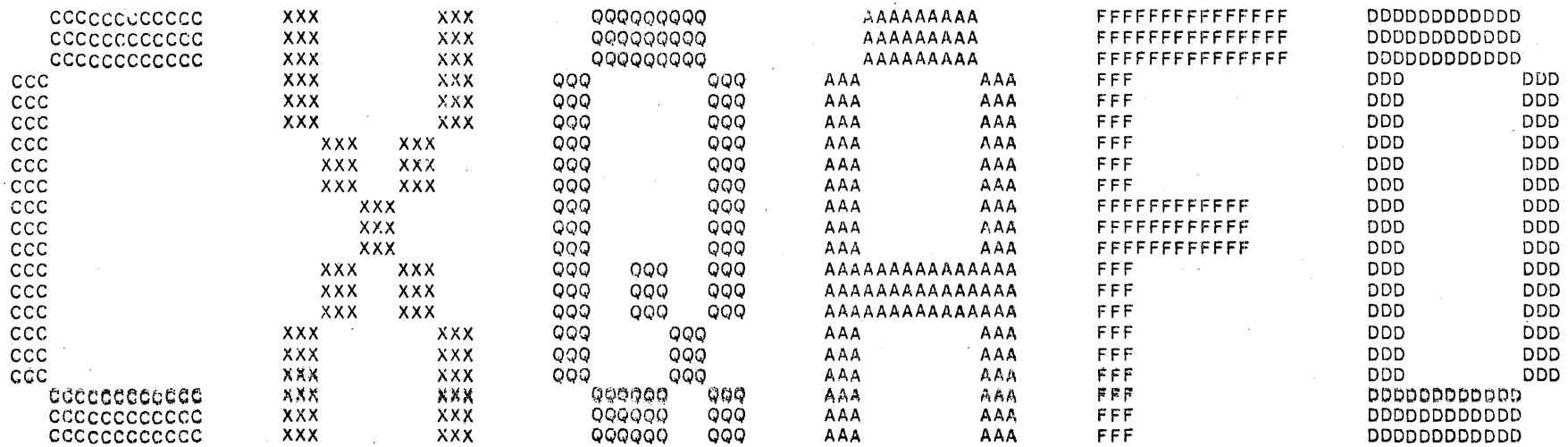

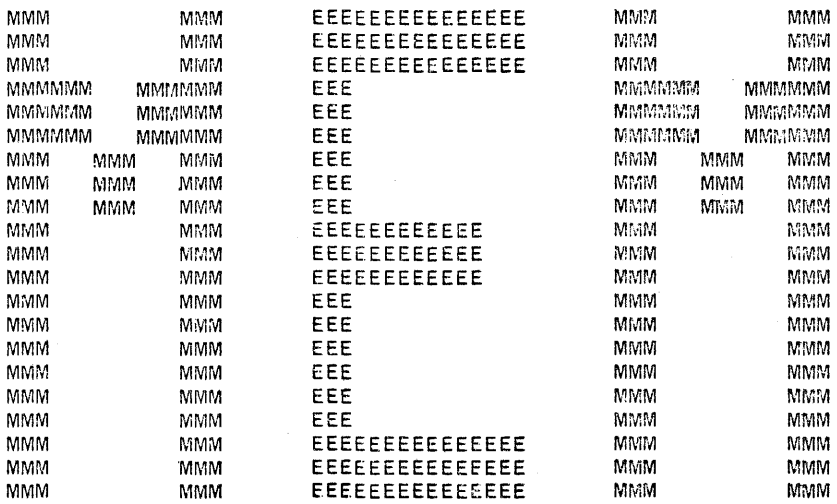


```
XERSPL version 102(2263)/3(61) running on MTA065
 *START* User BROWN,DICK [400,3114]  Job  CXQUAA Seq. 7057 Date 03-Jan-79 13:49:37 Monitor IPC-F 603 [7C0] *START*
/TO:ML21-4:BROWN, -- Distribution to ML21-4, slot 132
File: DSKM:CXQAFD.MEM<255>[400,1066] Created: 02-Oct-78 15:08:00 Printed: 03-Jan-79 13:50:10
QUEUE Switches:  /PRINT:ARROW /FILE:ASCII /COPIES:1 /SPACING:1 /LIMIT:728 /FORMS:NORMAL
```

IDENTIFICATION

PRODUCT CODE: AC-8234D-MC

PRODUCT NAME: CXQAFD0 DEC/X11 MOD PROGMR GDE

PRODUCT DATE: SEPTEMBER 1978

MAINTAINER: DEC/X11 Support Group

DEC/X11 MODULE PROGRAMMERS GUIDE

TABLE OF CONTENTS

TABLE OF CONTENTS (CONT)

1.0 INTRODUCTION

The purpose of this document is to explain how to code a DEC/X11 Exerciser Module. Before attempting to write a DEC/X11 module, the programmer must be familiar with the DEC/X11 exerciser package, concepts, and have a working knowledge of the MACY11 Assembly Language.

The following discussion assumes that the programmer understands the programming and hardware specification of the device the module is being written for, and has constructed a preliminary flow chart of the program.

2.0 DEC/X11 ABSTRACT

DEC/X11 is the system exerciser for the PDP-11 family. It is designed to promote system interaction and to detect system failures, if any, caused by said interaction. DEC/X11 is designed to be used as an overall system confidence test, and to provide an indication of the integrity of individual system components. To that effect, the individual exerciser modules can be written to be either simple or extensive, depending on the exact purpose.

The basic components of the DEC/X11 package are:

1. DEC/X11 monitor library.
2. DEC/X11 option/device exerciser modules.
3. DEC/X11 configurator/linker program.
4. DEC/X11 documentation.

The monitor library, exerciser option modules, and the configurator/linker program are used to generate a "Runtime exerciser" that is loadable by the standard ABS loader. The configurator/linker program is used to configure and link the desired monitor and modules and to generate the exerciser. The exerciser includes only the monitor and exerciser modules required to exercise the system. This document does not concern itself with the configuration-linking process. Description and instructions for the configurator/linker are found in the DEC/X11 Users Manual .

DEC/X11 software is in modular form. This format will allow the final exerciser to be only as large as necessary. The modular format also provides for easy updates and modifications.

3.0 DEFINITION OF A DEC/X11 OPTION MODULE

A module is a program which is dedicated to one device, controller or option and is interfaced to the DEC/X11 monitor. Each module is only part of the system exerciser which can be made up of numerous modules. The monitor provides various support and utility routines which can be used by each module.

The module should be written in such a way as to "Exercise" the device. It should not be written with a stand-alone diagnostic in mind. One must remember that when DEC/X11 is run, the stand-alone diagnostics have been run and the devices verified. The DEC/X11 exerciser modules are written to find system problems, not static problems. The module should be designed to exercise the devices with the emphasis on throughput and a minimum amount of overhead. It should be noted that tests requiring "operator intervention" should not be included in the modules.

4.0 DEC/X11 OPTION MODULE CLASSIFICATION

Modules need software hooks so that they can communicate with the monitor. Different types of modules require different software hooks. These software hooks are generated by the use of "Header Statement" macros. DEC/X11 modules have been categorized into six groups depending upon the type of option or device that is being tested. By using the various types of Header Statments that have been defined for each group (section 4.1 to 4.6), the software hooks that are required by the module will be generated when the module is assembled with the common library file called "DDXCOM.P11". This file includes macro call definitions and common tags that are required by the different types of modules.

NOTE

> The first thing that has to be determined by the programmer, is the type of module that is being written.

This has to be done to determine the appropriate "Header Statement" required by the module so that the proper software hooks will be generated. [Read the "Header" definitions described in section 4.1 to 4.6 to determine the proper Header Statement required.]

NOTE

> The "Header" statement macro must be the first line of code in the module.

4.1 BKMOD

The BKMOD type module runs in what is called the background mode. When writing BKMOD modules, the "START" and "RESTRT" tags must be at the same location. BKMOD's are used when exercising non-interrupt devices or functions. For example, the CPA module tests all the basic instructions of the PDP-11. The format for the header macro call for a BKMOD is shown below:

```
BKMOD    <ABCD >,ADDR,VECTOR,BR1,BR2,DVC,ICONST,IDNO
```

4.2 NBKMOD

The NBKMOD type module runs in non-restartable background mode,. This type of module is run first and only once. After it runs successfully it never runs again unless the exercise is aborted and started over. An example would be a module to check timing or parity on the system before the other modules were run. The format for the header macro call for an NBKMOD module is shown below:

```
NBKMOD    <ABCD >,ADDR,VECTOR,BR1,BR2,DVC,ICONST,IDNO
```

4.3 SBKMOD

The SBKMOD type module runs in special background mode. This type of module runs only once after every relocation of the exerciser. This module type runs before NBKMOD's. An example would be a multi-processor enviroment using a bus switch where the modules function would be required whenever the exerciser altered it's position in core. The format for the header macro is shown below:

```
SBKMOD    <ABCD >,ADDR,VECTOR,BR1,BR2,DVC,ICONST,IDNO
```

4.4 IOMOD

The IOMOD type module operates in an input/output Mode. These modules are interrupt driven and are capable of input/output operation. They are generally associated with buffer driven devices, ie: devices that do not do NPR's and do not have "word count" registers. Examples are: the TA-11 cassette, the floppy disk, papertape reader/punch, and line printers. The format for the header macro call for a IOMOD module is shown below:

```
IOMOD    <ABCD >,ADDR,VECTOR,BR1,BR2,DVC,ICONST,IDNO
```

4.5 IOMODX

The IOMODX type module is an extended IOMOD. It is used for NPR devices and provides added capabilities not required by IOMOD's. Some of these added capabilities include; use of the monitor supplied write buffer, ability to change the size of the read and write buffers, access to the monitor's CHECK DATA utility, and conversion routines to get 18-bit addresses from 16-bit addresses and, on certain cpu's, 22-bit addresses from 18-bit addresses. An example of this type of module would be the RK-11 disk module. The format for the header macro call for an IOMODX is as follows:

```
IOMODX   <ABCD >,ADDR,VECTOR,BR1,BR2,DVC,ICONST,IDNO,RBUFVA,RB
UFSZ,WBUFRQ
```

4.6 IOMODP

The IOMODP type module is a partially relocatable IOMODX. This means that the module, because of hardware restrictions, must be relocated only on certain fixed boundaries such as 32K. Please consult with the DEC/X11 group if this type of module is necessary. The format for the header macro call for the IOMODP is as follows:

```
IOMODP   <ABCD >,ADDR,VECTOR,BR1,BR2,DVC,ICONST,IDNO,RBUFVA,RB
UFSZ,WBUFRQ
```

4.7 IOMODR

The IOMODR type module is a restricted IOMOD which can not be relocated at all because of hardware restrictions. This type of module will only be permitted to run when the exerciser is in the lowest memory bank (relocated to 0). The Unibus tester module (BTB) is an example of this. Please consult the DEC/X11 group if this type of module is necessary. The format for the header macro call is as follows:

```
IOMODR   <ABCD >,ADDR,VECTOR,BR1,BR2,DVC,ICONST,IDNO
```

5.0 HEADER MACRO CALL

In the Header macro Statement, there is a string of arguments that must be defined. The type of module being defined determines which arguments to be used. These arguments are explained in detail in Sections 5.01 to 5.08.

An example of a header statement, taken from the RKA module, is shown below.

```
IOMODX <RKAA >, 177400,  220, 5, 0, 0, 512., 5, BUFIN, 256.,
1024.

NAME <ABCD >    ADDR  VECTOR BR1 BR2 DVC ICONST IDNO RBUFVA
RBUFSZ WBUFRQ
2b 1;.f;.j
```

The code generated, from this header call is shown on page 42.

5.01 Module Name Format

The format for naming the module in the title statement is as follows:

```
     <ABCD >
```

The argument consists of five characters within angle brackets.
The last character must be left blank and is filled by the configurator/linker at configuration time to identify multiple
copies of the same module. The first four characters which must be
supplied will follow the following convention:

AB — Any two alpha mneumonic characters that identify the hardware and thus the module. An example would be a "RP" for the RP-11 disk.

C — Alpha character to distinguish between two or more different modules for the same generic device. The sequence A, B, C, etc. must be used for each additional module. The RPB module for the RP04 would be an example of the second RP module that was written.

D — Alpha character used to specify the module revision. The sequence A, B, C, etc. must be used for each new

revision. RPBC would show that this would be the third
version of this program.

(SPACE) Numeric character used at configuration time to distinguish between multiple copies of the same module. This
space is filled by the configurator/linker. You will
always leave this as a space character when you code your source program.

5.02 ADDR address of Device or Option

This specifies the unibus address of the device or option. If more than one address is assigned, "ADDR" should specify the first address in the group of contiguous addresses. When the
header macro is executed, "ADDR" will be placed in word 6 [ADDR:] of the module interface [shown on page 42].

If the address will not be known until configuration time, "ADDR" should be specified as a 1. If the operator did not follow directions and set up the address, this will generate an
odd address system error. This will reduce the possibility of
chasing false errors.

"ADDR" must always be equal to zero for a BKMOD.

This word "ADDR:" must be used to derive any addresses the module needs to access the device registers. Two examples are shown below:

5.02.1 Example #1:

```
MOV   ADDR,R5        ;GET 1ST ADDRESS
 .
 .
 .
MOV   #10000,2(R5)   ;SET POWER CLEAR BIT IN COMMAND REG.
MOV   #1024.,4(R5)   ;LOAD BYTE COUNT REG.
MOV   WBUFPA,6(R5)   ;LOAD MEMORY ADDRESS REG.
MOV   CMD,2(R5)      ;LOAD THE COMMAND REG.
```

5.02.2 Example #2:

```
MOV   ADDR,R5        ;GET 1ST ADDRESS
MOV   R5,MTS         ;SET UP STATUS REG. POINTER
TST   (R5)+          ;MAKE CERTAIN REG. CAN BE ACCESSED
MOV   R5,MTC         ;SET UP COMMAND REG. POINTER
```

```
TST     (R5)+           ;MAKE CERTAIN REG. CAN BE ACCESSED
MOV     R5,MTBC         ;SET UP BYTE COUNT REG. POINTER
TST     (R5)+           ;MAKE CERTAIN REG. CAN BE ACCESSED
MOV     R5,MTMA         ;SET UP ADDRESS REG. POINTER
  .
  .
  .
MOV     #100,@MTC       ;SET POWER CLEAR BIT IN COMMAND REG.
MOV     WBUFPA,@MTMA    ;LOAD MEMORY ADDRESS REG.
```

The first technique requires less storage and is more efficient to use but requires that you save and restore R5 if you need to access registers in the interrupt service routine. This method is not recommended and should be avoided. The second technique is less efficient but eliminates the need to save and restore the registers in the interrupt service routine. It also makes the program easier to follow and maintain. This is the recommended method.

5.03 VECTOR Vector address of device or option

This argument specifies the vector address assigned to the device or option. If more than one vector is assigned, "VECTOR" will specify the first vector address in the group. When the header macro is executed, "VECTOR" will be placed in word 10 [VECTOR:] of the module interface [shown on page 42].

"VECTOR" must always be equal to zero for a BKMOD.

If no vector is assigned or if it will not be known until configuration time, "VECTOR" should be specified as a 1. If the operator did not follow directions and set up the vector, this will generate an odd address system error. This minimizes the possibility of this module destroying some other module's vector. "VECTOR" must be used in conjunction with BR1 and BR2 to set up the interrupt vectors.

5.04 BR1, BR2 Bus priority levels

"BR1", and "BR2 must always be equal to zero for a BKMOD.

These arguments specify the BR priority levels assigned to interrupt driven devices. Normally only BR1 will be used. BR2 will only have to be specified if the device has the capability of interrupting at two different levels. When the macro is executed the values specified are encoded and placed in word 12 of the module's interface [shown on page 42]. "BR1" is put into the low byte, and "BR2" into the high byte. For this reason, always use a "MOVB" instruction to load BR1 and BR2. The following example shows a use of "VECTOR" and "BR1, BR2".

5.04.1 Example:

```
MOV   VECTOR,R0       ;GET VECTOR ADDRESS
MOV   #MTINTR,(R0)+   ;POINT VECTOR TO INTERRUPT SERVICE.
MOVB  BR1,(R0)+       ;SET BR1 LEVEL
TSTB  (R0)+           ;SET BACK TO EVEN ADDRESS
```

5.05 DVC Device Count

DVC is specified at configuration time. This indicates the number of active devices to be exercised by the module. "DVC" contains the octal number of devices. It is converted in DVID1 so each bit represents one device. If the value of DVC is defined as a 0, DVID1 will get assembled to equal a 1. This means the module will try to test device 0. Up to 16 seperate devices may be selected. If, at configuration time, 3 devices were to be exercised, then the number 3 would be entered for the device count (DVC). When the configurator/Linker types out the DVC, it will now contain the number 7---one bit location representing each device. This is the number that is placed in DVID1, location 14 [shown on page 42] in the module interface. For example, if 3 were entered for DVC, then DVID1 would be set to:

```
BIT0 (1) = DEVICE 0 SELECTED
BIT1 (1) = DEVICE 1 SELECTED
BIT2 (1) = DEVICE 2 SELECTED
```

Hence 3 devices are selected and in DVID1 there is a 7. At run time "DVID1" can be modified to specify any combination of devices to be exercised. The module will use "DVID1" to determine which devices are to be exercised.

The only way to exercise non-consecutive devices is to modify "DVID1" after configuration and just before run time by using the MOD command. The MOD command is explained in the Users Manual.

The location "DVID1" should not be modified by any code in the module. The contents of "DVID1" should be moved to a work location and that location can be used by the module. A reason for modifying the temporary location would be to eliminate devices with hard errors from the exercise. The normal configured parameter must not be changed. An example of the use of "DVID1" is shown below:

5.05.1 Example:

Assume the module exercises 8 devices

```
      MOV   DVID1,TDVD  ;SAVE SELECTION PARAMETER
      MOV   #1,MSK      ;SET DEVICE MASK
      MOV   #-1,R1      ;SETUP R1
KC1:  INC   R1          ;GENERATE A NUMBER
      BIT   MSK,TDVD    ;SELECT BIT SET?
      BNE   1$          ;BR IF YES
      CMP   #10,R1      ;DONE 8 UNITS?
      BEQ   OUT         ;BR IF YES
1$:   MOVB  R1,CMD+1    ;SELECT UNIT
       .
       .
       .
      CODE TO EXERCISE SELECTED UNIT
       .
       .
       .
      ASL   MSK         ;SHIFT SELECT BIT
      BR    KC1         ;GO TEST NEXT UNIT
```

If, in the exercise routine, the module determines that the selected device was off line, the module could drop the device from the exercise.

```
      BIC   MSK,TDVD    ;DESELECT BAD DEVICE
      BNE   1$          ;BR IF ANY SELECT BITS SET
      END               ;DROP THE MODULE
1$:   continue
```

5.05.2 ICONST Iteration Constant

The Iteration Constant is the number of times the module should run to be considered end of pass. This number should be adjusted as follows:

The program run time for an invisible IOMOD (eg: floppy disk) should be 30-40 sec.
The program run time for a visible IOMOD (e.g. Line Printer) should be one minute.
The program run time for a BKMOD (e.g. CP Test) should be 20 sec.
ICONST is moved to location ICONT in the header during assembly. This location is used by the "ENDIT" macro call described in Section 7.12.

5.05.3 IDNO Module Identification Number

Each module has a unique I.D. number for use by APT. During assembly this number is moved to location IDNUM in the header. Consult the DECX Group for the appropriate number.

5.06 RBUFVA Virtual Address of the Read Buffer

This argument, passed with the IOMODX and IOMODP calls, is the starting location for the read buffer located within the module. The read buffer area contains the data that was read from the device. The most commonly used tag to denote the starting location of this buffer is "BUFIN". This read buffer should normally be placed towards the end of the module. The monitor uses this tag as a reference point when the check data [CKDATA] and get physical address [GETPA] macros are called.

5.07 RBUFSZ Read Buffer Size

This location, passed with the IOMODX and IOMODP calls, contains the actual size (or length) of the read buffer in words. Presently, read buffers are suggested to be a maximum of 256 words. The monitor uses this location as an indication of the number of words to be checked in a Check Data [CKDATA] call.

This value is assumed to be in Octal; therefore, a period must follow any number representing the base 10 system.

5.08 WBUFRQ Write Buffer Size Requested

This location, passed with the IOMODX and IOMODP calls, contains the desired write buffer transfer size in words. If the write buffer request is larger than the available write buffer, "THE LARGEST SIZE AVAILABLE" is stored in header location "WBUFSZ" [shown on page 42]. For this reason the module should always reference the "WBUFSZ" location for transfer size and not WBUFRQ.

This value is assumed to be in Octal, therefore, a period must follow any number representing the base 10 system. The standard default size is 1024. words (1K).

5.1 MODULE STATUS WORD

This status word is located at location 26 in the module header interface. When the module program is assembled, each macro generates different active bits in the status word of that module. The status word also contains informative bits of the module's condition.

The module status bits have the following meanings.

BIT 14 (1)	Module selected
BIT 14 (0)	Module deselected
BIT 13 (1)	Module had been running but was dropped by the monitor
BIT 13 (0)	Module was not dropped
BIT 11 (1)	Module is active
BIT 11 (0)	Module is inactive
IOMOD	100000
IOMODR	112000
IOMODP	102000
IOMODX	110000
BKMOD	000020
NBKMOD	001000
SBKMOD	000000

The right half of the status word indicates the processor status when running the module. In the BKMOD, bit 4 is set. Note: BIT 11 (active bit) will always be zero for BKMODS when it's status word is outputted by a keyboard command such as SUM, MAP OR EXAM.

5.2 SR1,SR2,SR3,SR4 SOFTWARE SWITCH REGISTERS

The software switch registers are located at locations 16-24 in each module header interface. These words can be used by the module program as general purpose program switches. These words can be used to define or specify a unique device option or to point to a specific routine in the module.

These words "MUST NOT" be modified by the software. Any SR1-SR4 options must be specified by the operator before the module is run.

All SR1-SR4 bit definitions must be defined in the module

documentation. This will enable to operator to correctly modify "SR1-SR4" to meet specific needs.

5.2.1 Example:

Consider the RK11 (Appendix B Page 51) module.

SR1 defined: BIT 2 RESET (0) TYPEOUT DATA LATE ERRORS.
BIT 2 SET (1) DO NOT TYPEOUT DATA LATE ERRORS

```
BIT   #BIT2,SR1  ;TYPEOUT ERROR?
BNE   6$         ;NO
MSGN$,DLTERR,BEGIN
```

In this example, SR1 is used to decide which direction the module will take. One way it will type out the error message and the other way it will omit the error message.

6.0 START TAG

The tag "START" must be used to identify the first executable instruction in the module.

When the module is assembled, the address of "START" is inserted into the "INIT" word in the module "Header" [shown on page 42] and the monitor uses the contents of this location as the starting address of the module.

6.1 RESTART TAG - RESTRT

This tag must be used to identify the restart location in the module. The monitor will return to this location after completing the final ENDIT call. "RESTRT" is normally at the beginning of the program, just after the initialization code. In "BKMOD" modules, the "START" and "RESTRT" tags must be in the same place.

7.0 MACRO CALLS

DEC/X11 has many macro calls available for use in the coding of DEC/X11 modules. Some are necessary and must be used, while others simply make life easier. These macros include calls for determining buffer sizes, check data routines, monitor service routines, and different error reporting routines.

7.1 GWBUFF Get Write Buffer

This macro is used in IOMODX and IOMODP modules to get write buffer information from the monitor. The monitor uses the "write buffer" as an area to write "from" and not "into". Information should never be written into the write buffer for it could be destroying its own program. The monitor will size memory to determine the available core left that can be used for a write buffer. The monitor then compares this value against the size of the write buffer requested (WBUFRQ) in the module header. If the write buffer size requested was larger than the amount available, the monitor will store the size available in the Write Buffer Size (WBUFSZ location 142 on page 43) location in the header of the module. If the amount available was larger than the amount requested, the amount that was requested in the Write Buffer Request (WBUFRQ) would be stored in the write buffer size (WBUFSZ) location.

NOTE

> For this reason it is essential that the module program use the "Write Buffer Size" (WBUFSZ) and not the "Write Buffer Size Requested" (WBUFRQ) value to obtain the size of the Write Buffer.

The GWBUFF macro also places the physical starting address of the write buffer into the "WBUFPA" location and the extended addressing bits into the "WBUFEA" location. The extended addressing bits (16 and 17) are placed into bit positions 4 and 5 since this is the location of the EA bits in most of the peripherals. The locations WBUFSZ and WBUFRQ in the module header must not be modified. This macro should be used before each new transfer cycle (eg: [GWBUFF], WRITE, Read, Check data, [GWBUFF], WRITE, Read, etc.).

The "WBUFSZ" value is a positive number and often must be negated before being used. The RKA module in "Appendix B" shows a good example on how to use this macro. An example is also shown below:

```
GWBUFF                  ;GET THE WRITE BUFFER INFORMATION
MOV   WBUFSZ,WCNT       ;GET WRITE BUFFER SIZE
NEG   WCNT              ;CONVERT TO A NEGATIVE NUMBER
MOV   WCNT,@RKWC        ;LOAD RK WORD COUNT REG
MOV   WBUFPA,@RKBA      ;LOAD RK BUS ADDRESS
BIS   WBUFEA,FUNCT      ;LOAD EXTENDED ADDRESSING BITS IN
                        ;FUNCTION COMMAND LOCATION.
```

The assembled appearance of GWBUFF will be "GWBUF$, BEGIN".

7.2 GETPA ADR Get Physical Address

This macro is used to convert a 16-bit virtual address to an 18-bit physical address. "ADR" is an argument which contains the virtual address. The monitor takes this virtual address, converts it to an 18-bit address, and places it in the two locations following the "ADR" location. These 3 locations are already in the header for of the "IOMODX,IOMODR" and IOMODP" modules.

NOTE

When using any other type of module, make sure that that the following three locations are provided, and in this order.

ADR: 16-bit Virtual Address
PA: will contain 16-bit physical address on return
EA: will contain extended addressing (16 and 17) in bit positions 4 and 5 on return

The call "GETPA" will normally be used to get the physical address of the read buffer, which the module will load into the device register before issuing a read command. This command may be used, though, by any type of module whenever a physical address is needed. The physical address equivalent of a virtual address will change only between an End Of Pass and restart. Thus "GETPA" calls should occur in the START/RESTART code only. The RK module in Appendix B uses this macro. Below is an example of "GETPA" macro.

```
GETPA RBUFVA            ;GET THE PHYSICAL ADDRESS OF THE
                        ;CONTENTS OF RBUFVA
MOV   RBUFPA,@RKBA      ;LOAD BUFFER ADDRESS OF DEVICE
BIS   RBUFEA,FUNCT      ;LOAD EXTENDED ADDRESSING BITS
                        ;IN FUNCTION COMMAND LOCATION
```

The assembled appearance of GETPA ADR will be "GETPA$. ADR".

7.3 MAP22 ADR Map 22 Bit Physical Address

This macro is used to convert an 18-bit unibus address to a 22-bit physical address. MAP22 is used on devices capable of 22 bit addressing, such as RH70 devices on the PDP-11/70. "ADR" is an argument which contains the lower 16 bits of the 18-bit address. Location XMEM, which immediately follows ADR, must contain the shifted extended address bits (16 + 17). The monitor converts this to a 22 bit address and places it in the 2 locations immediately following the "XMEM" location.

NOTE

When using this call, make sure four locations are provided in the module in the following order:

ADR: lower 16 bits of the 18 bit address.

XMEM: contains bits 16 and 17 in bit positions 4 and 5.

PA22: will contain the lower 16 bits of the physical 22 bit address.

EA22: will contain the extended addressing bits (16 to 22) in bit positions 0 through 4.

The "MAP22" call will normally be used to get the physical address of the read buffer, which the module will load into the device register before issuing a read command. Word "CONFIG" in the module header is loaded with the 22-bit addressing bit (bit 09) if the hardware and the monitor support 22-bit addressing. The term "ADDR22" is equated to bit 09 in DDXCOM, the macro file for DEC/X11 option modules.

An example for this macro taken from the RPBA module is shown below:

```
BIT     #ADDR22,CONFIG ;22-BIT ADDRESSING?
BEQ     1$             ;NO, BRANCH
MOV     @RHBA,PA18     ;YES, GET 18 BIT ADDRESS
ASR     XMEM           ;EA BITS IN XMEM PREVIOUSLY SETUP
ASR     XMEM           ;MOVE EA BITS
ASR     XMEM           ;TO POSITION 4 + 5
ASR     XMEM
MAP22   PA18           ;GET 22 BIT ADDRESS
MOV     PA22,@RHBA     ;SETUP BUS ADDRESS REGISTER
MOV     EA22,@RHBAE    ;SETUP EXTENDED ADDRESS REGISTER

;NEED NEXT FOUR LOCATIONS FOR MAP22

PA18:   0              ;LOWER 16 BITS OF THE 18 BIT
                       ;UNIBUS ADDRESS
XMEM:   0              ;EA BITS OF THE 18 BIT ADDRESS
                       ;IN BIT POSITIONS 4 + 5
PA22:   0              ;LOWER 16 BITS OF THE 22 BIT
                       ;UNIBUS ADDRESS
EA22:   0              ;EA BITS OF THE 22 BIT UNIBUS
                       ;ADDRESS (BITS 0 TO 4)
```

The assembled appearance of MAP22 ADR will bit "MAP22$,BEGIN,ADR".

7.4 CKDATA ADR,ERRET Request for monitor to Check Data

This macro requests the monitor to do a data comparison. It is used only in IOMODX and IOMODP modules. The "ADR" argument used with this macro must be the tag of the location that contains the lower 16 bit address of the data being compared. This location is immediately followed by a word containing the EA bits shifted into position 4 and 5, followed by a third word containing the size of the buffer.

The CKDATA macro compares the buffer indicated in the argument with the write buffer. The write buffer information is located in the "header" starting at WBUFPA and must have been previously set up by the call GWBUFF.

The "ADR" argument most commonly used with this call is RBUFPA. The IOMODX module has allowed room in the header for the locations needed. These locations in the header appear as follows:

```
RBUFPA:    ;Read buffer physical address
RBUFEA:    ;Read buffer EA bits
RBUFSZ:    ;Size of the Read Buffer
WBUFPA:    ;Write buffer physical address
WBUFEA:    ;Write buffer EA bits
WBUFRQ:    ;Write buffer size requested
WBUFSZ:    ;Write buffer size available
```

For example, the source macro call would be CKDATA RBUFPA and the assembled version will appear as:

```
CKDATA$,BEGIN,RBUFPA
```

The monitor will automatically handle all data error reporting associated with this Check Data (CKDATA) call.

The argument ERRET (if defined) will request the monitor to set return PC to that location if there was a data error. If there are no errors, or ERRET is not defined, processing will continue with the next instruction following the CKDATA call.

7.4.1 CHECK DATA ERROR

If the monitor discovers a data error while checking the buffer, it will report it. The printout looks the same as that for a regular data error (7.5.1). In a "CKDATA" call, all errors in a given transfer are counted as one error in the module error counter. This count is not incremented until after the data errors have been reported. The total number of words in the transfer and the total number of errors encountered are also reported.

7.5 DATERR

This call is used to report an 'in-core' data comparison errors. The "DATERR" call is usually never used in the IOMODX or IOMODP modules. Instead, they use the "CKDATA" call to request monitor checking of their data. Refer to section 7.4. Prior to using the "DATERR" call, the module must load certain words in the module interface. This is to enable the monitor to report address and contents information on the error typeout. The words that must be loaded are listed below:

ACSR — Address Control Status Register - Address of the control reg.

SBADR — Should Be Address - Address of the "Should Be" or correct data.

WASADR — Was Address - Address of what the data really "Was".

ASB — Actual data Should BE - Test data, what the data is supposed to be.

AWAS — Actual data Was - Actual data, what the data really was.

7.5.1 Example of the Data Printout

The data printout looks as follows:

```
ABCD0       PC XXXXXX  APC YYYYYY  PASS#NNNNN.  ERR#NNNNN.
CSRA AAAAAA S/B BBBBBB WAS WWWWWW  WRADR DDDDDD RDADR EEEEEE
```

Where:

ABCD0	Failing Module name.
PC XXXXXX	Actual 22 bit physical address of DATER$ call.
APC YYYYYY	Assembled PC of DATER$ call.
PASS#NNNNN.	Pass number during which error occurred (Decimal)
ERR#NNNNN.	Total number of errors encountered (Decimal)
CSRA AAAAAA	CSR address of failing device.
S/B BBBBBB	Data should be (Good Data)
WAS WWWWWW	Data was (Bad Data)
WRADR DDDDDD	Address of expected data
RDADR EEEEEE	Address of bad data.

7.6 HRDER , <COMMENT>

This is used to request that the monitor print out an error message for the module on the console. This call can be used for all hard errors. The optional argument "COMMENT" permits the cause of the error to be printed out as a comment statement in the assembled listing. prior to coding the "HRDER" call, the module code must load up the following words in the modules interface:

CSRA with the address of the control register
ACSR with the contents of the control register
ASTAT with the contents of the status register, if applicable.
ERRTYP with the code for the particular error. see APPENDIX D for list of error codes. If not listed, consult the DECX group and a new code will be generated.

These locations cannot be loaded with any information other than that indicated. The reason for loading these locations is that when the monitor first initializes the module, these locations are zeroed, and after each error, the status of these registers could change. After printing the error, the monitor transfers control back to the statement immediately following the "HRDER" call.

NOTE

If these locations do not get loaded prior to the 1st error call, they will be printed out as all zeros.

7.6.1 Example:

Assume R4 contains the CSR address and R5 contains the status register address:

```
MOV     R4,CSRA         ;LOAD ADDRESS OF CSR
MOV     (R4),ACSR       ;LOAD CONTENTS OF CSR
MOV     (R5),ASTAT      ;LOAD CONTENTS OF STATUS REG.
MOV     #11,ERRTYP      ;ILLEGAL INTERRUPT OR DONE NOT SET
HRDER, <ILLEGAL INTERRUPT>
BR      RESTRT
```

The comma between HRDER and <comment> signifies the argument "ADR" is not used. see section 7.7 for "ADR" explanation.

7.6.2 Error Printout

The error printout looks as follows:

```
ABCD0   PC XXXXXX  APC YYYYYY   PASS#NNNNN,   HARD ERR#NNNNN
CSRA AAAAAA   CSRC CCCCCC   STATC SSSSSS  ERRTYP NNNNN
```

Where:

ABCD0	Failing module name.
PC XXXXXX	Actual 22 bit physical address of error call
APC YYYYYY	Assembled pc of error call.
PASS#NNNNN.	Pass number during which error occurred (decimal)
HARD ERR#NNNNN.	Total number of errors encountered (decimal)
CSRA AAAAAA	CSR address of failing device, 0 if not applicable
CSRC CCCCCC	Contents of device CSR, 0 if not applicable
STATC SSSSSS	Contents of device status reg, if applicable
ERRTYP NNNNN	octal code for type of error

7.7 HDRER ADR <COMMENT> Extended Error Printout

This call requests the monitor to print out the standard "HRDER" message (section 7.6); and also the contents of all addresses in a table specified by the argument "ADR". The contents of all addresses in the table are printed out until the terminator (177777) is encountered. After the printing is done, control to the module resumes at the location immediately following the "HRDER" call.

7.7.1 Example Code:

To print out the contents of all the registers for the RK-11 Disk, the code would look like the following:

```
        HRDER  TABLE <DUMP THE CONTENTS OF THE RK DISK REG.>

TABLE:  177400        ;POINTERS TO THE RK REGISTERS
        177402
        177404
        177406
        177410
        175412
        177416
        177777        ;TABLE TERMINATOR
```

NOTE

Before each "HRDER" call, the locations CSRA, ACSR, ERRTYP and ASTAT must be

loaded as in 7.6.

7.7.2 "EXTENDED" Error Printout Example

```
ABCDO  PC XXXXXX APC YYYYYY  PASS# NNNNN.  HARD ERR# NNNNN.
CSRA AAAAAA  CSRC CCCCCC  STATC SSSSSS  ERRTYP NNNNNN
XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX XXXXXX
```

The first two lines of the extended error printout have the same meaning as the error printout in section 7.6.2. The third AND any additional lines, consist of up to eight (8) octal values per line. They are printed to provide additional information on the nature of the error. These octal values that are typed out should be well defined in the module's documentation, since only numbers are actually printed. If the table contains device registers, the registers in the table should be in the same order as the actual device registers.

7.7.3 SOFER ADR <COMMENT>

This call is used for "SOFT" data errors. It is called and used in the same manner as "HRDER". The arguments "ADR" and "COMMENT" are of course optional. The standard and extended error printouts give SOFT ERR#NNNNN. All prior setup for SOFER call should be done as shown in section 7.6. In addition, if "ADR" is used - refer to section 7.7.1.

7.8 MSGN ADR Message Call, MSGS ADR Message Call

These calls provide the means to print pure ASCIZ messages. In the ASCIZ string the apostrophe (') serves to delimit the message and the per-cent sign (%) is interpreted by the monitor as a CR/LF. ADR is the pointer to a table of messages. The table must terminate with a 177777 entry. The MSGN call produces a header line identifying the module name and the MSGS call does not.

7.8.1 Example of Source Code

```
MSGN      SOFT

SOFT:     MES2              ;MESSAGE POINTER
          177777            ;TERMINATOR

MES2:     .ASCIZ            '%SOFT ERROR%'
```

Example Code Assembled: MSGN$,SOFT,BEGIN

7.8.2 MSG ADR Message Call

This call provides the means to print pure ASCII messages (one message only - not a table as in MSGN). The apostrophe (') serves to delimit the message and the percent sign (%) is interpreted as a CR/LF. ADR is the address of the message. Since no header identifying the module is produced, the module name should be included somewhere in the message.

7.8.3 Example of Source Code

```
MSG     TEXT

TEXT:   .ASCIZ '%RPA - TOO MANY WRITE ERRORS%'
```

Example Code Assembled: MSG$,BEGIN,TEXT

7.9 BREAK Temporarily Return to the Monitor

This call is used to transfer control, temporarily back to the monitor. It allows the monitor to transfer control to pending requests while the current module is waiting for some asychronous event to occur before proceeding. When executed, the monitor obtains the return address of the module as the address immediately following the "BREAK" call. The monitor saves the general registers in the Module Header locations SVR0 through SVR6, then checks the queues for pending requests. When all previous requests have been serviced, the monitor restores the registers and returns control to the module at the instruction immediately following the break call. Example follows:

NOTE

No timer/timeout loops, wait loops, etc. of any kind are permitted without a BREAK in the loop.

7.9.1 Example of a wait loop Using the BREAK call

```
WAIT:  MOV   #177777,CLK  ;SET UP TIMER
1$:    BREAK              ;RETURN TO MONITOR TO LET OTHER
                          ;MODULES RUN
       BIT   #BIT6,@RKDS  ;IS THE DRIVE READY?
       BNE   2$           ;YES CONTINUE
       DEC   CLK          ;NO, WAIT SOME MORE+TRY AGAIN
       BNE   1$           ;WAIT AGAIN
       JSR   PC,DROP      ;TIMES OUT, DROP THE DRIVE.
```

The assembled code at 1$ will look like the following:

```
1$:
        BREAK$,BEGIN        ;RETURN TO MONITOR TO
        BREAK$,BEGIN        ;LET THE MODULES RUN
```

7.10 EXIT Exit to the Monitor - module waits for interrupt

This call is necessary since DEC/X11 modules are not permitted to "sit and wait" for interrupts to occur. It is used to return control of the processor back to the monitor when the module is waiting for an interrupt. It is important to remember that some instruction prior to the "EXIT" is necessary that will generate a subsequent interrupt. If this is not done, the module will never receive control again and in effect, stop running. For this reason an "EXIT" call is never used when coding a module for a non-interrupt driven device. Another important point to remember is that the "EXIT" call is not a RTI and should never be used to return from an interrupt service routine. When the "EXIT" command is executed, the monitor saves the general registers in locations SVR0 through SVR6 located in the header of the module [shown on page 42]. The monitor then passes control to the next module waiting to be serviced. The following is an example of the use of the EXIT call.

7.10.1 Example:

```
MOV    WB,TMWC          ;LOAD BYTE COUNT REGISTER
MOV    WBUFPA,TMBA      ;LOAD MEMORY ADDRESS REGISTER
BIS    WRCMD,TMCS       ;SET WRITE COMMAND AND INT ENABLE
EXIT                    ;RETURN TO MONITOR.
```

NOTE

> If there is no system clock in the exercise, and an interrupt does not occur after an "EXIT" call is used, the system will hang up and all other modules will stop after an end of pass.

The source code of this macro will be "EXIT". The assembled code will be "EXIT$".

7.11 PIRQ ADR Do an RTI and continue at tag ADR

This call is the DEC/X11 method used to exit an interrupt service routine and defer servicing at a lower priority. This call is placed first in the interrupt service routine whenever possible. This will increase the system throughput by defering non-critical interrupt service routines at a

lower priority. For instance, when the RK11 module receives an interrupt, it is known that none of its registers need immediate service or are in the process of changing. Therefore, a PIRQ is used and checking for errors is done at a lower level. On the otherhand, a communications device such as the DH11, may have to empty a silo buffer immediately or data may be lost. After the necessary functions have been taken care of, then a PIRQ or an RTI (if more interrupts are expected) may be used. When PIRQ is executed, the monitor stores the request in a first-in/first-out queue at priority 7 and does an RTI. This turns control of the processor back to whatever code had control when this module interrupted. It is necessary to keep interrupt service routines as short as possible to prevent the other modules from being temporarily locked out and possibly getting data late or timing errors.

When the PIRQ request is serviced from the monitor Queue, the general registers are restored from the module header locations SVR0 through SVR6. The monitor then transfers control back to the module at the location specified by the "ADR" argument.

NOTE

If it is necessary to use the general registers in the interrupt service routine before using the "PIRQ" request, the registers must first be saved. This is required because these registers belong to the code that was being executed when this module interrupted. These registers must also be restored before using the "PIRQ" or an "RTI".

7.11.1 Example:

```
;INTERRUPT SERVICE ROUTINE
NTRUPT:  PIRQ  1$          ;REQUEST TO HANDLE
                           ;INTERRUPT SERVICE
                           ;FOR THE RK MODULE
1$:      JSR   R5,ERRORS ;GO CHECK FOR ERRORS
```

Assembled Version:

```
NTRUPT:
         ;-------------------------------------------------
          PIRQ$,1$ BEGIN  ;QUEUE UP TO CONTINUE AT 1$ AND RTI
         ;-------------------------------------------------
1$:      JSR   R5,ERRORS
```

7.12 ENDIT End of Iteration

This call informs the monitor that an end of iteration has occurred. The monitor will increment Loc. ICOUNT in the header and compare ICONT with ICOUNT. If equal, the monitor will report end of pass on the console and restart it at the module's "RESTRT" tag. If not equal, the monitor restarts the module at the location immediately following the "ENDIT" call. The module program should never modify locations ICONT or ICOUNT.

NOTE

It is important to disable interrupts before each ENDIT call. If an interrupt occurs during an ENDIT call which signals end of pass, the whole exerciser may relocate and the return address from the interrupt will be incorrect.

7.12.1 ENDIT Example From the RK Module

PASS:

The assembled code at location PASS will look like:

```
PASS:
        ENDIT$,BEGIN    ;SIGNAL END OF ITERATION
                        ;monitor shall test end of pass
```

7.13 ENDMOD Drop Module from Exercise

This call is used to request that the monitor drop the module from the exercise. It would be used if the module detected a fatal error that would prevent the module from continuing. For example, the device selected for test was off line. When this macro is executed the monitor stops the module by setting bit 13 of the word "STAT" in module interface [shown on page 42]. This prevents the module from ever getting control again. It is important that you 'shut down' the module (eg: turn off all interrupt enables) prior to executing the "ENDMOD" call macro. The monitor prints the following message to inform the operator that the module was dropped:

```
ABCD0   DROPPED  AT  APC  YYYYY
```

7.14 OTOA NUM,ADR Octal to ASCII Conversion

This call converts one octal number to six ASCII characters. This may be useful prior to a print message using "MSGN". Six "BYTES" must be provided, starting at location "ADR", to

store the result."NUM" is a location containing the number to be converted.

7.14.1 Example

```
        OTOA    NUM,ADR      ;call macro to convert it to ascii
         .
         .
         .
        NUM:    12345        ;number to be converted
        ADR:    .BLKW 3      ;reserve six bytes.
```

when assembled the result would look like:

```
        OTOA$,BEGIN,NUM,ADR
```

and the result would be:

```
        ADR+0   060
           +1   061
           +2   062
           +3   063
           +4   064
           +5   065
```

7.15 BTOD NUM,ADR Binary to Decimal Conversion

This call converts a binary number to its decimal equivalent represented by 5 ASCII characters. This may be useful prior to a print message using "MSGN". Five bytes must be provided, starting at "ADR", to store the result. NUM is a location containing the number to be converted.

7.15.1 Example

```
        BTOD    NUM,ADR      ;call the macro to convert the
                             ;number
         .
         .
         .
        ADR:    .BLKW 3      ;reserve six bytes
        NUM:    00237        ;number to be converted
```

When assembled the result would look like:

```
        BTOD$,BEGIN,NUM,ADR
```

and the results would be:

```
        ADR+0   060
           +1   060
```

```
+2  061
+3  065
+4  071
```

7.16 RAND Call to Monitor for Random Number

This call is used to request the monitor to generate a new random number. Location RANNUM in the module header will contain the random number.

7.16.1 Example

```
.
.
RAND                ;call for random number
MOV    RANNUM,BLOCK ;To get new block
.
.
```

When assembled the result would look like:

```
.
.
RAND$,BEGIN
MOV    RANNUM,BLOCK ;To get new block
.
.
```

8.0 MODULE CODE

Standard PDP-11 code is used for writing DEC/X11 modules.

8.1 MODULE PROGRAM ORGANIZATION

In most cases, a DEC/X11 module can be divided into 3 sections. They are: initialization, interrupt service, and device service.

8.1.1 INITIALIZATION CODE

This code initializes the DEC/X11 module and the devices that are being tested.

8.1.2 DEVICE SERVICE CODE

This section is the main portion of the module and will be supported by subroutines. This will make the program flow easier to follow and easier to debug. This code will initialize, setup the device or options that are being tested and issue the I/O commands for the I/O devices.

8.1.3 INTERRUPT SERVICE CODE

This code is required to acknowledge the fact that an interrupt has been received and to provide service to that interrupt. It is possible using the PIRQ call to queue up a request to "SERVICE" the interrupting device at a later time. The philosophy applied, is that modules must spend a minimal amount of time in an interrupt service routine at a processor status other than 0. This must be done in order to prevent other devices from being locked out. See Section 7.11 for the proper use of the PIRQ instruction.

8.2 MODULE CODE RESTRICTIONS

Due to the monitor/module relationship, the following restrictions are in effect for exerciser modules:

a. Code must be capable of running on all PDP-11 family processors.

b. No HALT instructions (except during debug).

c. No WAIT instructions.

d. No EMT calls.

e. No user trap calls

f. No processor status word modifications.

g. I/O modules must not perform waiting loops that do not contain a BREAK call.

h. If some code must be performed in the interrupt service routine before the PIRQ, and the general purpose registers must be used, the GPRS must be saved first and then restored before the PIRQ instruction. The GPRS do not have to be saved when executing code after the PIRQ instruction.

i. The stack pointer must not be modified in order to exit an interrupt sequence (use PIRQ call).

j. No alteration of the module header may be made by the module, except, of course, use of the stack.

k. Load Medium Indicator: XXDP places in location 41 a code which indicates which type of device was the load medium for the current boot. Location 40 contains the actual device number (e.g. DK0,DK1),etc.). If your device is an XXDP supported load medium, your module must check these two locations to determine if one of the units was in fact the load medium. If it was,that unit cannot be tested and must be dropped. An example of the required code is in the module listing in APPENDIX A. For a list of XXDP supported load media, along with the meanings of the codes in locations 40 and 41, see the XXDP SUPPORT GROUP.

8.3 PROGRAMMING STANDARDS

Programming standards are needed to help insure similarity, consistency and uniformity throughout all the DEC/X11 modules. Using these standards will make life easier for all those who use DEC/X11. The standards are as follows:

1. Documentation is the 1st part of the module. It will include a brief abstract, requirements (hardware and software), pass definitions, execution time, configuration requirements, device/option setups, operation options, non-standard printouts and any other information that is pertinent to the module. A flow chart can also be added to the documentation to show the general operation of the module.

2. The header statement with its arguments should be on the first page following the documentation.

3. All constants and variables will be defined on the page following the module header code.

4. The tag "START" should be on the 1st line of the next page, indicating the first line of code.

5. Tags should have meaningful names.

6. All local tags will appear in numerical order beginning with 1$.

7. Every line of code will have at least 1 line of meaningful comments.

8. Messages should be placed at the end of the program.

9. Subroutines will be designed to support a single function.

10. All subroutines must contain a standard document header for subroutines. This will include the entry name, function or task of subroutine, the call that was used to get to the subroutine, the parameters passed (if any), registers used, external parameters, special error routines or notes.

11. Any general purpose registers (R0-R5) to be modified in a subroutine, must be saved immediately upon entry to the subroutine and restored just before leaving. Using registers to pass arguments is discouraged.

12. Nesting of subroutines is discouraged. Use of support subroutines, such as convert octal to ascii, and the like, are acceptable.

13. Multiple entry points are discouraged and should be

avoided. When used, all multiple entry points must be at the beginning of the routine, and branched to the proper location from there.

14. There will be only one exit from the routine and it will be the last line of executable code.

15. Subroutines must exit with an RTS instruction. All returns from subroutines (including error returns and any others) must be to the instructions immediately following the JSR call.

9.0 ASSEMBLING THE MODULE PROGRAM

The following discussion assumes the programmer knows how to edit and assemble programs using one of the following techniques:

1. On a PDP11 DOS system using "MACRO11".

2. On a PDP10 system using "MACY11"

The assembled object file will be used by the configurator/linker program to make it part of a system exerciser load module. The following steps are required:

1. Using an appropriate editor program, a module source file is created, having a file extension of ".P11".

2. Using the appropriate assembler the source program is assembled with the file DDXCOM.P11 [File containing the DEC/X11 macro call definitions and common tags].

3. After assembly, the following files should exist:

   a. A source file with a ".P11" extension.

   b. An object file with a ".OBJ" extension.

   c. An assembly listing file with a ".LST" (or similar) extension.

4. Using the appropriate Peripheral Interchange Program:

   a. Output the listing to a line printer.

   b. Output the object file, in PDP11 absolute loader format, to a paper tape or multi-media.

      The Configurator/Linker will accept input from paper tape, and multi-media. The only restriction is that all modules to be linked must be on the same medium.

Having the listing of the module object file on the appropriate input medium, the module is ready to be configured, linked, and run. Refer to the Configurator/Linker section in the DEC/X11 document [MAINDEC-11-DXQBA] for the required procedures.

10.0 MODULE CODE CHECK LIST

The following check list is an aid for the programmer in checking the module code. It highlights many problems encountered while writing the first modules.

1. Use "ADR" to set up the required address pointers.
2. Use "VECTOR", "BR1", and "BR2" to set up the vector area.
3. Use a byte instruction to set up "BR1" and "BR2".
4. Prior to an "EXIT" call, execute some code that will generate a subsequent interrupt.
5. If the module is driving a non-interrupt device, make certain that there are no "EXIT" or "PIRQ" calls.
6. Use an RTI instruction to return from an interrupt service routine only when the PIRQ instruction isn't used.
7. The "PIRQ" call should be used to exit an interrupt service routine.
8. Make sure that there are no MONITOR calls in the interrupt service routines.
9. The module must save and restore any registers that are needed during the interrupt service routine.
10. Prior to the "HRDER" and "SOFER" calls, load "CSRA", "ACSR,"ERRTYP",", and "ASTAT".
11. Prior to the "DATERR" call, load "CSRA", "ASB", "AWAS", "WASADR", and "SBADR".
12. Make sure the module can recover and continue running after non-fatal errors.
13. If a fatal error was found that makes it impossible to continue, use an "ENDMOD" call to drop the module.

14. Use an "ENDIT" call to report that the module has completed an iteration.

15. Make sure there are no wait loops in the module that do not include the BREAK call.

16. Take care of any pending interrupts prior to executing any monitor calls.

17. Absolute memory references are prohibited.

18. Do not use any HALT, WAIT, TRAP or EMT instructions.

19. Use "DVID1" properly to check for multiple devices.

20. If the device has extended memory capability, use "EABITS" properly to set up the extended memory bits.

21. Turn off interrupt enable prior to executing any monitor calls.

22. Use the tag "START" to identify the first executable instruction in the module code.

23. In "BKMOD" modules, the "START" and "RESTRT" tags must be in the same place.

11.0 DEC/X11 Monitor/Module Checkout Procedure

I. DOCUMENTATION

a. All documentation must be complete and correct.

Ii. CODE

a. The code must adhere to DEC/X11 Module standards for uniformity and compatability.

b. The code must adhere to DEC/X11 Module coding standards and conventions as outlined in this guide.

Iii. Evaluation

Every attempt must be made to validate the operational integrity of the module. By adhering to the following procedure, an acceptable level of quality can be assured.

a. RUNNING ALONE

The module must be capable of running alone under its default conditions.

1) The module must be run continuously for not less than 8 hours and preferably 16.

2) The module must be capable of running more than 1 device (or unit, or line, or channel, etc.), the absolute minimum number of 2 devices must be run.

3) All error reports and messages must be simulated and checked for accuracy.

4) All SR1-SR4 options must be used and validated as operational.

5) Various combinations of DVID1 must be tried and validated for proper operation.

6) Various transfer sizes must be tried (where applicable).

7) The module must be run in bipolar and/or MOS memory (if available).

8) The module must run on all PDP-11 processors including the PDP-11/70 and the LSI-11.

9) The module must be run (ie: the program relocated) in all banks of available memory on a system with memory management (KT11).

10) The module must be capable of recovering without errors from a powerfail in different banks of memory.

b. RUNNING IN A SYSTEM ENVIRONMENT

The module must be capable of running in a system environment with other DEC/X11 modules. The minimum system must include: KT11, 28K memory, high speed disk, a communications device, Magtape or Dectape, and an L or P clock.

1) The module must be run in a system environment for not less than 8 hours and preferably 24 hours.

2) The largest system (the smallest being the minimum system noted above) available must be used.

3) At least 1 disk must be included as part of the system.

4) At least 1 clock module (KW11-L, KW11-P) must be configured in as part of the system.

5) The module must be run on the shared bus of a DT03 (if available).

6) The module must be configured before any disk module to assure it does not wipe out any modules following it.

7) The same tests required when the module is running alone must be performed with the module operating in this system environment.

APPENDIX A

Summary Table of Macros

CODED APPEARANCE		ASSEMBLED APPEARANCE
GWBUFF	ADR	GWBUF$,BEGIN,ADR
GETPA	ADR	GETPA$,BEGIN,ADR
CKDATA	ADR	CDATA$,BEGIN,ADR
EXIT		EXIT$,BEGIN
PIRQ	ADR	PIRQ$,BEGIN,ADR
HRDER	ADR <COMMENT>	HRDER$,BEGIN,ADR ;COMMENT
SOFER	ADR <COMMENT>	SOFER$,BEGIN,ADR ;COMMENT
DATERR		DATER$,BEGIN
ENDIT		ENDIT$,BEGIN
ENDMOD		END$,BEGIN
BREAK		BREAK$,BEGIN BREAK$,BEGIN
MSGS	ADR	MSGS$,BEGIN,ADR
MSGN	ADR	MSGN$,BEGIN,ADR
MSG	ADR	MSG$,BEGIN,ADR
OTOA	NUM,ADR	OTOA$,BEGIN,NUM,ADR
BTOD	NUM,ADR	BTOD$,BEGIN,NUM,ADR
RAND		RAND$,BEGIN

APPENDIX B

Assembled Module Example (RKAA)

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 2
RKAA.P11    13-MAR-78 00:00

                .REM  __
```

IDENTIFICATION

PRODUCT CODE:	MAINDEC-11-DXRKA-G-D
PRODUCT NAME:	DEC/X11 RK11 MODULE
DATE:	JANUARY 1978
MAINTAINER:	DIAGNOSTIC GROUP
AUTHOR(S):	ROBERT E. UNDERWOOD

RKAA DEC/X11 SYSTEM EXERCISER MODULE MACY11 30A(1052) 20-MAR-78 15:47 PAGE 3
RKAA.P11 13-MAR-78 00:00

1. ABSTRACT

RKA IS AN IOMODX THAT EXERCISES RK02, RK03, RK04, RK05 DISK DRIVES ON AN RK11 CONTROLLER. IT EXERCISES THE DRIVES BY DOING WRITES, WRITE-CHECKS, READS, AND IN-CORE COMPARISONS. ALL ERRORS DETECTED ARE REPORTED ON THE CONSOLE TTY.

2. REQUIREMENTS

HARDWARE: 1 TO 8 RK DISK DRIVES WITH AN RK11 CONTROLLER

STORAGE:: RKAA REQUIRES:
1. DECIMAL WORDS: 1043
2. OCTAL WORDS: 02023
3. OCTAL BYTES: 4046

3. PASS DEFINITION

ONE PASS OF THE RKA MODULE CONSISTS OF 512 CYCLES OF THE BASIC TEST SEQUENCE (WRITE, WRITE-CHECK, READ, DATA-CHECK). THE TEST SEQUENCE WRITES 1024 WORDS, WRITE-CHECKS SAME, READS THE FIRST 256 WORDS, AND DATA-CHECKS SAME.

4. EXECUTION TIME

ONE PASS OF RKA RUNNING ALONE ON A PDP-11/40 TAKES APPROXIMATELY 1 MINUTE.

5. CONFIGURATION REQUIREMENTS

DEFAULT PARAMETERS:

DEVADR: 177400, VECTOR: 220, BR1: 5, DEVCNT: 1

REQUIRED PARAMETERS:

NONE

6. DEVICE/OPTION SETUP

MAKE CERTAIN THAT ALL DRIVES ARE POWERED UP, WRITE ENABLED, AND READY

7. MODULE OPERATION

TEST SEQUENCE:

A. SETUP DEVICE REGISTER ADDRESSES AND MODULE VARIABLES
B. RESET ALL DRIVES ON-LINE AND DROP ALL THAT ARE NOT
C. GET A DISK ADDRESS AND A FRESH BLOCK OF DATA
D. GET A DRIVE ADDRESS
E. DO A WRITE -- IF ERRORS, REPORT AND RETRY UP TO RETRY LIMIT
F. DO A WRITE-CHECK -- IF ERRORS, REPORT AND RETRY UP TO RETRY LIMIT
G. DO A READ -- IF ERRORS, REPORT AND RETRY UP TO RETRY LIMIT
H. DO A DATA-CHECK -- IF ERRORS, REPORT AND CONTINUE
I. IF END OF PASS, REPORT AND GO TO C
J. IF END OF DRIVES, GO TO C ELSE GO TO D

8. OPERATION OPTIONS

SR1 BIT 0 SET(1):
IF THE RETRY LIMIT IS EXCEEDED ON ANY FUNCTION, A HARD ERROR IS ASSUMED AND THE DRIVE IS DROPPED

SR1 BIT 0 CLEAR(0):
IF THE RETRY LIMIT IS EXCEEDED, THE FUNCTION IS ABORTED AND THE TESTING CONTINUES

SR1 BIT 2 SET(1):
WILL NOT TYPE OUT DATA LATE ERRORS BUT WILL KEEP TRACK OF THE NUMBER OF DATA LATE ERRORS

SR1 BIT 2 CLEAR(0):
TYPE OUT DATA LATE ERRORS AND KEEP TRACK OF THE NUMBER OF DATA LATE ERRORS IN DLTCNT

9. NON-STANDARD PRINTOUTS

A. MOST PRINTOUTS HAVE THE STANDARD FORMATS DESCRIBED IN THE DEC/X11 DOCUMENT

B. ERROR MESSAGES DUMP THE CONTENTS OF THE 8 RK11 REGISTERS IN THE FOLLOWING ORDER:

RKDS RKER RKCS RKWC RKBA RKDA RKMR RKDB

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 5
RKAA.P11    13-MAR-78 00:00

        000000'          IOMODX  <RKAA  >,177400,220,5,0,0,512.,5,BUFIN,256.,1024.
        000000'          MODULE  150000,RKAA  ,177400,220,5,0,0,512.,5,BUFIN,256.,1024.
                         .TITLE  RKAA DEC/X11 SYSTEM EXERCISER MODULE
                ;        DDXCOM  VERSION  5      26-JAN-78
                                                .LIST   BIN
                                 ;*****************************************************************
        000000'                  BEGIN:
        000000' 045522  054101      040  MODNAM: .ASCII  /RKAA / ;MODULE NAME.
        000005'    000                   XFLAG:  .BYTE   OPEN            ;USED TO KEEP TRACK OF WBUFF USAGE
        000006' 177400                   ADDR:   177400+0                ;1ST DEVICE ADDR.
        000010' 000220                   VECTOR: 220+0                   ;1ST DEVICE VECTOR.
        000012'    240                   BR1:    .BYTE   PRTY5+0         ;1ST BR LEVEL.
        000013'    000                   BR2:    .BYTE   PRTY0+0         ;2ND BR LEVEL.
        000014' 000001                   DVID1:  0+1                     ;DEVICE INDICATOR 1.
        000016' 000000                   SR1:    OPEN                    ;SWITCH REGISTER 1
        000020' 000000                   SR2:    OPEN                    ;SWITCH REGISTER 2
        000022' 000000                   SR3:    OPEN                    ;SWITCH REGISTER 3
        000024' 000000                   SR4:    OPEN                    ;SWITCH REGISTER 4
                                 ;*****************************************************************
        000026' 150000                   STAT:   150000                  ;STATUS WORD.
        000030' 000252'                  INIT:   START                   ;MODULE START ADDR.
        000032' 000252'                  SPOINT: MODSP                   ;MODULE STACK POINTER.
        000034' 000000                   PASCNT: 0                       ;PASS COUNTER.
        000036' 001000                   ICONT:  512.                    ;# OF ITERATIONS PER PASS=512.
        000040' 000000                   ICOUNT: 0                       ;LOC TO COUNT ITERATIONS
        000042' 000000                   SOFCNT: 0                       ;LOC TO SAVE TOTAL SOFT ERRORS
        000044' 000000                   HRDCNT: 0                       ;LOC TO SAVE TOTAL HARD ERRORS
        000046' 000000                   SOFPAS: 0                       ;LOC TO SAVE SOFT ERRORS PER PASS
        000050' 000000                   HRDPAS: 0                       ;LOC TO SAVE HARD ERRORS PER PASS
        000052' 000000                   SYSCNT: 0                       ;# OF SYS ERRORS ACCUMULATED
        000054' 000000                   RANNUM: 0                       ;HOLDS RANDOM # WHEN RAND MACRO IS CALLED
        000056' 000000                   RES1:   0                       ;RESERVED FOR MONITOR USE
        000060' 000000                   RES2:   0                       ;RESERVED FOR MONITOR USE
        000062' 000000                   SVR0:   OPEN                    ;LOC TO SAVE R0.
        000064' 000000                   SVR1:   OPEN                    ;LOC TO SAVE R1.
        000066' 000000                   SVR2:   OPEN                    ;LOC TO SAVE R2.
        000070' 000000                   SVR3:   OPEN                    ;LOC TO SAVE R3.
        000072' 000000                   SVR4:   OPEN                    ;LOC TO SAVE R4.
        000074' 000000                   SVR5:   OPEN                    ;LOC TO SAVE R5.
        000076' 000000                   SVR6:   OPEN                    ;LOC TO SAVE R6.
        000100' 000000                   CSRA:   OPEN                    ;ADDR OF CURRENT CSR.
        000102'                          SBADR:                          ;ADDR OF GOOD DATA, OR
        000102' 000000                   ACSR:   OPEN                    ;CONTENTS OF CSR.
        000104'                          WASADR:                         ;ADDR OF BAD DATA, OR
        000104' 000000                   ASTAT:  OPEN                    ;STATUS REG CONTENTS.
        000106'                          ERRTYP:                         ;TYPE OF ERROR
        000106' 000000                   ASB:    OPEN                    ;EXPECTED DATA.
        000110' 000000                   AWAS:   OPEN                    ;ACTUAL DATA.
        000112' 000416'                  RSTRT:  RESTRT                  ;RESTART ADDRESS AFTER END OF PASS
```

```
000114' 000000                  WDTO:   OPEN                    ;WORDS TO MEMORY PER ITERATION
000116' 000000                  WDFR:   OPEN                    ;WORDS FROM MEMORY PER ITERATION
000120' 000000                  INTR:   OPEN                    ;# OF INTERRUPTS PER ITERATION
000122' 000005                  IDNUM:  5                       ;MODULE IDENTIFICATION NUMBER=5
000124' 002560'                 RBUFVA: BUFIN                   ;READ BUFFER VIRTUAL ADDRESS
000126' 000000                  RBUFPA: OPEN                    ;READ BUFFER PHYSICAL ADDRESS
000130' 000000                  RBUFEA: OPEN                    ;READ BUFFER EA BITS
000132' 000400                  RBUFSZ: 256.                    ;SIZE OF THE READ BUFFER
000134' 000000                  WBUFPA: OPEN                    ;WRITE BUFFER PHYSICAL ADDRESS
000136' 000000                  WBUFEA: OPEN                    ;WRITE BUFFER EA BITS
000140' 002000                  WBUFRQ: 1024.                   ;WRITE BUFFER SIZE REQUESTED
000142' 000000                  WBUFSZ: OPEN                    ;WRITE BUFFER SIZE AVAILABLE
000144' 000000                  CDERCT: OPEN                    ;CDATA/DATCK ERROR COUNT
000146' 000000                  CDWDCT: OPEN                    ;CDATA/DATCK WORD COUNT
000150' 000000                  FREE:   OPEN                    ;RESERVED FOR FUTURE USE
        000040                          .REPT   SPSIZ           ;MODULE STACK STARTS HERE.
                                        .NLIST
                                        .WORD   0
                                        .LIST
                                        .ENDR
000252'                         MODSP:
                                ;*****************************************************************************
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 7
RKAA.P11    13-MAR-78 00:00
   207
   209  000256' 012767  000400  177630  START:  MOV     #256.,WDTO      ; WORDS TO MEM
   210  000264' 012767  002000  177624          MOV     #1024.,WDFR     ; WORDS FROM MEM
   211  000272' 012767  000003  177620          MOV     #3,INTR         ; # OF INTERRUPTS/ITERATION
   212  000300' 005067  003534                  CLR     SIDE            ; CLEAR FLAGS AND SIDE INDICATOR
   213  000304' 005067  002210                  CLR     DLTCNT          ; CLEAR DATA LATE EROR COUNTER
   214  000310' 016767  177500  002214          MOV     DVID1,DVICE     ; GET DRIVE INDICATOR
   215  000316' 016767  002210  002210          MOV     DVICE,DRIVE     ; ALSO SAVE IT IN DRIVE
   216  000324' 012767  177775  002210          MOV     #-3,BLK1        ; INITIALIZE BLOCK COUNTER
   217  000332' 005067  002200                  CLR     DRYVE           ; ZERO UNIT NUMBER
   218  000336' 012767  160000  002174          MOV     #160000,DRVSFT  ; INITIALIZE THE SHIFTED DRIVE #
   219  000344' 122737  000002  000041          CMPB    #BIT1,@#41      ; IS RK UNIT 0 THE LOAD MEDIUM ?
   220  000352' 001006                          BNE     3$              ; NO, CONTINUE
   221          012701  000000                  MOV     #0,R2           ; INITIALIZE DRIVE COUNT
   222          113700  000040                  MOVB    @#40,R0 ; GET LOAD MEDIUM COUNT
   223          12701   000001                  MOV     #1,R1           ; LOAD UP R1 TO POINT TO DRIVE 0
   224          105700                  1$:     TSTB    R0              ;IF R0 EQUAL TO 0 THEN
   225          001404                          BEQ     2$              ;GO TO 2$
                006301                          ASL     R1              ;ELSE UPDATE DRIVE POINTER
                105300                          DECB    R0              ;DECREMENT COUNT
                005202                          INC     R2              ; UPDATE DRIVE NUMBER
                000772                          BR      1$              ;TRY AGAIN
                130167  002150          2$:     BITB    R1,DVICE        ;IF DRIVE NOT SELECTED TO BE TESTED THEN
                001404                          BEQ     3$              ;GO TO 3$
                010267  002146                  MOV     R2,DRYVE        ;ELSE LOAD DRIVE ADDRESS TO BE DROPPED
                004767  000716                  JSR     PC,DROP         ;GO DROP IT
   224
   225  000370' 012767  177777  002140  1$:     MOV     #-1,DRYVE       ; INITIALIZE DRIVE COUNTER
   226  000376' 004767  001576                  JSR     PC,SETUP        ; GENERATE REGISTER ADDRESSES
   227  000402' 004767  001774                  JSR     PC,REZET        ; INITIALIZE RK REGS. AND ALL DRIVES
   228  000406' 005767  002120                  TST     DVICE           ; DROP THE MODULE ?
   229  000412' 001536                          BEQ     FINI            ; YES
   230  000414' 000404                          BR      RSTRT1          ;+    / THIS IS
   231  000416' 005767  002100          RESTRT: TST     PASCNT          ;+   /  SUPPORT
   232  000422' 001001                          BNE     RSTRT1          ;+  /   FOR
   233  000424' 000712                          BR      START           ;+ /    DT03
   234  000426'                         RSTRT1:                         ;+/     BUS SWITCH
   235  000426' 104415  000000' 000124'         GETPA$,BEGIN, RBUFVA            ;GET PHYSICAL ADDRESS FROM 16-BIT RBUFVA
   236  000434' 016767  177472  002112          MOV     RBUFSZ,WCNT2    ; SAVE READ BUFFER SIZE
   237  000442' 005467  002106                  NEG     WCNT2           ; GET THE 2'S COMPLEMENT
   238
   239  000446' 004767  000676          STRT:   JSR     PC,BLOCK        ; GET NEXT BLOCK NUMBER
   240  000452' 104414  000000'                 GWBUF$, BEGIN           ;GET WRITE BUFFER INFORMATION
   241  000456' 016767  177460  002066          MOV     WBUFSZ,WCNT1    ; SAVE WRITE BUFFER SIZE
   242  000464' 005467  002062                  NEG     WCNT1           ; GET THE 2'S COMPLEMENT
   243  000470' 016700  002046                  MOV     BLK1,R0         ; LOAD BLOCK # FOR CONVRT
   244  000474' 004767  001600                  JSR     PC,CONVRT       ; GENERATE DISK ADR. FROM BLOCK #
   245
   246  000500' 004767  001022          NEXT:   JSR     PC,DRVADR       ; GET A DRIVE ADDRESS
   247  000504' 005767  002022                  TST     DVICE           ; ANY DRIVES LEFT ?
   248  000510' 001477                          BEQ     FINI            ; NO, GO DROP THE MODULE
   249  000512' 132767  000010  003321          BITB    #BIT3,FLAG      ; ALL DRIVES DONE ?
   250  000520' 001352                          BNE     STRT            ; YES, GO GET ANOTHER BLOCK
   251  000522' 042767  160000  002000          BIC     #160000,DSKADR  ; CLEAR DRIVE ADDRESS
```

```
252  000530' 056767  002004  001772          BIS     DRVSFT,DSKADR   ; LOAD DRIVE ADDRESS
253  000536' 016777  001766  003030          MOV     DSKADR,@RKDA    ; LOAD DISK ADDRESS
254  000544' 032777  000040  003010          BIT     #BIT5,@RKDS     ; WRITE PROTECTED ?
255  000552' 001406                          BEQ     1$              ; NO, CONTINUE
256  000554' 004767  000526                  JSR     PC,DROP         ; YES, DROP THE DRIVE
257  000560' 104403  000000' 004014'         MSGN$,BEGIN,DRP ;ASCII MESSAGE CALL WITH COMMON HEADER
258  000566' 000744                          BR      NEXT            ; GO ON TO NEXT DRIVE
259  000570' 032777  000100  002764  1$:     BIT     #BIT6,@RKDS     ; DRIVE READY ?
260  000576' 001003                          BNE     2$              ; YES, CONTINUE
261  000600' 004767  001014                  JSR     PC,NOTRDY       ; NO, WAIT FOR READY
262  000604' 000720                          BR      STRT            ; TRY AGAIN
263  000606' 005067  003230          2$:     CLR     TRY1            ; ZERO RETRY COUNTERS
264  000612' 105067  003226                  CLRB    TRY3            ;
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 9
RKAA.P11    13-MAR-78 00:00
   265
   266
   267  000616' 004567  000212          GO:     JSR     R5,WRITE          ; WRITE SOME DATA
   268  000622' 000434                           BR      RETRY1            ; IF ERRORS, TRY IT AGAIN
   269  000624' 132767  000004  003207            BITB    #BIT2,FLAG        ; DID THE DISK OVERFLOW ?
   270  000632' 001407                           BEQ     GOA               ; NO, CONTINUE
   271  000634' 142767  000004  003177            BICB    #BIT2,FLAG        ; YES, CLEAR THE OVERFLOW FLAG
   272  000642' 012767  177775  001672            MOV     #-3,BLK1          ; RESET THE BLOCK NUMBER
   273  000650' 000676                           BR      STRT              ; START OVER AT BEGINNING OF DISK
   274  000652' 004567  000210          GOA:    JSR     R5,WRITCK         ; WRITE-CHECK THE DATA
   275  000656' 000430                           BR      RETRY2            ; IF ERRORS, TRY AGAIN
   276  000660' 004567  000234          GOB:    JSR     R5,READ           ; READ THE DATA WRITTEN
   277  000664' 000437                           BR      RETRY3            ; IF ERRORS, TRY AGAIN
   278  000666' 104412  000000' 000126'          CDATA$,BEGIN,RBUFPA       ; REQUEST FOR MONITOR TO CHECK DATA
   279  000674' 000676'                          .+2                       ; IF ERROR, CONTINUE
   280
   281
   283  000702'                          PASS:
   284  000702' 104413  000000'                  ENDIT$,BEGIN              ;SIGNAL END OF ITERATION.
   285                                                                     ;MONITOR SHALL TEST END OF PASS
   286  000706' 000674                           BR      NEXT
   287
   288  000710'                          FINI:
   289  000710' 104410  000000'                  END$,BEGIN                ; DROP THE MODULE
   290                                   ;       -------------------------
   291
   292
   293
   294  000714' 105267  003122          RETRY1: INCB    TRY1              ; COUNT THE RETRYS
   295  000720' 122767  000003  003114           CMPB    #3,TRY1           ; LIMIT EXCEEDED ?
   296  000726' 001333                           BNE     GO                ; NO, GO TRY IT AGAIN
   297  000730' 104403  000000' 003772'          MSGN$,BEGIN,EXCED1        ;ASCII MESSAGE CALL WITH COMMON HEADER
   298  000736' 000423                           BR      NEXTA             ; GO ON TO NEXT DRIVE
   299                                   ;       -------------------------
   300  000740' 105267  003077          RETRY2: INCB    TRY2              ; COUNT RETRYS
   301  000744' 122767  000003  003071           CMPB    #3,TRY2           ; LIMIT EXCEEDED ?
   302  000752' 001337                           BNE     GOA               ; NO, TRY AGAIN
   303  000754' 104403  000000' 004000'          MSGN$,BEGIN,EXCED2        ;ASCII MESSAGE CALL WITH COMMON HEADER
   304  000762' 000411                           BR      NEXTA             ; GO ON TO NEXT DRIVE
   305                                   ;       -------------------------
   306  000764' 105267  003054          RETRY3: INCB    TRY3              ; COUNT RETRYS
   307  000770' 122767  000003  003046           CMPB    #3,TRY3           ; LIMIT EXCEEDED ?
   308  000776' 001330                           BNE     GOB               ; NO, GO TRY AGAIN
   309  001000' 104403  000000' 004006'          MSGN$,BEGIN,EXCED3        ;ASCII MESSAGE CALL WITH COMMON HEADER
   310                                   ;       -------------------------
   311  001006' 032767  000001  177002  NEXTA:  BIT     #BIT0,SR1         ; DROP THE DRIVE ?
   312  001014' 001405                           BEQ     1$                ; NO, SKIP TO NEXT DRIVE
   313  001016' 004767  000264                   JSR     PC,DROP           ; YES, DROP OFFENDING DRIVE
   314  001022' 104403  000000' 004014'          MSGN$,BEGIN,DRP ;ASCII MESSAGE CALL WITH COMMON HEADER
   315  001030' 000167  177444          1$:     JMP     NEXT              ; GO ON TO NEXT DRIVE
   316                                   ;       -------------------------
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 10
RKAA.P11    13-MAR-78 00:00
   317
   318
   319                                  ;              -----  RK11 DISK DRIVERS  -----
   320
   321  001034' 012767  000503  001462  WRITE:  MOV     #503,FUNC        ; LOAD WRITE FUNCTION
   322  001042' 016777  001504  002520          MOV     WCNT1,@RKWC      ; LOAD WORD COUNT
   323  001050' 016777  177060  002514          MOV     WBUFPA,@RKBA     ; LOAD BUFFER ADDRESS
   324  001056' 016767  177054  001442          MOV     WBUFEA,XMEM      ; LOAD EXTENDED MEMORY BITS
   325  001064' 000462                          BR      GOGO             ; CONTINUE
   326  001066' 012767  000507  001430  WRITCK: MOV     #507,FUNC        ; LOAD WRITE-CHECK FUNCTION
   327  001074' 016777  001452  002466          MOV     WCNT1,@RKWC      ; LOAD WORD COUNT
   328  001102' 016777  177026  002462          MOV     WBUFPA,@RKBA     ; LOAD BUFFER ADDRESS
   329  001110' 016767  177022  001410          MOV     WBUFEA,XMEM      ; LOAD EXTENDED MEMORY BITS
   330  001116' 000445                          BR      GOGO             ; CONTINUE
   331  001120' 012767  000505  001376  READ:   MOV     #505,FUNC        ; LOAD READ FUNCTION
   332  001126' 016777  001422  002434          MOV     WCNT2,@RKWC      ; LOAD WORD COUNT
   333  001134' 016777  176766  002430          MOV     RBUFPA,@RKBA     ; LOAD BUFFER ADDRESS
   334  001142' 016767  176762  001356          MOV     RBUFEA,XMEM      ; LOAD EXTENDED MEMORY BITS
   335  001150' 000430                          BR      GOGO             ; CONTINUE
   336
   337  001152' 012777  000001  002406  CLEAR:  MOV     #1,@RKCS         ; ISSUE A CONTROL RESET
   338  001160' 004767  001260                  JSR     PC,WAIT1         ; GO WAIT FOR CONTROLLER READY
   339  001164' 016777  001350  002402          MOV     DRVSFT,@RKDA     ; RELOAD THE DRIVE ADDRESS
   340  001172' 032777  000100  002362          BIT     #BIT6,@RKDS      ; DRIVE READY ?
   341  001200' 001001                          BNE     2$               ; YES, CONTINUE
   342  001202' 000205                          RTS     R5               ; NO, ABORT DRIVE RESET
   343  001204' 012777  000015  002354  2$:     MOV     #15,@RKCS        ; ISSUE A DRIVE RESET
   344  001212' 004767  000530                  JSR     PC,WAIT          ; GIVE IT TIME TO COMPLETE
   345  001216' 012777  000001  002342          MOV     #1,@RKCS         ; ISSUE ANOTHER CONTROLLER RESET
   346  001224' 004767  001214                  JSR     PC,WAIT1         ; WAIT FOR CONTROLLER READY
   347  001230' 000205                          RTS     R5               ; RETURN
   348
   349  001232' 012777  001266' 176550  GOGO:   MOV     #NTRUPT,@VECTOR ; SET INTERUPT ENTRY POINTER
   350  001240' 016777  001264  002326          MOV     DSKADR,@RKDA     ; LOAD THE DISK ADDRESS
   351  001246' 056767  001254  001250          BIS     XMEM,FUNC        ; LOAD EXTENDED MEMORY BITS
   352  001254' 016777  001244  002304          MOV     FUNC,@RKCS       ; EXECUTE THE FUNCTION
   353  001262' 104400  000000'                 EXIT$,BEGIN                     ;EXIT TO MONITOR. MODULE WAIT FOR INTERRUPT.
   354
   355  001266'                         NTRUPT:
   356                                          ;-----------------------------------------------------------------
   357  001266' 000004  000000' 001274'         PIRQ$,BEGIN,1$           ; QUEUE UP TO CONTINUE AT 1$ AND RTI
   358                                          ;-----------------------------------------------------------------
   359
   360  001274' 004567  000516          1$:     JSR     R5,ERRORS        ; GO CHECK FOR ERRORS
   361  001300' 000205                          RTS     R5               ; ERRORS DETECTED, RETURN
   362  001302' 005725                          TST     (R5)+            ; NO ERRORS, SKIP RETRY
   363  001304' 000205                          RTS     R5               ; RETURN OK
   364                                  ;       -------------------------
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 11
RKAA.P11    13-MAR-78 00:00
   365
   366
   367  001306' 012701  000001          DROP:   MOV     #1,R1           ; INITIALIZE DROP PICKER
   368  001312' 016700  001220                  MOV     DRYVE,R0        ; GET THE DRIVE NUMBER
   369  001316' 001403                          BEQ     2$              ; IF DRIVE 0 GO DROP IT
   370  001320' 006301                  1$:     ASL     R1              ; NO, AIM AT THE NEXT DRIVE
   371  001322' 005300                          DEC     R0              ; IS THIS THE ONE ?
   372  001324' 001375                          BNE     1$              ; NO, LOOK AGAIN
   373  001326' 040167  001200          2$:     BIC     R1,DVICE        ; DROP THE DRIVE
   375                                          ;*****************************************************************
   376                                          ;CONVERT DRYVE TO ASCII AND
   377                                          ;STORE AT ADR1
   378  001336' 104420  000000' 002536'         QTOA$,BEGIN,DRYVE,ADR1
   379  001344' 004030
   380                                          ;*****************************************************************
   381  001346' 000207                          RTS     PC              ; RETURN
   382                                  ;       ------------------------
   383
   384
   385  001350' 062767  000003  001164  BLOCK:  ADD     #3,BLK1         ; STEP TO NEXT BLOCK
   386  001356' 022767  011277  001156          CMP     #4799.,BLK1     ; BLOCK LIMIT REACHED ?
   387  001364' 100002                          BPL     1$              ; NO, CONTINUE
   388  001366' 005067  001150                  CLR     BLK1            ; YES, RESET BLOCK #
   389  001372' 016767  001144  001144  1$:     MOV     BLK1,BLK2       ; READ WHERE WRITE
   390  001400' 000207                          RTS     PC              ; RETURN
   391                                  ;       ------------------------
   392
   393
   394  001402' 016700  001134          ROOM:   MOV     BLK1,R0         ; SAVE THE CURRENT BLOCK NUMBER
   395  001406' 012701  004537                  MOV     #2399.,R1       ; LOAD MAX. NUMBER OF BLOCK PER SIDE
   396  001412' 005002                          CLR     R2              ; ZERO REG. 2
   397  001414' 022700  004537                  CMP     #2399.,R0       ; IS SIDE 0 DONE ?
   398  001420' 002002                          BGE     1$              ; NO, CONTINUE
   399  001422' 162700  004540                  SUB     #2400.,R0       ; YES, NORMALIZE BLOCK # FOR SIDE 1
   400  001426' 012767  000400  001112  1$:     MOV     #256.,BSIZ      ; HI DENSITY BLOCK SIZE
   401  001434' 032777  004000  002120          BIT     #BIT11,@RKDS    ; HI DENSITY DRIVE ?
   402  001442' 001002                          BNE     2$              ; YES, CONTINUE
   403  001444' 006267  001076                  ASR     BSIZ            ; NO, SET TO 128 -- LO DENSITY
   404  001450' 160001                  2$:     SUB     R0,R1           ; GET # OF BLOCKS LEFT ON DISK
   405  001452' 066702  001070          3$:     ADD     BSIZ,R2         ; GET TOTAL NUMBER OF WORDS LEFT
   406  001456' 005301                          DEC     R1              ; ALL BLOCKS ADDED IN ?
   407  001460' 003374                          BGT     3$              ; NO, KEEP ADDING
   408  001462' 005702                          TST     R2              ; IS # OF WORDS LEFT ON DISK NEG. ?
   409  001464' 100404                          BMI     4$              ; YES
   410  001466' 005767  176450                  TST     WBUFSZ          ; IS TRANSFER SIZE NEG. ?
   411  001472' 100411                          BMI     7$              ; YES
   412  001474' 000403                          BR      5$              ; NO, GO COMPARE
   413  001476' 005767  176440          4$:     TST     WBUFSZ          ; IS TRANSFER SIZE POS. ?
   414  001502' 100003                          BPL     6$              ; YES
   415  001504' 020267  176432          5$:     CMP     R2,WBUFSZ       ; WAS THERE ENOUGH ROOM FOR THE TRANSFER ?
   416  001510' 002402                          BLT     7$              ; NO, RETURN OK
   417  001512' 005725                  6$:     TST     (R5)+           ; YES, MUST BE A REAL ERROR
   418  001514' 000205                          RTS     R5              ; RETURN, ERROR
   419  001516' 152767  000004  002315  7$:     BISB    #BIT2,FLAG      ; SET OVERFLOW FLAG
```

```
420  001524' 000205                    RTS     R5              ; RETURN OK
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 12
RKAA.P11    13-MAR-78 00:00
   421                                  ;       ------------------------
   422
   423
   424  001526' 005267  001004          DRVADR: INC     DRYVE           ; COUNT A DRIVE
   425  001532' 062767  020000  001000          ADD     #BIT13,DRVSFT   ; DRIVE COUNT LINED UP WITH RKDA
   426  001540' 142767  000010  002273          BICB    #BIT3,FLAG      ; CLEAR END OF DRIVES FLAG
   427  001546' 022767  000010  000762          CMP     #8.,DRYVE       ; ALL DRIVES CHECKED ?
   428  001554' 001404                          BEQ     1$              ; YES, GO FLAG END OF DRIVES
   429  001556' 006267  000752                  ASR     DRIVE           ; NO, IS NEXT DRIVE CHOSEN ?
   430  001562' 103361                          BCC     DRVADR          ; NO, GO TRY ANOTHER DRIVE
   431  001564' 000207                          RTS     PC              ; RETURN
   432
   433  001566' 152767  000010  002245  1$:     BISB    #BIT3,FLAG      ; SET END OF DRIVES FLAG
   434  001574' 012767  177777  000734          MOV     #-1,DRYVE       ; RESET DRIVE COUNTER
   435  001602' 012767  160000  000730          MOV     #160000,DRVSFT  ; ZERO THE SHIFTED DRIVE #
   436  001610' 016767  000716  000716          MOV     DVICE,DRIVE     ; RESTORE CHOSEN DRIVES
   437  001616' 000207                          RTS     PC              ; RETURN
   438                                  ;       ------------------------
   439
   440
   441  001620' 012767  177777  000710  NOTRDY: MOV     #-1,DRYVE       ; START WITH FIRST DRIVE
   442  001626' 012767  160000  000704          MOV     #160000,DRVSFT  ;
   443  001634' 016767  000672  000672          MOV     DVICE,DRIVE     ; RESET DRIVE SELECT
   444  001642' 004767  177660          1$:     JSR     PC,DRVADR       ; GET A DRIVE ADDRESS
   445  001646' 132767  000010  002165          BITB    #BIT3,FLAG      ; ALL DRIVES CHECKED ?
   446  001654' 001012                          BNE     2$              ; YES, RETURN
   447  001656' 016777  000656  001710          MOV     DRVSFT,@RKDA    ; NO, LOAD NEXT DRIVE ADDRESS
   448  001664' 032777  000100  001670          BIT     #BIT6,@RKDS     ; IS THIS DRIVE READY ?
   449  001672' 001363                          BNE     1$              ; YES, CONTINUE
   450  001674' 004767  000046                  JSR     PC,WAIT         ; NO, WAIT FOR IT
   451  001700' 000760                          BR      1$              ; GO CHECK REST OF DRIVES
   452  001702' 000207                  2$:     RTS     PC              ; RETURN
   453                                  ;       ------------------------
   454
   455
   456  001704' 014167  176176          ERSUB2: MOV     -(R1),ASB       ; LOAD THE DATA
   457  001710' 010167  176166                  MOV     R1,SBADR        ; LOAD ADDRESS OF DATA WRITTEN
   458  001714' 014267  176170                  MOV     -(R2),AWAS      ; LOAD THE DATA
   459  001720' 010267  176160                  MOV     R2,WASADR       ; LOAD ADDRESS OF DATA READ
   460  001724' 005721                          TST     (R1)+           ; RESET REG. 1
   461  001726' 005722                          TST     (R2)+           ; RESET REG. 2
   462
   463  001730' 016767  001632  176142  ERSUB1: MOV     RKCS,CSRA       ; LOAD ADR. OF CURRENT CSR
   464  001736' 017767  001624  176136          MOV     @RKCS,ACSR      ; LOAD CONTENTS OF CURRENT CSR
   465  001744' 000207                          RTS     PC              ; RETURN
   466                                  ;       ------------------------
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 14
RKAA.P11    13-MAR-78 00:00
   467
   468
   469
   470
   471  001746' 012767  077777  001604  WAIT:   MOV     #77777,CLK      ; SET THE TIMER
   472  001754'                         1$:
   473  001754' 104407  000000'                 BREAK$,BEGIN            ;TEMPORARY RETURN TO MONITOR....
   474  001760' 104407  000000'                 BREAK$,BEGIN            ;THEN CONTINUE AT NEXT INSTRUCTION.
   475  001764' 032777  000100  001570          BIT     #BIT6,@RKDS     ; DRIVE READY ?
   476  001772' 001010                          BNE     2$              ; YES, RETURN
   477  001774' 005367  001560                  DEC     CLK             ; NO, WAIT SOME MORE ?
   478  002000' 001365                          BNE     1$              ; YES, WAIT
   479  002002' 004767  177300                  JSR     PC,DROP         ; TIME-OUT, DROP THE DRIVE
   480  002006' 104403  000000' 004014'         MSGN$,BEGIN,DRP ;ASCII MESSAGE CALL WITH COMMON HEADER
   481  002014' 000207                  2$:     RTS     PC              ; RETURN
   482                                  ;       ------------------------
   483  002016' 004767  177706          ERRORS: JSR     PC,ERSUB1       ; LOAD ERROR INFORMATION
   484  002022' 032777  040000  001536          BIT     #BIT14,@RKCS    ; HARD ERROR ?
   485  002030' 001006                          BNE     1$              ; YES, GO REPORT
   486  002032' 032777  000003  001524          BIT     #3,@RKER        ; SOFT ERROR ?
   487  002040' 001041                          BNE     3$              ; YES, GO REPORT
   488  002042' 005725                          TST     (R5)+           ; NO, SKIP RETRY
   489  002044' 000205                          RTS     R5              ; RETURN OK
   490  002046' 032777  040000  001510  1$:     BIT     #BIT14,@RKER    ; DISK OVERFLOW ?
   491  002054' 001403                          BEQ     7$              ; NO, CONTINUE
   492  002056' 004567  177320                  JSR     R5,ROOM         ; YES, IS IT A REAL ERROR ?
   493  002062' 000442                          BR      5$              ; NO, CONTINUE
   494  002064' 032777  001000  001472  7$:     BIT     #BIT9,@RKER     ;DATA LATE ERROR?
   495  002072' 001411                          BEQ     2$              ;NO
   496  002074' 005267  000420                  INC     DLTCNT          ;INCREMENT ERROR COUNTER
   497  002100' 032767  000004  175710          BIT     #BIT2,SR1       ;TYPE OUT ERROR?
   498  002106' 001013                          BNE     6$              ;NO
   499  002110' 104403  000000' 004024'         MSGN$,BEGIN,DLTERR      ;ASCII MESSAGE CALL WITH COMMON HEADER
   500  002116'                         2$:
   501  002116' 104403  000000' 003762'         MSGN$,BEGIN,HARD        ;ASCII MESSAGE CALL WITH COMMON HEADER
   502  002124' 005067  175756                  CLR     ERRTYP
   503                                          ;*************************************************************
   504  002130' 104405  000000' 003562'         HRDER$,BEGIN,TABLE      ;
   505                                          ;*************************************************************
   506  002136' 004567  177010          6$:     JSR     R5,CLEAR        ; GO CLEAR OUT ERRORS
   507  002142' 000411                          BR      4$              ; RETURN
   508  002144'                         3$:
   509  002144' 104403  000000' 003766'         MSGN$,BEGIN,SOFT        ;ASCII MESSAGE CALL WITH COMMON HEADER
   510  002152' 012767  000001  175726          MOV     #1,ERRTYP       ; DATA ERROR
   511                                          ;*************************************************************
   512  002160' 104406  000000' 003562'         SOFER$,BEGIN,TABLE      ;
   513                                          ;*************************************************************
   514  002166' 000205                  4$:     RTS     R5              ; RETURN, ERRORS
   515  002170' 004567  176756          5$:     JSR     R5,CLEAR        ; CLEAR OUT ERRORS
   516  002174' 005725                          TST     (R5)+           ; SKIP RETRY
   517  002176' 000205                          RTS     R5              ; RETURN OK
   518                                  ;       ------------------------
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 15
RKAA.P11    13-MAR-78 00:00
   519
   520
   521  002200' 016700  175602          SETUP:  MOV     ADDR,R0         ; GET DEVICE ADDRESS
   522  002204' 010067  001352                  MOV     R0,RKDS         ; GENERATE CONTROLLER REGS. ADDRESSES
   523  002210' 005720                          TST     (R0)+
   524  002212' 010067  001346                  MOV     R0,RKER
   525  002216' 005720                          TST     (R0)+
   526  002220' 010067  001342                  MOV     R0,RKCS
   527  002224' 005720                          TST     (R0)+
   528  002226' 010067  001336                  MOV     R0,RKWC
   529  002232' 005720                          TST     (R0)+
   530  002234' 010067  001332                  MOV     R0,RKBA
   531  002240' 005720                          TST     (R0)+
   532  002242' 010067  001326                  MOV     R0,RKDA
   533  002246' 005720                          TST     (R0)+
   534  002250' 010067  001322                  MOV     R0,RKMR
   535  002254' 005720                          TST     (R0)+
   536  002256' 010067  001316                  MOV     R0,RKDB
   537
   538  002262' 016700  175522                  MOV     VECTOR,R0       ; GET THE VECTOR ADDRESS
   539  002266' 012720  000446'                 MOV     #STRT,(R0)+     ; SET POINTER JUST IN CASE
   540  002272' 116710  175514                  MOVB    BR1,(R0)        ; SET PRIORITY
   541
   542  002276' 000207                  2$:     RTS     PC              ; RETURN
   543                                  ;       ------------------------
   544
   545
   546  002300' 005001                  CONVRT: CLR     R1              ; ZERO REG. 1
   547  002302' 105067  001532                  CLRB    SIDE            ; ZERO THE SIDE INDICATOR
   548  002306' 012703  177764                  MOV     #-12.,R3        ; LOAD REG. 3
   549  002312' 012704  000013                  MOV     #11.,R4         ; LOAD REG. 4
   550  002316' 022700  004537                  CMP     #2399.,R0       ; IS BLOCK ON SIDE 0 ?
   551  002322' 002005                          BGE     1$              ; YES, CONTINUE
   552  002324' 152767  000020  001506          BISB    #BIT4,SIDE      ; NO, FLIP TO SIDE 1
   553  002332' 062700  173240                  ADD     #-2400.,R0      ; NORMALIZE BLOCK # FOR SIDE 1
   554  002336' 020400                  1$:     CMP     R4,R0           ; FIND THE RIGHT CYLINDER ?
   555  002340' 002003                          BGE     2$              ; YES, CONTINUE
   556  002342' 060300                          ADD     R3,R0           ; NO, SUBTRACT 12 SECTORS (1 CYLINDER)
   557  002344' 005201                          INC     R1              ; KEEP TRACK OF CYLINDER ADDRESS
   558  002346' 000773                          BR      1$              ; GO TRY AGAIN
   559  002350' 010067  000154          2$:     MOV     R0,DSKADR       ; LOAD THE SECTOR ADDRESS
   560  002354' 156767  001460  000146          BISB    SIDE,DSKADR     ; LOAD THE SIDE ADDRESS
   561  002362' 012702  000005                  MOV     #5,R2           ; SET UP FOR SHIFT
   562  002366' 006301                  3$:     ASL     R1              ; LINE UP CYL. ADR. WITH DSKADR
   563  002370' 005302                          DEC     R2              ; DONE ?
   564  002372' 003375                          BGT     3$              ; NO, GO SHIFT AGAIN
   565  002374' 050167  000130                  BIS     R1,DSKADR       ; YES, LOAD THE CYLINDER ADDRESS
   566  002400' 000207                          RTS     PC              ; RETURN
   567                                  ;       ------------------------
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 16
RKAA.P11    13-MAR-78 00:00
   568  002402' 012777  000001  001156  REZET:  MOV     #1,@RKCS        ; EXECUTE CONTROLLER RESET
   569  002410' 004767  000030           	JSR     PC,WAIT1        ; GO WAIT FOR CONTROLLER READY
   570  002414' 004767  177200           	JSR     PC,NOTRDY       ; MAKE SURE ALL CHOSEN DRIVES ARE READY
   571  002420' 004767  177102           1$:     JSR     PC,DRVADR       ; GET A DRIVE ADDRESS
   572  002424' 132767  000010  001407           BITB    #BIT3,FLAG      ; ALL DRIVES DONE ?
   573  002432' 001003                   	BNE     2$              ; YES, RETURN
   574  002434' 004567  176512           	JSR     R5,CLEAR        ; ISSUE DRIVE RESET AND CONTROLLER CLEAR
   575  002440' 000767                   	BR      1$              ; KEEP GOING
   576  002442' 000207                   2$:     RTS     PC              ; RETURN
   577                                   ;       ------------------------
   578
   579
   580  002444' 012767  077777  001106  WAIT1:  MOV     #77777,CLK      ; SET THE TIMER
   581  002452' 105777  001110           1$:     TSTB    @RKCS           ; CONTROLLER READY ?
   582  002456' 100417                   	BMI     2$              ; YES, CONTINUE
   583  002460' 104407  000000'          	BREAK$,BEGIN            ;TEMPORARY RETURN TO MONITOR....
   584  002464' 104407  000000'          	BREAK$,BEGIN            ;THEN CONTINUE AT NEXT INSTRUCTION.
   585  002470' 005367  001064           	DEC     CLK             ; WAIT SOME MORE ?
   586  002474' 001366                   	BNE     1$              ; YES
   587  002476' 012767  000003  175402   	MOV     #3,ERRTYP       ; CONTROLLER NOT READY
   588                                   	;*************************************************************
   589  002504' 104405  000000' 003562'  	HRDER$,BEGIN,TABLE      ; CONTROLLER NOT READY
   590                                   	;*************************************************************
   591  002512' 000167  176172           	JMP     FINI            ; GO DROP THE MODULE
   592  002516' 000207                   2$:     RTS     PC              ; READY, RETURN
   593                                   ;       ------------------------
   594
   595  002520' 000000                   DLTCNT: 0
   597  002524' 000000                   FUNC:   0
   598  002526' 000000                   XMEM:   0
   599  002530' 000000                   DSKADR: 0
   600  002532' 000000                   DVICE:  0
   601  002534' 000000                   DRIVE:  0
   602  002536' 000000                   DRYVE:  0
   603  002540' 000000                   DRVSFT: 0
   604  002542' 000000                   BLK1:   0
   605  002544' 000000                   BLK2:   0
   606  002546' 000000                   BSIZ:   0
   607  002550' 000000                   TBUF:   0
   608  002552' 000000                   WCNT1:  0
   609  002554' 000000                   WCNT2:  0
   610  002556' 000400                   BUFLEN: 256.
   611  002560' 000400                   BUFIN:  .BLKW   256.
   612  003560' 000000                   CLK:    0
   613  003562'                          TABLE:
   614  003562' 000000                   RKDS:   0
   615  003564' 000000                   RKER:   0
   616  003566' 000000                   RKCS:   0
   617  003570' 000000                   RKWC:   0
   618  003572' 000000                   RKBA:   0
   619  003574' 000000                   RKDA:   0
   620  003576' 000000                   RKMR:   0
   621  003600' 000000                   RKDB:   0
   622  003602' 177777                   	177777
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 17
RKAA.P11    13-MAR-78 00:00
   623  003604' 020040  044040  051101  MES1:    .ASCIZ  '   HARD ERROR'
   624  003612' 020104  051105  047522
   625  003620' 000122
   626  003622' 020040  051440  043117  MES2:    .ASCIZ  '   SOFT ERROR'
   627  003630' 020124  051105  047522
   628  003636' 000122
   629  003640' 020040  051104  053111  MES4:    .ASCIZ  '  DRIVE  '
   630  003646' 020105  000040
   631  003652' 020040  051104  050117  MES5:    .ASCIZ  '  DROPPED%'
   632  003660' 042520  022504     000
   633  003665'    040  042522  051124  MES6:    .ASCIZ  ' RETRY EXCEEDED%'
   634  003672' 020131  054105  042503
   635  003700' 042105  042105  000045
   636  003706' 053440  044522  042524  MES7:    .ASCIZ  ' WRITE'
   637  003714'    000
   638  003715'    040  051127  052111  MES8:    .ASCIZ  ' WRITE-CHECK'
   639  003722' 026505  044103  041505
   640  003730' 000113
   641  003732' 051040  040505  000104  MES9:    .ASCIZ  ' READ'
   642  003740' 040504  040524  046040  MES10:   .ASCIZ  'DATA LATE ERROR%'
   643  003746' 052101  020105  051105
   644  003754' 047522  022522     000
   645          003762'                        .EVEN
   646  003762' 003604'                HARD:   MES1
   647  003764' 177777                         177777
   648  003766' 003622'                SOFT:   MES2
   649  003770' 177777                         177777
   650  003772' 003706'                EXCED1: MES7
   651  003774' 003665'                        MES6
   652  003776' 177777                         177777
   653  004000' 003715'                EXCED2: MES8
   654  004002' 003665'                        MES6
   655  004004' 177777                         177777
   656  004006' 003732'                EXCED3: MES9
   657  004010' 003665'                        MES6
   658  004012' 177777                         177777
   659  004014' 003640'                DRP:    MES4
   660  004016' 004035'                        NUMB
   661  004020' 003652'                        MES5
   662  004022' 177777                         177777
   663  004024' 003740'                DLTERR: MES10
   664  004026' 177777                         177777
   665  004030' 000005                ADR1:   .BLKB   5
   666  004035'    000                NUMB:   .BYTE   0
   667  004036' 000000                         .WORD   0
   668  004040'    000                SIDE:   .BYTE   0
   669  004041'    000                FLAG:   .BYTE   0
   670                                        .EVEN
   671  004042'    000                TRY1:   .BYTE   0
   672  004043'    000                TRY2:   .BYTE   0
   673  004044'    000                TRY3:   .BYTE   0
   674          004046'                       .EVEN
   675
   676          000001                        .END
```

RKAA DEC/X11 SYSTEM EXERCISER MODULE MACY11 30A(1052) 20-MAR-78 15:47 PAGE 19
RKAA.P11 13-MAR-78 00:00 CROSS REFERENCE TABLE -- USER SYMBOLS

```
ACSR    000102R          178#    464*
ADDR    000006R          145#    521
ADR1    004030R          378     665#
ASB     000106R          182#    456*
ASTAT   000104R          180#
AWAS    000110R          183#    458*
BEGIN   000000R          142#    235     240     257     278     284     289     297     303     309     314     353     357
                         378     473     474     480     499     501     504     509     512     583     584     589
BIT0  = 000001           207#    221     311
BIT1  = 000002           207#    219
BIT10 = 002000           207#
BIT11 = 004000           207#    401
BIT12 = 010000           207#
BIT13 = 020000           207#    425
BIT14 = 040000           207#    484     490
BIT15 = 100000           207#
BIT2  = 000004           207#    269     271     419     497
BIT3  = 000010           207#    249     426     433     445     572
BIT4  = 000020           207#    552
BIT5  = 000040           207#    254
BIT6  = 000100           207#    259     340     448     475
BIT7  = 000200           207#
BIT8  = 000400           207#
BIT9  = 001000           207#    494
BLK1    002542R          216*    243     272*    385*    386     388*    389     394     604#
BLK2    002544R          389*    605#
BLOCK   001350R          239     385#
BREAK$= 104407           207#    473     474     583     584
BR1     000012R          147#    540
BR2     000013R          148#
BSIZ    002546R          400*    403*    405     606#
BTOD$ = 104421           207#
BUFIN   002560R          189     611#
BUFLEN  002556R          610#
CDATA$= 104412           207#    278
CDERCT  000144R          197#
CDWDCT  000146R          198#
CLEAR   001152R          337#    506     515     574
CLK     003560R          471*    477*    580*    585*    612#
CNT     002522R          208*    231     282*    596#
CONVRT  002300R          244     546#
CSRA    000100R          176#    463*
DATCK$= 104411           207#
DATER$= 104404           207#
DLTCNT  002520R          213*    496*    595#
DLTERR  004024R          499     663#
DRIVE   002534R          215*    429*    436*    443*    601#
DROP    001306R          223     256     313     367#    479
DRP     004014R          257     314     480     659#
DRVADR  001526R          246     424#    430     444     571
DRVSFT  002540R          218*    252     339     425*    435*    442*    447     603#
DRYVE   002536R          217*    225*    368     374     424*    427     434*    441*    602#
DSKADR  002530R          251*    252*    253     350     559*    560*    565*    599#
DVICE   002532R          214*    215     221     228     247     373*    436     443     600#
```

```
DVID1    000014R          149#     214
ENDIT$=  104413           207#     284
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 20
RKAA.P11    13-MAR-78 00:00             CROSS REFERENCE TABLE -- USER SYMBOLS
END$  = 104410            207#    289
ERRORS  002016R           360     483#
ERRTYP  000106R           181#    502*    510*    587*
ERSUB1  001730R           463#    483
ERSUB2  001704R           456#
EXCED1  003772R           297     650#
EXCED2  004000R           303     653#
EXCED3  004006R           309     656#
EXIT$ = 104400            207#    353
FINI    000710R           229     248     288#    591
FLAG    004041R           249     269     271*    419*    426*    433*    445     572     669#
FREE    000150R           199#
FUNC    002524R           321*    326*    331*    351*    352     597#
GETPA$= 104415            207#    235
GO      000616R           267#    296
GOA     000652R           270     274#    302
GOB     000660R           276#    308
GOGO    001232R           325     330     335     349#
GWBUF$= 104414            207#    240
HARD    003762R           501     646#
HRDCNT  000044R           162#
HRDER$= 104405            207#    504     589
HRDPAS  000050R           164#
ICONT   000036R           159#
ICOUNT  000040R           160#
IDNUM   000122R           188#
IMODX.= 000000            200#    241
INIT    000030R           156#
INTR    000120R           187#    211*
MAP22$= 104416            207#
MES1    003604R           623#    646
MES10   003740R           642#    663
MES2    003622R           626#    648
MES4    003640R           629#    659
MES5    003652R           631#    661
MES6    003665R           633#    651     654     657
MES7    003706R           636#    650
MES8    003715R           638#    653
MES9    003732R           641#    656
MODNAM  000000R           143#
MODSP   000252R           157     205#
MSGN$ = 104403            207#    257     297     303     309     314     480     499     501     509
MSGS$ = 104402            207#
MSG$  = 104401            207#
NEXT    000500R           246#    258     286     315
NEXTA   001006R           298     304     311#
NOTRDY  001620R           261     441#    570
NTRUPT  001266R           349     355#
NULL  = 000000            207#
NUMB    004035R           660     666#
OPEN  = 000000            144     150     151     152     153     169     170     171     172     173     174     175     176
                          178     180     182     183     185     186     187     190     191     193     194     196     197
                          198     199     207#
OTOA$ = 104420            207#    378
```

```
PASCNT  000034R          158#
PASS    000702R          283#
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 21
RKAA.P11    13-MAR-78 00:00             CROSS REFERENCE TABLE -- USER SYMBOLS
PIRQ$ = 000004              207#     357
POPSP = 005726              207#
POPSP2= 022626              207#
PRTY  = 000000              207#
PRTY0 = 000000              148      207#
PRTY1 = 000040              207#
PRTY2 = 000100              207#
PRTY3 = 000140              207#
PRTY4 = 000200              207#
PRTY5 = 000240              147      207#
PRTY6 = 000300              207#
PRTY7 = 000340              207#
PS    = 177776              207#
PSW   = 177776              207#
PUSH  = 005746              207#
PUSH2 = 024646              207#
RAND$ = 104417              207#
RANNUM  000054R             166#
RBUFEA  000130R             191#     334
RBUFPA  000126R             190#     278      333
RBUFSZ  000132R             192#     236
RBUFVA  000124R             189#     235
READ    001120R             276      331#
RESTRT  000416R             184      231#
RES1    000056R             167#
RES2    000060R             168#
RETRY1  000714R             268      294#
RETRY2  000740R             275      300#
RETRY3  000764R             277      306#
REZET   002402R             227      568#
RKBA    003572R             323*     326*     333*     530*     618#
RKCS    003566R             337*     343*     345*     352*     463      464      484      526*     568*     581      616#
RKDA    003574R             253*     339*     350*     447*     532*     619#
RKDB    003600R             536*     621#
RKDS    003562R             254      259      340      401      448      475      522*     614#
RKER    003564R             486      490      494      524*     615#
RKMR    003576R             534*     620#
RKWC    003570R             322*     327*     332*     528*     617#
ROOM    001402R             394#     492
RSTRT   000112R             184#
RSTRT1  000426R             230      232      234#
SBADR   000102R             177#     457*
SETUP   002200R             226      521#
SIDE    004040R             212*     547*     552*     560      668#
SOFCNT  000042R             161#
SOFER$= 104406              207#     512
SOFPAS  000046R             163#
SOFT    003766R             509      648#
SPOINT  000032R             157#
SPSIZ = 000040                1#     200
SR1     000016R             150#     311      497
SR2     000020R             151#
SR3     000022R             152#
SR4     000024R             153#
```

```
START   000252R          156     208#    233
STAT    000026R          155#
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 22
RKAA.P11    13-MAR-78 00:00             CROSS REFERENCE TABLE -- USER SYMBOLS
STRT     000446R           239#     250      262      273      539
SVR0     000062R           169#
SVR1     000064R           170#
SVR2     000066R           171#
SVR3     000070R           172#
SVR4     000072R           173#
SVR5     000074R           174#
SVR6     000076R           175#
SYSCNT   000052R           165#
TABLE    003562R           504      512      589      613#
TBUF     002550R           607#
TRPDFD=  000022            207#
TRY1     004042R           263*     294*     295      671#
TRY2     004043R           300*     301      672#
TRY3     004044R           264*     306*     307      673#
VECTOR   000010R           146#     349*     538
WAIT     001746R           344      450      471#
WAIT1    002444R           338      346      569      580#
WASADR   000104R           179#     459*
WBUFEA   000136R           194#     324      329
WBUFPA   000134R           193#     323      328
WBUFRQ   000140R           195#
WBUFSZ   000142R           196#     241      410      413      415
WCNT1    002552R           241*     242*     322      327      608#
WCNT2    002554R           236*     237*     332      609#
WDFR     000116R           186#     210*
WDTO     000114R           185#     209*
WRITCK   001066R           274      326#
WRITE    001034R           267      321#
XFLAG    000005R           144#
XMEM     002526R           324*     329*     334*     351      598#
.      = 004046R           279      611#     645#     665#     674#
```

```
RKAA DEC/X11 SYSTEM EXERCISER MODULE    MACY11 30A(1052)  20-MAR-78  15:47  PAGE 24
RKAA.P11     13-MAR-78 00:00            CROSS REFERENCE TABLE -- MACRO NAMES
BKMOD        1#
BREAK        1#     472      583
BTOD         1#
CKDATA       1#     278
DATACK       1#
DATERR       1#
DFSEVN       1#     207
DSEVNT       1#     207
END          1#     288
ENDIT        1#     283
EQUATS       1#     207
EXIT         1#     353
GETPA        1#     235
GWBUFF       1#     240
HRDER        1#     503      588
IOMOD        1#
IOMODP       1#
IOMODR       1#
IOMODX       1#     136
MAP22        1#
MODULE       1#     137
MSG          1#
MSGN         1#     257      297      303      309      314      480      499      500      508
MSGS         1#
NBKMOD       1#
OTOA         1#     375
PIRQ         1#     355
RAND         1#
SBKMOD       1#
SOFER        1#     511
. ABS.   000000      000
         004046      001
 ERRORS DETECTED:  0
 DEFAULT GLOBALS GENERATED:  0
 RKAA,RKAA/CRF/SOL=DDXCOM,RKAA
 RUN-TIME: 1 2 .4 SECONDS
 RUN-TIME RATIO: 37/4=8.1
 CORE USED:  7K  (13 PAGES)
```

APPENDIX C

Source Module Example (RKAA)

```
.REM

                        IDENTIFICATION

        PRODUCT CODE:   MAINDEC-11-DXRKA-E-D
        PRODUCT NAME:   DEC/X11 RK11 MODULE
        DATE:           21 MAR 1974
        MAINTAINER:     DIAGNOSTIC GROUP
        AUTHOR(S):      ROBERT E. UNDERWOOD
        REVISED BY:     S.J.H.(3/75)
```

1. ABSTRACT

RKA IS AN IOMODX THAT EXERCISES RK02, RK03, RK04, RK05 DISK DRIVES ON AN RK11 CONTROLLER. IT EXERCISES THE DRIVES BY DOING WRITES, WRITE-CHECKS, READS, AND IN-CORE COMPARISONS. ALL ERRORS DETECTED ARE REPORTED ON THE CONSOLE TTY.

2. REQUIREMENTS

HARDWARE: 1 TO 8 RK DISK DRIVES WITH AN RK11 CONTROLLER
STORAGE: RKA REQUIRES 983 WORDS OF STORAGE

3. PASS DEFINITION

ONE PASS OF THE RKA MODULE CONSISTS OF 512 CYCLES OF THE BASIC TEST SEQUENCE (WRITE, WRITE-CHECK, READ, DATA-CHECK). THE TEST SEQUENCE WRITES 1024 WORDS, WRITE-CHECKS SAME, READS THE FIRST 256 WORDS, AND DATA-CHECKS SAME.

4. EXECUTION TIME

ONE PASS OF RKA RUNNING ALONE ON A PDP-11/40 TAKES APPROXIMATELY 1 MINUTE.

5. CONFIGURATION REQUIREMENTS

DEFAULT PARAMETERS:
DEVADR: 177400, VECTOR: 220, BR1: 5, DEVCNT: 1

REQUIRED PARAMETERS:
NONE

6. DEVICE/OPTION SETUP

MAKE CERTAIN THAT ALL DRIVES ARE POWERED UP, WRITE ENABLED, AND READY

7. MODULE OPERATION

TEST SEQUENCE:

A. SETUP DEVICE REGISTER ADDRESSES AND MODULE VARIABLES
B. RESET ALL DRIVES ON-LINE AND DROP ALL THAT ARE NOT
C. GET A DISK ADDRESS AND A FRESH BLOCK OF DATA
D. GET A DRIVE ADDRESS
E. DO A WRITE -- IF ERRORS, REPORT AND RETRY UP TO RETRY LIMIT
F. DO A WRITE-CHECK -- IF ERRORS, REPORT AND RETRY UP TO RETRY LIMIT
G. DO A READ -- IF ERRORS, REPORT AND RETRY UP TO RETRY LIMIT
H. DO A DATA-CHECK -- IF ERRORS, REPORT AND CONTINUE
I. IF END OF PASS, REPORT AND GO TO C
J. IF END OF DRIVES, GO TO C ELSE GO TO D

8. OPERATION OPTIONS

SR1 BIT 0 SET(1):
IF THE RETRY LIMIT IS EXCEEDED ON ANY FUNCTION, A HARD ERROR IS ASSUMED AND THE DRIVE IS DROPPED.

SR1 BIT 0 CLEAR(0):
IF THE RETRY LIMIT IS EXCEEDED, THE FUNCTION IS ABORTED AND THE TESTING CONTINUES.

SR1 BIT 2 SET(1):
WILL NOT TYPE OUT DATA LATE ERRORS BUT WILL KEEP TRACK OF THE NUMBER OF DATA LATE ERRORS.

SR1 BIT 2 CLEAR(0):
TYPE OUT DATA LATE ERRORS AND KEEP TRACK OF THE NUMBER OF DATA LATE ERRORS IN DLTCNT.

9. NON-STANDARD PRINTOUTS

A. MOST PRINTOUTS HAVE THE STANDARD FORMATS DESCRIBED IN THE DEC/X11 DOCUMENT

B. ERROR MESSAGES DUMP THE CONTENTS OF THE 8 RK11 REGISTERS IN THE FOLLOWING ORDER:

RKDS RKER RKCS RKWC RKBA RKDA RKMR RKDB

```
        IOMODX  <RKAA  >,177400,220,5,0,0,512.,5,BUFIN,256.,1024.
        .PAGE

START:  MOV     #256.,WDTO      ; WORDS TO MEM
        MOV     #1024.,WDFR     ; WORDS FROM MEM
        MOV     #3,INTR         ; # OF INTERRUPTS/ITERATION
        CLR     SIDE            ; CLEAR FLAGS AND SIDE INDICATOR
        CLR     DLTCNT          ; CLEAR DATA LATE EROR COUNTER
        MOV     DVID1,DVICE     ; GET DRIVE INDICATOR
        MOV     DVICE,DRIVE     ; ALSO SAVE IT IN DRIVE
        MOV     #-3,BLK1        ; INITIALIZE BLOCK COUNTER
        CLR     DRYVE           ; ZERO UNIT NUMBER
        MOV     #160000,DRVSFT  ; INITIALIZE THE SHIFTED DRIVE #
        CMPB    #BIT1,@#41      ; IS RK UNIT 0 THE LOAD MEDIUM ?
        BNE     1$              ; NO, CONTINUE
        BIT     #BIT0,DVICE     ; YES, AND IS DRIVE 0 CHOSEN ?
        BEQ     1$              ; NO, CONTINUE
        JSR     PC,DROP         ; YES, GO DROP IT

1$:     MOV     #-1,DRYVE       ; INITIALIZE DRIVE COUNTER
        JSR     PC,SETUP        ; GENERATE REGISTER ADDRESSES
        JSR     PC,REZET        ; INITIALIZE RK REGS. AND ALL DRIVES
        TST     DVICE           ; DROP THE MODULE ?
        BEQ     FINI            ; YES
        BR      RSTRT1          ;+     / THIS IS
RESTRT: TST     PASCNT          ;+    /  SUPPORT
        BNE     RSTRT1          ;+   /    FOR
        BR      START           ;+ /      DT03
RSTRT1:                         ;+/       BUS SWITCH
        GETPA   RBUFVA
        MOV     RBUFSZ,WCNT2    ; SAVE READ BUFFER SIZE
        NEG     WCNT2           ; GET THE 2'S COMPLEMENT

STRT:   JSR     PC,BLOCK        ; GET NEXT BLOCK NUMBER
        GWBUFF
        MOV     WBUFSZ,WCNT1    ; SAVE WRITE BUFFER SIZE
        NEG     WCNT1           ; GET THE 2'S COMPLEMENT
        MOV     BLK1,R0         ; LOAD BLOCK # FOR CONVRT
        JSR     PC,CONVRT       ; GENERATE DISK ADR. FROM BLOCK #

NEXT:   JSR     PC,DRVADR       ; GET A DRIVE ADDRESS
        TST     DVICE           ; ANY DRIVES LEFT ?
        BEQ     FINI            ; NO, GO DROP THE MODULE
        BITB    #BIT3,FLAG      ; ALL DRIVES DONE ?
        BNE     STRT            ; YES, GO GET ANOTHER BLOCK
        BIC     #160000,DSKADR  ; CLEAR DRIVE ADDRESS
        BIS     DRVSFT,DSKADR   ; LOAD DRIVE ADDRESS
        MOV     DSKADR,@RKDA    ; LOAD DISK ADDRESS
        BIT     #BIT5,@RKDS     ; WRITE PROTECTED ?
        BEQ     1$              ; NO, CONTINUE
        JSR     PC,DROP         ; YES, DROP THE DRIVE
        MSGN    DRP
        BR      NEXT            ; GO ON TO NEXT DRIVE
1$:     BIT     #BIT6,@RKDS     ; DRIVE READY ?
        BNE     2$              ; YES, CONTINUE
```

```
        JSR     PC,NOTRDY       ; NO, WAIT FOR READY
        BR      STRT            ; TRY AGAIN
2$:     CLR     TRY1            ; ZERO RETRY COUNTERS
        CLRB    TRY3            ;
        .PAGE


GO:     JSR     R5,WRITE        ; WRITE SOME DATA
        BR      RETRY1          ; IF ERRORS, TRY IT AGAIN
        BITB    #BIT2,FLAG      ; DID THE DISK OVERFLOW ?
        BEQ     GOA             ; NO, CONTINUE
        BICB    #BIT2,FLAG      ; YES, CLEAR THE OVERFLOW FLAG
        MOV     #-3,BLK1        ; RESET THE BLOCK NUMBER
        BR      STRT            ; START OVER AT BEGINNING OF DISK
GOA:    JSR     R5,WRITCK       ; WRITE-CHECK THE DATA
        BR      RETRY2          ; IF ERRORS, TRY AGAIN
GOB:    JSR     R5,READ         ; READ THE DATA WRITTEN
        BR      RETRY3          ; IF ERRORS, TRY AGAIN
        CKDATA  RBUFPA


PASS:   ENDIT
        BR      NEXT

FINI:   ENDMOD  < DROP THE MODULE>
;       -------------------------


RETRY1: INCB    TRY1            ; COUNT THE RETRYS
        CMPB    #3,TRY1         ; LIMIT EXCEEDED ?
        BNE     GO              ; NO, GO TRY IT AGAIN
        MSGN    EXCED1
        BR      NEXTA           ; GO ON TO NEXT DRIVE
;       -------------------------
RETRY2: INCB    TRY2            ; COUNT RETRYS
        CMPB    #3,TRY2         ; LIMIT EXCEEDED ?
        BNE     GOA             ; NO, TRY AGAIN
        MSGN    EXCED2
        BR      NEXTA           ; GO ON TO NEXT DRIVE
;       -------------------------
RETRY3: INCB    TRY3            ; COUNT RETRYS
        CMPB    #3,TRY3         ; LIMIT EXCEEDED ?
        BNE     GOB             ; NO, GO TRY AGAIN
        MSGN    EXCED3
;       -------------------------
NEXTA:  BIT     #BIT0,SR1       ; DROP THE DRIVE ?
        BEQ     1$              ; NO, SKIP TO NEXT DRIVE
        JSR     PC,DROP         ; YES, DROP OFFENDING DRIVE
        MSGN    DRP
1$:     JMP     NEXT            ; GO ON TO NEXT DRIVE
;       -------------------------
        .PAGE
```

```
;               -----  RK11 DISK DRIVERS  -----

WRITE:  MOV     #503,FUNC       ; LOAD WRITE FUNCTION
        MOV     WCNT1,@RKWC     ; LOAD WORD COUNT
        MOV     WBUFPA,@RKBA    ; LOAD BUFFER ADDRESS
        MOV     WBUFEA,XMEM     ; LOAD EXTENDED MEMORY BITS
        BR      GOGO            ; CONTINUE
WRITCK: MOV     #507,FUNC       ; LOAD WRITE-CHECK FUNCTION
        MOV     WCNT1,@RKWC     ; LOAD WORD COUNT
        MOV     WBUFPA,@RKBA    ; LOAD BUFFER ADDRESS
        MOV     WBUFEA,XMEM     ; LOAD EXTENDED MEMORY BITS
        BR      GOGO            ; CONTINUE
READ:   MOV     #505,FUNC       ; LOAD READ FUNCTION
        MOV     WCNT2,@RKWC     ; LOAD WORD COUNT
        MOV     RBUFPA,@RKBA    ; LOAD BUFFER ADDRESS
        MOV     RBUFEA,XMEM     ; LOAD EXTENDED MEMORY BITS
        BR      GOGO            ; CONTINUE

CLEAR:  MOV     #1,@RKCS        ; ISSUE A CONTROL RESET
        JSR     PC,WAIT1        ; GO WAIT FOR CONTROLLER READY
        MOV     DRVSFT,@RKDA    ; RELOAD THE DRIVE ADDRESS
        BIT     #BIT6,@RKDS     ; DRIVE READY ?
        BNE     2$              ; YES, CONTINUE
        RTS     R5              ; NO, ABORT DRIVE RESET
2$:     MOV     #15,@RKCS       ; ISSUE A DRIVE RESET
        JSR     PC,WAIT         ; GIVE IT TIME TO COMPLETE
        MOV     #1,@RKCS        ; ISSUE ANOTHER CONTROLLER RESET
        JSR     PC,WAIT1        ; WAIT FOR CONTROLLER READY
        RTS     R5              ; RETURN

GOGO:   MOV     #NTRUPT,@VECTOR ; SET INTERUPT ENTRY POINTER
        MOV     DSKADR,@RKDA    ; LOAD THE DISK ADDRESS
        BIS     XMEM,FUNC       ; LOAD EXTENDED MEMORY BITS
        MOV     FUNC,@RKCS      ; EXECUTE THE FUNCTION
        EXIT

NTRUPT: PIRQ    1$

1$:     JSR     R5,ERRORS       ; GO CHECK FOR ERRORS
        RTS     R5              ; ERRORS DETECTED, RETURN
        TST     (R5)+           ; NO ERRORS, SKIP RETRY
        RTS     R5              ; RETURN OK
;       -------------------------
        .PAGE


DROP:   MOV     #1,R1           ; INITIALIZE DROP PICKER
        MOV     DRYVE,R0        ; GET THE DRIVE NUMBER
        BEQ     2$              ; IF DRIVE 0 GO DROP IT
1$:     ASL     R1              ; NO, AIM AT THE NEXT DRIVE
        DEC     R0              ; IS THIS THE ONE ?
        BNE     1$              ; NO, LOOK AGAIN
2$:     BIC     R1,DVICE        ; DROP THE DRIVE
        MOV     DRYVE,-(R6)     ; PUSH DRIVE # ONTO STACK
        OTOA    DRYVE,ADR1
```

```
        RTS     PC              ; RETURN
;       -------------------------

BLOCK:  ADD     #3,BLK1         ; STEP TO NEXT BLOCK
        CMP     #4799.,BLK1     ; BLOCK LIMIT REACHED ?
        BPL     1$              ; NO, CONTINUE
        CLR     BLK1            ; YES, RESET BLOCK #
1$:     MOV     BLK1,BLK2       ; READ WHERE WRITE
        RTS     PC              ; RETURN
;       -------------------------

ROOM:   MOV     BLK1,R0         ; SAVE THE CURRENT BLOCK NUMBER
        MOV     #2399.,R1       ; LOAD MAX. NUMBER OF BLOCK PER SIDE
        CLR     R2              ; ZERO REG. 2
        CMP     #2399.,R0       ; IS SIDE 0 DONE ?
        BGE     1$              ; NO, CONTINUE
        SUB     #2400.,R0       ; YES, NORMALIZE BLOCK # FOR SIDE 1
1$:     MOV     #256.,BSIZ      ; HI DENSITY BLOCK SIZE
        BIT     #BIT11,@RKDS    ; HI DENSITY DRIVE ?
        BNE     2$              ; YES, CONTINUE
        ASR     BSIZ            ; NO, SET TO 128 -- LO DENSITY
2$:     SUB     R0,R1           ; GET # OF BLOCKS LEFT ON DISK
3$:     ADD     BSIZ,R2         ; GET TOTAL NUMBER OF WORDS LEFT
        DEC     R1              ; ALL BLOCKS ADDED IN ?
        BGT     3$              ; NO, KEEP ADDING
        TST     R2              ; IS # OF WORDS LEFT ON DISK NEG. ?
        BMI     4$              ; YES
        TST     WBUFSZ          ; IS TRANSFER SIZE NEG. ?
        BMI     7$              ; YES
        BR      5$              ; NO, GO COMPARE
4$:     TST     WBUFSZ          ; IS TRANSFER SIZE POS. ?
        BPL     6$              ; YES
5$:     CMP     R2,WBUFSZ       ; WAS THERE ENOUGH ROOM FOR THE TRANSFER ?
        BLT     7$              ; NO, RETURN OK
6$:     TST     (R5)+           ; YES, MUST BE A REAL ERROR
        RTS     R5              ; RETURN, ERROR
7$:     BISB    #BIT2,FLAG      ; SET OVERFLOW FLAG
        RTS     R5              ; RETURN OK
;       -------------------------
        .PAGE

DRVADR: INC     DRYVE           ; COUNT A DRIVE
        ADD     #BIT13,DRVSFT   ; DRIVE COUNT LINED UP WITH RKDA
        BICB    #BIT3,FLAG      ; CLEAR END OF DRIVES FLAG
        CMP     #8.,DRYVE       ; ALL DRIVES CHECKED ?
        BEQ     1$              ; YES, GO FLAG END OF DRIVES
        ASR     DRIVE           ; NO, IS NEXT DRIVE CHOSEN ?
        BCC     DRVADR          ; NO, GO TRY ANOTHER DRIVE
        RTS     PC              ; RETURN

1$:     BISB    #BIT3,FLAG      ; SET END OF DRIVES FLAG
        MOV     #-1,DRYVE       ; RESET DRIVE COUNTER
```

```
        MOV     #160000,DRVSFT  ; ZERO THE SHIFTED DRIVE #
        MOV     DVICE,DRIVE     ; RESTORE CHOSEN DRIVES
        RTS     PC              ; RETURN
;       -------------------------

NOTRDY: MOV     #-1,DRYVE       ; START WITH FIRST DRIVE
        MOV     #160000,DRVSFT  ;
        MOV     DVICE,DRIVE     ; RESET DRIVE SELECT
1$:     JSR     PC,DRVADR       ; GET A DRIVE ADDRESS
        BITB    #BIT3,FLAG      ; ALL DRIVES CHECKED ?
        BNE     2$              ; YES, RETURN
        MOV     DRVSFT,@RKDA    ; NO, LOAD NEXT DRIVE ADDRESS
        BIT     #BIT6,@RKDS     ; IS THIS DRIVE READY ?
        BNE     1$              ; YES, CONTINUE
        JSR     PC,WAIT         ; NO, WAIT FOR IT
        BR      1$              ; GO CHECK REST OF DRIVES
2$:     RTS     PC              ; RETURN
;       -------------------------

ERSUB2: MOV     -(R1),ASB       ; LOAD THE DATA
        MOV     R1,SBADR        ; LOAD ADDRESS OF DATA WRITTEN
        MOV     -(R2),AWAS      ; LOAD THE DATA
        MOV     R2,WASADR       ; LOAD ADDRESS OF DATA READ
        TST     (R1)+           ; RESET REG. 1
        TST     (R2)+           ; RESET REG. 2

ERSUB1: MOV     RKCS,CSRA       ; LOAD ADR. OF CURRENT CSR
        MOV     @RKCS,ACSR      ; LOAD CONTENTS OF CURRENT CSR
        RTS     PC              ; RETURN
;       -------------------------
        .PAGE



WAIT:   MOV     #77777,CLK      ; SET THE TIMER
1$:     BREAK
        BIT     #BIT6,@RKDS     ; DRIVE READY ?
        BNE     2$              ; YES, RETURN
        DEC     CLK             ; NO, WAIT SOME MORE ?
        BNE     1$              ; YES, WAIT
        JSR     PC,DROP         ; TIME-OUT, DROP THE DRIVE
        MSGN    DRP
2$:     RTS     PC              ; RETURN
;       -------------------------
ERRORS: JSR     PC,ERSUB1       ; LOAD ERROR INFORMATION
        BIT     #BIT14,@RKCS    ; HARD ERROR ?
        BNE     1$              ; YES, GO REPORT
        BIT     #3,@RKER        ; SOFT ERROR ?
        BNE     3$              ; YES, GO REPORT
        TST     (R5)+           ; NO, SKIP RETRY
        RTS     R5              ; RETURN OK
1$:     BIT     #BIT14,@RKER    ; DISK OVERFLOW ?
```

```
        BEQ     7$              ; NO, CONTINUE
        JSR     R5,ROOM         ; YES, IS IT A REAL ERROR ?
        BR      5$              ; NO, CONTINUE
7$:     BIT     #BIT9,@RKER     ;DATA LATE ERROR?
        BEQ     2$              ;NO
        INC     DLTCNT          ;INCREMENT ERROR COUNTER
        BIT     #BIT2,SR1       ;TYPE OUT ERROR?
        BNE     6$              ;NO
        MSGN    DLTERR
2$:     MSGN    HARD
        CLR     ERRTYP
        HRDER   TABLE
6$:     JSR     R5,CLEAR        ; GO CLEAR OUT ERRORS
        BR      4$              ; RETURN
3$:     MSGN    SOFT
        MOV     #1,ERRTYP       ; DATA ERROR
        SOFER   TABLE
4$:     RTS     R5              ; RETURN, ERRORS
5$:     JSR     R5,CLEAR        ; CLEAR OUT ERRORS
        TST     (R5)+           ; SKIP RETRY
        RTS     R5              ; RETURN OK
;       -------------------------
        .PAGE


SETUP:  MOV     ADDR,R0         ; GET DEVICE ADDRESS
        MOV     R0,RKDS         ; GENERATE CONTROLLER REGS. ADDRESSES
        TST     (R0)+
        MOV     R0,RKER
        TST     (R0)+
        MOV     R0,RKCS
        TST     (R0)+
        MOV     R0,RKWC
        TST     (R0)+
        MOV     R0,RKBA
        TST     (R0)+
        MOV     R0,RKDA
        TST     (R0)+
        MOV     R0,RKMR
        TST     (R0)+
        MOV     R0,RKDB

        MOV     VECTOR,R0       ; GET THE VECTOR ADDRESS
        MOV     #STRT,(R0)+     ; SET POINTER JUST IN CASE
        MOVB    BR1,(R0)        ; SET PRIORITY

2$:     RTS     PC              ; RETURN
;       -------------------------


CONVRT: CLR     R1              ; ZERO REG. 1
        CLRB    SIDE            ; ZERO THE SIDE INDICATOR
        MOV     #-12.,R3        ; LOAD REG. 3
        MOV     #11.,R4         ; LOAD REG. 4
        CMP     #2399.,R0       ; IS BLOCK ON SIDE 0 ?
```

```
        BGE     1$              ; YES, CONTINUE
        BISB    #BIT4,SIDE      ; NO, FLIP TO SIDE 1
        ADD     #-2400.,R0      ; NORMALIZE BLOCK # FOR SIDE 1
1$:     CMP     R4,R0           ; FIND THE RIGHT CYLINDER ?
        BGE     2$              ; YES, CONTINUE
        ADD     R3,R0           ; NO, SUBTRACT 12 SECTORS (1 CYLINDER)
        INC     R1              ; KEEP TRACK OF CYLINDER ADDRESS
        BR      1$              ; GO TRY AGAIN
2$:     MOV     R0,DSKADR       ; LOAD THE SECTOR ADDRESS
        BISB    SIDE,DSKADR     ; LOAD THE SIDE ADDRESS
        MOV     #5,R2           ; SET UP FOR SHIFT
3$:     ASL     R1              ; LINE UP CYL. ADR. WITH DSKADR
        DEC     R2              ; DONE ?
        BGT     3$              ; NO, GO SHIFT AGAIN
        BIS     R1,DSKADR       ; YES, LOAD THE CYLINDER ADDRESS
        RTS     PC              ; RETURN
;       ------------------------
        .PAGE
REZET:  MOV     #1,@RKCS        ; EXECUTE CONTROLLER RESET
        JSR     PC,WAIT1        ; GO WAIT FOR CONTROLLER READY
        JSR     PC,NOTRDY       ; MAKE SURE ALL CHOSEN DRIVES ARE READY
1$:     JSR     PC,DRVADR       ; GET A DRIVE ADDRESS
        BITB    #BIT3,FLAG      ; ALL DRIVES DONE ?
        BNE     2$              ; YES, RETURN
        JSR     R5,CLEAR        ; ISSUE DRIVE RESET AND CONTROLLER CLEAR
        BR      1$              ; KEEP GOING
2$:     RTS     PC              ; RETURN
;       ------------------------


WAIT1:  MOV     #77777,CLK      ; SET THE TIMER
1$:     TSTB    @RKCS           ; CONTROLLER READY ?
        BMI     2$              ; YES, CONTINUE
        BREAK
        DEC     CLK             ; WAIT SOME MORE ?
        BNE     1$              ; YES
        MOV     #3,ERRTYP       ; CONTROLLER NOT READY
        HRDER   TABLE   < CONTROLLER NOT READY>
        JMP     FINI            ; GO DROP THE MODULE
2$:     RTS     PC              ; READY, RETURN
;       ------------------------

DLTCNT: 0
FUNC:   0
XMEM:   0
DSKADR: 0
DVICE:  0
DRIVE:  0
DRYVE:  0
DRVSFT: 0
BLK1:   0
BLK2:   0
BSIZ:   0
TBUF:   0
WCNT1:  0
```

```
WCNT2:  0
BUFLEN: 256.
BUFIN:  .BLKW   256.
CLK:    0
TABLE:
RKDS:   0
RKER:   0
RKCS:   0
RKWC:   0
RKBA:   0
RKDA:   0
RKMR:   0
RKDB:   0
        177777
        .PAGE
MES1:   .ASCIZ  '   HARD ERROR'
MES2:   .ASCIZ  '   SOFT ERROR'
MES4:   .ASCIZ  '  DRIVE  '
MES5:   .ASCIZ  '  DROPPED%'
MES6:   .ASCIZ  ' RETRY EXCEEDED%'
MES7:   .ASCIZ  ' WRITE'
MES8:   .ASCIZ  ' WRITE-CHECK'
MES9:   .ASCIZ  ' READ'
MES10:  .ASCIZ  'DATA LATE ERROR%'
        .EVEN
HARD:   MES1
        177777
SOFT:   MES2
        177777
EXCED1: MES7
        MES6
        177777
EXCED2: MES8
        MES6
        177777
EXCED3: MES9
        MES6
        177777
DRP:    MES4
        NUMB
        MES5
        177777
DLTERR: MES10
        177777
ADR1:   .BLKB   5
NUMB:   .BYTE   0
        .WORD   0
SIDE:   .BYTE   0
FLAG:   .BYTE   0
        .EVEN
TRY1:   .BYTE   0
TRY2:   .BYTE   0
TRY3:   .BYTE   0
        .EVEN
```

```
.END
```

TABLE OF ERROR CODES

ERROR	TYPE
0	Not Defined
1	Data Error
2	Data Late
3	Controller not ready
4	Block not found
5	Block missed
6	Device off-line, non-existent or not ready
7	Selection ERROR
10	Non-existent memory
11	Illegal interrupt occurred or "Done" did not set
12	Premature end of file encountered
13	Rewind error (rewind took too long)
14	# of interrupts incorrect
15	Incorrect vector address
16	"Busy" won't clear in time
17	Unknown receiver error
20	Unknown transmitter error
21	Overrun error
22	Framing error
23	Device failed to interrupt
24	Time-out-shift out error

ERROR TYPE_____	____
25	Bit stuck in Register or DID not change state in TIME
26	A-D CONVERSION OUT OF SPEC.
27	Interrupt enable error
30	Unknown ERROR during data Transfer
31	A/D RMS or peak noise exceeded limit
32	NPR error
33	Device not in maintenance mode
34	Device will not initialze
35	Buffer fill error
36	Unable to execute a Read FUNCTION
37	Unable to excute a write function
40	Transfer read bit did not set
41	Transmit data late error
42	Active bit in register should be set - not cleared
43	Cyclic redundancy check error detected
44	Flag should not be set
45	Floating point mathemaTical operation produced INCORRECT results
46	Clock overflow failed to trigger A/D conversion
47	Controller would not clear
50	Data set line change
51	BAD SEEK

ERROR	TYPE_____	____
52	MICRO CODE NOT LOADED	