Techniques Development Laboratory
Computer Program NWS TDL CP 89-3

EXTENDED MEMORY LIBRARY FOR AFOS APPLICATIONS

Silver Spring, Md.
June 1989

U.S. DEPARTMENT OF COMMERCE / National Oceanic and Atmospheric Administration / National Weather Service

PREFACE

The Techniques Development Laboratory's (TDL's) computer program (CP) series is a subset of TDL's technical memorandum series. The CP series documents computer programs written at TDL primarily for the Automation of Field Operations and Services (AFOS) computers.

The format for the series follows that given in the AFOS Handbook 5, Reference Handbook, Volume 6: Applications Programs, Part 1: Policy and Procedures, published by the Office of Technical Services/AFOS Operations Division.

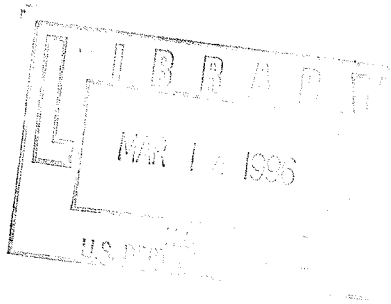
NOAA Techniques Development Laboratory
Computer Program NWS TDL

CP 83-1 Gross Sectional Analysis of Wind Speed and Richardson Number. Gilhousen, Kemper, and Vercelli, May 1983. (PB83205062)
CP 83-2 Simulation of Spilled Oil Behavior in Bays and Coastal Waters. Hess, October 1983. (PB84122597)
CP 83-3 AFOS-Era Forecast Verification. Heffernan, Newton, and Miller, October 1983. (PB84129303)
CP 83-4 AFOS Monitoring of Terminal Forecasts. Vercelli, December 1983. (PB84145697LL)
CP 83-5 Generalized Exponential Markov (GEM) Updating Procedure for AFOS. Herrmann, December 1983. (PB84154822LL)
CP 84-1 AFOS Display of MDR Data on Local Map Background. Newton, July 1984. (PB84220797)
CP 84-2 AFOS Surface Observation Decoding. Perrotti, September 1984. (PB85137586)
CP 84-3 AFOS-Era Forecast Verification. Miller, Heffernan, and Ruth, September 1984. (PB86148319LL)
CP 85-1 AFOS Monitoring of Terminal Forecasts. Vercelli and Norman, May 1985. (PB85236388LL)
CP 85-2 AFOS Terminal Forecast Decoding. Vercelli, Norman, and Heffernan, October 1985. (PB86147360LL)
CP 85-3 AFOS-Era Forecast Verification. Ruth, Miller, and Heffernan, October 1985. (PB86148319LL)
CP 87-1 AFOS Terminal Aerodrome Forecast Formatting. Wantz and Eggers, July 1987. (PB8810449LL)
CP 87-2 AFOS-Era Forecast Verification. Ruth and Alex, July 1987. (PB88125570LL)
CP 87-3 Forecast Review. Wolf, July 1987. (PB88125588LL)
CP 87-4 AFOS Monitoring of MDR Data Using Flash Flood Guidance. Norman and Newton, October 1987. (PB88137450LL)
CP 87-5 AFOS Terminal Forecast Quality Control. Vercelli and Leaphart, December 1987. (PB88169925LL)
CP 88-1 AFOS Terminal Forecast Decoding. Vercelli and Leaphart, August 1988. (PB89101240LL)
CP 89-1 Structure Flow Diagram Generator. Adams, March 1989. (PB89195978AS)
CP 89-2 String Search. Adams, March 1989. (PB89195986AS)

NOAA Techniques Development Laboratory
Computer Program NWS TDL CP 89-3

EXTENDED MEMORY LIBRARY FOR AFOS APPLICATIONS

Mark A. Leaphart



Techniques Development Laboratory
Silver Spring, Md.
June 1989

UNITED STATES DEPARTMENT OF COMMERCE
Robert A. Mosbacher
Secretary

National Oceanic and Atmospheric Administration
William E. Evans, Under Secretary

National Weather Service
Elbert W. Friday, Jr.
Assistant Administrator

TABLE OF CONTENTS

TABLE OF CONTENTS

EXTENDED MEMORY LIBRARY FOR AFOS APPLICATIONS

Mark A. Leaphart

1. INTRODUCTION

The Data General Corporation's Eclipse S/230 minicomputer limits every applications program to 32K of logical memory. This is often very burdensome to programmers who are trying to develop software on Automation of Field Operations and Services (AFOS). The Techniques Development Laboratory (TDL) has developed an extended memory library, XMEM.LB, which provides the programmer with a way of using more than 32K words of logical memory. This collection of FORTRAN IV callable assembly language routines interfaces with the Real Time Disk Operating System (RDOS) window mapping facility. The capability and flexibility of XMEM.LB was adapted from the Data General FORTRAN 5 version (Data General Corporation, 1984). All routines in XMEM.LB were assembled with the Data General Macro Assembler, Revision 6.30.

This document provides information about extended memory and window mapping, including setting up extended memory, handling data in extended memory, definitions of error codes, and a description of the modules in XMEM.LB with examples showing how to use them.

2. EXTENDED MEMORY

The allotment of memory (both logical and extended), defined in 1024-word blocks, is done via the CLI SMEM command at system start time. The logical address space is the amount of memory that can be referenced by instructions in a program (Data General Corporation, 1975a). Memory outside of the logical address space is called extended address space or extended memory (Data General Corporation, 1979). The amount of extended memory available to a program will depend on the amount of memory allotted to the background and the amount of memory the program takes up. The difference between these two amounts will make up extended memory. For example, at a WSFO, background memory is allotted in the AFOS start-up macro. If 128 blocks are allotted to background, and an applications program uses all 32K logical address space (32 blocks), another 96 blocks will be available for window mapping.

3. WINDOW MAPPING

Window mapping allows the programmer to gain direct access to portions of extended memory; it also allows the programmer to transfer blocks between extended memory and disk (Data General Corporation, 1979). A window is an area (basically an array) in a program that you use to make references to extended memory. There can only be one window in your program and it must be aligned, in memory, on a 1024-word boundary. The window area can be defined anywhere in your program as long as its location in memory is below the NMAX value of your program. The NMAX value can be found in your program's load map. When a program is being loaded by the RLDR, NMAX is used as an address counter for Relocatable Binary (.RB) files. NMAX is the address of the next .RB file placed in the save and overlay file. When the RLDR is done with the current .RB file, the RLDR updates NMAX by the number of instructions that the

current .RB file takes up. After the program is loaded, the NMAX value you see in the load map is the program's address space minus 1. The program will execute instructions in locations 0 through NMAX - 1. The way that the window references extended memory can be best described as analogous to the FORTRAN EQUIVALENCE statement. The blocks in the window and in extended memory share the same memory location. The main difference between the two is that you can only equivalence two data objects at a time per EQUIVALENCE statement, but with the window you can reference different blocks of extended memory as many times as you like. Once you define your window, your window will always be referencing extended memory. All updates (intentional and unintentional) to the window will simultaneously be updated in extended memory.

4. SETTING UP EXTENDED MEMORY

There are two parts to setting up extended memory and each part consists of a couple of steps. The first part is done by the programmer before program execution, and the second part is done by the program at execution time.

The first part consists of aligning the window on a 1024-word boundary. The first step is making sure that the window is in COMMON and in its own labeled common block. The location of the labeled common block with respect to other labeled common blocks is important. The window must be the only labeled common block or the last labeled common block in the routine. The reason for this is due to the way the FORTRAN IV compiler (Rev 5.20 and 5.57) handles common blocks. The compiler handles common blocks in the reverse order that you list them (see Figs. 1a & 1b).

The second step involves setting up the load line so that the address of the routine that defines the window starts on a 1024-word boundary. This is done through the CLI RLDR command. The RLDR command has a local switch that allows you to start a routine's code at a given address. The format of the switch is "n/N", where n is an octal address and a multiple of 1024 (decimal):

```
RLDR 2000/N Mainprgm .............
```

Using the "/N" switch with a program that uses extended memory will leave an unused portion of memory located just before the address of the "/N" switch. In this unused portion, you can place any subroutine(s) which fit(s) into memory between the address where the RLDR starts loading subroutines and the address given by the "/N" switch (see Figs. 2a & 2b). (The address where the RLDR starts loading subroutines will vary from program to program.) For example:

```
RLDR Subprgm 2000/N Mainprgm Subprgms ..........
```

If you do this, you must use another local switch of the RLDR command to name the save and overlay file. Normally, the RLDR gives the save and overlay file the name of the first binary file in the load line, and in this case the RLDR will give the save and overlay file the names "Subprgm.SV" and "Subprgm.OL". The local switch that you use to name the save and overlay file is "name/S", where name is Mainprgm.

```
RLDR Subprgm 2000/N Mainprgm Subprgms Mainprgm/S ..........
```

The second part in setting up extended memory consists of defining the window in your program. First you must determine the amount of extended memory available to your program. Second, you must define a map from the window in your program to extended memory. These two steps must be done in this order to ensure that the rest of the extended memory routines will work properly. VMEM and MAPDF are the routines that perform these steps and are described in more detail starting on page 10.

5. HANDLING DATA IN EXTENDED MEMORY

When developing software using routines from the extended memory library, the programmer needs to know how to look at extended memory. By knowing how extended memory is viewed, the programmer can decide which view best fits his/her data. Routines have been developed so that the programmer can look at extended memory in two ways. Extended memory may be thought of as: 1) one large FORTRAN array, or 2) broken up into many FORTRAN arrays, each one the size of the window.

VSTASH and VFETCH are the routines to use when viewing extended memory as one large FORTRAN array. REMAP is the routine to use when viewing extended memory as an aggregate of many arrays. These routines are described in more detail on pages 17 and 38.

The two views differ in their methods of accessing extended memory. For the first view, data are accessed by a word index. This is done by specifying the word location in extended memory and the number of elements you want to access. The window is used as a buffer between extended memory and an array aggregate. All calculations to the data are done in the array aggregate, and after the calculations are done, the data must be put back into extended memory. Accessing the data this way, when compared to the second method, is slow. A lot of time is spent copying data to and from extended memory. With the second method, data are accessed in terms of 1024-word blocks. This is done by specifying the block location of the FORTRAN array in extended memory and the block location in the window. All calculations to the data are done using the window. The data do not have to be copied back into extended memory because extended memory is referenced directly. Accessing the data this way is very fast. No data is actually transfered; only the position in extended memory that the window points to is changed.

XMEM.LB provides the programmer with routines (EWRB and ERDB) that write and read data to and from disk and extended memory. The second view of extended memory is ideally suited for EXTENDED DIRECT BLOCK DISK I/O because the window is a multiple of 1024, and hence, a multiple of 256. (EXTENDED DIRECT BLOCK I/O, like regular BLOCK I/O, reads and writes data in terms of 256-word blocks.) The first view is not really suited for EXTENDED DIRECT BLOCK I/O because it accesses data on a word index, but if the element size used happens to be a multiple of 256, EXTENDED DIRECT BLOCK I/O can be used.

Because the window size has to be a multiple of 1024 words, often the window may be bigger than the size that is really needed. For example, if an array of 2200 words is needed, the smallest size the window can be is 3072. Within each of the many FORTRAN arrays, the last 872 words would not be used. When this happens, extended memory becomes fragmented. This does not happen when extended memory is viewed as one large FORTRAN array. Here, the data are put

consecutively into extended memory, and all unused portions are at the end of extended memory.

6. LOADING

XMEM.LB can be placed anywhere in the load line before the FORT.LB. A typical load line for a FORTRAN program appears as follows:

```
RLDR 2000/N Mainprgm Subprgms <UTIL XMEM FORT SYS AFOSE>.LB
```

The extended memory library provides you with the option of placing XMEM.LB in its own overlay, and the overlay name is OXMEM. XMEM.LB can be placed anywhere in the overlay structure. A typical load line for a FORTRAN program with an overlay structure appears as follows:

```
RLDR 2000/N Mainprgm Subprgms [Overlay1, XMEM.LB, Overlay3]
   <UTIL FORT SYS AFOSE>.LB
```

7. ERROR RETURNS

Bad error returns from XMEM.LB routines are the RDOS error codes plus three which give you the respective FORTRAN error codes. A good error return is the FORTRAN error code of one. In some instances, when the End-of-File condition is met or Disk Space Exhaustion occurs, the error code is recognized by the routine, all activity for that routine stops, and the FORTRAN error code is returned as one. This will be discussed later in the COMMENTS section of some of the Module Descriptions.

8. XMEM.LB

There are nine subroutines and five functions contained in six modules in the extended memory library. The library was designed with flexibility in mind. Several of the routines allow the programmer a choice of multiple forms of the CALL statement (see the attached Module Descriptions), thus giving the programmer a greater range of use of the software. Several related routines share a lot of the same code, so these routines are merged into one module with multiple entry points. By getting rid of redundant code, you cut the size of the library down, thus giving the programmer optimal code. There is a Module Description starting on page 10 for each module, followed by examples that show how to use the routines in each module. All references to FORT.LB came from Run Time Library User's Manual FORTRAN IV (Data General Corporation, 1975b), and the documentation for the variables in the ARGUMENT section of the Module Description came from the FORTRAN 5 Programmer's Guide (RDOS) (Data General Corporation, 1984).

9. REFERENCES

Data General Corporation, 1975a: Programmer's Reference Manual: Eclipse Line Computers, Data General Corporation, Westboro, Mass., 121 pp.

_____, 1975b: RUN TIME LIBRARY User's Manual FORTRAN IV, Data General Corporation, Westboro, Mass., 186 pp.

_____, 1978: Extended Relocatable Loaders User's Manual, Data General Corporation, Westboro, Mass., 66 pp.

_____, 1979: Real Time Disk Operating System (RDOS) Reference Manual, Data General Corporation, Westboro, Mass., 204 pp.

_____, 1984: FORTRAN 5 Programmer's Guide (RDOS), Data General Corporation, Westboro, Mass., 122 pp.

```
      COMMON/IB/IBUF(10)
      COMMON/JB/JBUF(25)
      COMMON/MB/MBUF(16)
      COMMON/KB/KBUF(15)
C
      STOP
      END
```

Figure 1a. Example program with four labeled common blocks.

```
LINE  TITLE     ADDRESS (8)

04    .MAIN     000445          The RLDR's Load Map has two parts:  a memory map
05    STOPP     000503          and a list of symbols.  The memory map contains
.       .         .             the .RB titles, and the list of symbols contains
.       .         .             assembly instructions and logical addresses of
18    TMIN      002014          variables and labeled common blocks.  Here,
19    NSAC3     002110          lines 4 through 19 are the memory map and lines
20                              22 through 95 are the list of symbols.
21
22      NMAX      002110        The RLDR builds the save file upward in memory
23      ZMAX      000076        (Data General Corporation, 1978).  A routine
24      CSZE      000000        with a lower address means it was handled before
25      EST       000000        a routine with a higher address.
26      SST       000000
27
28      ESV.Z     000006
29      .STOP     000050
30      .IOPR     000051
.         .         .
.         .         .
48      QSP       000074
49      NSP       000075        Lines 51 through 54 hold the logical addresses
50      USTAD     000400
51    C KB        000445  000017
52    C MB        000464  000020
53    C JB        000504  000031
54    C IB        000535  000012
55      .MAIN     000551
56      EXIT      000564        of the four labeled common blocks in Fig. 1a.
57      .I        000717        A common block has a "C" preceding its name and
.         .         .           its length is shown to the right of its address.
.         .         .           Compare the order of the common blocks in Fig. 1a
95      FRTSK     177777        with lines 51 through 54.  You will see that the
                                order is reversed.
```

Figure 1b. Abbreviated Load Map of the program in Fig. 1a.

```
RLDR/P EIJLL IERX TIMPR RDIDG 2000/N DRE400AEXT PROPEN E400AEXT RDIDE
ANAL1EXT
[RDSTNEXT ELLIJ, XLTAGEXT FSTGS CHGDT CUTIT, ESLPEXT, BCDEXT]
[RDTDL WRTDL FLOPN RDARG FLNAM IASCI DAY19 IERCK CLFIL, PRTGR SETUP
 BESEL, ESPEXT CLOSEXT SMOTH ITRP, A40RD A40WR DATMP]
DRE400AEXT/S DRE400AEXT.LM/L
<F5ISA SYS>.LB LONGTRACE F5<IO MATH1 MATH2 ENV BGDR>.LB AFOSE.LB
```

Figure 2a. Load Line of LAMP'S Objective Analysis program, DRE400AEXT, using extended memory and using the local switch "/S".

```
DRE400AEXT.SV    LOADED BY RLDR REV 07.10
EIJLL     000456          The RLDR started loading the .RB's from the load
IERX      000642          line at address 000456.  The programmer was able to
TIMPR     000767          place four subroutines before the local switch "/N".
RDIDG     001067
.MAIN     002000       <---- Main program, DRE400AEXT
PROPEN    004454
E400A     004702
.BDTA     010344
RDIDE     010344
ANAL1     010704
       012207
       000,000  RDSTN 012207
                ELLIJ 013401 000736
       000,001  XLTAG 012207 000261
       000,002  FSTGS 012207
                CHGDT 014112
                CUTIT 014302 002475
       000,003  ESLP  012207 000401
       000,004  BCD   012207 002544
       015207
  .
  .
  .
  .
  .
  .
  .
```

Figure 2b. Partial Load Map of program DRE400AEXT. Load Line is in Fig. 2a.

11. LIBRARY INFORMATION AND SPECIFICATIONS

EXTENDED MEMORY LIBRARY

LIBRARY INFORMATION

PROGRAM NAME: XMEM.LB

AAL ID: LBS021
Revision No.: 01.00

PURPOSE: Provides the programmer with a method for using more than 32K of logical memory. XMEM.LB is a collection of FORTRAN IV callable assembly language routines that interfaces with the RDOS window mapping facility. Window mapping allows the programmer to gain access to memory outside a program's logical address space. This memory is known as extended memory.

LIBRARY ROUTINES

VMEM - Returns the amount of extended memory that is available to a program.

MAPDF - Defines the window map or redefines the permanent element size.

REMAP - Does a logical window transfer by placing blocks from extended memory address space into the window.

EWRB - Writes a series of disk blocks from extended memory into a random or contiguously organized file.

ERDB - Reads a series of disk blocks from a random or contiguously organized file into extended memory.

VDUMP - Copies blocks from extended memory to a disk file.

VLOAD - Initializes extended memory to the contents of a disk file.

VFETCH - Copies one or more elements from extended memory to an array aggregate.

VSTASH - Copies one or more elements into extended memory from an array aggregate.

IVF - Fetches one integer number from extended memory.

VF - Fetches one real number from extended memory.

CVF - Fetches one complex number from extended memory.

DVF - Fetches one double precision number from extended memory.

DCVF - Fetches one double precision complex number from extended memory.

LOAD LINE

RLDR X000/N MAINPRGM (user-defined code) ...
... XMEM.LB FORT.LB ...

X000 is multiple of 2000 (octal), and XMEM.LB must precede FORT.LB somewhere in the load line.

LIBRARY SPECIFICATIONS

Developmemt Programmer(s):		Maintenance Programmer(s):	
Mark A. Leaphart		Mark A. Leaphart	
Location:	Techniques Development Laboratory	Location:	Techniques Development Laboratory
Phone:	FTS 427-7639	Phone:	FTS 427-7639
Language:	Macro Assembler/Rev 6.30	Type:	FORTRAN IV callable assembly routines

Library creation date: - May 11, 1988

Disk space: Library files - 2 RDOS blocks

LIBRARY REQUIREMENTS

Library files:

NAME	COMMENTS
LBS021	Contains all necessary routines that will allow the programmer to set up the window and access extended memory.

VMEM

I. IDENTIFICATION

Module name: VMEM
Date: May 11, 1988
Function: Determines the amount of extended memory.
Language: Assembly

II. PURPOSE

Determines the amount of extended memory available to a program.

III. ENTRY POINTS

VMEM

IV. CALLING METHOD

```
CALL VMEM(BLOCKS,IER)
```

V. ARGUMENTS

BLOCKS = An integer variable that receives the number of free 1024-word blocks of extended memory.
IER = An integer variable that receives the routine's error return code.

VI. ERROR RETURNS

NONE

VII. REFERENCED EXTERNALS

.VMEM (System call)

FORT.LB

.CPYL transfers effective address of a caller's argument list to its called subroutine's stack.

.FRET restores a caller's accumulators and state of carry upon exit from the called subroutine, and returns to the next instruction following the call.

VIII. COMMENTS

1) VMEM must be called after calls to OVOPN and MEMI (if you use these routines).

2) VMEM is the first routine in sequence you need to call to set up extended memory.

Module name: VMEM

3) You must call VMEM to determine the amount of extended memory, otherwise the rest of the extended memory routines will not work properly.

4) You need to call VMEM only once in your program.

5) It is possible for VMEM to return zero as a value for BLOCKS. The return code will be one.

MAPDF

I. IDENTIFICATION

Module name: MAPDF
Date: May 11, 1988
Function: Defines a logical window.
Language: Assembly

II. PURPOSE

Defines a window map or redefines the permanent element size (ELMSIZ). MAPDF will assign relative block numbers 0 through BLOCKS - 1 to extended memory.

III. ENTRY POINTS

MAPDF

IV. CALLING METHOD

1) CALL MAPDF(BLOCKS,WINDOW,WINSIZ,[ELMSIZ,]IER)

2) CALL MAPDF(ELMSIZ,IER) - redefines the element size.

VI. ARGUMENTS

1) BLOCKS = An integer that specifies the total number of blocks of memory you want to allocate for window mapping use.
   WINDOW = An aggregate in your program through which references to extended memory are made.
   WINSIZ = An integer that specifies the size of the window in 1024-word blocks.
   ELMSIZ = An integer that specifies the size of an element in words. If omitted, the element size is one.
   IER = An integer variable that receives the routine's error return code.

2) ELMSIZ = An integer that specifies the new permanent element size.
   IER = An integer variable that receives the routine's error return code.

VI. ERROR RETURNS

FORTRAN:

25 = No more memory available.

3100 = Window aggregate does not begin on 1024-word boundary.

Module name: MAPDF

VII. REFERENCED EXTERNALS

.MAPDF (System call)

FORT.LB

.FARL transfers effective address of a caller's argument list to its called subroutine's stack and counts the number of arguments transferred to the called subroutine's stack.

.FRET restores a caller's accumulators and state of carry upon exit from the called subroutine, and returns to the next instruction following the call.

VIII. COMMENTS

1) You must call MAPDF after calls to OVOPN and MEMI (if you use these routines).

2) MAPDF is the second routine in sequence you need to call to set up extended memory.

3) You may only call the first form of MAPDF once (see CALLING METHOD). If you call MAPDF a second time, you will get a error.

4) You can call the second form of MAPDF as many times as you like.

5) After the call to MAPDF (form 1), blocks zero through WINSIZ - 1 of extended memory are mapped to the window. Blocks zero through WINSIZ - 1 are initialized to the data in the window.

6) After swapping to another program and returning, you must redefine your window (form 1) before using any other feature of the extended memory library. This is the only exception to comment 3.

7) The window will always be referencing some block(s) in extended memory.

8) Arguments BLOCKS and WINSIZ are in terms of 1024-word blocks.

9) If your window is not aligned on a 1024-word boundary, and if you use the error checking routine, ERROR, to see if an error has occurred, the following message will be typed on the console:

UNKNOWN ERROR CODE 6031: XXX.SV

where XXX is the name of the program. 6031 is the octal representation of 3097 decimal. This is RDOS's reaction to an unknown system error reported to routine ERROR. The error return for a window not on a 1024-word boundary (FEW1K) was taken from the FORTRAN 5 error file, F5ERR.FR, Revision 6.10. Please refer to the program's load line and make sure that the local switch "/N" is a multiple of 1024 decimal and precedes the routine that defines the

Module name: MAPDF

window. If that doesn't work, look in the main program and make sure that the common block that contains the window is the last common block listed or the only common block listed.

10) A good way of determining the window size (WINSIZ) is by the following formula:

```
WINSIZ = (ES * DW) / 1024
```

where ES is the element size and has the following values:

```
1 for INTEGER
2 for REAL
4 for COMPLEX
4 for DOUBLE PRECISION
8 for DOUBLE PRECISION COMPLEX
```

and DW is the dimension of the window (a multiple of 1024).

e.g., You have a window whose type is REAL and whose dimension is 2048. By the formula, WINSIZ is 4.

```
(2 * 2048) / 1024 = 4
```

11) The blocks in the window and in extended memory are both ordered relative to zero. The window is ordered consecutively from 0 to WINSIZ - 1. Extended memory is ordered from 0 to BLOCKS - 1. RDOS has an elaborate scheme that numbers the blocks in extended memory (Data General Corporation, 1979). You can't assume that block N in extended memory will always be in the same location on each run.

```
C         FILES ACCESSED
C
C            GCAST = GRIDDED DATA. (INPUT/OUTPUT)
C            GSAVE = DIGITAL VALUES/GRID FIELDS. (INPUT/OUTPUT)
C             GELE = ATTRIBUTES OF ELEMENTS. (INPUT/OUTPUT)
C
C         EXTENDED MEMORY
C
C          BLOCK 0  =  SCRATCH BLOCK - ANY SUBROUTINE CAN USE IT FOR THE DUR-
C                      ATION OF THAT SUBROUTINE.
C          BLOCK 1  =  ASCII DISPLAY INSTRUCTIONS FOR MAP BACKGROUND.
C          BLOCK 2  =  ASCII DISPLAY INSTRUCTIONS FOR GRID MENU OPTIONS.
C          BLOCK 3  =  PACKED DATA FROM WHICH IDATA() IS DETERMINED FROM.
C          BLOCK 4  =  ASCII DISPLAY INSTRUCTIONS FOR GRID DISPLAY.
C
C         NONSYSTEM ROUTINES CALLED.
C            OCHN,GSET,WRCLS,MODIFY
C
      EXTERNAL OGSET,OMOD
      COMMON/VTIME/JMON(12),IDUM(4),LEAP
      COMMON/GDMDIR/IGCHN,ICON,ICHNL,IOVRLY
      COMMON/WIN/IWINDOW(1024)  ; NOTICE THE LOCATION OF THE COMMON BLOCK
      DIMENSION IBOXES(4,MBOX),IDATE(3)
      DIMENSION NCLD(3,MCLOUD)
      DATA JMON/31,28,31,30,31,30,31,31,30,31,30,31/
C
      IZERO=0
      ICON=-1
      ICHNL=-1
      IOVRLY=1
C
C        OPEN CHANNEL TO THE OVERLAY FILE.
      CALL GCHN(IOV,IER)
      CALL OVOPN(IOV,"GMOD.OL",IER)
      CALL OVLOD(IOV,OGSET,IZERO,IER)
C
C        DEFINE YOUR WINDOW.
      CALL VMEM(NBLOCKS,IER)
      CALL MAPDF(NBLOCKS,IWINDOW,1,IER)
      CALL ERROR(IER,"MAPDF ERROR")
C
C        OPEN CHANNELS TO THE DATA FILES.
      CALL OCHN("$GDM",IGCHN)
      CALL OCHN("GCAST",IGCCHN)
      CALL OCHN("GELE",IGECHN)
      CALL OCHN("GSAVE",IGSCHN)
```

Example 1. A partial subroutine that shows how to set up the window and extended memory (VMEM and MAPDF).

```
C         EXTENDED MEMORY
C
C              BLOCK 0 TO 4  =  GRAPHIC INSTRUCTIONS FOR WIND BARBS.
C              BLOCK 5 TO 9  =  GRAPHIC INSTRUCTIONS FOR WIND SPEED, IF /S
C                               SWITCH IS USED.
C            BLOCK 10 TO 14  =  GRAPHIC INSTRUCTIONS FOR U-COMPONENT, IF /U
C                               SWITCH IS USED.
C            BLOCK 15 TO 19  =  GRAPHIC INSTRUCTIONS FOR V-COMPONENT, IF /V
C                               SWITCH IS USED.
C
      LOGICAL TOGGLE,FLAG,SWITCH(4),TOG,SKIP,EVERY1,EOF
      DIMENSION ITIME(3),NWORD(4),IEX(4),LOC(4),IHS(9)
      COMMON/BLK/IFN(6),IBDR(11),JDAY(2,7),JMO(2,12),JPRES(MPRES),
     .           JFT(MPRES),MB(2),IFEET(4),MONTHS(12,0:1),LABEL(6,3)
      COMMON/WIN/IWINDOW(5120)
      DATA IBDR/0,0,4095,0,4095,3071,0,3071,0,0,-1/
      DATA JDAY/"SUN MON TUE WED THU FRI SAT "/, MB/"MB"/, IFEET/"1000 FT"/
      DATA JMO/"JAN FEB MAR APR MAY JUN JUL AUG SEP OCT NOV DEC "/
      DATA JPRES/1000,900,800,700,600,500,400,300,250,200,150,100/
      DATA MONTHS/0,31,59,90,120,151,181,212,243,273,304,334,
     .            0,31,60,91,121,152,182,213,244,274,305,335/
      DATA LABEL/"WIND SPEED",0,"U-COMPONENT","V-COMPONENT"/
C
C        INITIALIZE SWITCHES.
      IS='<0>S'
      IU='<0>U'
      IV='<0>V'
C
C        SET YOUR DEFAULTS.
      IWINSIZ=5
      NHOURS=16
      JZMIN=0
C
C        SET UP EXTENDED MEMORY.
      CALL VMEM(NBLOCKS,IER)
      CALL MAPDF(NBLOCKS,IWINDOW,IWINSIZ,IER)
      CALL ERROR(IER,"MAPDF ERROR")
C
C        READ COMMAND LINE FOR SWITCHES.
      CALL FCOM(ICH,IER)
C        LOOK FOR GLOBAL SWITCHES.
      CALL COMCM(ICH,IBUF,JBUF,IER)
      SWITCH(1)=-1                    ; WIND BARBS
      SWITCH(2)=ISWSET(JBUF,IS)       ; WIND SPEED
      SWITCH(3)=ISWSET(JBUF,IU)       ; U-COMPONENT
      SWITCH(4)=ISWSET(JBUF,IV)       ; V-COMPONENT
```

Example 2. A partial subroutine that shows how to set up the window and extended memory (VMEM and MAPDF).

REMAP

I. IDENTIFICATION

Module name: REMAP
Date: May 11, 1988
Function: Window transfer.
Language: Assembly

II. PURPOSE

Does a logical window transfer by placing blocks from the extended memory address area into the window.

III. ENTRY POINTS

REMAP

IV. CALLING METHOD

1) CALL REMAP(WB,MB,[NB,]IER)

2) CALL REMAP(BN,IER)

V. ARGUMENTS

1) WB - An integer that specifies the starting block number, relative to zero, in the window.
   MB - An integer that specifies the starting block number, relative to zero, in extended memory.
   NB - An optional integer variable that specifies the number of blocks you want to remap to the window area. If omitted, one block is remapped to the window.
   IER - An integer variable that receives the routine's error return code.

2) BN - An integer that specifies the block number in extended memory that gets mapped to block zero of the window.
   IER - An integer variable that receives the routine's error return code.

VI. ERROR RETURNS

FORTRAN:

29 - Illegal starting address.

VII. REFERENCED EXTERNALS

.REMAP (Task call)

Module name: REMAP

FORT.LB

.FARL transfer effective address of a caller's argument list to its called subroutine's stack and counts the number of arguments transferred to the called subroutine's stack.

.FRET restores a caller's accumulators and state of carry upon exit from the called subroutine, and returns to the next instruction following the call.

VIII. COMMENTS

1) No data transfer is done in a remap operation.

2) Remapping can best be described as analogous to the FORTRAN EQUIVALENCE statement. After the remap operation occurs, the block(s) in the window and in extended memory share the same memory location. One remapping operation stays in effect until another remapping operation is done.

3) Arguments WB, MB, and NB are in terms of 1024-word blocks.

4) The difference between form 1 and form 2 is the number of blocks that get mapped to the window and where the blocks get mapped to. Form 1 can map multiple blocks anywhere in the window (0 to WINSIZ - 1) and form 2 maps one block to block 0 of the window. You can use form 2 when you know you are only using block 0 of the window.

```
      OVERLAY ODDAT
      PARAMETER LEAD=0
      PARAMETER NMISS=9999
      PARAMETER JCHAR=3
      PARAMETER IOPT=-1
      SUBROUTINE DISDAT(IDATA,IRV,MAXCOL,MAXROW,IX,IY,IGCHN,ICON,ICHNL,
     .                  IOVRLY,IZ,IGR,IUP,IBRT,NBLOCK,IRC)
C
C        NOVEMBER 1987        MARK LEAPHART    TDL FTS 427-7639
C        FORTRAN IV/REV 5.57  DG ECLIPSE S230  RDOS/REV 6.17
C        PURPOSE
C           CONVERT NUMERICAL DATA FROM IDATA() INTO ASCII AND THEN
C           DISPLAY IT ON THE SCREEN.
C        VARIABLES
C              IDATA()  -  CONTAINS DATA FROM SELECTED ELEMENT AND
C                          PROJECTION HOUR.
C                IRV()  -  TELLS IF THE GRID POINT IS ON/OFF.
C                             0: OFF  1: ON
C               MAXCOL  -  MAXIMUM NUMBER OF COLUMNS DISPLAYABLE ON THE
C                          SCREEN.
C               MAXROW  -  MAXIMUM NUMBER OF ROWS DISPLAYABLE ON THE SCREEN.
C                IX,IY  -  THE ASCII DATA, HELD IN IASCII(), STARTS AT
C                          THESE COORDINATES.
C                IGCHN  -  CHANNEL TO THE GDM.
C                 ICON  -  CONSOLE OF THE GDM YOU ARE USING.
C                ICHNL  -  CHANNEL OF THE CONSOLE YOU ARE USING.
C               IOVRLY  -  OVERLAY OF THE CHANNEL YOU ARE USING.
C                   IZ  -  ZOOM LEVEL (0-4):
C                          0:  1:1  1:  4:1  2:  9:1  3:  16:1  4:  25:1
C                  IGR  -  TELLS WHETHER OR NOT TO LOAD THE GDM INSTRUC-
C                          TIONS INTO IASCII().
C                          0: DON'T LOAD  1: LOAD
C                  IUP  -  TELLS WHAT TO UPDATE:
C                          0: UPDATE ONLY GRIDPOINTS THAT ARE ON.
C                          1: WHOLE GRID.
C                          2: SCREEN.
C                 IBRT  -  USE BRITENESS LEVELS?  0: NO  1: YES
C               NBLOCK  -  BLOCK (1024-WORD) IN EXTENDED MEMORY WHERE THE
C                          INSTRUCTIONS FOR THE ASCII REPRESENTATIONS OF THE
C                          DATA IS STORED.
C                  IRC  -  RETURN CODE.
C        INTERNAL
C             IASCII()  -  CONTAINS ASCII CHARACTERS MASKED FOR BRITENESS.
C                JCHAR  -  NUMBER OF ASCII BYTES TO CONVERT.
C                NMISS  -  MISSING VALUE INDICATOR.
C                 LEAD  -  SUPPRESS OR INSERT LEADING ZEROES.
C                          0: SUPPRESS  1: INSERT
C                 NPTR  -  WHERE THE ASCII CHARACTERS START IN IASCII().
C               NBYTES  -  NUMBER OF BYTES WRITTEN TO THE SCREEN.
```

Example 3. A partial subroutine that shows the use of REMAP. (Continued on the following page.)

```
C                IOPT  =  WRITE OPTION:
C                           1: CLEAR CHANNEL AND WRITE.
C                           2: CLEAR OVERLAY WITHIN CHANNEL AND WRITE.
C                           OTHERWISE JUST WRITE.
C                LINE  =  NUMBER OF WORDS PER LINE (PACK).
C                IOFF  =  VALUE REPRESENTING OFF.
C             IPOINT()  =  HOLDS THE UNPACKED ASCII REPRESENTATION FOR THE
C                          GRIDPOINT.
C             ITHOLD()  =  HOLDS THE THRESHOLDS OF THE NINE LEVELS OF
C                          BRITENESS.
C                INCCW  =  NUMBER OF PACKED CARRIAGE RETURNS WORDS.
C               NWORDS  =  NUMBER OF WORDS OF DISPLAY INSTRUCTIONS.
C                   JP  =  CURRENT POSITION IN IPOINT().
C                    N  =  START POSITION OF A LINE WITHIN IASCII().
C                   IP  =  CURRENT POSITION WITHIN A LINE.
C                 JVAL  =  TEMPORARY VARIABLE USED INSTEAD OF AN ARRAY
C                          ELEMENT (IDATA(I,J)).
C                I9990  =  ANY VALUE GREATER THAN THIS VALUE IS CONSIDERED
C                          MISSING.
C                 ICRS  =  A WORD CONTAINING TWO CARRIAGE RETURNS.
C        NONSYSTEM ROUTINES CALLED.
C           INTRC,CONVRT,WRCLS,INITAR,TROUBL
C
      COMMON/II/IASCII(1024)
      COMMON/WIN/IWINDOW(1024)
      DIMENSION IDATA(MAXCOL,MAXROW),IRV(MAXCOL,MAXROW)
      DIMENSION ITHOLD(8),IPOINT(6)
C
      IZERO=0
      IRC=IZERO
      IAVRG=IZERO
      ITWO=2
C
C       REMAP BLOCK (NBLOCK) INTO THE WINDOW.
      CALL REMAP(NBLOCK,IER)
      IF(IER.NE.1)CALL TROUBL("REMAPPING ERROR ","IN DISDAT")
C
      ICRS='<15><15>'
      NPTR=5
      IF(IGR.NE.1)GO TO 5
C
C        LOAD IN INSTRUCTIONS ONCE.
      CALL INTRC(IWINDOW,IP,ICON,ICHNL,IOVRLY,IX,IY,IZ,IRC)
      IGR=IZERO
C
C        UPDATE THE GRID FIELD.
 5    IP=NPTR
      IF(IUP.EQ.ITWO)GO TO 60
      IF(IUP.NE.1)GO TO 40
     .
     .
```

Example 3 (cont.)

```
      SUBROUTINE CDVVT(IWINDOW,IDATA,XIN,U,LENGTH,MAXCOL,MAXROW,IFCOL,LCOL,
     .                 NROW,NB,SCALE)
C
C        NOVEMBER 1988        MARK LEAPHART    TDL FTS 427-7639
C        FORTRAN IV/REV 5.57  DG ECLIPSE S230  RDOS/REV 6.17
C        PURPOSE
C           COMPUTES TWO OF THE SIX LINEAR KINEMATIC PROPERTIES OF A FLOW
C           FIELD (DIVERGENCE AND VORTICITY) AT THE CENTROID OF A TRIANGLE
C           OF STATIONS BASED ON THE LINEAR VECTOR POINT FUNCTION (LVPF)
C           METHOD DESCRIBED BY ZAMORA ET AL. (1987)
C
C        VARIABLES
C             IWINDOW()  -  USED TO ACCESS U- AND V-COMPONENTS OF THE THREE
C                           STATIONS FROM EXTENDED MEMORY AND STORE DIVER-
C                           GENCE AND VORTICITY VALUES IN EXTENDED MEMORY.
C               IDATA()  -  BUFFER ARRAY THAT CONTAINS THE VORTICITY VALUES
C                           AFTER COMPUTATIONS THE VORTICITY VALUES ARE
C                           STORED IN EXTENDED MEMORY.
C              XIN(I,J)  -  (I=1) THE CONTRIBUTION TO THE U-COMPONENT CAL-
C                           CULATED AT THE CENTROID OF THE TRIANGLE BY EACH OF
C                           THE THREE STATIONS AT THE POINTS OF THE TRIANGLE.
C                           (I=2) THE CONTRIBUTION TO THE V-COMPONENT CAL-
C                           CULATED AT THE CENTROID OF THE TRIANGLE BY EACH OF
C                           THE THREE STATIONS AT THE POINTS OF THE TRIANGLE.
C                           (I=3) THE CONTRIBUTION TO THE STRETCHING
C                           DEFORMATION CALCULATED AT THE CENTROID OF THE TRI-
C                           ANGLE BY EACH OF THE THREE STATIONS AT THE POINTS
C                           OF THE TRIANGLE.
C                           (I=4) THE CONTRIBUTION TO THE SHEARING DEFORMATION
C                           CALCULATED AT THE CENTROID OF THE TRIANGLE BY EACH
C                           OF THE THREE STATIONS AT THE POINTS OF THE TRI-
C                           ANGLE.
C                           (I=5) THE CONTRIBUTION TO THE DIVERGENCE CAL-
C                           CULATED AT THE CENTROID OF THE TRIANGLE BY EACH OF
C                           THE THREE STATIONS AT THE POINTS OF THE TRI-
C                           ANGLE.
C                           (I=6) THE CONTRIBUTION TO THE VORTICITY CAL-
C                           CULATED AT THE CENTROID OF THE TRIANGLE BY EACH OF
C                           THE THREE STATIONS AT THE POINTS OF THE TRI-
C                           ANGLE.
C                           STATION 1 (J=1,2) STATION 2 (J=3,4) STATION 3
C                           (J=5,6).
C                  U()  -  THE THREE NON-COLINEAR HORIZONTAL WIND MEASURE-
C                           MENTS: (U1,V1,U2,V2,U3,V3).
C               LENGTH  -  NUMBER OF KINEMATIC PROPERTIES.
C         MAXCOL,MAXROW  -  DIMENSIONS OF IDATA().
C                IFCOL  -  FIRST COLUMN IN IWINDOW() THAT HAVE DATA.
C                 LCOL  -  LAST COLUMN IN IWINDOW() THAT HAVE DATA.
C                 NROW  -  NUMBER OF ROWS IN IWINDOW() THAT HAVE DATA.
C                   NB  -  BLOCK IN EXTENDED MEMORY WHERE THE DIVERGENCE
```

Example 4. A subroutine that shows the use of REMAP. (Continued on the following page.)

```
C                          VALUES ARE TO BE PLACED.  VORTICITY VALUES ARE
C                          PLACED IN BLOCK NB+1.
C                SCALE  =  SCALING FACTOR.  ALL VALUES ARE IN TERMS OF
C                          0.00001 RADIANS PER SECOND.
C        INTERNAL
C                PTFIVE  =  ROUND UP FACTOR.
C                  ICP   =  ROW INDEX, MAXCOL*(J-1) FOR (J=1,NROW).
C                  NCP   =  COLUMN INDEX, ICP+I FOR (I=IFCOL,LCOL).
C                   JB   =  BLOCK NUMBER OF U- AND V-COMPONENTS.
C                HOLD()  =  HOLDS THE DIVERGENCE AND VORTICITY VALUE AT A
C                          GRID POINT.  (1) DIVERGENCE  (2) VORTICITY
C                   IP   =  CURRENT POSITION IN HOLD().
C                 ICOL   =  CURRENT COLUMN POSITION IN XIN().
C                  ICR   =  CURRENT ROW POSITION IN XIN() AND U().
C                 IDIVG  =  COLUMN IN XIN() WHEN MULTIPLIED BY U() WILL
C                          GIVE YOU THE DIVERGENCE VALUE AT THAT GRID
C                          POINT.
C                 IVORT  =  COLUMN IN XIN() WHEN MULTIPLIED BY U() WILL
C                          GIVE YOU THE VORTICITY VALUE AT THAT GRID
C                          POINT.
C
C        EXTENDED MEMORY
C         BLOCK 0,1  =  U- & V-COMPONENTS OF STATION 1 FOR GRID POINTS (I,J)
C                       (I=IFCOL,LCOL) (J=1,NROW)
C         BLOCK 2,3  =  U- & V-COMPONENTS OF STATION 2 FOR GRID POINTS (I,J)
C                       (I=IFCOL,LCOL) (J=1,NROW)
C         BLOCK 4,5  =  U- & V-COMPONENTS OF STATION 3 FOR GRID POINTS (I,J)
C                       (I=IFCOL,LCOL) (J=1,NROW)
C           BLOCK 6  =  DIVERGENCE VALUES FOR GRID POINTS (I,J) (I=IFCOL,LCOL)
C                       (J=1,NROW)
C           BLOCK 7  =  VORTICITY VALUES FOR GRID POINTS (I,J) (I=IFCOL,LCOL)
C                       (J=1,NROW)
C
C        NONSYSTEM ROUTINES CALLED.
C           WMOV
C
      DIMENSION IWINDOW(1024),IDATA(MAXCOL,MAXROW),XIN(LENGTH,LENGTH),
     .          U(LENGTH),HOLD(2)
      PTFIVE=.000005
      IDIVG=5
      IVORT=6
      IZERO=0
      ZERO=IZERO
C
C        COMPUTE DIVERGENCE AND VORTICITY FOR EVERY GRID POINT WITH DATA.
      ICP=IZERO
      DO 50 J=1,NROW
      DO 40 I=IFCOL,LCOL
      NCP=ICP+I
```

Example 4 (cont.)

```
C
C        RETRIEVE THE U- AND V-COMPONENTS FROM EXTENDED MEMORY.
      JB=IZERO
      DO 10 K=1,LENGTH
      CALL REMAP(JB,IER)
      CALL ERROR(IER,"REMAP ERROR")
      U(K)=IWINDOW(NCP)
      JB=JB+1
 10   CONTINUE
C
C        REMAP TO THE BLOCK FOR DIVERGENCE.
      CALL REMAP(NB,IER)
      IP=1
C
C        COMPUTE DIVERGENCE AND VORTICITY FOR THE GRID POINT.
      DO 30 ICOL=IDIVG,IVORT
      SUM=ZERO
      DO 20 ICR=1,LENGTH
      SUM=SUM+U(ICR)*XIN(ICOL,ICR)
 20   CONTINUE
      HOLD(IP)=SUM+PTFIVE
      IP=IP+1
 30   CONTINUE
C
C        STORE THE DIVERGENCE VALUE IN EXTENDED MEMORY
C        AND SAVE THE VORTICITY VALUE.
      IWINDOW(NCP)=HOLD(1)*SCALE
      IDATA(NCP)=HOLD(2)*SCALE
 40   CONTINUE
      ICP=ICP+MAXCOL
 50   CONTINUE
C
C        STORE THE VORTICITY VALUES IN EXTENDED MEMORY.
      CALL REMAP(NB+1,IER)
      CALL WMOV(IDATA,MAXCOL*NROW,IWINDOW)
C
      RETURN
      END
```

Example 4 (cont.)

VRW

I. IDENTIFICATION

Module name: VRW
Date: May 11, 1988
Function: EXTENDED DIRECT BLOCK I/O.
Language: Assembly

II. PURPOSE

EWRB writes a series of disk blocks from extended memory to a random or contiguously organized disk file. The contents of the window are unchanged, as are the contents of extended memory.

ERDB reads a series of disk blocks from a random or contiguously organized file into extended memory. The current contents of the window are unchanged unless one of the extended memory blocks read into is in the window.

III. ENTRY POINTS

EWRB

ERDB

IV. CALLING METHOD

```
CALL EWRB(CHAN,DB,MB,BC,[PC,]IER)

CALL ERDB(CHAN,DB,MB,BC,[PC,]IER)
```

V. ARGUMENTS

CHAN = An integer that specifies the RDOS channel on which the disk file was opened.
DB = An integer that specifies the initial disk block of the disk file you want to write/read.
MB = An integer that specifies the initial disk block of extended memory into which data are written/read.
BC = An integer that specifies the number of disk blocks you want to transfer. The maximum is 127 disk blocks.
PC = An optional integer variable that receives the number of disk blocks transferred successfully in the event an End-of-File condition is met or when disk space is exhausted.
IER = An integer variable that receives the routine's error return code.

VI. ERROR RETURNS

FORTRAN (BOTH):

3 = Illegal channel number.

Module name: VRW

6 - Illegal command for device.

7 - Not a randomly- or contiguously-organized file.

9 - End of file.

16 - No file is open on this channel.

35 - File not accessible by direct I/O.

63 - Address outside of address space.

68 - Disk time-out occurred.

FORTRAN (EWRB):

11 - File is write protected.

26 - Disk space is exhausted.

FORTRAN (ERDB):

10 - File is read protected.

27 - File read error (Mag tape or cassette: bad tape).

VII. REFERENCED EXTERNALS

.EWRB (System call)

.ERDB (System call)

FORT.LB

.FARL transfers effective address of a caller's argument list to its called subroutine's stack and counts the number of arguments transferred to the called subroutine's stack.

.FRET restores a caller's accumulators and state of carry upon exit from the called subroutine, and returns to the next instruction following the call.

VIII. COMMENTS

1) Do not write/read more than 127 disk blocks at a time. If you do, beware; you can crash your system and possibly get a MAP.DR error on the file you are writing/reading to/from.

2) Argument DB is in terms of disk blocks, not 1024-word blocks.

3) The value of argument MB reflects the location in extended memory in terms of disk blocks. If there are N 1024-word blocks avail-

Module name: VRW

able, then MB can range from 0 through (N * 4) - 1. Extended memory can be pictured as:

1024-word Block	Absolute Disk Block	Relative Disk Block
0	0	0
	1	1
	2	2
	3	3
1	4	0
	5	1
	6	2
	7	3
2	8	0
	9	1
	10	2
	11	3
.	.	.
.	.	.
N - 1	(N * 4) - 4	0
	(N * 4) - 3	1
	(N * 4) - 2	2
	(N * 4) - 1	3

```
      SUBROUTINE RDGRD(IFILE,IDATA,IWINDOW,MAXCOL,MAXROW,MROW,IGCHN,NPR,
     .                 NPRJ,KELM,JBLK,NBSR,NBITS,NBLOCK,IRC)
C
C        NOVEMBER 1987         MARK LEAPHART    TDL FTS 427-7639
C        FEBRUARY 1988         MARK LEAPHART    REVISED
C        FORTRAN IV/REV 5.57   DG ECLIPSE S230  RDOS/REV 6.17
C        PURPOSE
C           READ IN THE DATA FOR THE CURRENT ELEMENT AND PROJECTION HOUR
C           AND UNPACK IT.
C
C        VARIABLES
C              IFILE()  =  NAME OF THE FILE YOU ARE READING FROM.
C              IDATA()  =  RECEIVES THE DATA OF THE SELECTED ELEMENT AND
C                          PROJECTION HOUR.
C            IWINDOW()  =  DATA AREA THROUGH WHICH REFERENCES TO EXTENDED
C                          MEMORY ARE MADE.  ALSO HOLDS THE PACKED DATA.
C               MAXCOL  =  MAXIMUM NUMBER OF COLUMNS DISPLAYABLE ON THE
C                          SCREEN.
C               MAXROW  =  MAXIMUM NUMBER OF ROWS DISPLAYABLE ON THE SCREEN.
C                 MROW  =  NUMBER OF ROWS IT TAKES TO MAKE IDATA() TWO DISK
C                          BLOCKS LONG.
C                IGCHN  =  CHANNEL TO THE FILE HELD IN IFILE().
C                  NPR  =  CURRENT PROJECTION HOUR.
C                 NPRJ  =  TOTAL NUMBER OF PROJECTIONS.
C                 KELM  =  ELEMENT INDEX.  THERE ARE 45 TOTAL ELEMENTS BUT
C                          ONLY 10 DISTINCT ELEMENTS.
C                            (1)  TEMP
C                            (2)  DEW
C                            (3)  12HR POP
C                            (4)  12HR QPF
C                            (5)  SNOW AMOUNT
C                            (6)  WIND DIRECTION
C                            (7)  WIND SPEED
C                            (8)  VISIBILITY
C                            (9)  CLOUDS
C                            (10) WEATHER
C                 JBLK  =  BLOCK NUMBER OF THE ELEMENTS' ATTRIBUTES.
C                 NBSR  =  NUMBER OF BITS TO SHIFT RIGHT.
C                NBITS  =  NUMBER OF BITS THE ELEMENT VALUE TAKES UP.
C               NBLOCK  =  BLOCK (1024-WORD) IN EXTENDED MEMORY WHERE YOU
C                          WANT TO READ IN THE DATA.
C                  IRC  =  RETURN CODE.
C
C        INTERNAL
C                  NBLK  =  NUMBER OF DISK BLOCKS TO READ.
C                  IBLK  =  STARTING DISK BLOCK TO READ.
C                  MASK  =  PATTERN OF BITS THAT PRESERVE THE ELEMENT VALUE.
C                  IVAL  =  VALUE OF THE UNPACKED ELEMENT VALUE.
C                    NB  =  BIT NB TELLS YOU THAT THE VALUE, BEFORE PACKING,
C                           WAS A NEGATIVE VALUE AND IT WAS LOST IN THE
```

Example 5. A subroutine that shows the use of ERDB and REMAP. (Continued on the following page.)

```
C                          PACKING.  SO, WHEN BIT NB IS SET, THEN YOU WANT
C                          TO MAKE A FULL WORD NEGATIVE VALUE.
C              POSITIVE  =  LOGICAL VARIABLE THAT TELLS YOU THAT THE ELEMENT
C                          CAN HAVE ONLY POSITIVE VALUES.
C                             TRUE - POSITIVE ONLY
C                            FALSE - POSITIVE AND NEGATIVE
C                 NMIS  =  MISSING VALUE INDICATOR.
C                   IP  =  MAXCOL*(K-1)+J FOR (K=1,MAXROW) (J=1,MAXCOL).
C                IDBLK  =  DISK BLOCK IN EXTENDED MEMORY WHERE THE DATA IS TO
C                          BE READ IN.
C                  NEG  =  CONTAINS A BIT PATTERN WHEN OR'D WITH IVAL WILL
C                          GIVE YOU A NEGATIVE NUMBER.
C        NONSYSTEM ROUTINES CALLED.
C           TROUBL
C
      DIMENSION IFILE(10),IWINDOW(1024)
      DIMENSION IDATA(MAXCOL,MROW)
      LOGICAL POSITIVE
      IRC=0
      IP=0
      ITWO=2
      ITEN=10
      NMIS=9999
C
C        DETERMINE WHICH MASK TO USE TO MAKE A NEGATIVE
C        NUMBER IF THERE IS ONE.
      NEG=177400K
      IF(KELM.EQ.ITEN)NEG=174000K
      POSITIVE=KELM.NE.1.AND.KELM.NE.ITWO.AND.KELM.NE.ITEN
C
C        FIGURE OUT THE STARTING BLOCK TO READ.
      IBLK=(NPR-1)+(JBLK-1)*(NPRJ+1)
      IBLK=IBLK+IBLK
      IDBLK=NBLOCK*4
      NBLK=ITWO
C
C        READ THE DISK BLOCKS INTO BLOCK (NBLOCK) OF EXTENDED MEMORY.
      CALL ERDB(IGCHN,IBLK,IDBLK,NBLK,IER)
      IF(IER.NE.1)CALL TROUBL("READING ",IFILE)
C
      MASK=ITWO**NBITS-1
      NB=NBITS-1
C
C        REMAP BLOCK (NBLOCK) INTO BLOCK ZERO OF THE WINDOW.
      CALL REMAP(NBLOCK,IER)
      IF(IER.NE.1)CALL TROUBL("REMAPPING ERROR ",IFILE)
C
C        UNPACK THE DATA OF THE ELEMENT.
      DO 20 K=1,MAXROW
      DO 20 J=1,MAXCOL
      IP=IP+1
```

Example 5 (cont.)

```
C         ISOLATE THE VALUE.
      IVAL=ISHFT(IWINDOW(IP),NBSR)
      IVAL=IAND(IVAL,MASK)
C
C         SEE IF THE VALUE IS MISSING.
      IF(IVAL.EQ.MASK)IVAL=NMIS
      IF(IVAL.EQ.NMIS.OR.POSITIVE)GOTO 10
C
      IF(ITEST(IVAL,NB))IVAL=IOR(IVAL,NEG)
      IVAL=IVAL+100
C
 10   IDATA(J,K)=IVAL
 20   CONTINUE
C
      RETURN
      END
```

Example 5 (cont.)

```
      SUBROUTINE STELMS(MAXELM,ISDBLK,IGCHN,NDB,NBLK,IWINDOW,IWS)
C
C        DECEMBER 1988         MARK LEAPHART    TDL FTS 427-7639
C        FORTRAN IV/REV 5.57   DG ECLIPSE S230  RDOS/REV 6.17
C        PURPOSE
C           STORE BLOCKS OF ELEMENTS (TEMP, DEW, WEATHER, ETC.) CONSECU-
C           TIVELY IN A DISK FILE.  THE ORDER IS DETERMINED FROM THE ELEMENT
C           DIRECTORY LOCATED IN THE FIRST BLOCKS, ISDBLK, OF THE FILE.
C
C        VARIABLES
C             MAXELM  =  MAXIMUM NUMBER OF ELEMENTS.
C             ISDBLK  =  DISK BLOCK OF THE ELEMENT DIRECTORY.
C              IGCHN  =  CHANNEL TO THE DATA FILE.
C                NSB  =  BLOCK (1024-WORD) IN EXTENDED MEMORY TO PLACE THE
C                           ELEMENT DIRECTORY.
C               NBLK  =  NUMBER OF BLOCKS (1024-WORD) THE ELEMENT DIRECTORY
C                           TAKES UP.
C          IWINDOW()  =  HOLDS THE ELEMENT DIRECTORY.
C                IWS  =  SIZE OF THE WINDOW (A MULTIPLE OF 1024).
C        INTERNAL
C                MBLK  =  NUMBER OF DISK BLOCKS THE ELEMENT DIRECTORY TAKES UP.
C               NELMS  =  NUMBER OF ELEMENTS TO PROCESS.
C                  IP  =  CURRENT POSITION IN THE ELEMENT DIRECTORY.
C               ICBLK  =  CURRENT DISK BLOCK IN THE DATE FILE.
C                 IPC  =  NUMBER OF BLOCKS WRITTEN TO THE DISK FILE ON ONE
C                           WRITE.
C
      DIMENSION IWINDOW(IWS)
C
C        READ THE ELEMENT DIRECTORY INTO EXTENDED MEMORY AND
C        PLACE IT IN THE WINDOW.
      MBLK=NBLK*4
      CALL ERDB(IGCHN,ISDBLK,NSB*4,MBLK,IER)
      CALL ERROR(IER,"ERDB ERROR STELMS")
      CALL REMAP(0,NSB,NBLK,IER)
      CALL ERROR(IER,"REMAP ERROR STELMS")
C
C        GET THE NUMBER OF ELEMENTS TO PROCESS.
      NELMS=IWINDOW(1)
      IF(NELMS.GT.MAXELM)GO TO 20
C
C        INITIALIZE POINTERS.
      ITWO=2
      IP=ITWO
      ICBLK=ISDBLK+MBLK
C
C        PROCESS ALL THE ELEMENTS.
      DO 10 K=1,NELMS
      NDB=IWINDOW(IP+1)
```

Example 6. A subroutine that shows the use of EWRB and ERDB. (Continued on the following page.)

```
      CALL EWRB(IGCHN,ICBLK,IWINDOW(IP),NDB,IPC,IER)
      IF(IER.NE.1)GO TO 30
      ICBLK=ICBLK+NDB
      IP=IP+ITWO
 10   CONTINUE
 20   RETURN
C
C        FATAL ERROR.
 30   TYPE'EWRB ERROR STELMS IER=',IER
      TYPE'NDB,ICBLK,IWINDOW(IP),IPC,IP=',NDB,ICBLK,IWINDOW(IP),IPC,IP
      STOP
      END
```

Example 6 (cont.)

VDL

I. IDENTIFICATION

Module name: VDL
Date: May 11, 1988
Function: EXTENDED DIRECT BLOCK I/O
Language: Assembly

II. PURPOSE

VDUMP copies blocks from extended memory to a disk file.

VLOAD initializes extended memory to the contents of a disk file. If the disk file is smaller than the amount of extended memory currently defined, VLOAD will initialize up to the size of the disk file.

III. ENTRY POINTS

VDUMP

VLOAD

IV. CALLING METHOD

CALL VDUMP(CHAN,NDB,SDB,SMB,[BC,]IER)

CALL VLOAD(CHAN,NDB,SDB,SMB,[BC,]IER)

V. ARGUMENTS

CHAN = An integer that specifies the RDOS channel on which the disk file was opened.
NDB = An integer that specifies the number of disk blocks you want to copy from/to extended memory.
SDB = An integer that specifies the initial disk block of the disk file you want to write/read.
SMB = An integer that specifies the initial disk block of extended memory into which data are written/read.
BC = An optional integer variable that receives the number of disk blocks successfully written/read if the write/read operation can't be completed due to an End-of-File condition or disk space exhaustion.
IER = An integer variable that receives the routine's error return code.

VII. ERROR RETURNS

FORTRAN (BOTH):

3 = Illegal channel number.

6 = Illegal command for device.

Module name: VDL

7 = Not a randomly- or contiguously-organized file.

16 = No file is open on this channel.

35 = File is accessible by direct I/O.

63 = Address outside address space.

68 = Disk time-out occurred.

FORTRAN (VDUMP):

11 = File is write protected.

FORTRAN (VLOAD):

10 = File is read protected.

27 = File read error (Mag tape or cassette: bad tape).

VII. REFERENCED EXTERNALS

FORT.LB

.FARL transfers effective address of a caller's argument list to its called subroutine's stacks and counts the number of arguments transferred to the called subroutine's stack.

.FRET restores a caller's accumulators and state of carry upon exit from the called subroutine, and returns to the next instruction following the call.

.FCAL calls a subroutine which has no page zero entry, or calls a subroutine which has a page zero entry without using the page zero entry.

XMEM.LB

EWRB writes a series of disk blocks from extended memory to a random or contiguously organized file.

ERDB reads a series of disk blocks from a random or contiguously organized file to extended memory.

VIII. COMMENTS

1) Because the most you can read or write at one time with ERDB and EWRB is 127 disk blocks, you must make multiple calls to these routines when you want to read or write more than 127 disk blocks. VLOAD and VDUMP have been developed with this in mind. VLOAD and VDUMP should be used in place of making multiple calls to ERDB and EWRB for reading or writing large number of disk blocks.

Module name: VDL

2) Argument SDB is in terms of disk blocks, not 1024-word blocks.

3) The value of argument SMB reflects the location in extended memory in terms of disk blocks. If there are N 1024-word blocks available, then SMB can range from 0 through (N * 4) - 1. Extended memory can be pictured as:

1024-word Block	Absolute Disk Block	Relative Disk Block
0	0	0
	1	1
	2	2
	3	3
1	4	0
	5	1
	6	2
	7	3
2	8	0
	9	1
	10	2
	11	3
.	.	.
.	.	.
N - 1	(N * 4) - 4	0
	(N * 4) - 3	1
	(N * 4) - 2	2
	(N * 4) - 1	3

4) If you try to exceed the number of blocks in extended memory, VLOAD will read only the number of disk blocks it takes to exceed extended memory. If the argument BC is passed in, then BC will receive the number of disk blocks actually read.

5) If you try to read more disk blocks than you have in the file, VLOAD will recognize the EOF condition, stop, and return a 1. If the argument BC is passed in, then BC will receive the number of blocks actually read.

6) If a disk space exhaustion error occurs during VDUMP, VDUMP will recognize the error, stop, and return a 1. If the argument BC is passed in, then BC will receive the number of disk blocks actually written.

7) VDUMP will lengthen a random file if you dump more disk blocks than the original size of the file or if the number of disk blocks added to the starting disk block minus one is greater than the original size of the file.

```
      SUBROUTINE LDDATA(IFILE,IGCHN,ISMB,NSPR,NPR,MAXPR,NBLK,NELM)
C
C        DECEMBER 1988         MARK LEAPHART    TDL FTS 427-7639
C        FORTRAN IV/REV 5.57   DG ECLIPSE S230  RDOS/REV 6.17
C        PURPOSE
C           LOAD IN CONSECUTIVE HOUR'S WORTH OF DATA INTO EXTENDED
C           MEMORY.
C        VARIABLES
C             IFILE()  -  NAME OF THE FILE YOU WANT TO LOAD INTO EXTENDED
C                         MEMORY.
C               IGCHN  -  CHANNEL TO THE FILE THAT YOU WANT TO PUT INTO
C                         EXTENDED MEMORY.
C                ISMB  -  STARTING DISK BLOCK OF EXTENDED MEMORY INTO
C                         WHICH DATA IS WRITTEN.
C                NSPR  -  STARTING PROJECTION HOUR.
C                 NPR  -  NUMBER OF PROJECTION HOURS YOU WANT TO LOAD INTO
C                         EXTENDED MEMORY.
C               MAXPR  -  MAXIMUM NUMBER OF PROJECTION HOURS.
C                NBLK  -  NUMBER OF DISK BLOCKS AN ELEMENT TAKES UP.
C                NELM  -  NUMBER OF ELEMENTS PER PROJECTION HOUR.
C        INTERNAL
C                NBPR  -  NUMBER OF BLOCKS FOR ALL THE ELEMENTS FOR ONE
C                         PROJECTION HOUR.
C              ISDBLK  -  STARTING DISK BLOCK OF THE DISK FILE YOU WANT TO
C                         READ.
C                 NDB  -  NUMBER OF DISK BLOCKS YOU WANT TO READ INTO
C                         EXTENDED MEMORY.
C                 IER  -  ERROR RETURN.
C        NONSYSTEM ROUTINES CALLED.
C           TROUBL
C
      DIMENSION IFILE(10)
      IF(NSPR+NPR-1.GT.MAXPR)GO TO 10
      NBPR=NBLK*NELM
      ISDBLK=(NSPR-1)*NBPR
      NDB=NBPR*NPR
C
C        READ THE DATA INTO EXTENDED MEMORY.
      CALL VLOAD(IGCHN,NDB,ISDBLK,ISMB,IER)
      IF(IER.NE.1)CALL TROUBL("READING ",IFILE)
      RETURN
C
C        FATAL ERROR.
 10   TYPE"ERROR, CAN'T LOAD IN ALL THE DATA",NSPR,NPR
      STOP
      END
```

Example 7. A subroutine that shows the use of VLOAD.

```
      SUBROUTINE CHECK(IGCHN,ISDBLK,ID,LDBLK,IWINDOW,IWS)
C
C        DECEMBER 1988         MARK LEAPHART    TDL FTS 427-7639
C        FORTRAN IV/REV 5.57   DG ECLIPSE S230  RDOS/REV 6.17
C        PURPOSE
C           LOADS IN A FILE INTO EXTENDED MEMORY AND GOES THROUGH AND CHECKS
C           THE VALIDITY OF THE DATA (DONE IN CHECKR).  IF ENOUGH OF THE DATA
C           HAS BEEN ALTERED THEN THE DATA IN EXTENDED MEMORY WILL BE WRIT-
C           TEN TO DISK WITH ONE CALL AS OPPOSED WITH SEVERAL CALLS.
C        VARIABLES
C             IGCHN  =  CHANNEL TO THE DATA FILE.
C            ISDBLK  =  STARTING DISK BLOCK IN THE FILE YOU WANT TO START
C                       LOADING THE DATA.
C                ID  =  DIRECTORY ID.
C             LDBLK  =  BLOCK IN THE FILE WHERE THE DIRECTORY ID'S ARE KEPT.
C           IWINDOW()  =  HOLDS THE DATA THAT IS TO BE CHECKED.
C                IWS  =  SIZE OF THE WINDOW.
C
C        INTERNAL
C             ICBLK  =  CURRENT BLOCK IN THE DISK FILE.
C              MBLK  =  MEMORY BLOCK YOU WANT TO PUT THE DATA.
C                NB  =  NUMBER OF 1024-WORD BLOCKS READ FROM THE FILE.
C               IBC  =  NUMBER OF DISK BLOCKS READ FROM THE FILE.
C               NCB  =  CURRENT 1024-WORD BLOCK YOU ARE PROCESSING.
C              LOAD  =  THE NUMBER OF DISK BLOCKS TO LOAD INTO EXTENDED
C                       MEMORY.  IF YOU DON'T WANT TO CALCULATE THE NUMBER
C                       OF DISK BLOCKS IN EXTENDED MEMORY, JUST USE A VERY
C                       LARGE NUMBER AND VLOAD WILL STOP WHEN IT IS ABOUT TO
C                       EXCEED EXTENDED MEMORY.
C              DUMP  =  A LOGICAL VARIABLE USED TO INDICATE THAT YOU WANT TO
C                       DUMP EXTENDED MEMORY TO DISK.
C        NONSYSTEM ROUTINES CALLED.
C           CHECKR
C
      DIMENSION IWINDOW(IWS)
      LOGICAL DUMP
C
      IZERO=0
      ICBLK=ISDBLK
      LOAD=10000
      MBLK=IZERO
C
C        FILL UP EXTENDED MEMORY.
 10   CALL VLOAD(IGCHN,LOAD,ICBLK,MBLK,IBC,IER)
      CALL ERROR(IER,"VLOAD ERROR")
      IF(IBC.EQ.IZERO)GO TO 30
C
C        PROCESS THE DATA.
      NB=(IBC+1)/4
```

Example 8. A subroutine that shows the use of VDUMP and VLOAD. (Continued on the following page.)

```
      NCB=IZERO
      DO 20 K=1,NB
      CALL REMAP(NCB,IER)
      CALL ERROR(IER,"REMAP ERROR")
      CALL CHECKR(ID,IWINDOW,LDBLK,DUMP)
      NCB=NCB+1
 20   CONTINUE
C
C        SEE IF YOU NEED TO DUMP EXTENDED MEMORY.
      IF(DUMP)CALL VDUMP(IGCHN,IBC,ICBLK,MBLK,IER)
      ICBLK=ICBLK+IBC
      GO TO 10
C
 30   RETURN
      END
```

Example 8 (cont.)

SF

I. IDENTIFICATION

Module name: SF
Date: May 11, 1988
Function: Copies data to and from extended memory.
Language: Assembly

II. PURPOSE

VFETCH copies one or more elements from extended memory to an array aggregate.

VSTASH copies one or more elements into extended memory from an array aggregate.

IVF, VF, DVF, CVF, and DCVF fetch one integer, real, double precision, complex, and double precision complex number, respectively, from extended memory. DVF, CVF, and DCVF must be declared double precision, complex, and double precision complex, respectively.

III. ENTRY POINTS

VFETCH

VSTASH

IVF

VF

DVF

CVF

DCVF

IV. CALLING METHOD

```
CALL VFETCH(DATA,INDEX[,ELMS[,SIZE]])

CALL VSTASH(DATA,INDEX[,ELMS[,SIZE]])

I=IVF(INDEX)

R=VF(INDEX)

D=DVF(INDEX)

C=CVF(INDEX)

DC=DCVF(INDEX)
```

Module name: SF

V. ARGUMENTS

DATA = An array aggregate that defines the area in which data is written from/to extended memory.
INDEX = An integer that specifies the word index of the position in extended memory where the element or the first of a number of consecutive elements is to be copied.
ELMS = An integer that specifies the number of elements you want to copy. If omitted, one element is copied.
SIZE = An integer that specifies the element size for the current transfer. If omitted, the permanent element size given by the most recent call to MAPDF is used. (Note: If you are passing SIZE to VFETCH/VSTASH, you must pass ELMS also, even though ELMS may be one.
I = An integer variable.
R = A real variable.
D = A double precision variable.
C = A complex variable.
DC = A double precision complex variable.

VI. ERROR RETURNS

NONE

VII. REFERENCED EXTERNALS

XMEM.LB

REMAP does a logical window transfer by placing blocks from the extended address space into the window.

FORT.LB

.FARL transfers effective address of a caller's argument list to its called subroutine's stack and counts the number of arguments transferred to the called subroutine's stack.

.FRET restores a caller's accumulators and state of carry upon exit from the called subroutine, and returns to the next instruction following the call.

.FCAL calls a subroutine which has no page zero entry, or calls a subroutine which has a page zero entry without using its page zero entry.

.SMPY performs a multiplication of two signed integers.

VIII. COMMENTS

1) VSTASH transfers ELMS * SIZE words from the aggregate DATA to extended memory, beginning at offset (INDEX - 1) * SIZE.

Module name: SF

2) VFETCH transfers ELMS * SIZE words from extended memory, beginning at offset (INDEX - 1) * SIZE, into the aggregate DATA.

3) For IVF, VF, DVF, CVF, and DCVF to work properly, the argument in MAPDF, ELMSIZ, must be set to 1, 2, 4, 4, and 8, respectively.

4) Use VFETCH and VSTASH for placing/fetching data one right after another in/from extended memory.

5) All the routines contained in SF use the window to copy the data from and into extended memory. After each routine has finished, the window may not be referencing the same block(s) as it was before the routine was initiated.

6) No error checking is done.

```
      OVERLAY OEVIS
      SUBROUTINE EVIS(NSTA,LTAG)
C
C        MAY 1986   CHAMBERS, GLAHN    TDL   ECLIPSE (MARD)
C        MAY 1987   CHAMBERS, GLAHN    TDL   REVISED
C
C        PURPOSE
C           TO PREPARE VISIBILITY DATA FOR ANALYSIS.  LTAG( ) SET = 1
C           FOR ANY VALUE NOT IN RANGE 0 THRU 40 MI.  NOTE THAT ARCHIVED
C           HOURLY DATA HAD VISIBILITIES GT 40 MI. SET TO 40 MI.  THIS
C           VERSION USES EXTENDED MEMORY.
C
C        DATA SET USE
C           NONE.
C
C        VARIABLES
C
C          INPUT
C                NSTA  =  NUMBER OF STATIONS FOR WHICH DATA ARE AVAILABLE.
C
C          INPUT - OUTPUT
C             LTAG( )  =  0 INDICATES DATA ARE JUDGED GOOD AT THIS POINT.
C                         1 DATA IS OUT OF VALID RANGE, DO NOT USE.
C
C           INTERNAL
C               XX(J)  =  WORK ARRAY USED FOR THIS EXTENDED MEMORY VERSION.
C                         (J=1,6)
C                DATA  =  CONTAINS DATA TO ANALYZE.  THIS VARIABLE IS STORED
C                         IN EXTENDED MEMORY WITH A CALL TO VSTASH, AND IS
C                         RETRIEVED BY A CALL TO VFETCH.  THE LOCATION IN
C                         EXTENDED MEMORY IS (I-1)*6+6 WHERE I=1,NSTA.
C
C     DIMENSION XX(6)
      DIMENSION LTAG(NSTA)
      EQUIVALENCE (XX(6),DATA)      ; NOTE: THIS IS A FORTRAN 5 ROUTINE
C
C        VALUE OF DATA RETRIEVED FROM EXTENDED MEMORY.
C
      DO 200 K=1,NSTA
      CALL VFETCH(XX,(K-1)*6+6,6)
      IF(DATA.GE.0..AND.DATA.LE.40.)GO TO 200
      LTAG(K)=1
 200  CONTINUE
      RETURN
      END
```

Example 9. A subroutine that shows the use of VFETCH.

```
      SUBROUTINE UCOMP(WRK1,UV,NXM,NYM)
C
C        MAY 1987   GLAHN, WOLF   TDL   ECLIPSE (MARD)
C
C        PURPOSE
C           TO COMPUTE ADVECTIVE U-WIND IN GRID UNITS PER HALF HOUR.
C           THIS SUBROUTINE REPLACES THE FOLLOWING CODE IN NON-
C           EXTENDED VERSION, AVWND:
C
C                 DO 110 JY=2,JYM1
C                 DO 100 IX=2,IXM1
C                 UV(IX,JY)=D1(IX,JY)*((P(IX,JY-1)-P(IX,JY+1))*CBDA-
C                1          (PG(IX,JY)-PG(IX,JY+1))*CADVM)
C           100   CONTINUE
C           110   CONTINUE
C
C           EXTENDED MEMORY IS SET UP AS FOLLOWS:
C
C              POSITION                  ARRAY
C                    1                   PG( , )
C                NXYM+1                  D1( , )
C               2*NXYM+1                  P( , )
C               3*NXYM+1                  U( , )
C
C        DATA SET USE
C           NONE.
C
C        VARIABLES
C         WRK1(IX,JY)  =  ARRAY USED IN EXTENDED MEMORY VERSION TO TAKE PLACE
C                         OF ARRAYS P( , ), PG( , ), AND D1( , ) FROM NON-
C                         EXTENDED AVWND.  (IX=1,NXM) (JY=1,NYM). (INTERNAL)
C           UV(IX,JY)  =  ADVECTIVE U-WINDS IN GRID UNITS PER HALF HOUR ON THE
C                         1/4 BEDIENT GRID. (IX=1,NXM) (JY=1,NYM) (OUTPUT)
C             NXM,NYM  =  DIMENSIONS OF WRK1( , ) AND UV( , ).  (INPUT)
C                CBDA  =  FRACTION OF 500-MB HEIGHTS USED TO ADVECT MAP
C                         FEATURES.
C               CADVM  =  WEIGHTING CONSTANT USED FOR MOUNTAIN INFLUENCE ON
C                         ADVECTIVE WINDS.
C
C        NONSYSTEM SUBROUTINES CALLED.
C            NONE.
C
C     DIMENSION WRK1(NXM,NYM),UV(NXM,NYM)
C
      NXYM=NXM*NYM
      NXM1=NXM-1
      NYM1=NYM-1
      CADVM=2.
      CBDA=.55
```

Example 10. A subroutine that shows the use of VFETCH. (Continued on the following page.)

```
C
      CALL VFETCH(WRK1,1,NXYM)
C
      DO 110 JY=2,NYM1
      DO 100 IX=2,NXM1
      UV(IX,JY)=(WRK1(IX,JY-1)-WRK1(IX,JY+1))*CADVM
 100  CONTINUE
 110  CONTINUE
C
      CALL VFETCH(WRK1,2*NXYM+1,NXYM)
C
      DO 210 JY=2,NYM1
      DO 200 IX=2,NXM1
      UV(IX,JY)=CBDA*(WRK1(IX,JY-1)-WRK1(IX,JY+1))-UV(IX,JY)
 200  CONTINUE
 210  CONTINUE
C
      CALL VFETCH(WRK1,NXYM+1,NXYM)
C
      DO 310 JY=2,NYM1
      DO 300 IX=2,NXM1
      UV(IX,JY)=UV(IX,JY)*WRK1(IX,JY)
 300  CONTINUE
 310  CONTINUE
C
      RETURN
      END
```

Example 10 (cont.)

```
      SUBROUTINE SUMX(WRK,H,NXLHB,NYLHB,T,J)
C
C       MAY     1987   GLAHN, WOLF   TDL   ECLIPSE (MARD)
C       OCTOBER 1988   GLAHN, WOLF   TDL   REVISED
C
C       PURPOSE
C          TO COMPUTE SUMS OF CROSS-PRODUCTS OF PREDICTORS AND PREDICTANDS.
C
C       DATA SET USE
C          NONE.
C
C       VARIABLES
C         WRK(IX,JY)  =  WORK ARRAY FOR SUMMING HEIGHTS. (IX=1,NXLHB)
C                        (JY=1,NYLHB).   (INTERNAL)
C           H(IX,JY)  =  500-MB HEIGHT FIELD FOR A PARTICULAR PROJECTION
C                        (IX=1,NXLHB) (JY=1,NYLHB).   (INPUT)
C        NXLHB,NYLHB  =  DIMENSIONS OF WRK( , ) AND H( , ).   (INPUT)
C              T(J)   =  HOLDS PROJECTIONS FOR TERMS: LINEAR, QUADRATIC,
C                        CUBIC, AND CONSTANT, FOR J=1,4 RESPECTIVELY.
C                        (INPUT)
C                 J   =  NUMBER OF THE TERM WHICH THE ROUTINE IS CALCULATING
C                        COEFFICIENT FOR:  LINEAR, QUADRATIC, CUBIC, AND
C                        CONSTANT FOR J=1,4 RESPECTIVELY. (INPUT)
C
C       NONSYSTEM SUBROUTINES CALLED.
C          NONE.
C
      DIMENSION WRK(NXLHB,NYLHB),H(NXLHB,NYLHB)
      DIMENSION T(4)
C
      NXYLHB=NXLHB*NYLHB
      CALL VFETCH(WRK,(J-1)*NXYLHB+1,NXYLHB)
C
      DO 200 JY=1,NYLHB
      DO 190 IX=1,NXLHB
      WRK(IX,JY)=WRK(IX,JY)+H(IX,JY)*T(J)
  190 CONTINUE
  200 CONTINUE
C
      CALL VSTASH(WRK,(J-1)*NXYLHB+1,NXYLHB)
      RETURN
      END
```

Example 11. A subroutine that shows the use of VSTASH and VFETCH.

```
      OVERLAY ORDSN
      SUBROUTINE RDSNR(KFIL10,KFIL12,KFIL4,KFIL5,KFIL7,JFILE,JDATE,JTIME,NXP,
     1                 NYP,XMESHL,ORIENT,X,ND1,NSTA,ID,JD,ND9,IER)
C
C        JUNE 1987   GLAHN, CHAMBERS   TDL   ECLIPSE (MARD)
C
C        PURPOSE
C           TO READ CALL LETTERS, LATITUDE, AND LONGITUDE FROM JFILE AND
C           COMPUTE GRID LOCATIONS.  THESE ARE STORED IN EXTENDED MEMORY.
C
C        DATA SET USE
C           KFIL4    =  UNIT NUMBER FOR TDL ELEMENT KEY FILES.  (INPUT)
C           KFIL5    =  UNIT NUMBER FOR TDL ELEMENT DATA FILES.  (INPUT)
C           KFIL7    =  UNIT NUMBER FOR READING ID SET FOR ACCESSING TDL
C                       DATABASE FILES FROM FILE 'IDELM.DA'.   (INPUT)
C           KFIL10   =  UNIT NUMBER FOR CURRENT CONSOLE.   (OUTPUT)
C           KFIL12   =  UNIT NUMBER FOR OUTPUT (PRINT) FILE.   (OUTPUT)
C
C        VARIABLES
C          INPUT
C               KFIL4   =  UNIT NUMBER FOR TDL ELEMENT KEY FILES.
C               KFIL5   =  UNIT NUMBER FOR TDL ELEMENT DATA FILES.
C               KFIL7   =  UNIT NUMBER FOR READING ID SET FOR ACCESSING TDL
C                          DATABASE FILES FROM FILE 'IDELM.DA'.
C             KFIL10    =  UNIT NUMBER FOR CURRENT CONSOLE.
C             KFIL12    =  UNIT NUMBER FOR OUTPUT (PRINT) FILE.
C            JFILE(J)   =  4 CHARACTERS (J=1,2) DENOTING THE FIRST 4 CHARACTERS
C                          OF THE TDL ELEMENT FILES.  IF JFILE(1) = 9999, THE
C                          DEFAULT NAME IS USED.
C             JDATE(J)   =  DATE, CONSISTING OF YEAR (4 DIGITS), MONTH, AND DAY
C                          (J=1,3), OF DATA TO ACCESS.
C               JTIME    =  HOUR OF DATA TO ACCESS.
C              NXP,NYP   =  NORTH POLE POSITION WITH RESPECT TO LOWER LEFT
C                          CORNER (1,1).
C             XMESHL     =  GRID SPACING IN KM.
C             ORIENT     =  LONGITUDE PARALLEL TO GRID COLUMNS.
C                ND1     =  SIZE OF SEVERAL ARRAYS.
C                ND9     =  DIMENSION OF ID( ) AND JD( ).
C
C
C          OUTPUT
C                NSTA     =  NUMBER OF STATIONS FOR WHICH DATA ARE AVAILABLE.
C                            NSTA*5 4-BYTE (2 RDOS WORDS) LOCATIONS ARE FILLED IN
C                            EXTENDED MEMORY.  THE FIRST 5 LOCATIONS ARE USED FOR
C                            THE CALL LETTERS, LATITUDE, LONGITUDE, X-GRID LOCA-
C                            TION AND Y-GRID LOCATION OF THE FIRST STATION (IN
C                            THAT ORDER).  LOCATION 6 IS RESERVED FOR THE OBSER-
C                            VATIONS FOR THAT STATION.  THE SAME VARIABLES FOR
C                            THE REST OF THE STATIONS FOLLOW.
```

Example 12. A subroutine that shows the use of VSTASH and VFETCH. (Continued on the following page.)

```
C             IER  =  RETURN FROM DATABASE ROUTINE RDTDL.  A RETURN OF 1
C                     MEANS ALL 3 RECORDS WERE READ CORRECTLY.  ANY OTHER
C                     VALUE IS RETURNED FROM ONE OF THE READS.  IER IS
C                     ALSO USED INTERNALLY FOR SYSTEM ERROR RETURNS.
C
C          INTERNAL
C                XP1  =  STATION LATITUDE (SEE NSTA).
C                YP1  =  STATION LONGITUDE (SEE NSTA).
C                 XP  =  STATION X-GRID LOCATION (SEE NSTA).
C                 YP  =  STATION Y-GRID LOCATION (SEE NSTA).
C              ID(J)  =  IDENTIFICATION INFORMATION FOR READING ELEMENT FILE
C                     PREPARED BY RDIDE (J=1,ND9).
C              JD(J)  =  CONTAINS INFORMATION FROM KEY RECORD RETURNED AFTER
C                     READING ELEMENT FILE (J=1,ND9).
C               ICH2  =  DUMMY CHANNEL NUMBER NECESSARY IN THE CALL TO OVLOD.
C                     NOTE:  THIS SUBROUTINE IS WRITTEN IN FORTRAN 5.
C              XX(J)  =  WORK ARRAY TO ASSIST WITH EXTENDED MEMORY ACCESS
C                     (J=1,6).
C               X(J)  =  WORK ARRAY, DIMENSION OF ND1.
C
C        NONSYSTEM SUBROUTINES CALLED.
C            RDIDE, IERX, RDTDL, IERCK, ELLIJ
C            OVERLAY FILE IS CALLED FOR RDTDL BUT NOT FOR OTHER ROUTINES.
C
      DIMENSION X(ND1)
      DIMENSION ID(ND9),JD(ND9)
      DIMENSION JFILE(2),JDATE(3),XX(6)
      EQUIVALENCE (XX(2),XP1),(XX(3),YP1),(XX(4),XP),(XX(5),YP)
      EXTERNAL ORDTD
C
C        READ CALL LETTERS OF STATIONS IN ELEMENT FILE AND PUT
C        INTO EXTENDED MEMORY.
C
      CALL RDIDE(KFIL10,KFIL12,KFIL7,28,200,JDATE,JTIME,0,ID,ND9)
      CALL OVLOD(ICH2,ORDTD,0,IER)
      IF(IER.NE.1)CALL IERX(KFIL10,KFIL12,'RDSTN ','100 ')
      CALL RDTDL(KFIL10,KFIL12,KFIL4,KFIL5,JFILE,JDATE,JTIME,ID,JD,
     1           ND9,0,X,ND1,2,IER)
      IF(IER.NE.1)GO TO 200
      NSTA=JD(2)
C
C        NSTA EQUALS NUMBER OF ITEMS OF DATA IN THE CALL LETTERS
C        RECORD OF THE ELEMENT FILE.
C
      DO 100 K=1,NSTA
      CALL VSTASH(X(K),(K-1)*6+1,1)
  100 CONTINUE
C
C        READ LATITUDES AND PUT INTO EXTENDED MEMORY.
```

Example 12 (cont.)

```
C
      CALL RDIDE(KFIL10,KFIL12,KFIL7,29,200,JDATE,JTIME,0,ID,ND9)
      CALL RDTDL(KFIL10,KFIL12,KFIL4,KFIL5,JFILE,JTIME,JDATE,ID,JD,
     1           ND9,0,X,ND1,2,IER)
      IF(IER.NE.1)GO TO 200
C
      DO 125 K=1,NSTA
      CALL VSTASH(X(K),(K-1)*6+2,1)
 125  CONTINUE
C
C        READ LONGITUDES AND PUT INTO EXTENDED MEMORY.
C
      CALL RDIDE(KFIL10,KFIL12,KFIL7,30,200,JDATE,JTIME,0,ID,ND9)
      CALL RDTDL(KFIL10,KFIL12,KFIL4,KFIL5,JFILE,JDATE,JTIME,ID,JD,
     1           ND9,0,X,ND1,2,IER)
      IF(IER.NE.1)GO TO 200
C
      DO 150 K=1,NSTA
      CALL VSTASH(X(K),(K-1)*6+3,1)
 150  CONTINUE
C
C        COMPUTE GRID COORDINATES IN XP AND YP UNLESS STATION
C        POSITION IS MISSING.  ASSUME XP1 = 9999. FOR MISSING
C        POSITION.
C
      DO 175 K=1,NSTA
      CALL VFETCH(XX(2),(K-1)*6+2,2)
      IF(XP1.NE.9999.)GO TO 165
C        POSITION MISSING.
      XP=9999.
      YP=9999.
      GO TO 170
C
 165  CALL ELLIJ(XP1,YP1,XMESHL,ORIENT,FLOAT(NXP),FLOAT(NYP),XP,YP)
 170  CALL VSTASH(XX(4),(K-1)*6+4,2)
 175  CONTINUE
C
      RETURN
      END
```

Example 12 (cont.)