digital
software

OS/8
Device Extensions User's Guide

Order No. AA-D319A-TA

OS/8
Device Extensions User's Guide

Order No. AA-D319A-TA

ABSTRACT

This document describes the software support for the KT8A Memory Management Option and the RX02 and RL01 devices.

SUPERSESSION/UPDATE INFORMATION: This manual is an update of sections of the OS/8 Handbook (DEC-S8-OSHBA-A-D).

OPERATING SYSTEM AND VERSION: OS/8 V3D

To order additional copies of this document, contact the Software Distribution Center, Digital Equipment Corporation, Maynard, Massachusetts 01754

digital equipment corporation · maynard, massachusetts

First Printing, December 1978

The postage-prepaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

CONTENTS (Cont.)

FIGURES

TABLES

1.0 INTRODUCTION AND OVERVIEW

The OS/8 V3D Device Extensions support the following new devices under OS/8.

- The KT8A Memory Management Option (limited support)
- The RL01 Disk and Controller
- The RX02 Double-Density Diskette and Controller

In addition, the Extensions package is a support release for RTS/8 V3 and MACREL/LINK Version 2, both of which can use the extended memory provided by the KT8A. RTS/8 V3 also supports the RL01 and RX02.

The Extensions package is a superset of some OS/8 modules. It remains completely compatible with OS/8 V3D and contains the modules listed in Table 1.

Table 1
Device Extensions Modules

System Programs:		
Name	New Version Number	Comment
OS8 MONITOR	3S	system head, capable of being run from the device, supports 128K words of memory
ABSLDR.SV	6A	loads binary and image code into fields >7
PAL8.SV	13A	uses fields >7
CCL.SV	7A	CCL MEMORY command recognizes up to 128K words available in system
PIP.SV	14A	works with RL01, RX02, VXA0, and new system head
RESORC.SV	4A	includes RL01, RX02, and VXA0
BOOT.SV	7A	includes primary bootstrap for RL01, RX02, and VXA0
RXCOPY.SV	5A	formats single or double density diskettes, copies single density to single density and double density to double density
FUTIL.SV	8A	recognizes new core control block format for programs in extended memory

(continued on next page)

Table 1 (Cont.)
Device Upgrade Kit Modules

Patches: (to be added with LOAD and SAVE commands)		
Name	New Version Number	Comment
BPAT.BN		Patch for BASIC
FPAT.BN		Patch for FORTRAN IV
Handlers: (to be inserted with BUILD)		
RLSY.BN		RL01 System Handler
RL0.BN		RL01 Non-system Handlers
RL1.BN		
RL2.BN		
RL3.BN		
RLC.BN		
VXSY.BN		VXA0 System Handler
RXSY1.BN		RX01 System Handler
RXSY2.BN		RX02 System Handler
RXNS.BN		RX01-RX02 Non-system Handler

This manual assumes that the user is familiar with the material in the following documents:

<u>OS/8 Handbook</u> (DEC-S8-OSHBA-A-D)
<u>OS/8 Handbook Update</u> (DEC-S8-OSHBA-A-DN4)
<u>OS/8 Software Support Manual</u> (DEC-S8-OSSMB-A-D)
<u>KT8A Memory Management Control User's Guide</u> (EK-KTO8A-UG-001)
<u>RL8A Disk Controller Maintenance Manual</u> (EK-RL8A-TM-001)

1.1 Distribution Media

The OS/8 Device Extensions are distributed on the following media.

RX02 diskette
RL01 disk
RK05 disk
TD8E DECtape

1.2 The RESORC Program

The new RESORC program lists the system and non-system handlers for the RX02 and RL01 devices. In addition, it lists a special handler -- called VXA0 -- that enables you to use the extended memory provided by the KT8A as though it were a separate device.

RESORC now has an overlay structure, enabling a user who buys the source program to enter information on user-written handlers.

1.3 Changes in BASIC and FORTRAN IV for RL01 and RX02 Users

RL01 and RX02 users must add the following patches to the BASIC and FORTRAN IV run-time systems so that these programs recognize and properly allocate space in memory for the second page of the system handlers.

To patch the BASIC run-time system, enter the following commands.

```
.LOAD SYS:BRTS.SV/I
.LOAD dev:BPAT.BN
.SAVE SYS:BRTS.SV
```

where

dev   is the distribution device

BPAT.BN   is the BASIC patch

To patch the FORTRAN IV run-time system, enter the following commands.

```
.LOAD SYS:FRTS.SV/I
.LOAD dev:FPAT.BN
.SAVE SYS:FRTS.SV
```

where

dev   is the distribution device

FPAT.BN   is the FORTRAN IV patch

1.4 System-Wide Changes for Users with the KT8A

KT8A users must ensure that user-written programs and user-written handlers do not contain the following combination of instruction steps.

```
CIF     /Change instruction field
IOT     /Any PDP8 IOT instruction
JMP I   /The instruction that does the CIF
```

If you enable the KT8A and turn on the interrupts (for example, to run OS/8 as a background task under RTS8), the KT8A hardware will return to the wrong place on traps between the CIF and JMP I instructions.

1.5 Changes in PIP for RL01, RX02, and VXA0

The new version of PIP recognizes the RL01, RX02, and VXA0 devices. PIP sets the proper length for directories on the ZERO command and determines whether it is dealing with a double-density or single-density diskette.

PIP also recognizes the new Monitor head. If you attempt to use the Y option on the old version of PIP to move the new system head, PIP responds with the error message

```
BAD SYSTEM HEAD
```

1.6 The BOOT.SV Program

The BOOT.SV program now includes a primary bootstrap for RL01, RX02, and VXA0. The format is

```
.BOOT dd
```

or

```
.BOOT
/dd
```

where

dd is a legal OS/8 device specification, including RL, RX, or VX.

1.7 Changes in FUTIL

The new version of the OS/8 file utility program FUTIL recognizes the new Core Control Block format for user-programs in extended memory. For a complete description of FUTIL, see the OS/8 Handbook Update.

2.0 BOOTSTRAP AND BUILD INSTRUCTIONS

Since the Extensions package includes the system head, you can bootstrap the RX01, RX02, RL01, or RK05 distribution medium as a system device.

Table 2 and Table 3 contain the bootstraps for the RX02 and RL01 device. The bootstraps for the RK05 disk and the TD8E DECtape are included in Chapter 1 of the OS/8 Handbook.

The new handlers must be inserted into OS/8 with the BUILD program. For information on adding handlers to OS/8, see the BUILD chapter in the OS/8 Handbook.

NOTE

The console instructions in Tables 2 and 3 describe a PDP8-A. For other PDP8 computers, see the OS/8 Handbook.

Table 2
RX02 Bootstrap

1.	Press in order the MD and DISP buttons to display memory data in the octal readout.
2.	Press in order 0 and LXA to select memory field 0.
3.	Press in order 20 and LA to start loading instructions at location 20.

(continued on next page)

Table 2 (Cont.)
RX02 Bootstrap

4. Deposit the following octal values, terminating each value with D NEXT.

00020	1061
00021	1046
00022	0060
00023	3061
00024	7327
00025	1061
00026	6751
00027	7301
00030	4053
00031	4053
00032	7004
00033	6755
00034	5054
00035	6754
00036	7450
00037	5020
00040	1061
00041	6751
00042	1061
00043	0046
00044	1032
00045	3060
00046	0360
00047	4053
00050	3002
00051	2050
00052	5047
00053	0000
00054	6753
00055	5033
00056	6752
00057	5453
00060	0420
00061	0020

5. After you have deposited all the values, press 0033 and LA to start the program at location 33.

6. To start the bootstrap program, press INIT and RUN.

Table 3
RL01 Bootstrap

1. Press in order the MD and DISP buttons to display memory data in the octal readout.

2. Press, in order, 0 and LXA to select memory field 0.

3. Press, in order, 1 and LA to start loading instructions at address 1.

(continued on next page)

Table 3 (Cont.)
RL01 Bootstrap

4. Deposit the octal values given below, following each value with D NEXT.

Address	Contents
00001	6600
00002	7201
00003	4027
00004	1004
00005	4027
00006	6615
00007	7002
00010	7012
00011	6615
00012	0025
00013	7004
00014	6603
00015	7325
00016	4027
00017	7332
00020	6605
00021	1026
00022	6607
00023	7327
00024	4027
00025	0377
00026	7600
00027	0000
00030	6604
00031	6601
00032	5031
00033	6617
00034	5427
00035	5001

5. After all values are deposited, press, in order, 0001 and LA to allow the program to start at location 1.

6. Press, in order, INIT and RUN to start the bootstrap program.

The complete RX02 and RL01 bootstrap programs are listed in Appendix A and B.

3.0 THE KT8A MEMORY MANAGEMENT OPTION

The OS/8 V3D Device Extensions provide limited support for the KT8A Memory Extension and Management Option, which increases the amount of allowable memory in PDP8 systems from 32K to a maximum of 128K words.

The KT8A supports all available sizes of continuous memory from 32K to 128K.

System programs, devices, and languages that run in 32K under OS/8 will also operate with the new monitor. In addition, systems with the KT8A and 128K software support will run user-written programs in memory fields 0 to 37. OS/8 high-level languages and system programs, however, do not make use of memory greater than 32K words.

This section describes the OS/8 commands and PAL8 instructions that allow you to run user-written programs in fields 0 through 37. In addition, it includes a subroutine for finding the amount of memory available at run-time and describes a program that enables you to change the Core Control Block of a program in complex SAVE operations.

This section also notes current software restrictions on the use of the extended memory.

For a description of the KT8A device, including operating and programming instructions, see the KT8A Memory Management Control User's Guide (EK-KT08A-UG-001).

3.1 128K Monitor and CCL Commands -- SAVE, ODT, and MEMORY

The SAVE and ODT monitor commands now support fields 0 to 37. The CCL MEMORY command finds the highest field available in hardware up to field 37. MEMORY also limits the available fields in software, but this feature is currently restricted to 32K.

NOTE

> The OS/8 Monitor currently requires that all user-written programs contain at least one segment (1-page minimum) below 32K.

3.1.1 The SAVE Command

The SAVE Command - The SAVE command makes a memory-image file of the program currently in memory, assigns it a name, and saves it on a device. You can specify areas in memory that you want to save in fields from 0 to 37.

The format of the command, including all optional arguments, is

```
SAVE device:file.ex ffnnnn-ffmmmm,ffpppp;ffssss=cccc
```

where

ffnnnn — is a 6-digit octal number representing a field from 0 to 37 (ff) and the first address of a continuous portion of memory you want to save.

ffmmmm — is the final address (in the same field) of the section of memory you want to save.

ffpppp — is a 6-digit octal number representing the field and address of one location in memory. If you specify a single address on an even-numbered page in the command, SAVE writes the entire page on which the location occurs. If you specify an odd-numbered page, SAVE also saves the preceding page.

;ffssss — is a 6-digit octal number representing the field and starting address of the program you are saving.

=cccc — is a 4-digit octal number representing the contents of the Job Status Word for the program. (See below.)

If you omit the extension on the file name, SAVE appends .SV. If you omit the other arguments, SAVE finds the locations it requires in the current Core Control Block. (For a discussion of the Core Control Block, see the OS/8 Handbook and the OS/8 Software Support Manual.)

The SAVE command places the following restriction on arguments in the command line.

- You must specify the output device. SAVE does not default to DSK.
- The first and last location of a segment in memory (ffnnnn-ffmmmm) that you wish to SAVE must both exist in the same field. You may not cross field boundaries. In the following example, both entries specify field 22.

  ```
  .SAVE SYS:EXAMPL 220055-220643
  ```

- When you specify an area on a page, SAVE takes the entire page. If you call for another part of that page in the same command line, SAVE sends a BAD ARGS error message to the terminal informing you that it has already saved the page.

  ```
  .SAVE RXA1:FLOP 120077-120122, 120146-120177
  ```

  The first argument writes locations 77 to 122 in field 12 on to RXA1 and calls the file FLOP.SV. The second argument, which specifies locations on the same page, produces the error message

  ```
  BAD ARGS
  ```

- Do not SAVE locations 7600-7777 in fields 0, 1, and 2. The resident Monitor code resides in these areas of memory. To avoid accidently destroying a portion of the Monitor, restrict SAVE operations involving 7600 to fields above 2.
- If you specify an address on an odd-numbered page, SAVE can save it only if it also saves the preceding page. The system does this automatically.

If you wish to specify more locations in a SAVE command than you can fit in a single command line, use the SAVECB program described in Section 3.5.

NOTE

> The Monitor START command currently accepts field specifications in the range of 0 to 7 only.

3.1.2 **The ODT Command** - ODT accepts and returns 6-digit addresses in the following commands.

```
ffnnnn/
ffnnnnB
ffnnnnG
```

where

ff	is a field from 0 to 37
nnnn	is a location

The D and F command allow you to specify fields in the range of 0 to 37. To indicate the first eight fields, type a single octal digit (0-7). Note that this is a change from previous versions of ODT, which required you to enter field specifications as multiples of 10 (for example, field 2 as 20). Table 4 summarizes all of the OS/8 128K ODT commands. For complete information on ODT, see the chapter on the ODT program in the <u>OS/8 Handbook</u>.

Table 4
128K ODT Command Summary

Command	Operation
ffnnnn/	Open location ffnnnn, where ff is a field from 0 to 37. ODT displays the contents of the location, prints a space, and waits for you to enter a new value for the location or close the location. If you omit ff, ODT assumes field 0.
	Reopen the most recently opened location.
nnnn;	Deposit nnnn in the currently opened location, close the location, and open the next location in the sequence for modification. The semicolon (;) lets you deposit a series of octal values in sequential locations. To skip locations in the sequence, type a semicolon for each location you wish to leave unchanged.
RETURN key	Close the currently open location.
LINE FEED key	Close the currently open location, open the next location in the sequence for modification, and display its contents.
n+	Open the current location plus n and display the contents.
n-	Open the current location minus n and display the contents.
uparrow or circumflex	Close the location, read its contents as a memory-reference instruction and open the location it points to, displaying its contents. • ODT makes no distinction between instruction op-codes when you use this command. It treats all op-codes as memory-reference instructions. • Take care when you use this command with indirectly referenced auto-index registers. If you use the command in this way, the contents of the auto-index register is incremented by one. Check to see that the register contains the proper value before proceeding.

(continued on next page)

Table 4 (Cont.)
128K ODT Command Summary

Command	Operation
(underline)	Close the current location, read the contents as a twelve-bit address, and open that location for modification, displaying its contents.
ffnnnnB	Establish a breakpoint at location ffnnnn, where ff indicates a field from 0 to 37. ODT permits only one breakpoint at a time.
ffnnnnG	Transfer control of the program to location ffnnnn, where the first two digits (ff) represent a memory field.
B	Remove the breakpoint, if one exists.
A	Open for modification the location in which ODT stored the contents of the accumulator when it encountered the breakpoint.
L	Open for modification the location in which ODT stored the contents of the Link when it encountered the last breakpoint.
M	Open the Search Mask location, initially set to 7777. To change the Search Mask, type a new value into the location.
M <LF>	Open the lower search-limit location. Type in the location (four octal digits) where the search will terminate.
M <LF><LF>	Open the upper search-limit location. Type in the location (four octal digits) where the search will terminate.
nnnnW	Search the portion of memory defined by the upper and lower limits for the octal value nnnn. The search must be restricted to a single memory field. See the F command.
D	Open for modification the location containing the data field (0 to 37) that was in effect at the last breakpoint. To change the field, enter a number from 0 to 37.
F	Open for modification the word containing the field (0 to 37) used by ODT in the last W or uparrow command (search or indirect addressing) or in the last breakpoint, depending on which occurred most recently. To modify this location, enter a number from 0 to 37.
CTRL/O	Interrupt a lengthy search output and wait for the next ODT command.
DELETE key	Cancel a number previously typed, up to the last non-numeric character entered. ODT responds with a question mark, after which you enter the correct location.

3.1.3 **The CCL Memory Command** - The MEMORY command finds the highest field available in hardware up to field 37. It also limits the available fields in software, but this feature is currently restricted to 32K words.

The format of the command is

```
MEMORY
```

or

```
MEMORY nn
```

where

nn is an octal number in the range of 0 to 37 representing the number of 4K fields available to OS/8.

Table 5 lists all the values of n (memory fields in octal) and the corresponding memory-size.

Table 5
Field Specifications for 128K MEMORY Command

n	Words of Memory
0	all available memory
1	8K
2	12K
3	16K
4	20K
5	24K
6	28K
7	32K

To limit memory, enter the highest file you want to make available to OS/8 in the command line. For example, the following command limits the available memory to 16K words.

```
.MEM 3
```

To find the amount of memory that OS/8 is using, type the command with no argument.

```
.MEMORY

12K OF 32K MEMORY
```

In this example, MEMORY prints the information that a 32K system has been limited to 12K words.

MEMORY caused the execution of the CCL.SV program.

3.2 128K PAL8

The following PAL8 instructions accept field specifications in the range of 0 to 37, permitting you to run programs in areas above 32K.

3.2.1 **The FIELD Pseudo-Operator** - The pseudo-op FIELD instructs PAL8 to output a field setting so that it can recognize more than one memory field.

The format of this pseudo-op is

```
FIELD ff
```

where

ff — is an integer, a previously defined symbol, or an expression in the range 0 to 37.

FIELD causes the PAL8 assembler to output a field setting from 0 to 37 during the second pass of assembly. This setting, which appears as the high-order bits of the location counter in the program listing, tells the ABSLDR which field to load information into.

For example, the following FIELD pseudo-op specifies memory field 26. The next line sets the location counter to begin at 400. Note that the FIELD instruction must precede the starting location.

```
FIELD 26        /CORRECT EXAMPLE
*400
```

The following example is incorrect and will not generate the code you want.

```
*400            /INCORRECT EXAMPLE
FIELD 26
```

3.2.2 **Specifying Data and Instruction Fields -- CDF and CIF** - The CDF and CIF instructions let you specify field 0 to 37 as data and instruction fields. Entering the argument requires knowledge of the bit arrangement of these two instructions.

```
                    A CDE B
CDF     6201    110 010 000 001

CIF     6202    110 010 000 010
```

Bits A CDE B indicate the data or instruction field that the program will jump to at the next indirect JMP or JMS. (The positioning of ABCDE is eccentric as ACDEB maintains compatibility between KT8A and existing 32K systems.)

To specify a field from 0 to 7, you use bits CDE only. The format of the instruction is

```
CDF or CIF n0
```

where

n0 — is an octal number that PAL8 ORs with the instruction code

n — is an octal digit from 0 to 7 (bits CDE)

For example, this instruction

```
CDF     60
```

specifies field 6 by causing PAL8 to do the following OR.

		A CDE B
Instruction code	6201	110 010 000 001
Argument	60	000 000 110 000
	6261	110 010 110 001

To specify a field from 10 to 17, use bits CDE and set bit B. The format of the instruction is

```
CDF or CIF n4
```

where

n4 is an octal number that PAL8 ORs with the instruction code

n is an octal value from 0 to 7 (bits CDE)

4 is an octal value indicating a field range of 10 to 17 (sets bit B)

For example, this instruction

```
CDF 64
```

indicates field 16.

Keep in mind that to call for fields above field 7 (above 32K) with CDF and CIF, you must first load the KT8A Extended Mode Register with the LXM instruction. For example, the following code deposits (7777 in field 12, location 1000.

```
LXM
CDF 24
TAD (7777
DCA I (1000
```

To specify a field from 20 to 27, use bits CDE and set bit A. The format is

```
CDF or CIF 1n0
```

where

1n0 is an octal number that PAL8 ORs with the instruction

1 is an octal value indicating field range 20 to 27 (sets A)

n is a value from 0 to 7 (bits CDE)

For example, this instruction

```
CDF 160
```

indicates field 26.

To specify a field from 30 to 37, use bits CDE and set bit A and B. The format is

```
CDF or CIF 1n4
```

where

1n4	is an octal number that PAL8 ORs with the instruction
1...4	are octal values indicating a field range of 30 to 37 (set bits A and B)
n	is an octal digit in the range 0 to 7 (bits CDE)

For example, this instruction

```
CDF 164
```

specifies field 36

One way to avoid confusion with this unusual bit configuration is to define high fields with convenient mnemonics. For example:

```
F36=164
CDF F36
```

3.2.3 **The ABSLDR** - The ABSLDR will load information into any field from 0 to 37 that you specify in the FIELD pseudo-op. However, the ABSLDR option /n is restricted to fields 0 to 7 only.

The =ffnnnn option sets the starting address of the program in memory to ffnnnn, where ff is a field from 0 to 37 and nnnn is a location. If you omit the option or specify 0, the ABSLDR inserts a starting address of 0200 in field 0.

3.3 Determining Memory-Size at Run-Time

It is frequently helpful to know the amount of memory currently available to the program you are running. The sub-routine in Figure 1 determines the amount of memory available in a 128K system at run-time. The program returns a value in the range of 0 to 40 to indicate the first non-existent field in the system.

To use this routine above 32K, you must first load the Extended Mode Register with the LXM instruction. For complete information on the Extended Mode Register, see the KT8A Memory Management Control User's Guide.

```
/SUBROUTINE TO DETERMINE MEMORY SIZE      PAL8-V10A 04-AUG-78

             /SUBROUTINE TO DETERMINE MEMORY SIZE

             /THIS SUBROUTINE WORKS ON ANY PDP-8 FAMILY
             /COMPUTER. THE VALUE, FROM 1 TO 40 OCTAL,
             /OF THE FIRST NON-EXISTENT MEMORY FIELD IS
             /RETURNED IN THE AC.

             /NOTE -- THIS ROUTINE MUST BE PLACED IN FIELD 0

00200  0000  CORE,   0
00201  7300          CLA CLL
00202  6201  COR0,   CDF 0             /(NEEDED FOR PDP-8L)
00203  1242          TAD     CORSIZ    /GET FIELD TO TEST
00204  0222          AND     COR37     /MASK USEFUL BITS
00205  7112          CLL RTR           /TRANSFORMS
00206  7012          RTR               /"37" TO "174"
00207  7002          BSW               /FOR CDF
00210  7430          SZL
00211  1243          TAD     C4
00212  0235          AND     COREX
00213  3214          DCA     .+1       /SET UP CDF TO FIELD
00214  6201  COR1,   CDF               /CDF IS PROCESSED HERE
00215  1640          TAD I   CORLOC    /SAVE CURRENT CONTENTS
00216  7000  COR2,   NOP               /(HACK FOR PDP-8!)
00217  3214          DCA     COR1
00220  1216          TAD     COR2      /7000 IS A "GOOD" PATTERN
00221  3640          DCA I   CORLOC
00222  0037  COR37,  37                /(HACK FOR PDP-8.,NO-OP)
00223  1640          TAD I   CORLOC    /TRY TO READ BACK 7000
00224  7400  CORX,   7400              /(HACK FOR PDP-8.,NO-OP)
00225  1224          TAD     CORX      /GUARD AGAINST "WRAP-AROUND"
00226  1241          TAD     CORV      /TAD (1400)
00227  7640          SZA CLA
00230  5235          JMP     COREX     /NON-EXISTENT FIELD EXIT
00231  1214          TAD     COR1      /RESTORE CONTENTS DESTROYED
00232  3640          DCA I   CORLOC
00233  2242          ISZ     CORSIZ    /TRY NEXT HIGHER FIELD
00234  5202          JMP     CORO
00235  6201  COREX,  CDF               /LEAVE WITH DATA FIELD 0
00236  1242          TAD     CORSIZ    /1ST NON-EXISTENT FIELD
00237  5600          JMP I   CORE
00240  0224  CORLOC, CORX              /ADDRESS TO TEST IN EACH FIELD
00241  1400  CORV,   1400              /7000+7400+1400=0
00242  0001  CORSIZ, 1                 /CURRENT FIELD TO TEST
00243  0004  C4,     4
```

Figure 1 Memory-Size Subroutine

3.4 The VXA0 Extended-Memory Device

The VXA0 device handler enables you to use the extended memory provided by the KT8A as though it were a separate device. You call VXA0 in the same way that you call any system device. For example, this command

```
.COPY VXA0:SAMPLE<RXA0:SAMPLE
```

copies a program called SAMPLE into an area of memory above 32K words.

The VXA0 device provides high speed I/O for users with diskettes or users who want the performance of a fixed-head disk type of storage device.

3.5 The SAVECB Program

SAVECB is a demonstration program that enables you to alter the contents of a program's Core Control Block. You will find this routine useful in a SAVE with arguments involving more fields in memory than you can specify in a single SAVE command line. This is likely to happen in systems with 128K words of memory, since the number of fields you may wish to specify increases from 10 to 40 (octal).

The format for summoning SAVECB is

```
R SAVECB
* file.SV
```

where

file.SV — is the name of program whose CCB you want to change

SAVECB responds with a number sign (#) to indicate that it is ready to accept one of the following commands.

TYPE	displays core control block of file.SV
Affmmmm-ffnnnn	adds segment to CCB
Sffmmmm-ffnnnn	subtracts segment from CCB

To exit from the program, type

```
#@
```

This writes the updated Core Control Block onto the system area of the device. In order to change the program's CCB, you must load the program with the R command (typing CTRL/C to abort execution) and then create a memory-image file with SAVE.

For example, assume you want to save segments of program FLOP.SV as a memory-image file called FLAP.SV. First, you modify the CCB with SAVECB.

```
.R SAVECB
* FLOP.SV
```

SAVECB responds with a number sign (#). To inspect the CCB of your program, type

```
#TYPE
```

SAVECB displays the starting location of the program, its Job Status Word, and the segments in memory that it uses.

```
START=0000    JSW=2000

CORE SEGMENTS:

040200-040377,020200-020377,016400-017377,000000-007577
```

To add segments to the CCB, enter them after the prompt.

```
#A30200-30600,40600-40777
```

Now examine the CCB again.

```
*TYPE

     START=000200      JSW=2000

CORE SEGMENTS:

     040200-040377,040600-040777,030200-030600,020200-020377
     016400-017377,000000-007577
```

To place this core control block in the system area on the device, type @ after the prompt.

```
     *@
```

To make a memory-image file of the segments specified in the CCB run FLOP.SV with the R command aborting execution with CTRL/C. Then save the segments under the new name with a SAVE command without arguments.

```
     .R FLOP.SV
     ^C
     .SAVE FLAP.SV
```

To change a segment, first subtract the entire segment with the S command. Then enter the altered version with the A command.

4.0 THE RX02 DUAL-DENSITY DISKETTE

The OS/8 V3D Device Extensions include system and non-system handlers for RX01 and RX02, the devices for single-density and double-density diskettes. The new handlers run on both RX01 and RX02 hardware.

NOTE

- The old OS/8 handlers, including BOOT/RX, will not run on RX02.
- An RX02 with a single-density hardware switch set is identical to an RX01.

4.1 RX02 Device Names

To specify an RX02 diskette in an OS/8 command line, enter the same device names you use for RX01. OS/8 recognizes the following permanent names.

DSK	Default output device, usually same as SYS
SYS	System device, usually the diskette in drive 0
RXA0	The diskette in drive 0
RXA1	The diskette in drive 1

SYS is most accurately defined as the device that you have bootstrapped. This is usually the device in drive 0. However, the hardware will also bootstrap a device in drive 1, making SYS and DSK equivalent to RXA1. The permanent names RXA0 and RXA1 remain unchanged.

4.2 Formatting Diskettes for RX02

Diskettes arrive from the factory already formatted for use in a single-density RX01 drive. To format them for RX02, use the RXCOPY program with the /D option, specifying the diskette you want to re-format as an output device. (If you enter a device by itself in the command line, RXCOPY considers it to be an output device.)

Diskettes formatted for the RX02 device contain 981 blocks (besides the directory) in a 12-bit mode.

4.3 RX01 and RX02 Compatibility

- A double-density system diskette runs only on an RX02 double-density drive. Similarly, a single-density SYS requires an RX01 drive.
- RX02 accepts both single-density and double-density non-system diskettes. The non-system handler determines which kind of device it is dealing with and proceeds accordingly.
- RX01 hardware accepts only diskettes formatted for single-density use.

NOTE

> If you place an RX02 diskette on an RX01 drive, you can currently write to it without producing an error message. Avoid this procedure, as it results in a "mixed" diskette.

Table 6 matches single-density and double-density diskettes -- both system and non-system -- with the drives that they run on.

Table 6
RX01-RX02 Compatibility

Diskette Type	Drive type	
	Single density	Double density
Single-density System	X	
Double-density System		X
Single-density Non-system	X	X
Double-density Non-system		X

4.4 Interleaving

OS/8 writes blocks on single-density diskettes with an interleave of 2. This means that it skips a block between each block that it reads or writes. With double-density diskettes, OS/8 uses an interleave scheme of 3, skipping two blocks. Tables 7 and 8 show the interleave schemes for both single- and double-density diskettes.

Table 7
OS/8 Single Density Diskette Interleave Scheme

OS/8 Logical Block (octal)	Diskette Sectors (track/sector--decimal)			
0	1/1	1/3	1/5	1/7
1	1/9	1/11	1/13	1/15
2	1/17	1/19	1/21	1/23
3	1/25	1/2	1/4	1/6
4	1/8	1/10	1/12	1/14
5	1/16	1/18	1/20	1/22
6	1/24	1/26	2/1	2/3
7	2/5	2/7	2/9	2/11
10	2/13	2/15	2/17	2/19
11	2/21	2/23	2/25	2/2
12	2/4	2/6	2/8	2/10
13	2/12	2/14	2/16	2/18
14	2/20	2/22	2/24	2/26
15	3/1	3/3	3/5	3/7
.				
.				
.				

Table 8
OS/8 Double Density Diskette Interleave Scheme

OS/8 Logical Block (octal)	Diskette Sectors (Track/sector--decimal)	
0	1/1	1/4
1	1/7	1/10
2	1/13	1/15
3	1/19	1/22
4	1/25	1/2
5	1/5	1/8
6	1/11	1/14
7	1/17	1/20
8	1/23	1/26
9	1/3	1/6
10	1/9	1/12
11	1/15	1/18
12	1/21	1/24
13	2/1	2/4
.		
.		
.		

OS/8 does not use Track 0, and you cannot access it in the 12-bit mode.

4.5 Using RXCOPY with RX02

RXCOPY copies both single-density and double-density diskettes on RX02 drives. If the output diskette does not match the input diskette, RXCOPY will re-format it to the proper density.

In default mode, RXCOPY compares the two diskettes for identical contents before it makes a copy. For a quicker transfer, use the /N option, which inhibits the comparison.

For double-density transfers involving a comparison of contents, RXCOPY will use 16K words of memory if it is available on the system for faster operation. If possible, use the MEMORY command to provide the necessary memory.

4.5.1 **Formatting Diskettes with RXCOPY** - RXCOPY with the /S and /D option formats diskettes for single-density or double-density use. To format a diskette, enter it by itself in the command line, followed by the option. (RXCOPY considers a device entered by itself to be an output device.)

For example, the following command sequence re-formats the diskette in drive 1 from single-density to double-density.

```
.R RXCOPY
*RXA1:/D
.
```

To change it back to single-density, type

```
.R RXCOPY
.RXA1:/S
.
```

4.5.2 **RXCOPY Options** - RXCOPY provides the following options.

- /P RXCOPY pauses and waits for user response before and after transfers. To continue, type Y.
- /N RXCOPY transfers the contents of one diskette to another but does not check for identical contents.
- /M RXCOPY checks both diskettes for identical contents and lists the areas that do not match but performs no transfer.
- /R RXCOPY reads every block on the input device and lists bad sectors but performs no transfer.
- /V RXCOPY prints its current version number.
- /S RXCOPY formats the diskette specified as an output device to single-density.
- /D RXCOPY formats the diskette specified for output to double-density.
- /C This option is equivalent to default copy and match.

5.0 THE RL01 DISK

This section describes the booting, formatting and building of the RL01 disk pack with the OS/8 Operating System, using the RL01 OS/8 software support package.

The RL01 disk pack -- a high-density mass storage device -- utilizes bad-block mapping. Bad blocks occur during the manufacture of disks or develop as a result of use and age. Bad blocks that are present after manufacture are recorded in factory-written lists. Each disk preserves its own individual list. The RLFRMT formatter program detects and lists new bad blocks that occur during disk operation in the field. Each RL01 disk maintains up to 45 bad blocks; this allows the life of the disk to be prolonged as a mass storage device.

The RL01 requires a PDP-8A,E,F or M with at least 12K of memory. Non-omnibus PDP-8 family computers are not hardware-compatible with the RL01.

System and non-system RL01 handlers are standard two-page OS/8 handlers. Two-page handlers require 12K of memory because the second page of the handler resides in the last page of field 2.

BATCH may be run using RL01 disks, even on a system with 12K words of memory. However, disk formatting cannot be done under BATCH.

This section includes a system description, detailing disk, RL8A controller and software formats. In addition, it contains bootstrap procedures and operating instructions, including a detailed presentation of messages that are generated during disk formatting.

5.1 System Description

The RL01 disk pack has three logical "devices" that are designated as Device A, Device B and Device C. Figure 2-1 shows device designation on an RL01 disk.

The OS/8 Device Extensions provide for the standard OS/8 System and Non-System I/O transfer of 1 to 32 (decimal) memory pages to or from any one of three RL01 "devices". The "devices" are located on any one of four RL01 disk drives.

Disk data-space consists of 777 (octal) tracks. As shown in Figure 2, data on a single track is made up of 40 equal length sectors numbered 0 through 47 (octal). This results in 20 (decimal) blocks, four assigned to Device C and 16 to Device A or B.

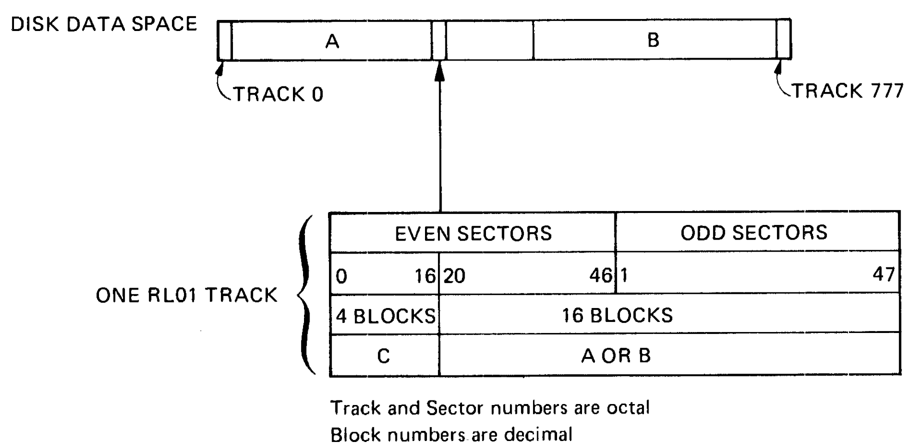


Figure 2 Devices A, B, C on RL01 Disk

Approximately 10,000 (decimal) OS/8 blocks are supported per drive, 40% as Device A, 40% as Device B, and 20% as Device C. This scheme provides some user control over the tradeoff between the number of devices and the length of each device.

Device C has a different length from Devices A and B. In general, Device C is used only when a maximum amount of data is to be stored on the disk.

Each device supports up to 15 (decimal) bad blocks to provide bad-block mapping. These blocks may be thought of as "spares", and should never be accessed by the user. This support involves "invisible" mapping of OS/8 block numbers into the set of actual good disk blocks; no utility program need be changed (including SQUISH), and user awareness of this feature is not required. Bad-block lists are kept resident to reduce the extra reads required.

Bad-block lists that occur during disk manufacture are maintained in factory-generated lists which are stored on track 777 of the disk. The OS/8 system preserves five copies of the factory list, all of which are identical.

When a disk is initially formatted using OS/8, the formatting program (RLFRMT) ascertains that the disk is new. The program then reads in the factory list, and checks the disk for any new bad blocks. The factory list and the new bad-block list are then combined, and, after the user's go-ahead, the formatting program writes the newly-generated OS/8 bad-block list on track 0 of the disk.

When running OS/8, you may generate an I/O error because of a bad block. You can check this by again running the formatter program. RLFRMT ascertains that the disk is already formatted, so it reads in the previous OS/8 bad-block list and checks the disk for any new bad blocks. When you instruct it to proceed, the formatting program writes the updated OS/8 bad-block list on track 0 of the disk. You should not allow the bad block list to be written if an unexpectedly large number of bad blocks are reported; formatting to remove bad blocks is a permanent, irreversible procedure.

During an I/O transfer, the handler first reads in the OS/8 bad-block list for the device. The system effectively maps around the bad blocks. This has the effect of making them appear to have disappeared, so that standard OS/8 block numbers can be used.

All permanent information stored on RL01 disk packs (such as bad-block lists) is protected from destruction by OS/8 handler calls by being "outside of" the OS/8 data space.

An annulus data scheme reduces the average intra-device seek time. This means that data continues from the track on surface 0 to the track on surface 1 for each cylinder.

The bootstrap routine is under 32 (decimal) words in length, and suitable for ROM implementation and/or direct toggle-in.

Three tries (two retries) are attempted before an I/O error is reported.

NOTE

Unless otherwise noted, all numbers in this section are octal.

5.1.1 **Disk Format** - The format of the RL01 disk is as follows:

Track	Sector	Contents
0	0	Reserved for future use by DIGITAL
0	2	Reserved for future use by DIGITAL
0	4	Reserved for future use by DIGITAL
0	6	Reserved for future use by DIGITAL
0	10	Reserved for future use by DIGITAL
0	12	Reserved for future use by DIGITAL
0	14	Bad Block Lists for Devices A and B
0	16	Bad Block List for Device C
0	20	Device A, Block 0 (first half)
0	22	Device A, Block 0 (second half)
.	.	.
.	.	.
.	.	.
1	0	Device C, Block 0 (first half)
1	2	Device C, Block 0 (second half)
.	.	.
.	.	.
.	.	.
400	20	Device B, Block 0 (first half)
400	22	Device B, Block 0 (second half)
.	.	.
.	.	.
.	.	.
777	0	Disk Pack Serial Number, List of Manufacturing-Detected Bad Sectors and Field-Detected Bad Sectors.

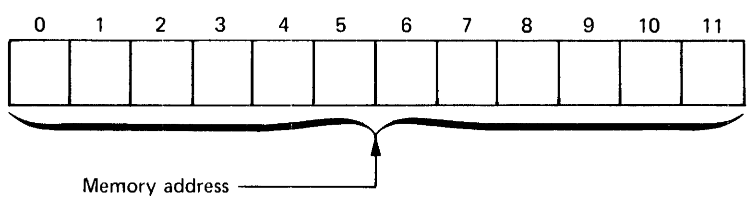
NOTE

RLFRMT.PA contains complete descriptions of bad block list formats as comments at the start of the program.

5.1.2 **RL8A Controller Format** - The following registers perform software control of the system.

Memory Address Register:

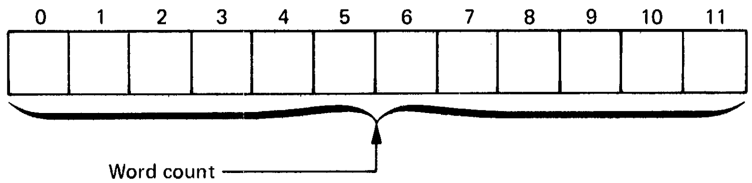
The Memory Address Register is a 12-bit register that contains the location at which the first transfer is to be performed.



Memory Address Register

Word Count Register:

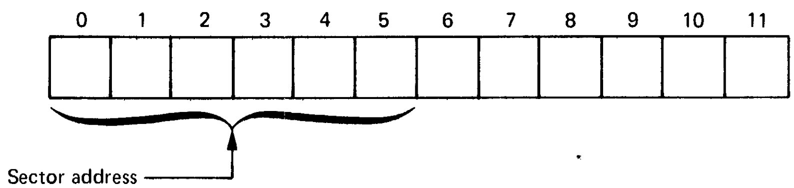
The Word Count Register is a 12-bit register that contains the negative of the number of words to be transferred at one time.



Word Count Register

Sector Address Register:

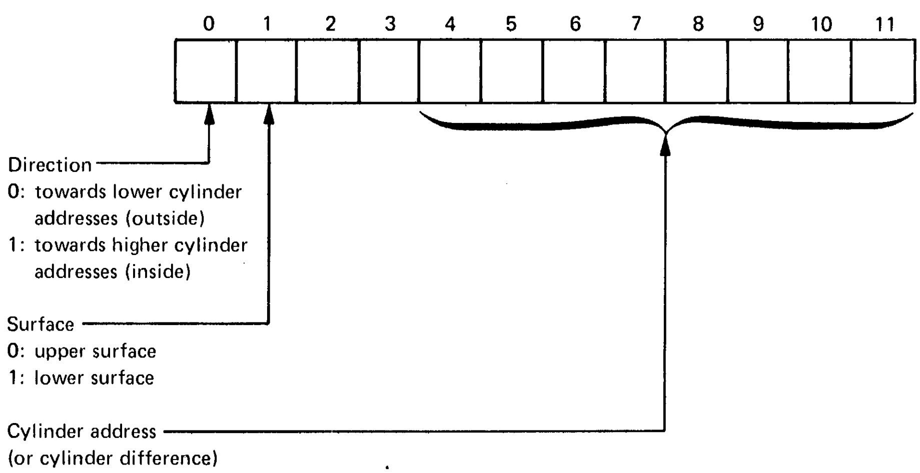
The Sector Address Register contains the sector address in bits 0 through 5.



Sector Address Register

Command Register A:

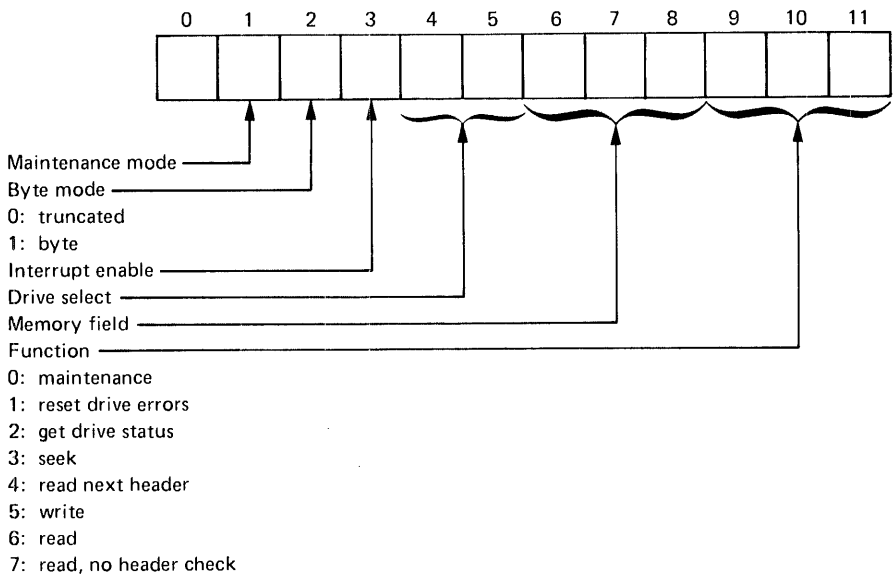
Command Register A contains the direction, surface and cylinder address. It has the following format:



Command Register A

Command Register B:

Command Register B designates maintenance mode, byte mode, interrupt enable, drive select, memory field and function. It has the following format.



Command Register B

5.1.3 **Instruction Set** - The following instructions operate the disk system.

Note that the AC is cleared after a transfer from the AC to a register in the controller. Also, the AC is cleared first before a transfer is made from a controller register to the AC.

The skip instructions are skip and then clear IOT's; that is, if a given condition (function done) is true, the function-done flag will be cleared at the completion of the skip IOT.

Octal Code	Mnemonic	Function
6600	RLDC	Clear device, all registers, AC and flags (do not use to terminate a disk function)
6601	RLSD	Skip on function done flag, then clear it
6602	RLMA	Load memory address register from AC
6603	RLCA	Load command register "A" from AC
6604	RLCB	Load command register "B" from AC, execute command
6605	RLSA	Load sector address register from AC bits 0-5
6606	----	Spare (will clear the AC)
6607	RLWC	Load word count register from AC
6610	RRER	Read error register into AC bits 0-2 and 11
6611	RRWC	Read word count register into AC
6612	RRCA	Read command register "A" into AC

Octal Code	Mnemonic	Function
6613	RRCB	Read command register "B" into AC
6614	RRSA	Read sector address register into AC bits 0-5
6615	RRSI	Read (silo) word (8-bit) into AC bits 4-11
6616	----	Spare (does not clear AC)
6617	RLSE	Skip on composite error flag, then clear it

5.1.4 **OS/8 Data Space** - The layout of OS/8 data space on Devices A, B, and C is as follows:

Devices A and B

Block	Track	Sectors
0	0	20,22
1	0	24,26
2	0	30,32
3	0	34,36
4	0	40,42
5	0	44,46
6	0	1,3
7	0	5,7
10	0	11,13
11	0	15,17
12	0	21,23
13	0	25,27
14	0	31,33
15	0	35,37
16	0	41,43
17	0	45,47
20	1	20,22
.	.	.
.	.	.
.	.	.

Device C

Block	Track	Sectors
0	1	0,2
1	1	4,6
2	1	10,12
3	1	14,16
4	2	0,2
.	.	.
.	.	.
.	.	.

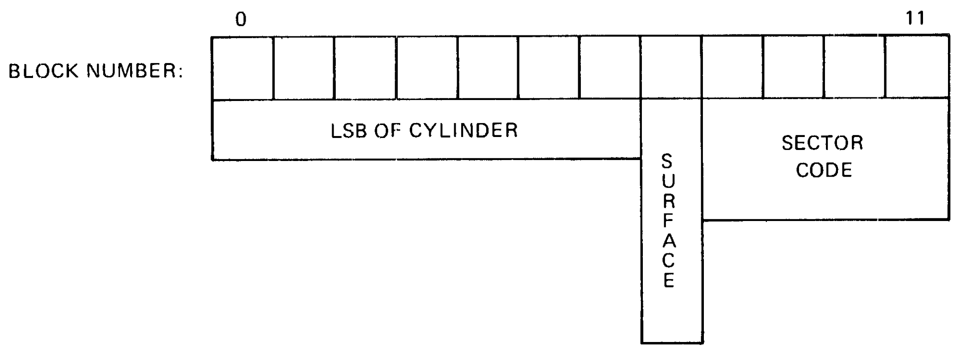
5.1.5 **Converting Block Numbers to Hardware Disk Addresses** - Use the following procedures.

For Devices A and B:

The sector address is 4 times the sector code minus 27. If the sector address is negative, add 47.

Device A has MSB of cylinder = 0 (cylinders 0-177).
Device B has MSB of cylinder = 1 (cylinders 200-377).

The block number software format for Devices A and B is shown in the following diagram.

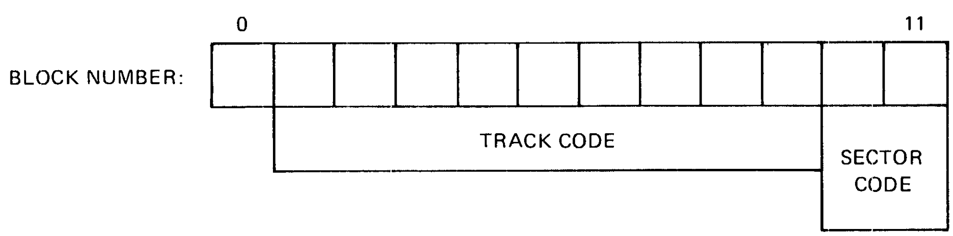


Block Number Format for Devices A and B

For Device C:

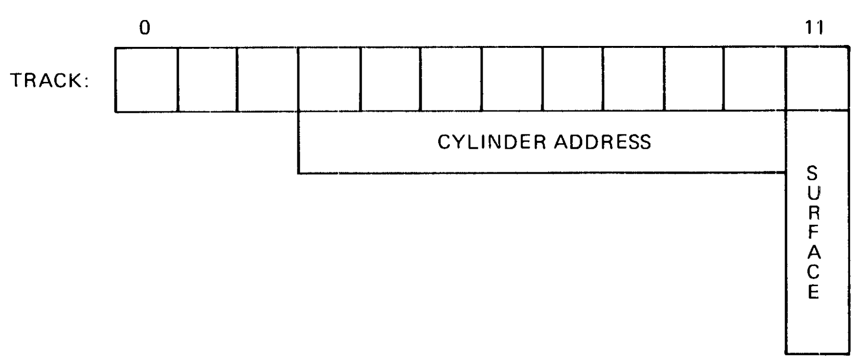
The sector address is 4 times the sector code. The track is one plus the track code. Tracks 0 and 777 cannot be addressed; this ensures the integrity of the factory-detected and OS/8 bad-block lists, which reside on these tracks.

The block number software format for Device C is shown in the following diagram.



Block Number Format for Device C

The track software format for Devices A, B, and C is shown in the following diagram.



Track Format for Devices A, B, and C

5.2 Handler Description

The standard OS/8 device designation format cannot be used with the RL01. Normally, the standard format would use "RLA0: to represent Device A of unit (drive) 0. The single-word format used internally to store device names does not distinguish between "RK" devices and "RL" devices, resulting in erroneous RESORC reports, and in other anomalies. The RL01 therefore uses "RL0A" to represent unit 0, Device A, and so forth. Table 9 provides information on the RL01 handlers. SYS is the same device as RL0A (Drive 0, Device A).

Table 9
RL01 Handler Information

Device Name	Entry Point Offset (Octal)	File Name (Group Name)	Device Type	Device Code (Octal)	Octal Length (Blocks)	Decimal Length (Blocks)
SYS	07	RLSY	RL01	26	7761	4081
RL0A	44	RL0	RL01	26	7761	4081
RL0B	40	RL0	RL01	26	7761	4081
RL0C	50	RLC	RL01	31	3751	2025
RL1A	45	RL1	RL01	26	7761	4081
RL1B	41	RL1	RL01	26	7761	4081
RL1C	54	RLC	RL01	31	3751	2025
RL2A	46	RL2	RL01	26	7761	4081
RL2B	42	RL2	RL01	26	7761	4081
RL2C	60	RLC	RL01	31	3751	2025
RL3A	47	RL3	RL01	26	7761	4081
RL3B	43	RL3	RL01	26	7761	4081
RL3C	64	RLC	RL01	31	3751	2025

A brief description of RL01 handler operation is as follows:

1. When initially called, each RL01 handler executes once-only code to read in the bad block list for its drive. The handler error return is taken (with AC=4000) if an I/O error occurs or if the bad-block list is found to be invalid (a valid bad-block list begins with a special identification code).

2. Get handler arguments.

3. Map each block to be transferred around bad blocks by incrementing the block number once for each bad block (as listed in the bad-block list for the requested device) less than or equal to the present block. This procedure makes bad blocks effectively "disappear."

4. Transfer one page/sector at a time, up to the requested number of pages.

5. If an I/O error occurs for any RL01 read or write operation, retry twice then take the System or Non-System Handler error return with AC=4000.

5.3 Loading and Bootstrap Procedure

The following sequence of operations occurs during bootstrapping to the RL01.

1. BOOT-1, the primary bootstrap routine, is read into locations 00001-00035 from a ROM, from BOOT.SV, or toggled in through the console switches. The starting address is 00001. BOOT-1 clears Drive 0 and reads and starts BOOT-2. If an I/O error occurs, BOOT-1 will repeat until it is successful.

2. BOOT-2 occupies locations 00000-00177. BOOT-2 reads the OS/8 Resident Monitor into the last pages of fields 0, 1, and 2. If an I/O error occurs, BOOT-2 will "hang" as an indication of failure to boot.

3. BOOT-2 then calls the Keyboard Monitor by jumping to location 07605.

NOTE

> Never replace the system disk pack without rebooting; each pack has its own OS/8 block numbering scheme that is determined during formatting.

Replace non-system disk packs only after the Monitor dot appears on the terminal. This is done to ensure that the bad-block list read by the handler is correct.

5.3.1 Loading the RL01 Disk Pack - Prepare an RL01 Disk Pack for loading as follows:

1. Separate the protective cover from the disk pack, using the following steps.

   a. Lift the cartridge by grasping the handle with the right hand.

   b. Support the cartridge from underneath with the left hand.

   c. Lower the handle and push the handle slide to the left with the thumb of the right hand.

   d. Raise the handle to its upright position to separate the cartridge from the protection cover.

2. Place the cartridge in the drive shroud with the handle recess facing the rear of the machine.

3. Rotate the cartridge a few degrees clockwise and counter-clockwise to ensure that it is properly seated within the shroud.

4. Gently lower the handle to a horizontal position to engage the drive spindle.

5. Place the protection cover on top of the cartridge.

6. Carefully close the drive lid.

7. Push the "LOAD/RUN" pushbutton.

5.3.2 **Booting from BOOT.SV** - Boot from the BOOT.SV program by using the BOOT or R commands as follows:

```
.BOOT
/RL
```

or

```
.BOOT RL
```

or

```
.R BOOT
/RL
```

5.3.3 **Booting from the Console Switches** - The following procedure enters the bootstrap program into PDP-8/A memory.

1. Press in order the MD and DISP buttons to see what octal numbers are being deposited.

2. Press, in order, 0 and LXA to select memory field 0.

3. Press, in order, 1 and LA to start loading instructions at address 1.

4. Deposit the octal values given in Table 3, following each value with D NEXT.

5. After all values are deposited, press, in order, 0001 and LA to allow the program to start at location 1.

6. Press, in order, INIT and RUN to start the bootstrap program.

5.3.4 **ROM Bootstrap Switch Settings** - Set the bootstrap switch settings for ROM's labeled 465A2 and 469A2 as follows:

Program	S2-5	S2-6	S2-7	S2-8	S1-1	S1-2	S1-3	Memory Address
RL8A	OFF	ON	OFF	OFF	OFF	ON	OFF	4000

5.4 Operating Instructions

You must format new RL01 disk packs by running the RLFRMT program prior to any OS/8 use (including system building). This is required because RLFRMT constructs and writes specially formatted bad-block lists on the pack.

OS/8 RL01 operations on disks that have not been formatted with RLFRMT result in error reports. Therefore, you should run RLFRMT even before using BUILD to build a new system head.

Device C non-system handlers are provided to access all available storage capacity of the RL01 disk packs. Transfers to or from Device C are slower than those to or from Devices A and B. This is so because, while Device A and B use 80% of each track, Device C only uses 20% of each track (only 4 blocks are stored on each track). Thus, the time spent in seeking new tracks will be higher for Device C.

Because different RL01 packs may have different patterns of bad blocks, it is good practice to end an OS/8 session with the monitor "BOOT" command (or "R BOOT"), so that other users will be able to type "RL" to boot their disks. Of course, this procedure is unnecessary if the computer system has a hardware bootstrap for the RL01.

5.4.1 **Disk Formatting** - Format all new RL01 disk packs prior to any use under OS/8, including system building. Mount the RL01 disk pack (Section 4.2). Format the RL01 disk by using the following procedure:

1. Type

   .R RLFRMT

   to run the formatter program.

   RLFRMT Vvp is printed on the terminal signifying the start of the operation where:

   v is the version number
   p is the patch level letter

   the program then prompts with

   DRIVE?

2. Type the drive number (0-3) on which the pack is mounted.

   The formatter program then reads all blocks on the disk to detect any new bad blocks. The process takes 35 to 40 seconds. After this period, an initial display is presented as follows:

   ```
   UNFORMATTED (NEW) DISK PACK SERIAL NUMBER nnnnnnnnn
   FACTORY-DETECTED BAD BLOCKS: NONE
   NEWLY-FOUND BAD BLOCKS: NONE
   NEW OS/8 BAD BLOCKS: NONE
   FORMAT PACK WITH THIS NEW LIST?
   ```

   The messages are explained in Table 10, RLFRMT Formatter Messages.

3. Type a "Y" or "N" (followed by a RETurn) in response to the last message "FORMAT PACK WITH THIS NEW LIST?" to either allow or prevent the writing of the new OS/8 bad-block lists. The program signifies completion of this operation by displaying

   DONE
   DRIVE ?

   Type CTRL/C to return to the OS/8 monitor. Remove the pack or designate another drive for formatting.

The following example illustrates possible messages that may be generated during a particular sequence of OS/8 RL01 formatting operations if bad blocks are found.

```
OS/8 (OLD) DISK.
WARNING: ALL FACTORY-WRITTEN LISTS DESTROYED.

PREVIOUS OS/8 BAD BLOCKS: NONE
NEWLY-FOUND BAD BLOCKS: A 6374    A  6375    B  0360
B 4347    B 4350    C0514    C 0515    C 2073

WARNING: AN ADDITIONAL BAD BLOCK FOUND.
ZERO DISK BEFORE USE!

NEW OS/8 BAD BLOCKS: A 6374    A 6375    B 0360
B 4347    B 4350    C 0514    C 0515    C 2073

FORMAT PACK WITH THIS NEW LIST?
```

The formatter program then writes or does not write special OS/8 bad block lists on the pack, depending on a "Y" or "N" user response. These lists include only the factory-detected and newly-detected bad blocks for new packs, or previous OS/8 and newly-detected bad blocks for old packs. Warnings are given for various conditions as appropriate (see Table 10).

NOTE

Reformatting a previously-used disk pack will make any newly-detected bad blocks effectively disappear from the pack. Any files located at or after any such new bad blocks, however, will turn to garbage due to the implicit renumbering of all OS/8 blocks past those points.

Table 10 lists normal formatter messages, operator error messages, and program error messages.

Table 10
RLFRMT Formatter Messages

1. Normal Messages	
Message	Meaning
RLFRMT Vvp	Identifies start of operations. "v" is version number, "p" is patch level letter.
DRIVE ?	Requests user to type drive number and RETURN key.
UNFORMATTED (NEW) DISK PACK	Disk does not contain OS/8 bad block lists, either because the disk is brand new or because these lists have been destroyed by non-DIGITAL software or diagnostic programs.

(continued on next page)

Table 10 (Cont.)
RLFRMT Formatter Messages

1. Normal Messages (Cont.)

Message	Meaning
OS/8 (OLD) DISK PACK	Disk contains valid OS/8 bad block lists. (A formatted pack contains octal 0123 in words 100 - 177 of sector 16 (octal) of surface 0 of cylinder 0).
SERIAL NUMBER xxxxxxxxxx	The serial number is the ten digit octal number assigned to the pack at time of manufacture.
FACTORY-DETECTED BAD BLOCKS	The list of bad blocks found at manufacturing time is printed in the format "d nnnn", where d=A,B, or C (the device) and nnnn = the block number on that device which is bad.
PREVIOUS OS/8 BAD BLOCKS	The current OS/8 bad block lists are printed.
NEWLY-FOUND BAD BLOCKS	The list of bad blocks just found by read-checking the entire disk is printed.
NEW OS/8 BAD BLOCKS	This list results from combining the previously printed lists. It is the list that is written on the pack as the new OS/8 bad block lists.
FORMAT PACK WITH THIS NEW LIST?	User types "Y" or "N" to allow or prevent writing the new OS/8 bad block lists.
DONE	Indicates that new OS/8 bad block lists have been written on the pack. The pack now may be removed if desired. "DONE" is always followed by "DRIVE?" to allow formatting another pack.

2. Operator Error Messages

Message	Meaning
PLEASE SPECIFY DRIVE NUMBER (0-3) ON WHICH PACK TO BE FORMATTED IS MOUNTED.	RLFRMT could not interpret user response to "DRIVE?". User can try again.
PLEASE WRITE-ENABLE DRIVE, THEN HIT RETURN!	RLFRMT found the selected drive write-locked just before attempting to write new OS/8 bad block lists on the pack.

(continued on next page)

Table 10 (Cont.)
RLFRMT Formatter Messages

3. Warning Messages (formatting can still be done)	
Message	**Meaning**
WARNING: AN ADDITIONAL BAD BLOCK FOUND. ZERO DISK BEFORE USE!	If the user permits the new OS/8 bad block lists to be written, the OS/8 block numbering scheme will be changed due to a new bad block found during the read-check of the entire pack. This will make "garbage" out of any files located at and after the bad block number.
WARNING: BAD BLOCK IN SYSTEM AREA. DO NOT USE AS SYSTEM DISK!	A new bad block was found during the read-check of the pack. This new bad block was on Device A between 0 and 66 inclusive. Since no bad blocks are allowed in this area for the system device (due to bootstrapping constraints), permitting the pack to be formatted disallows future use as a system device. Non-system use is unaffected.
WARNING: ALL FACTORY-WRITTEN LISTS DESTROYED	All copies of the manufacturing-detected bad block list and disk pack serial number have been destroyed by non-Digital software. Formatting continues, assuming no factory-detected bad blocks.

4. Error Messages (formatting cannot be done)	
Message	**Meaning**
FATAL I/O ERROR	If this message appears immediately, it indicates that the OS/8 bad block lists contain physical I/O errors. The pack should not be used further under OS/8. If this message appears after attempting to write new OS/8 bad block lists, it indicates that an I/O error occurred. The most common cause will be a write-locked drive.

(continued on next page)

Table 10 (Cont.)
RLFRMT Formatter Messages

4. Error Message (formatting cannot be done) (Cont.)	
Message	Meaning
CANNOT FORMAT DISK	All error messages end with this one, to indicate that the formatting operation has failed. This message is always followed by "DRIVE?" to allow formatting another pack.
OVER 15 BAD BLOCKS ON ONE DEVICE	The new OS/8 bad block lists to be written contain more than the maximum number of bad blocks supported under OS/8.
OVER 63 NEWLY-FOUND BAD BLOCKS	Indicates RL01 hardware problem detected during read-check of disk or a pack with more than 63 bad blocks. RL01 diagnostics should be run and the drive and/or controller fixed before attempting to format disk packs.

5.5 System Building

The following procedure is used for building a system.

1. Format the disk pack as described in Section 5.4.1.

2. Run BUILD from any device. BUILD is the system generation program for OS/8 (see the OS/8 Handbook for a detailed description of BUILD).

3. Load RLSY.BN, RL0.BN, RL1.BN, RL2.BN, RL3.BN, or RLC.BN as desired (see Table 9 for names of devices in each group). For example, a complete system for two disk drives would include SYS, RL0B, RL0C, RL1A, RL1B, and RL1C. A partial system to support all four drives could include SYS, RL0A, RL0B, RL1A, RL1B, RL2A, RL2B, RL3A, and RL3B.

4. Issue the BOOT(strap) command. This will build an RL01 system on RL0A, and start it. It then asks a question as to whether a new (zero) directory should be written on the new device. Answer yes to place a zero directory on the device. RUN all programs with the RUN command until moved to the RL01 disk pack.

APPENDIX A

RX02 BOOTSTRAP PROGRAM

```
                                   PAL8-V10A NO DATE

        7301  AC1=CLL CLA IAC
        7326  AC2=CLL CLA CML RTL
        7327  AC6=CLL CLA CML IAC RTL        /RX02'S MUST RUN ON AN OMNI-BUS !!
        7330  AC4000=CLL CLA CML RAR
        7350  AC3777=CLL CLA CMA RAR
        7346  AC7775=CLL CLA CMA RTL
              /
              /  DEVICE IOT SYMBOLIC EQUATES
              /
        6751  LCD=6751                   /LOAD COMMAND
        6752  XDR=6752                   /TRANSFER DATA
        6753  STR=6753                   /SKIP IF READY TO TRANSFER
        6754  SER=6754                   /SKIP ON ERROR
        6755  SDN=6755                   /SKIP ON DONE
              /
              /
              /
        0020          *20
              /
00020   1061  READ,   TAD     UNIT       /TRY NEXT COMBINATION OF DENSITY AND UNIT
00021   1046          TAD     CON360     /ADDING IN 360
00022   0060          AND     CON420     /KEEPING ONLY 420 BITS
00023   3061          DCA     UNIT       /CYCLES 400,420,0,20,400,,,,,,,,
00024   7327          AC6                /COMMAND TO READ DISK
00025   1061          TAD     UNIT       /UNIT AND DENSITY
00026   6751          LCD                /COMMAND TO CONTROLLER
00027   7301          AC1                /TO SET SECTOR AND TRACK TO 1
00030   4053          JMS     LOAD       /SECTOR TO CONTROLLER, LEAVES AC ALONE
00031   4053          JMS     LOAD       /AND TRACK
00032   7004  LITRAL, 7004               /LEAVING A 2 IN AC; SERVES AS LITERAL
              /
              /  FOLLOWING IS PART OF WAIT LOOP, SAME SECONDARY BOOTS, OLD PRIMARY BOOT
              /
00033   6755  START,  SDN                /HAS DONE COME UP; CODE STARTS HERE!
00034   5054          JMP     LOAD+1     /NO, GO CHECK FOR READY TO TRANSFER
              /
              /  NOW, DONE OR ERROR
              /
00035   6754          SER                /SKIP ON AN ERROR, TRY ANOTHER DENSITY ETC.
00036   7450          SNA                /NASTY, AC=2 FOR ABOUT TO DO SILO, 0 ON START-UP
00037   5020          JMP     READ       /START-UP, GO SET UP UNIT, THEN READ TO SILO
00040   1061          TAD     UNIT       /AC ALREADY 2, PUT IN UNIT, DENSITY
00041   6751          LCD                /TO EMPTY THE SILO
00042   1061          TAD     UNIT       /SET UP LOC 60 FOR OLD SECONDARY BOOT
00043   0046          AND     CON360     /KEEPING ONLY DENSITY BIT
00044   1032          TAD     LITRAL     /ADDING IN 7004, BECAUSE THAT'S WHAT SYS WANTS
00045   3060          DCA     RX1SAV     /OLD SECONDARY BOOT MOVES IT TO HANDLER
00046   0360  CON360, 360                /LITERAL; EXECUTES IN LINE AS A NO-OP
              /                          /FALLS THRU TO NEXT PAGE OF LISTING
              /
              /
              /  FOLLOWING CODE SAME AS OLD PRIMARY BOOT
              /
00047   4053          JMS     LOAD       /GRAB NEXT ITEM FROM SILO
00050   3002          DCA     2          /TRADITION; SECONDARY BOOT STARTS LOADING AT 2 !
00051   2050          ISZ     50         /INCREMENT LOAD ADDRESS
00052   5047          JMP     47         /GO BACK FOR ANOTHER
              /
              /  SECONDARY BOOT LOADS OVER PRIMARY BOOT UNIT LOCATION 47 IS LOADED,
              /  THEN CONTROL PASSES TO SECONDARY BOOT
              /
```

```
00053  0000  LOAD,   0                /SUBROUTINE TO GIVE AND TAKE DATA FROM CONTROLLER
00054  6753          STR              /IS HE READY TO TALK TO US?
00055  5033          JMP     START    /NO, IS HE PERHAPS DONE WITH SILO, OR IN ERROR?
00056  6752          XDR              /YES, DATA IN OR OUT;IF DATA TO CONTROLLER, AC UNCHANGED
00057  5453          JMP I   LOAD     /NO MAGIC, JUST EXIT FROM SUBROUTINE
             /
             /  60 GOES TO OLD SECONDARY BOOT
             /  61 HAS DENSITY AND UNIT THAT BOOTED SUCCESSFULLY
             /
             /
             CON420,                  /USE IT TO HOLD 420 LITERAL TO START OUT
00060  0420  RX1SAV, 420              /UNIT^20+7004 TO GO TO SYS HANDLER
00061  0020  UNIT,   20               /<DENSITY^400>+<UNIT^20> THAT BOOTED OK
             /
             $
```

APPENDIX B

RL01 BOOTSTRAP PROGRAM

```
                  AC0001=CLA IAC
                  AC0003=CLA CLL CML IAC RAL
                  AC0006=CLA CLL CML IAC RTL
                  AC2000=CLA CLL CML RTR

00001   6600      BOOT,   RLDC            /CLEAR CONTROLLER REGISTERS
00002   7201              AC0001          /CLEAR DRIVE REGISTERS
00003   4027              JMS IO
00004   1004              TAD             /AC=1004 (BYTE MODE READ HEADER
                                          /FUNCTION). NOTE THAT THIS WORD
                                          /MUST BE AT LOC 0004!
00005   4027              JMS IO          /READ NEXT HEADER IN ORDER TO
                                          /FIND OUT CURRENT CYLINDER
00006   6615              RRSI            /READ HEADER BYTE #1
00007   7002              BSW             /GET LSB OF CYLINDER
00010   7012              RTR
00011   6615              RRSI            /READ HEADER BYTE #2
00012   0025              AND C377
00013   7004              RAL             /CONSTRUCT CYLINDER ADDRESS
00014   6603              RLCA            /USE IT AS DIFFERENCE WORD TO
                                          /SEEK TO CYLINDER 0, SURFACE 0
00015   7325              AC0003          /AC=SEEK FUNCTION
00016   4027              JMS IO          /SEEK TO TRACK 0
00017   7332              AC2000          /AC=SECTOR 20 (OS8 BLOCK 0)
00020   6605              RLSA            /LOAD SECTOR ADDRESS
00021   1026              TAD C7600
00022   6607              RLWC            /LOAD WORD COUNT FOR 1 PAGE
00023   7327              AC0006          /READ FUNCTION
00024   4027              JMS IO          /READ SECONDARY BOOTSTRAP
                  /READING IN SECONDARY BOOTSTRAP PREVENTS "IO" FROM
                  /RETURNING. CONTROL CONTINUES IN SECONDARY BOOTSTRAP.

00025   0377      C377,   377
00026   7600      C7600,  7600

                  /SUBROUTINE TO DO I/O TO DISK
00027   0000      IO,     0
00030   6604              RLCB            /EXECUTE THE FUNCTION
00031   6601              RLSD            /WAIT UNTIL DONE
                                          /NOTE: THIS WORD AND NEXT
                                          /ONE MUST BE LOCATED HERE
                                          /IN ORDER TO MATCH UP WITH
                                          /SIMILAR INSTRUCTIONS CON-
                                          /TAINED IN THE SECONDARY
                                          /BOOTSTRAP.
00032   5031              JMP .-1
00033   6617              RLSE
00034   5427              JMP I IO        /NO ERROR; RETURN
00035   5001              JMP BOOT        /ERROR: TRY AGAIN
```

INDEX

OS/8
Device Extensions
User's Guide
AA-D319A-TA

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- [ ] Assembly language programmer
- [ ] Higher-level language programmer
- [ ] Occasional programmer (experienced)
- [ ] User with little programming experience
- [ ] Student programmer
- [ ] Other (please specify)________

Name________ Date________

Organization________

Street________

City________ State________ Zip Code or Country________

Please cut along this line.

Do Not Tear - Fold Here and Tape

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

RT/C SOFTWARE PUBLICATIONS ML 5-5/E45
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MASSACHUSETTS 01754

Do Not Tear - Fold Here

Cut Along Dotted Line