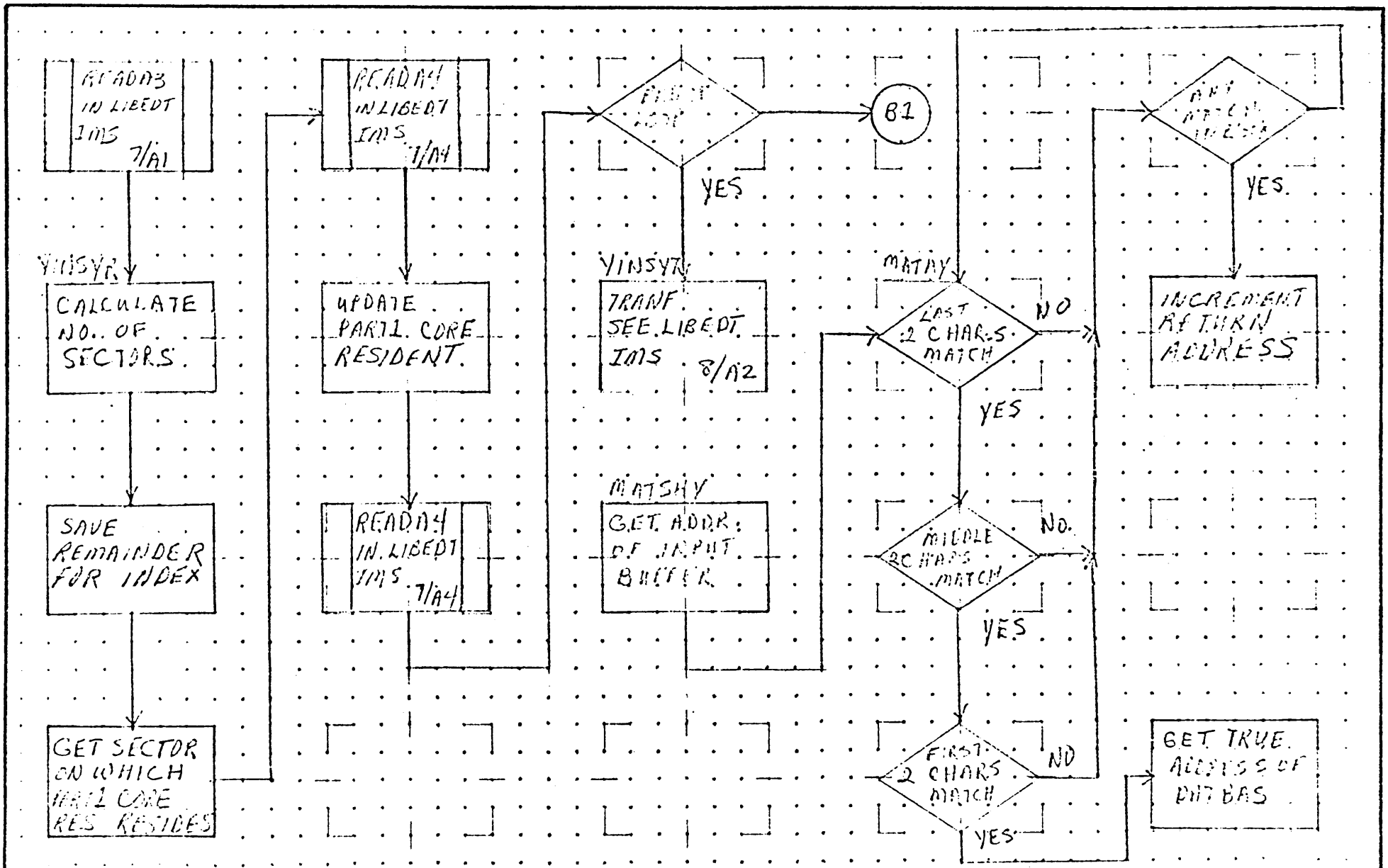
PRINTED IN U.S.A.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☑ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LIBEDT		PROJECT MGR.		
	PAGE 23 OF 23		PROJECT NAME		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY CJC	DATE 7/14/71	TASK NAME		

AUG 5 1971

60000



CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS IMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE [illegible]		PROJECT MGR.			
FLOWCHART ☒		PAGE 12 OF 13	PROJECT NAME [illegible]			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY CDC	DATE 7/68/71	TASK NAME [illegible]			

AUG 5 1971

000043

AA1385 (FORMERLY CA127-1)

PRINTED IN U.S.A.

6.0 Re-entrant FORTRAN Package

6.1 This package has been reworked to run core resident in Part 1. It is to be loaded via an ¤Y statement to LIBEDT. This will put it in the topmost partition toward $FFFF_{16}$ where it may be used by any FORTRAN program running in partitioned core. All entry points are put in the CREP1 table and linked to by the Part 1 loader only.

This is essentially the same package released with MSOS 3.0 (Sections 64 and 65 of IMS). The types of changes necessary were as follows:

1. A transfer address was necessary because the Part 1 loader requires transfer address on any program loaded under ¤A or ¤Y to LIBEDT.

2. 16 bit arithmetic must be used in locating the parameters passed in subroutine calls made from Part 1. This means bit 15 may no longer be used to indicate relative parameters to most of the subroutines to indicate indirect parameters to the floating point arithmetic subroutine (entry point FLOT). To accomodate this the following things were done:

   a. All 15 bit arithmetic was removed.

   b. All checks for and calculations of relative parameters were removed.

   c. All indirect parameters in calls to FLOT were removed. This made necessary the addition of LOCORE locations 47_{16} through 50_{16} to those locations used by FORTRAN and saved on interrupt by RDISP.

   d. Check for and calculation of indirect addresses was removed from FLOT.

   e. Relative parameters in calls to FLOT (which are¤ allowed) must have bit 15 set if the reference is backward.

3. Only Part 1 type requests may be used with this package so bit 14 had to be set in the request codes. Also Part 1 indirect request (request code 16) had to be used.

¤ This is an incorrect modification and should be removed.
M.E.D.

6.2 Following is a table of modifications by routine.

ROUTINE	MODIFICATION	COSY NUMBERS OF LINES MODIFIED
IOCODE	Transfer address, ENCODE, added	81
INITL1	Remove calculation of relative parameters	43-45
Q8QIO	Remove calculation of relative parameters	68-70, 179-181
	Part 1 indirect request used	279, 294
	Bit 14 set in request codes	369, 370
FORTRA	Remove calculation of relative parameters	100-102, 125-129, 212-214, 234-236, 278-280
	Part 1 requests used	108, 110, 113, 169, 191, 192, 262
Q8EXPR	Remove calculation of relative parameters	68-70
	Indirect parameters removed from calls to FLOT	149, 153, 178, 181
Q8PRMR	Remove calculation of relative parameters	31, 52-54
Q8ABR	Remove calculation of relative parameters	23-26
	Indirect parameters removed from call to FLOT	23-26, 31
IFALTER	Remove calculation of relative parameters	22-26
SIGNR	Remove calculation of relative parameters	17-21

ROUTINE	MODIFICATION	COSY NUMBERS OF LINES MODIFIED
	Indirect parameters removed from call to FLOT	17-21, 24, 25, 26, 29-31, 45
FXFLR	Remove calculation of relative parameters	31-35, 88-92
EXPRGR	Remove calculation of relative parameters	18-23
	Indirect parameters removed from call to FLOT	18-23, 28
	Relative addresses changed to 16 bit relative on calls to FLOT ⋈	80, 117, 134, 137, 152
SQRTFR	Remove calculation of relative parameters	51-53
LNPRGR	Remove calculation of relative parameters	17-22
	Indirect parameters removed from call to FLOT	17-22, 27
TANHR	Remove calculation of relative parameters	11-18
	Create 16 bit relative addresses in calls to FLOT ⋈	67, 69, 71
SNCSR	Remove calculation of relative parameters	18-23, 54-61
	Indirect parameters removed from calls to FLOT	18-23, 28, 54-61, 66
ARCPGR	Remove calculation of relative parameters	18-20
	Indirect parameters removed from calls to FLOT	18-25, 28

⋈ This modification is not necessary.

M.E.D.

CONTROL DATA CORPORATION
Arden Hills Development DIVISION

AUG 9 1971

00047

DOCUMENT CLASS IMS PAGE NO.
PRODUCT NAME 1700 MSOS 65K Special System
PRODUCT MODEL NO. MACHINE SERIES 1700

ROUTINE	MODIFICATIONS	COSY NUMBERS OF LINES MODIFIED
FLOATR	Remove code to handle indirect parameters	198-206
	Remove 15 bit addressing arithmetic	210-211

AA 3777

PRINTED IN U.S.A.

SECTION B

NEW ROUTINES FOR

65K MSOS

1.0.0 PART 1 - 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.

1.0.1 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.

1.0.2 LOADING OF THE LOADER

The Loader may be brought into core by the LIBEDT ¤A or ¤Y statement.

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 Loader divides the remaining unprotected core into pages for the command sequence, entry point names and external names. The page length is 195 words of which 3 words are flags for paging control. {192 word pages are written to mass memory}

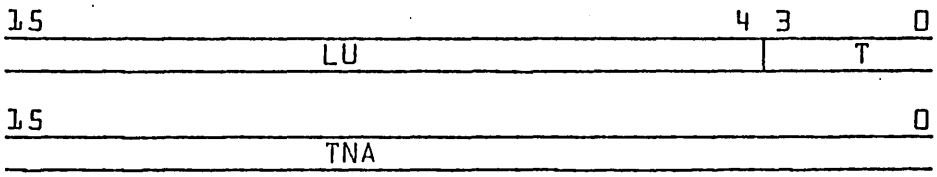
The upper limit of unprotected core is contained in location $F6. It is defined as the highest {toward Part 1} unprotected address + 1. The lower limit of unprotected core is contained in location $F7. It is defined as the lowest {toward 0} unprotected address - 1.

1.0.3 OPERATION OF THE LOADER

A Loader operation is initiated with a jump to the lowest {toward 0} core location occupied by the Loader.

1.0.4 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	Set program base addresses
2	Patch program entry points
3	Patch to Part 1 core resident entry points {CREP1}
4	Build table of Part 1 core resident entry points
5	Patch to Part 0 core resident entry points
6	Print memory map
7	Print unpatched externals

The "LU" field contains the logical unit number of the input device to be used for this operation. This field is ignored if the T field is a 1,2,4,6, or 7.

The TNA field contains the core address of the program bases. This information is significant only if the Loader operation is a set bases function. The Q register is ignored in all other cases.

1.0.5 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.

1.0.6 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.

1.0.7 SET PROGRAM BASES OPERATION

The purpose of this operation is to set the bases for the program(s) to be loaded, divide unprotected core into pages, and to initialize the page flags. This function must be the first function given to the loader after it is read into core by LIBEDT. The T field is 1 and the LU field is ignored. The Q register contains the address of the first word of the program bases in LIBEDT.

The program bases are:

Word 0 Address of start partition
Word 1 Last word address + 1 of the end partition
Word 2 Address of system data base (0 is system data not used)
Word 3 Address of system common base (0 if system common not used)

1.0.8 PATCH PROGRAM ENTRY POINTS

This operation is performed subsequent to relocatable binary loading. The Loader will attempt to match external names in the Loader External table with entry point names in the Loader Entry point table. The last transfer name encountered is matched with an entry point name and the address is stored in the jump instruction that precedes the command sequence. If no unpatched externals exist after the patching operation, the pages are written out to the disk.

1.0.9 PATCH TO PART 1 CORE RESIDENT ENTRY POINTS (CREP1)

If T = 3 on input, the Loader reads the CREP1 table into core and stores these entry points into its entry point table, and patches any entry points referenced by other programs. If no unpatched externals exist after the patching operation, the pages are written out to the disk.

If a duplicate entry point is encountered in the Loader entry point table while loading the CREP1 table, the CREP1 entry point is ignored and no diagnostic is issued.

1.1.0 BUILD TABLE OF PART 1 CORE RESIDENT ENTRY POINTS

This function is given by LIBEDT when it encounters a *Y statement and must be given before the CREP table is stored into the Loader entry point table. The program(s) entry points are squeezed together and stored on the disk behind the command sequence. Locations in the loader which contain address and length of CREP1 are pointed to by I on return to LIBEDT. LIBEDT will move the table to an appropriate place on the disk and update the Sector Availability Table.

1.1.1 PATCH TO CORE RESIDENT

If T = 5 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.

If a duplicate entry point name is encountered in the Loader entry point table, the CREP entry point is ignored and no diagnostic is issued. If no unpatched externals exist, the pages are written out to the disk.

1.1.2 MEMORY MAP OPERATION

The T field is 6 if the Loader is to produce a memory map. The LU field is ignored and the Q register is ignored.

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.

1.1.3 PRINT UNPATCHED EXTERNALS

This operation can be performed after any relocatable or table loading operation. The 6 character names of all unpatched externals are printed on the system print medium. An E is output to the comment device. The operator must type in ⋈ (cr) to resume operations or ⋈ T (cr) to indicate the load operation should be abandoned.

1.1.4 AVAILABLE CORE FOR PAGES

The portion of the block of unprotected core not occupied by the Loader is available for pages. Pages start {$F7} + 1 and are in 195 word blocks.

1.1.5 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 that the input device is a card reader.

NAME OF BLOCK	IDENTIFICATION
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.

1.1.6 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 summarized below:

EXIT FROM THE LOADER

1.1.6.1 Program loading continues until interrupted by an EOL statement, an operating system control statement, or an error condition.

1.1.6.2 If loading is terminated by an EOL statement, the A register contains zero, the Q register contains zero, and the I register contains the address of the locations in the Loader that contain the sector address and number of words of the command sequence.

1.1.6.3 If loading is terminated by an operating system control statement, the A register contains the address of the operating system control statement.

1.1.6.4 If the loader operation is terminated by an irrecoverable error, the A register contains negative zero.

If loader operation is terminated with unpatched externals, the Q register contains negative zero.

SUMMARY OF TERMINATION PARAMETERS

1.1.6.5 A register:
1. Zero if operation terminated normally
2. Negative zero, if operation was terminated by an irrecoverable error
3. Address of monitor control statement if operation was terminated by a control statement.

1.1.6.6 Q register:
1. Zero if no unpatched externals occur after a loader operation
2. Negative zero if unpatched externals occur after a loader operation

NOTE: The Q register content is only significant after functions of 2,4,5, and 7.

1.1.6.7 I register:
1. Contains address of the locations in the Loader that contain the sector address and length of the command sequence or CREPL table.

NOTE: The I register content is only significant after a function of 0 and 4.

1.1.7 OPERATION OF 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 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 EXTM pseudos in any of the other subprograms which reference this name. Because of the EXTM 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 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 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.

1.1.8 LOADER TABLE

The Loader Table comprises a list of entry points and externals processed during a sequence of Loader Requests. The Loader Table consists of an External table, a Hash table for entry points and an Overflow table for entry points. The Loader Table is written into the pages in unprotected core.

1.1.8.1 Each entry in the Hash table and Overflow table consists of five words.

EXAMPLE
ENT ABCDEF

0	A	B
1	C	D
2	E	F
3	VALUE	
4	POINTER	

Value is the address of the entry point.

Pointer is a link to entries in the Overflow table with the same hash value.

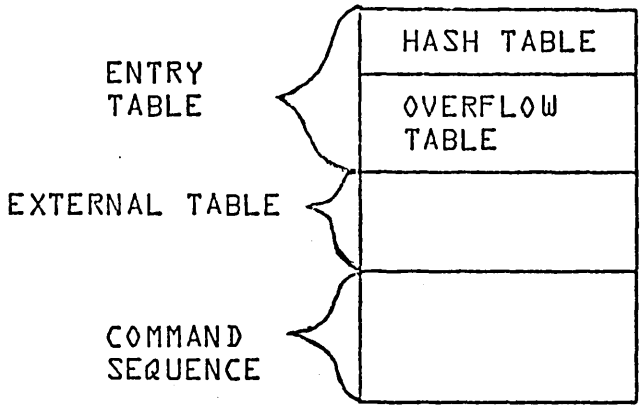
1.1.8.2 Each entry in the External table consists of four words.

EXAMPLE
EXT ABCDEF

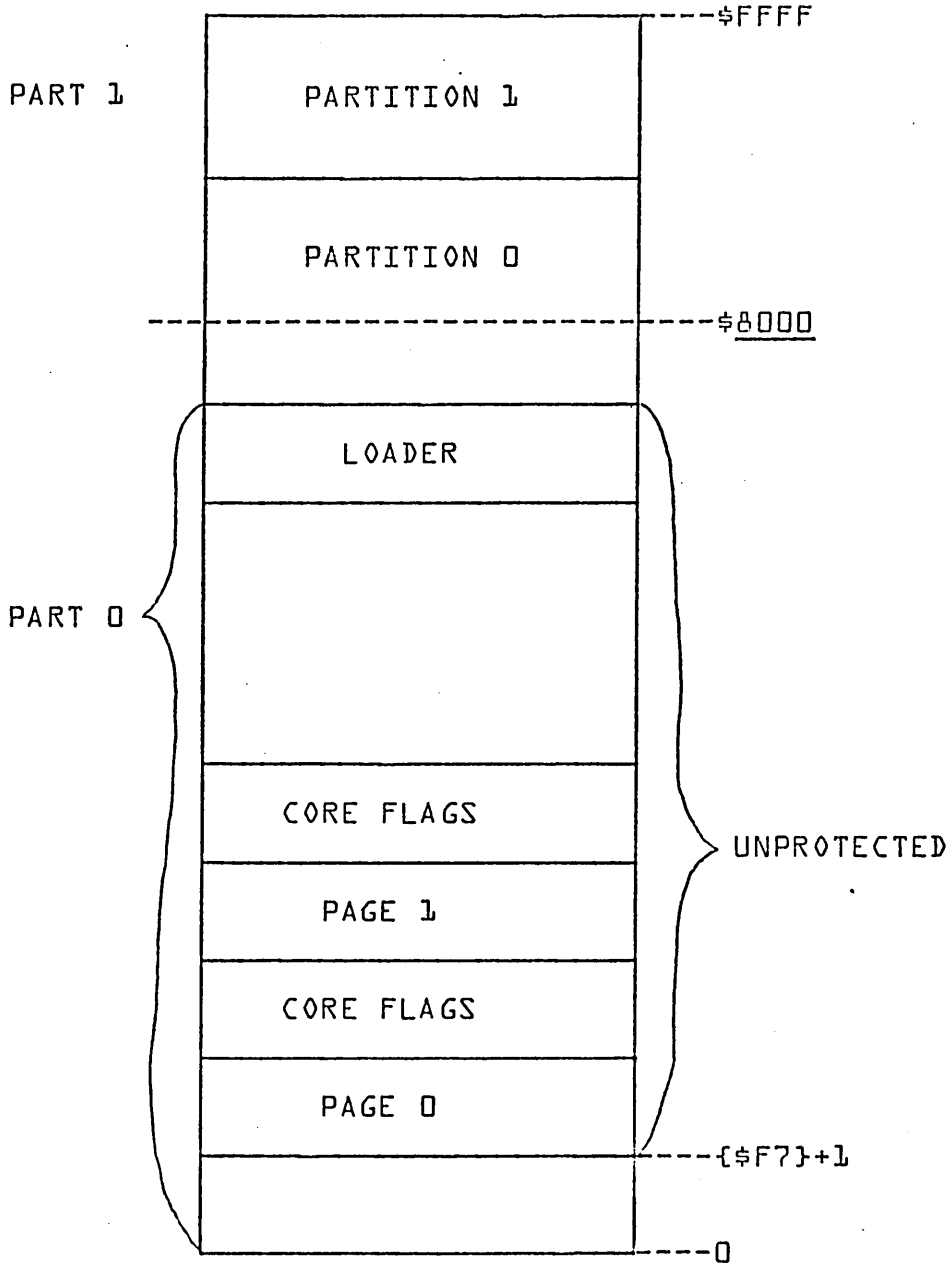
0	AB
1	CD
2	EF
3	POINTER

Bit 15 of word zero indicates a relative or absolute external.

Bit 15 of word one is used to determine if the external is patched.



CORE MAP WITH PART 1 LOADER



1.2.0 LOAD1 ROUTINE - INITIALIZATION

The LOAD1 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.

The 1st instruction to be executed upon entry to the Loader is a return jump to the location whose label is PART 1. Prior to entering the Loader, the entrance parameters have been placed in the A&Q registers.

The only function of PART 1 is to record the entrance parameters in temporary storage locations:

{A register}——— AINP
{Q register}——— QINP

1.2.1 CONBAS

The function of CONBAS is to establish the relocation base for the Constant Table and record entrance parameters.

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 as 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.

1.2.2 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 147th 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 147+{I}. This address is placed in the location 57+{I}. The value for the starting address of this input area +1 = 148+{I} is placed in the location 58+{I}. Also the value for the starting address -3 = 144+{I} is placed in the location 62+{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 PART 1 with a return jump, the value = °LOAD+1° is placed in the location PART 1. Therefore, the location 23+{I} is set to the value = {PART1}-1.

The 31st 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 LOAD1 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 LOAD1, it is necessary to declare ISAV as an external name in the other module.

1.2.3 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 .
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 LOAD1 for those locations in the Constant Table preset to address constants where these address constants refer to locations in other routines.

1.2.4 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	EXTCTR	points to next available location in External Table
8	ENDSW	=1 if last relocation byte in RBD or BZS block
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

ORDINAL	NAME	USE:
12	CSQNUM	no. of sectors reserved before command sequence start
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	NOTLNK	will be a 1 if unpatched externals are found
17	ENDINP	highest address in load time core for command sequence storage
18	BLANKS	ASCII code for spaces = $2020
19-22		USED for tempory storage
23	BASE	base address of loader
24	WRDCNT	address or character reference counter
25	COUNT1	counter
26	BZSSW	used by subroutines common to RBDPR0 & BZSPR0; =1 for BZS block; =0 for RBD block
27	BLKCNT	block counter
28	SW6	index counter
29	ASAV	temporary storage for A register
30	QSAV	temporary storage for Q register
31	ISAV	storage location for constant table relocation base

ORDINAL	NAME	USE:
32-34	XFRNAM	storage of 6 character XFR name
34-38	NAME	temporary storage
39-42	TABSCH	Loader Table search routine entrance and return
43	HSHADR	relative address of ENT or EXT name on mass memory
44	HASHCD	HASH CODE OF ENT or EXT
45-50	TEMP	Temporary locations used in hash routine
51-52	BINASC	Storage of ASCII code for number conversion
53-56	PRINT3	error output routine {resume operator}
57	INPXC0	contains address constant = ▽INPUT▽ or 147+{I}
58	INPXC1	contains address constant = ▽INPUT▽+1 or 148+{I}
59-61	PRINT2	error output routine {Stop Operation}
62	INDXCC	contains address constant = ▽INPUT-3▽ or 144+{I}
63-64	NXTINP	jump instruction to read next input block
65-80	HEXDIG	ASCII codes for hex digits
81-86	ADJOVF	routine to perform address arithmetic
87-91	CONVRT	binary to ASCII conversion routine entrance AND RETURN
91-94	PRINT4	print out routine {prints characters name and 4 digit hex address}

ORDINAL	NAME	USE:
95-98	PRINTS	print out routine {prints 6 character names}
99	AINPUT	contains A register upon entry to loader
100	QINPUT	contains Q register upon entry to loader
101-104	LINK1	entrance to routine which patches an entry to an external
105	MASINP	contains logical unit for MDRIV
106	HSHTBL	number of hash codes {entries} in hash table
107	PGELGN	length of page with core flags
108	LGEPGE	largest command sequence page that has been stored into
109	IGNORE	flag to ignore duplicate Ent points during crep load
110	LNKSTR	address of linktable
111	LNKCTR	next available location in link table.
112	LNKEND	last address +1 in linktable
113-114	INPLUN	logical unit number for input {L&A parameters for calling sequence}
115	ADDR	temporary storage for loop control
116	EXTSTR	word address of start of external table
117	CORADR	contains {$F7}+1
118	PRODAT	non zero if protected data is declared

ORDINAL	NAME	USE:
119	PROCOM	non zero if protected common is declared
120	PAGE	length of page on mass memory, must be a multiple of 96
121	CSQCTR	last address of program command sequence storage +1
122	OVSTR	word address of overflow table start
123	OVCTR	points to next available location in overflow table
124	DIFCON	temporary storage
125	ENTSEC	start sector of ent/ext tables
126	CSQSEC	start sector of command sequence image
127	MAXPGE	maximum page number that can be used on mass memory
128	NOPAGE	number of pages in unprotected core
129	PARBAS	address of start partition
130	PARLIM	last word address +1 of last partition
131	STRSEC	start sector of image on mass memory
132	MDSWCT	number of words stored on mass memory
133	XFRADR	transfer address of name from transfer block
134	AHOLD	temporary storage for A register
135	QHOLD	temporary storage for Q register
136	SECTOR	°SECTOR° = 96_{10}

ORDINAL	NAME	USE:
137-138	CMNXIT	address of common exit routine
139	EXTSWT	non-zero if processing External block
140	SAVEA	temporary storage
141	SECTEM	sector number of CREP1 table storage on mass memory temporary storage
142-143	TEMP3	temporary storage
144	PROGCT	number of words of command sequence in the program being loaded
145-242	INPUT	input buffer

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.

1.2.5 Exit from LOAD1

Exit from LOAD1 is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT	BRANCH	
	JMP	BRANCH	

1.3.0 BRNCH1

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 eight locations labeled as follows:

RBLOAD	relocatable binary load
STBASE	set bases function
LNKENT	link program entry points
LNCRP	link to crep1 table
BLDCR1	build crep1 table
LNKCR1	link to crep table
MAPS	print memory map
PRNEXT	print unpatched externals

1.3.1 Constant Table Storage Referenced by BRNCH1

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
AINPUT	101
BINASC	51
BLANKS	18
BLKCTR	27
COMBAS	1
CONVRT	90
DATBAS	2
INPCTR	15
INPUT	147
INPXC0	57
INPXC1	58
INPLUN	115
PRINT3	54
PRINT4	95
PRINT2	59
QINPUT	102
SW6	28
TABSCH	40
PROBAS	3
PARBAS	131
MSDWCT	134
STRSEC	133

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
IGNORE	111
SAVEA	142
XIT	139
MASINP	107
EXTSTR	118
INPWRD	10
CS@SEC	128
LINBUF	147
BLKCTR	27
NOTLNK	16
EXTCTR	7

1.3.2 The usage of constants storage is as follows:

1. AINPNT contains the logical unit number of the input unit if a relocatable binary program is being loaded in bits 3-15 and the type of loading operation in bits 0-3. AINPUT is used to determine the type of load operation.
2. BINASC contains ASCII code for four hex digits of entry point address.
3. BLANKS contains two spaces.
4. BLKCTR is used for a loop control.
5. COMBAS if COMBAS equals zero there is no common storage.
6. CONVRT is the address of a jump to convert.
7. DATBAS is zero if no data storage is used.
8. INPCTR is used to address location of command sequence storage.
9. INPUT is the address of the input buffer.
10. INPXC0 is the address of the input buffer.
11. INPXC1 is the address of INPUT +1.
12. INPLUN is the logical unit number for input device.
13. PRINT3 is used to print an error message.

14. PRINT4 is used to print names and relocation bases for common block and data block.

15. PRINT2 is used to print an error message and terminate the load.

16. QINPUT contains core address of the first of four sequential locations containing the base addresses used by the loader to set limits.

17. SW6 is used as a loop counter

18. TABSCH is the address of a jump to TABSCH.

19. PROBAS is the relocation base for the program currently being loaded.

20. PARBAS is address of start partition.

21. MSDWCT is the number of words on mass memory.

22. STRSEC is the start sector of image of mass memory.

23. IGNORE is the flag to ignore duplicate entry points.

24. SAVEA is used as an exit parameter.

25. XIT is the address of a jump to CMNXIT.

26. MASINP is the logical unit used for MDRIV.

27. EXTSTR is the word address of the start of the external table.

28. INPWRD is used to address the first word of entry point table.

29. CSQSEC is the start sector of command sequence image.

30. LINBUF is used to put blanks into input buffer.

31. NOTLNK is a one if unpatched externals are found.

32. EXTCTR points to next available location in external table.

1.3.3 Communication Region Constants Used by BRNCH1

	CONSTANT	LOCATION
1.	$7FFF	MASK1 = $42
2.	ordinal for mass storage device containing scratch area	$B3
3.	number of 1st sector in scratch and area on mass storage	$C1
4.	ordinal for mass storage device containing Program Library and Program Library Directory	$C2
5.	address for return to monitor	$EE
6.	entry to operating system	$F4
7.	ordinal for standard input device for system	$F9
8.	address of system Library Directory	$EB

1.3.4 Entrance to BRNCH1

The BRNCH1 routine is entered directly from CONBAS of the LOAD1 routine. The BRNCH1 is entered by the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	BRANCH	

where the name BRANCH is declared as an entry point in the BRNCH1 routine and as an external in the LOAD routine.

1.3.5 RBLOAD - Relocatable Binary Loading

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 LIDRV1 subroutine.

If the input device selected by RBLOAD is the mass storage device containing the scratch area, the location PROSEC is 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 location PROSEC is in LIDRV1. If the input device is not a mass storage unit this location contains zero.

Next a return jump is made to the beginning of the routine which performs the actual loading operation. The name of this routine is LOADR1. The name LOADER is declared as an entry point within the LOADR1 routine and as an external in the BRNCH1 routine.

1.3.6 Exit from RBLOAD

The following occurs upon exit from RBLOAD:

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.

1.3.7 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 16 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

SSSSXXXXXXSShhhhSSSSXXXXXXSShhhhSSSSXXXXXXSShhhhSSSSXXXXXXSS

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.

1.3.8 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 LDXIT1.

2. The exit parameters are placed in the A, Q & I register as follows:

   0 ⟶ A

   0 ⟶ Q

3. A return jump to the monitor is made from LDXIT1.

1.3.9 Subroutines Used by and External to BRNCH1

Subroutines used by and external to the BRNCH1 subprogram are as follows:

PRINT3	SQUEEZ
PRINT4	PATCH
PRINT5	WRTOUT
CONVRT	PRINT6
CDRIV	STBASE
MDRIV	LNKENT
LDRIV	LNKCRI
LOADER	BLDCRI
DISKRD	

1.3.9.1 PRINT3

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 in the A register. The information placed in the A register consists of either 2

characters or of 1 character followed by a space. Entrance is made to the PRINT3 subroutine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	PRINT3{54}	
	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.

1.3.9.2 PRINT4

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{94}	
	RTJ-	PRINT4,I	

The output device used by the PRINT4 subroutine is the same as that used by PRINT3.

1.3.9.3 TBSCH1

Given the storage area for a 6 character name, the TBSCH1 subroutine is used to search for this name in the Loader Table. Prior to entering the TBSCH1 subroutine, the location INPCTR is set to the 1st word address of storage for the given 6 character name. Upon exit from TBSCH1 -

a. if no Loader Table entry had been encountered with a name to match the given name, then -

—0 ⇒ {SW6}

b. If a Loader Table entry with a matching name had been located, then -

0 ⇒ (SW6)

Entry to TBSCHL is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	TBSCH,I	

1.3.9.4 TBSTRL

The TBSTRL routine is used to enter a name and address into the Loader Table. Prior to entering the TBSTRL 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 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.

1.3.9.5 CNVRTL

The CNVRTL 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 CNVRTL, the binary number is placed in the A register. Upon return from CNVRTL, the locations BINASC and BINASC+1 contain the ASCII code for the 4 hexadecimal digits.

CNVRTL is entered by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EQU	CONVRT(90)	
	RTJ-	CONVRT,I	

NOTE: This subroutine is also used by PRINT4.

1.3.9.6 LCDRV1

The entry point for LCDRV1 is CDRIV. LCDRIV1 is the subroutine used for I/O operations on the typewriter. Prior to entering the LCDRV1 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 LCDRV1 subroutine is made by execution of the following instruction -

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	CDRIV	
	RTJ	CDRIV	

1.3.9.7 LMDRV1

The entry point for LMDRV1 is MDRIV. LMDRV1 is the subroutine used for I/O operations on a mass storage device. Prior to entering LMDRV1, 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.

Also, prior to entering LMDRV1, 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 LMDRV1 routine is by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	MDRIV	
	RTJ	MDRIV	

1.3.9.8 LLDRV1

The entry point for LLDRV1 is LDRIV. LLDRV1 is the subroutine which is used for output on the standard list device for the system. Prior to entry to LLDRV1, the A register is set to the starting address and Q to the number of words for the output operation. Entry to LLDRV1 is made by execution of the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT⋈	LDRIV	
	RTJ	LDRIV	

1.3.9.9 LOADR1

LOADR1 is the common subroutine used by RBLOAD, and LNKCRP to perform the functions of reading and processing relocatable binary input blocks. Entry to LOADR1 is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT⋈	LOADER	
	RTJ	LOADER	

1.3.10 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. The entry point names are as follows:

IDRIV
CDRIV
MDRIV
LDRIV

1.4.0 LIDRV1

The routine whose name is LIDRV1 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. The entry point for LIDRV1 is IDRIV.

1.4.1 Constant Table Storage Referenced by LIDRV1

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
INPLUN	115
AINPUT	101
INPUT	147

The constants are used as follows:

1. 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.
2. AINPUT contains logical unit number
3. INPUT buffer use for I/O

1.4.2 Communication Region Constants Used by LIDRV1

CONSTANT	LOCATION
$7FFF	$42 = MASK1
$8000	$21 = MASK2

1.4.3 Entry to LIDRV1

The input parameters to LIDRV1 are as follows:

1. (A) contains starting address.
2. The information pertaining to the logical unit number for the input device is stored at the 115th and 116th 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 contains the sector number for the record to be read.
4. (Q) is zero if binary, one if ASCII.

Entry is made to LIDRV1 with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT¤	IDRIV	
	RTJ	IDRIV	

1.4.4 LIDRV1 I/O Request

The input parameters for the I/O requests are handled in the following way:

1. Starting Address

   The contents 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 LIDRV1 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. The value 60 represents the maximum size record that can be read from the input device.

3. Mode

   If {Q} is a one upon entry to LIDRV1, the M parameter in the calling sequence is set for the ASCII mode of operation. If {Q} is a zero upon entry to LIDRV1, 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.

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- in this case it is an indirect request.

RTJ- {IO}

Once the operation is initiated, the location THREAD is set non-zero. The LIDRV1 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 sector number stored in PROSEC would have been increased by 1.

When loading CREP Table INPUT+58 and INPUT+59 contain next sector address of CREP Table. The last three words of the input block are cleared to zero before returning to the caller. The CREP table is a table of core resident entry points in PART 0.

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 LIDRV1.

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, LIDRV1 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 LIDRV1. If bit 9 ≠ 0, an error condition due to motion failure has occured. The A register is set to +1 to indicate the alarm condition and a jump is made to the exit from LIDRV1.

1.4.5 Exit from LIDRV1

Exit from LIDRV1 is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	(IDRIV)	

1.5.0 LCDRV1

The routine whose name is LCDRV1 is used for either input or output operations on the communication device. The entry point for LCDRV1 is CDRIV.

1.5.1 Entry to LCDRV1

The input parameters to LCDRV1 are as follows:

1. The A register contains the starting address. The A register is always assumed to be positive by the LCDRV1 routine.

2. The word count is in the Q register if LCDRV1 is to perform an output operation. The Q register is otherwise zero.

Entry to LCDRV1 is made with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	EXTM	CDRIV	
	RTJ	CDRIV	

1.5.2 LCDRV1 I/O Request

The input parameters for the LCDRV1 requests are handled in the following way:

1. Starting Request

   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 non zero, the request code 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 LCDRV1 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.

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 LCDRV1 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 LCDRV1. 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 LCDRV1. If bit 2 of the location CFLAG is a 1, the operation 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 LCDRV1.

1.5.3 Exit from LCDRV1

Exit from LCDRV1 is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	{CDRIV}	

1.6.0 LMDRV1

The routine whose name is LMDRV1 is used for either input or output operations on the mass storage device.

1.6.1 NAME USED IN DOCUMENTATION — STORAGE POSITION IN CONSTANT TABLE

PRINT2	59
MASINP	107

Constant Storage is Used as Follows:

PRINT2 — print error message

MASINP — contains logical unit for LMDRV1

1.6.2 Communication Region Constants Used by the LMDRV1 Routine

CONSTANT	LOCATION
$8000	$21 = MASK2 $F4 = IO

1.6.3 Entry to LMDRV1

The entry point for LMDRV1 is MDRIV.

The input parameter to LMDRV1 are as follows:

1. The A register contains 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 location SECTNO contains the number for the starting sector.

ERRORS

Error message generated by LMDRV1 is E1.

Entry to MDRIV is made with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	MDRIV	
	RTJ	MDRIV	

1.6.4 LMDRVL 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 LMDRVL 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 exit from LMDRVL.

1.6.5 Exit from LMDRVL

Exit from LMDRVL is made by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	{MDRIV}	

1.7.0 LLDRV1

The routine whose name is LLDRV1 is used for output on the list device.

1.7.1 Entry to LLDRV1

The entry point for LLDRV1 is LDRIV.

The input parameters to LLDRV1 are as follows:

1. The A register contains the starting address. The A register is assumed to be positive by the LLDRV1 routine.

2. The Q register contains the word count. The Q register is assumed to be positive by the LLDRV1 routine.

Entry is made to LLDRV1 with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	EXT*	LDRIV	
	RTJ	LDRIV	

1.7.2 LLDRV1 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     {IO}
```

Once the operation is initiated, the location THREAD is set to a non zero value. The LLDRV1 routine will wait in a loop until completion occurs. While the I/O operation is in progress, location THREAD remains at a non zero value. Upon completion of the operation the location THREAD is reset to zero. Also, upon completion of the operation, an exit is made from the LLDRV1 routine.

1.7.3 Exit from LLDRV1

Exit is made from LLDRV1 by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	(LDRIV)	

1.7.4 Subroutines Accessed Via Constant Table

The last paragraph of item 1.2.3 describes how a subrouting 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:

ADJOVF
CONVRT
TABSCH
TABSTR
PRINT3
PRINT2 - STOP
PRINT4
PRINT5
LINK1
CMNXIT
NXTBLK

1.8.0 ADJOF1

The ADJOF1 is a subroutine which performs 16 bit address arithmetic.

1.8.1 Constant Table Storage Referenced by ADJOF1

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
PRINT2	59
BGRLSW	9
ADJXIT	81
INPREL	11

USAGE OF CONSTANT TABLE BY ADJOVF.

PRINT2 is used to print error message. Error message from this routine is an E9.

BGRLSW upper bit is set if there is backwards relocation.

ADJXIT is used to exit from ADJOF1.

INPREL contains relative flag for a word of command sequence storage.

1.8.2 Entry to ADJOF1

Prior to entry to the ADJOF1 subroutine, the two operands for the address arithmetic are placed in the A and Q register. Entry to the ADJOF1 subroutine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	ADJOF,I	

1.8.3 Address Arithmetic

This subroutine is used primarily to add the relative value for a relocatable address to a relocation base. Upon entry to ADJOF1, the A register contains the relocatable address in bits 0-15, and the Q register contains the relocation base. Upon exit from ADJOF, bits 0-15 of the A register contain the 16 bit result for the address arithmetic together with the original setting for the sign bit.

1.8.4 Address Arithmetic for Positive Address Relocation

The address arithmetic for positive address relocation is:

relative value + relocation base - absolute value

A zero relocation flag means absolute relocation.

There will be no negative address relocation since it cannot be implemented with 16 bit arithmetic.

1.8.5 Exit from ADJOF1

Exit is made from the ADJOF1 subroutine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	ADJXIT,I	
	EQU	ADJXIT{8L}	

1.9.0 CNVRT1

The CNVRT1 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.

1.9.1 Constant Table Storage Referenced by CNVRT1

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
BINASC	51
AHOLD	136
HEX	65
CNVXIT	89

Constant storage is used as follows:

1. BINASC consisting of 2 words, is filled with the ASCII code for exactly 4 characters representing hex digits.

2. AHOLD is used to hold the bits for the binary number during the conversion process.

3. HEX 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.

4. CNVXIT is used for exit from CNVRT1.

1.9.2 Entry to CNVRT1

Prior to entry to CNVRT1, 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 CNVRT1 by executing the following instruction:

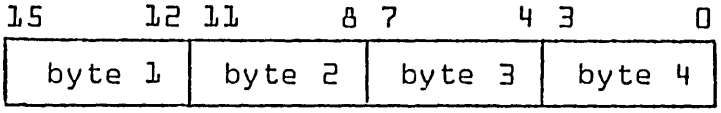
LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	CNVRT,I	

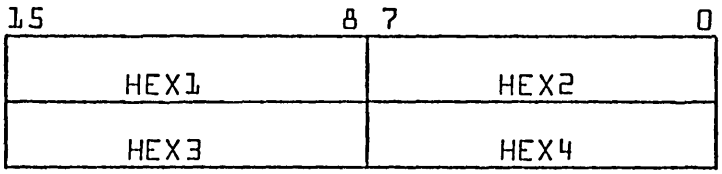
1.9.3 CNVRT1 Operation

Upon entry to CNVRT1, the A register contains a 16 bit binary number. This number is held in AHOLD during the CNVRT1 operation. The CNVRT1 will 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.

A register:





1.9.4 Exit from CNVRT1

The exit is made from CNVRT1 by execution of the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CNVXIT,I	

1.9.5 Subroutines Used by and Internal to CNVRT1

GETHEX is used by CNVRT1 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.

1.10.0 LSTOT1

There are five routines used by other subprograms of the Loader as standard routines for output operations on the standard list device for the system. The five subroutines are:

PRINT3
PRINT2
PRINT4
PRINT5
PRINT6

1.10.1 PRINT3

PRINT3 is a subroutine used for printing error messages. The error messages consist of the letter E followed by 1 or 2 hexadecimal digits. Operation within the Loading Procedure may be resumed immediately following the error message output.

1.10.1.1 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	

1.10.1.2 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 LLDRV1 where the address LDRIV is declared as an external.

1.10.1.3 Exit from PRINT3

Exit from the PRINT3 routine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PR3XIT,I	
	EQU	PR3XIT{53}	

1.10.1.4 Subroutines Used by and External to PRINT3

LDRIV is used for output on the list device. This routine is entered with a return jump to its entry point whose name is LDRIV.

1.10.2 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.

1.10.2.1 Entry to PRINT2

The A register is set prior to entering PRINT2 in the same manner as described for PRINT3. Entry to PRINT2 is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PRINT2,I	

1.10.2.2 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.

1.10.2.3 Exit from PRINT2

Exit is made from STOP with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CMNXIT,I	

where the name CMNXIT is declared as an entry point in the LOAD routine.

1.10.2.4 Subroutines Used by and Internal to PRINT2

PRINT3 which is entered for error message output with a return jump to the address = 54+{I}.

1.10.3 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.

1.10.3.1 Constant Table Storage Referenced by PRINT4

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
CONVRT	90
BINASC	51
PR4XIT	93

The constants are used as follows:

1. BINASC locations are used for storage of hex digits after number conversions have binary to ASCII.

2. PR4XIT is used to exit from PRINT4.

3. CONVRT is used to convert hex word to ASCII.

1.10.3.2 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 characters of the output. PRINT4 is entered with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	PRINT4,I	

1.10.3.3 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 and 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 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.

1.10.3.4 Exit from PRINT4

Exit from PRINT4 is with the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PR4XIT,I	
	EQU	PR4XIT{93}	

1.10.3.5 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 = 90+{I}

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.

1.10.4 PRINT5

The PRINT5 routine is used to print a six character name.

1.10.4.1 Constant Table Storage Referenced by PRINT5

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
INPCTR	15
PR5XIT	97

INPCTR contains the 1st word address for storage of the 6 character name to be printed.

PR5XIT is used to exit from PRINT5.

1.10.4.2 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	

1.10.4.3 Output Message Generation

The address in INPCTR is placed in the A register. The length of the output is placed in the Q register (4 words). A return transfer is made to LDRIV where this name is referred to as an external address. A transfer to the exit from PRINT5 is made upon return from LDRIV.

1.10.4.4 Exit from PRINT5

Exit is made from PRINT5 with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PR5XIT,I	
	EQU	PR5XIT(97)	

1.10.4.5 Subroutines Used by and External to PRINT5

LDRIV is used to print the 6 character name.

1.10.5 PRINT6 is used to put the external name into the print buffer and print the external name on the list device. PRINT6 use LDRIV to list the external name on the list device.

1.10.6 STOP is another routine in LSTOUT used by the loader. STOP sets {A} and {Q} to minus zero and exits by the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CMNXIT,I	

1.11.0 LINK11

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 LINK11 subroutine will replace each of the link addresses in a string with the entry point address for the particular name. When $7FFF is encountered the link table is checked for an address of the $7FFF in the command sequence. If a match is found the $7FFF is not the end of the string and the search for the end of string continues as $7FFF is a pointer to the next link address. The link table contains the address of any program relocation value that has been absolutized to $7FFF.

1.11.1 Constant Table Storage Referenced by LINK11

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
ENTPNT	13
LINK	14
INPWRD	10
NGRLSW	9
PRINT2	59
LK1XIT	103
LNKSTR	112
LNKCTR	113

1.11.2 Constant Table Storage is Used as Follows:

1. ENTPNT — contains the absolute value of the entry point address in bits 0-15.
2. LINK — contains the absolute value of the 1st link address in a string.
3. INPWRD — is used as temporary storage during program execution.

4. NGRLSW — is set positive or negative prior to entry to LINK11. Determines whether patching is for absolute or relative addressing.

5. PRINT2 — is used to print error messages.

6. LK1XIT — is used to exit from LINK11.

7. LNKSTR — contains the address of link table.

8. LNKCTR — points to next available location in link table.

1.11.3 Communication Region Constants Used by LINK11

CONSTANT	LOCATION
$7FFF	$42 = MASK1
$8000	$21 = MASK2

1.11.4 Entrance to LINK11

Both LINK and ENTPNT are set prior to entering LINK11. Prior to entry to LINK11 the location NGRLSW is set either positive or negative. The location NGRLSW is set with {{INPCTR}} = 2 characters of the name. LINK11 is then entered with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	LINK1,I	

1.11.5 Patching

Upon entry to the subroutine LINK11, 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. If the contents of NGRLSW is positive, patching for the absolute mode of addressing will occur.

LINK11 will replace the contents of every location in the string of the link addresses by the entry point address in ENTPNT.

{ENTPNT} ⇒ {LINK} at bits 0-14, bit 15 of {LINK} is not changed

{INPWRD} ⇒ {LINK}

If the contents of NGRLSW are negative, LINK11 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:

bits 0-14 of {{LINK}} ⇒ INPWRD

{ENTPNT} ⇒ {LINK} {LINK} at bits 0-15

{INPWRD} ⇒ LINK

LINK11 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 LINK11 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.

1.11.6 Internal Subroutines Used by LINK11.

LNKCK compares link table entry to the address of the $7FFF link. If an equivalent address is found, $7FFF is a pointer to the next link address, otherwise it is the end of the string. If $7FFF is found to be the end of string an exit is made from LINK11 through CMNXIT.

1.11.7 Exit from LINK11.

Exit is made from the LINK11 routine using the following instructions:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	LKIXIT,I	
	EQU	LKIXIT{103}	

1.12.0 LOADR1

The functions of the LOADR1 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.

1.12.1 Constant Table Storage Referenced by the LOADR1 Routine

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
CSQCTR	123
DATBAS	2
DATLIM	5
INPXC0	57
INPUT	147
NXTINP	63
PRINT2	59
PRINT3	54
SECTOR	138
CSQLIM	6
PARBAS	131
MSDWCT	134
CSQSEC	128
STRSEC	133

1.12.2 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. DATBAS — relocation base for data storage.
3. DATLIM — highest address of data storage +1.
4. INPXC0 — 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 LOADR1.

6. NXTINP — is used to get next block

7. PRINT2 — is used to print error message.

8. PRINT3 — is used to print error message

9. CSQLIM — highest address of command sequence storage +1.

10. SECTOR — contains the word count for a sector on mass storage. Word count = 96.

11. PARBAS — address of start of partition.

12. MSDWCT — number of words stored on mass memory.

13. CSQSEC — starting sector of command sequence image.

14. STRSEC — start sector of image on mass memory.

1.12.3 Communication Region Constants Used by LOADR1

CONSTANT	LOCATION
$7FFF	$42 = MASK1

1.12.4 Entry to the LOADR1 Routine

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ	LOADER	

where -

the name LOADR1 is referenced as an external within the program from which the return jump is made, and as an entry point name within the LOADR1 routine.

1.12.5 Reading Input Records

At the location SW1AXC, 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, an 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. All 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.

1.12.6 Reading Relocatable Binary Input Blocks

The LOADR1 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 LOADR1 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 starting address.

3. The Q register is set to zero for binary mode.

4. 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 1st relocatable binary input block, and subsequent to reading each XFR block, the address for the jump instruction at SW1 is set to SW1A.

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

$$\{SW1+1\} = SW1C - SW1 - 1$$

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 °E1° is printed and loading terminates.

1.12.7 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 character 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. Sequential ASCII records need not occur on a continuous external medium. The records are read {°¤T carriage return°} or until the alarm condition arises.

Refer to item 1.12.6 for the procedure to be followed should an alarm condition arise.

1.12.8 Types of ASCII Records

1. ¤T carriage return is an End of Load Statement.

1.12.9 Branching to Process an Input Block

The contents of the first word in the input buffer determines branching. Branching is to process relocatable binary input and it is determined by the bits {INPUT}
15 13.

1.12.10 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 and gives you an E3. If the NAM block is received, the loader will reset the jump address in SW1 to SW1B and then jump to NAMPR0 {refer to 1.13}.

The address NAMPR0, is feferenced as an external name by jump instructions within the LOADR1 Routine.

1.12.11 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 RBDPR0, BZSPR0, ENTPR0, EXTPR0 and XFRPR0 accordingly. The loader will jump to the PRINT2 error exit each time a NAM, block is received while the SW1 jump instructions within the LOADR1 routine.

1.12.12 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.

1.12.13 SW1F

Subsequent to the most recent return jump to LOADR1, 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.

1.12.14 Exit from the LOADR1 Routine.

Exit from the LOADR1 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 LOADR1 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
	JMP-	PRINT2,I	

1.12.15 Subroutines Used by and External to the LOADR1 Routine

1. PRINT2 is used for error message output. Operation is terminated following printout.

POSSIBLE ERRORS.

E1
E3

2. MDRIV is used to write command sequence storage onto mass storage. Also, it is used to write the Loader Table onto mass storage.

3. IDRIV is used to read input records from the input device.

The PRINT2, routine is entered via the entry point location in the Constant Table.

IDRIV is entered with return jump to its entry point address.

1.13.0 NAMPR1

NAMPR1 is the routine which is used to process NAM block.

1.13.1 Constant Table Storage Referenced by the NAMPR1 Routine.

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
AINPUT	101
BLANKS	18
COMBAS	1
COMLIM	4
CSQLIM	6
DATBAS	2
DATLIM	5
ENTPNT	13
INPCTR	15
INPUT	147
PROBAS	3
PRODAT	120
PARLIM	132
ASAV	29
BINASC	51
INPXC0	57
INPXC1	58
INPXCC	62
NXTINP	63
PRINT3	54
PRINT4	94
PRINT5	98
QINPUT	102
PROGCT	146
PROCOM	121
PRINT2	59
QSAV	30
SW6	28

1.13.2 The constants are used as follows:

1. AINPUT — contains one of the entrance parameters for the current Loader Operation. {Refer to item 1.2}
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. 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.)

6. DATBAS — contains the relocation base for the Execution Time Data Storage Block Reservation.

7. DATLIM — contains the upper limit address (equal to last address +1) of the Execution Time Data Storage Block Reservation.

8. ENTPNT — contains the address which is to be inserted into the 4th word of a Loader Table entry.

9. 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.

10. INPUT — is the 60 word buffer for storage of relocatable binary input.

11. PROBAS — contains the Execution Time Relocation Base for the program whose NAM block is currently being processed.

12. QSAV — is used for temporary storage of operands in the Q register.

13. 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.

14. PRODAT — non zero if protected data is delcared.

15. PARLIM — LWAD+1 of last partition

16. ASAV — tempory storage for a register.

17. BINASC — temporary storage for words converted from binary to ASCII.

18. INPXC0 — address of input buffer.

19. INPXC1 — address of input +1.

20. INPXCC — address of input -3.

21. NXTINP — used to get next block.

22. PRINT3 — used to print errors.

23. PRINT4 — used to print 6 character name and 4 digit hexadecimal address.

24. PRINT5 — used to print a six character name.

25. QINPUT — (Q) is stored here upon entry to the Loader.

26. PROGCT — number of words in program.

27. PROCOM — non zero if protected common is declared.

28. PRINT2 — used to print error

1.13.3 Entry to the NAMPR1 Routine

The name NAMPR1 is declared as an entry point name in the NAMPR1 routine and is an external in the LOADR1 routine. Entry to the NAMPR1 routine is with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	NAMPR0	

1.13.4 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 printer. In addition, the six character program name in the NAM block is printed on the printer together with the base address of the program.

1.13.5 Reserving Common Storage

The value equal to the number of words to be set aside for common storage is contained in the second word of the input buffer, INPUT+1. If {INPUT+1} is zero, common storage is not reserved. Instead, the program transfers immediately to NAMPR1.

If {INPUT+1} is non zero, {COMBAS} is checked. 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, common storage will be reserved 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.

Once common storage has been reserved PROCOM is checked to determine if common is protected. If PROCOM is zero common is loaded in the last partition. If common is protected core overflow is ignored, otherwise NAMPR1 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, a jump is made to NAMPR1 to further process the NAM block. If not, a transfer to DATERR which

transfers to PRNNAM. PRNNAM transfers to PRINT4 which lists name and relocation. Upon return from PRNNAM, a jump is made to PRINT2 to print the error indication °E4°.

1.13.6 Reserving Data Storage

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, no data storage is reserved. Instead, the program transfers immediately to NAMPR2.

If {INPUT+2} is non zero, {DATBAS} is checked.

If {DATBAS} is zero, {PRODAT} is checked.

If {PRODAT} is non zero it indicates Data is to be loaded in the partition.

If {INPUT+2} is zero {INPUT+3} is checked for command sequence.

Once data is declared, {DATBAS} is set to the execution time relocation base of data storage. Data storage is reserved by assigning available space in the command sequence storage:

1. {PROBAS} ⟶ DATBAS

2. {DATBAS}+{INPUT+2} ⟶ DATLIM ⟶ PROBAS ⟶ CSQLIM

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, a check is made 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} $\leq$ PARLIM 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} if {COMBAS} $\neq$ 0

If neither of these conditions holds true, an error has occured 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.

Space is reserved in execution time core for the Execution Time Data Storage Block Reservation:

{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.

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.

1.13.7 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} ≠ 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} ≤ {COMBAS} if {COMBAS} ≠ 0

or

{PROBAS}+{INPUT+3} ≤ {PARLIM} 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} ⟶ CSQLIM.

1.13.8 NAMPR3 - Print Program Name and Execution Time Relocation Base

A test is made to determine if the current Loader Operation is a Crep Table Load Operation. The program name and 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 a Crep 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 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

SSSSXXXXXXSShhhh

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}.

1.13.9 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 {refer to 1.12.6} to read and process the next input block. If overflow has occurred, an error indication of °E5° is printed using the PRINT2 subroutine. No more relocatable binary input blocks are read as Loading terminates. The error exit parameters will be recorded in AINPUT and @INPUT.

1.13.10 Exit From NAMPR1

If no core overflow had occurred, exit from NAMPR1 is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	NXTINP,I	JUMP TO READ NEXT INPUT BLOCK
	EQU	NXTINP{63}	

1.13.11 Subroutines Used by and External to NAMPR1

1. PRINT2 is used to print error indications.
2. PRINT4 is used to print the program name and execution time relocation base of the program currently being read in.

1.13.12 Subroutines Used by and Internal to NAMPR1.

1. PRNNAM is used to list name and relocation.
2. SIGNCK this routine is entered with the upper limit to tested in {A} and lower limit in {Q}.

1.14.0 RBDBZL

RBDBZL is the routine which is used to process RBD and BZS blocks.

1.14.1 Constant Table Storage Referenced by RBDBZS

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
ASAV	29
BLKCNT	27
BZSSW	26
COMBAS	1
COUNTL	25
CSQLIM	6
DATBAS	2
DATLIM	5
ENDSW	8
INPCTR	15
INPREL	11
INPWRD	10
INPXCO	57
PROBAS	3
QSAV	30
WRDCNT	24
ADJOVF	82
AINPUT	101
PRODAT	120
PROCOM	121
PARLIM	132
COMLIM	4
NXTINP	63
PRINT3	54
PROGCT	146
QINPUT	102
SW6	28
WRDCNT	24
PRINT2	59
LNKSTR	112
LNKCTR	113
LNKEND	114

1.14.2 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.

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. DATLIM — contains the upper limit address (equal to last address + 1) of the Execution Time Data Storage Block Reservation.

9. 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.

10. INPCTR — contains the execution time address assigned to a word from the command sequence input.

11. 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.

12. INPWRD — contains a command sequence word from an input block entry.

13. 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.

14. PROBAS — contains the Execution Time Relocation Base for the program currently being loaded.

15. ØSAV — is used for temporary storage of operands in the Ø register.

16. WRDCNT — is used as an address counter to reference words in the input block (either RBD or BZS) entries.

17. ADJØVF — address of jump to ADJØVF.

18. AINPUT — (A) is stored here upon entry to the loader.

19. PRØDAT — is non zero if protected data is declared.

20. PRØCØM — is non zero if protected common is declared.

21. PARLIM — LWAD+1 of last partition.

22. CØMLIM — highest address of common storage+1.

23. NXTINP — is address of jump to NXTBLK.

24. PRINT3 — is used to print error messages.

25. PRØGCT — is the number of words in a program.

26. ØINPUT — (Ø) is stored here upon entry to the loader.

27. SW6 — is used as a switch for relocation type.

28. WRDCNT — address of character reference counter.

29. PRINT2 — is used to print error messages.

30. LNKSTR — is the address of LNKTBL.

31. LNKCTR — pointer to next available location in LNKTBL.

32. LNKEND — last address +1 in LNKTBL.

1.14.3 Entry to RBDBZ1

Both names, RBDPRØ and BZSPRØ are declared as entry point names in the RBDBZ1 routine and as externals in the LØADR1 routine. Entry to either of these routines is as follows:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	RBDPR0	
	JMP	BZSPR0	

1.14.4 RBDPR0

Command sequence data from an RBD block is placed in core by RBDPR0. 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 bits in size. Zero is the leading bit for all but the last relocation byte. The leading bit of the relocation byte for the last entry in the block is set to a one.

1.14.5 Initialization for RBD Block Processing at RBDPR0

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 RBDPR0, 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 RBDPR0 and BZSPR0 in order to extract all the information pertinent to one entry in the input buffer.

1.14.6 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 startine address.
2. INPREL contains a 2 bit relocation byte.
3. 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} = 10, {INPWRD} is a value relative to the common storage relocation base of {COMBAS}.

4. 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 data storage relocatable or common storage relocatable, the addresses for the jump instructions SW2 and SW3 are reset to SW2B and SW3B respectively. To get the starting address for relocation, a return jump is made to ADJUST subroutine.

If the starting address is either data relocatable, absolute, or common relocatable 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 {CSALIM} is placed in ASAV.

2. Branching to SW2B: The execution time limit address for data storage = {DATLIM} is placed in ASAV.

3. Branching to SW2C: The limit address communication region storage = °$E4° is placed in ASAV.

1.14.7 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 NXWRD -

1. INPWRD contains a word of the command sequence.

2. INPREL contains the 2^0 and 2^1 bits of the 4 bit relocation byte.

3. 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.

1. {INPREL} = 00 is {INPWRD} is an absolute value.

2. {INPREL} = 01 is {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	INPREL	VALUE ADJUSTED FOR RLCTN
ABSOLUTE	00	{INPWRD}
POS PROGR. RLCTN	01	{INPWRD}+{PROBAS}
POS COM STOR RLCTN	10	{INPWRD}+{COMBAS}
POS DAT STOR RLCTN	11	{INPWRD}+{DATBAS}

Negative relocation is not implemented in the PART1 LOADER because of 16 bit arithmetic.

The address arithmetic necessary for adjusting a relative value for relocation is accomplished by the closed subroutine ADJOF1. The closed subroutine is entered by a return jump to a location in the constant table.

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}≠0 a jump is made to NXTINP to process the next input block.

If {INPCTR} ≥ {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 execution time core for program relocatable command sequence currently being loaded. An error indication of °E5° is printed using the PRINT3 subroutine.

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 an attempt to load information into protected core. 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.

1.14.8 BZSPR0

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 bytes except 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.

1.14.9 Initialization for BZS Block Processing

As part of initialization for BZSPR0, 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 -1 in order to process a BZS block.

1.14.10 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 BZSPR2, 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. ENDSW contains the 2^3 bit of the 4 bit relocation byte.
4. 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 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 4 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} = 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 value for the relocation base of the Execution Time Data Storage Block Reservation.

1.14.11 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. 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 SW2A is reset to SW2B while that in SW3 is set to SW3B.

3. If the starting address for the BZS block is absolute, the address in jump instruction SW2, originally set to SW2A is reset to SW2C while that in SW3 is set to SW3C.

1.14.12 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 SW2A: The execution time limit address for program relocatable input or {CS@LIM} is placed in ASAV, and a jump is made to BZSPR2.

2. Branching to SW2B: The execution time limit address for data storage = {DATLIM} is placed in ASAV, and a jump is made to BZSPR2.

3. Branching to SW2C: The limit address for communication region storage = ▽$E4▽ is placed in ASAV, 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. 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 SW3A will occur if the starting address in the BZS entry is relative to the execution time program relocation base.

2. Branching to SW3B will occur if the starting address is relative to the relocation base for the data block.

3. Branching to SW3C will occur if the starting address is absolute.

If a 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.

1.14.13 Exit from RBDBZ1

Exit from either routine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	NXTINP,I	JUMP TO NXTINP TO READ

1.14.14 Subroutines Used by and Internal to RBDPR0 and BZSPR0

There are two subroutines assembled into the same program with RBDPR0 and BZSPR0. These are NXTWRD and ADJUST.

1.14.15 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 RBDPR0, the the switch BZSSW has been set to zero. If entry to NXTWRD is from BZSPR0, 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 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 3 is placed in ENDSW.

2. The size of the BZS block reservation is placed in BLKCNT.

If INPREL is zero it means that absolute relocation is used. If absolute relocation is not used a check is made for $7FFD or $7FFE. The FORTRAN COMPILER generates these numbers for backwards 15 bit relocation in FLOT CALLS. Bit 15 of these two values must be set to cause 16 bit address arithmetic to generate an end around carry.

BACKWARDS RELOCATION - BGRLSW is set to zero if backwards relocation occurs otherwise it is non-zero. For backwards relocation 16 bit address arithmetic must generate an end around carry to simulate 15 bit wrap-around.

1.14.16 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 16 bit address arithmetic is carried out in the ADJOFL subroutine. Prior to entering ADJOFL, the A register contains the relative address contained in INPWRD. The relocation base is in the Q register. Upon return from ADJOFL, 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 RBDPR0 or BZSPR0,

      SW2C ⟶ SW2, and

      SW3C ⟶ SW3.

   Following this a jump is made from ADJUST.

2. Program Relocatable Addresses

   Following a return transfer to ADJ0VF, the relative address is increased by the command sequence relocation base, and the result is placed in INPCTR:

   {INPWRD} + {PR0BAS} ⟶ INPCTR

   A jump is made to exit from ADJUST.

3. Data Storage Relocatable Addresses.

   Following a return transfer to ADJ0VF, 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 RBDPR0 or BZSPR0,

      SW2B ⟶ SW2, and SW3B ⟶ SW3.

1.14.17 STRLNK

This routine checks to see if a command sequence word is program relocatable and a check is made to see if the LINK TABLE is full. If the link table is full and E15 error is printed and the load operation terminates. If the command sequence word contains $7FFF its address is put in the link table.

1.14.18 LINK TABLE

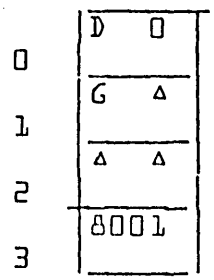
The link table contains the addresses of any program relocation value that has been absolutized to $7FFF. The LINK11 routine uses the link table to determine if a $7FFF is a true end of string when patching. Both the assembler and compiler use $7FFF to indicate the end of string of an external to a program.

```
EXAMPLE:        EXT     DOG
                LDA+    DOG
                STA+    DOG

                 0      C400
                 1      7FFF X The end of string has no relo-
                                  cation
                 2      6400
                 3      0001 X This pointer is PGM relocatable.
```

The execution time or partition base is $7FFE. The command sequence is absolutized to look like this:

```
EXECUTION TIME    ADDRESS        CODE

                  $7FFE          C400
                  $7FFF          7FFF X
                  $8000          6400
                  $8001          7FFF X This location points back
                                        to location 7FFF.
```

The entry for DOG in the external table looks like this:



Points to first location of link string in program.

The LINK table contains an $8001 indicating that the $7FFF stored in the location with the execution time address of $8001 is an address pointing to the absolute location of $7FFF.

1.14.19 Subroutines Used by and External to RBDPR0 and BZSPR0

1. PRINT3 is used for printing error indications.
2. ADJ0VF is used by ADJUST for address arithmetic.

1.15.0 XFRPR1

XFRPR1 is the routine which processes an XFR block.

1.15.1 Constant Table Storage Referenced by XFRPR1

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
AINPUT	101
BLANKS	18
CS@LIM	6
ENDINP	17
INPUT	147
PROBAS	3
XFRNAM	32
NXTINP	63

1.15.2 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. NXTINP contains address of NXTBLK.

4. CS@LIM 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. ENDINP contains the highest load time address for command sequence storage.

6. INPUT is the 1st location of a 60 word area reserved for storage of relocatable binary input.

7. PROBAS is set to the execution time relocation base for the next relocatable binary program to be loaded.

8. XFRNAM contains the 6 character transfer name from the last XFR block processed which had a name.

1.15.3 Entry to XFRPR1

The name XFRPR1 is declared as an entry point name in the

XFRPR1 routine and as an external in the LOADR1 routine. Entry to XFRPR1 is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	XFRPR0	

1.15.4 Recording Transfer Name

When the Loader is brought into core, the locations XFRNAM, XFRNAM+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, XFRPR1 replaces the current contents of XFRNAM, XFRNAM+1 and XFRNAM+2 with the ASCII code for this 6 character name.

1.15.5 Recording Relocation Base for Next Program

Prior to the exit from XFRPR1:

1. {CS@LIM} ⟶ PROBAS where {PROBAS} = execution time relocation base of next program to be read in.

1.15.6 Exit from XFRPR1

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 XFRPR1 is then made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	NXTINP,I	GO TO NXTINP TO BEGIN

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. an EOL block
3. if no data is read by the Loader, it is the same as if an EOL block had been received.

If the Loader Operation is either a Subroutine Load or a Program Load, exit from XFRPR1 is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	ALRM0K	

where the name ALRM0K is referenced as an external in XFRPR1 and as an entry point in the LOADR1 routine.

1.16.0 ENTEX1

The ENTEX1 subprogram of the Loader is used to process EXT and ENT blocks.

1.16.1 Constant Table Storage Referenced by ENTEX1

NAME USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
ENTPNT	13
INPCTR	15
INPWRD	10
INPXCC	62
PROBAS	3
SW6	28
EXTSWT	141
IGNORE	111
NXTINP	63
PRINT3	54
PRINT5	98
TABSCH	40

1.16.2 The constants are used in the following way:

1. ENTPNT — contains the address which is to be inserted into the 4th word of a Loader Table entry.
2. INPCTR — is used as an address counter to reference entries in the input block (either ENT or EXT).
3. INPWRD — is used as a temporary storage location.
4. INPXCC — contains the address constant = ᵛINPUT+3ᵛ.
5. PROBAS — contains the execution time relocation base for the program currently being loaded.
6. SW6 — if positive it indicates a match was found in TABSCH.
7. EXTSWT — non-zero if processing EXT block.
8. IGNORE — flag to ignore duplicate entry points during crep table load.
9. NXTINP — contains jump to NXTBLK routine in loader.

10. PRINT3 used to print error messages.

11. PRINT5 used to print 6 character name.

12. TABSCH contains jump to TABSCH.

1.16.3 Entry to ENTEX1

The names ENTPR0 and EXTPR0 are declared as entry point names in the ENTEX1 routine and as external names in the LOADR1 Routine. Entry to either of these routines is as follows:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	ENTPR0	
	JMP	EXTPR0	

1.16.4 ENTPR0

An entry in an ENT block contains a 6 character entry point name and a 15 or 16 bit entry point address. ENTPR0 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 entry point name from the input block. ENTPR0 will enter the entry point name and address into the loader entry point table.

CONDITION 2: is that there exists an entry in the Loader entry point table which contains a 6 character entry point name which matches the 6 character entry point name in the input block. The loader outputs a message and ignores the duplicate entry point.

1.16.5 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 an entry point name equal to zero.

INPCTR is used by the closed subroutine NXTNAM to extract entries from the ENT block. Initially INPCTR is increased by 4 upon each entry to NXTNAM from ENTPR0. This will set INPCTR to the first word address of the next entry in the ENT block.

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. NXTINP branches to NXTBLK in the LOADR1 module.

1.16.6 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. SW6=0 is a match has been found.
2. If a matching name had not been found, {SW6} is negative. HSHADR points to address where entry point name will be stored.

1.16.7 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.

1.16.8 EXTPR0

An entry in an EXT block contains a 6 character external name and a 15 or 16 bit link address.

1.16.9 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.

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 EXTPR1. The contents of INPCTR is increased by 4 upon each entry to NXTNAM from EXTPR1. This will set INPCTR to the first word address of the next entry in the EXT block. The routine NXTNAM is used to extract entries from the EXT block. 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.

1.16.10 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

1.16.11 Subroutines Used by and Internal to ENTPR0 and EXTPR0

The NXTNAM routine is used by both ENTPR0 and EXTPR0 to extract the next entry to be processed from the input block.

1.16.12 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	

Address ——— is the entry point address when processing entry point names if an externa. block is being processed, address is the link address of the name.

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} ——— 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 ——— INPCTR

A jump is made to the exit from NXTNAM.

1.16.13 Subroutines Used by and External to ENTPR0 and EXTPR0

1. TABSCH is used to search the Loader Table for a name to match the name in the input block entry being processed.

2. ENTSTR is used to enter a name from the input block entry being processed into the Loader Table.

3. EXST0R is used to get address of next available slot in External Table.

4. PRINT3 is used by ENTPR0 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.

5. PRINT5 is used by ENTPR0 to print the entry point name causing the error described in 4.

1.17.0 STBASE is used to initialize most of the constants in the constant table. The routine that references STBASE is BRNCH1. STBASE is entered by the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	STBASE	

STBASE is declared as an external in the BRNCH1 module.

1.17.1 NAMES USED IN DOCUMENTATION — STORAGE POSITION IN CONSTANT TABLE

CS@NUM	12
COUNT1	25
PGELGN	109
BASE	23
NOPAGE	130
CMNXIT	139
TEMP	149
ADDR	117
ENTSEC	127
OVSTR	124
OVCTR	125
EXTSTR	118
EXTCTR	7
CS@SEC	128
MAXPGE	129
PRINT2	59
CS@LIM	6
LNKSTR	112
LNKCTR	113
LNKEND	114
@INPUT	102
PARBAS	131
PROBAS	3
PARLIM	132
DATBAS	2
PRODAT	120
DATLIM	5
PROCOM	121
COMLIM	4
COMBAS	1
CORADR	119

1.17.2 The constants are used as follows:

1. CSØNUM is the number of sectors reserved for entry external tables before command sequence start.
2. COUNT1 is used to store the complement of the number of pages.
3. PGELGN is the length of page with core flags.
4. BASE is the base address of the loader.
5. NOPAGE is the number of pages in unprotected core.
6. CMNXIT is used to set up exit parameters and to exit program.
7. TEMP is used as a temporary storage location.
8. ADDR is used for the address of command sequence page.
9. ENTSEC is the starting sector of ENT/EXT tables.
10. OVSTR word address of overflow table start.
11. OVCTR point to next available location in overflow table.
12. EXTSTR word address of start of EXT table.
13. EXTCTR points to next available location in EXT table.
14. CSØSEC starting sector of command sequence image.
15. MAXPGE maximum page number that can be used on mass memory.
16. PRINT2 is used to print error messages and terminate load.
17. CSØLIM highest address of command sequence storage +1.
18. LNKSTR is the address of the link table.
19. LNKCTR is the next available location in the LINK TABLE.

20. LNKEND is the last address +1 in the LINK TABLE.

21. @INPUT @ is stored here upon entering the loader. @ input contains the core address of the program bases.

22. PARBAS is address of the start partition to be loaded.

23. PARLIM LWAD+1 of last partition to be loaded.

24. DATBAS is the relocation base for data storage.

25. PROBAS is the relocation base for the program currently being loaded.

26. PRODAT is non zero if protected data is declared.

27. DATLIM is the highest address of data storage+1.

28. PROCOM is non zero if protected common is declared.

29. COMLIM is the highest address of common storage+1.

30. COMBAS is the relocation base for common storage.

31. CORADR contains {$F7}+1.

1.17.3 Entry to STBASE

Prior to entry to STBASE, @INPUT is set to the core address of the first four sequential locations containing the base address used by the loader to set limits.

1.17.4 Constants set by STBASE are set in the following manner:

Contents of the address in @INPUT ⇒ PARBAS.

PARBAS+2 ⇒ PROBAS

PROBAS ⇒ CS@LIM

Last address +1 of end of partition ⇒ PARLIM.

DATA BASE ⇒ DATBAS if not zero. If data base is zero it is assigned within the partition.

LVLSTR is the start of allocatable core for level 0

LVLSTR ⇒ PRODAT

PRODAT ⇒ DATLIM

LVLSTR can be found in the core allocator table in SYSBUF.

If common is system common

COMMON BASE ⇒ PROCOM

PROCOM ⇒ COMBAS

TOP of part 0 ⇒ COMLIM

If no system common is declared.

PARLIM ⇒ COMLIM

LOWEST unprotected location -1 ⇒ CORADR

CORADR ⇒ ADPAGE

{BASE OF LOADER - START OF UNPROTECTED} ÷ 192 → NOPAGE.

NOPAGE ⇒ COUNT1

If no pages are available a jump to PRINT2 is executed to print an E11 and terminate the load.

0 ⇒ ADDR {address of command sequence page}

-COUNT1 ⇒ COUNT1

Initialization of core flags is done in the following way:

ADDR ⇒ NUMBER

0 ⇒ REFER

REFER ⇒ MODIFY

This is done in a program loop to initialize the core flags of all the pages in core {refer to 1.25.3}.

BASE OF ENT/EXT TABLES ⇒ ENTSEC

CSQNUM + ENTSEC ⇒ CSQSEC

A check is made on mass memory to see if there is room for the command sequence pages.

1.17.5 To exit from STBASE the following instruction is executed:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CMNXIT,I	EXIT

1.18.0 LNKENT

LNKENT uses a return jump to patch all program externals and a return jump to TABSCH to search the loader table for a matching transfer name. On return from TBSCH, SW6 is positive if a match is found in the loader table. If SW6 is negative on return from TABSCH an E13 is printed and the load terminates. LNKENT stores the transfer address in the second word of command sequence.

1.18.1 NAMES USED IN DOCUMENTATION — STORAGE POSITION IN CONSTANT TABLE

XFRNAM	32
XFRADR	135
PARBAS	131
SW6	28
TABSCH	40
INPCTR	15
ENTPNT	13
PRINT2	59

1.18.2 The constants are used in the following way:

1. XFRNAM — is used for storage of a six character transfer name.
2. XFRADR — is the transfer address of the name from the last transfer block encountered.
3. PROBAS — is the address of the start partition of the load.
4. SW6 — is used as a switch indicator upon return from TABSCH. SW6 is positive if a match is found.
5. TABSCH — address of TABSCH in LOAD.
6. INPCTR — is used to address location of command sequence storage.
7. ENTPNT — contains address associated with name in ENT or EXT table.
8. PRINT2 — is used to print error messages and terminate loading.

External Subprograms Used by LNKENT.

PATCH	is used to patch all program externals.
TABSCH	is used to search table for a matching transfer name. If no entry is found an E13 is printed and the load terminates.
DISKWR	is used to store transfer address in the second word of the command sequence.

1.18.3 Exit From LNKENT

The following instruction is used to exit LNKENT:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	EXIT	

Where EXIT is declared external. EXIT is in the LNKCR1 module.

1.19.0 LNKCR1

LNKCR1 is used to link the CREP1 table to program externals. The CREP1 table is a table of core resident entry points in PART 1. The first location of the entry point table contains the number of sectors the table uses. The end of the table and the end of the sector will be indicated by minus zero.

LNKCR1 uses TABSCH to search the entry point table for a match with the CREP1 entry points. On return from TABSCH, SW6 is checked to see if a duplicate entry point was found. If a duplicate entry point is found in the Loader entry point table while loading the CREP1 table, the CREP1 entry point is ignored and no diagnostic is issued.

1.19.1 NAMES USED IN DOCUMENTATION

	STORAGE POSITION IN CONSTANT TABLE
CMNXIT	139
SAVEA	142
MASINP	107
INPUT	147
INPXC0	57
INPCTR	15
TEMP3	144
ENTPNT	13
TABSCH	40
SW6	28
NOTLNK	16

1.19.2 The constants are used in the following way:

1. CMNXIT — is used to exit from this program and to store exit parameters.
2. SAVEA — is used as a temporary storage location for sector of CREP1 table.
3. MASINP — contains logical unit for MDRIV. LNKCR1 sets MASINP to $8C2.
4. INPUT — is used as an input buffer.
5. INPXC0 — contains the address of the input buffer.
6. INPCTR — is used to address locations of command sequence storage.

7. TEMP3 is used as a temporary storage location for the number of sectors in the CREPL table.

8. ENTPNT contains address associated with name in ENT or EXT tables.

9. TABSCH address of a jump to TABSCH.

10. SW6 is used to indicate if a match is found on return from TABSCH.

11. NOTLNK is a one if unpatched externals are found.

1.19.3 External Subprograms Used by LNKCRL.

1.19.3.1 TABSCH is used to search entry point table for a match. Prior to the return jump to TABSCH the (A) register is set to the entry point value to be searched for.

1.19.3.2 ENTSTR is used to store an entry in the HASH TABLE or OVERFLOW TABLE.

1.19.3.3 PATCH is used to patch CREPL table with externals.

1.19.3.4 WRTOUT is used to write out pages to mass memory if there are no unpatched externals.

1.19.3.5 MDRIV is used to read in a sector of CREPL table from mass storage. MDRIV is referenced in the subroutine READIN.

1.19.4 Internal Subroutines Used by LNKCRL

1.19.4.1 READIN is used to read a sector of the CREPL table into the input buffer. The last location +1 of the input buffer is set to a minus zero to indicate the end of the sector.

1.19.5 Exit from LNKCRL

To exit from LNKCRL the following instruction is executed:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	CMNXIT,I	

1.20.0 BLDCR1

This routine picks up the PART 0 external references to core resident PART 1 programs and puts them in the external table. A return jump is made to patch externals and squeezes entry points together in the Entry table.

The CREP1 table is stored behind the command sequence on the disk. LIBEDT will move the table to an appropriate place on the disk and update the sector availability table.

1.20.1 NAMES USED IN DOCUMENTATION

	STORAGE LOCATION IN CONSTANT TABLE
CSQSEC	128
SECTOR	138
PARBAS	131
INPWRD	10
SECTEM	143
INPCTR	15
TEMP3	144
SW6	28
ENTPNT	13
TEMP1	46
HSHADR	43
OVSTR	124
OVCTR	125
LGEPGE	110
MSDWCT	134
STRSEC	133
HSHTBL	108

1.20.2 Constant storage is used as follows:

1. CSQCTR — last address of program command sequence storage +1.
2. SECTOR — number of words in SECTOR.
3. PARBAS — address of start of partition.
4. INPWRD — contains end of command sequence storage.
5. SECTEM — sector number of CREP1 table storage on mass memory.
6. INPCTR — used to address location of command sequence storage.

7. TEMP3 temporary storage.

8. SW6 is used as a counter.

9. ENTPNT contains address associated with name in ENT or EXT table.

10. TEMP1 used as temporary storage.

11. HSHADR relative address of ENT or EXT on mass memory.

12. OVSTR word address of overflow table start.

13. OVCTR points to next available location in the overflow table.

14. LGEPGE largest command sequence page that has been stored into.

15. MSDWCT number of words stored on mass memory.

16. STRSEC starting sector of image on mass memory.

17. HSHTBL number of hash codes in hash table.

1.20.3 Entry to BLDCR1

BLDCR1 is declared external to load and is entered by a 2 word jump instruction.

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	BLDCR1	

1.20.4 External Subprograms Used by BLDCR1

1.20.4.1 EXSTOR is used to store address of next available slot in the external table into HSHADR. The counter for the External table is incremented to the next position. EXSTOR is declared external in BIDCR1 and is entered by a return jump. Upon entry to EXSTOR the A register contains the address associated with the name in the external table.

1.20.4.2 PATCH — is used to patch PART 0 externals. PATCH is declared as an external in BLDCR1 and is entered by a return jump. Prior to entry to PATCH all of the PART 0 external references to core resident PART 1 programs are put in the External table.

1.20.4.3 WRT0UT — is used to write out ENTRY-EXTERNAL pages on mass memory. A page must be modified before it can be written on mass memory.

1.20.4.4 DISKWR — is used to write into crep table block.

1.20.5 Subroutines Used Internal to BLDCR1

1.20.5.1 SQUEEZ — is used to squeeze entry points in the hash table together. Entry points in the overflow table are consecutive and do not need to be squeezed. SQZBIT is called by SQUEEZ to squeeze the bits in the bit table together. SQUEEZ and SQZBIT are entered by return jumps. $FFFF is written at the end of the entry table to indicate the end of the table. SQUEEZ determines which page to write the CREP1 table into. The first word of the CREP1 table contains the number of sectors of the crep table block.

1.20.5.2 M0VTBL — is used to read a word by calling DISKRD and write a word by calling DISKWR. The entry points are stored consecutively from the hash table.

1.20.6 Exit Procedure:

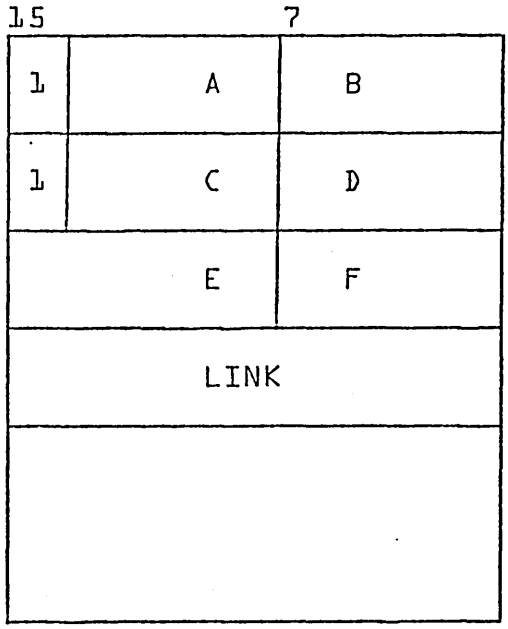
$EE is stored in second word of a two word jump at BLDXIT and a return jump is made through $EE. $EE is a location in the communications region used for return to the job processor by the loader.

1.21.0 PATCH

PATCH is used to patch externals referenced in programs being loaded. Bit 15 of word zero indicates a relative or absolute external. Bit 15 of word one is used to determine if the external is patched.

EXAMPLE

EXT ABCDEF



1.21.1 NAMES USED IN DOCUMENTATION

	STORAGE POSITION IN CONSTANT TABLE
EXTCTR	7
EXTSTR	118
SECTOR	138
TEMP3	144
EXTADR	143
INPXCO	57
INPCTR	15
ABRLSW	9
TABSCH	40
SW6	28
NOTLNK	16
LINK	14
ENTPNT	13
LINK1	104

1.21.2 Communications Region Used

ZERO {$22} = 0

MASK2 {$21} = $8000

1.21.3 Usage of Constant Table is as Follows:

1. EXTCTR points to the next location in external table.
2. EXTSTR word address of start of external table.
3. SECTOR number of words in a sector.
4. TEMP3 temporary location used as a counter.
5. EXTADR sector number of CREP1 table storage on mass memory.
6. INPXCO address of input buffer.
7. INPCTR used to address location of command sequence storage.
8. ABRLSW is used as an absolute or relative External switch.
9. TABSCH address of TABSCH routine.
10. SW6 indicates external is patched.
11. NOTLNK will be a one if unpatched externals are found.
12. LINK contains address associated with name in loader table.
13. ENTPNT contains address associated with name in ENT or EXT table.
14. LINK1 address of LINK1 routine.

1.21.4 Subprograms used by PATCH are:

1.21.4.1 DISKRD used to get a word from mass memory.

1.21.4.2 DISKWR used to write a PATCH word on mass memory.

1.21.4.3 TABSCH used to search entry points for a match.

1.21.4.4 LINKL If a match is found in TABSCH an RTJ- LINKL is made to patch externals. Upon return from LINKL the °PATCHED° flag in word 2 of the external table entry is set and the patched word is written out by DISKWR. If no match is found NOTLNK is set to a one.

1.21.5 PATCH is entered by a

RTJ PATCH

to exit from PATCH

JMP⋈ {PATCH}

If there are no externals in the External table upon entry to PATCH a return is made to the calling program.

1.22.0 TBSCH1

The TBSCH1 subroutine is used to search the Loader Table for an entry containing a name to match a name supplied by the calling program. The entry point name for TBSCH1 is TABSCH.

1.22.1 Constant Table Storage Referenced by TBSCH1

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
SW6	28
AHOLD	136
INPCTR	15
SCHXIT	39
HSHADR	43
TEMP	45
ENTPNT	13

1.22.2 Constant Storage is used as follows:

1. AHOLD is used as temporary storage for the A register during execution of the TABSCH subroutine.
2. INPCTR contains the first address of storage of the name being searched for in the Loader Table.
3. SW6 is used as a counter for program execution within a loop and also is set to indicate if a match is found while searching the loader table.
4. SCHXIT is used to exit from program.
5. TEMP is a temporary location.
6. ENTPNT contains address associated with name in entry or external table.
7. HSHADR is the relative address of a hash table entry.

1.22.3 Communication Region Constants Used by TBSCH1

CONSTANT	LOCATION
$0000	$2 = LPMSK

1.22.4 Entry to TBSCHL

Prior to entering TBSCHL, 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 TBSCHL with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	TABSCH,I	

1.22.5 Searching Operation

Upon entry to the TBSCHL subroutine -

1. The location INPCTR contains the first word address of some entry in the input block.

1.22.5.1 TBSCHL does a return jump to HASH to hash the entry point name. The result of HASH gives the appropiate address of an entry point name. On return from HASH a return jump is made to BITCHK. BITCHK checks the bit table to see if an entry has been made in that hash table position.

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 from TBSCHL.

The location HAHADR contains the base address of the loader table and SW6 is used as an index counter in searching the loader table. Initially, upon entry to TBSCHL, the location SW6 is cleared to zero. TBSCHL will compare bits 0-14 of the first 3 words of each entry of the LOADER TABLE with bits 0-14 of the first 3 words of the entry from the INPUT BLOCK (as the result of indexing through the table) as follows:

{{INPCTR}} is compared with {{HSHADR}+SW6}}

{{INPCTR}+1} is compared with {{HSHADR}+{SW6}+1}

{{INPCTR}+2} is compared with {{HSHADR}+{SW6}+2}

If a comparison is made between the name in a LOADER Table entry and the given name, a jump is made to the exit from TBSCHL. Upon exit from TBSCHL the address obtained by the addition

{HSHADR}+{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:

1.22.6 Exit from TBSCH1

Exit from the TBSCH1 subroutine is made with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	SCHXIT,I	

1.22.7 Subprograms external to TABSCH:

1.22.7.1 HASH used to hash entry or external name.

1.22.7.2 BITCHK used to check bit table to see if there is an entry for that position in the Hash Table.

1.22.7.3 DISKRD is used to get a word from mass memory.

These subprograms are entered by executing the following instruction:

RTJ NAME

To exit from these routines the following instruction is executed:

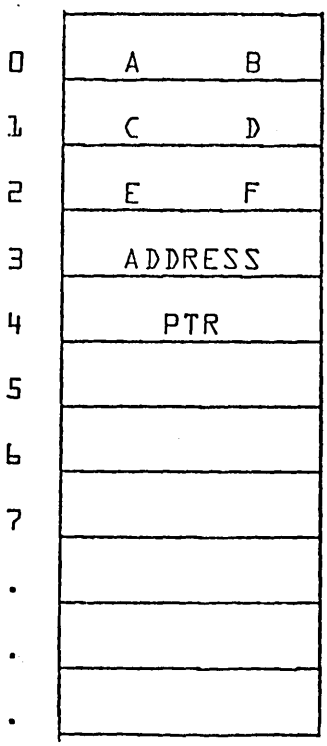
JMP {NAME}

1.23.0 HASH

This routine hashes the ASCII characters of an entry point or external name and computes the address in the hash table. To get the hash code for an entry the ASCII characters are added together and the sum is divided by the number of possible entries in the hash table. The remainder is the Hash code.

For each entry in the hash table five words are used. The hash code is multiplied by 5 to find the address of the entry in the hash table. The hash table is 15 pages in length and the overflow table is 10 pages in length. There are 576 possible entries in the hash table.

EXAMPLE OF ENTRY IN HASH TABLE.

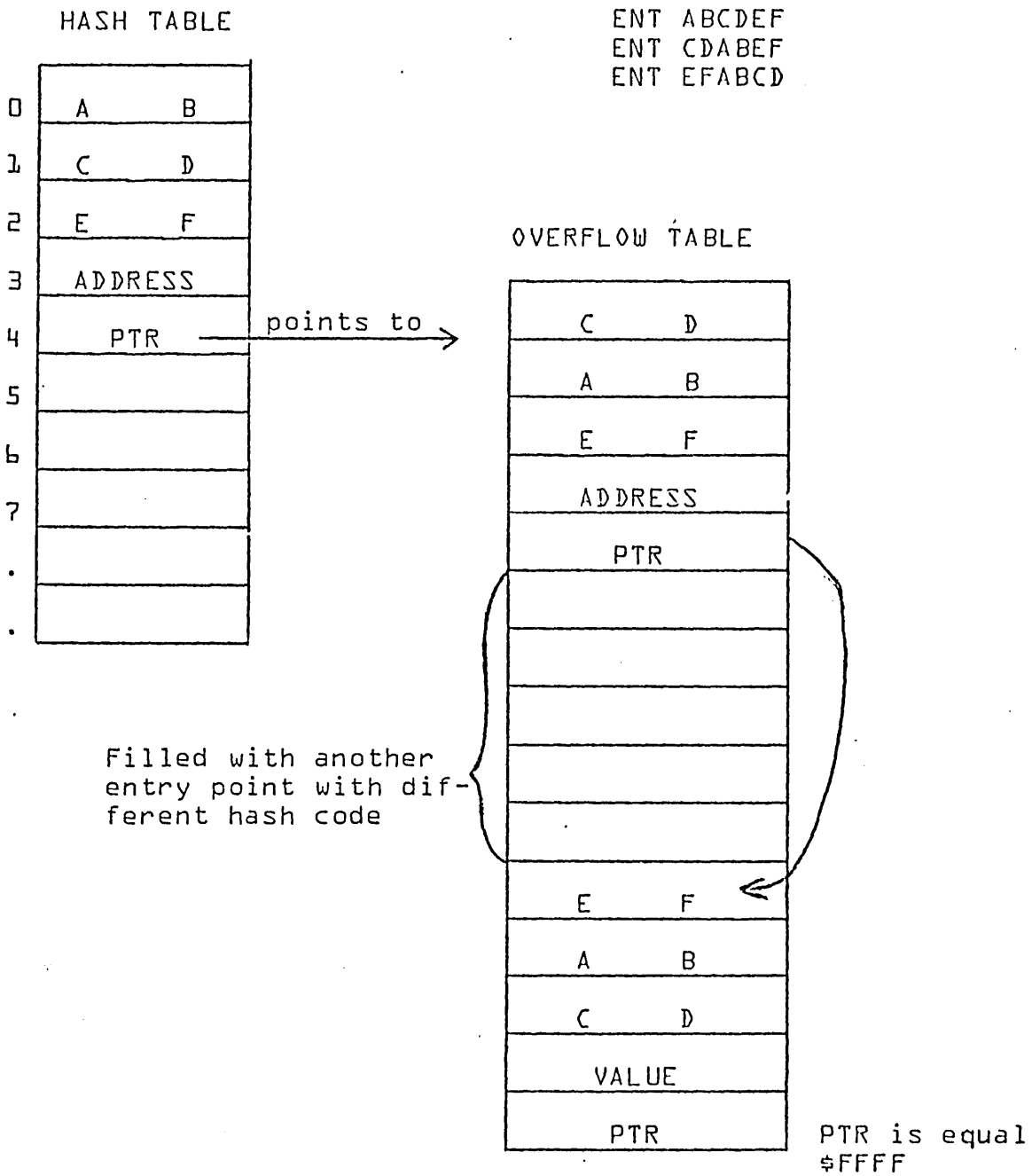


ENT ABCDEF

PTR is equal to $FFFF if it is the last entry in the series of entry points with the same hash code.

The overflow table is used in the same manner as the hash table when the hash entry for that hash code is full. When it is necessary to make an entry in the overflow table, the pointer (PTR) is set to point to that entry. This links the entry points with the same hash code together.

EXAMPLE OF USE OF OVERFLOW TABEL.



1.23.1 ENTRY POINT NAME: HASH

This program is used by TBSCHL and is declared external. HASH is entered by a RTJ HASH.

1.23.2 Constant table storage referenced by HASH

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
HASHCD	44
SW6	28
INPCTR	15
TEMP	45
HSHADR	43
HSHTBL	108

1.23.3 Communications region used:

LPMSK{2} = 0

FIVE{$43} = 5

ZERO{$22} = 0

1.23.4 Constant storage is used as follows:

1. HASHCD — contains hash code of ENT or EXT being hashed.
2. SW6 — is used as a counter.
3. INPCTR — is used to address location of command sequence storage.
4. TEMP — first of 5 temporary storage locations used by hash routine.
5. HSHADR — relative address of ENT or EXT on mass memory.
6. HSHTBL — is the number of hash codes in hash table.

1.23.5 Exit from HASH

LABEL	OPCODE	ADDRESS	REMARKS
	JMP	{HASH}	

1.24.0 TBSTR1

The TBSTR1 routine is used to store an entry point name and its address into the entry point table or an external name and its link address into the external table.

1.24.1 Constant Table Storage Referenced by TBSTR1.

NAMES USED IN DOCUMENTATION	STORAGE POSITION IN CONSTANT TABLE
EXTSTR	118
OVCTR	125
HSHADR	43
TEMP	45
EXTCTR	7
HASHCD	44
PRINT2	59
INPCTR	15
ENTPNT	13
SW6	28

1.24.2 Constant table storage is used as follows:

1. ENTPNT — contains the absolute core address associated with the name to be stored in the Loader Table.
2. SW6 — is used for loop control as a counter.
3. EXTSTR — word address of start of external table.
4. OVCTR — points to next available location in overflow table.
5. HSHADR — starting address available slot in entry point table.
6. TEMP — temporary location.
7. EXTCTR — points to next available location in external table.
8. HASHCD — hash code of entry or external.
9. PRINT2 — used to print error message.

1.24.3 Entry to TBSTR1

Prior to entering TBSTR1, 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 TBSTR1 with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ-	TABSTR,I	

1.24.4 Table Store Operation

Upon entry to TBSTR1 -

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 ENTPNT contains the address for this name.

The TBSTR1 subroutine will first make sure that by adding another entry to the Loader Table, there will be no overflow of the loader table. Overflow of the loader table will result if - that hash table entry and the overflow table are full. If this condition should arise, a jump will be made to the PRINT2 error routine where an error indication of E2 is printed. Loading is then terminated.

The address for the name is placed in the fourth word of the entry as follows:

{ENTPNT} ⟶ {VALUE} if {SW6} = 0

a return jump is then made to the routine STORE.

The ▽pointer▽ is placed in the last word of each new entry as follows:

{$FFFF} ⟶ {VALUE}

and a return jump is then made to the routine STORE.

1.24.5 Internal Subroutines Used by TBSTR1.

1.24.5.1 STORE — is used to write a word. This routine is entered by an

```
RTJ*    STORE
```

To exit from STORE:

```
JMP*    {STORE}
```

STORE is referenced by STRENT.

1.24.5.2 STRENT — This routine gets the hash address of the ENT name and stores it in the appropriate table. A $FFFF is stored in the pointer word for the entry point. This routine is entered by

```
RTJ*    STRENT
```

To exit this routine:

```
JMP*    {STRENT}
```

1.24.5.3 EXSTOR — EXSTOR stores the external name and link address in the next available slot in the external table and increments the external table counter. EXSTOR is also an entry point. EXSTOR is declared as an external in the BLDCR1 module. EXSTOR is entered by

```
RTJ     EXSTOR
```

to exit:

```
JMP     {EXSTOR}
```

1.24.5.4 BITCHK — BITCHK is an entry point in TBSTR1 and is entered by an

```
RTJ     BITCHK
```

to exit:

```
JMP     {BITCHK}
```

The hash code is divided by 16 to find the word and bit position. If the bit is a one it indicates an entry in that position in the Hash table.

1.24.6 Exit from TBSTR1

Following a loader table entry, exit is made from TBSTR1 with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	{ENTSTR}	

If overflow of Loader Table would have occurred as a result of the Loader Table entry, exit is made from the TBSTR1 routine with the following instruction:

LABEL	OPCODE	ADDRESS	REMARKS
	JMP-	PRINT2,I	

1.25.0 PAGE

1.25.1 Calling Sequences

1. To get a word

ON ENTRY: {A} register = execution address of word referenced
{Q} Register saved

RTJ DISKRD

NUM $8000 or 0 $8000 indicates ENT-EXT PAGE reference.

ON EXIT: {A} Register = Word reference 0
{Q} Register is restored

2. To store a word

ON ENTRY: {A} register = execution address of word referenced
{Q} register saved

RTJ DISKWR

ADC VALUE value to be stored

NUM $8000 or 0

ON EXIT: {Q} Register restored

1.25.2 Entry Points:

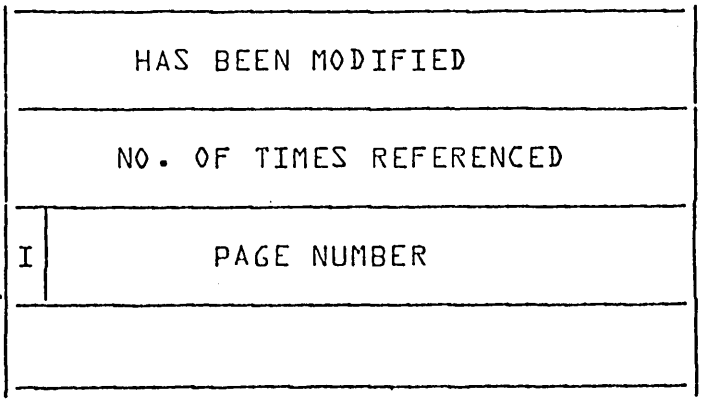
DISKRD - get a word.

DISKWR - store a word.

WRTOUT - write out all pages that have been changed.

1.25.3 PAGE in Core

These page flags are initialized in the STBASE module.



I = 0 if CSQ

I = 1 if ENT/EXT

1.25.4 MDRIV is used to write out the pages onto mass memory. Pages are only written on mass memory when used or modified. MDRIV is declared external to PAGE. When all the pages in core are used the page referenced the fewest number of times is written out to mass memory. After externals are patched by the LNKENT, LNKCR1, and LNKCRP functions the pages that have been modified are written out in order on mass memory.

1.25.5 NAMES USED IN DOCUMENTATION — STORAGE POSITION IN CONSTANT TABLE

PAGLGN	109
PAGE	122
MASINP	107
LGEPGE	110
PARBAS	131
NOPAGE	130
ADDR	117
CORADR	119
MAXPGE	129
CSQSEC	128
ENTSEC	127
PRINT2	59

1.25.6 Communications Region Used

H10{$27} = $10

NZERO{$12} = $FFFF

LPMSK{2} = 0

ZERO{$22} = 0

ONEBIT{$23} = 1

1.25.7 Constant Storage is Used as Follows:

1. PAGLGN — is the length of a page with core flags.
2. PAGE — length of page on mass memory, must be a multiple of 96.
3. MASINP — contains logical unit for MDRIV.

4. LGEPGE — largest CSQ page that has been stored into.

5. PARBAS — address of start partition.

6. NOPAGE — number of pages in unprotected core.

7. ADDR — is used as an address constant.

8. CORADR — contains {$F7}+1.

9. MAXPGE — maximum page number that can be used on mass memory.

10. CSQSEC — starting sector of CSQ image.

11. ENTSEC — starting sector of ENT/EXT tables.

12. PRINT2 — used to print error messages.

1.25.8 Entry to PAGE

PAGE is entered by a

LABEL	OPCODE	ADDRESS	REMARKS
	RTJ	DISKRD	

or

	RTJ	DISKWR	

where DISKRD and DISKWR are externals to the calling program.

1.25.9 To exit from this routine a

LABEL	OPCODE	ADDRESS	REMARKS
	JMP*	{DISKRD}	

is executed.

1.25.10 Other constants and flags used by the PAGE program are:

RWFLAG — 0 if read
1 if write

PGFLAG — storage for ENT-EXT/CSQ FLAG.

WORD — word address within a page.

PAGENO	page number being sought.
LSTPGE	page last referenced.
MINREF	counter for page used the fewest times.
MAXREF	counter for page used the most times.
MINBAS	core address of page used fewest times.
ADPAGE	core address of page to be referenced.

1.25.11 Possible Errors in PAGE.

E2

E12

These errors are printed in PRINT2 and loading terminates.

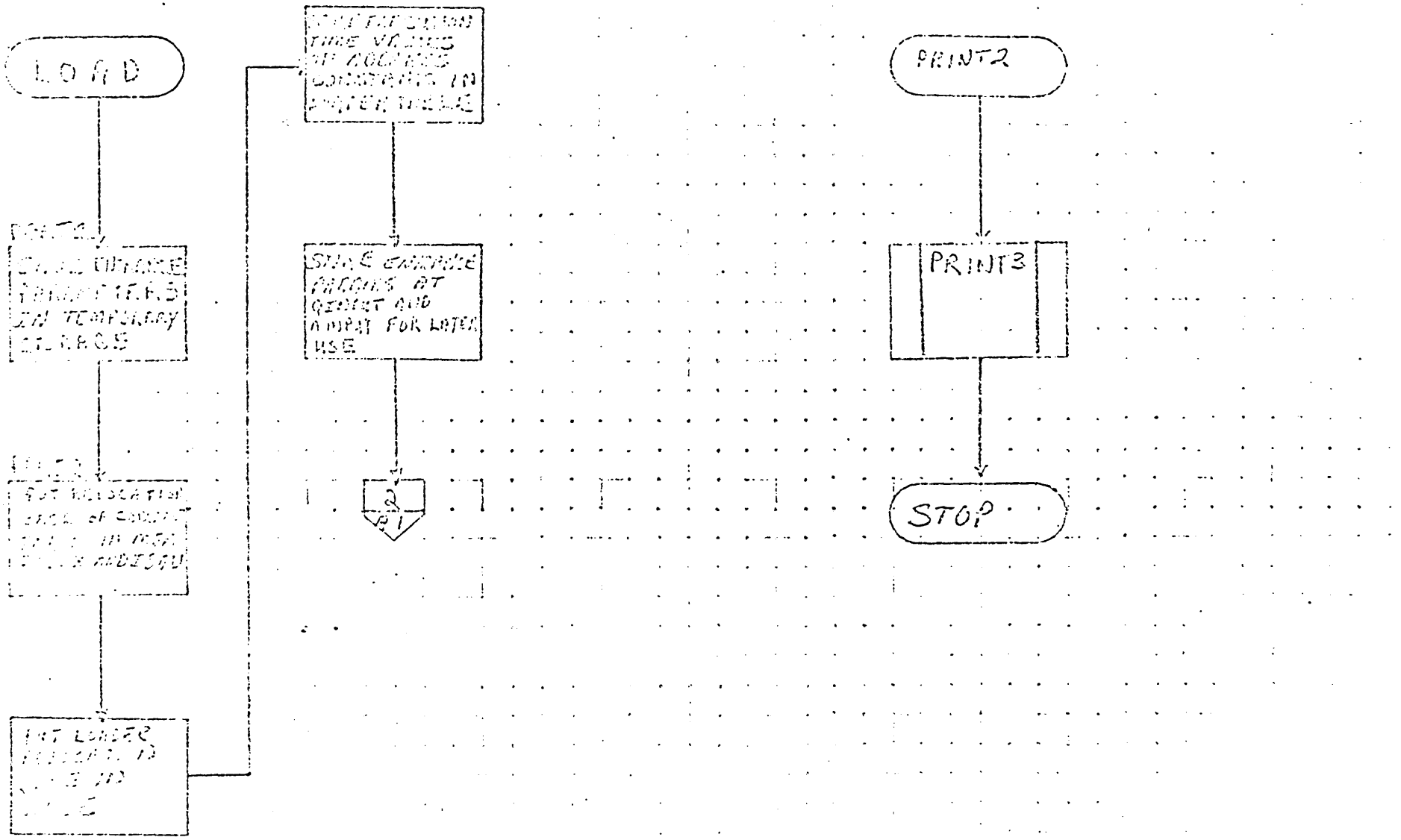
1.26.0

PART1 LOADER FLOWCHART INDEX

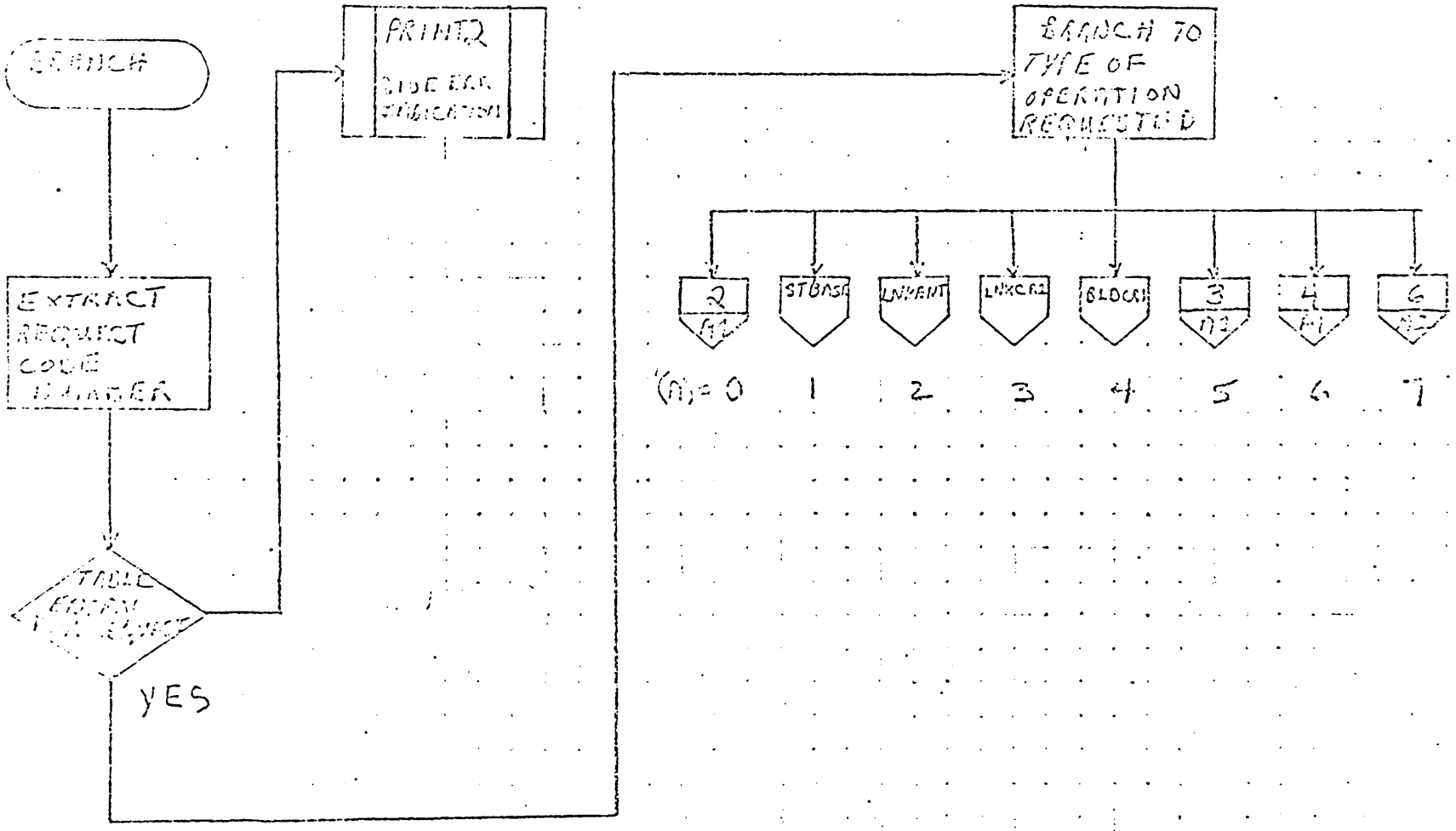
ROUTINE	PAGE NO. IN MODULE	COORDINATES
LOAD1	1	A1
PRINT2	1	A4
BRNCH1	1	A1
RBLOAD	2	A1
CMNXIT	2	B4
LNKCRP	3	A1
MAPS	4	A1
PRNEXT	6	A1
LIDRV1		
IDRIV	1	A1
LCDRV1		
CDRIV	1	A1
LMDRV1		
MDRIV	1	A1
LLDRV1		
LDRIV	1	A1
ADJOF1	1	A1
CNVERT1	1	A1
LSTOT1		
PRINT3	1	A1
STOP	1	A2
PRINT4	1	A3
PRINT5	1	A5
PRINT6	2	A1
LINK11	1	A1
LNKCK	3	A1
LOADR1	1	A1
NXTBLK	1	B2
NAMPR1	1	A1
PRNNAM	9	A1
SIGNCK	10	A4

DOCUMENT CLASS IMS PAGE NO.
PRODUCT NAME 1700 MSOS 65K Special System
PRODUCT MODEL NO. MACHINE SERIES 1700

ROUTINE	PAGE NO. IN MODULE	COORDINATES
RBDBZ1		
BZSPR0	1	A1
RBDPR0	1	A2
NXTWRD	8	A1
ADJUST	9	A1
STRLNK	11	A1
ENTEX1		
ENTPR0	1	A1
EXTPR0	2	A1
NXTNAM	2	A2
XFRPR1	1	A1
STBASE	1	A1
STORE	4	A5
LNKENT	1	A1
LNKCR1	1	A1
EXIT	2	C1
READIN	2	A3
BLDCR1	1	A1
SQUEEZ	3	A1
SQZBIT	4	A1
MOVTBL	6	A1
PATCH	1	A1
TBSCH1	1	A1
GET	2	A4
HASH	1	A1
TBSTR1		
ENTSTR	1	A1
STRENT	2	A1
EXSTOR	3	A1
STORE	3	A3
BITCHK	4	A1
PAGE		
DISKRD	1	A1
DISKWR	1	A2
FNDSEC	10	A1
WRTOUT	11	A1

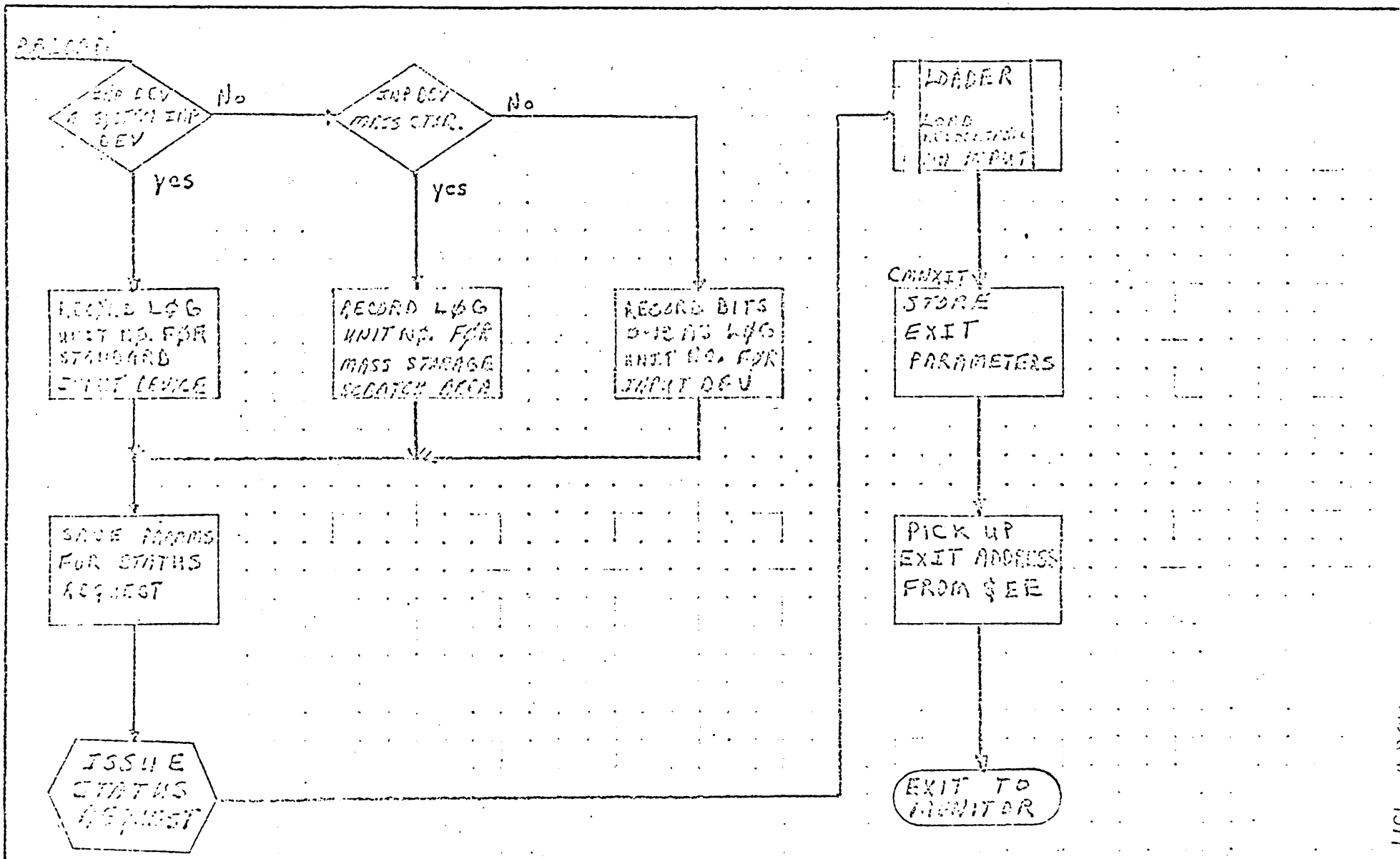


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☑ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1415 MACH. TYPE 1700	PROJECT NO.	NO.	APPROVED	DATE
	DOCUMENT TITLE LOAD 1	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

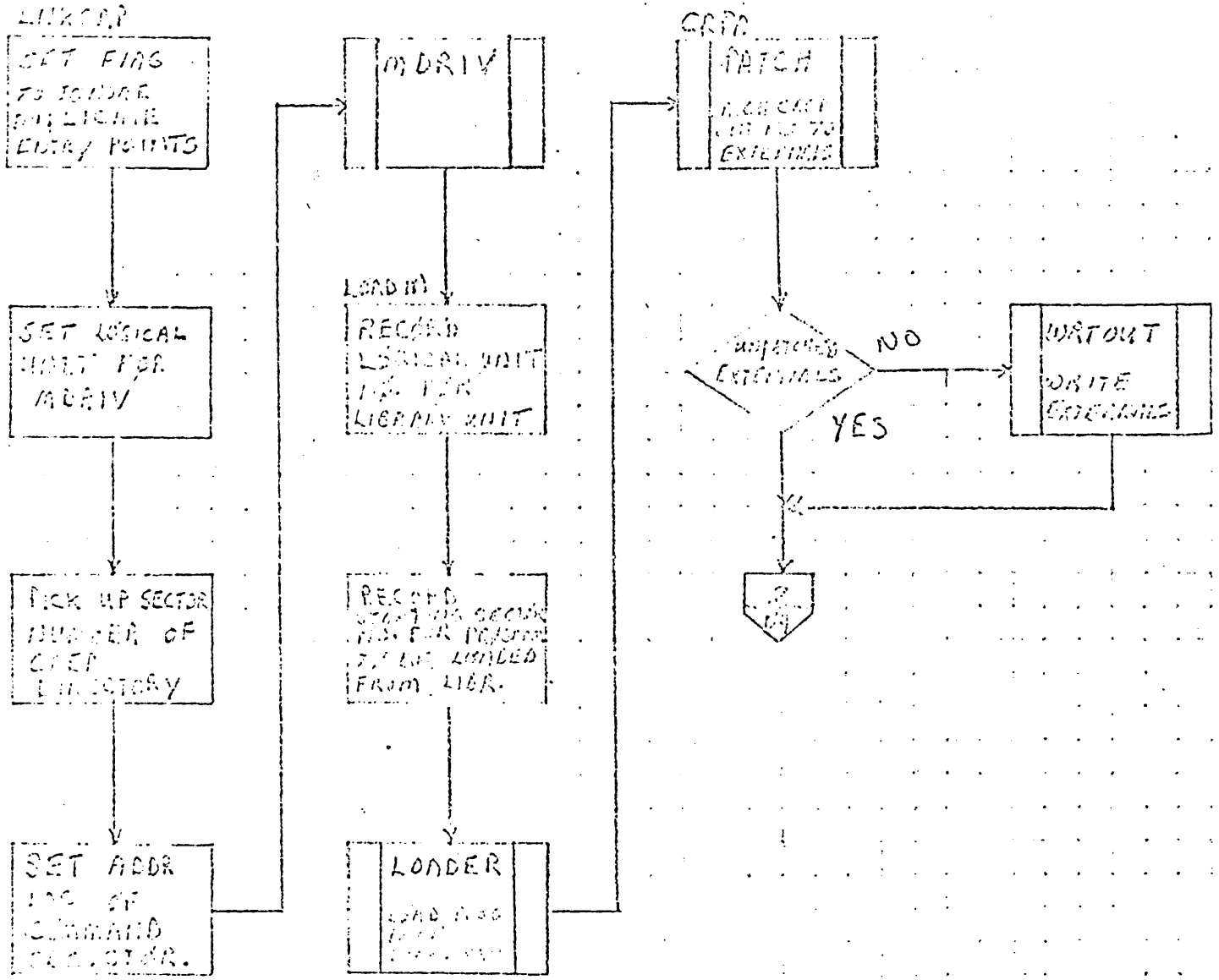


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BRNCH1	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY GAG DATE 4/20/71	TASK NAME PART 1 LOADER			

AUG 5 1971

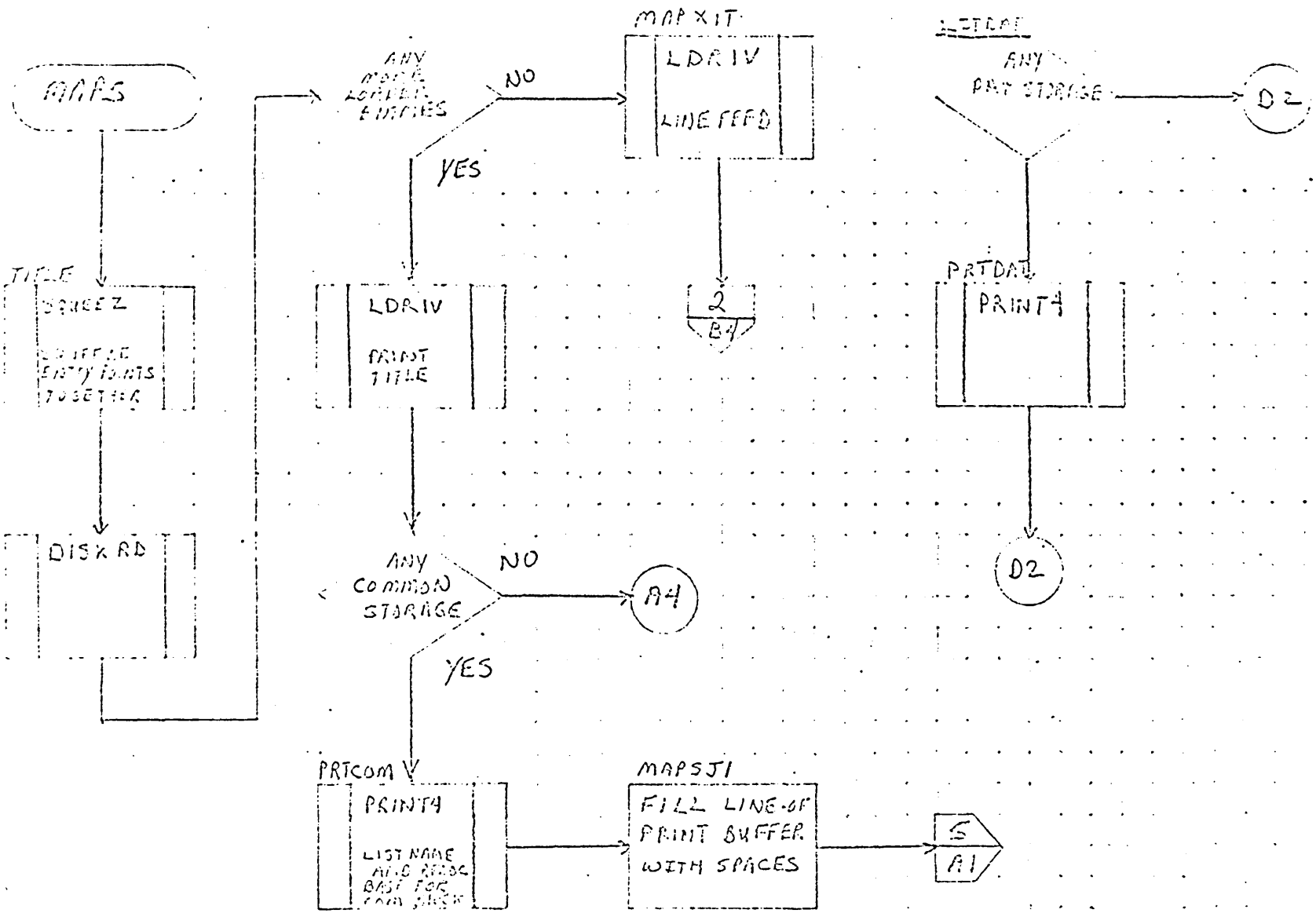


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TAS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BRNCH 2	PROJECT MGR.			
	PAGE 2 OF 7	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY CAG DATE 6/6/68	TASK NAME PART 2 LOADER			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TAS MACH. TYPE 1700	PROJECT NO.	PCN	APPROVED	DATE
	DOCUMENT TITLE BENCH1	PROJECT MGR.			
	PAGE 3 OF 7	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY GDC DATE 6/6/71	TASK NAME PART1 LOADER			

AUG 9 1971



AUG 3 1971

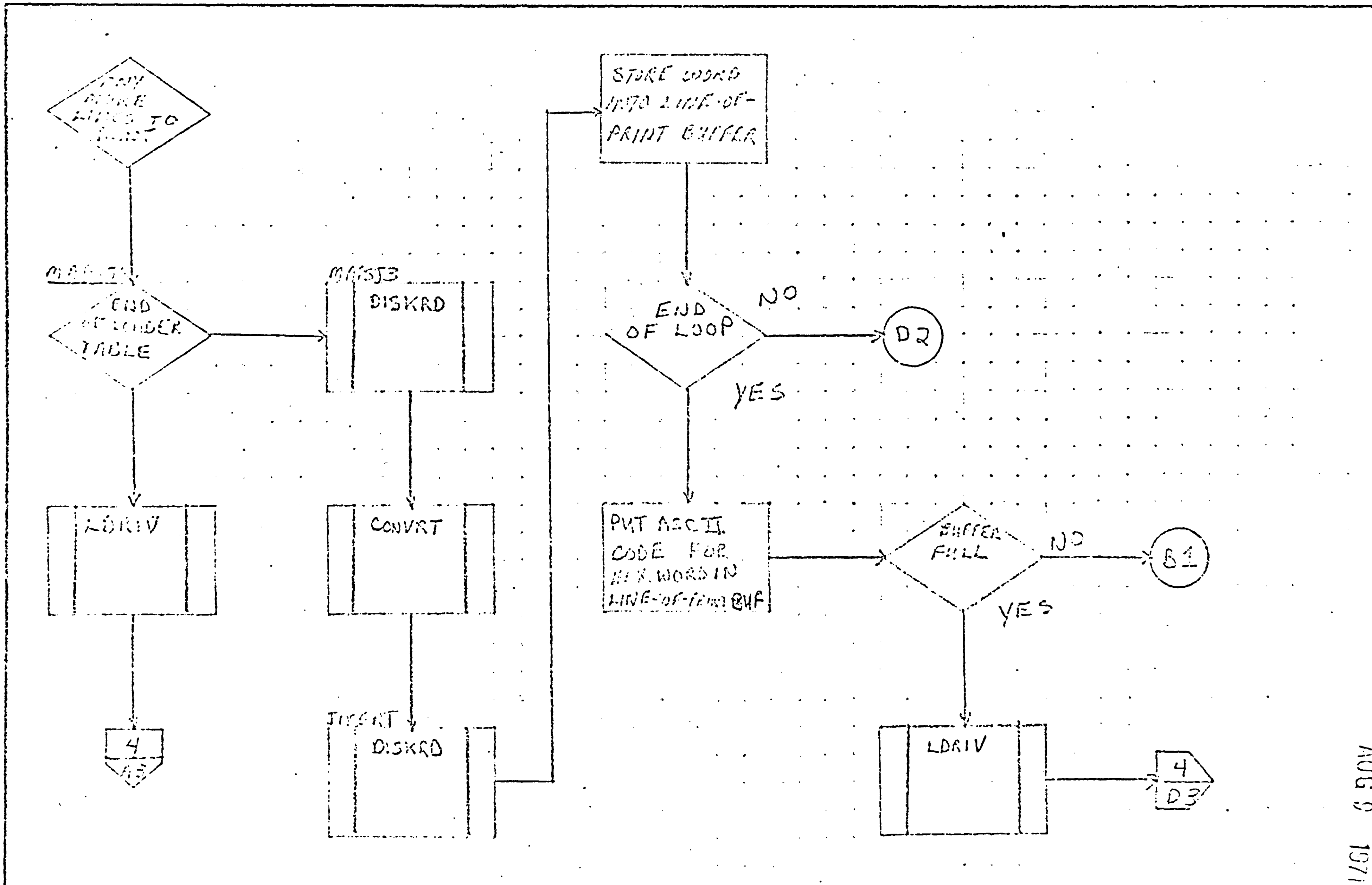
CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE MINCH1		PROJECT MGR.			
PAGE 4 OF 7		PROJECT NAME 1700 MAPS			
NUMBER	ISSUE DATE	TASK NO.			
DRAWN BY GDG	DATE 11/24/71	TASK NAME MAPXIT LDRIV			

1 2 4

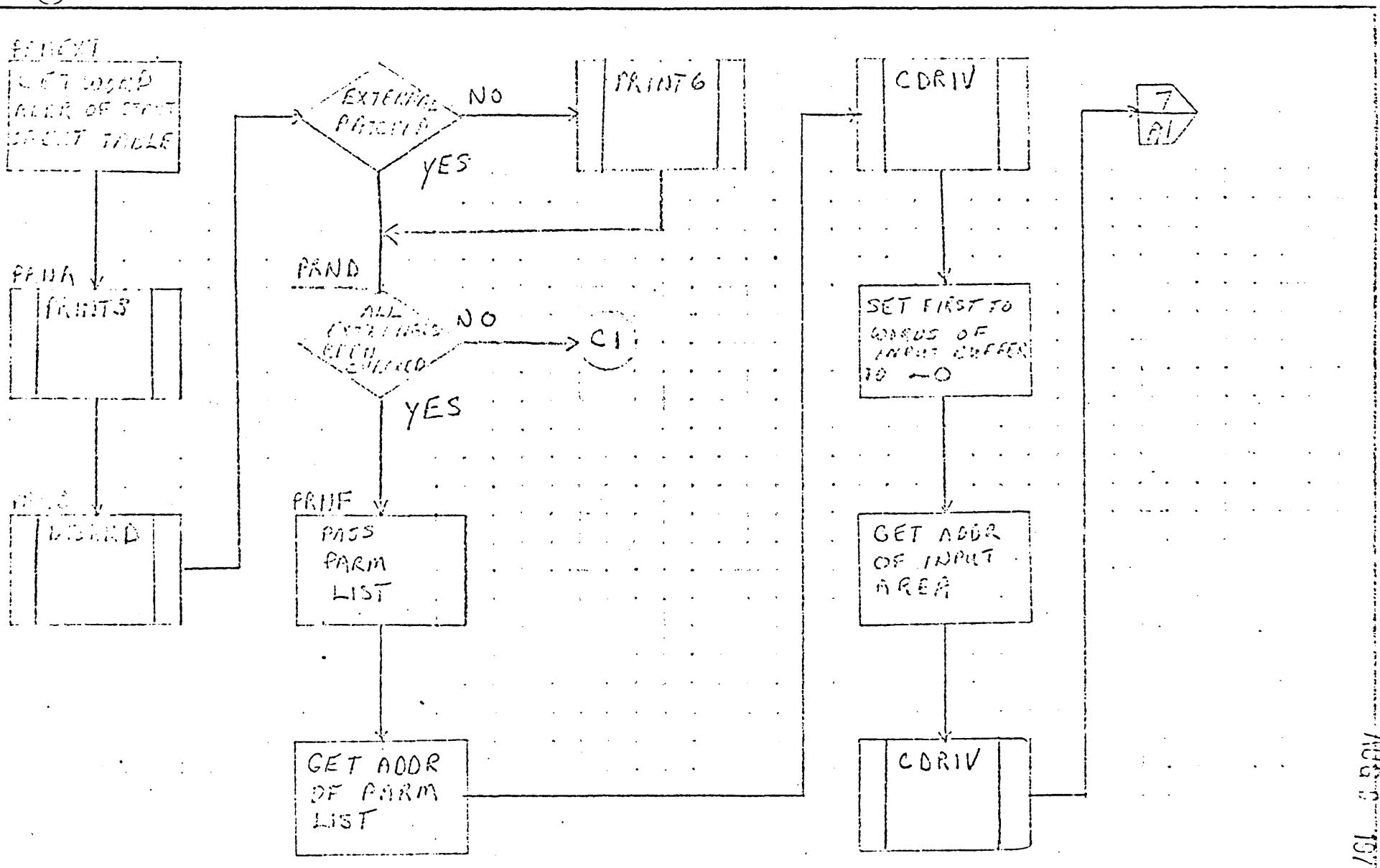
A B C D



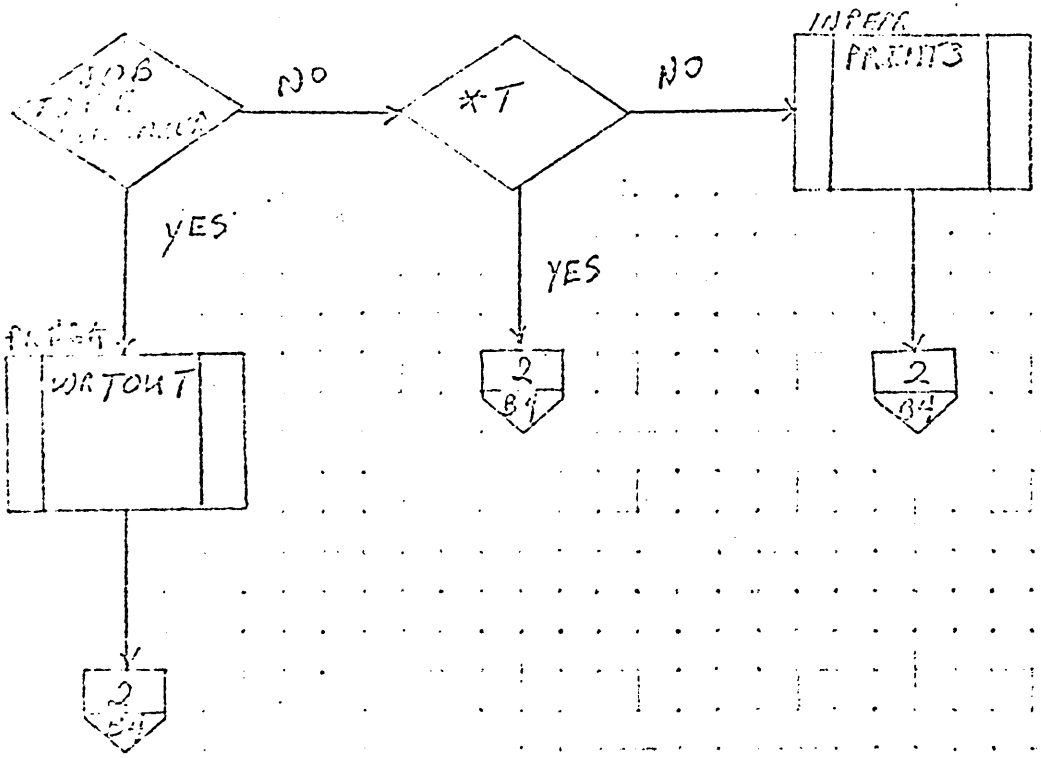
AUG 9 1971

CONTROL DATA CORPORATION SOFTWARE DOCUMENT			
SAMPLE CODE ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV / APPROVED / DATE
FLOWCHART ☒	DOCUMENT TITLE RANCH1	PROJECT MGR.	
DECISION TABLE ☐	PAGE 5 OF 7	PROJECT NAME 1700 MSOS	
OTHER ☐	NUMBER ISSUE DATE	TASK NO.	
	DRAWN BY G.A.C. DATE	TASK NAME	

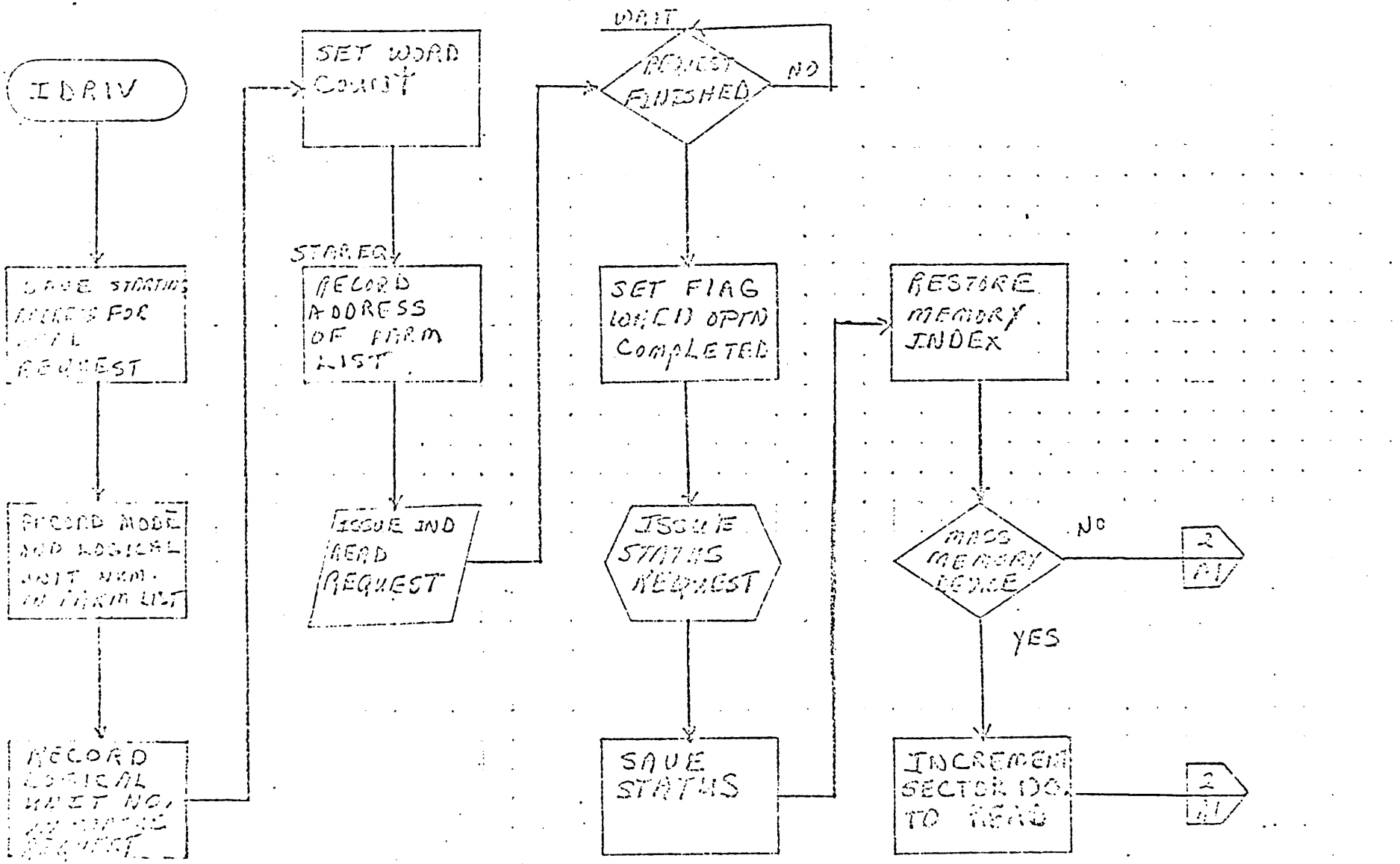
PRINTED IN U.S.A.



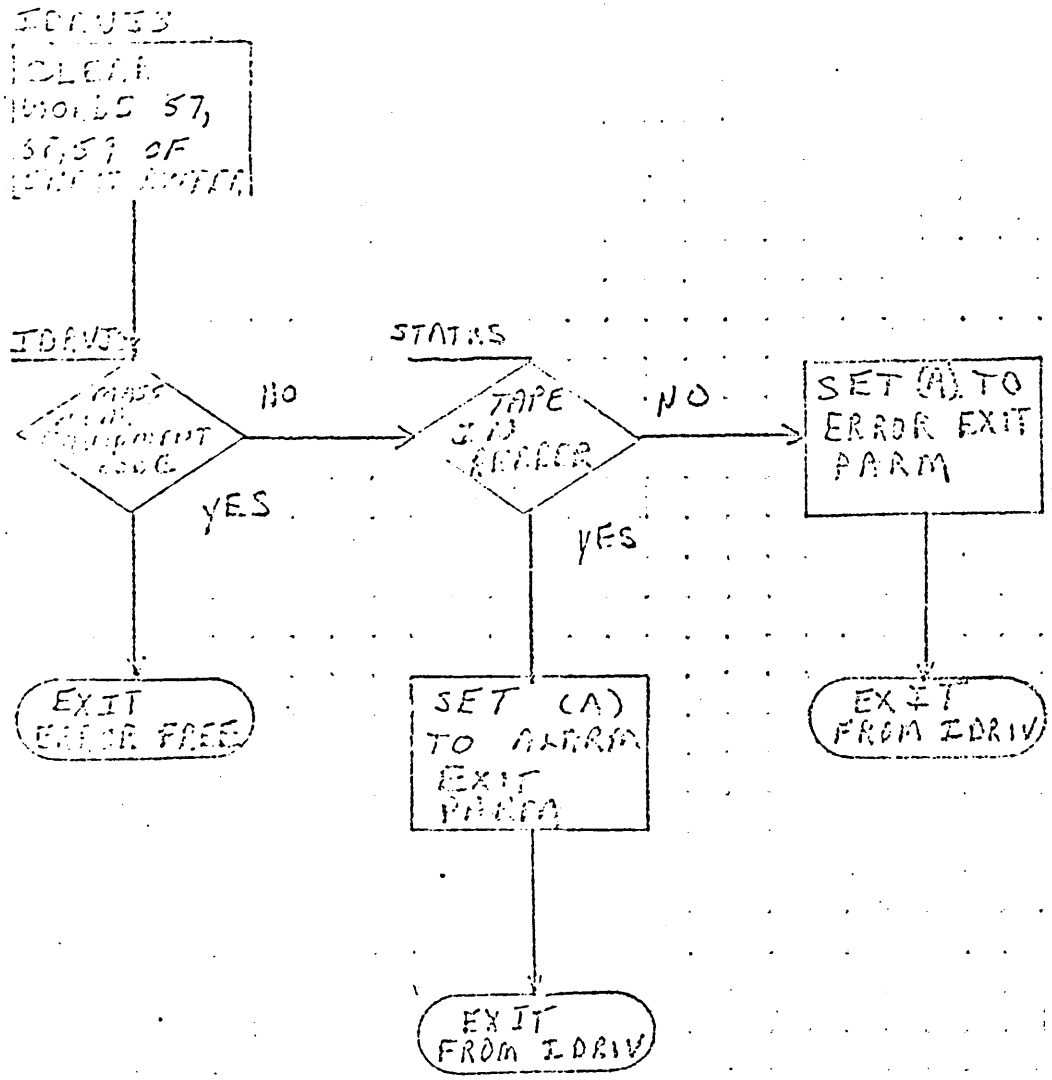
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TAS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BENCH1	PROJECT MGR.			
	PAGE 6 OF 7	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY G D C DATE 4/30/70	TASK NAME BENCH1 LOADER			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS J.A.S. MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE BRANCH1.	PROJECT MGR.			
	PAGE 7 OF 7	PROJECT NAME 1700 M.S.O.S			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY GDC DATE 1/23/68	TASK NAME PART 1 LOADER			

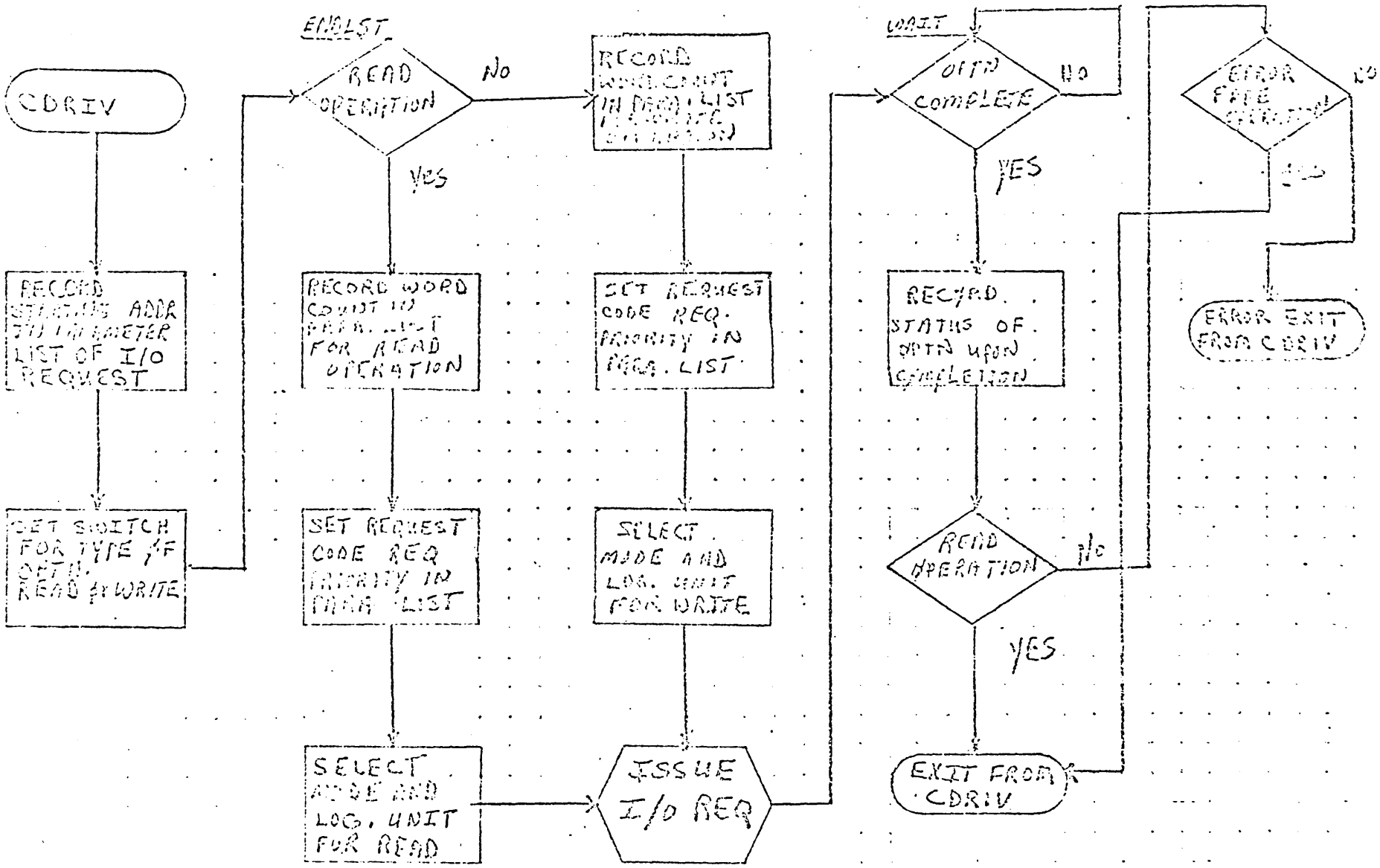


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JOBS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LIBRV1	PROJECT MGR.			
	PAGE 1 OF 3	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C.D.C. DATE (illegible)	TASK NAME PART 1 LOADER			



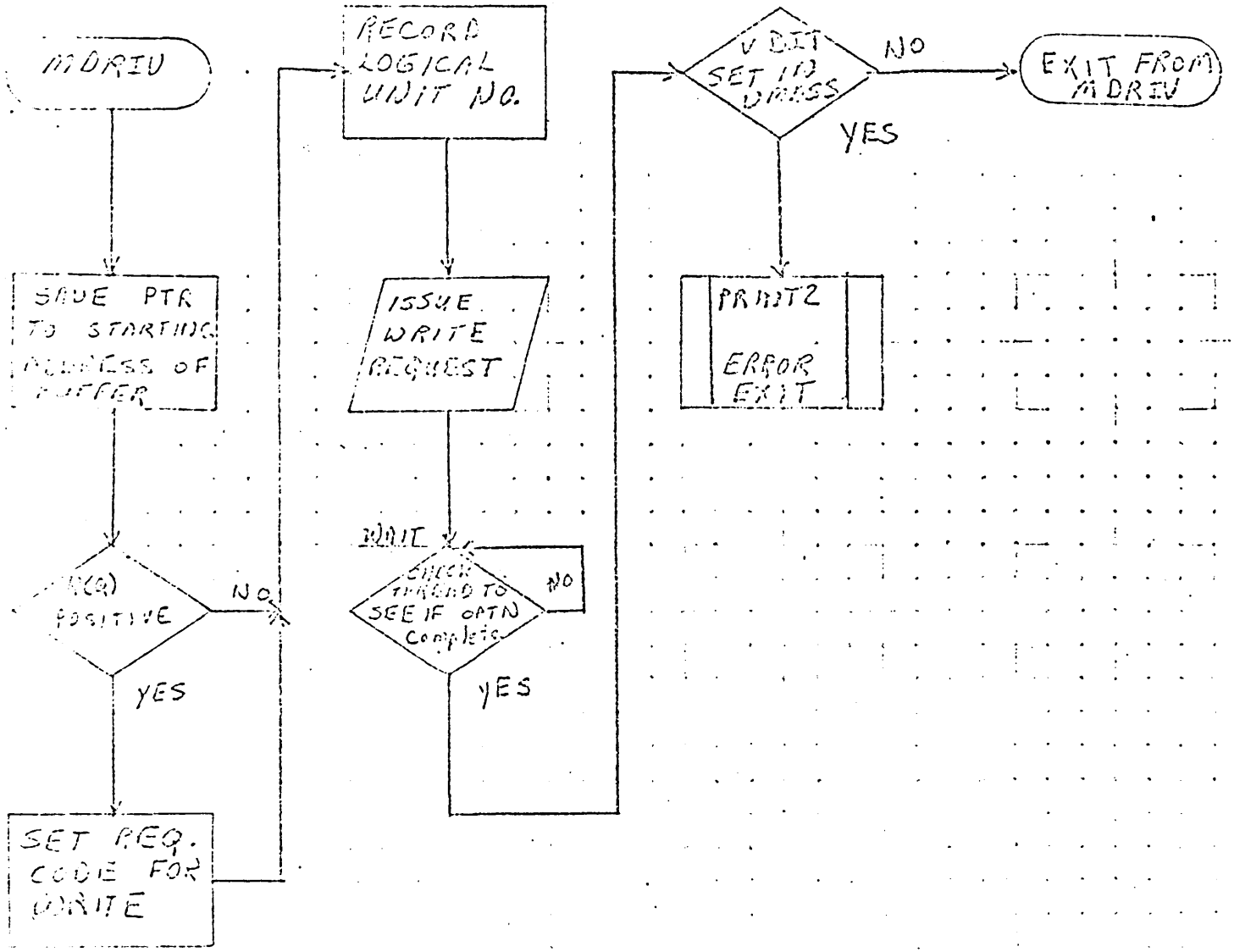
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS J.A.S	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE IDRIV1		PROJECT MGR.		
	PAGE 2 OF 2		PROJECT NAME 1700 ASCS		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY GDC	DATE 6/xx/71	TASK NAME PART 1 LOADER		

AUG 5 1971

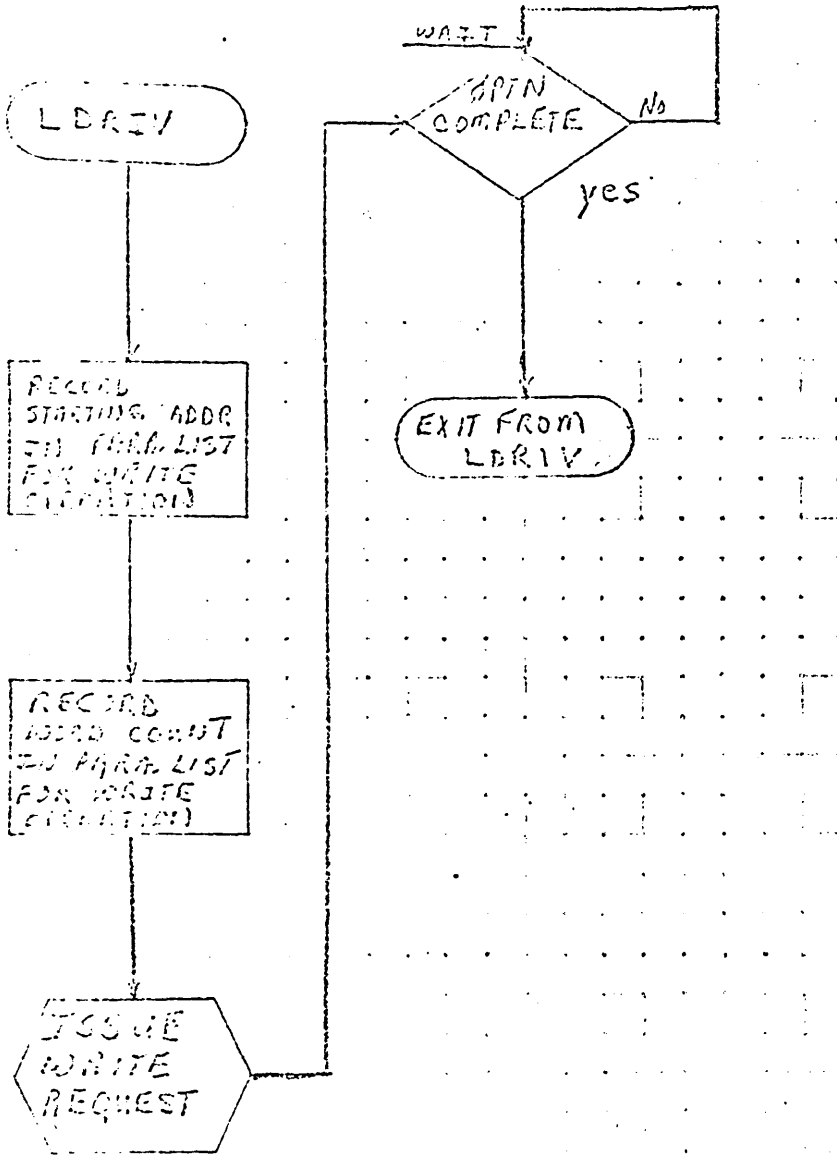


AUG 9 1971

CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LCDRV1	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY CDC DATE October	TASK NAME PART 1 LCDRV1			

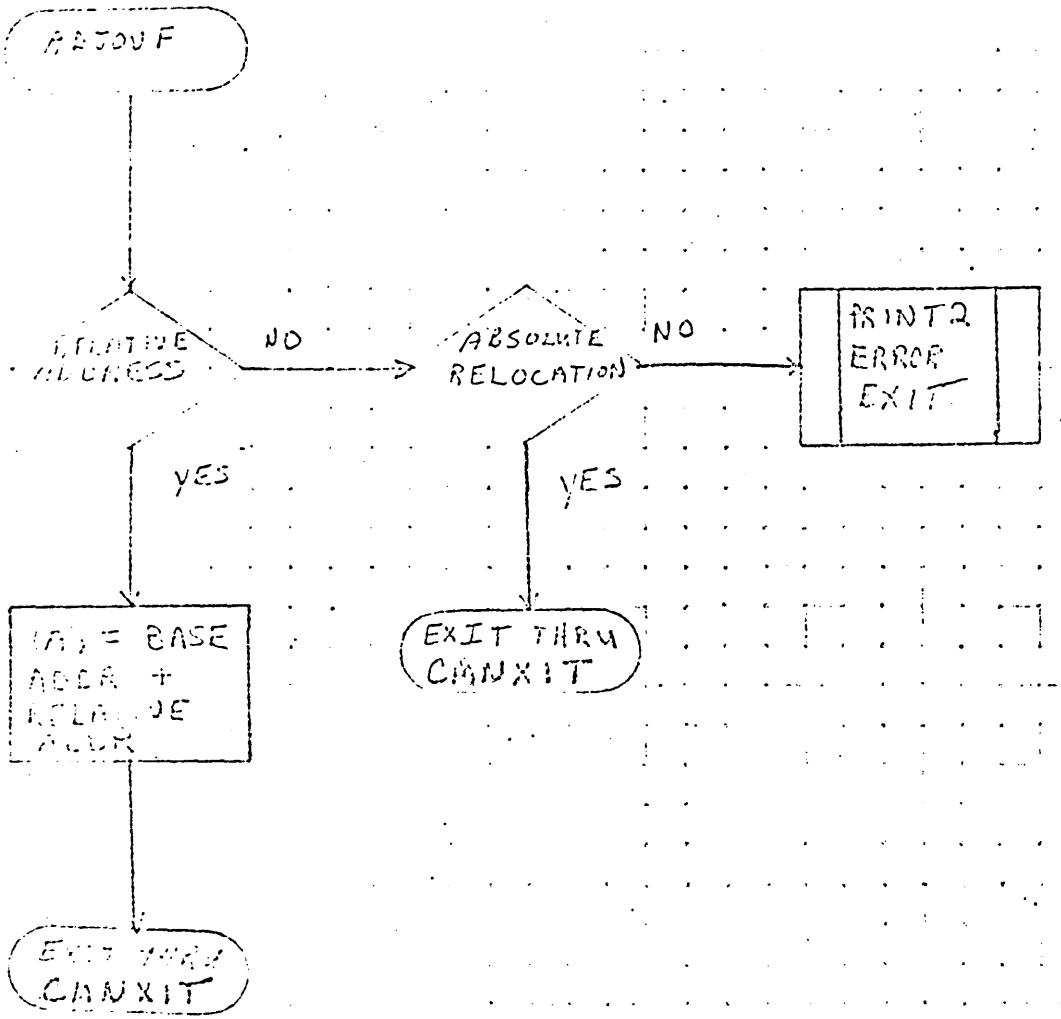


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS XMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MDRIV	PROJECT MGR.			
	PAGE OF	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY G D G DATE 4/11/71	TASK NAME PRINT, LOADER			



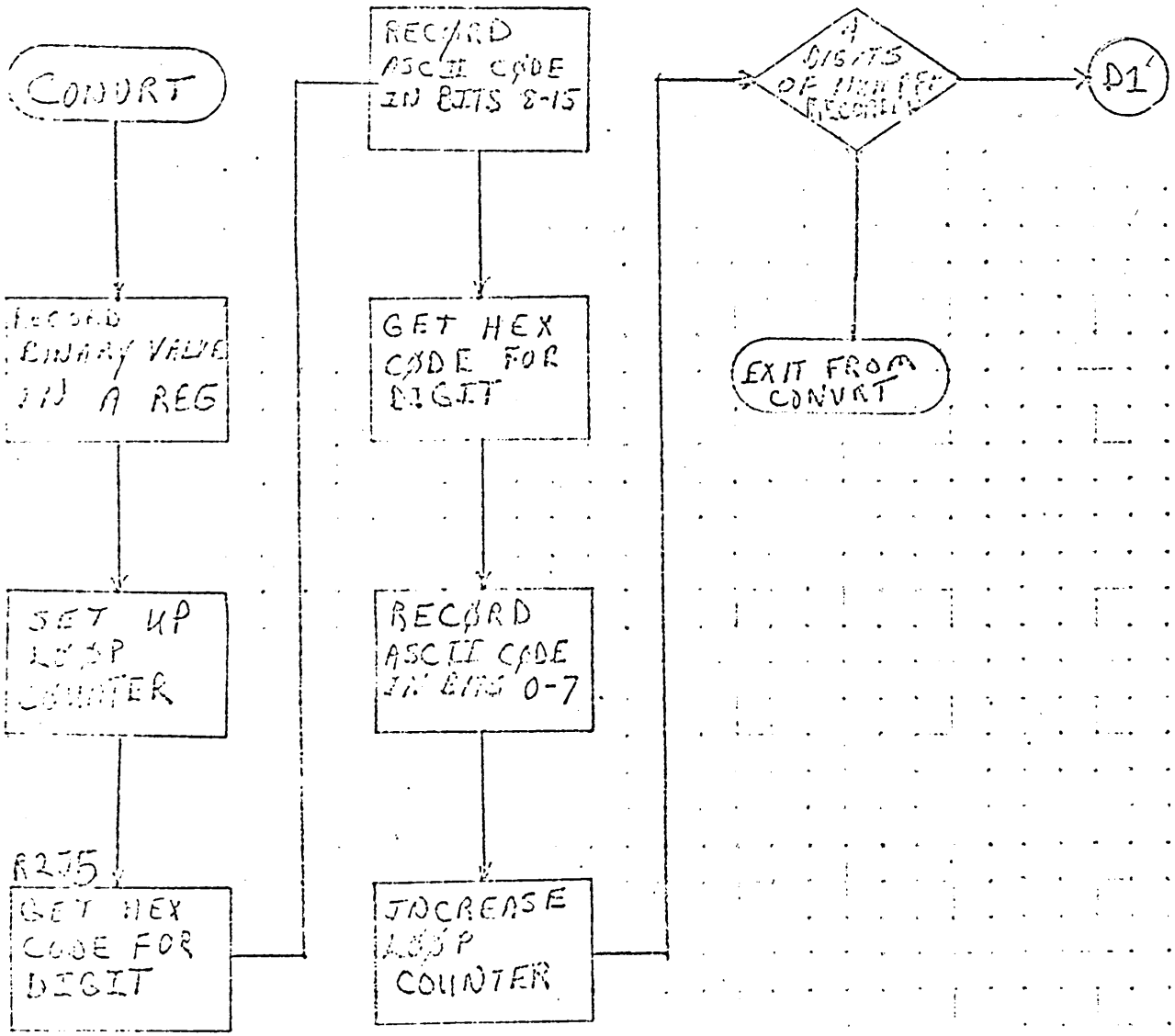
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LDRV1	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C.D.G. DATE 1/20/71	TASK NAME P.P.T. LOADER			

AUG 9 1971

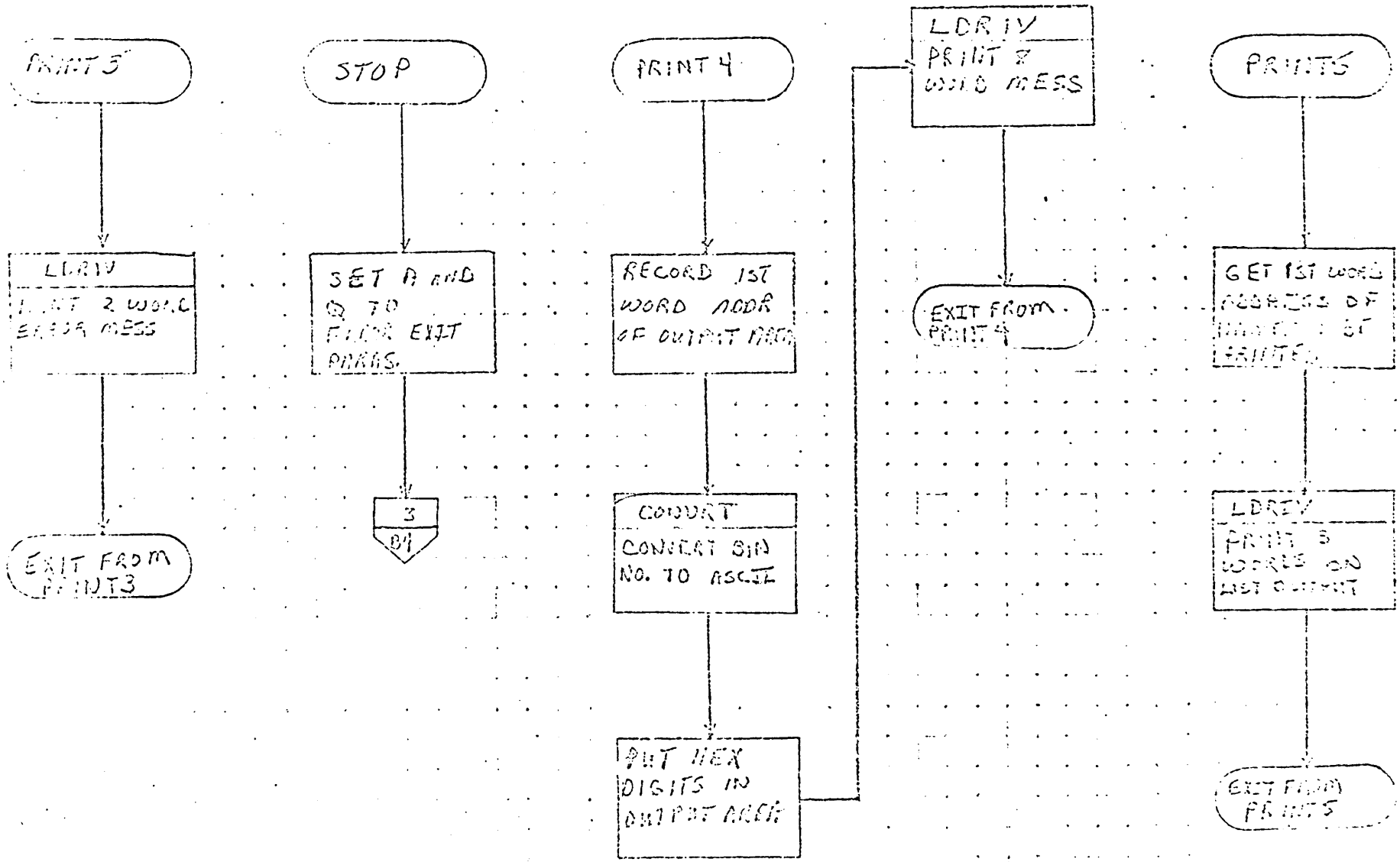


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS COMPASS MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ADJOF1	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME 1700 NOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C.D.C. DATE 6/11/70	TASK NAME PASS1 LOADER			

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 CNVRT1	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME 1700 M205			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY G.D.C. DATE 6/6/68	TASK NAME PART 1 CONVERT			

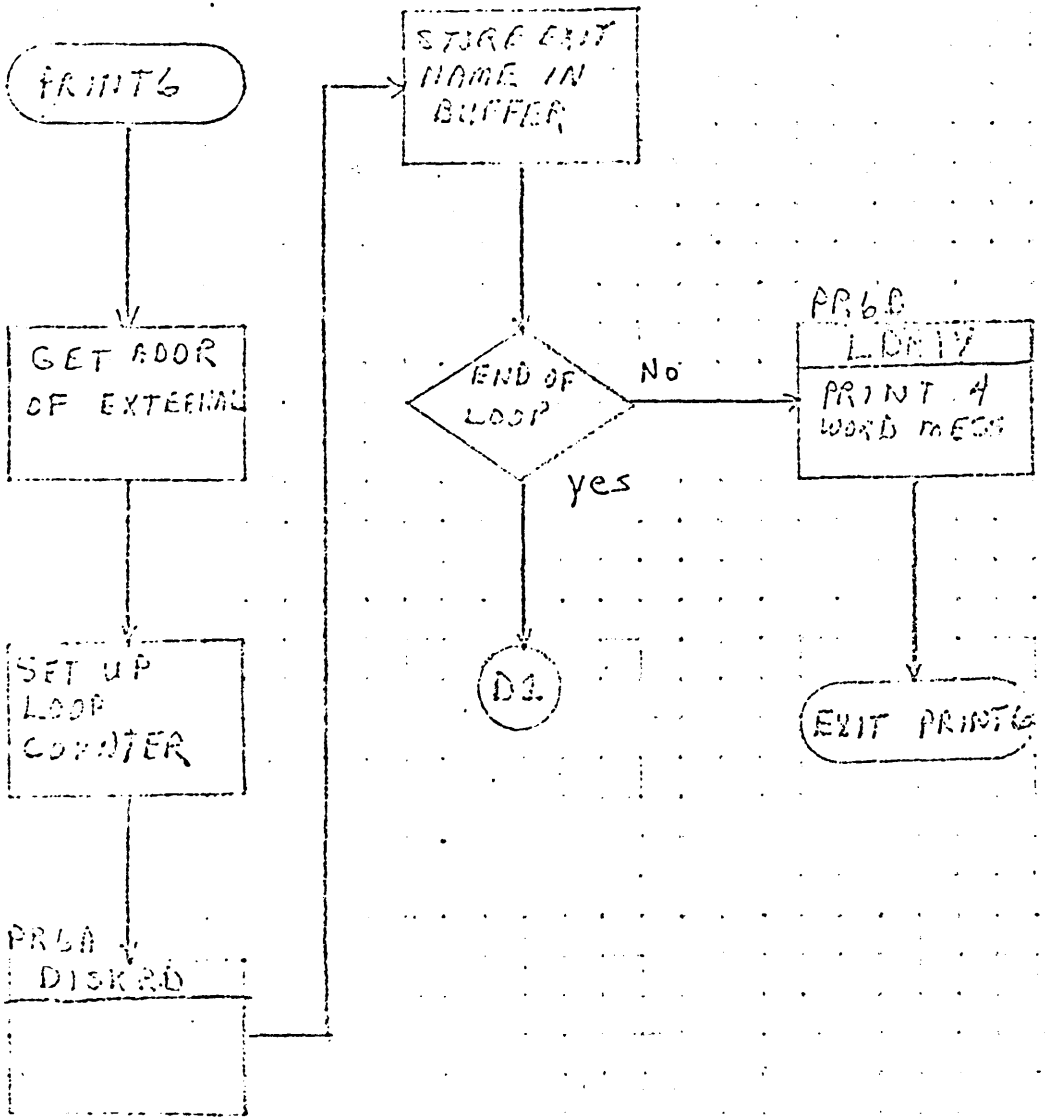


CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

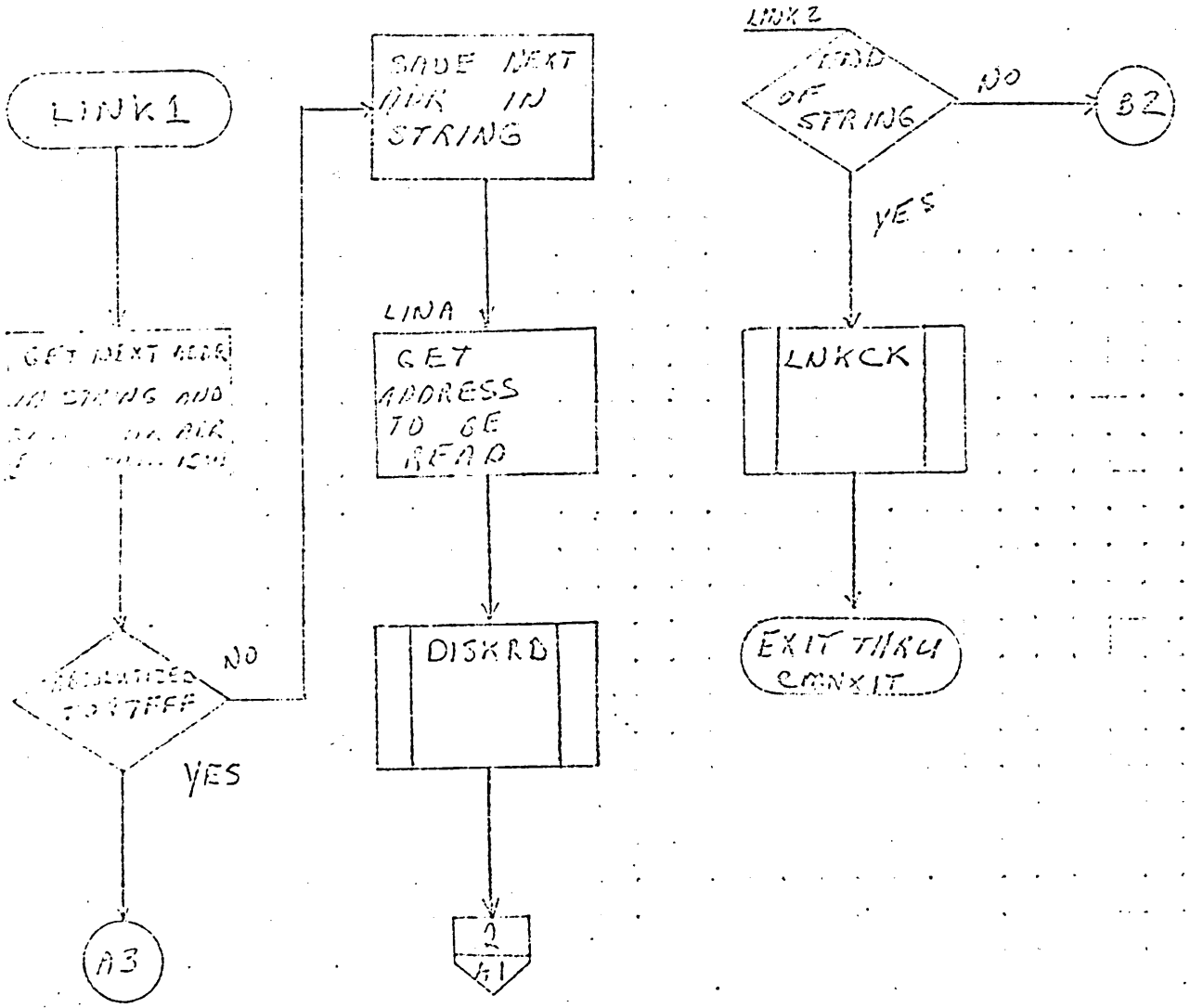
SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS TM5	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE LSTOT1		PROJECT MGR.			
PAGE 1 OF 3		PROJECT NAME 1700 MSOS			
NUMBER	ISSUE DATE	TASK NO.			
DRAWN BY	DATE	TASK NAME			

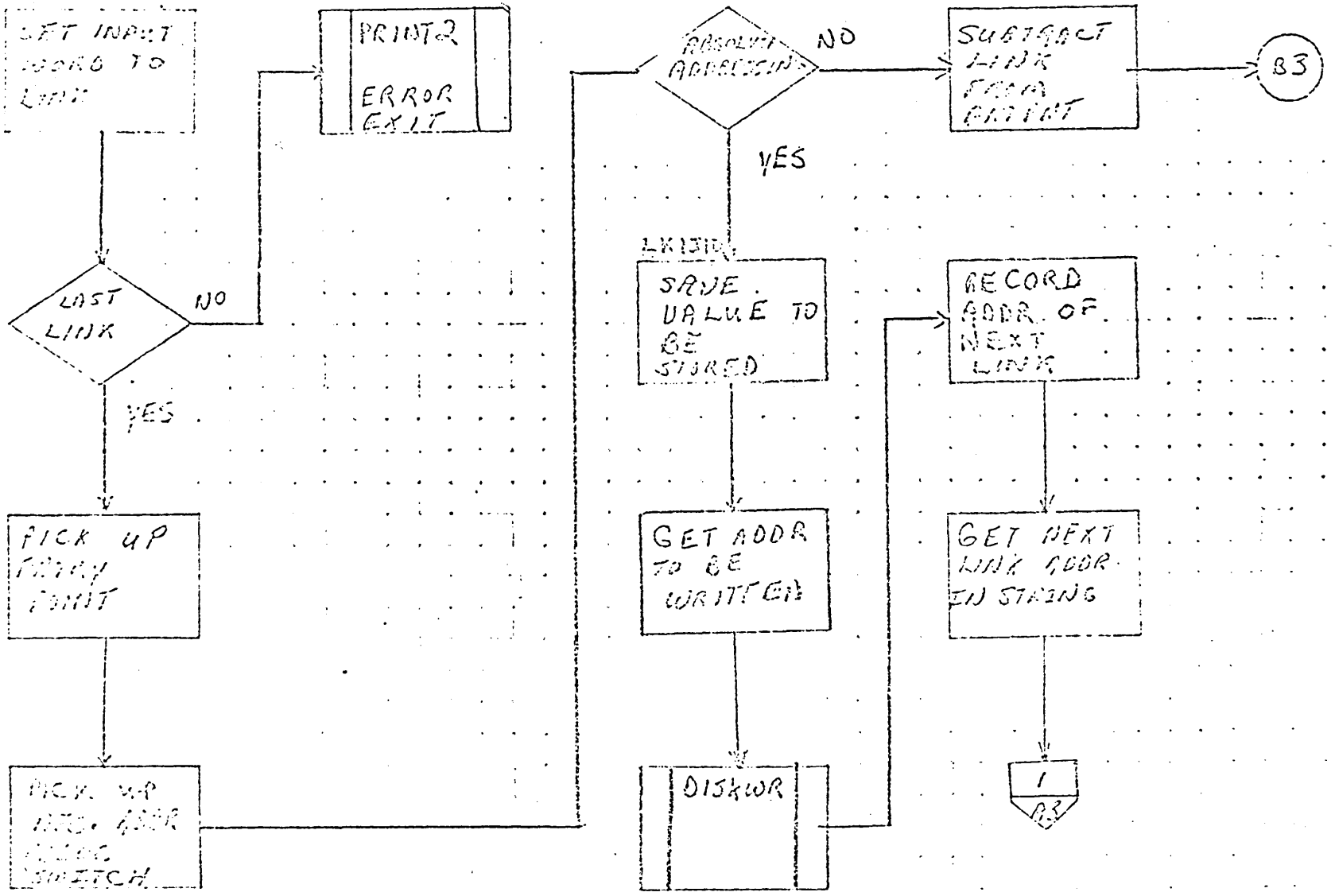
PRINTED IN U.S.A.



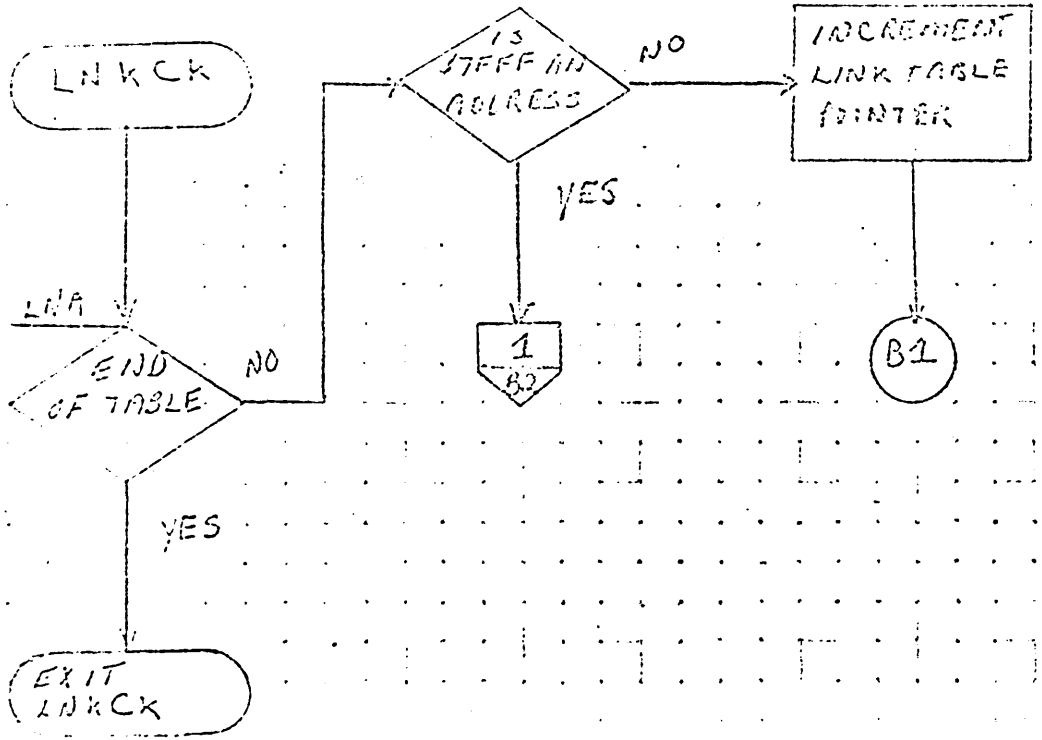
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 1.0C MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LSTCT1	PROJECT MGR.			
	PAGE 2 OF 2	PROJECT NAME 1700 MISC.			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE [illegible]	TASK NAME [illegible]			



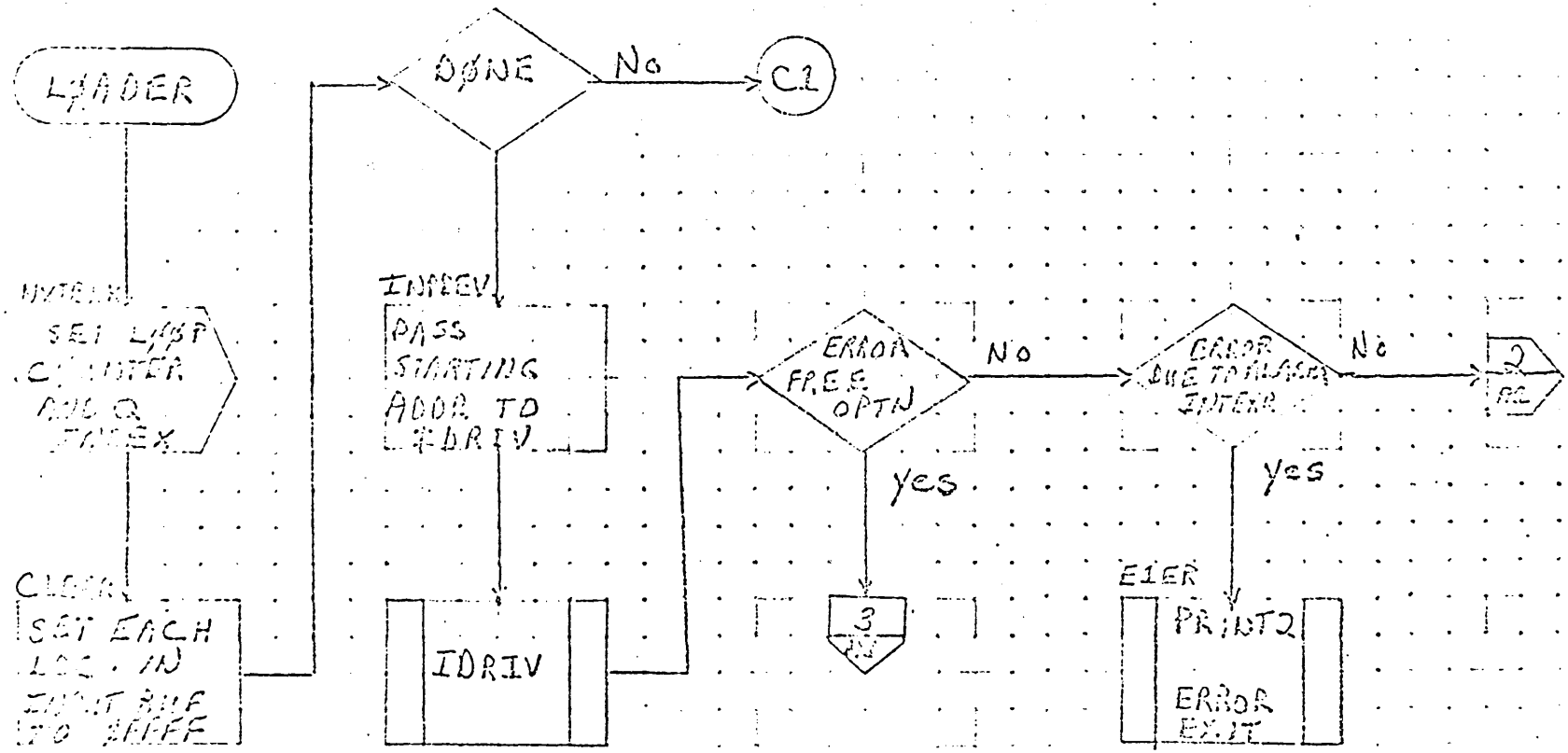
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOW CHART SECTION TABLE	DOCUMENT CLASS: EMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TYPE: LINK 1		PROJECT MGR.			
	PAGE 1 OF 3		PROJECT NAME: 1700 NOS			
	AUTHOR: ISSUE DATE		TASK NO.			
	PROGRAMMER: C.C.	DATE: 6/6/71	TASK NAME: PART 1 LOADER			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART X DECISION TABLE OTHER	DOCUMENT CLASS IAS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LINK 3.1	PROJECT MGR.			
	PAGE 2 OF 3	PROJECT NAME 1700 MOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C.C.C. DATE 4/20/71	TASK NAME PART 1 LINKER			

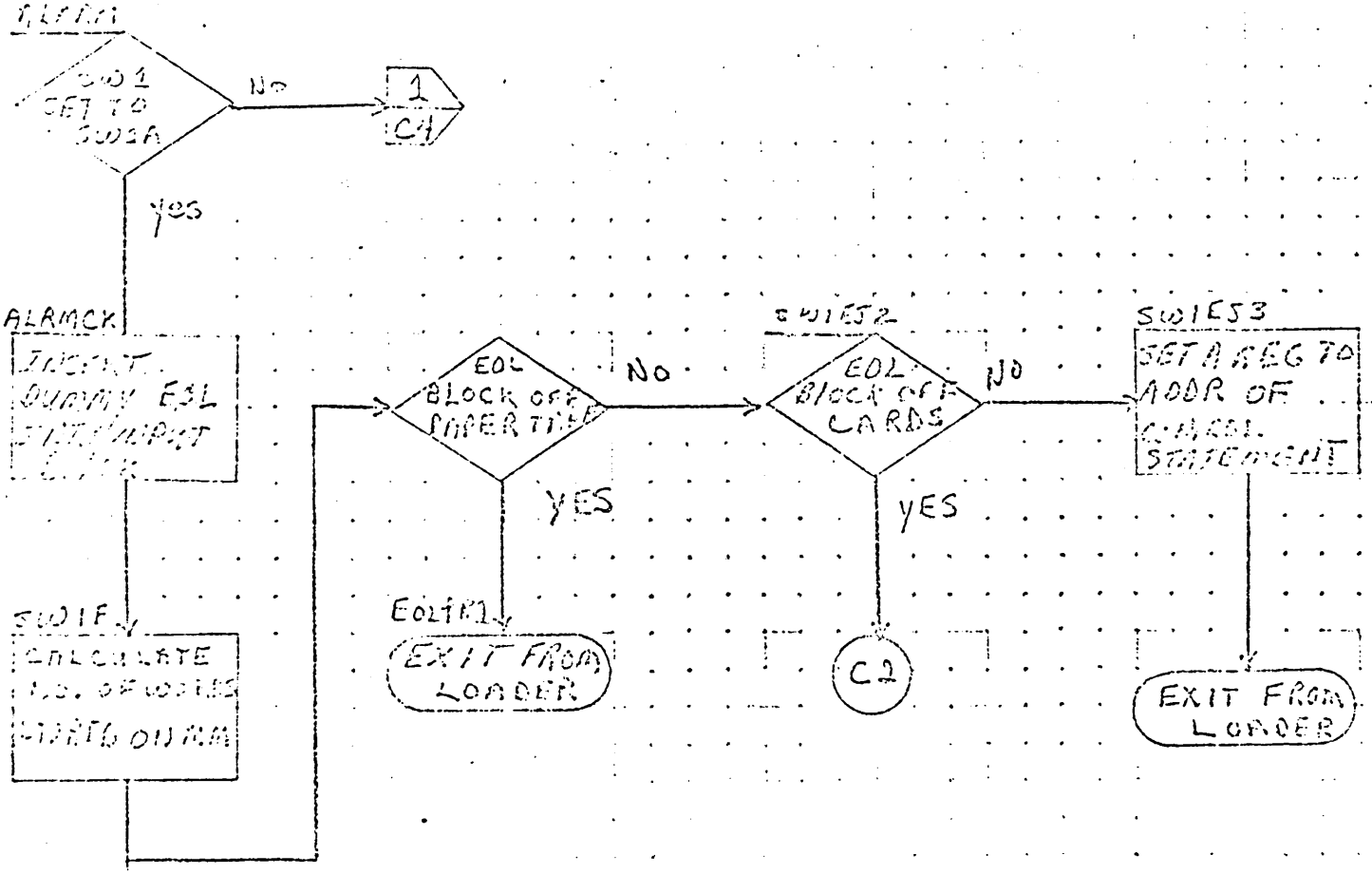


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: TMS MACH. TYPE: 1700	PROJECT NO.		APPROVED	DATE
	DOCUMENT TITLE: LINK 1_1	PROJECT MGR.			
	PAGE 3 OF 3	PROJECT NAME: 1700 MSOS			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY: GDG DATE: 4/21/67	TASK NAME: [illegible]			



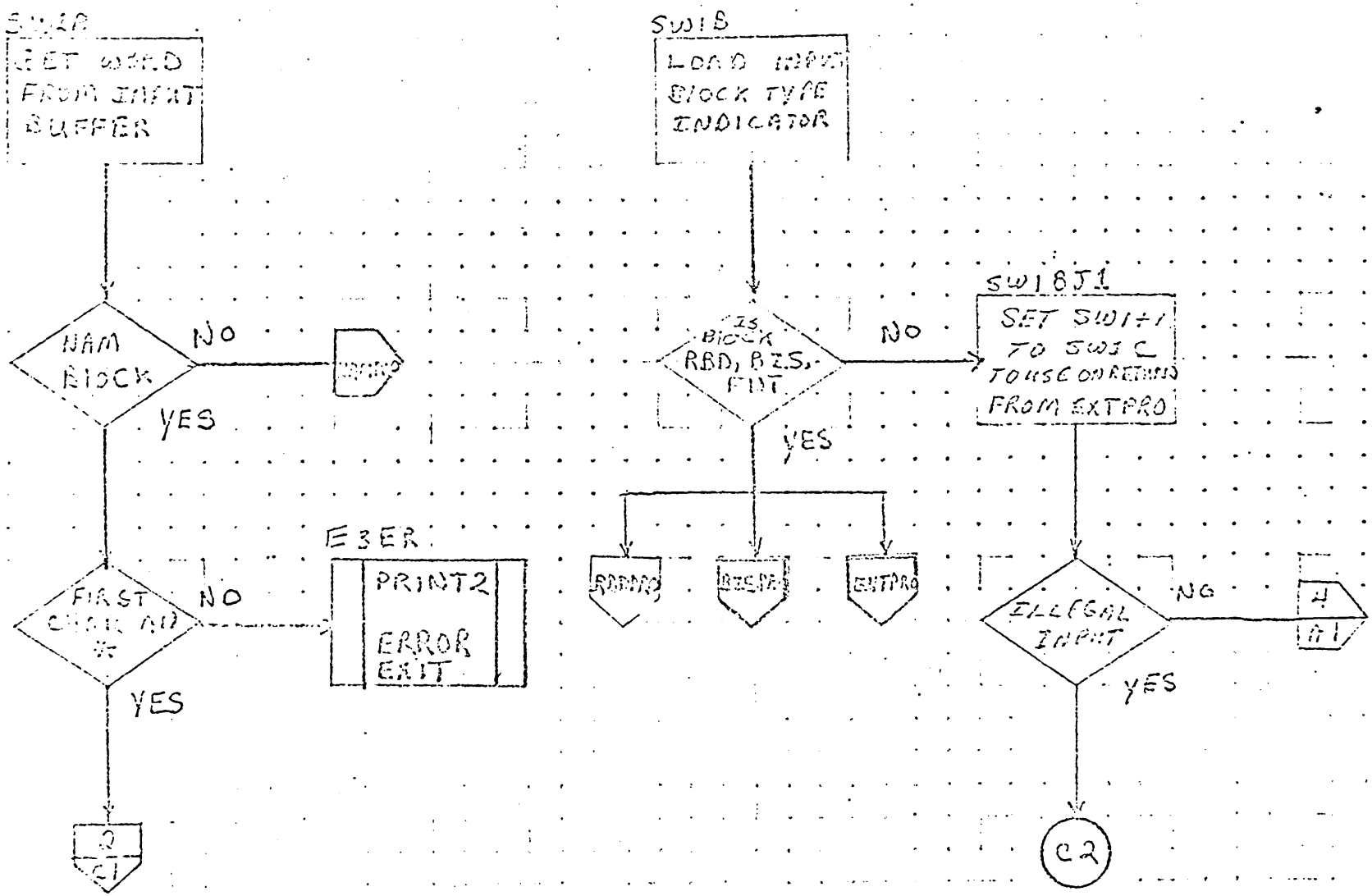
AUG 9 19..

CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LOADR1	PROJECT MGR.			
	PAGE 1 OF 4	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY GDG DATE 8/[illegible]	TASK NAME PART1 LOADER			



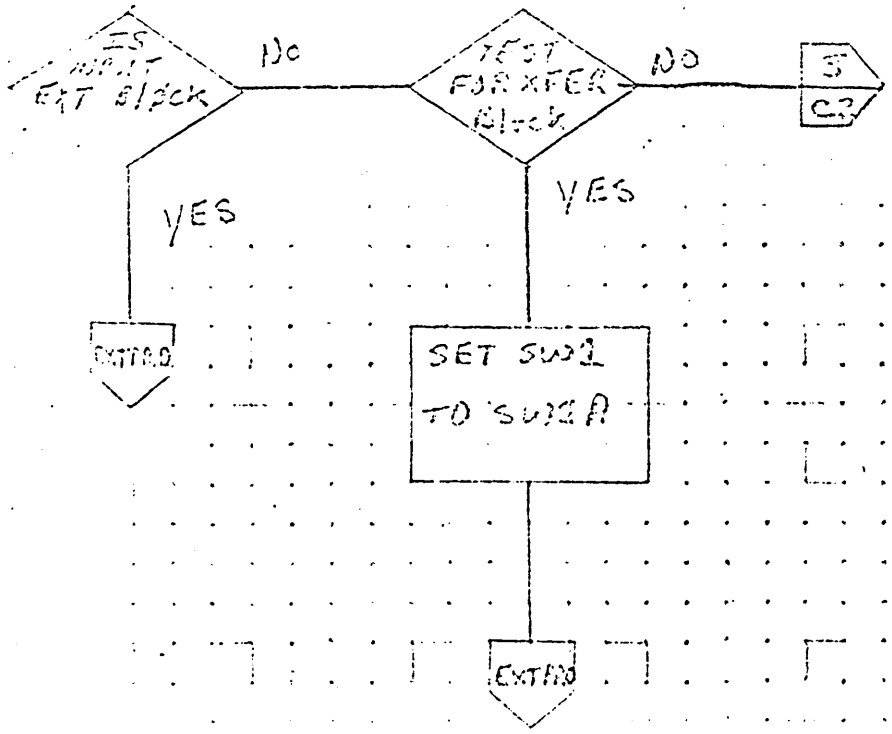
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LOADR1	PROJECT MGR.			
	PAGE 2 OF 4	PROJECT NAME 1700 MASS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY GNG DATE 6 Aug 71	TASK NAME PART 1 LOADER			

AUG 6 1971

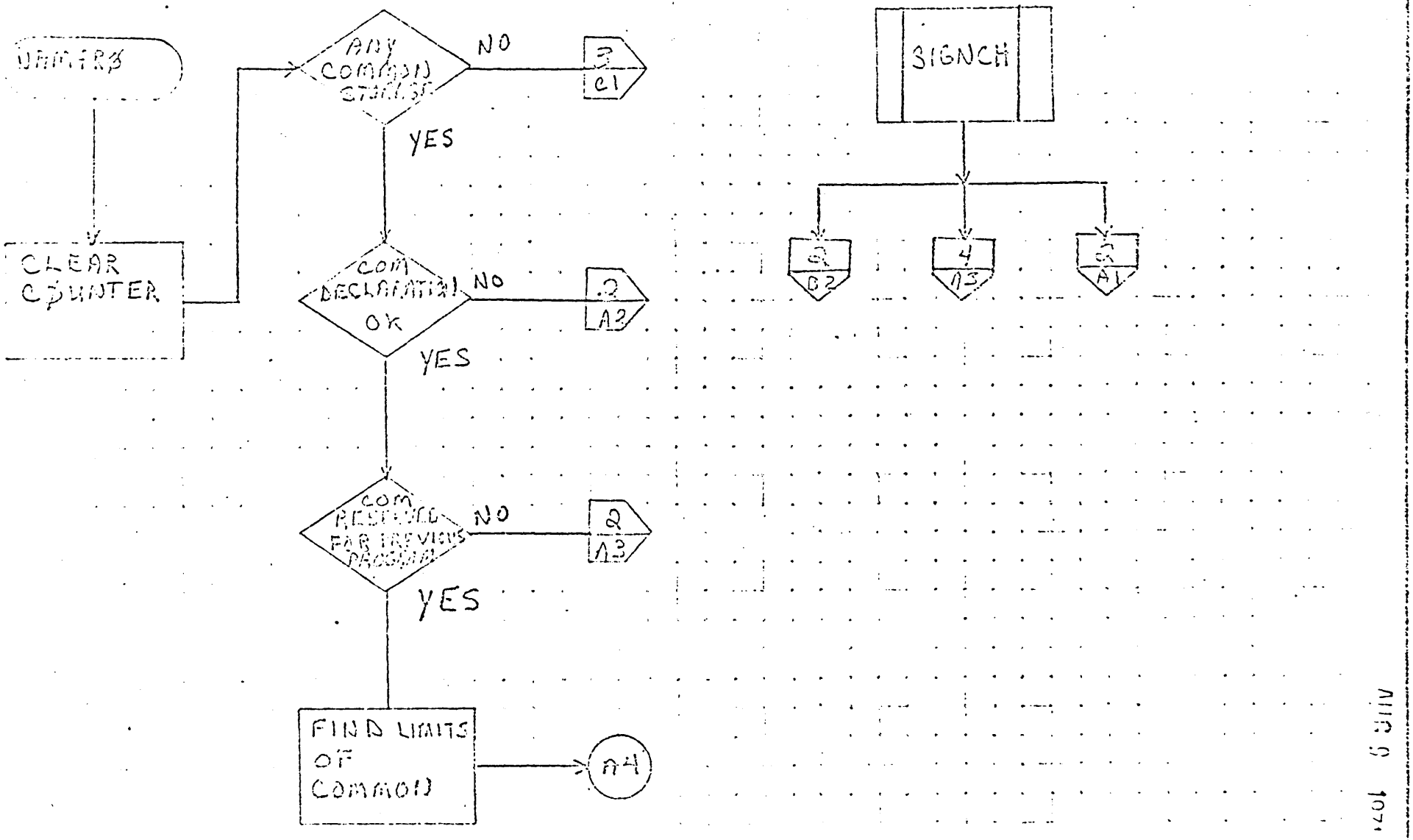


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE LOADR1		PROJECT MGR.		
	PAGE 3 OF 4		PROJECT NAME 1700 MSOS		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY GDG	DATE 4/24/71	TASK NAME PART1 LOADER		

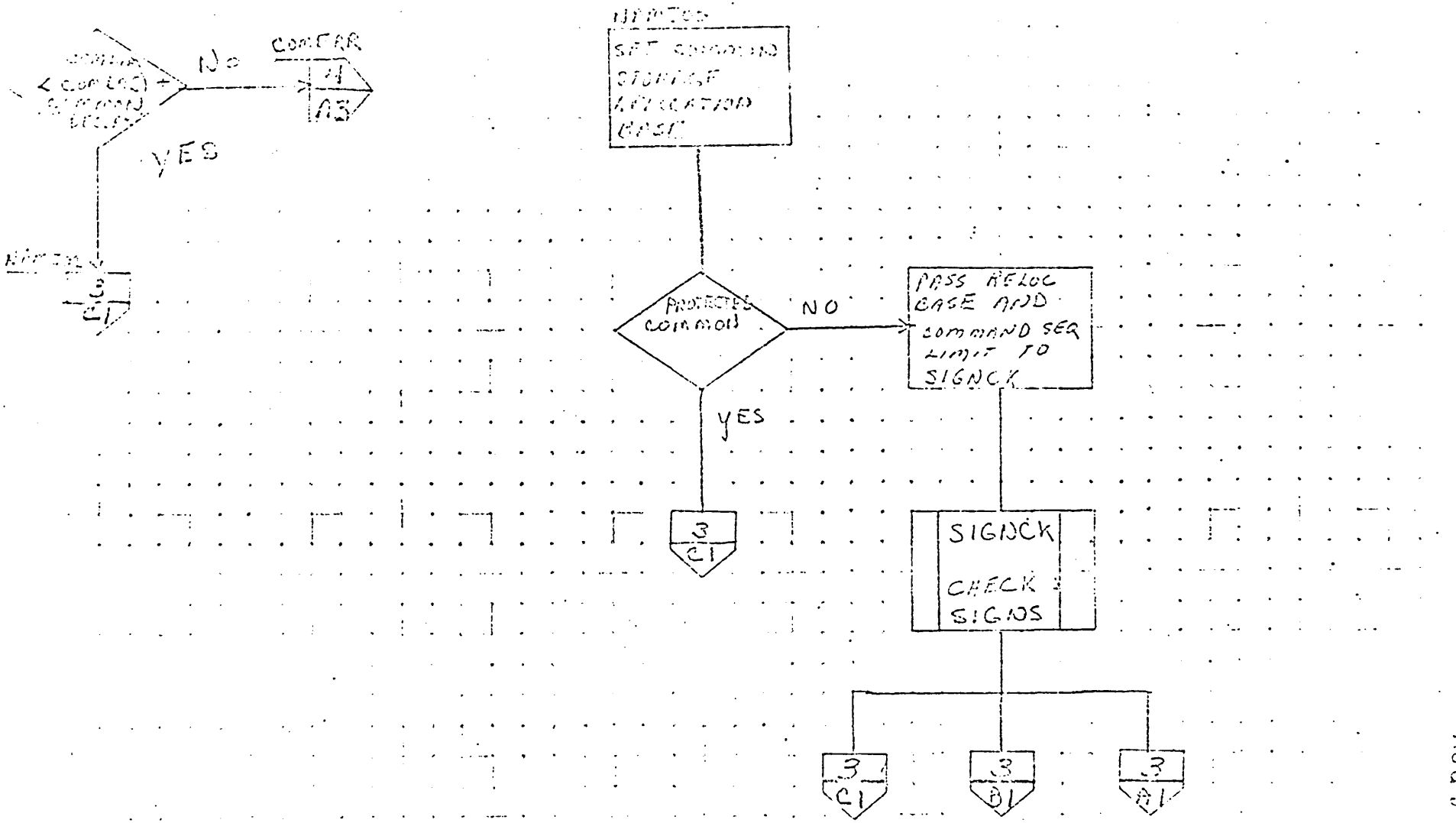
AUG 9 1971



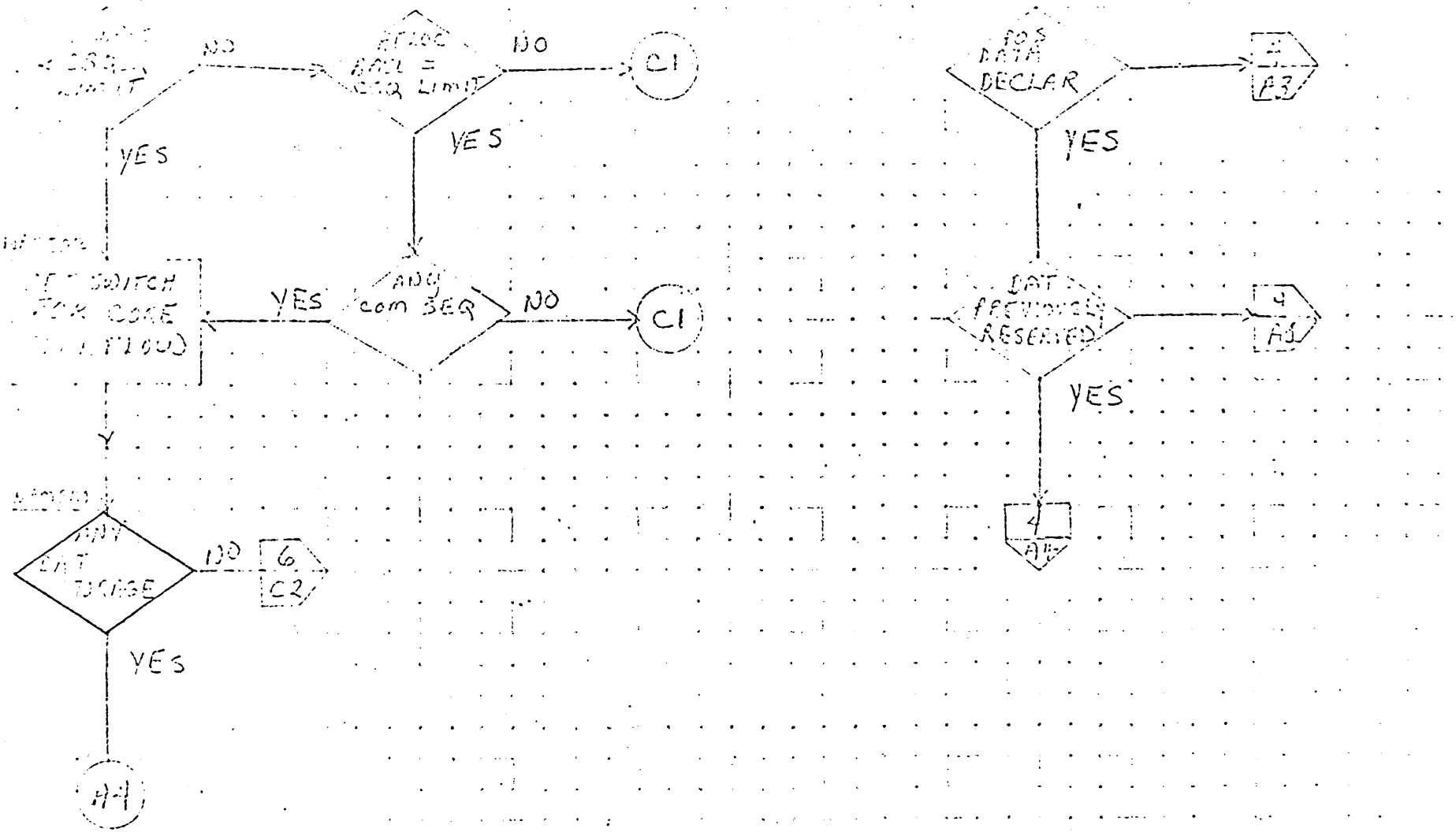
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TAS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LOADR1	PROJECT MGR.			
	PAGE 4 OF 4	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY GDC DATE [illegible]	TASK NAME [illegible]			



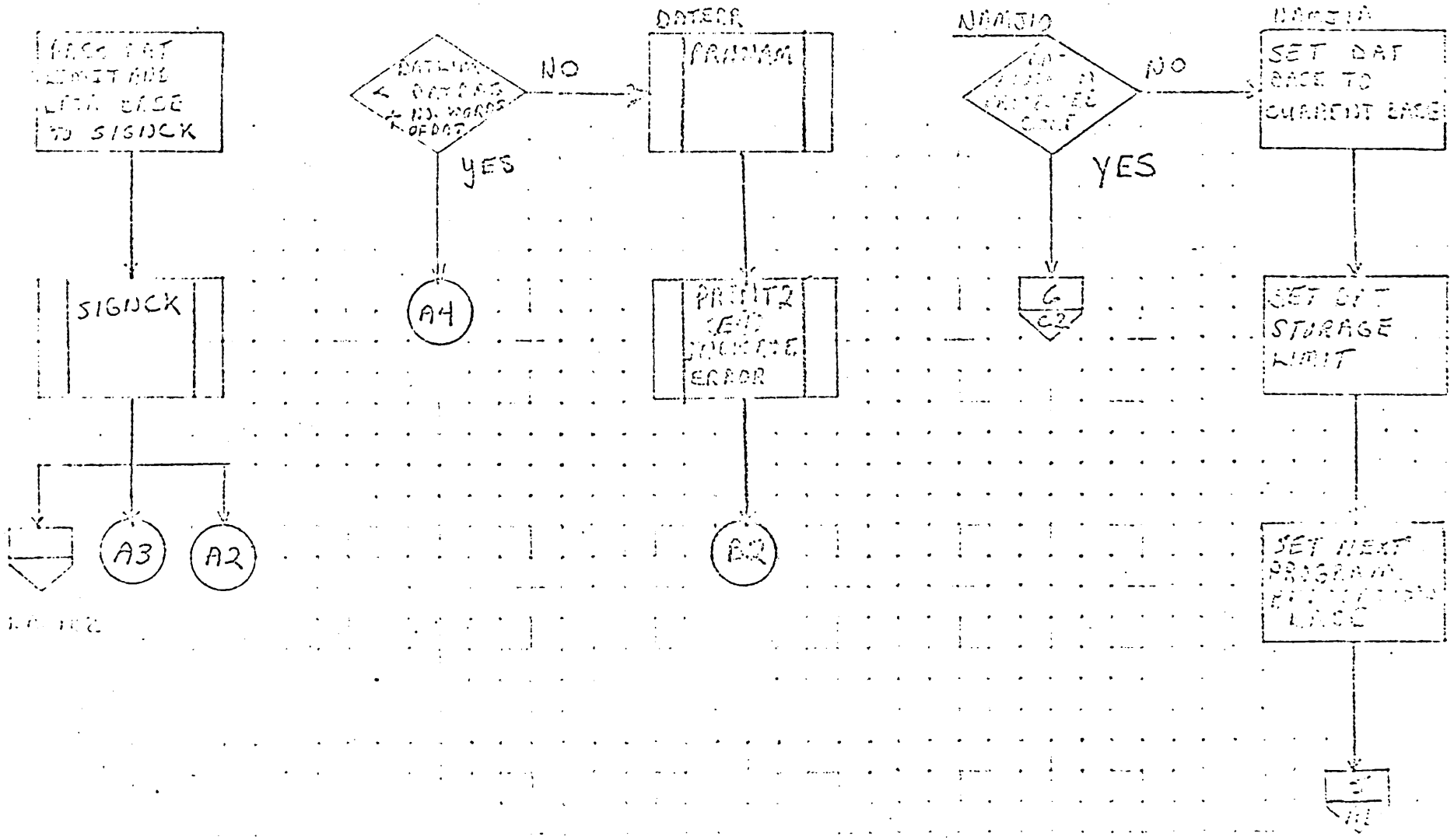
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS T.I.P. MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE NAMAR1	PROJECT MGR.			
	PAGE 1 OF 10	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY G.D.C. DATE 6/5/67	TASK NAME PART 1 OF 2			



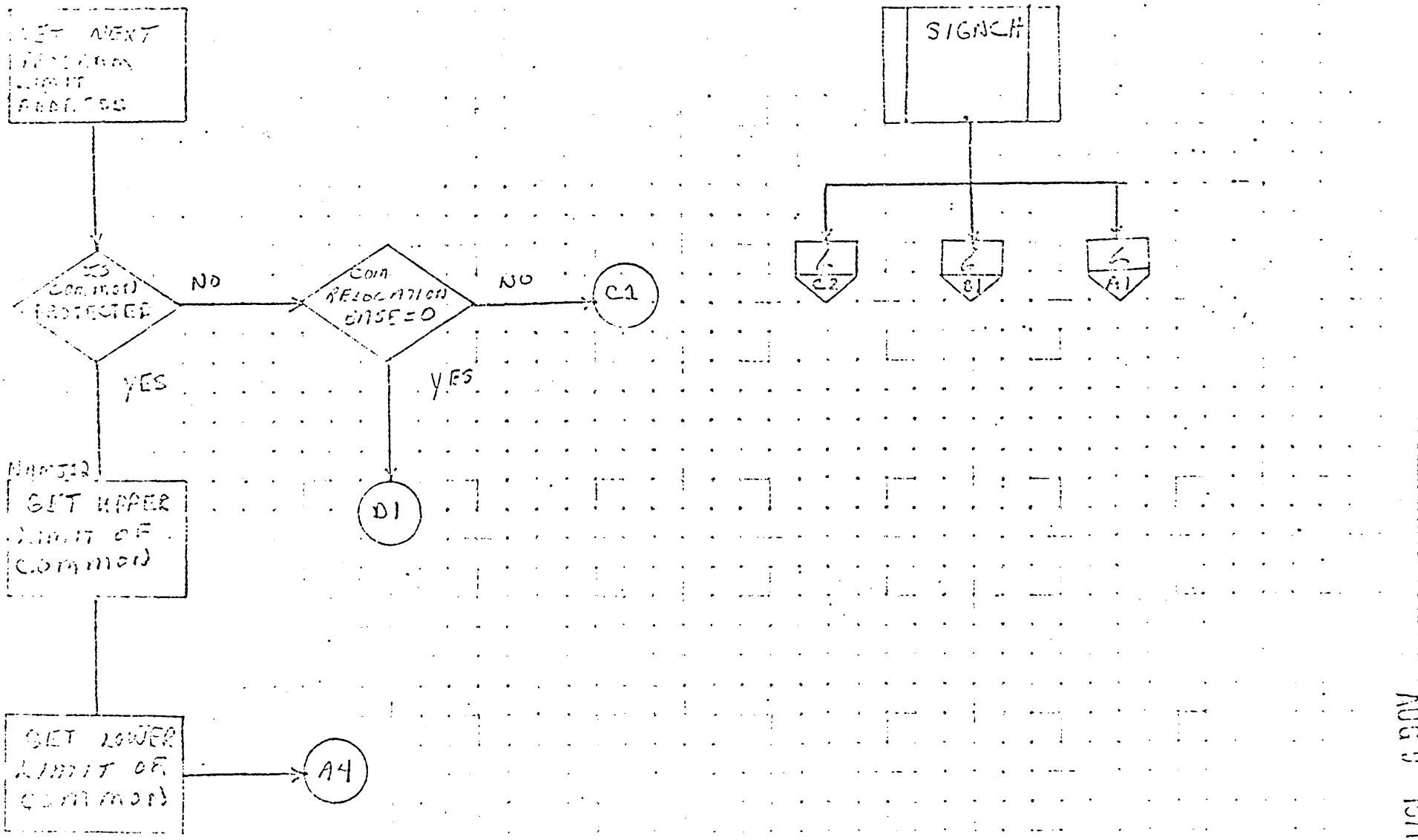
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS T (illegible)	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE MAINFR1		PROJECT MGR.		
FLOWCHART ☐		PAGE 2 OF 16	PROJECT NAME 1700 MSOS		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY GDC	DATE (illegible)	TASK NAME PART 1 LOADER		



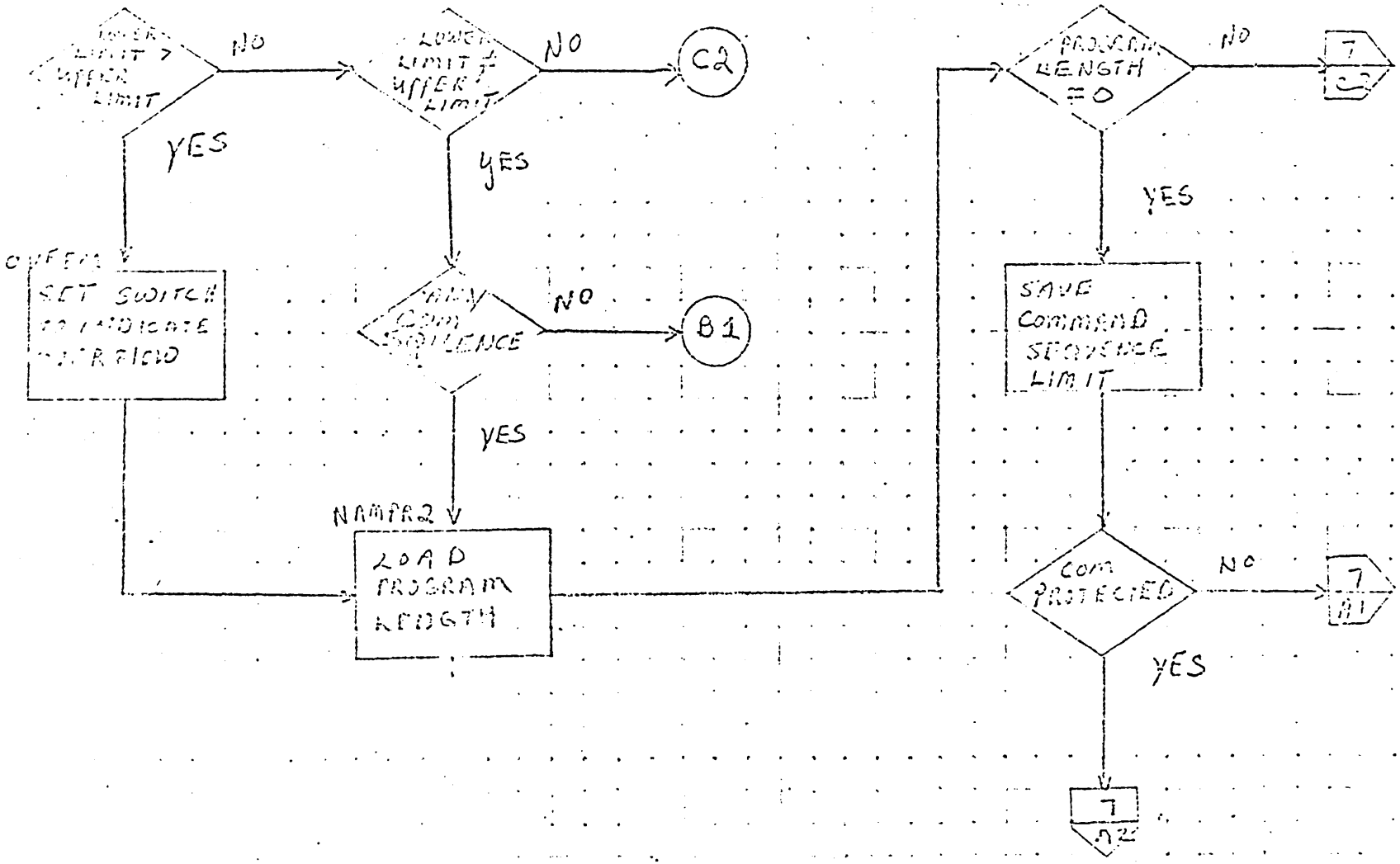
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS: JMS	MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE: LABEL 1		PROJECT MGR.			
FLOWCHART ☐		PAGE 2 OF 10	PROJECT NAME: 1700 MSOS			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY: GDG	DATE: 6/3/71	TASK NAME: PART 1 LOADER			



CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS JMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE NAMBR1		PROJECT MGR.			
	PAGE 1 OF 10		PROJECT NAME 1700 MSOS			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY GDG	DATE 11-1-71	TASK NAME PART1 LOADER			

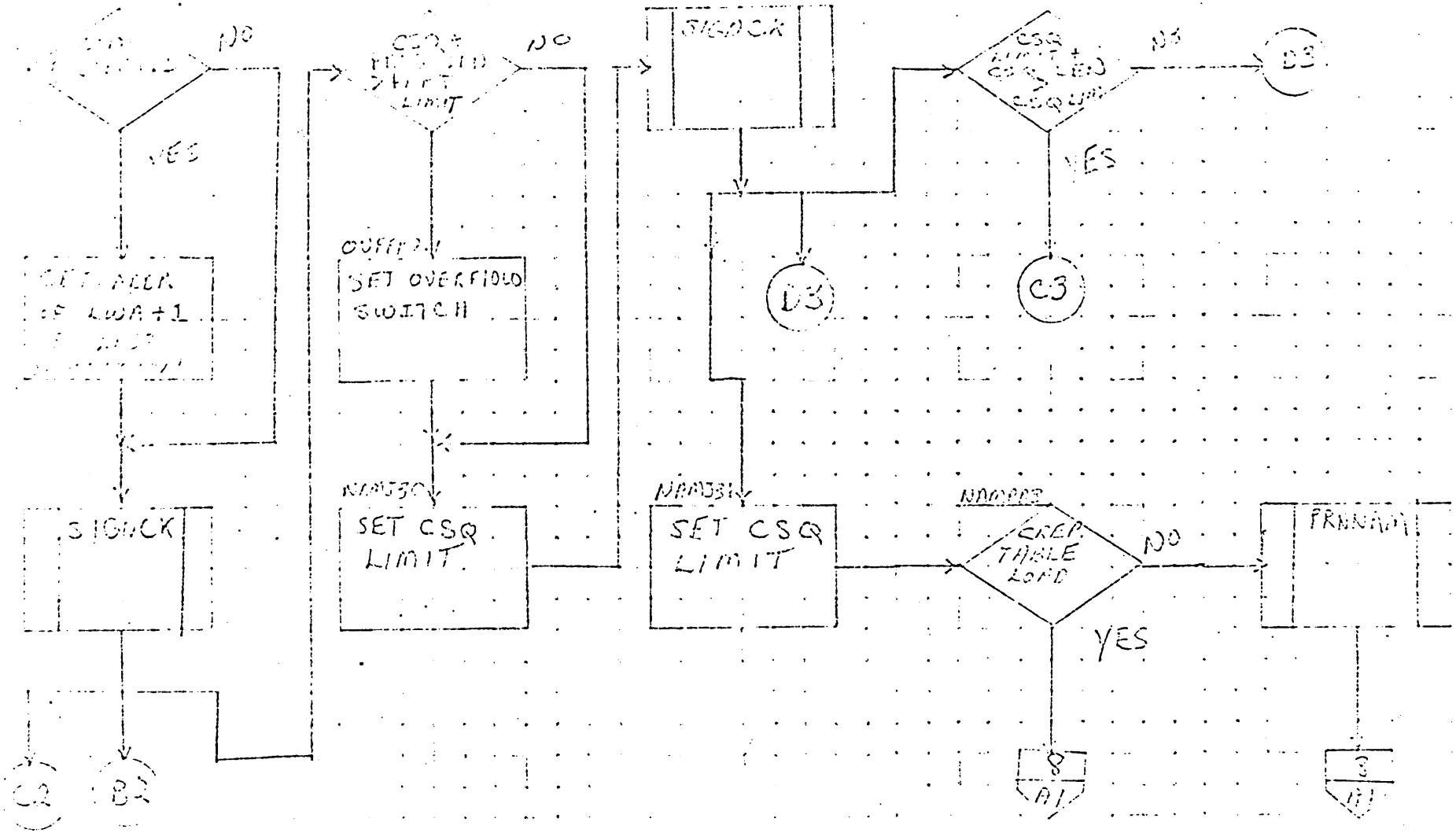


CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE MAIN1		PROJECT MGR.			
		PAGE 5 OF 10	PROJECT NAME 1700 MSOS			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY GNG	DATE 4/[illegible]	TASK NAME [illegible]			

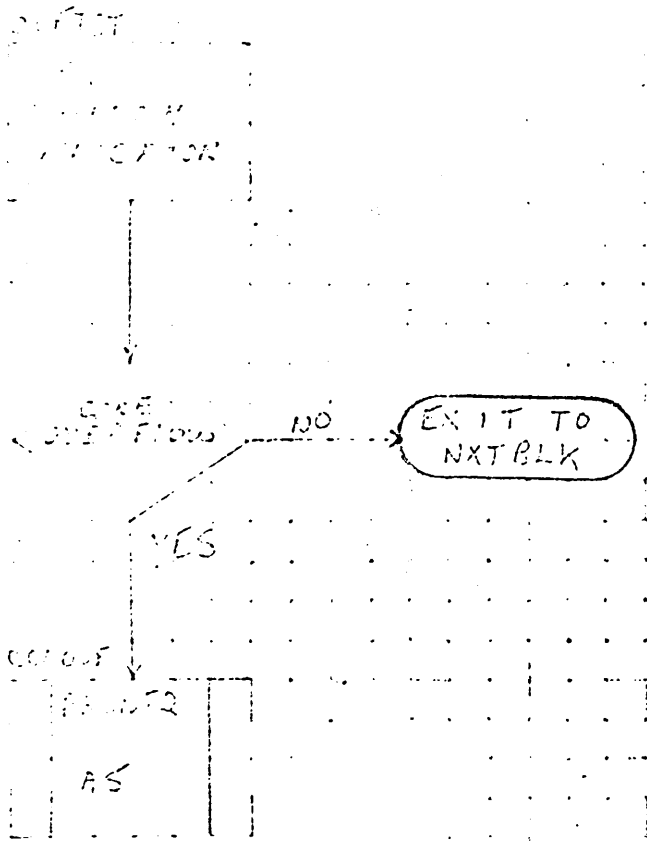


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS: EMS MACH. TYPE: 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE: NAMTR1	PROJECT MGR.			
	PAGE 6 OF 10	PROJECT NAME: 1700 MSOS			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY: DC DATE: 6/1972	TASK NAME: NAMTR1 SAMPLE			

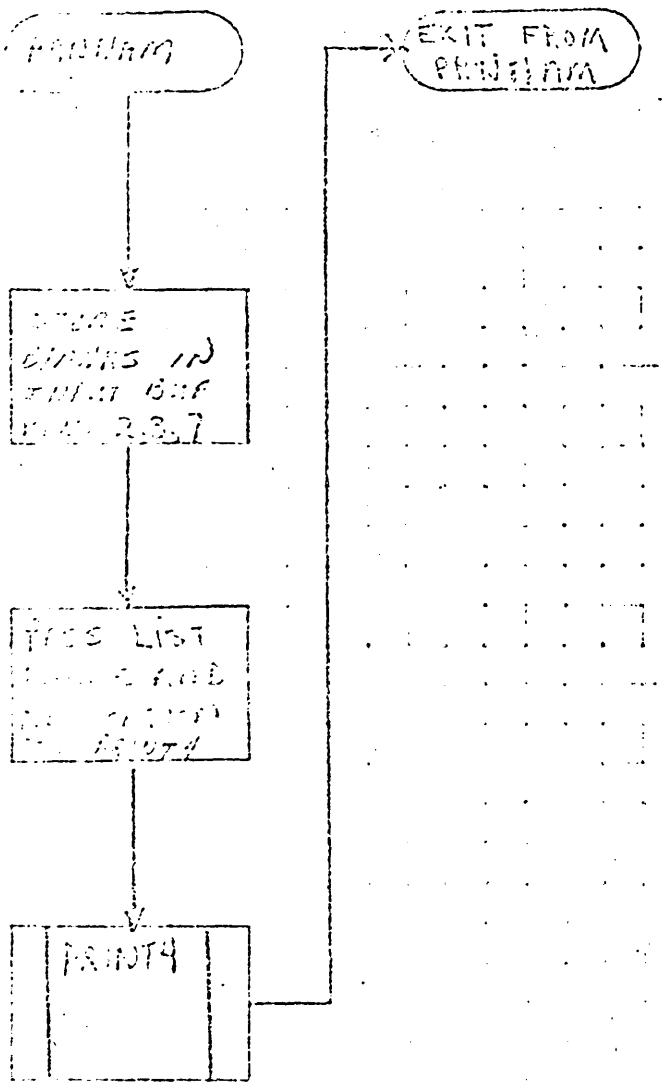
AUG 3 1971



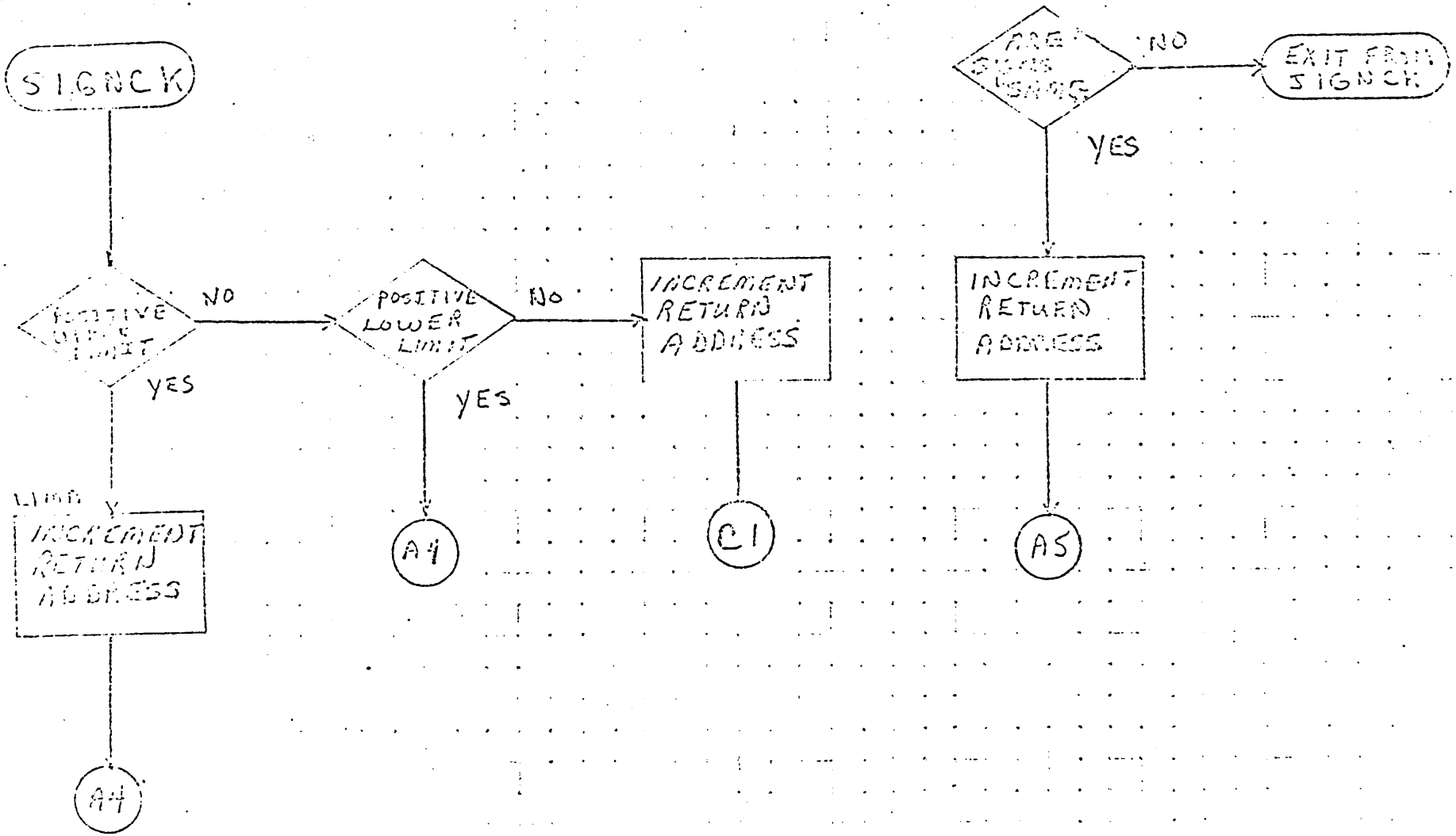
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE 1700	PROJECT NO.	REV.	APPROVED	DATE
	DOCUMENT TITLE [illegible] 1	PROJECT MGR.			
	PAGE 7 OF 10	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE [illegible]	TASK NAME [illegible]			



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

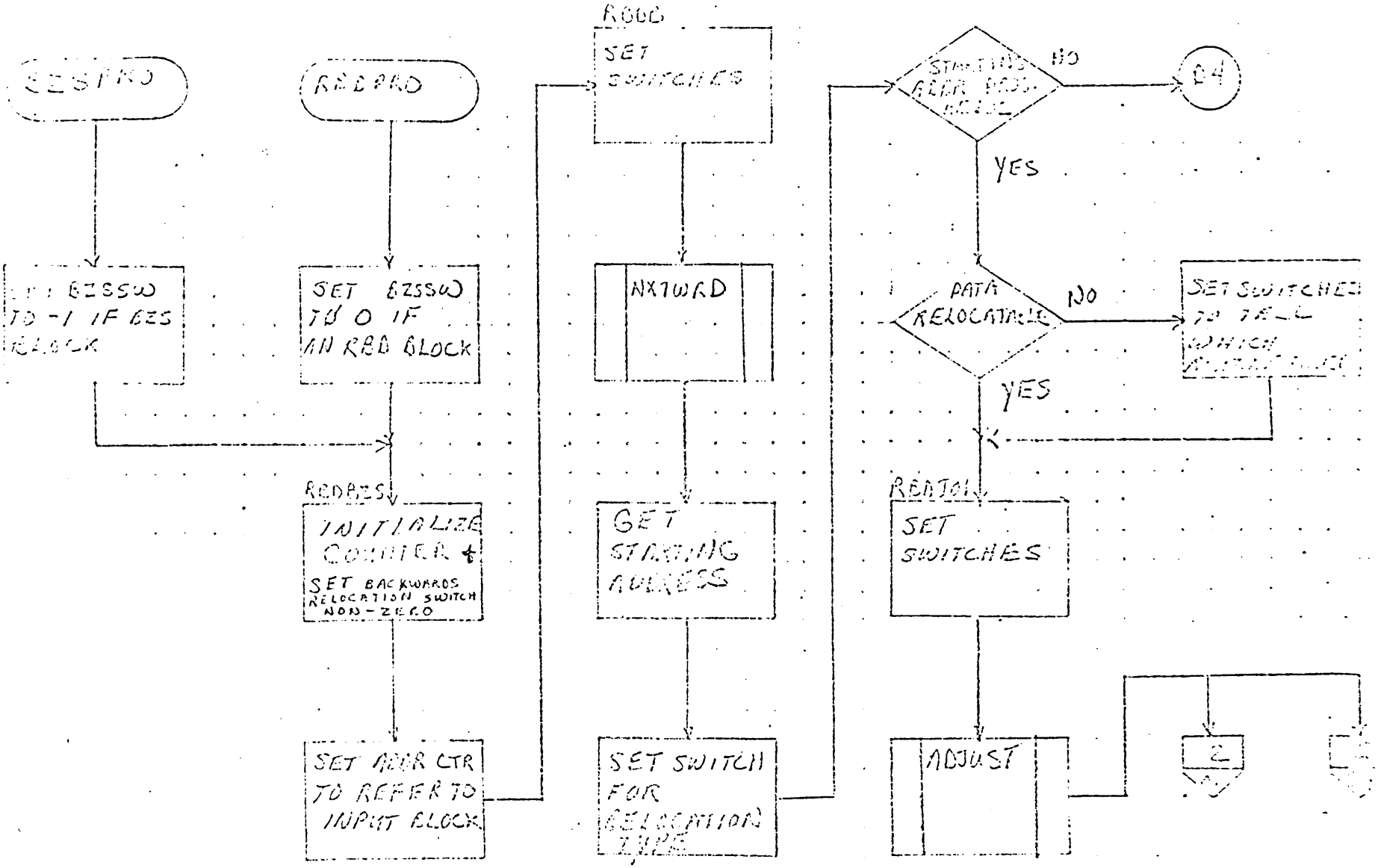


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐	DOCUMENT CLASS [illegible] MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
FLOWCHART ☐	DOCUMENT TITLE [illegible] R1	PROJECT MGR.			
DECISION TABLE ☐	PAGE [illegible] OF [illegible]	PROJECT NAME 1700 MSOS			
OTHER ☐	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY GDS DATE [illegible]	TASK NAME PART4 LOADER			

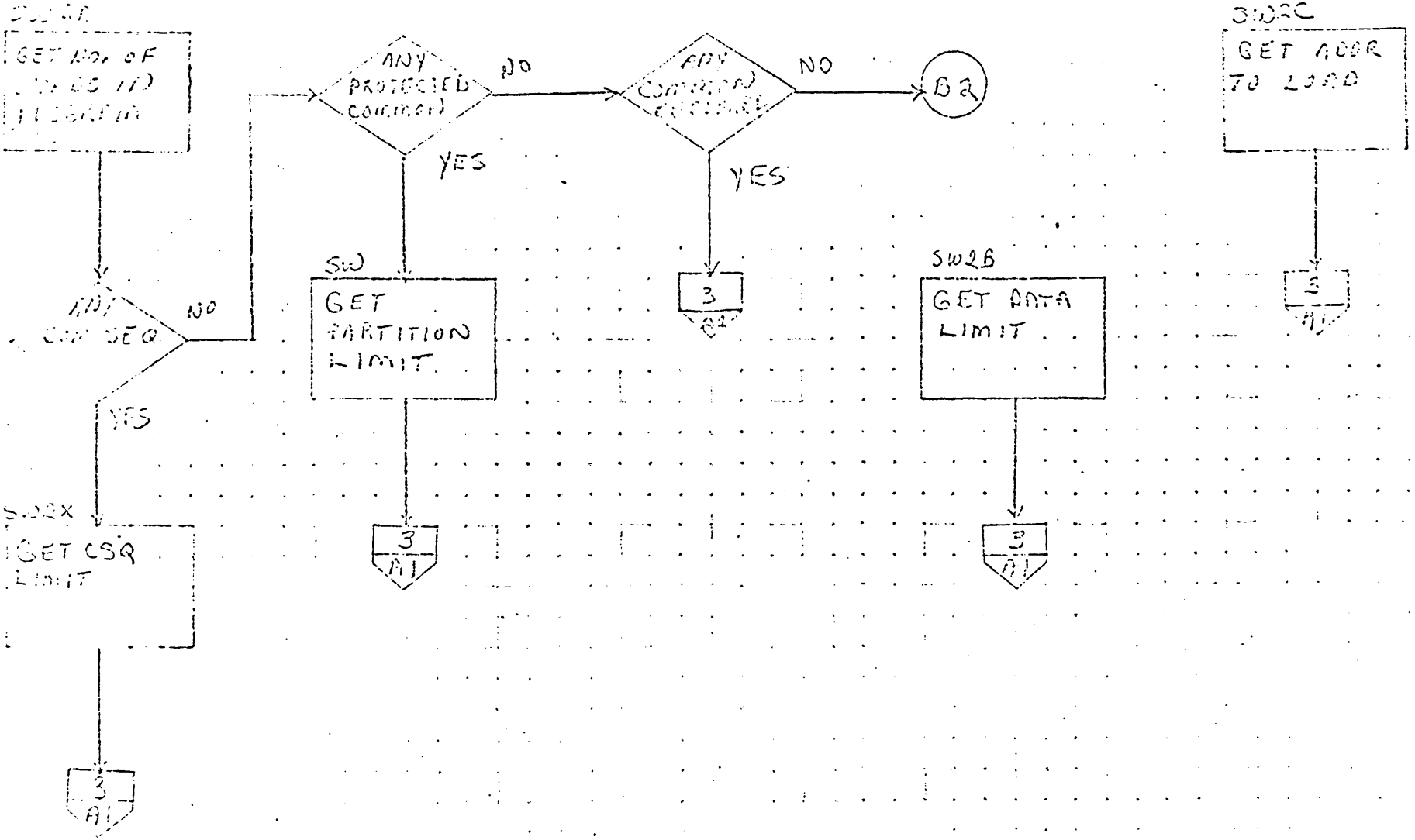


AUG 9 1971

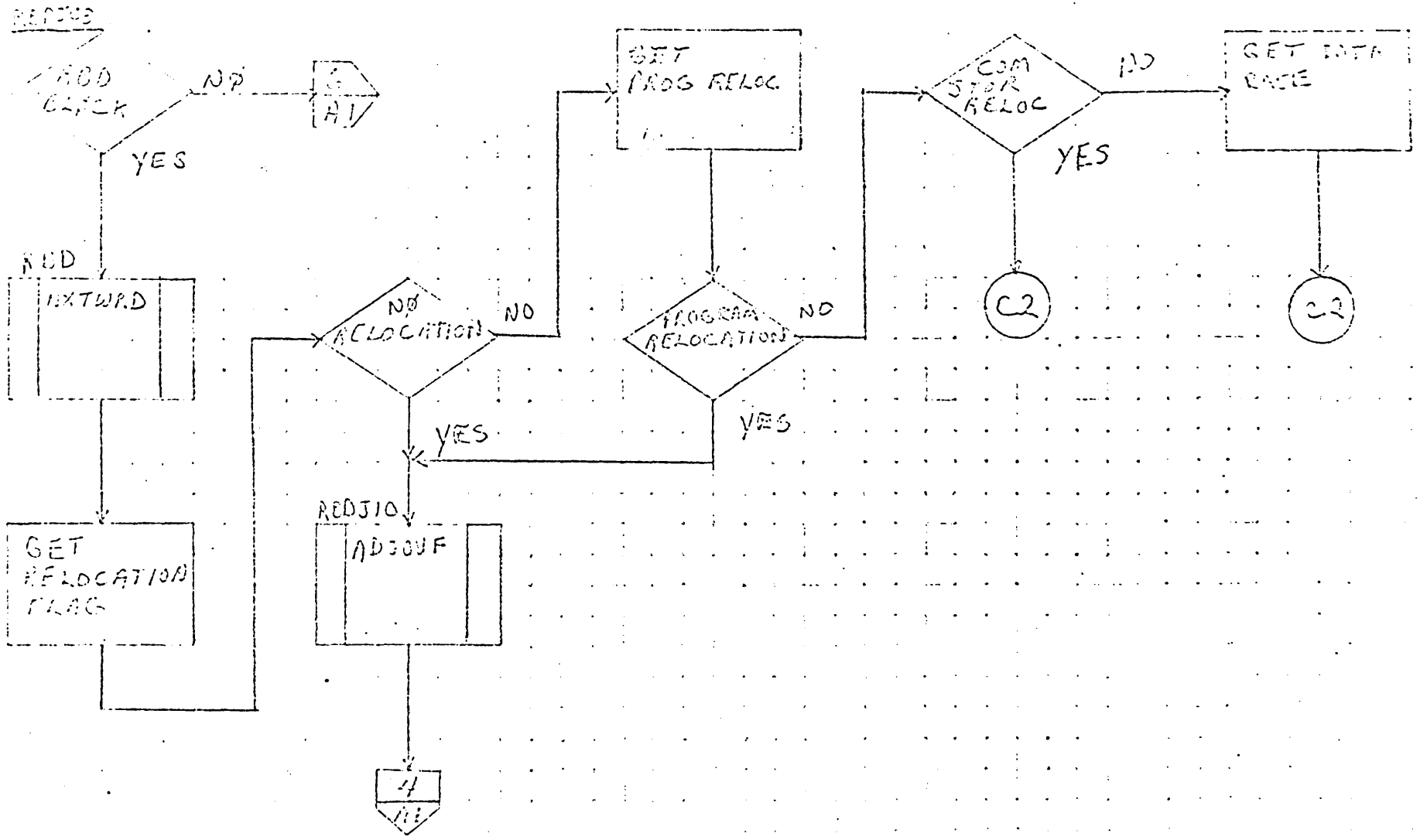
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE 1A-R1		PROJECT MGR.		
FLOWCHART ☐		PAGE 10 OF 10	PROJECT NAME 1700 MSOS		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY G.D.C.	DATE	TASK NAME		



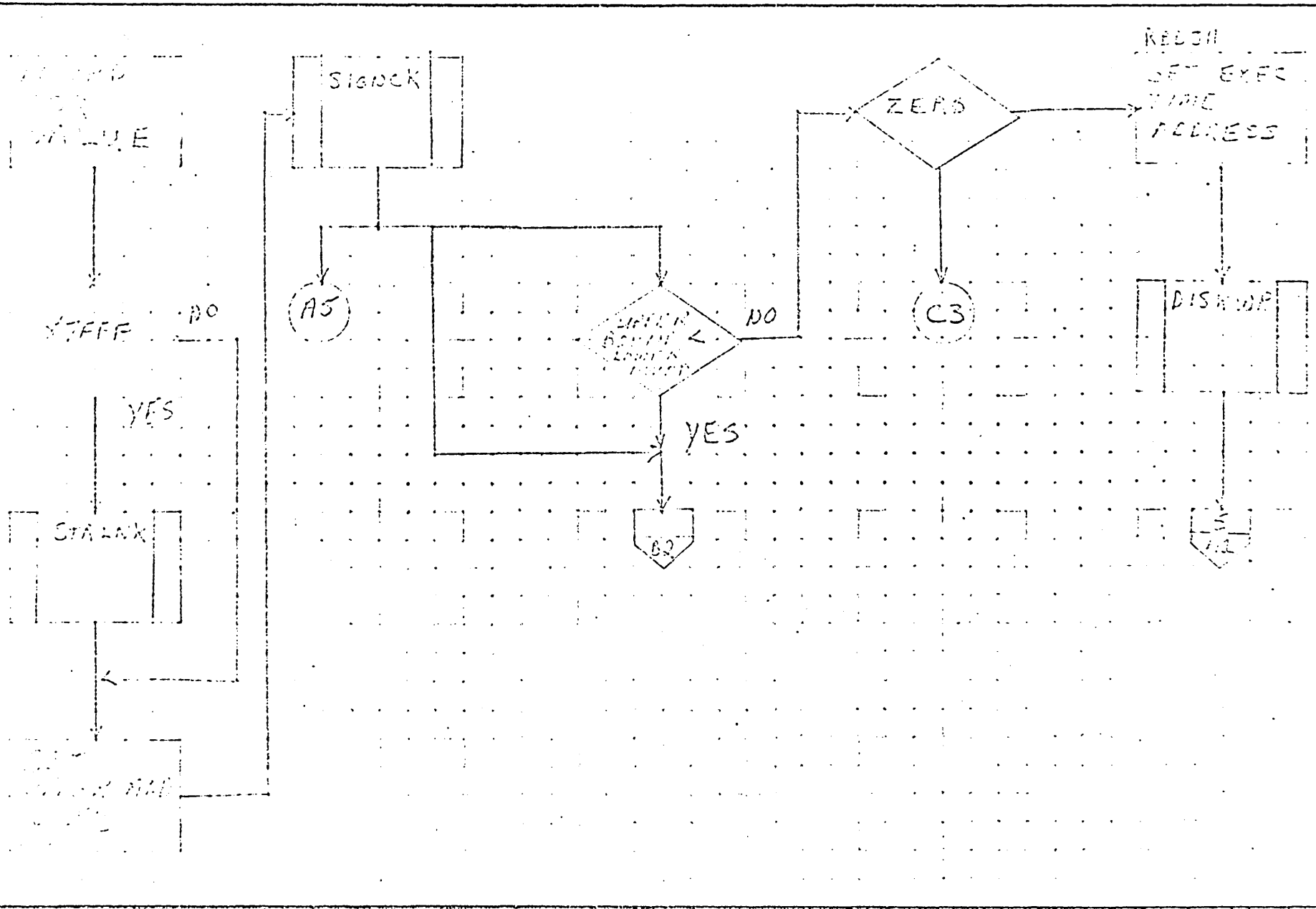
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ADDR 1	PROJECT MGR.			
	PAGE 1 OF 11	PROJECT NAME TMS RDOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY CAC DATE	TASK NAME PASS 1 READ			



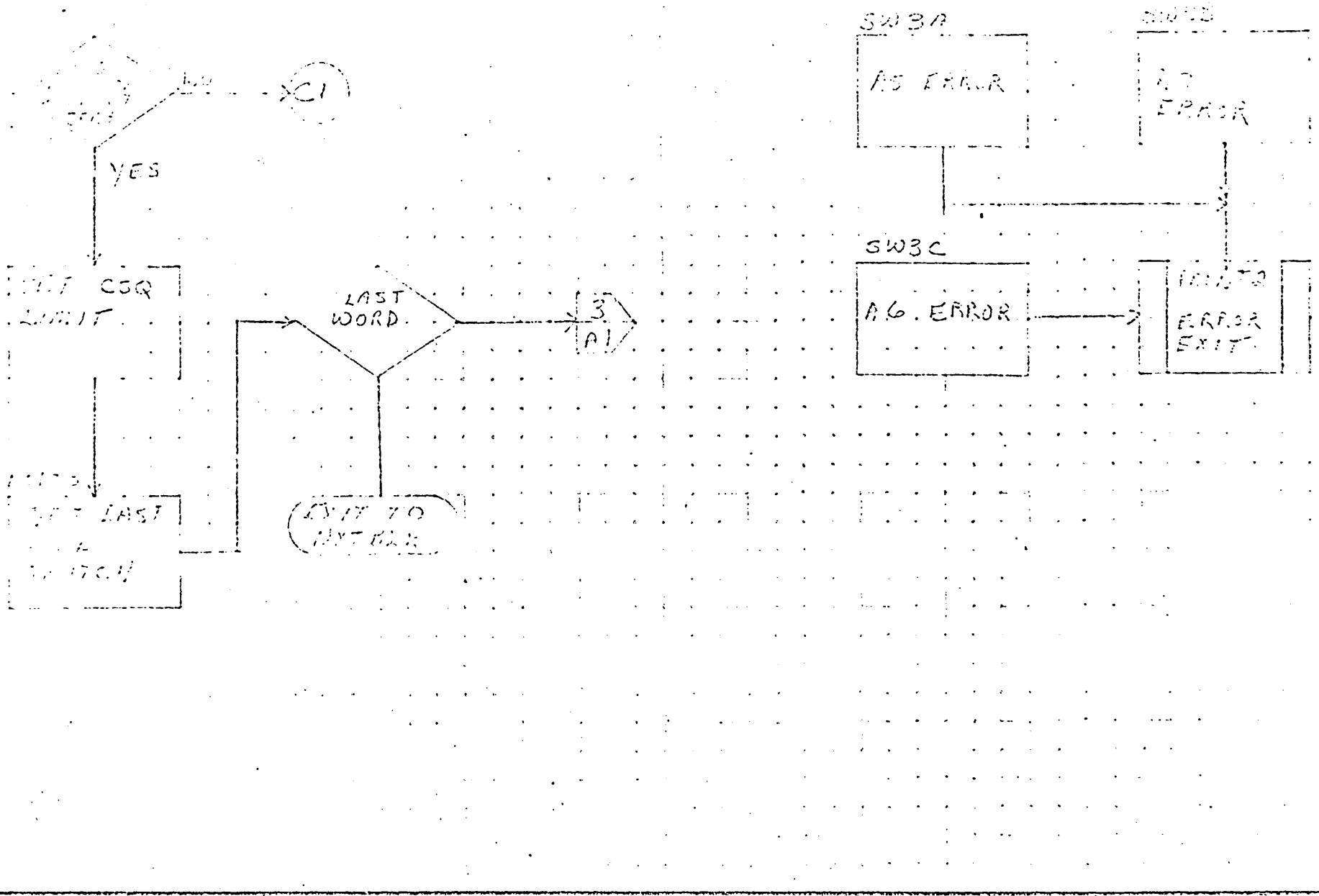
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE BMF71		PROJECT MGR.		
		PAGE 2 OF 11	PROJECT NAME 1700 MSOS		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IRC	MACH. TYPE 1700	PROJECT NO.	DCN	APPROVED / DATE
	DOCUMENT TITLE READ1		PROJECT MGR.		
		PAGE 3 OF 11	PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME PART1		

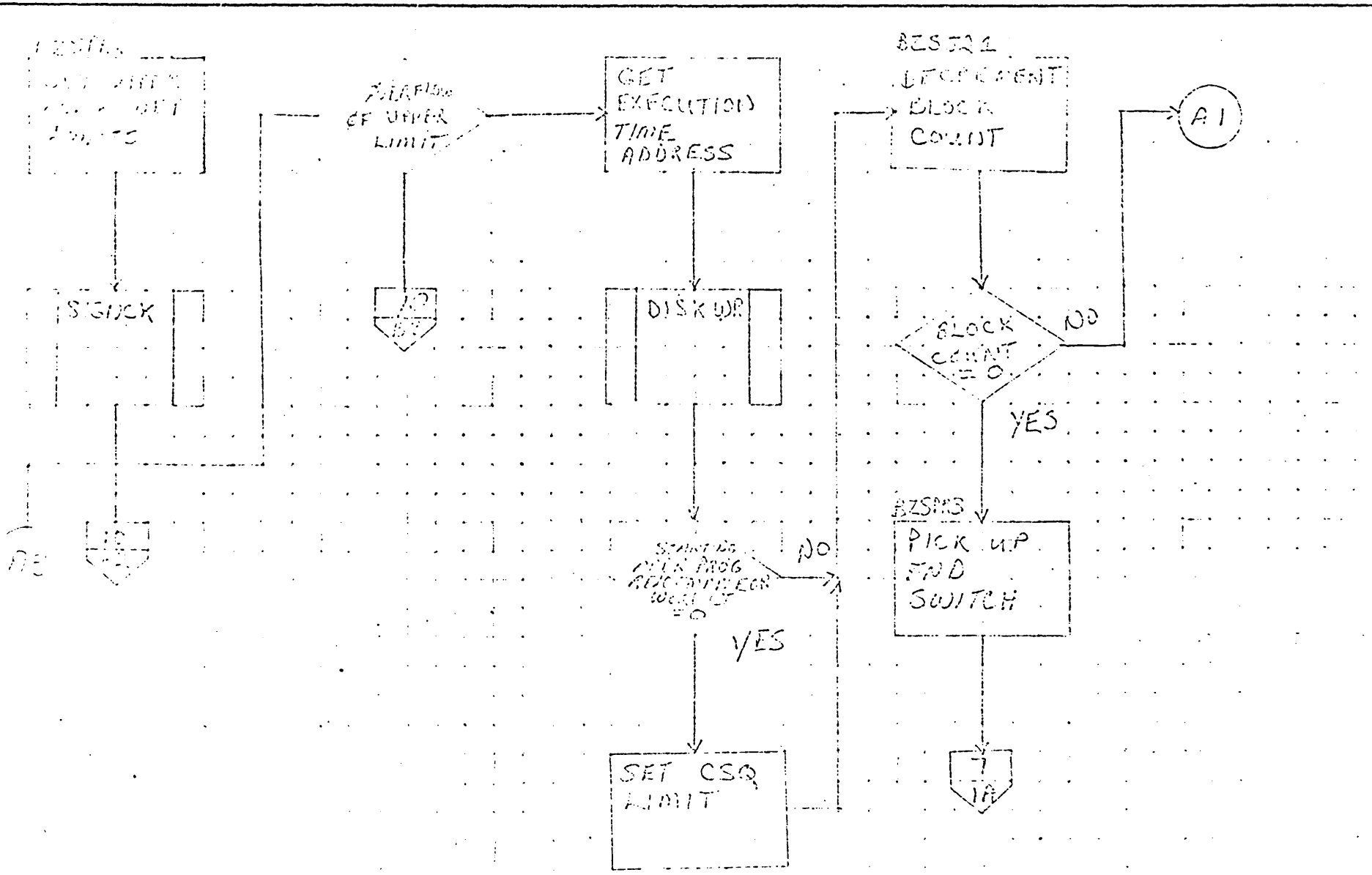


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE [illegible] 1	PROJECT MGR.			
	PAGE 11 OF 11	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN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



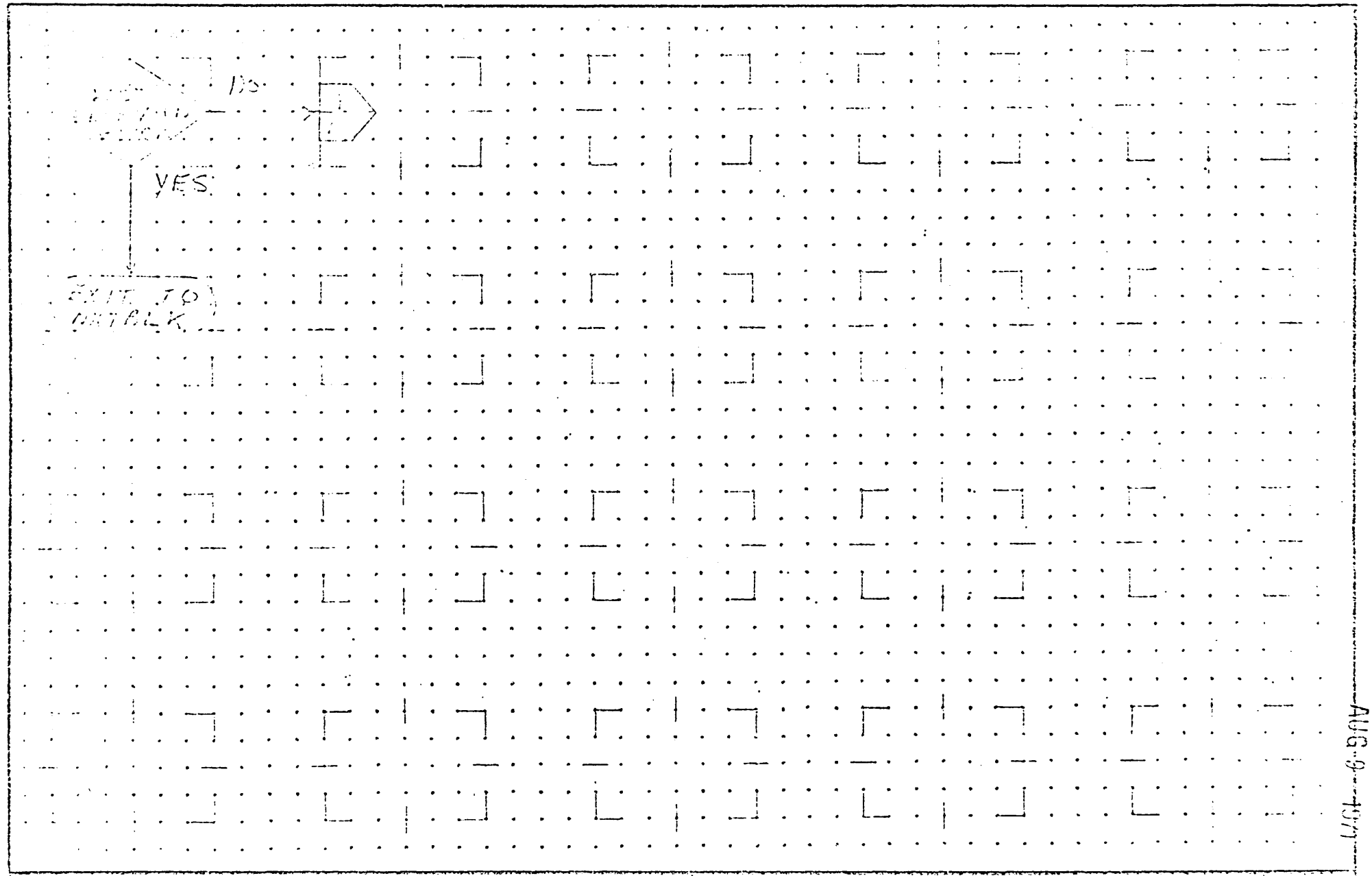
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS TM	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE PART 1		PROJECT MGR.			
FLOWCHART ☐		PAGE 5 OF 11	PROJECT NAME 1700 MASS			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY CLC	DATE 4/70	TASK NAME PART 1 CODE			

AUG 5 1971



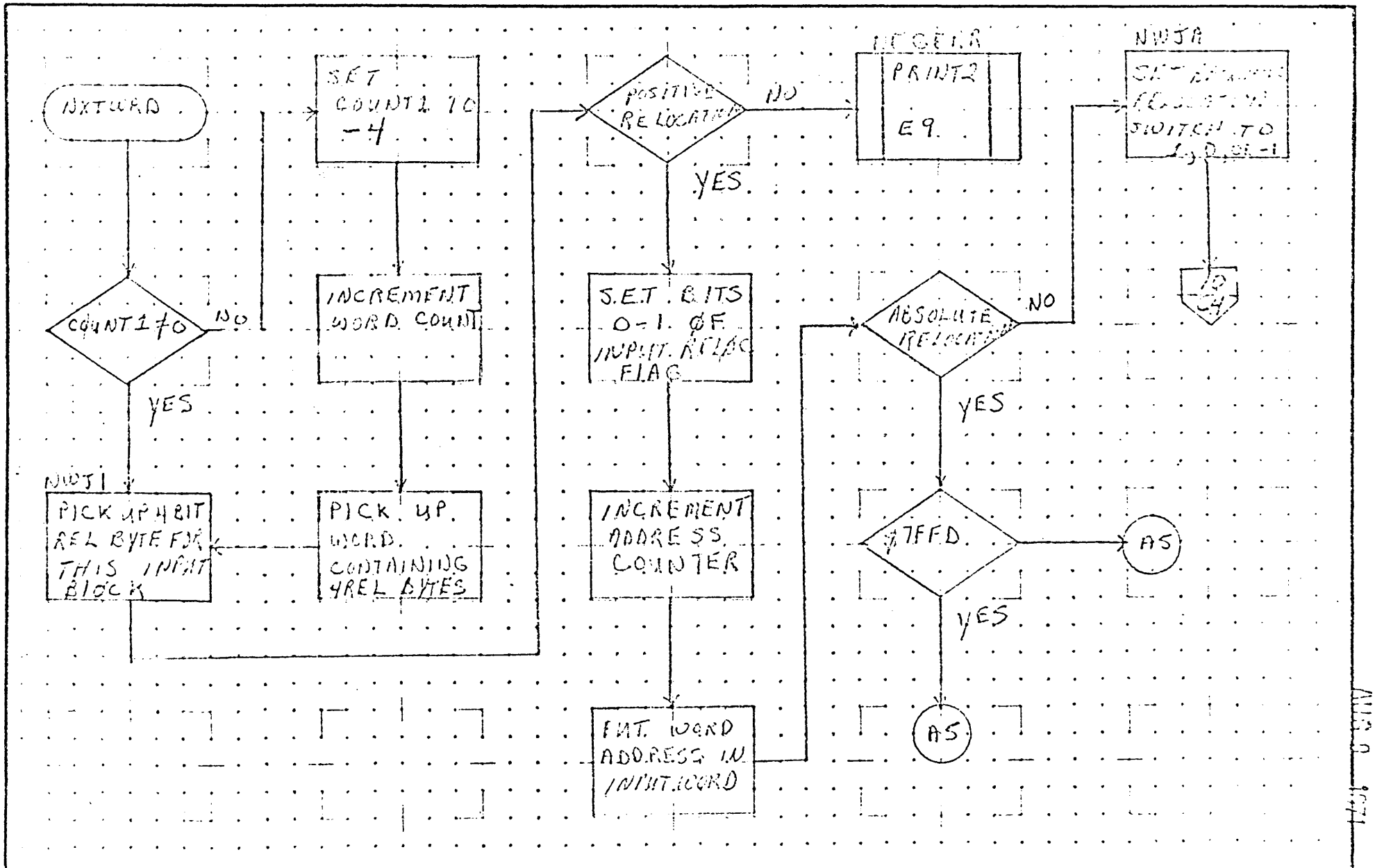
AUG 6 1971

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

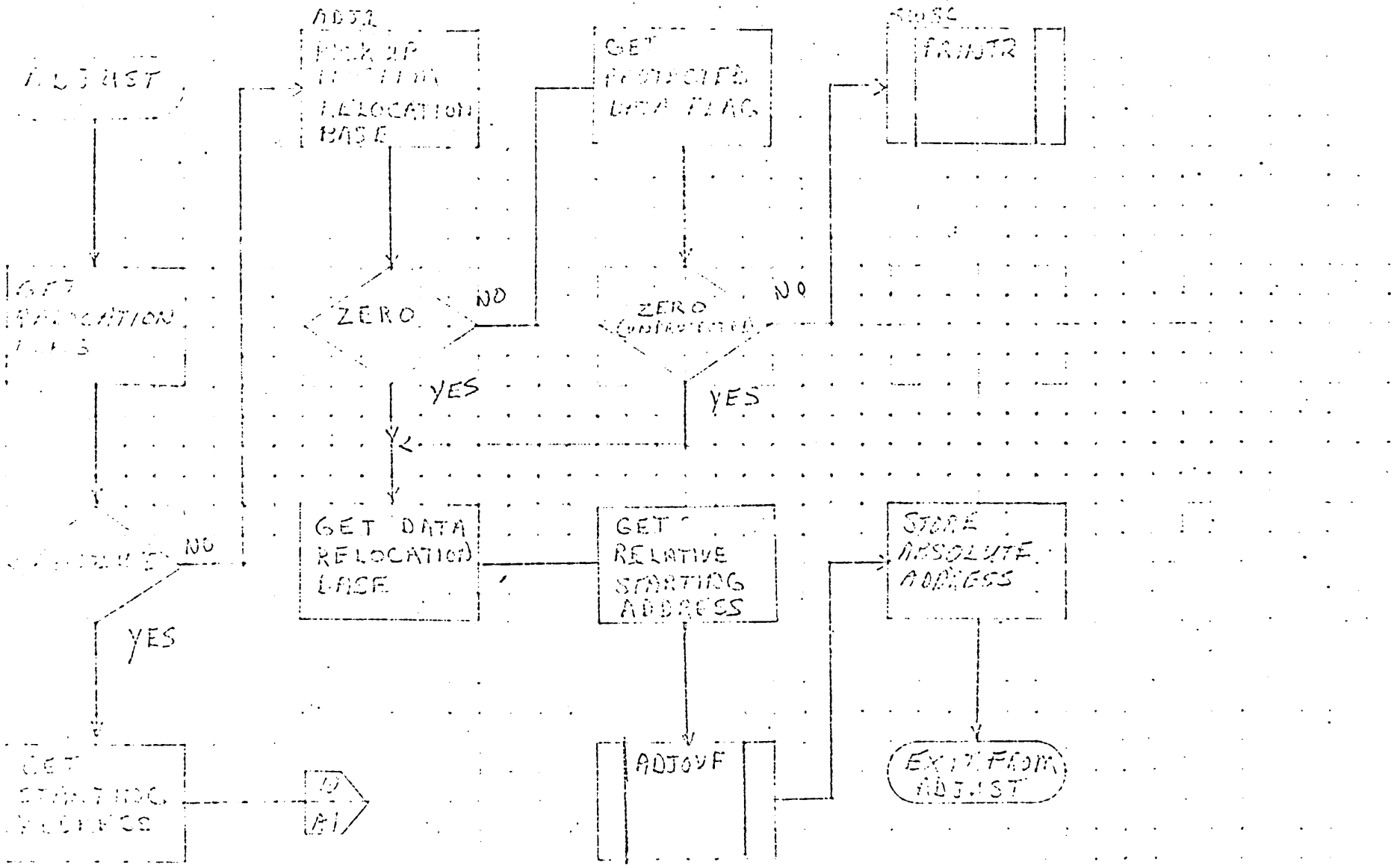


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	PAGE OF		PROJECT NAME		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY / DATE		TASK NAME		

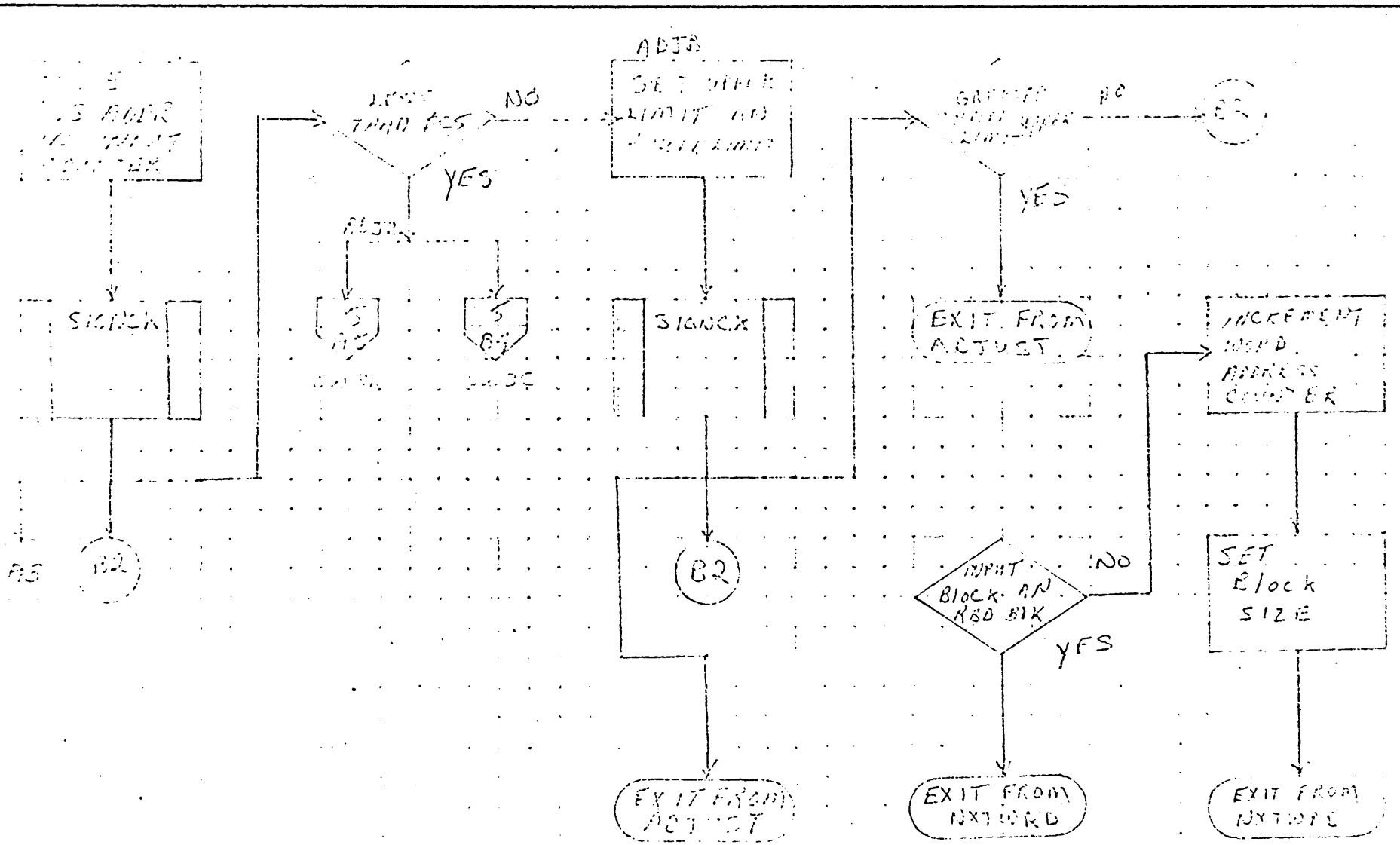
AUG 9 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IMS MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE RBDBZ1	PROJECT MGR.			
	PAGE 3 OF 11	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY CDC DATE 4-10-67	TASK NAME PART 1 LOADER			

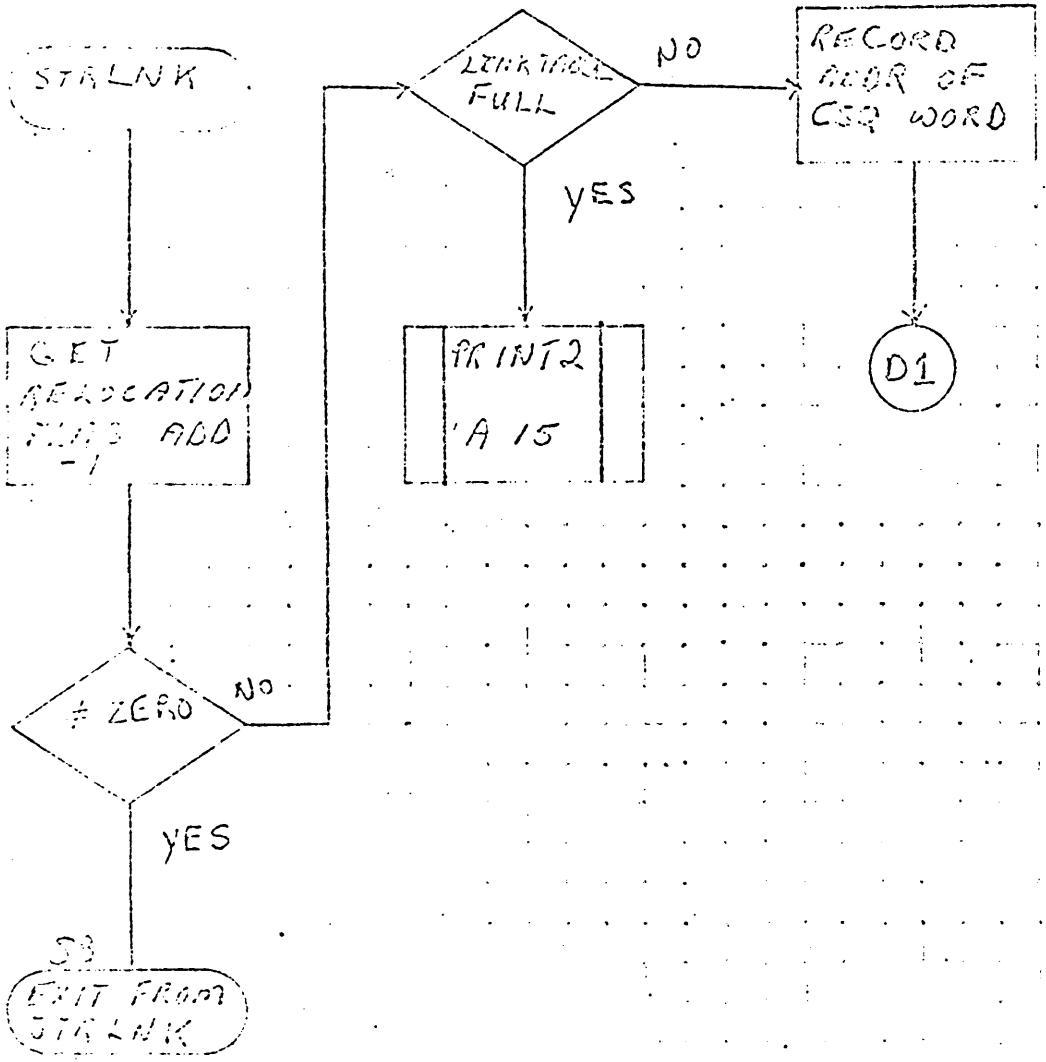


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	PAGE OF		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			



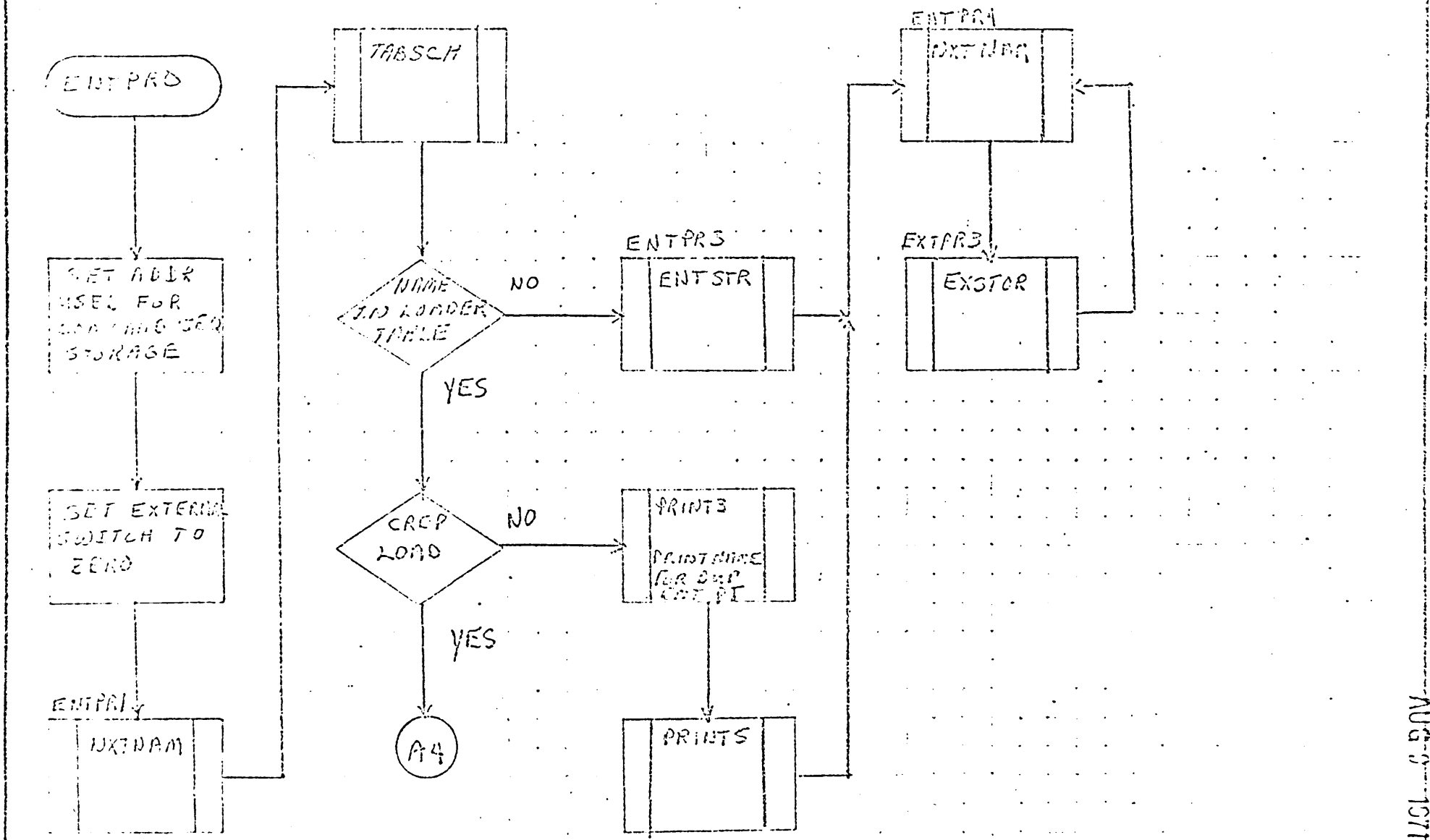
CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	TC	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☒		PAGE OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

PRINTED IN U.S.A.



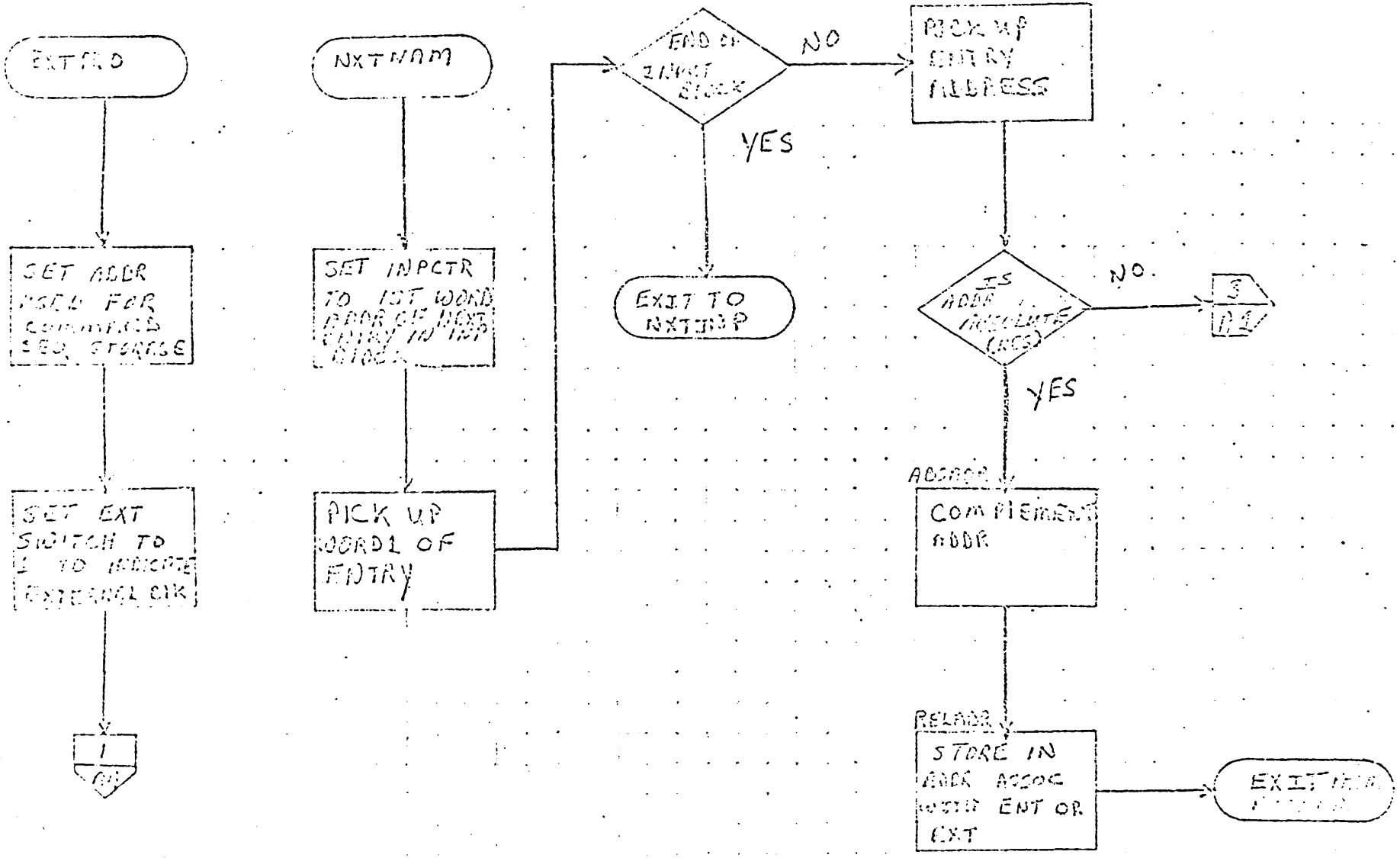
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE [illegible]	PROJECT MGR.			
	PAGE 11 OF 11	PROJECT NAME [illegible]			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY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

AUG 4—1971

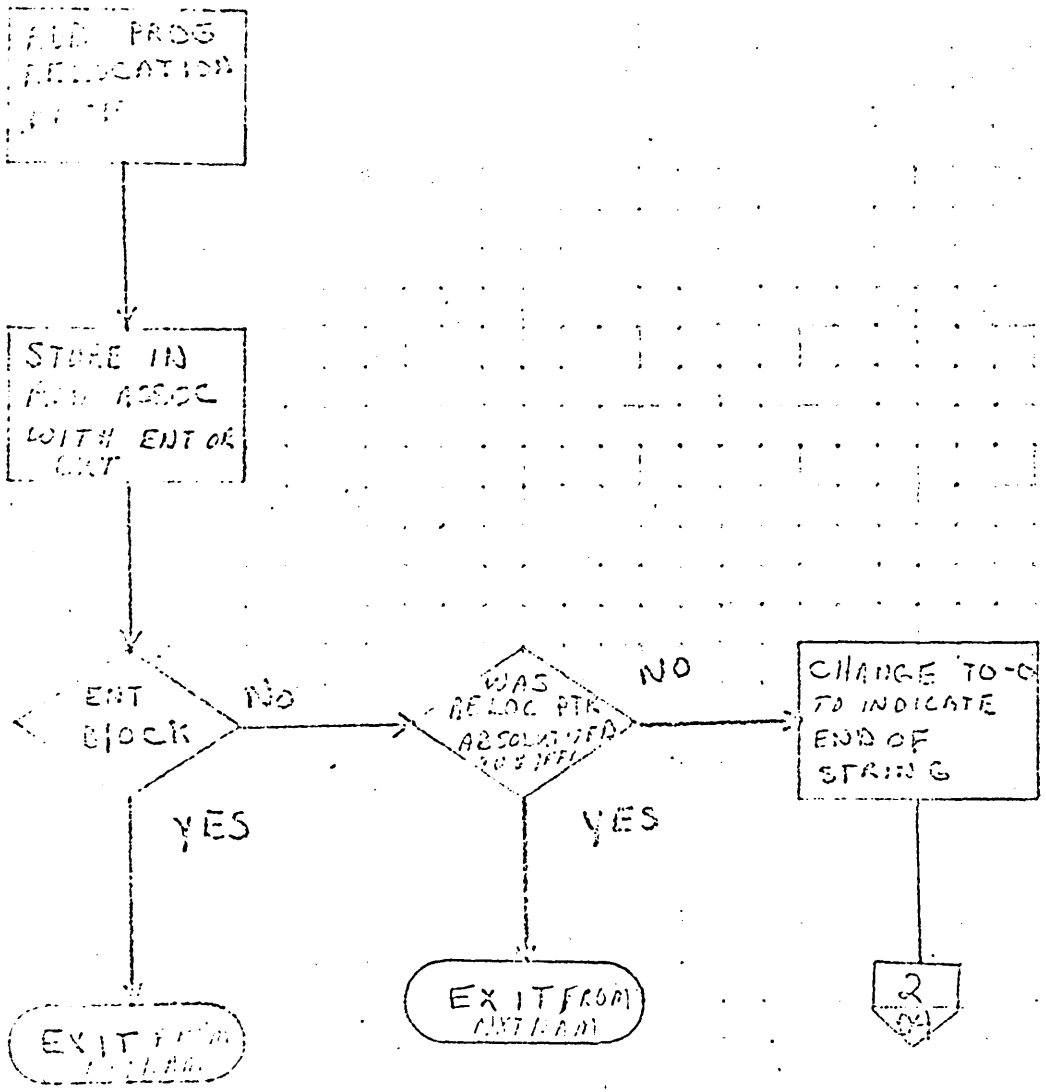


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE EXTTx1	PROJECT MGR.			
	PAGE 1 OF 3	PROJECT NAME [illegible]			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE [illegible]	TASK NAME [illegible]			

AUG 3 1971

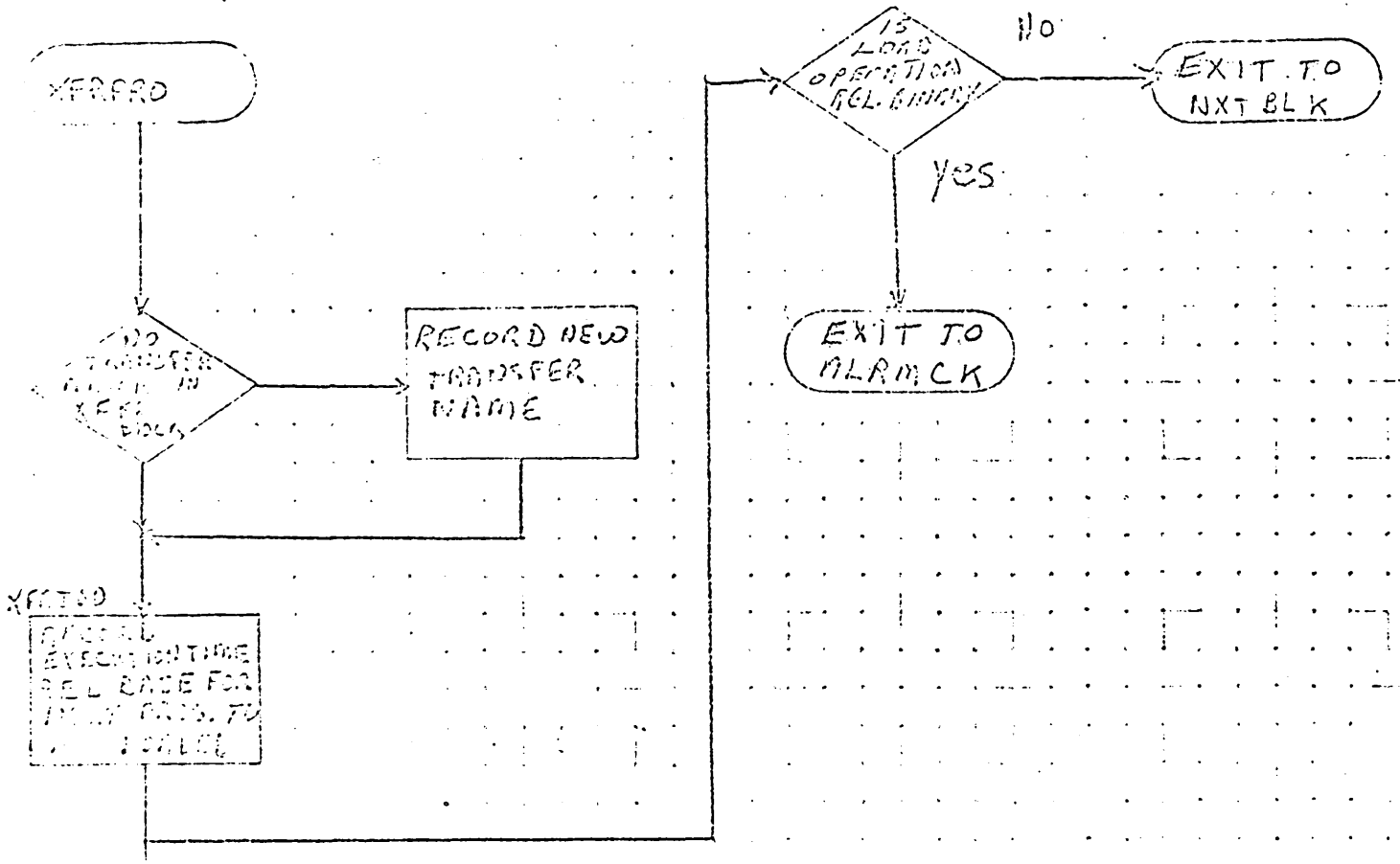


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS T/A... MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ENTEXT	PROJECT MGR.			
	PAGE 2 OF 4	PROJECT NAME ...1700			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY ...D G. DATE ...	TASK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



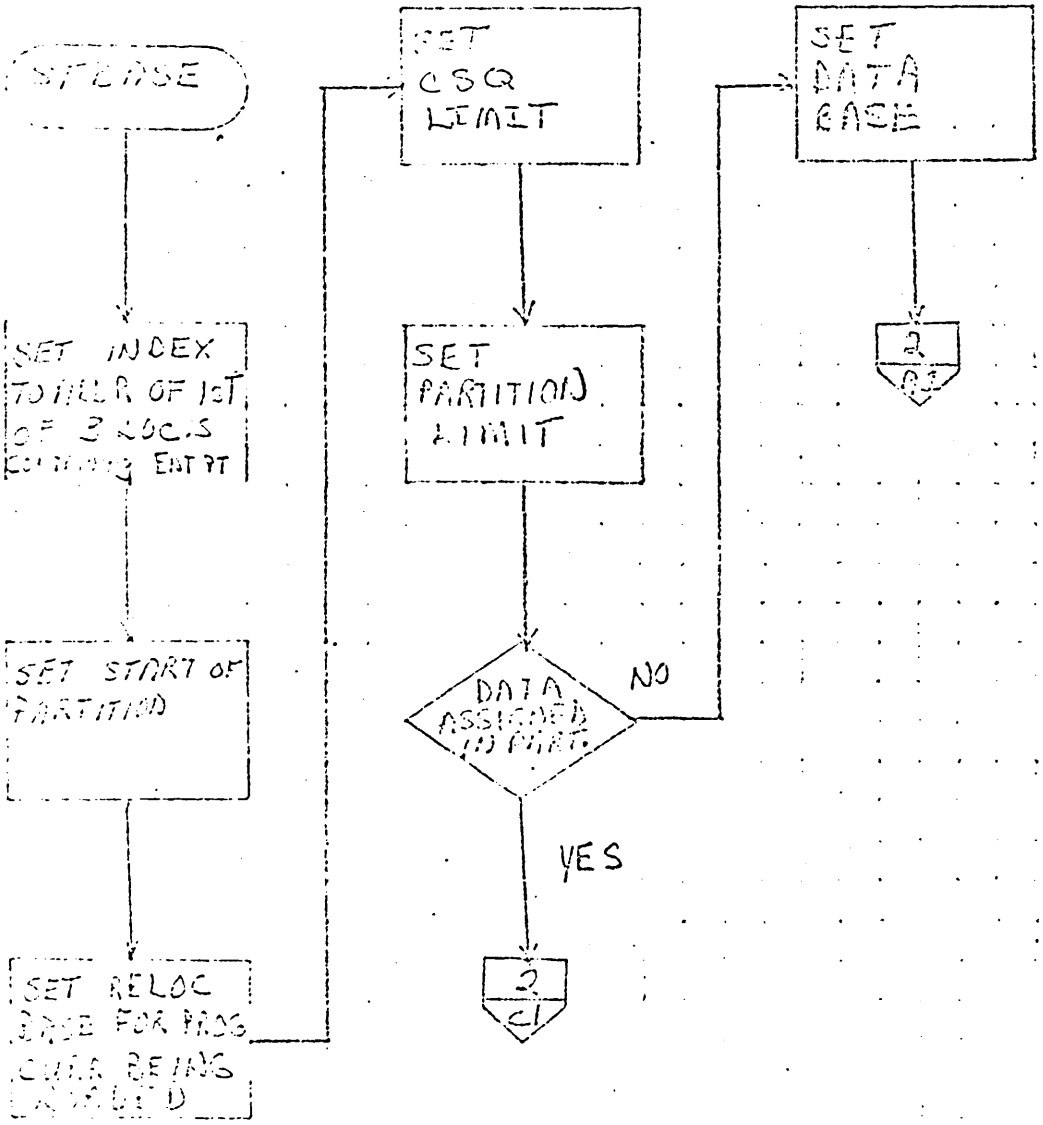
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	DEPT	APPROVED	DATE
	DOCUMENT TITLE		PROJECT MGR.			
	PAGE OF		PROJECT NAME			
	NUMBER / ISSUE DATE		TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AUG 6 1971



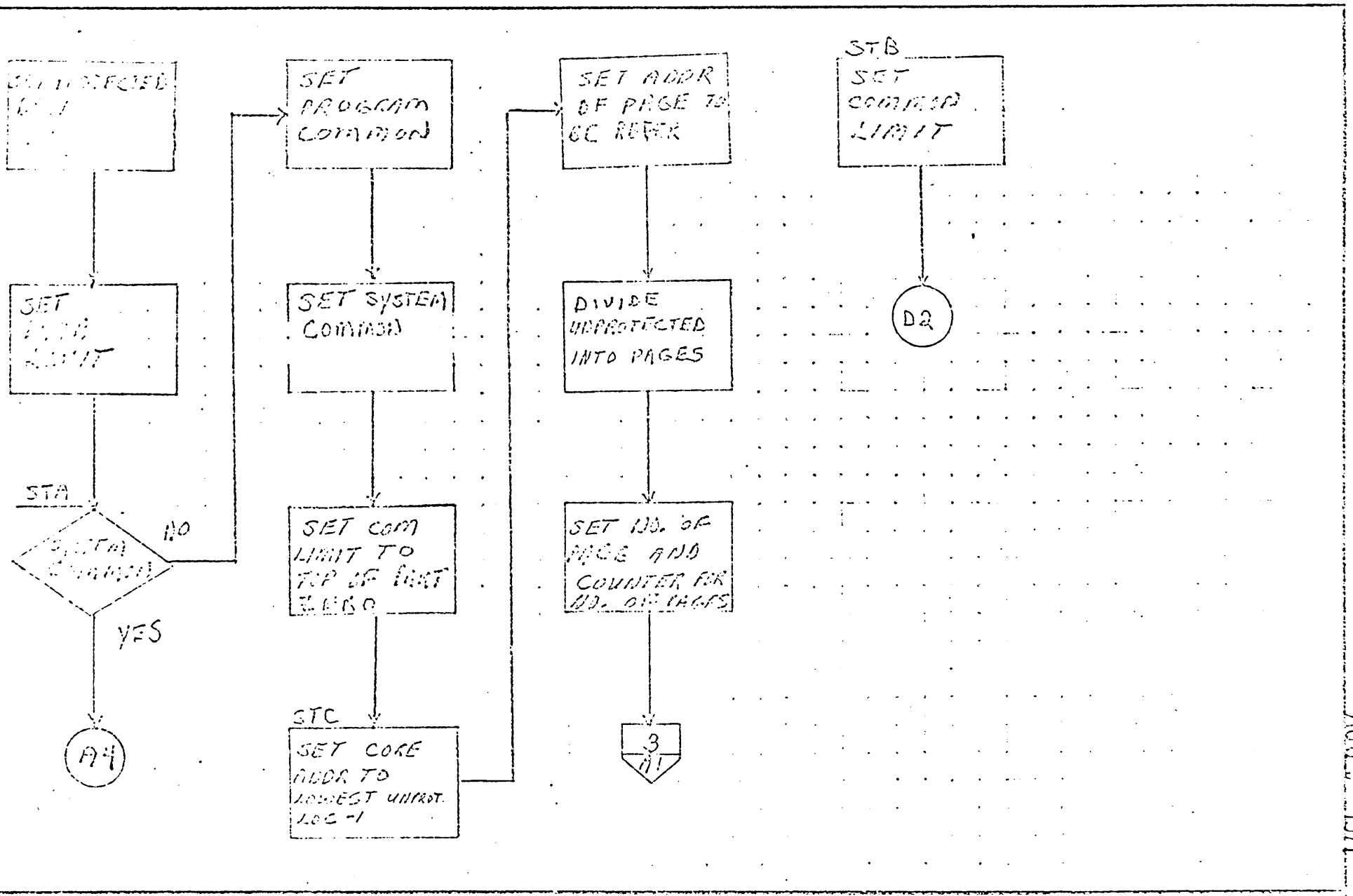
CONTROL DATA CORPORATION SOFTWARE DOCUMENT	DOCUMENT CLASS: IMS	MACH. TYPE: [illegible]	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT TITLE: XERPR1		PROJECT MGR.		
	PAGE 1 OF 1		PROJECT NAME: 1700 MSOS		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY: [illegible]	DATE: [illegible]	TASK NAME: [illegible]		

AUG 6 1971

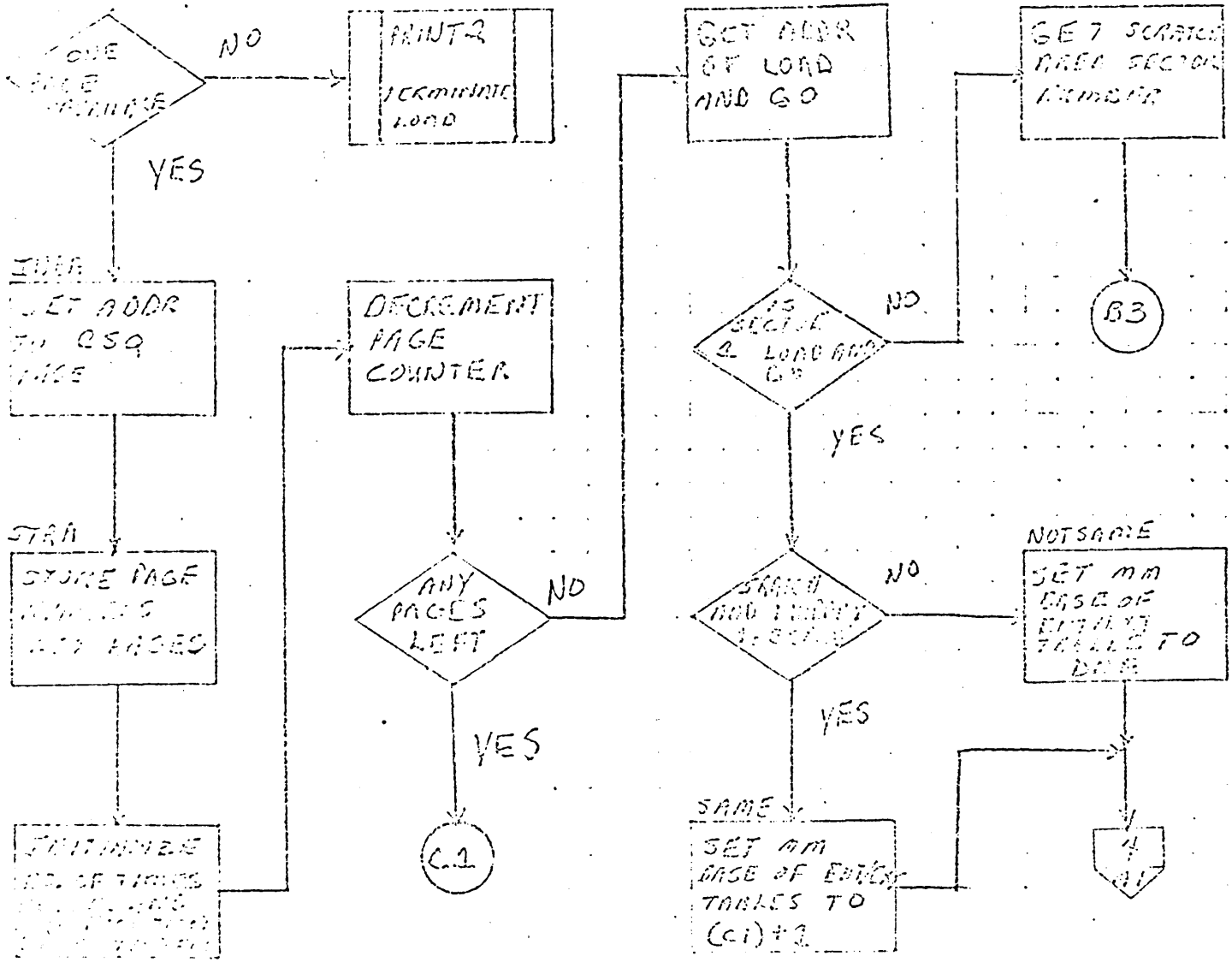


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE STGASE	PROJECT MGR.			
	PAGE 1 OF 11	PROJECT NAME [illegible]			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible]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

AUG 3 1971

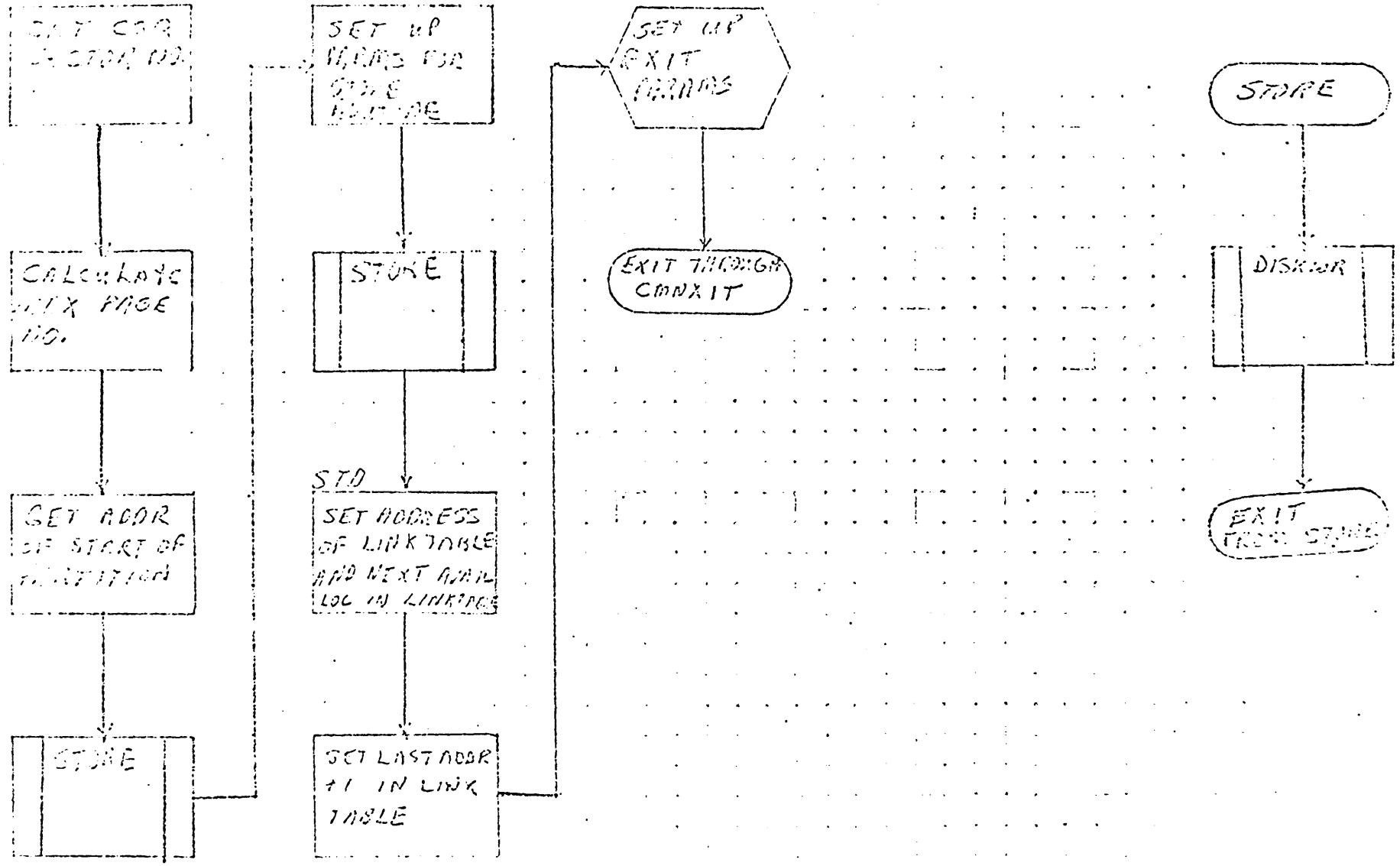


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE FLOWCHART DECISION TABLE OTHER	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.		
	DOCUMENT TITLE ST PAGE		PROJECT MGR.		
	PAGE 3 OF 4		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY CDC	DATE	TASK NAME		



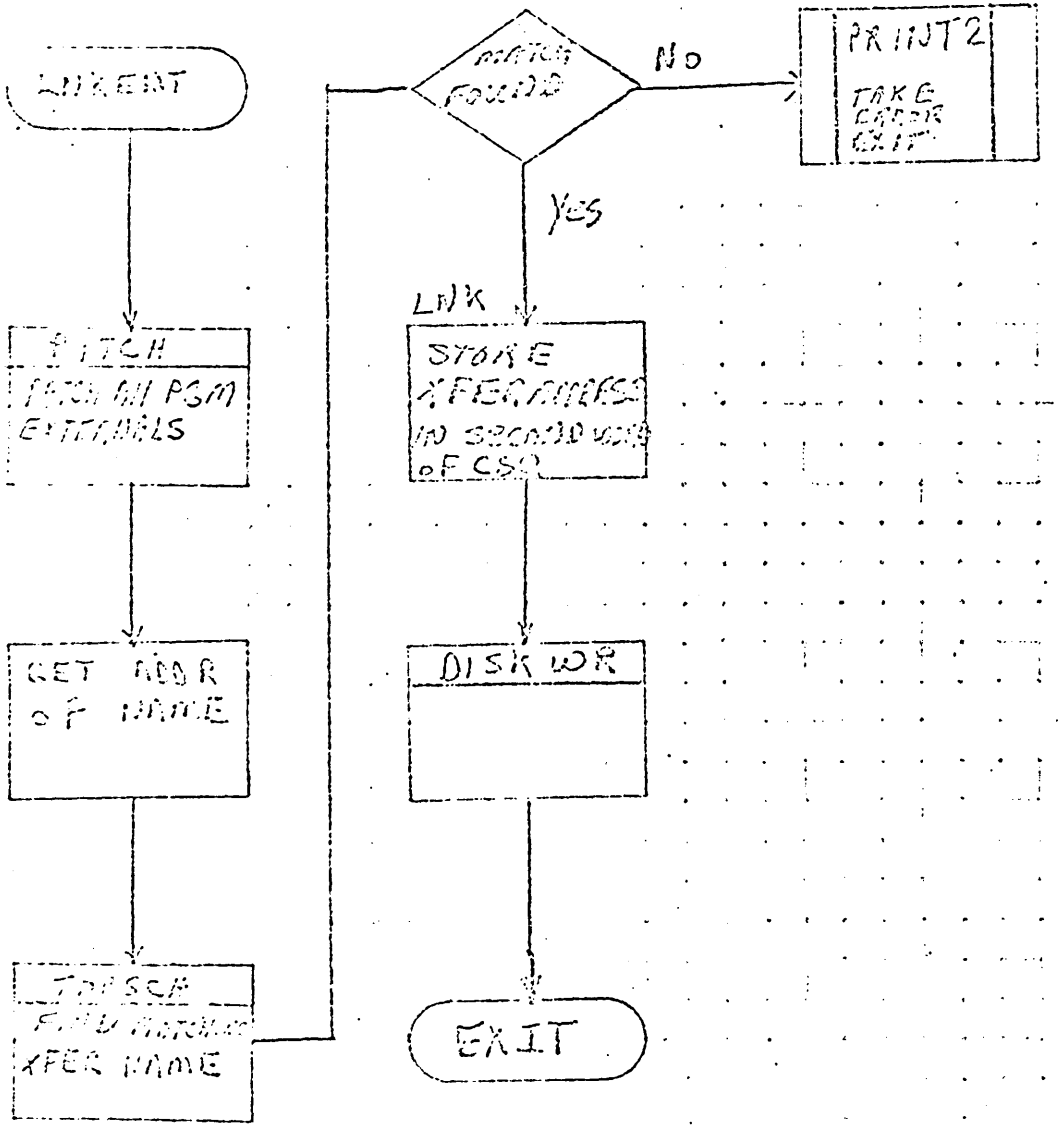
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	TSN	APPROVED / DATE
	DOCUMENT TITLE STRASR		PROJECT MGR.		
	PAGE 1 OF 1		PROJECT NAME		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY / DATE		TASK NAME		

AUG 9 1971



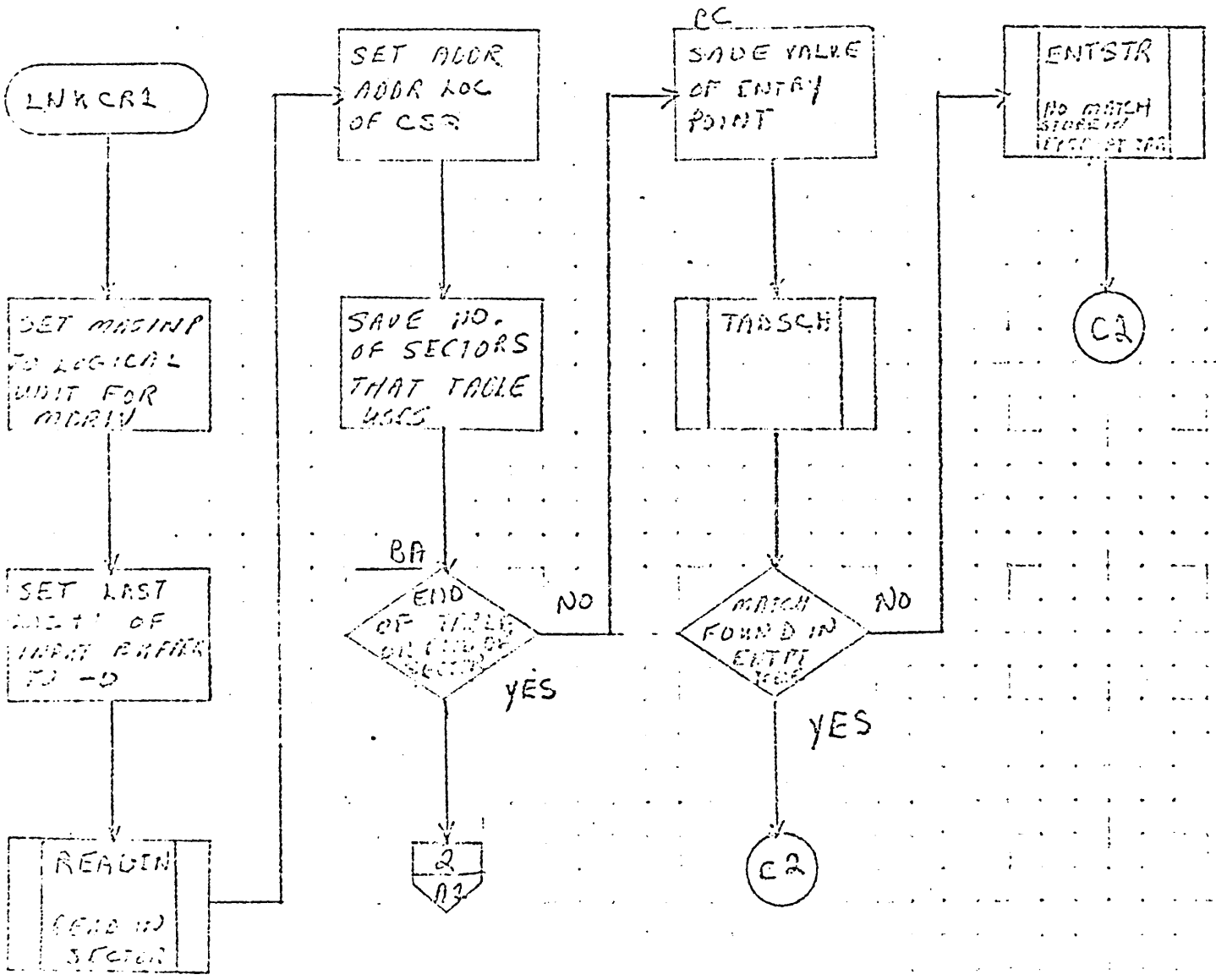
AUG 9 1971

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

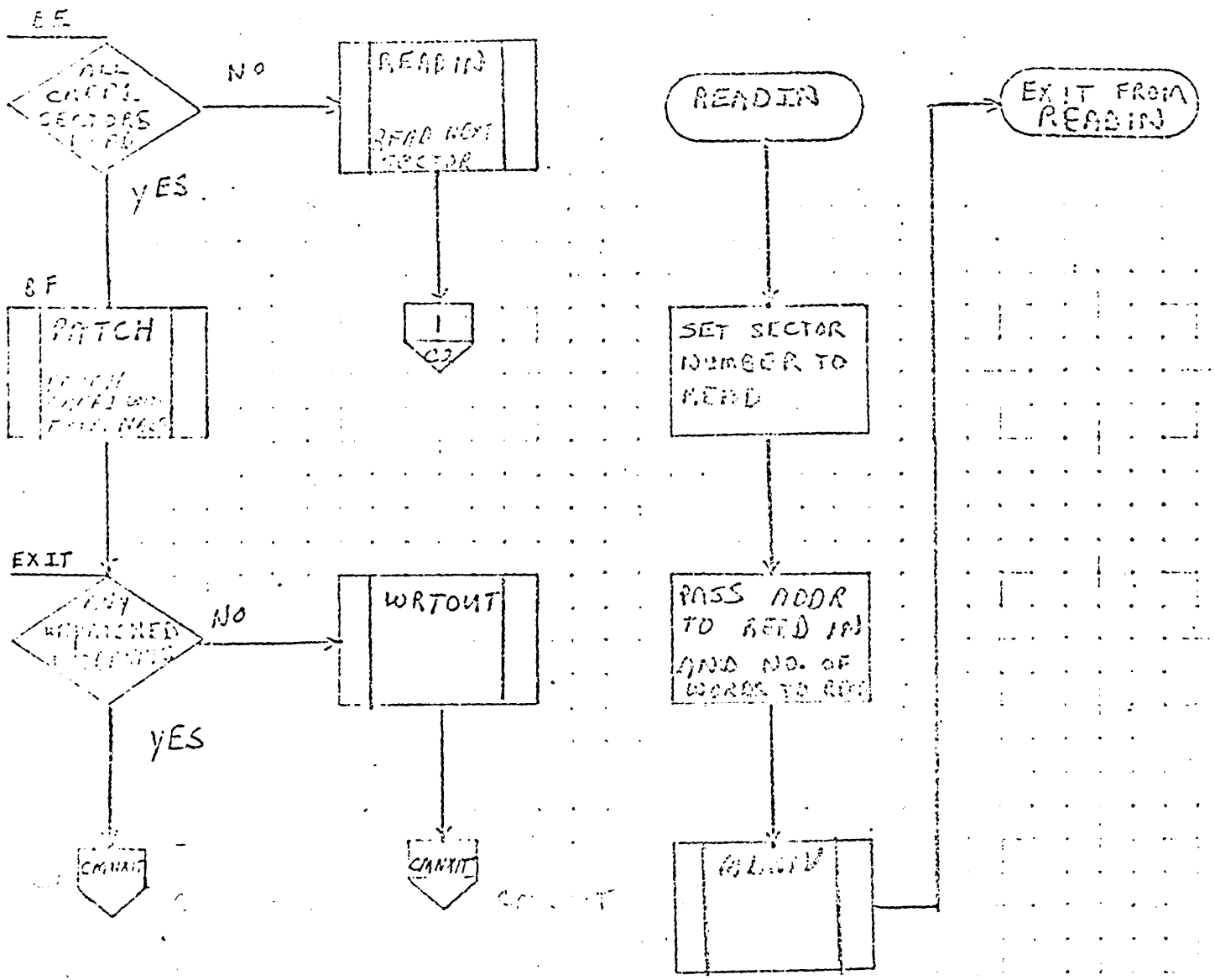


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS A MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LINKFIX	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AUG 5 1971

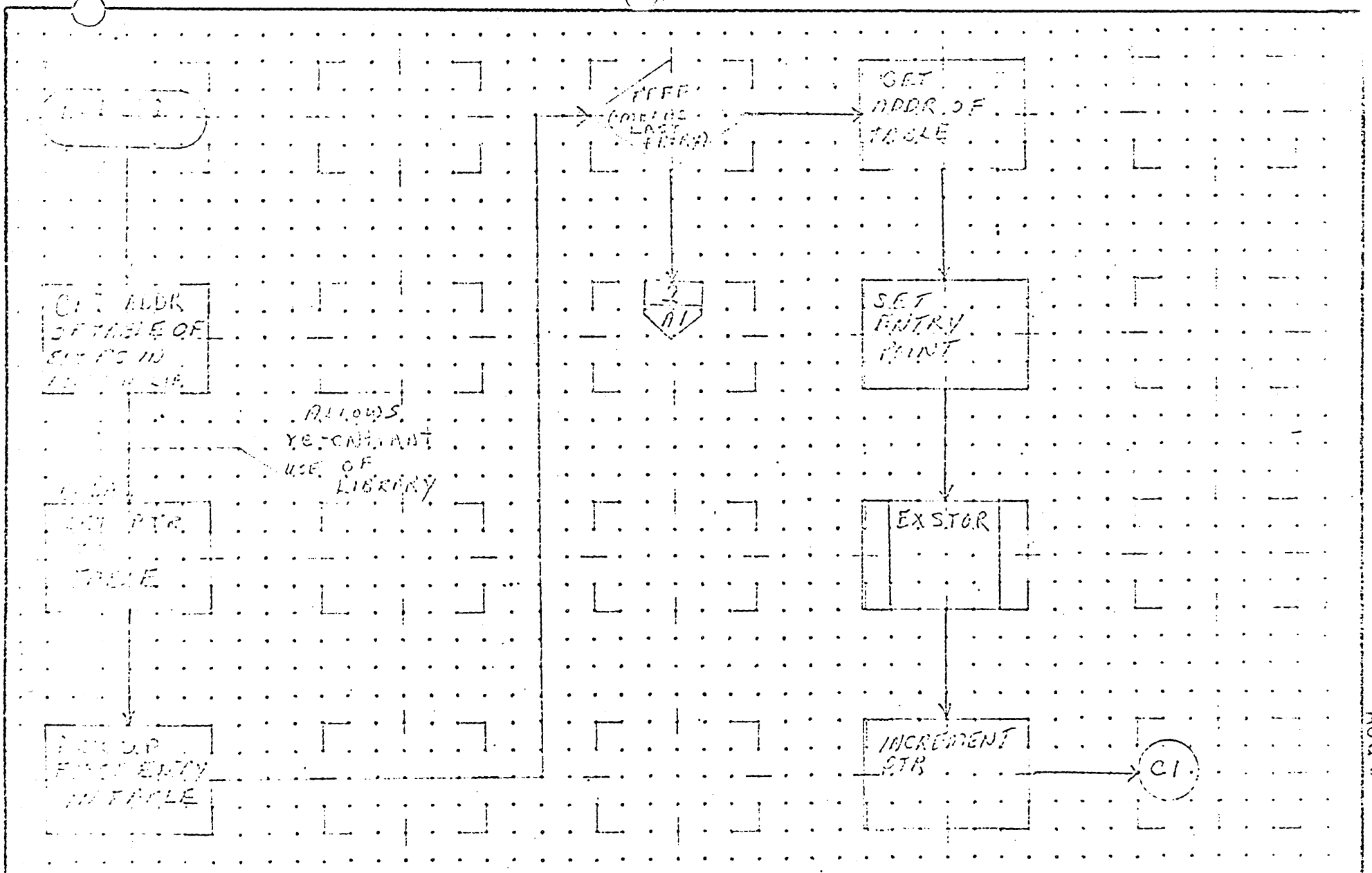


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LNKCR1	PROJECT MGR.			
	PAGE 1 OF [illegible]	PROJECT NAME [illegible]			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE [illegible]	TASK NAME [illegible]			



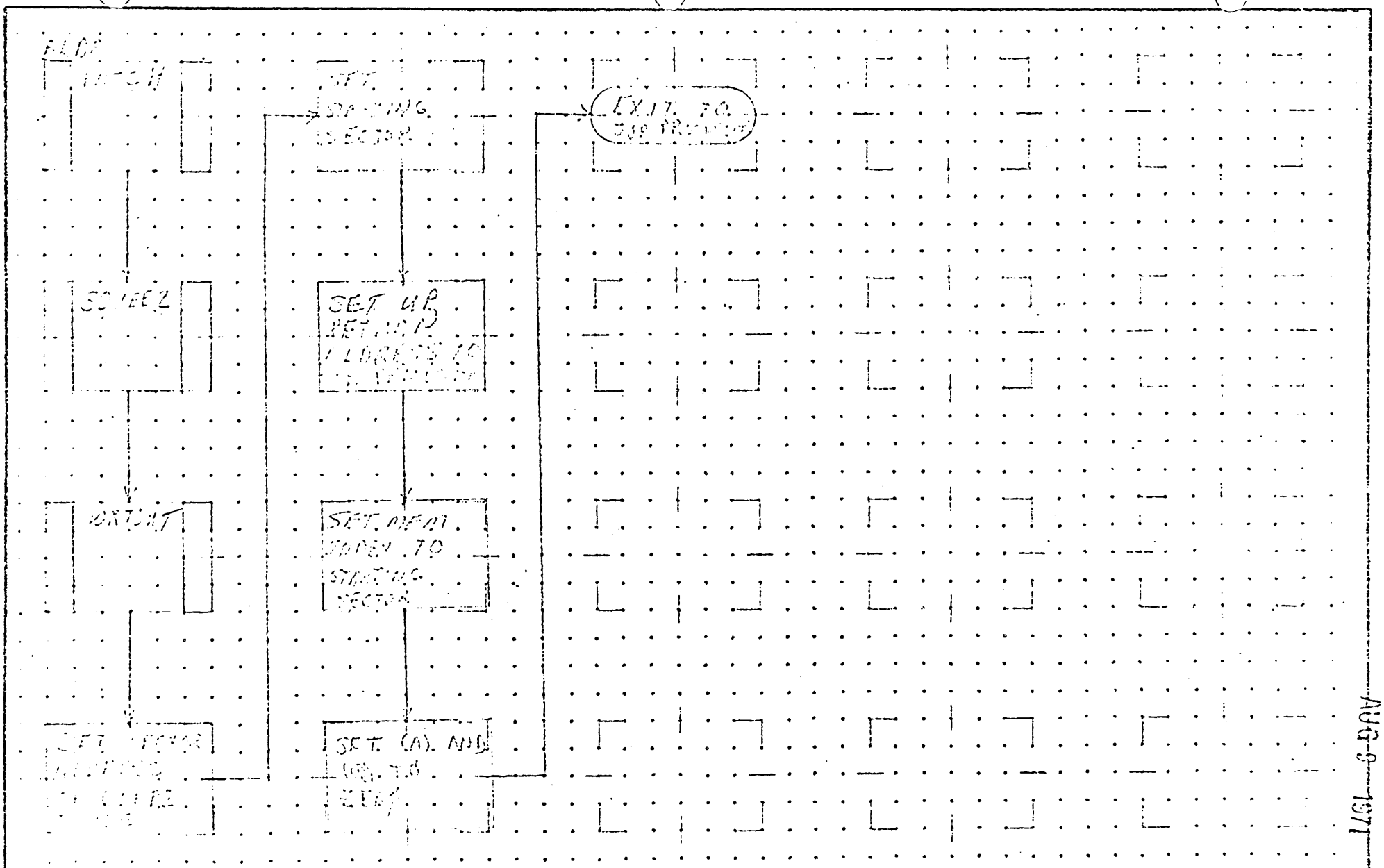
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE LNKCPd.	PROJECT MGR.			
	PAGE 2 OF [illegible]	PROJECT NAME [illegible]			
	NUMBER ISSUE DATE	TASK NO.			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

AUG 9 1971



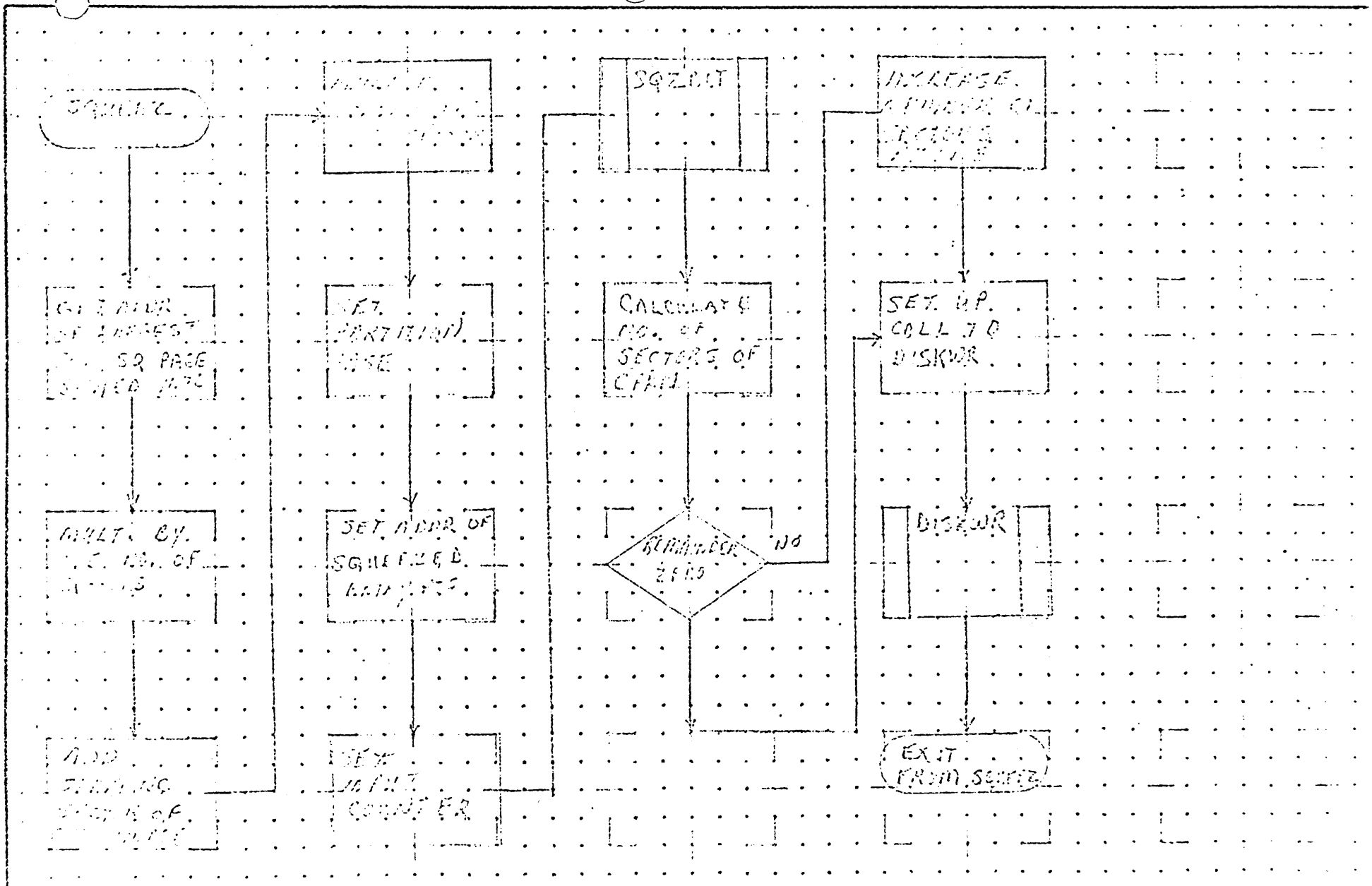
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS J. MS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE [illegible]		PROJECT MGR.		
	PAGE 1 OF 6		PROJECT NAME 1700 MSOS		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY [illegible]	DATE [illegible]	TASK NAME [illegible]		

AUG 68 1701 [illegible]



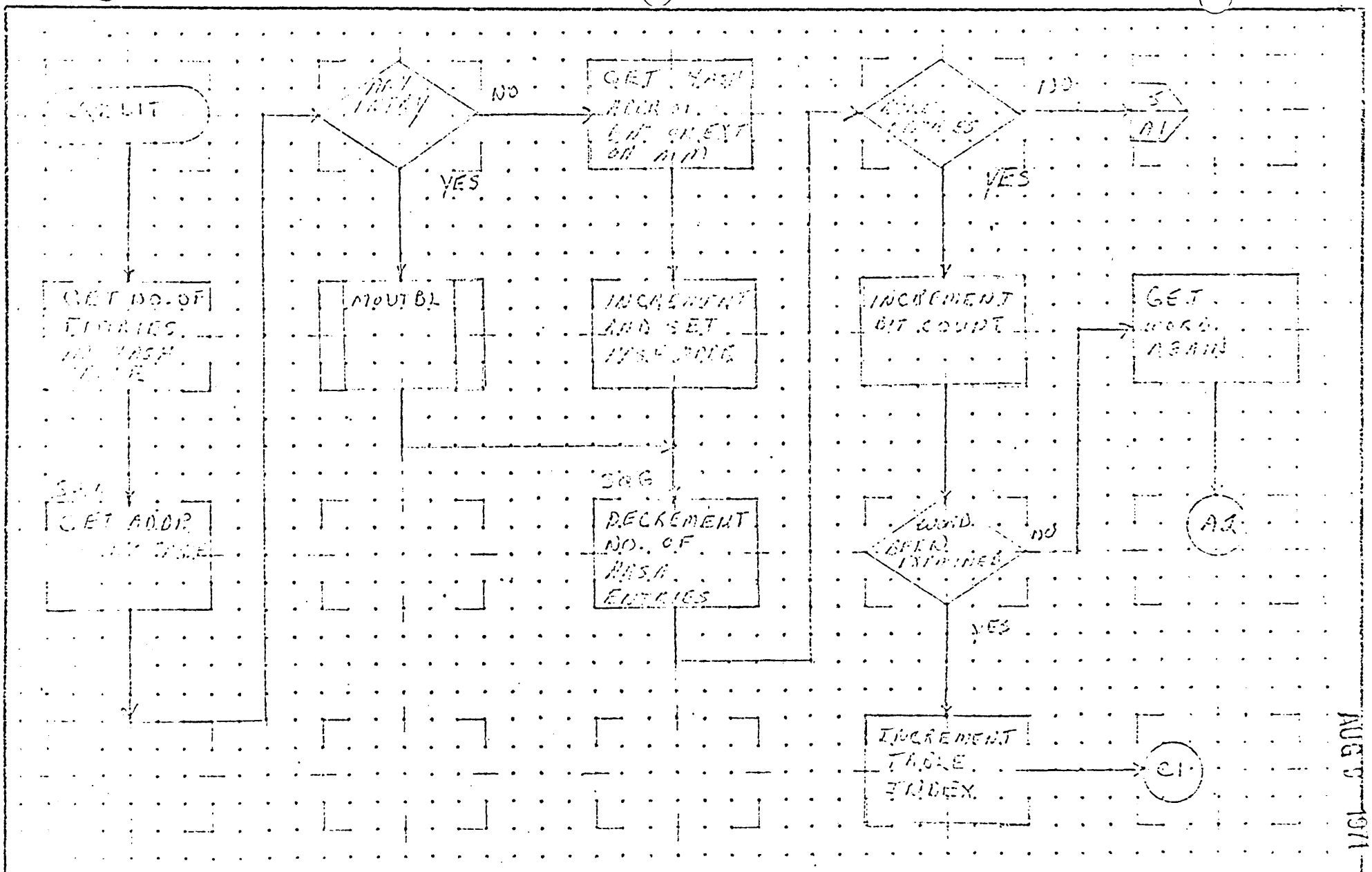
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TMS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE ELDCH1	PROJECT MGR.			
	PAGE 5 OF 6	PROJECT NAME 1700 SYSTEM			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE 8/6/71	TASK NAME			

AUG 9 1971

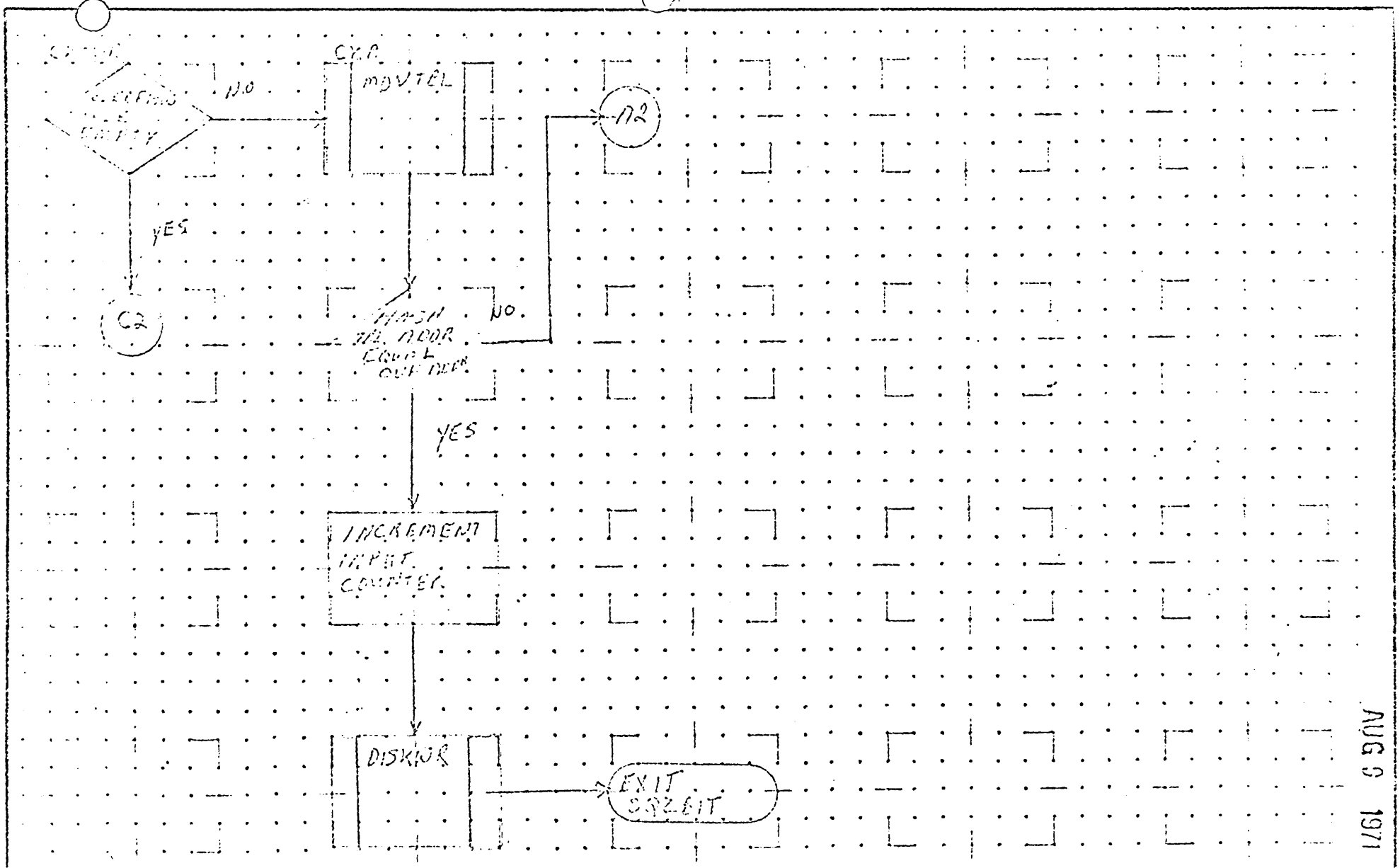


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐		PAGE 3 OF	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AUG 4 1971

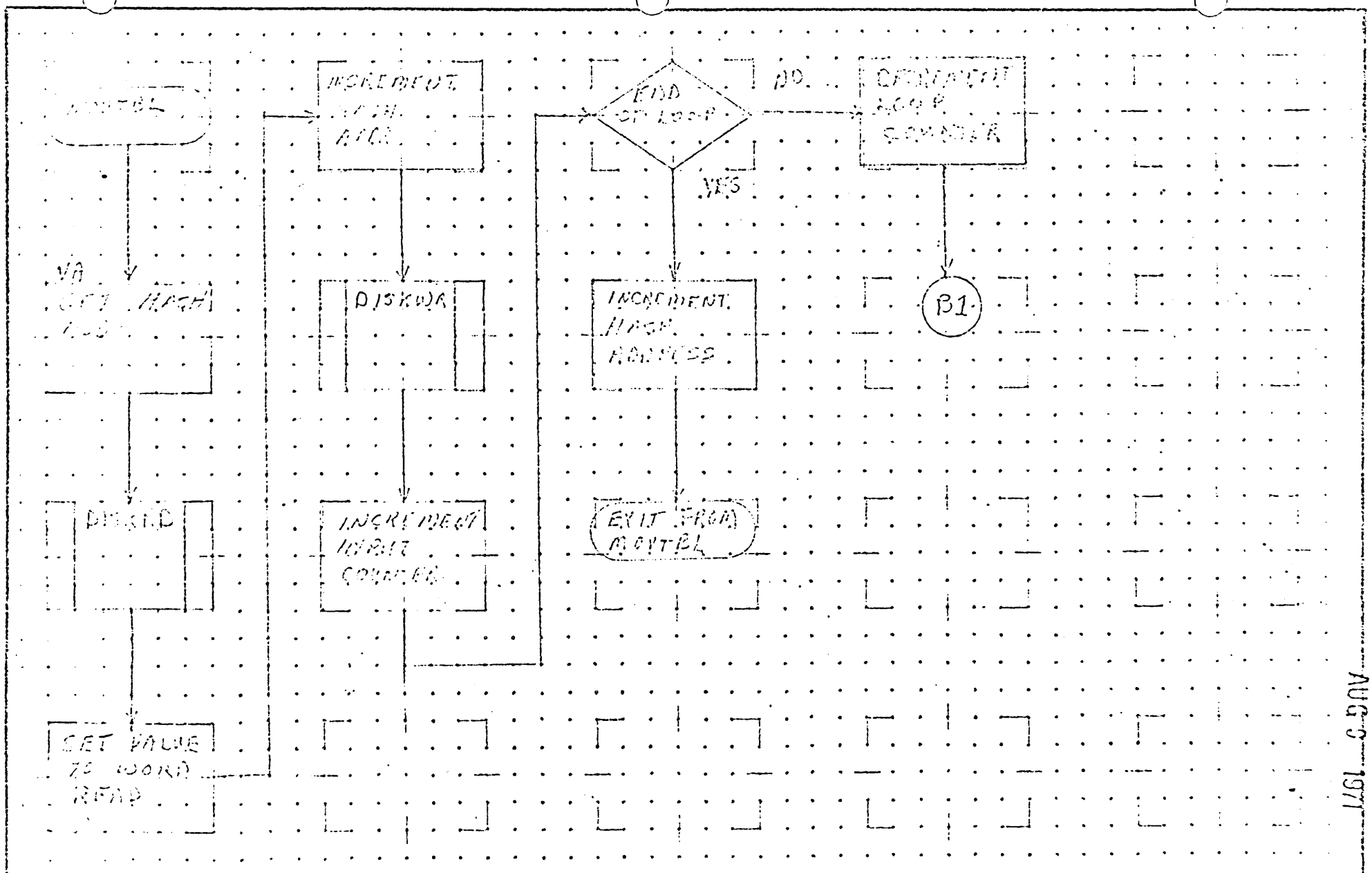


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐					
DOCUMENT CLASS JPS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE	
DOCUMENT TITLE HASH1	PROJECT MGR.				
PAGE 4 OF 7	PROJECT NAME 1700 MONITOR				
NUMBER ISSUE DATE	TASK NO.				
DRAWN BY DATE	TASK NAME [illegible]				



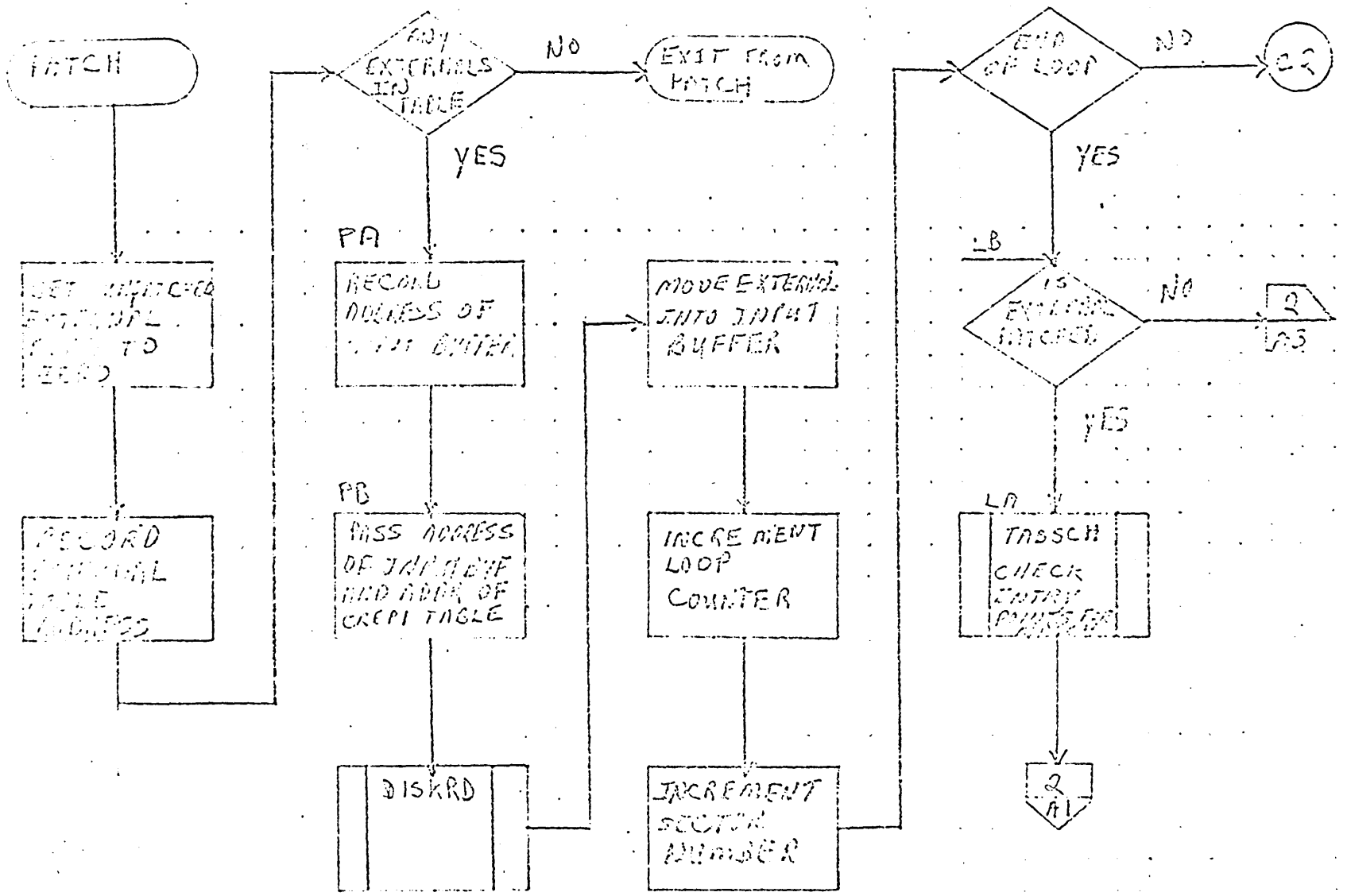
AUG 5 1971

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



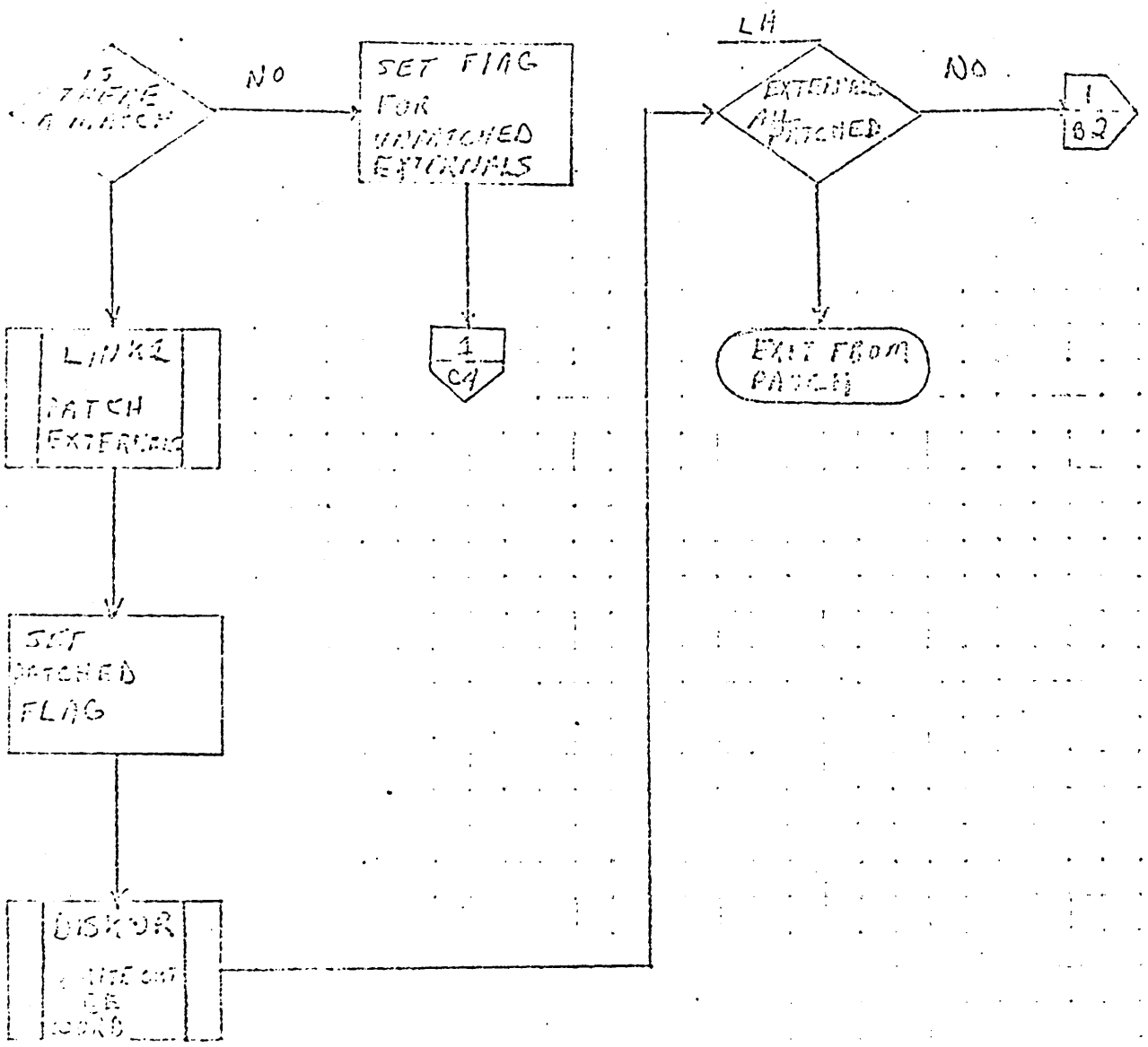
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE DISKRD		PROJECT MGR.			
	PAGE 6 OF 6		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

AUG 3 1971



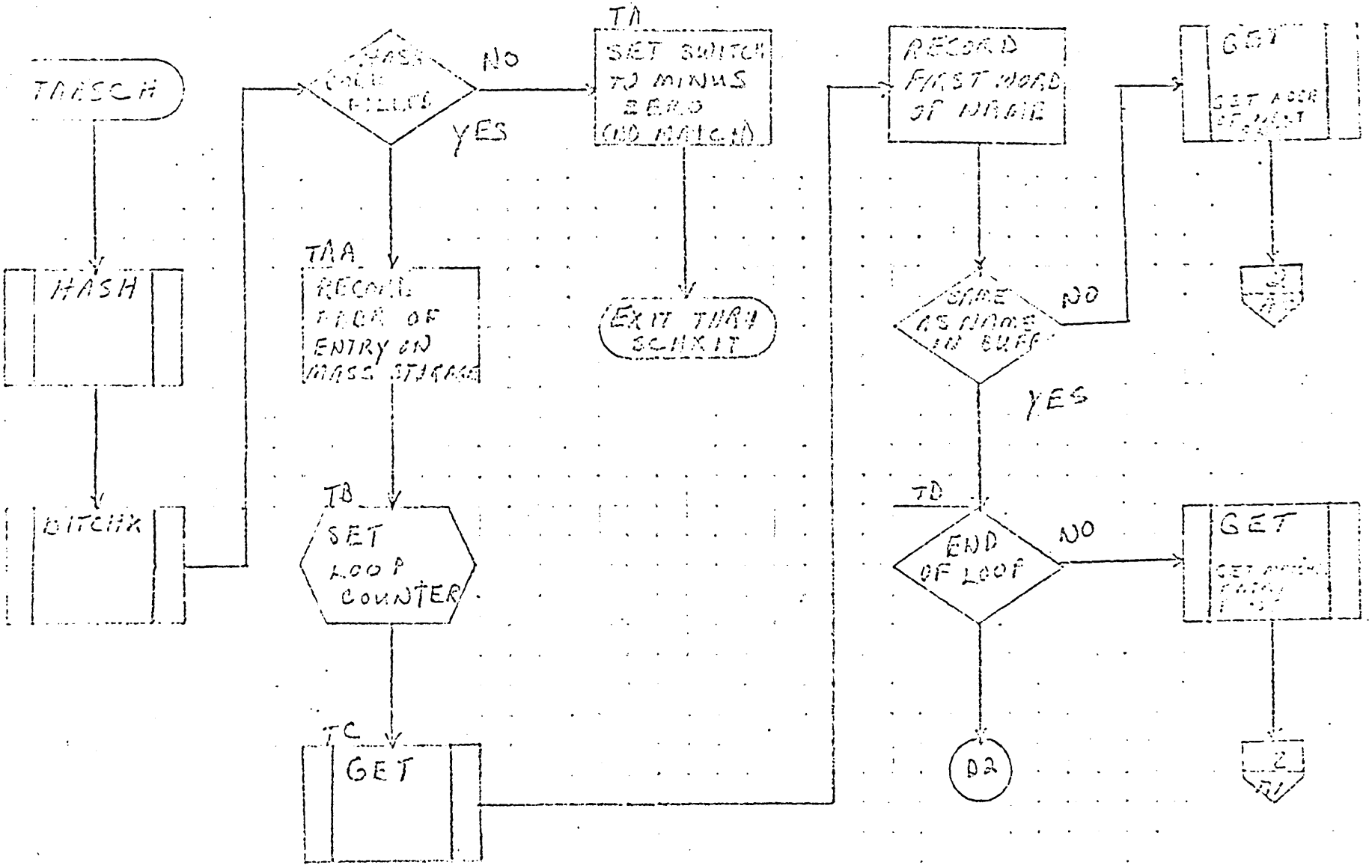
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PATCH		PROJECT MGR.		
	PAGE 1 OF 2		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AUG 9 1971

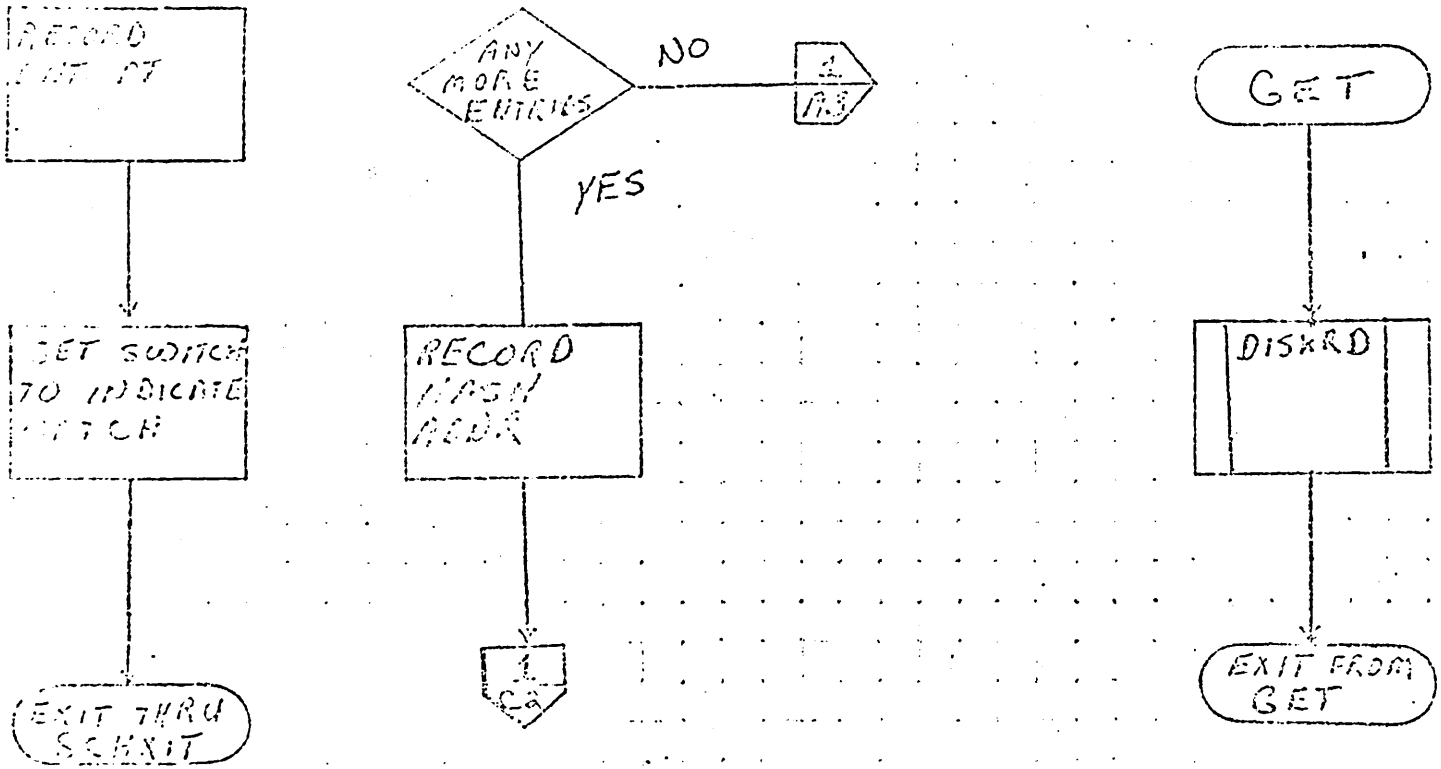


AUG 9 1971

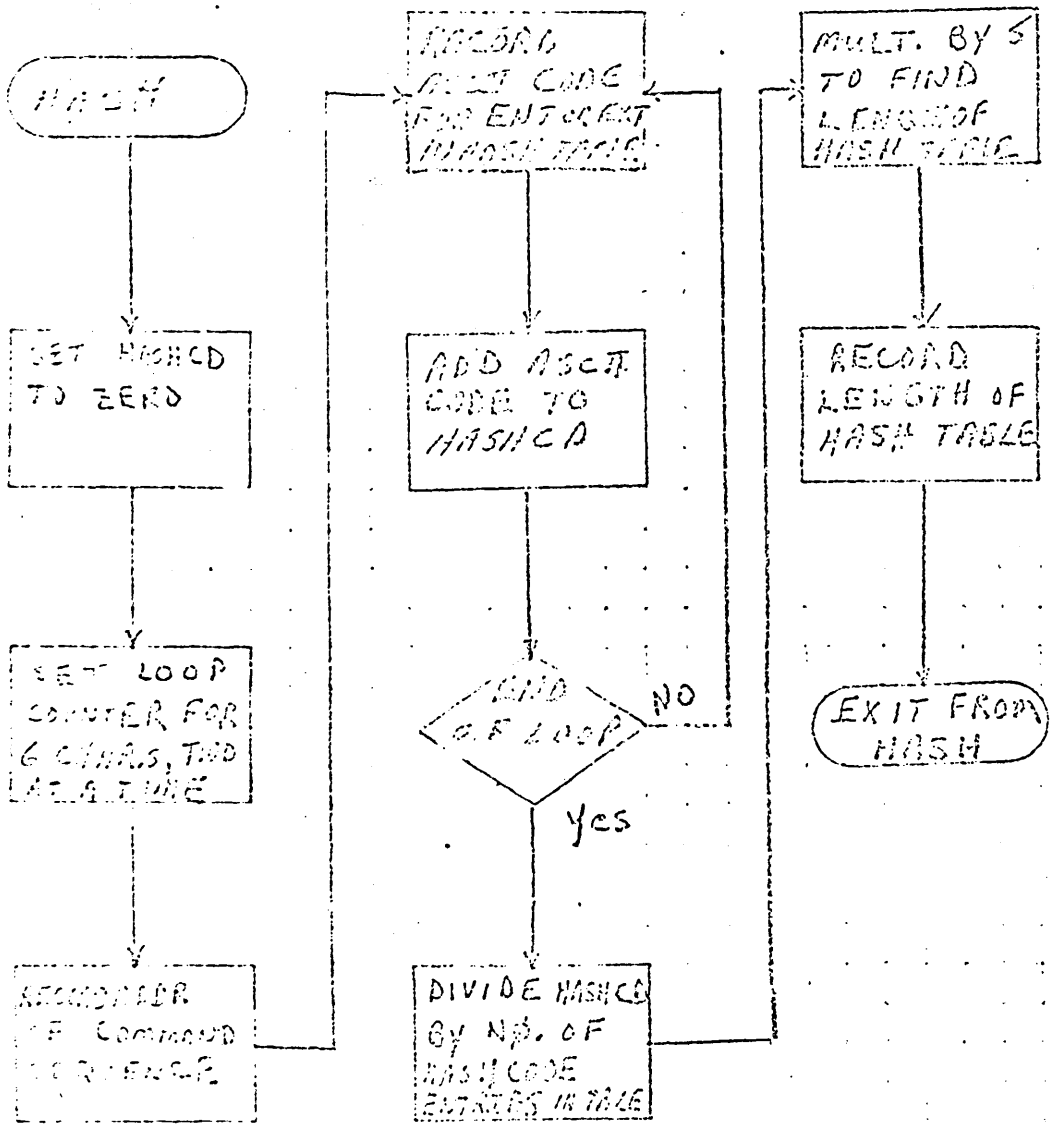
CONTROL DATA CORPORATION SOFTWARE DOCUMENT						
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 7	MACH. TYPE	PROJECT NO.	TECH	APPROVED	DATE
	DOCUMENT TITLE PATCH		PROJECT MGR.			
	PAGE 2 OF		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME PATCH LOADER			



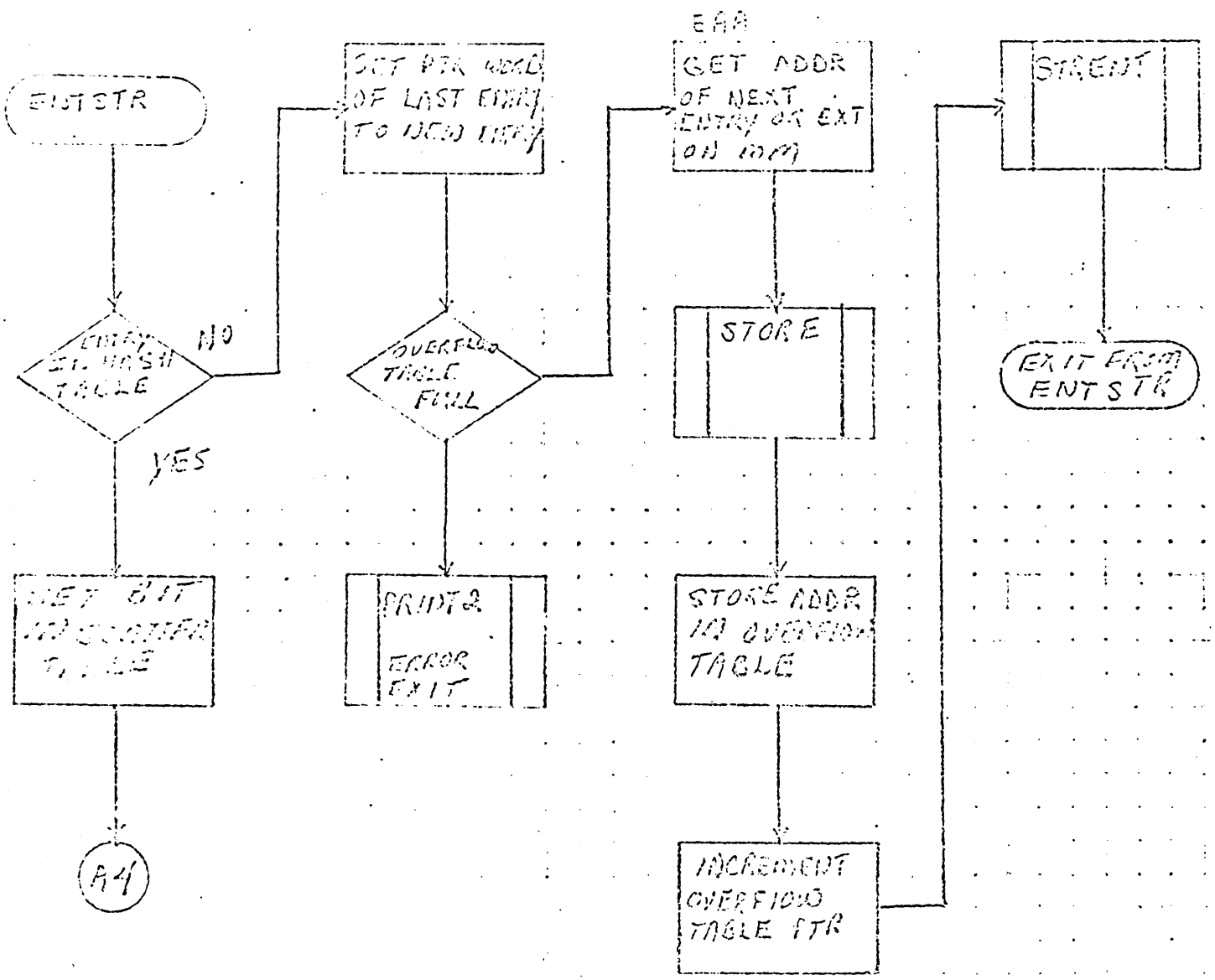
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TBSCH1		PROJECT MGR.		
	PAGE 1 OF		PROJECT NAME		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY / DATE		TASK NAME		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE TBSCH2		PROJECT MGR.			
	PAGE 2 OF 2		PROJECT NAME			
	NUMBER	ISSUE DATE	TASK NO.			
	DRAWN BY	DATE	TASK NAME			

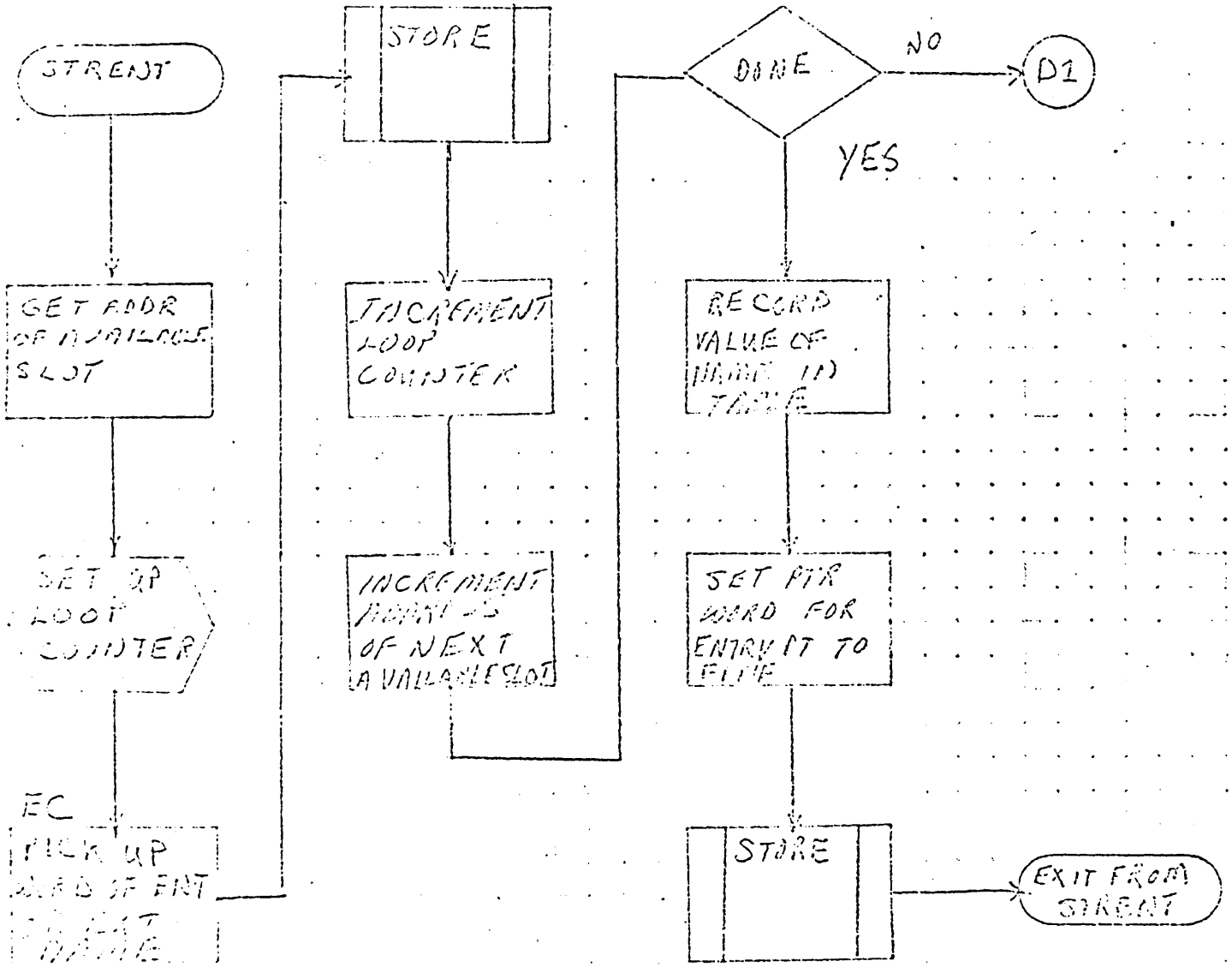


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED
	DOCUMENT TITLE HASH		PROJECT MGR.		
	PAGE OF		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

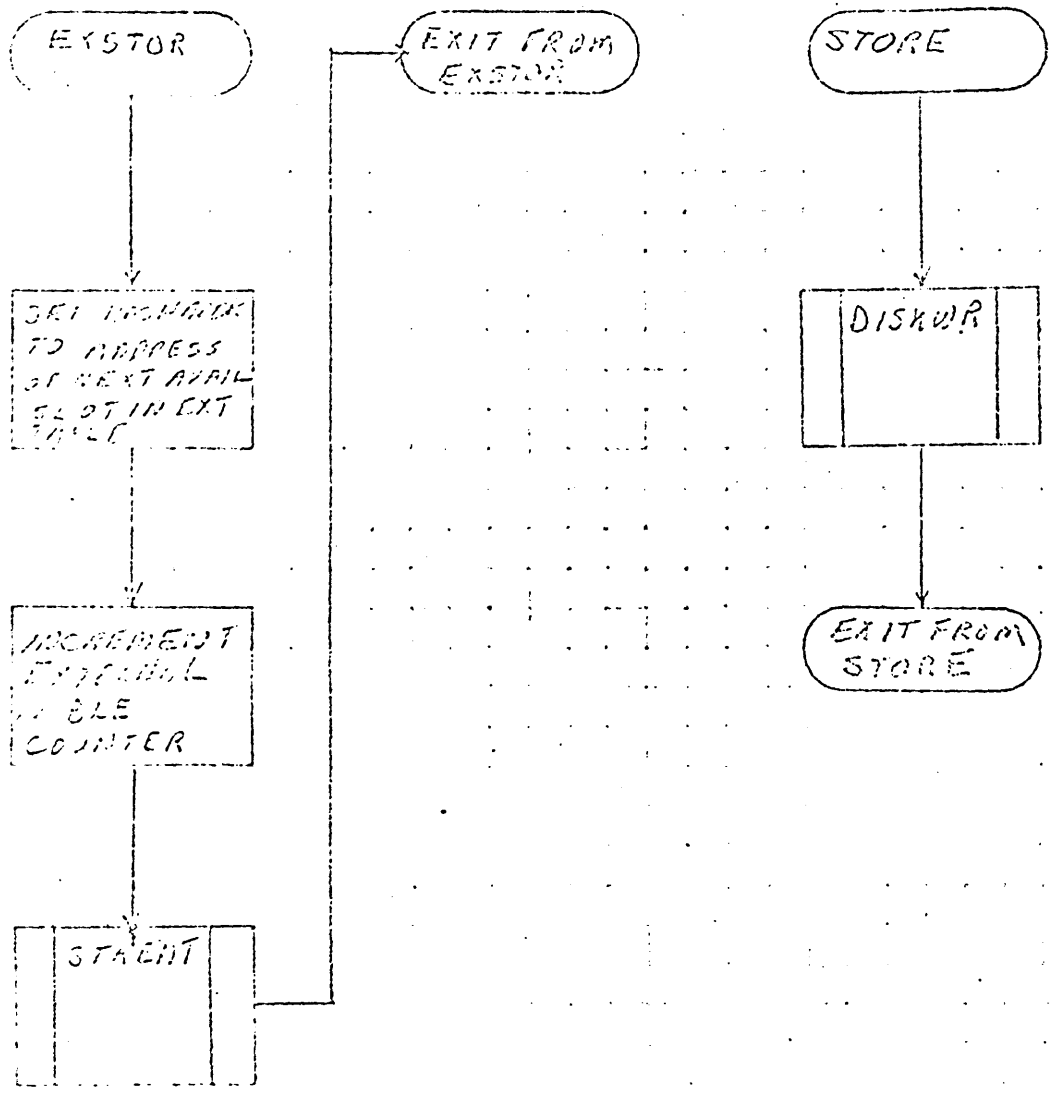


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS / MACH. TYPE	PROJECT NO.			
	DOCUMENT TITLE TBSTR1	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER / ISSUE DATE	TASK NO.			
	DRAWN BY / DATE 1/15/71	TASK NAME			

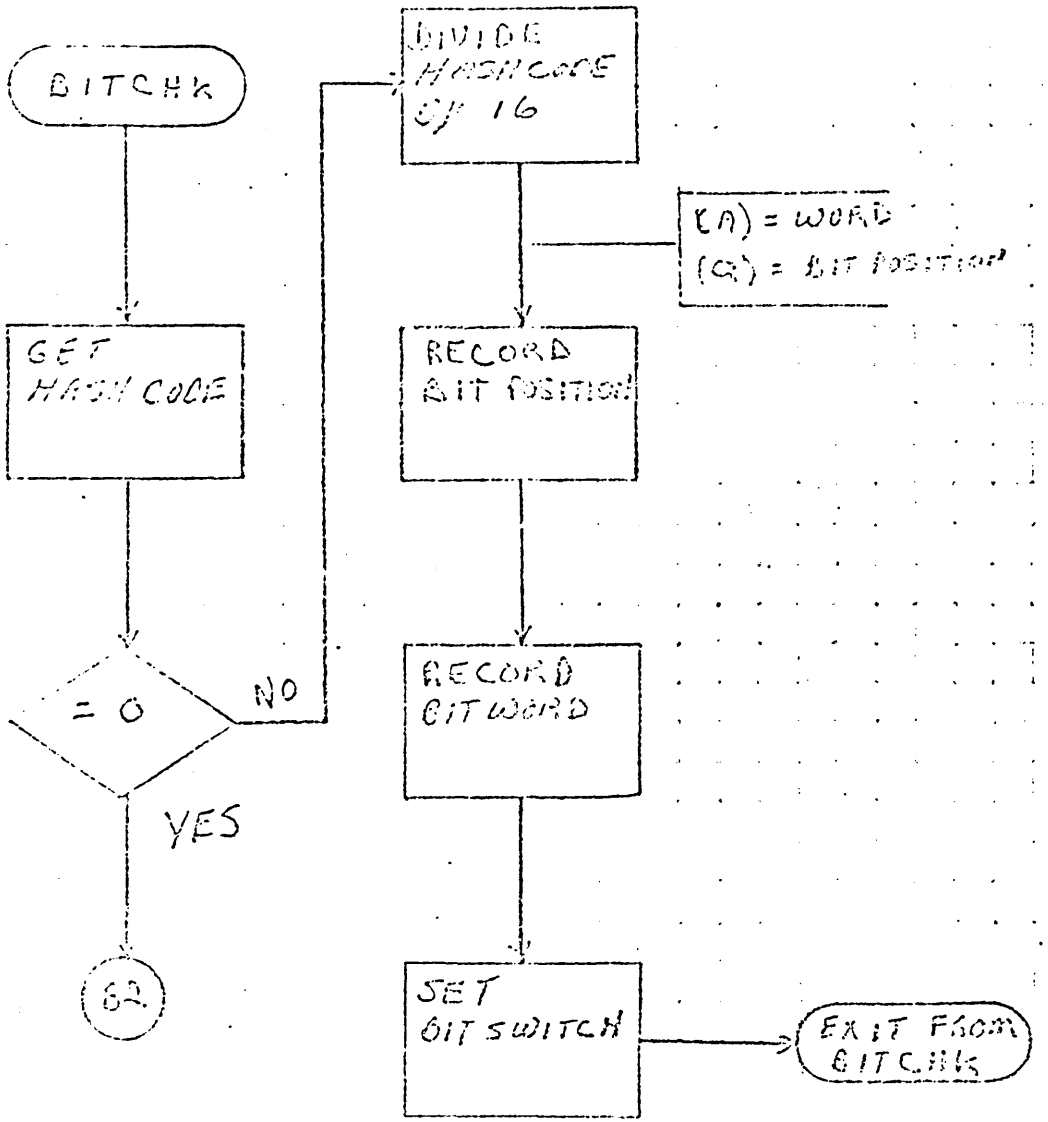
AUG 9 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TASTR2		PROJECT MGR.		
	PAGE 2 OF		PROJECT NAME		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY / DATE		TASK NAME		

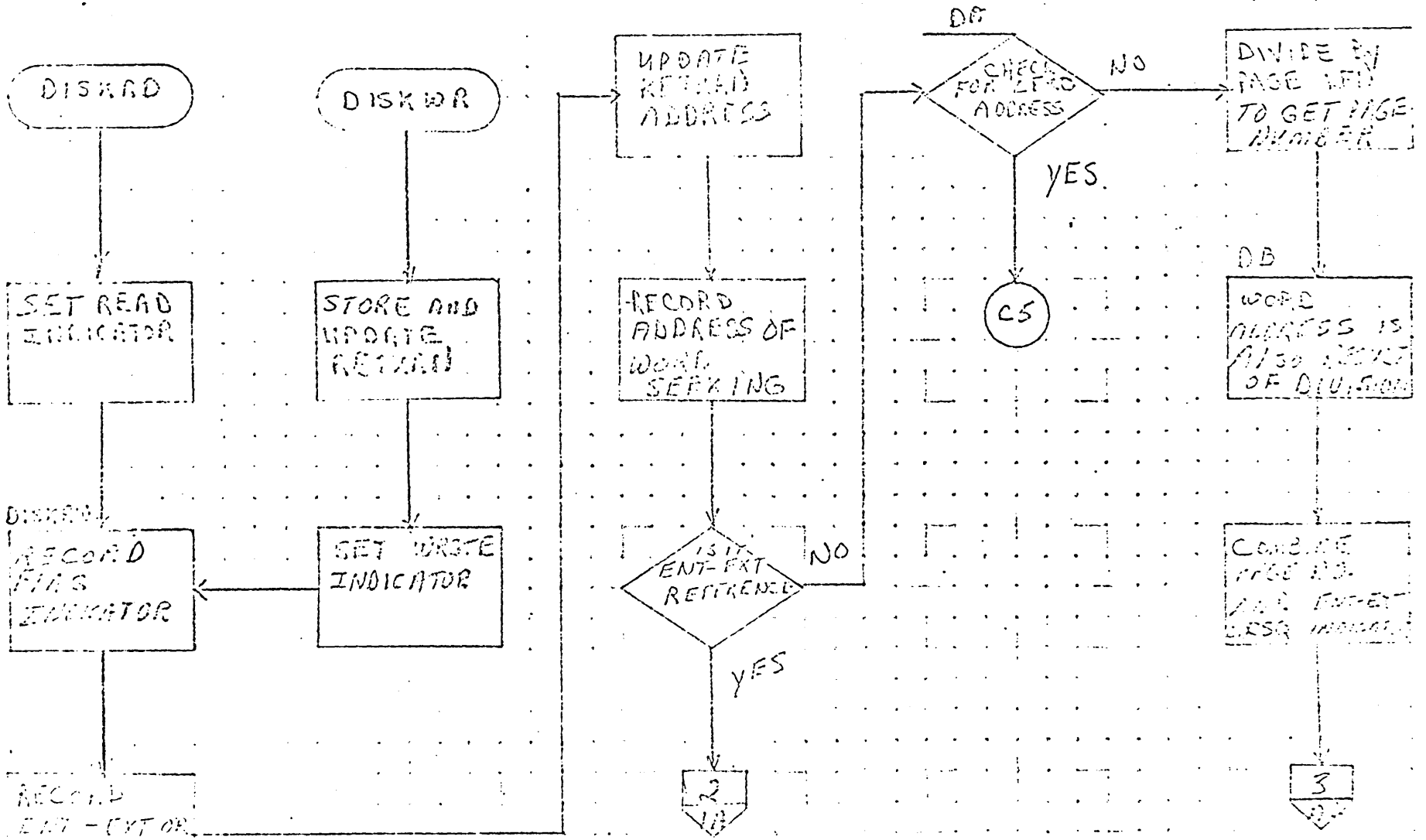


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS [illegible] MACH. TYPE [illegible]	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE EXSTR1	PROJECT MGR.			
FLOWCHART ☐	PAGE 3 OF [illegible]	PROJECT NAME [illegible]			
DECISION TABLE ☐	NUMBER ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY DATE	TASK NAME			



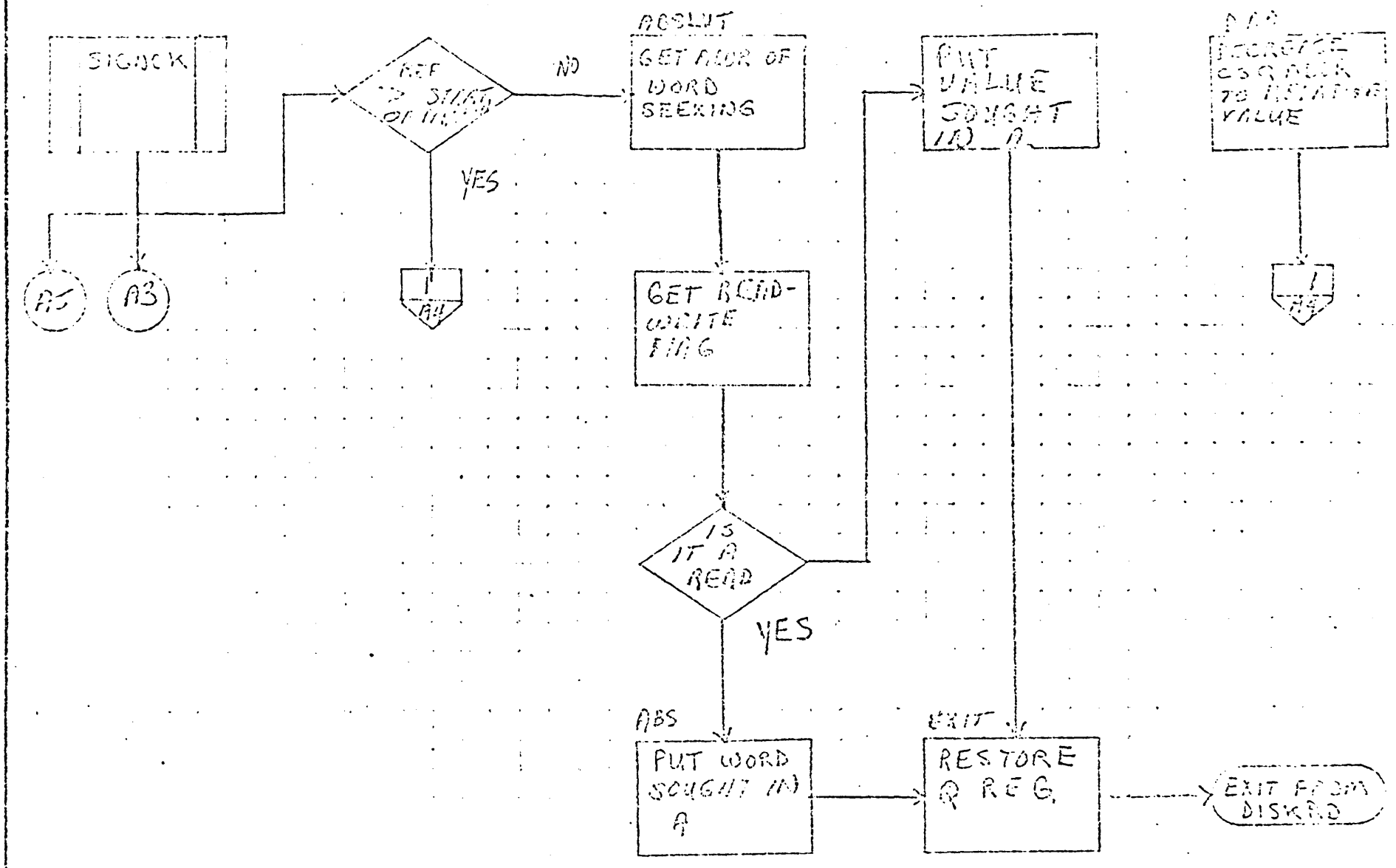
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE TBSTRT		PROJECT MGR.		
	PAGE 4 OF 11		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AUG 3 1971



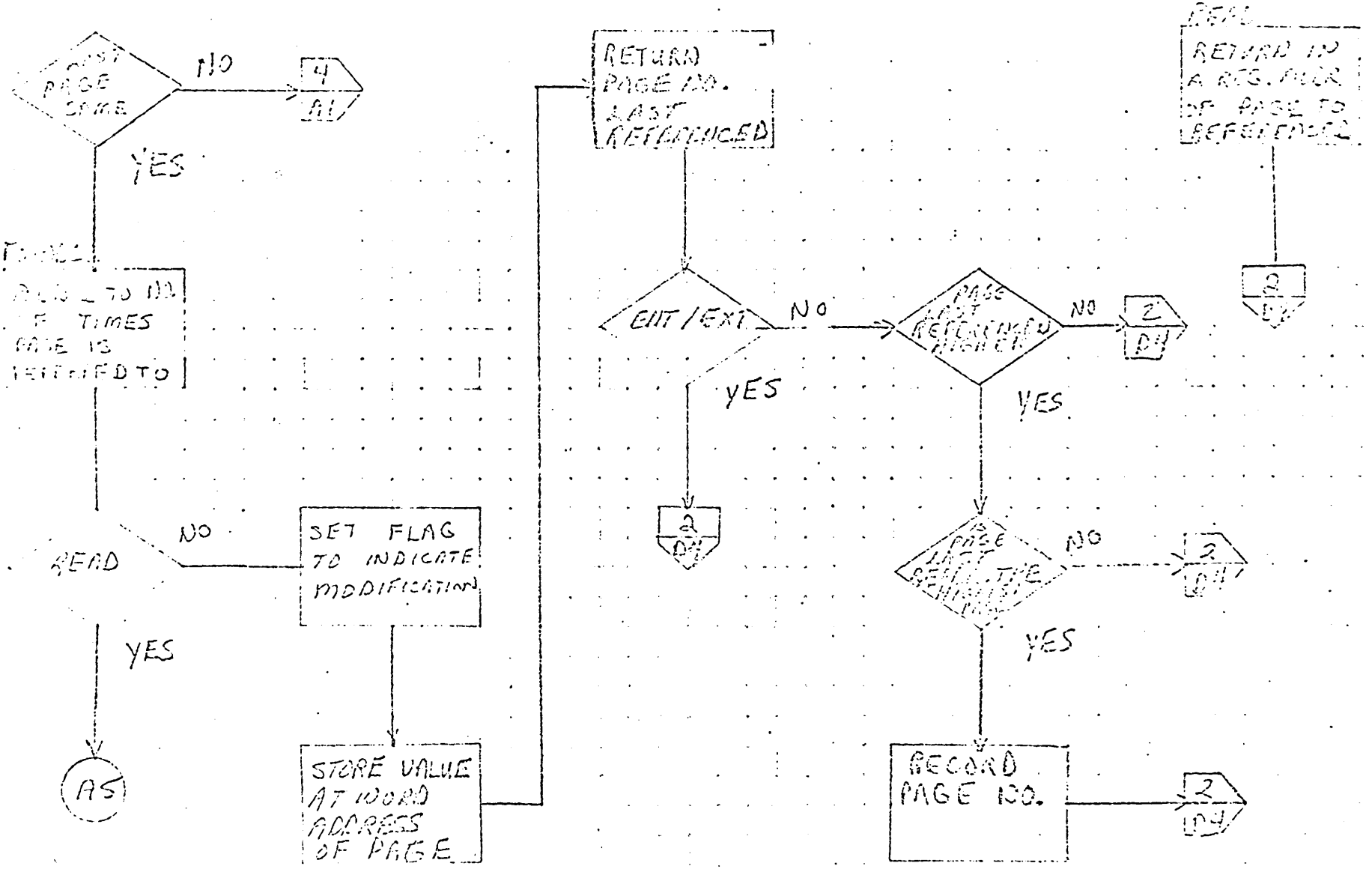
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PAGE		PROJECT MGR.		
	PAGE 1 OF 11		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AUG 1971



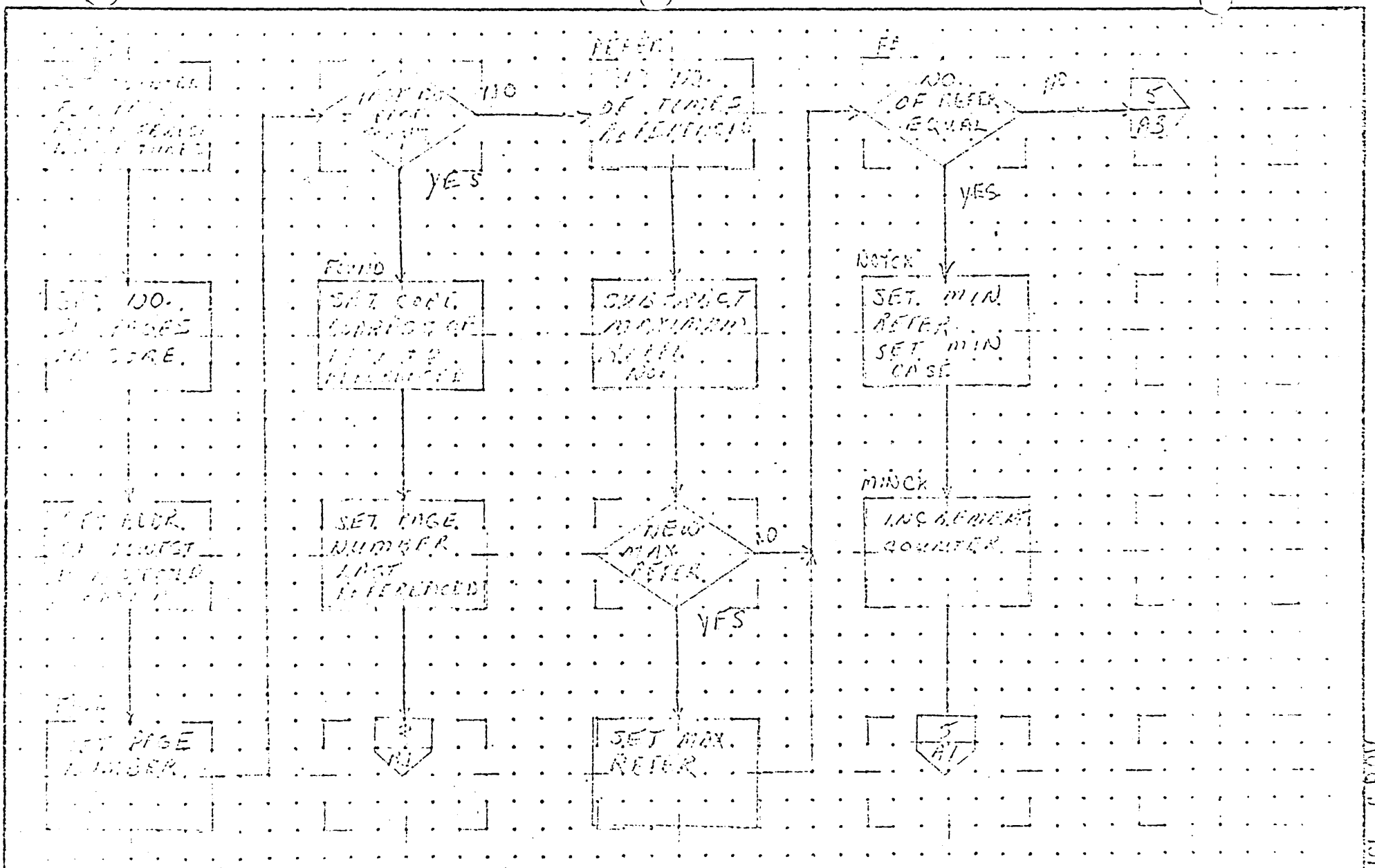
CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE PAGE		PROJECT MGR.		
FLOWCHART ☐	PAGE 3 OF 11		PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AUG 9 1971

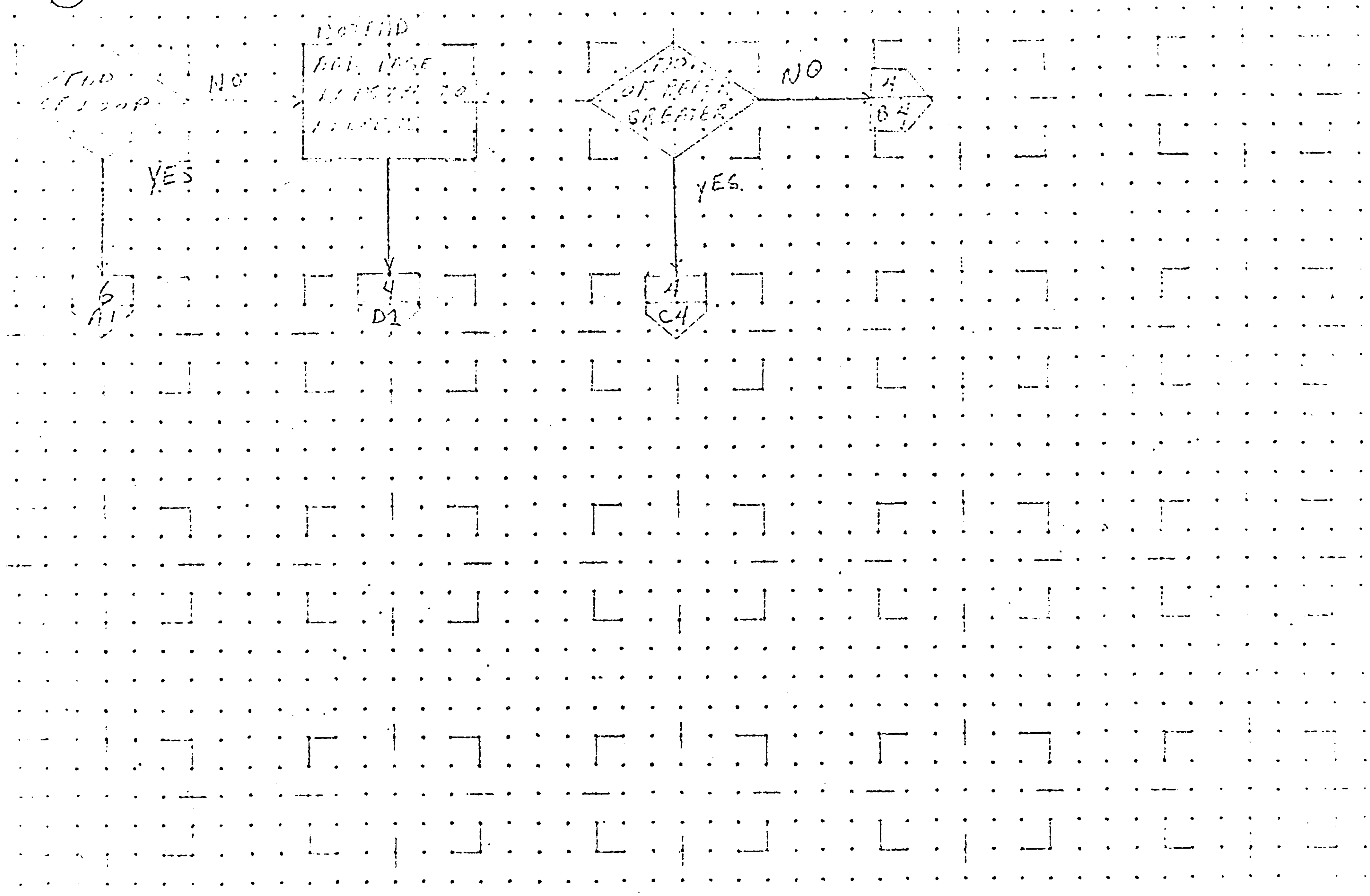


CONTROL DATA CORPORATION						
SOFTWARE DOCUMENT	DOCUMENT CLASS TIMS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.			
FLOWCHART ☐		PAGE OF	PROJECT NAME			
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.			
OTHER ☐	DRAWN BY	DATE	TASK NAME			

AUG 9 1971



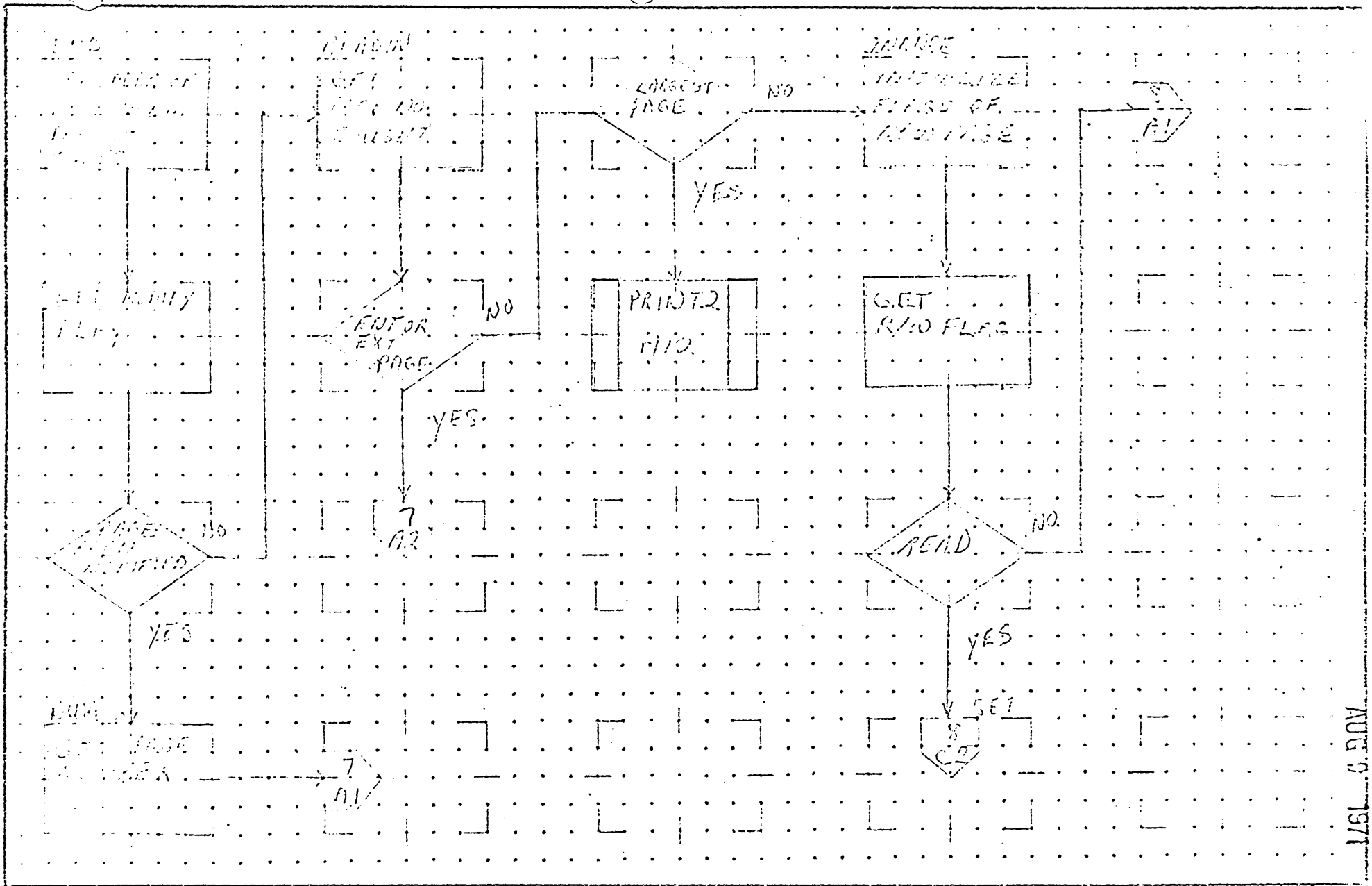
CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE / FLOWCHART / DECISION TABLE / OTHER	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	PAGE OF		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		



CONTROL DATA CORPORATION
SOFTWARE DOCUMENT

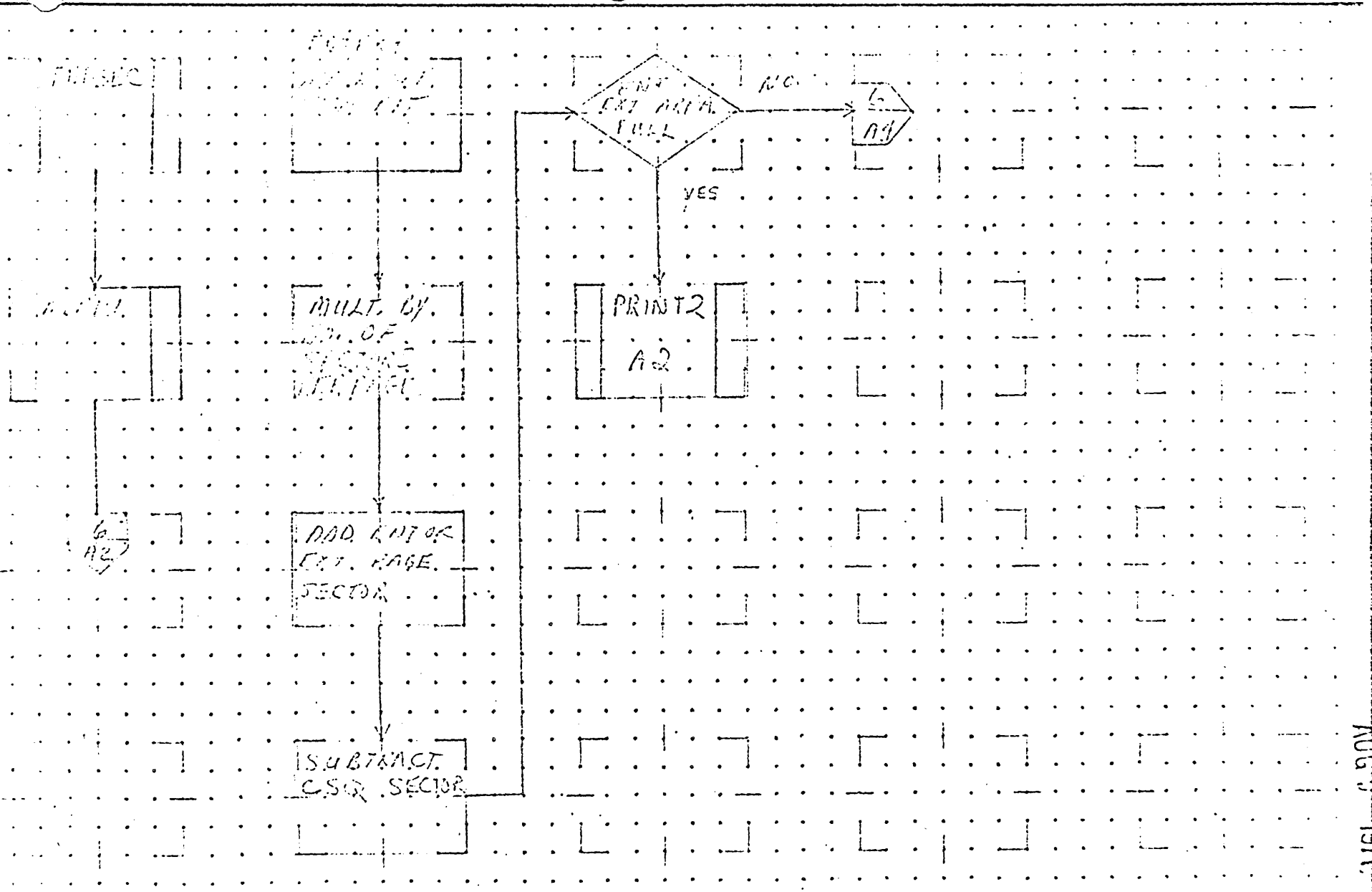
SAMPLE CODE ☐
FLOWCHART ☐
DECISION TABLE ☐
OTHER ☐

DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
DOCUMENT TITLE		PROJECT MGR.			
	PAGE OF 11	PROJECT NAME			
NUMBER	ISSUE DATE	TASK NO.			
DRAWN BY	DATE	TASK NAME			



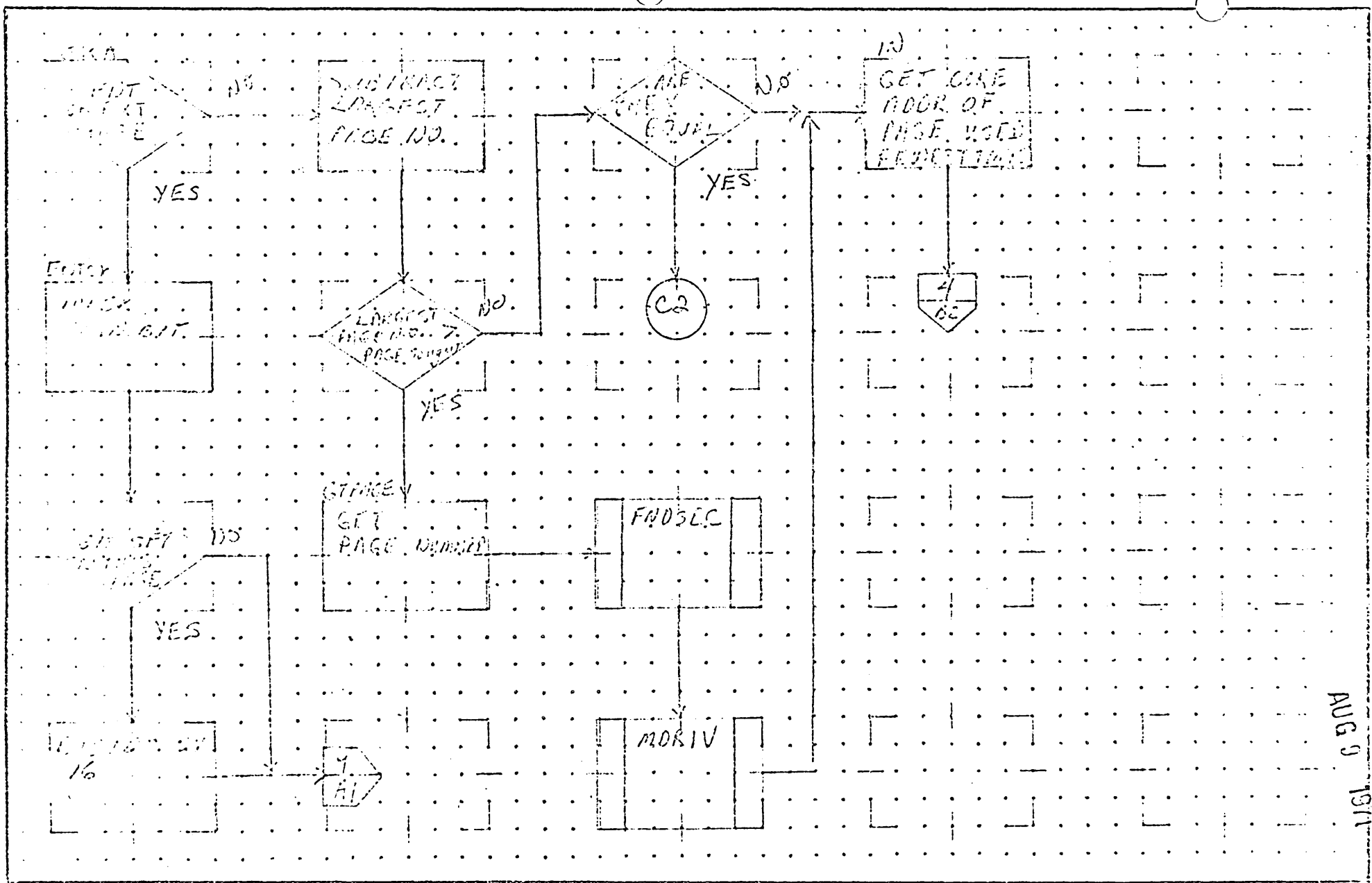
CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS TAC MACH. TYPE	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PAGE	PROJECT MGR.			
	PAGE 6 OF 11	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY DATE	TASK NAME			

AUG 3 1971

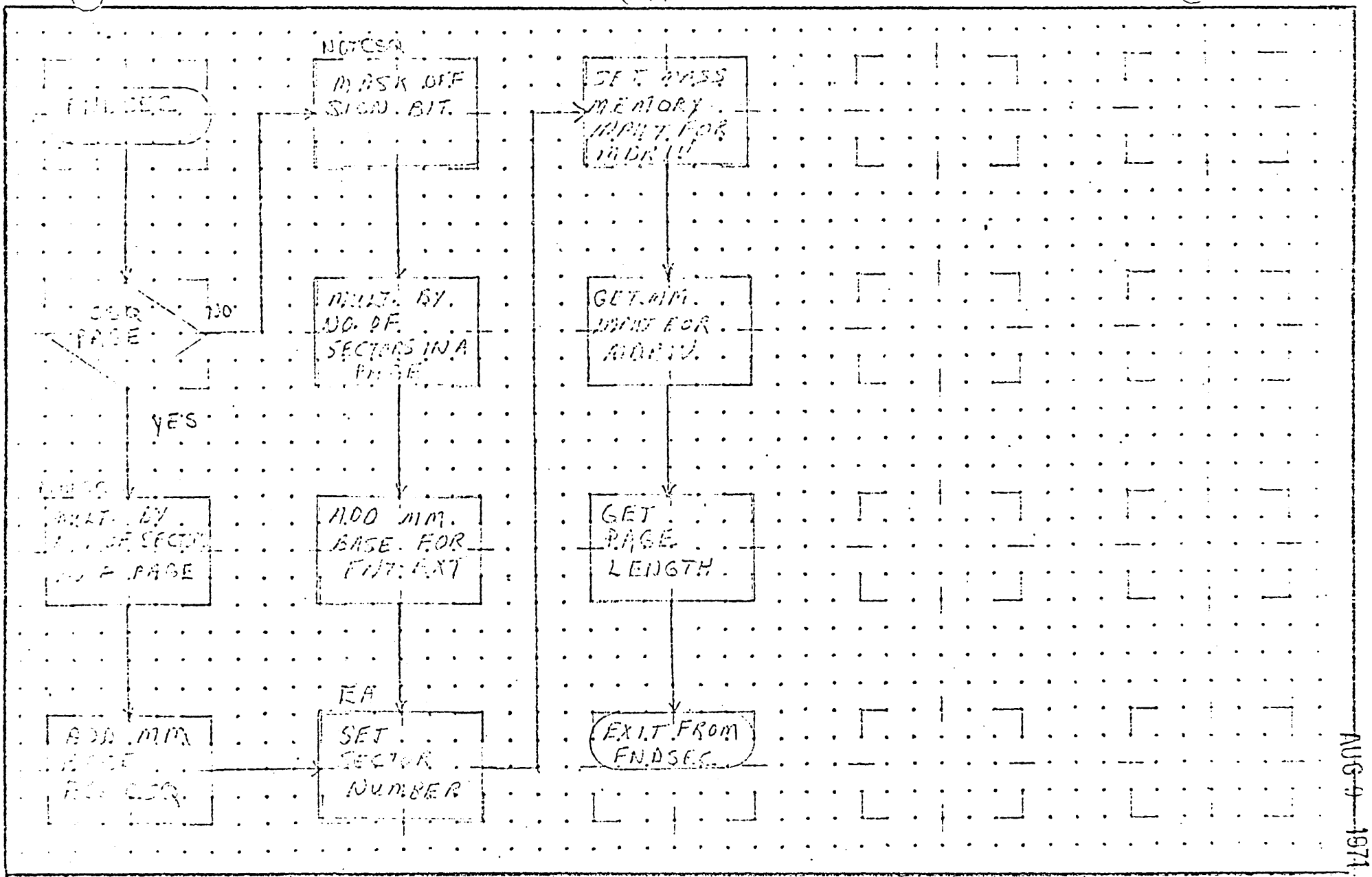


CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☐		PAGE OF	PROJECT NAME		
DECISION TABLE ☐	NUMBER	ISSUE DATE	TASK NO.		
OTHER ☐	DRAWN BY	DATE	TASK NAME		

AUG 3 1971

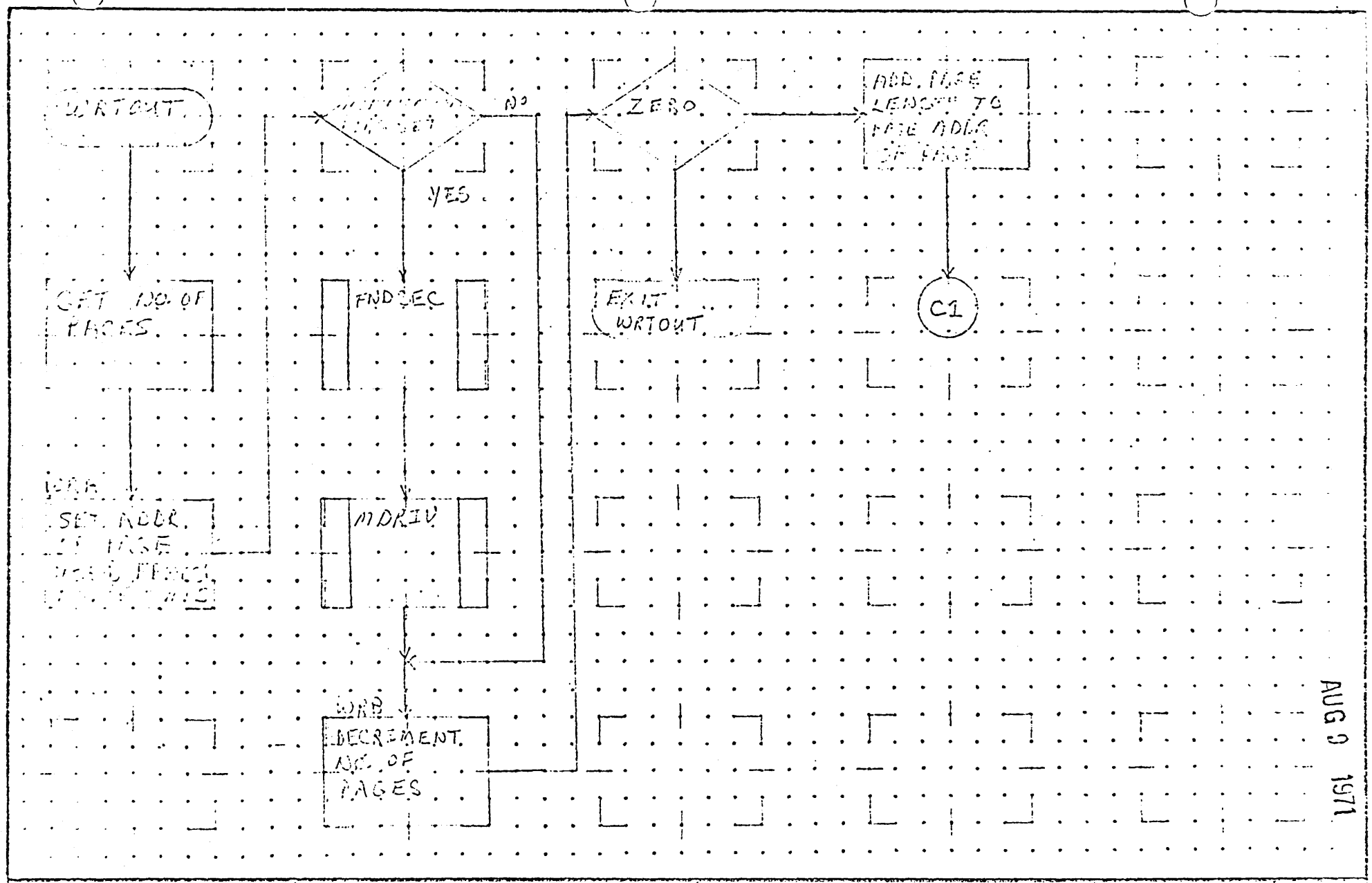


CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	PAGE 9 OF 11		PROJECT NAME		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY G.D.C.	DATE 1/6/71	TASK NAME PART 1 LOADER		



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS [illegible] MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE [illegible]	PROJECT MGR.			
	PAGE [illegible] OF 11	PROJECT NAME 1700 MSOS			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY C. D. C. DATE 4/21/71	TASK NAME PAGER Loader			

AUG 9 1971



AUG 9 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 PAGE	PROJECT MGR.			
	PAGE 11 OF	PROJECT NAME 1700 M.S.O.S.			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY G.D.C. DATE 1/8/71	TASK NAME PART 3 LOADER			

2.0 Partitioned Core Allocator - PRTCDR

2.1 General Description

This routine is designed to allocate the partitions which make up Part 1 of the MSOS 65K Special System.

Partitions are numbered 0 to 15 and there may be as many as 16 partitions in a system. Partition 0 is the lowest (toward 0) in core.

PRICDR essentially has three sections. One section is entered to allocate partitions for a partitioned core request or for a directory request. Another section does the table manipulation and searching for waiting requests. The third section is entered to release partitions.

2.2 Entry Points

PRTCDR	Entry to the section which does the table searching, etc.
K65T12	Entry to release partitions.
PTNREL	Schedule request code which is stuffed by the dynamic core allocator the first time it is entered.
K65C0R	Entry for directory request core allocation.
K65T10	Entry for partitioned core requests.
PTNALC	Schedule request code which is stuffed by the dynamic core allocator the first time it is entered.

2.3 Externals

PARTBL	Partitioned core table located in SYSBUF. This table is divided into three sections all of them 16 words long. The first section contains the beginning location for each partition. It is filled with -0 for any partitions not defined. The second section is the tops of partition threads. There is a thread for each partition defined which is initialized to -0. This section

is filled with 0 for any partitions not defined. The third section contains 16 locations initialized to 0. Whenever a partition or block of partitions is assigned the location of this section associated with the first partition of the block is set to a bit pattern which describes what partitions are in the block. This bit pattern is used to decide what should be released on a release request.

BUSY — Partitions busy word located in SYSBUF. Bits 0 through 15 of this word correspond to partitions 0 through 15 respectively. This word is initialized with bits corresponding to partitions defined set to zero and bits corresponding to undefined partitions set to 1.

LSTLOC — Last location plus one of partitioned core. This word is also located in SYSBUF.

DIP — Driver in progress flag. If this location of SYSBUF is not zero it indicates that the partitioned core allocator has been interrupted during its scan of partition threads.

RPMASK — Location in RW which contains a mask to extract request priority from word 0 of a request.

SCHERR — Entry in RDISP when core is unable to be allocated.

2.4 Detailed Description

2.4.1 Partition Allocation Request Entry

This section is entered at K65T10 or K65C0R for a partitioned core request or a directory request respectively.

On entry

Q = Pointer to request parameter list
I = Pointer to volatile storage

This section checks to see if the requested partition exists. If it does not the error exit from the allocation portion is scheduled and jump is made to REQXT to release volatile etc. If the partition is legal, the request is threaded to the beginning partition's thread. If the allocation portion is

not in progress or it is but has scanned past the partition to which this request is threaded, the allocation portion is scheduled.

2.4.2 Partition Allocation Section

This section runs at the same priority level as the dynamic core allocator. It is scheduled by the Partitioned Core Release request processor or by the Partitioned Core Allocation request processor.

First a scan is made of all the partitions defined in the system beginning at partition 0 to see if any requests can be filled. If not exit is made to the dispatcher.

If a directory request is found which can be filled a read is scheduled from mass storage to bring in the program. If a partitioned core request is found which can be filled the completion address is scheduled, if there is one, and the scan is continued unless it is at the end of the partitions in which case exit is made to the dispatcher.

2.4.3 Partition Release Request Entry

K65T12 is entered from T12 if the request has bit 14 set in parameter word 0.

On entry:

Q = Pointer to request parameter list
I = Pointer to volatile storage

The release request processor checks to see if the partition actually exists and is busy. If so the beginning partition of the block is located and the busy word and use word are set to indicate that the block has been released. After a block is released exit is made to the Partition Allocation section.

2.4.4 Request Parameter Lists

2.4.4.1 Partitioned Core Release Parameter List

Word 0 Request code with bit 14 set and bit zero indicating exit to dispatcher after release.

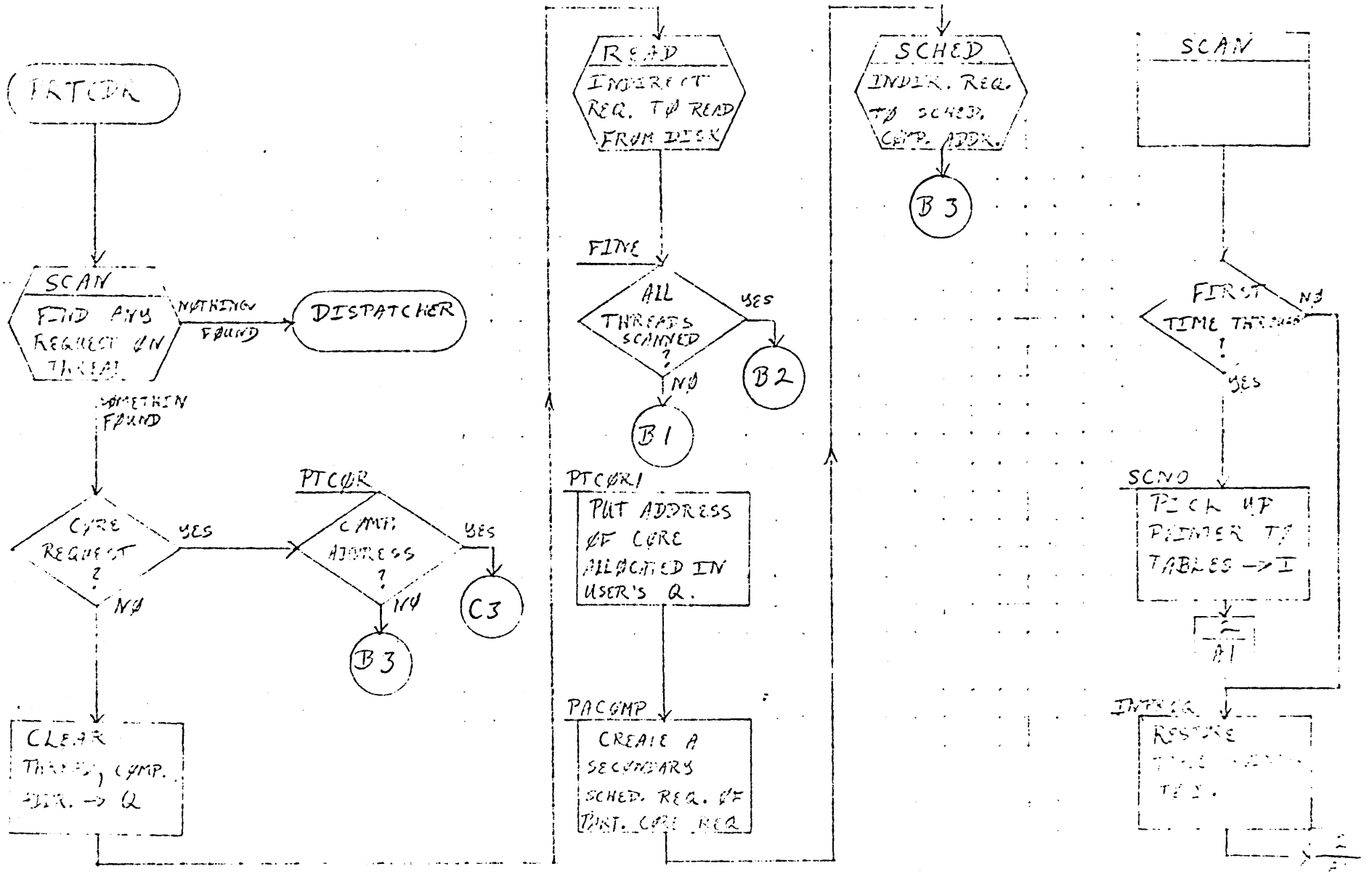
Word 1 Address within block to be released.

2.4.4.2 Partitioned Core Request Parameter List

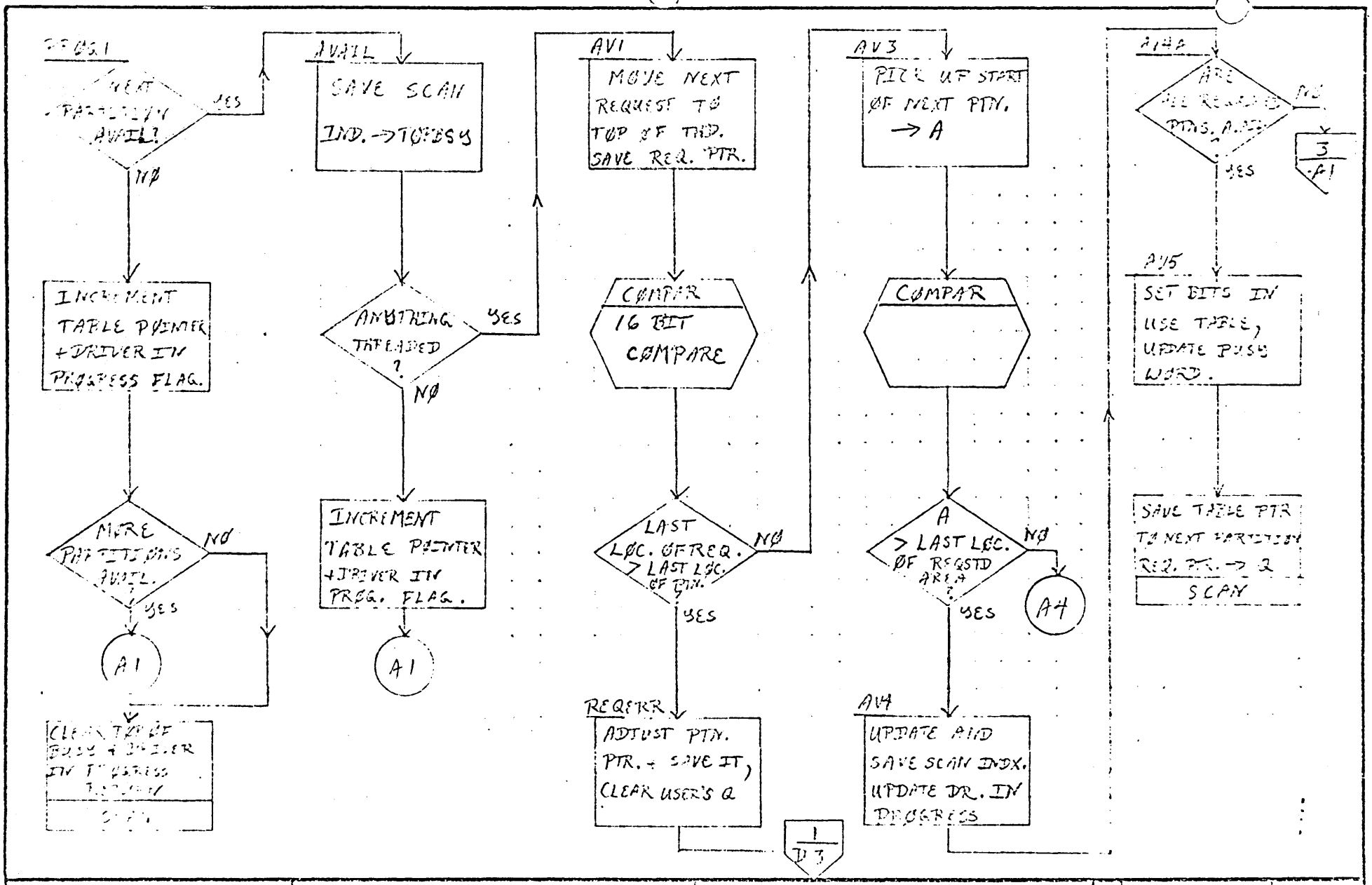
Word 0	Request code and priority levels
Word 1	Completion address
Word 2	Thread
Word 3	Requesters Q
Word 4	Number of words
Word 5	Starting partition {15 or less}

2.4.5 Volatile Contents

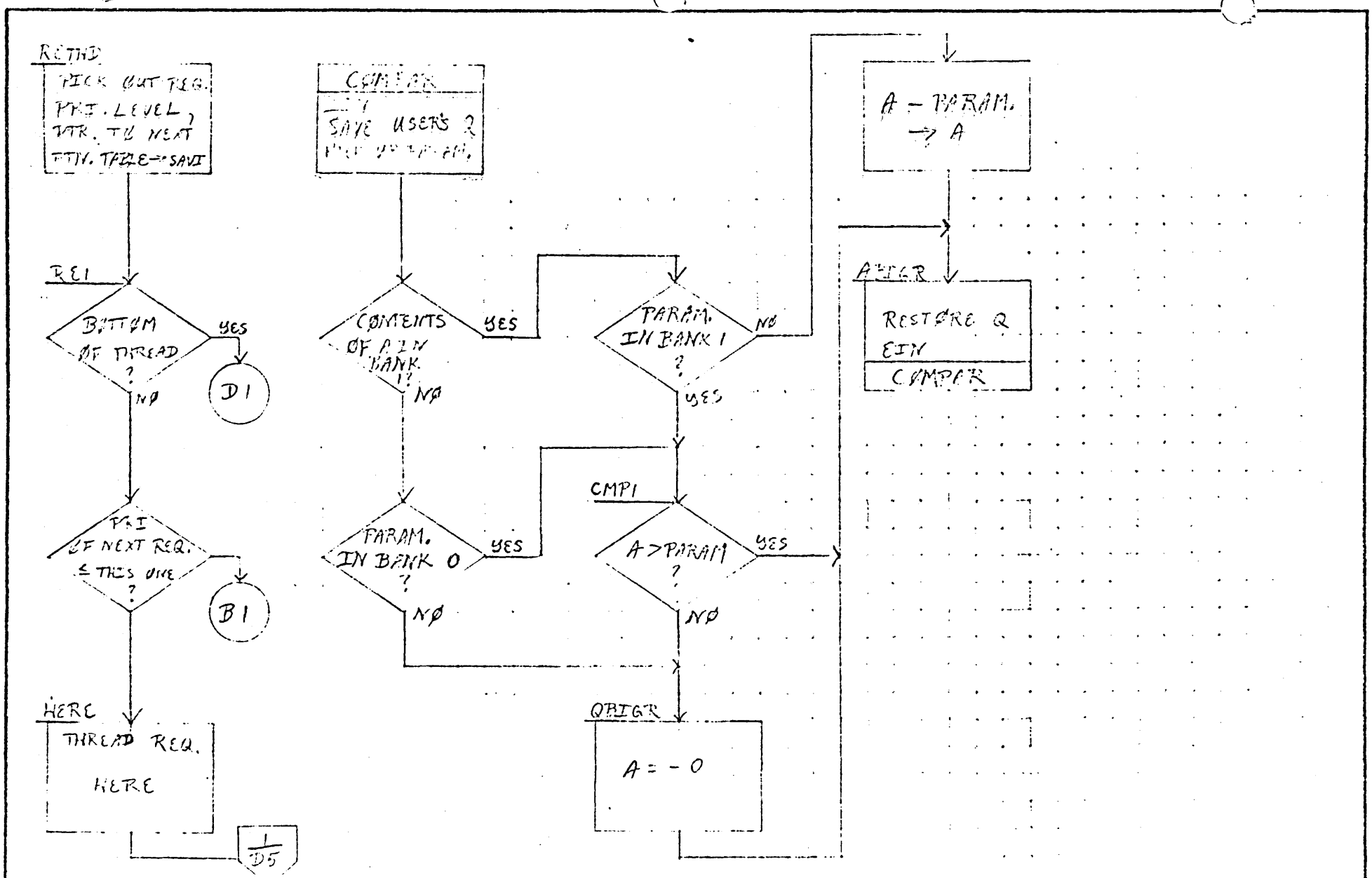
Word 0	Q register of requester
Word 1	A register of requester
Word 2	I register of requester
Word 3	Return address from request processor.
Word 4	Request priority {Partitioned core or directory requests only}
Word 5	Pointer to request
Word 6	Request word 1
Word 7	Request code
Word 8	Indirect request indicator



CONTROL DATA CORPORATION					
SOFTWARE DOCUMENT	DOCUMENT CLASS	MACH. TYPE PPU-C	PROJECT NO.	REV	APPROVED / DATE
SAMPLE CODE ☐	DOCUMENT TITLE		PROJECT MGR.		
FLOWCHART ☑		PAGE 1 OF 5	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 [illegible] MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE PRTY DV	PROJECT MGR.			
	PAGE 2 OF 5	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE [illegible]	TASK NAME			

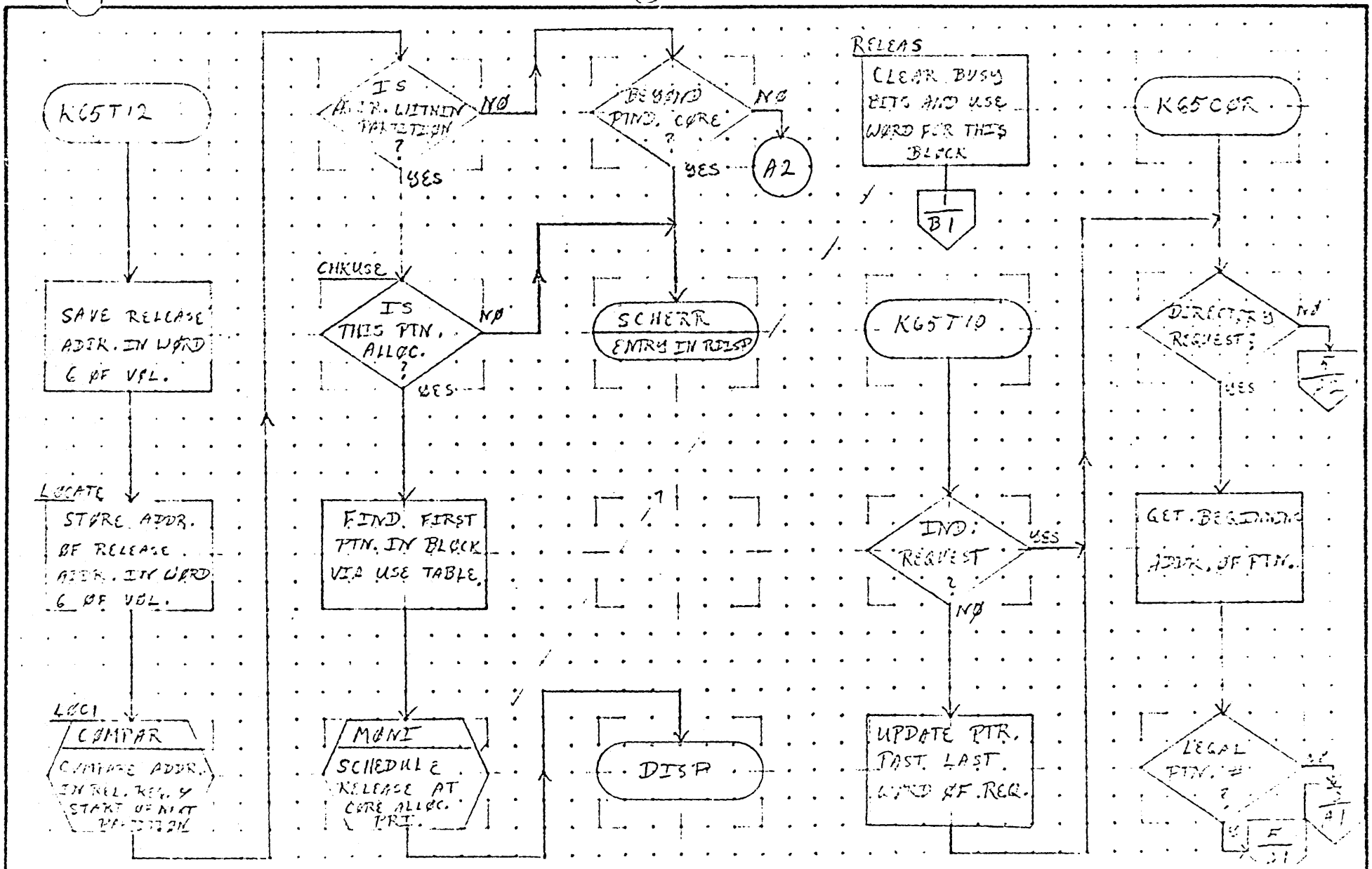


CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS 2115	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE		PROJECT MGR.		
	PAGE 3 OF 5		PROJECT NAME		
	NUMBER	ISSUE DATE	TASK NO.		
	DRAWN BY	DATE	TASK NAME		

AUG 9 1971

AA1385 (FORMERLY CA127-1)

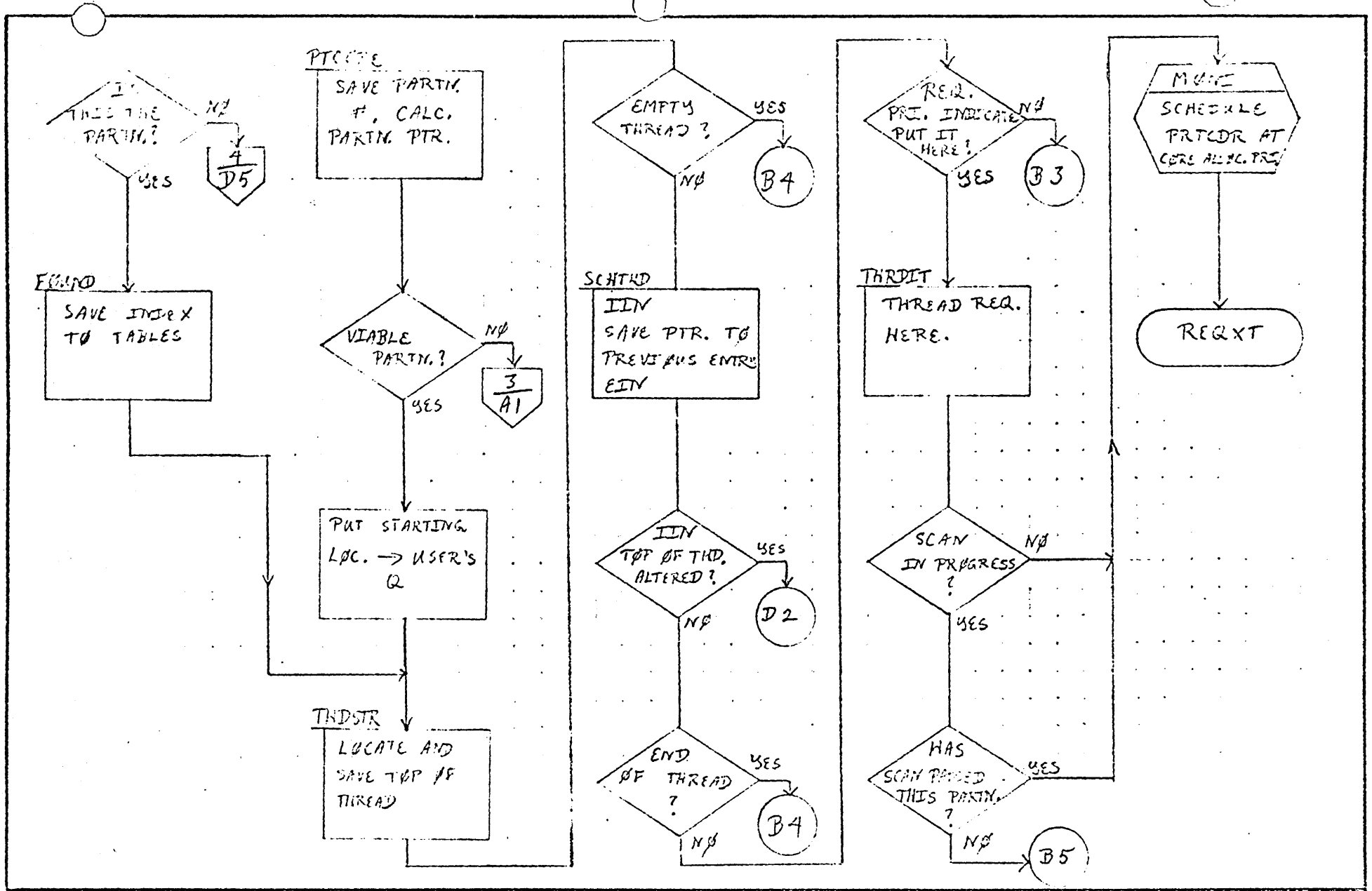
PRINTED IN U.S.A.



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

AA1385 (FORMERLY CA127-11)

AUG 9 1971



CONTROL DATA CORPORATION SOFTWARE DOCUMENT					
SAMPLE CODE ☐ FLOWCHART ☐ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS IRC	MACH. TYPE 1700	PROJECT NO.	REV	APPROVED / DATE
	DOCUMENT TITLE PRTCTR		PROJECT MGR.		
	PAGE 5 OF 5		PROJECT NAME		
	NUMBER / ISSUE DATE		TASK NO.		
	DRAWN BY MFD	DATE 8/11/71	TASK NAME		

AA1385 (FORMERLY CA127-11)

AUG 9 1971

100253

3.0 Part 1 Indirect Request Processor - T16

3.1 General Description

Because bit 15 is not always available to indicate an indirect request, request code 16 has been taken to indicate this. This request should be used whenever there is the possibility that the request referenced indirectly is located in Bank 1 of core.

3.2 Entry Point

T16 - Entry from M0NI

3.3 Externals

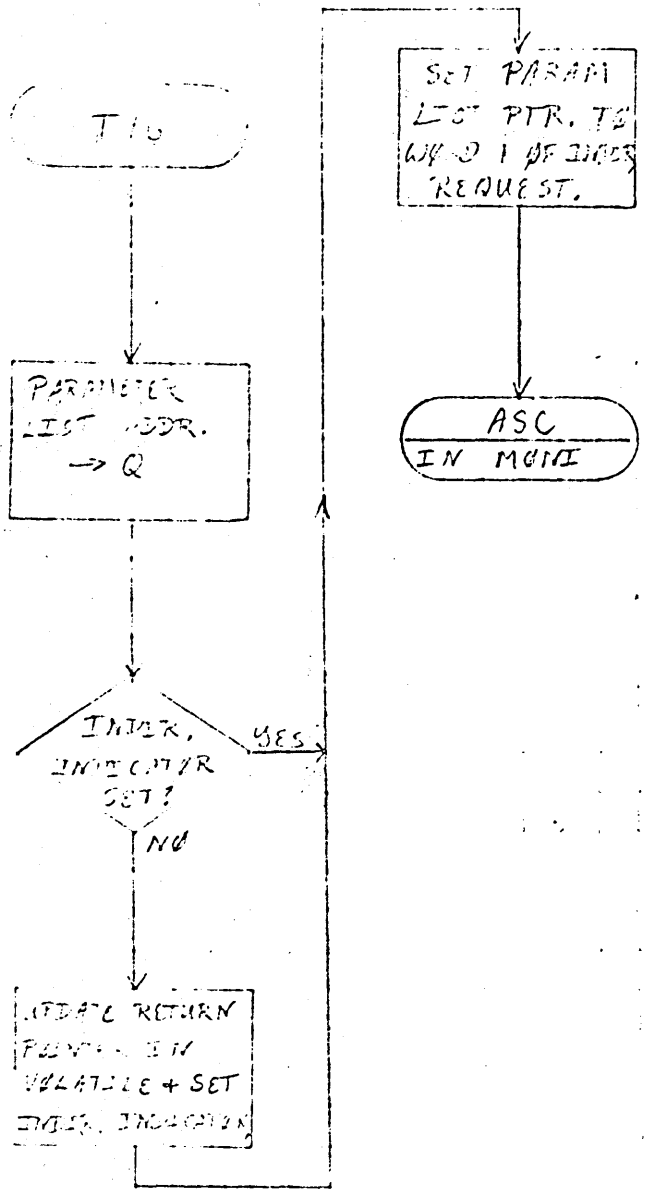
ASC - Entry in M0NI

3.4 Detail Description

This routine is entered from M0NI with

A = Pointer to the parameter list
I = Pointer to Volatile Storage

Word 8 of Volatile is set to -0 if it has not already been set and the pointer to the parameter list is put in word 5 of volatile. Exit is made to ASC in M0NI which continues processing the request.



CONTROL DATA CORPORATION SOFTWARE DOCUMENT SAMPLE CODE ☐ FLOWCHART ☒ DECISION TABLE ☐ OTHER ☐	DOCUMENT CLASS T1MS MACH. TYPE 1700	PROJECT NO.	REV	APPROVED	DATE
	DOCUMENT TITLE T16	PROJECT MGR.			
	PAGE 1 OF 1	PROJECT NAME			
	NUMBER ISSUE DATE	TASK NO.			
	DRAWN BY [illegible] DATE 7/6/71	TASK NAME			