THE mini•tasker

DECUS
RT-11 SIG NEWSLETTER

APRIL 1982 VOL. 8-2

Contributions to the newsletter should be sent to:

Ken Demers
MS-44
United Technologies Research Center
East Hartford, Conn. 06108
(203) 727-7527 or 7240

Other communications can be sent to:

John T. Rasted
JTR Associates
58 Rasted Lane
Meriden, Conn. 06450
(203) 634-1632

or

RT-11 SIG
C/O DECUS
One Iron Way
MR2-3/E55
Marlboro, Mass. 01752
(617) 467-4141

TABLE OF CONTENTS

USER INPUT

ON THE USE OF VIRTUAL(EXTENDED) MEMORY

Using FORTRAN VIRTUAL in applications running before without it, slows down these applications considerably. This is illustrated by the table below which shows that at least FORTRAN VIRTUAL is 3* slower than standard FORTRAN.

TABLE

!10 times transfer(Program DO-loop,READ+WRITE for device) of 1024 words !

	PDP-11/34	PDP-11/03	LSI-11/23	
using FORTRAN low-mem array:	37.			(Threaded)
" " " :	14.	29.	17.	(In-line)
" " VIRTUAL " :	103.			(Threaded)
" " VIRTUAL " :	89.			(In-line)
" MACRO low-mem array:	3.01	6.66	3.86	
on VM: .READW/.WRITW :	7.			
on RK05 " :	81.			
on RX01 " :	420.			

Note: All values are nr. clock ticks(1 clocktick=20. ms. !).

The graph included shows that the VM: performs better than VIRTUAL for recordsizes larger then 100. words, while for records larger than 256. words it is more then 10 times faster! The vertically displayed values show the time in ms. it takes, to read from+write to memory, the nr. of points displayed horizontally.

I include the source of a routine to manipulate in a more a less transparant way data residing on VM: as data in an array.
More arrays may be used within a program by using different filenames.
In the same way more jobs may use virtual memory simultaneously.

H. T. M. Haenen
Dept. Clin. Neurology AZG
P.O. Box 30.001
9700 RB GRONINGEN
Holland

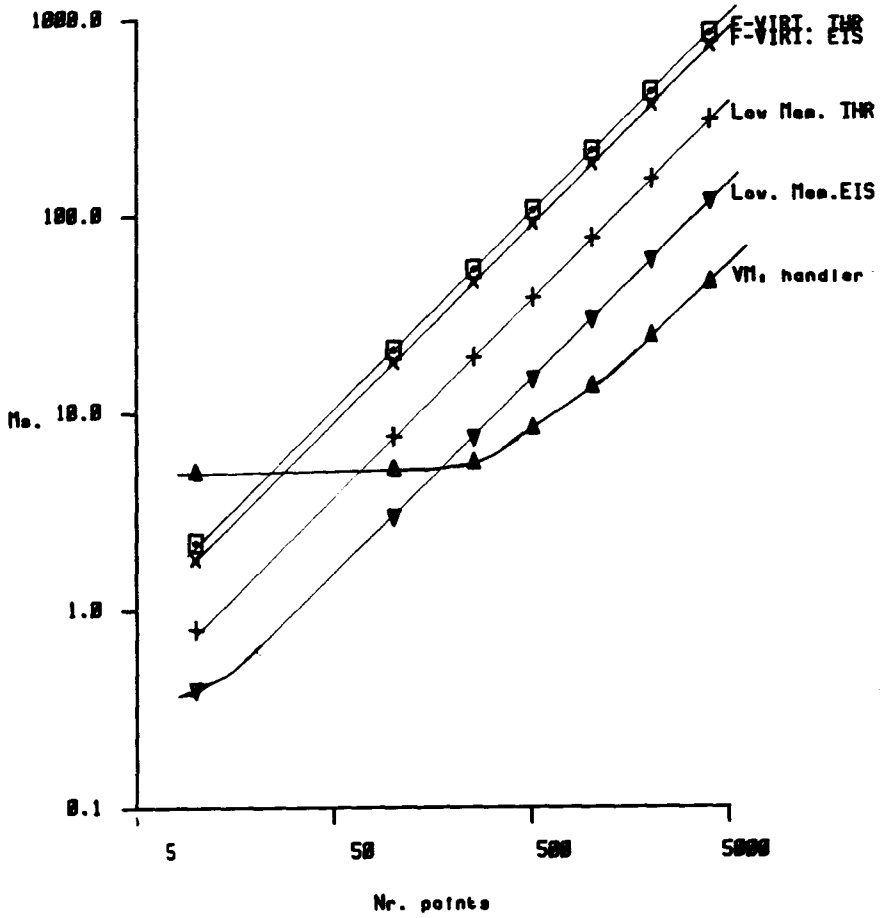
```
.TITLE  VMARR

; H.H. Klin. Neuro. dec-81

; Motivation:
; The VM: handler is more then 10* faster in data- transports then
; using FORTRAN VIRTUAL. Even faster then FORTRAN itself.

; The routines:

; CALL INIARR(NRELEMENTS,ITYPE)
```

```
;       Initialize an array with the dimension of NRELEMENTS
;       (=NRELEMENTS/256.+1 blocks for ITYPE=1).
;       In fact a file is opened on VM: with name VARRAY.TMP
;       ITYPE = 1   : Integer array
;               2   : Real
;               3   : Double precision

; CALL GETARR(BUFFER,INDEX,NRELEMENTS,INDBUF)

;       Read from virtual into low-memory buffer BUFFER
;       a number of NRELEMENTS starting at index INDEX
;       BUFFER must be large enough to contain the red elements
;       and minimum size is 256. words.
;       INDBUF points to the same element in BUFFER as INDEX
;       does for the whole virtual array.

; CALL PUTARR(BUFFER,INDEX,NRELEMENTS,INDBUF)

;       Write to virtual from low memory. Arguments as for GETARR.

; CALL FINARR(ICLOSE)

;       Closes virtual array buffer. If ICLOSE=-1 then array is purged.
;       Else the array is conserved as a file on VM:.

;-------------------------------------------------------------------------

;
;       0       1       2       3       etc.  <-- block adres on VM:
;      !-------!-------!-------!----
;      !----------------->!                     INDEX
;                      !->!                     Extra words transferred due
;                                               to block bounderies.
;                         !------>!             NRELEMENTS
;                      !--------->!             INDBUF

        .MCALL .EXIT,.PRINT,.ENTER,.LOOKUP,.READW,.WRITW,.CLOSE
        .MCALL .PURGE
        .ENABL LC

INIARR::
        TST     (R5)+
        MOV     @(R5)+,R0       ;Nr. elements
        MOV     @(R5)+,R1       ;ITYPE 1,2,3
        DEC     R1
        ASH     R1,R0           ;Nr. words 2=2*, 3=4*
        MOV     R1,TYPE         ;Save type.
        MOV     R0,R2
        CLRB    R0
        MOV     R0,R5
        SWAB    R5              ;Nr. blocks
        SUB     R0,R2           ;Remaining words?
        BEQ     2$
        INC     R5              ;Yes, then 1 block more
;
;  Open channel on VM:
;  Use channel #14. FORTRAN always starts with #0 for its LUN's.
;
```



```
2$:
        .ENTER  #AREA,#14.,#FILO,R5
        BCS     LFAILO
        RETURN
;
; Error returns:
;
LFAILO: .PRINT  #NOLKO
        .EXIT
;-----------------------------------
FILO:   .RAD50  /VM VARRAYTMP/
;-----------------------------------
NOLKO:  .ASCIZ  /VMARR LOOKUP-F/
ERR:    .ASCIZ  /VMARR HARD IO ERR/
FLAG:   .BYTE   0
        .EVEN

;  End of initializing code
;  ------------------------
GETARR::
        CLRB    FLAG
        BR      START
PUTARR::
        MOVB    #1,FLAG
START:  TST     (R5)+
        MOV     (R5)+,R1                ;Buffer adres
        MOV     @(R5)+,R4               ;Index
        DEC     R4                      ;#0 START
        MOV     TYPE,R0
        ASH     R0,R4                   ;Make nr. words
        MOV     R4,R3
        CLRB    R3                      ;Multiple of 256.
        SUB     R3,R4                   ;Remaining words
        SWAB    R3                      ;Block adress
        MOV     @(R5)+,R2               ;Nr. elements to do
        ASH     R0,R2                   ;"   words    "  "
        ADD     R4,R2                   ;Extra words due to blk bound
        NEG     R0
        ASH     R0,R4                   ;Make nr. elemnts
        INC     R4                      ;Index start at #1
        MOV     R4,@(R5)+               ;And store it.
        TSTB    FLAG
        BNE     1$
        .READW  #AREA,#14.,R1,R2,R3
        BCS     HRDERR
        RETURN

1$:     .WRITW  #AREA,#14.,R1,R2,R3
        BCS     HRDERR
        RETURN
HRDERR: .PRINT  #ERR
        .EXIT
;  Close/Purge channel

FINARR::
        TST     (R5)+
        CMP     #-1,@(R5)               ;Purge?
        BEQ     1$
        .CLOSE  #14.
        RETURN
```

```
I$:     .PURGE  #14.
        RETURN

AREA:   .BLKW   5
TYPE:   .WORD   0

        .END
```



How I installed a lineprinter

Who hasn't installed a serial device at least once? I must have done it a dozen times by now. But, despite the fact that serial devices are the easiest there are, Murphy always seems to attend their installation. This story tells of a typical *dumb* installation, and includes a *dumb*, but effective, solution to the problem of connecting a device with modem controls to an interface that doesn't support them (DLV11J).

It all started when the high-quality qume printer blew-up and had to be sent to Düsseldorf for extended repairs - leaving us without a printer. I decided it was time to get a cheap back-up printer - and so I purchased the cheapest I could find - an epson mx80.

The printer arrived Saturday in a box with lots of little bits and pieces and not much documentation. I should confess at this point that I don't know much about hardware - just enough to argue with hardware engineers when it comes to working out whether a problem is software or hardware. So, the first thing I did was ring Mike, who knows hardware. He wasn't home. This didn't suprise me - he always seems to go to Berlin when I need him for hardware.

I unpacked the printer and read the documentation. At this point I worked out which connectors and cables I needed. This took about an hour. Then, I copied the connectors I had made for the qume. Unfortunately, I didn't have a berg connector gun and so I ended up doing them by hand. This took another hour.

At this point Mike turned up unannounced and unaware. He wanted to use my garage to play around with his new toy - i.e. install a radio/cassette in his brand new sports car. Thus, he wasn't much interested in playing around with my El Cheapo printer but I *managed* to blackmail him into at least doing the necessary soldering (which I abhore).

He had to solder three wires to the printer end. Well, he did this but nothing seemed to function. So he looked at what I had done at the DLV-11J-end and looked at me like I was a madman. The quality of my connections apparently left something to be desired. I explained that I didn't have a berg gun (which I can't operate anyway) and we decided to connect the wires directly to the DLV-11J without using a socket.

It still didn't work, and I suggested to Mike's blank face that perhaps we should check the pin numbers at the printer-end (by this time we had assembled quite a range of documentation including 2 LSI handbooks, the DLV-11J drawings and the epson manuals). Mike checked them and then, told me sheepishly that not once in the 50 serial devices he had connected had he ever got the pin numbers right first time.

He resoldered the connection and the printer started to at least print things. It would print B's but not A's. 'Looks like software' said Mike as dissappeared in the direction of his auto. I was still considering the validity of his seemingly dubious statement, when he returned and proceeded to unplug and remove the standard lamp near the printer. When I objected he reminded me that he had lent it to me some two years before. He left me in the dark.

As I suspected the printer was switched to expect parity - so I wrote my first parity routine (easier than dismantling the printer to change the switch). Well, of course this took two tries - since I provided even-parity and it wanted it odd. But, finally I could print the entire character set - which is when I found out the distributors had shipped it to me set up for german umlauts instead of ascii. But that was just another switch somewhere in the middle of the machine. At the same time a much more critical problem reared its ugly head.

When I was looking for this low-cost printer I scanned a dozen product descriptions. I must have merged two descriptions and I had the impression that the epson would support ctrl-s/ctrl-q for print speed synchronisation. I was wrong. The epson was using the data-terminal-ready signal to sync. As most of you will know a DLV-11J does not support this signal. Hmm.

Well, I had three options. First I could send the printer back and it hadn't exactly endeared itself to me. Second, I could buy a DLV-11E, but I would be better off investing such money in a printer that supported ctrl-s/ctrl-q. Third, I could try to make the damn thing work.

I went out to the garage and spoke to Mike who was stuck upside down in the back seat of his car cursing himself for not paying the extra DM 200 ($75) for the installation of the casette/radio. I asked him whether I could somehow use the data-terminal-ready signal (DTR) to drive the transmitter-done line. He told me that DTR was a 12-volt level, not a 5-volt pulse and I would blow the board up. So I asked him about the other signals and went off to think.

Well, I had a lot of time to think because Nicole turned up to do some photocopying, and the photocopier broke down three times in a row. The third time was more critical since this little rubber band that drove the drum came off. I am scared stiff of the photocopier since I always manage to burn my fingers in there, so after trying to fix it for a half an hour I told Nicole to get Karin who had more experience. Karin fixed it in about one minute. I asked her how often this had happened before and was deflated to learn that it was also the first time she had seen the problem.

The next delay was caused by the German ritual of having Coffee and Cake. Karin, Nicole, Mike, Karin's parents, our two baby boys and I tried to celebrate the ritual around a table that was designed for maximum four persons. After that Karin forced Mike to take us in his new car to see two new houses (we need a bigger place for a bigger table).

Well, I decided in my own naive software-person-like way that all I needed to do was get the DTR into my machine somehow. Now, I found that the data-in signal for the receiver is also an EIA signal. I asked Mike what would happen if I fed the DTR signal into the data-in line and he said that nothing would probably happen but that I wouldn't blow anything up. He soldered the connection for me.

I had theorised that what I should get is a framing error and thats exactly what happened. I wasn't sure if I would get one or two framing errors but it turned out to be a single framing error each time the epson raised the DTR signal. It works. Whats more I can get an interrupt off it. In fact its easier to use than a DL-11E.

Now I don't know if this technique will work with all such connections. There are in fact two DTR signals - the second, Reverse Channel, is the complement of DTR. In fact I think used Reverse Channel by mistake. Anyway, if you want to use this techinque just try both of them.

Well, the rest of the work was just plodding to bump the baud rate up, disable the German umlauts and get the paper size right (which took about an hour since I had to work out how to divide by three). The epson is real slow (I mean really slow) on line feeds, so I have set it up to use small characters all the time and use 2/3 size paper (A5), this makes it 33% faster. It also means that the listings will take up less room on the work tables that are also too small.

What has this to do with RT-11. Well, as you will have assumed, I put together a device handler for the printer as I went along. With each discovery I changed the handler to reflect my improved view of reality. Compile and load took under a minute. I used the handler to help solve most of the problems. In most other systems I know, the handler itself would be the largest problem. And they ask me why I love RT-11?

Ian Hammond - Am Feldborn 22 - D-34 Göttingen - Germany - Tel: +49 (551) 23828

USER REQUESTS

PCKTECHNOLOGYDIVISION

322 L.I.E. South Service Road, Melville, N.Y. 11747 • Tel. 516-454-4400 • TWX 510-224-6596

Dear Mr. Demers:

I am an RT-11 user with a need for a "visi-calc"-like tabulating program to run under RT-11. To this date I have had no luck finding anyone who has written such a program for use on DEC computers.

Any assistance would be a great help. There is a real need for such a program within our Division.

Your assistance in this matter will be sincerely appreciated.

Very truly yours,

Jon Crowell

Jonathan C. Crowell
DECUS #: 158532

UPCOMING SYMPOSIUM INFORMATION

TECO TUTORIAL I - INTRODUCTORY EDITING

TECO-TUTORIAL I is a session designed to introduce TECO to new and beginning users. TECO is the only editor available across most Digital Equipment Corporation operating systems: RSTS/E, RSX-11/M, RSX-11/D, VAX/VMS, IAS, RT-11, OS/8, TOPS-10 and TOPS-20. A "novice subset" of TECO commands will be presented in an operating system independent manner. Examples will be carefully explained for this set of commands.

Users who have hesitated in introducing TECO on their site are invited to attend this presentation to see an effective means of teaching TECO. The presentation method includes two speakers (TECO in stereo, no less), a session handout and an introductory level publication, "How to Use TECO."

It is emphasized that this session is <u>not</u> paced for the experienced TECO user.

Presented by: Steven Stepanek, California State University, Northridge
J. A. Hayes, California State University, Northridge
Boyce Cowgill, Dynamic Sciences, Inc.

VTEDIT TUTORIAL

This tutorial presents VTEDIT, the full screen, key pad text editor available across most DEC operating systems. The topics covered are VTEDIT conventions, access and exit, editing commands (move cursor, insect/move text, delete text, search text), learn mode and interpreting the on-line screen "instant".

Presented by: J. A. Hayes, California State University, Northridge
Boyce Cowgill, Dynamic Sciences, Inc.

PRE-SYMPOSIUM SEMINAR

1982 SPRING DECUS ATLANTA, GEORGIA

SUNDAY MAY 9 FROM 9:00 am to 5:00 pm

TECO-FROM INTERMEDIATE EDITING TO PROGRAMMING FEATURES

This tutorial covers intermediate to advanced editing commands for TECO, Text Editor and Corrector, for the DIGITAL operating systems: RSTS/E, RSX-11M, RSX-11D, IAS, RT-11, OS/8, TOPS-10, TOPS-20, and VAX/VMS. Additionally, it will include the programming features that make TECO one of the most powerful editors available across operating systems.

The morning session will include:

- Fundamental TECO concepts
- Helpful tidbits and other essentials
- TECO accesses
- Editing commands

The afternoon session will cover:

- Overview of advanced editing and programming concepts
- Input/Output capability
- Data structures/storage
- Ability to loop
- Conditional execution
- Ability to perform operations on data structures
- Q-Registers - Numeric and text storage
- Extended search and match constructs
- Advanced TECO examples

Who should attend:

- Users and programmers who wish a comprehensive, well-structured tutorial on TECO
- Users and programmers who work on multiple operating systems and who prefer to use only one editor
- Individuals wishing to learn how to present TECO in effective manner at their own sites

Instructors: Joyce Hayes and Steven Stepanek
California State University, Northridge, CA

Migration/Compatibility Between DEC Operating Systems

"'Compatibility' is a big word in any language - and it means different things to different people. The manufacturer looks at Compatibility in the large, as a cost/performance problem. For the user, Compatibility is a micro-management task: "Can I connect device X to system Y on processor Z?" This difference of approach is the primary reason for this report which presents the User View of Compatibility".

So reads an introductory paragraph of a draft report issued last year by the European DECUS Migration Working Group. The scope of this report (presently 40 pages in length) includes:

- o HARDWARE - bus, processor, instruction set, devices, memory map
- o OPERATING SYSTEMS - evolution, processors, configurations
- o FILE structures, names, capabilities
- o TERMINAL commands, control keys, command files, DCL commands, running programs
- o PROGRAM commands, editors, utilities
- o LANGUAGES
- o OPERATING SYSTEM structure, architecure, emulators, terminology, calls
- o FILE STRUCTURE LAYOUTS

Obviously, the question of migration and compatibility is an issue for any organization, regardless of size. A Birds of a Feather session was held at the Fall '81 symposium to discuss this issue, and to see what interest there was in expanding on the European report. As it turns out, a great deal of interest exists, and we have now scheduled a workshop for Atlanta, Wednesday, 12th May at 3:30 p.m.-5:30 p.m. in Salon B Room of the Atlanta Hilton and Towers Hotel. Steve Hargrave, Software Support Specialist for the TOEM Marketing Group, has expressed enthusiasm for this project, particulary as he is presently involved in a similar study for RSX to VAX, and Steve has indicated that he will participate in our session.

Although this workshop landed in the RT-11 schedule, the subject impacts all SIGS, and probably the Site Management SIG more than any other. This workshop will be an interactive session that will allow users to freely discuss the various problems and solutions that accompany the migration from one operating system to another. The object of this workshop is to get together a working group to generate a document that will help to demystify these issues. This kind of document is sorely needed and does not yet formally exist; however, with the foundation we've been provided, and with the talent that exists within the DECUS membership, we can produce a document that will make migration efforts easier for everyone.

Any person who has found any solutions to any problem relating to this subject is encouraged to present this information in Atlanta; however, we would like to collect as much data prior to Atlanta, if at all possible, that will be incorporated into a second draft, to be issued subsequent to the workshop.

A copy of the European Migration Report can be obtained by contacting me at the address below. Feel free to note any comments or suggestions separately or directly onto the report.

Shirley M. Hooper
Bausch & Lomb
Instrument & Systems Division
9545 Wentworth Street
Sunland, California 91040
(213) 352-6011 ext. 291

PAST SYMPOSIUM INFORMATION

RT-11 SJ/FB/XM Performance Report Revisited

Ned W. Rhodes
Melpar Division
E-Systems Inc.
7700 Arlington Boulevard
Falls Church, Va. 22046

ABSTRACT

Test routines were developed and timing tests were run to see which of the RT-11 monitors (Single Job (SJ), Foreground/Background (FB), or Extended Memory (XM)) would provide the fastest execution of user programs. The test routines were also run under TSX and along with the VM (Virtual Memory) handler under the FB environment. The test routines were broken into three groups. Within each group, the test routine was linked so as to use no overlays, disk resident overlays, and virtual overlays. Two groups used virtual arrays for data storage and one did not. Two types of virtual array support routines were used. One set stored the array in extended memory while the other set used disk storage. The results show that the FB monitor and the VM handler provide the fastest environment and should be used with large, heavily overlaid programs. The results also show that to optimize the use of virtual overlays, programs should minimize the number of times virtual overlay segments are called.

1.0 Old Business

As the title implies, this is a second look at RT-11 performance characteristics. The first look revealed that the virtual overlay handler had a serious design flaw that resulted in poor performance of virtually overlayed jobs. Since that time, a patch has been applied to the virtual overlay handler and one of the purposes of this paper is to determine just how effective that patch is.

The other result of the first paper was in the form of a series of questions concerning the VM (Virtual Memory) handler and virtual arrays. The VM handler had the best performance characteristic of any of the test configurations, and it was postulated that if virtual arrays could made disk resident and placed on the VM device, that that configuration would be faster than the equivalent XM configuration. This paper will examine a support package that makes virtual arrays disk resident, and it will show the performance of that package.

2.0 Introduction

Since its introduction in 1973, RT-11 has grown from a Single Job (SJ) monitor to a Foreground/Background (FB) monitor and finally to an Extended Memory monitor (XM). With three RT-11 monitors available, it is sometimes hard to choose the monitor that is correct for your application. The purpose of this paper is to present some performance characteristics of the different RT-11 monitors that will make the choice easier.

My discussion of RT-11 performance characteristics will center around four areas. I first want to establish which RT-11 operating environment allows the fastest execution of user programs. Next I want to explore extended memory overlays and compare them with disk resident overlays. Additionally, I will look at the performance of the patched virtual overlay handler. Thirdly, I want to investigate the VM or Virtual Memory handler that was distributed on the Fall 1979 RT swap tape. And finally, I want to study how much time is required to implement virtual arrays in the various RT-11 operating environments using the standard extended memory support package as well as a support package for disk based virtual arrays.

I have divided this paper into a number of sections. I will first discuss the RT-11 operating system in general. Next I will discuss extended memory management techniques. Those two sections will provide the necessary background information in order to understand the test routine which I created to measure RT-11

performance. This test routine exists in two versions. The first version uses regular arrays for data storage while the second version uses virtual arrays. I will be comparing the time differences between the two versions of the test routine. I have also included TSX-PLUS, a multi-user version of RT-11, in some of my timing comparisons for reference. Finally, I will answer the following questions:

1. Which RT monitor provides the fastest program execution environment?

2. How much faster than disk resident overlays are extended memory overlays and how can I optimize their usage?

3. How much slower do programs execute under TSX-Plus than under the XM monitor?

4. How much additional time is added to program execution when virtual arrays are used and does the XM environment provide a faster implementation of virtual arrays than the SJ or FB environments?

5. Will the use of the VM (Virtual Memory) handler decrease program execution times? Why or why not?

6. What is the performance of disk based virtual arrays?

7. Does the patched virtual overlay handler provide any additional performance over the unpatched version?

My particular use of RT-11 is in a high-speed, data acquisition environment where speed is the most important parameter. The rest of this paper will address only that parameter and give recommendations as to which configuration of RT-11 to use to allow the fastest execution of programs.

3.0 RT-11 Operating System

RT-11 was introduced in 1973 and has always been designed to be a single-user operating system that has many applications in the real-time environment where operating system characteristics like small size, efficiency, reliablity, high throughput, low interrupt latency times and ease of use are important. We have seen RT grow from a Single Job (SJ) monitor to a Foreground/Background (FB) monitor and finally to an Extended Memory (XM) monitor. Each of the different RT-11 monitors has different characteristics and capabilities; let me briefly describe them.

3.1 RT-11 Single Job (SJ) monitor

The Single Job monitor is the smallest of the three monitors, requiring only 2K words for the resident portion. It supports only one job and programs can access up to 28K words of memory. Data may be stored in memory above 28K by using virtual arrays in FORTRAN. The SJ monitor supports all the system utility programs and most of the keyboard commands and programmed requests.

3.2 RT-11 Foreground/Background (FB) monito

This monitor supports two jobs, one in the foreground, and the other in the background. The foreground job is given priority over the background job and always executes until a blocking condition exists, such as an I/O transfer, a timed wait, or a wait for an external interrupt. Only when the foreground job becomes blocked can the background job run. The foreground was designed to accommodate a time-critical task such as real-time data acquisition. All the system utilities and language processors run as background tasks. All tasks that will run in the SJ monitor will run in the background of the FB environment, provided there is enough memory to accommodate both the foreground and the background task, if both are loaded simultaneously. The FB monitor is only slightly larger than the SJ monitor and, like the SJ monitor, programs may address only 28K words of memory and data arrays may be placed in extended memory. A special system generation option allows up to six 'system jobs' (a special type of foreground job) to execute along with the foreground and the background jobs, provided they do not access the I/O page. This monitor offers the most services for the least amount of memory.

3.3 RT-11 Extended Memory (XM) monitor

The XM monitor, not introduced until Version 3, is the largest of the RT-11 monitors and provides the most services. The XM monitor has all the features of the FB monitor plus it provides a set of programmed requests that allow jobs to extend their logical address space beyond the 32K word limit imposed by the 16-bit word length of the PDP-11. The XM monitor requires a system with an Extended Instruction Set (EIS), a KT-11 memory management unit (or MMU chip on 11/23's) and greater than 32K words of addressable memory. Extended memory may be utilized

for large arrays of data, in FORTRAN, by declaring the array virtual. Or, programmed requests may be issued to map logical program address space to portions of extended memory. The linker has been modified so that portions of the program may be overlaid in extended memory, instead of overlaying from disk. For many applications, this can significantly reduce the amount of disk I/O time and decrease program execution times. The XM monitor is the only monitor that supports programmed requests for the use of extended memory.

3.4 TSX-Plus Operating System

As I have mentioned, RT-11 is distributed as three separate monitors and was designed to be a single-user operating system. S & H Computing has an RT-11 like product, called TSX-PLUS, that is worth mentioning. Essentially it is a multi-user RT-11 system with extensions. All jobs that run under the SJ monitor and do not access the I/O page, will run under TSX, as will all the language processors and RT-11 system utilities. I have included TSX-Plus in this paper because it provides a good environment for program development and data reduction, for multiple users, and it is almost fully compatible with existing RT-11 programs. TSX-Plus does not support extended memory based virtual arrays or extended memory overlays, but it will support the disk based virtual array package.

4.0 Extended Memory Management Techniques

The PDP-11 word length of 16-bits will allow a program to directly address only 32K words of memory, although the UNIBUS can address up to 128K words because of its eighteen address lines. The directly addressable memory in the range of 0-32K is usually termed low memory, while memory in the range of 32-128K is considered extended memory. The PDP-11's normally reserve the upper 4K words of the address space as the I/O page. Therefore, programs only really have 28K of user space available for each job. This 28K limit can never be extended, but the KT-11 memory management unit provides the means for programs to address any of the 128K words of physical memory, by mapping the 16-bit program virtual address to anywhere in the 128K word physical address space.

The KT-11 is composed of two sets of Active Page Registers (APR), one for the Kernel or supervisor mode and one for the user. Bits in the Program Status word (PSW) of the CPU determine which mode you are in. Each APR is composed of two registers called Page Address Registers (PAR) and Page Descriptor Registers (PDR). This pair of registers determine how the program's virtual address space maps into the PDP-11 physical memory. Table 1 shows the relationship between virtual memory addresses and PAR/PDR registers. For example, the virtual memory address range of PAR/PDR-7 normally is mapped to the PDP I/O page, while PAR/PDR sets 0-6 are used for user programs. Each PAR/PDR set controls up to 4K words of memory. The disadvantage of using the KT-11 is that 0.12 microseconds (on an 11/34) is added to execution times for every memory cycle used. Each PDP-11 instruction may require multiple memory accesses, depending on the instruction type, in order to execute so that the time will add up.

The KT-11 may be utilized in a number of ways. One way is to use programmed requests and let the XM monitor set up the KT-11 registers for you. Another way to use the KT-11 is to manipulate the hardware registers yourself. This would be very complicated and dangerous in a true multi-user environment, but in RT-11, it is a reasonable method of operation in certain circumstances.

Virtual Address Range	PAR/PDR
000000 - 017776	0
020000 - 037776	1
040000 - 057776	2
060000 - 077776	3
100000 - 117776	4
120000 - 137776	5
140000 - 157776	6
160000 - 177776	7

TABLE 1

4.1 FORTRAN Virtual Array Support

DEC FORTRAN has been extended to allow for the storage of data arrays in extended memory. These arrays are called virtual arrays and space is allocated for them in extended memory by the FORTRAN Object Time System (OTS) when it initializes for program execution. Due to the differences in the RT-11 monitors, there are two sets of virtual array support routines. One set of routines supports the SJ and FB environments and the other set supports the XM environment.

The virtual array support routines for the SJ and FB monitors manipulate the KT-11 hardware registers directly, which implies that other users cannot be manipulating the registers at the

same time. In order to support virtual arrays, the FORTRAN OTS maps the job and RT-11 itself to Kernel space and the virtual arrays to user space. When a virtual array element is used, the OTS turns on the KT-11 memory management unit, selects user space, accesses the element and then turns off the KT-11. This means that for most of the program, the memory management unit is off and so the added delay associated with having the unit turned on is minimized.

The FORTRAN virtual array support routines for the XM environment use a different approach than the SJ and FB routines. Instead of manipulating the KT-11 hardware registers directly, these routines use the XM programmed requests. When the program is initializing, the OTS 'buys' (from the monitor) a block of memory to hold the virtual arrays. Then, the OTS uses PAR/PDR set 7 for its own use for the rest of the program. This means that FORTRAN programs that use virtual arrays cannot access the I/O page directly, which is not the case under the FB and SJ monitors. Whenever an element is outside the currently addressed 4K boundary of PAR/PDR set 7, a programmed request is issued to bring the element into the window so that greater than 4K words of virtual array space may be used. Note that the KT-11 is always enabled under XM so that all programs should run slower than under SJ or FB.

The disk based virtual array support package stores the array elements in a file on the DK: device. The program then keeps between 2 and 16 blocks (user defined where one block is 256 words) of array information in main memory. If an array element is required that is not currently in memory, then one of the blocks of information is written out to disk and the block containing the desired element is read back into memory. It is obvious that programs that use this support package will benefit by processing the array elements in sequential order to minimize disk accesses.

4.2 XM Monitor and Extended Memory

The main difference between the XM monitor and the FB monitor is the fact that the XM monitor controls all of physical memory. It knows how much physical memory is attached to the machine and it allocates extended memory on demand. Any program or utility that requires extended memory, requests the space with a programmed request, and returns the memory upon exiting. The XM environment is very controlled and jobs do not and cannot share memory among themselves.

The XM monitor also provides a virtual .SETTOP feature. Normally when jobs issue a .SETTOP request, they are given all of available memory from the top of the program to the bottom of the resident monitor. This means that although the program can address up to 28K words (not including the I/O page), it is only given 28K words minus the size of the monitor. Under XM, when a .SETTOP is issued, the monitor actually allocates a full 28K words of addressable memory to the program. If the full 28K words are not available in low memory, the monitor allocates the additional space in extended memory and handles the mapping for the program in a transparent manner.

Another feature of the XM monitor has to do with the overlaying of programs. Before Version 4 of RT-11, all overlays were disk overlays and segments were read off of disk and overlaid in memory as required. The Version 4 linker has the capability of using extended memory for virtual overlays. So, instead of reading the overlaid routine off of disk, all the overlay segments are stored in extended memory and whenever a segment is called, the run-time overlay handler maps to the particular segment in extended memory instead of reading the segment out of disk storage. This method can significantly reduce program execution time because there is no disk I/O required to bring in the needed segment.

4.3 VM (Virtual Memory) handler

The first time I encountered the VM handler was on the Fall 1979 RT SIG swap tape. It is a full RT-11 handler that makes extended memory look like an RT-11 random access device. That means that you can use it as a system device and actually boot an RT-11 system from it. It directly manipulates the KT-11 hardware registers and is therefore incompatible with the XM monitor. The size of the VM device depends on how much extended memory is on your system. If the full 128K words of available memory are attached to the system, the VM device has 372 blocks of user space. The easiest way to think about the VM device is to think of it as a super-fast disk device that crashes when a new RT-11 monitor is booted. (Actually, the installation code in the VM handler destroys the contents of the VM device as it is determining the amount of memory attached to the system.).

I have decided to include the VM handler in this paper because I believe it provides a very good alternative to re-linking your existing programs to use the /V overlay option of the Version 4 linker. When the VM device is accessed, the handler determines where the block of data is located in extended memory. Then it maps itself to that portion of extended

memory using PAR/PDR register set 7, and then proceeds to transfer the data to lower memory like any non-XM RT-11 handler would. The I/O time is very fast because the access time involves only a simple calculation, and the data transfer is nothing more than a memory-to-memory transfer.

4.4 TSX-PLUS

TSX-PLUS is a multi-user version of RT-11. Each user is allocated up to 28K words of memory somewhere in the available physical memory. Note that each user is given a full 28K word partition, unlike regular RT-11 where the user space is normally 28K words minus the size of the monitor, resident handlers, and USR (User Service Routine). TSX-PLUS does not support any extended memory programmed requests or extended memory overlays. It looks to the user like the SJ monitor, only there can be many jobs running at the same time. Each user is protected from each other and the KT-11 memory management unit usage is transparent to the user.

5.0 The Hardware and Software System

Because this is a timing test, it is important to know something about the test environment. All of the tests were executed on an LSI 11/23 CPU with FPP and MMU chips and 128K words of MOS memory. The system uses one 10 MB U.S. Design winchester disk drive for main storage. The RT-11 monitors used were RT-11 SJ V04.00B, RT-11 FB V04.00E, and RT-11 XM V04.00G; TSX-PLUS version 2.0 was also tested. MACRO-11 version 4.00 and FORTRAN version 2.5 were the language processors used. All programs were linked with the Version 6.01C linker. Note that faster program execution times will be observed for the disk overlaid test configurations if faster disks are used. Let me again add that the purpose of this paper is not to report absolute program execution times, but to report qualitative program execution times to be used for comparisons.

6.0 The Test routine

For the purposes of this paper, I have decided to concentrate only on one parameter--execution time. In my environment, we are most interested in how fast the processor runs our data reduction programs and our data acquisition software. So, for this RT-11 performance measurement, I needed to create a computation bound program. This test routine had to be simple and I knew that once I had established a base-line execution time, I would then re-link the routine using the various options available to me in the linker, and run it in the different RT-11 operating environments. In all cases I will run the test routine as the only job in the system to minimize the interaction with other jobs. My purpose is not to study how many jobs can run in RT-11 at the same time nor is it my purpose to study how the number of running jobs affects system performance. The test routine is made up of five separate .OBJ modules that I will explain.

6.1 Main test routine -- TEST

The test routine is really nothing more than a routine that compares the FORTRAN sine function with a sine function written in MACRO. The line frequency clock is used for timing, and for all the times listed, the error can be one clock tick or 16.6 milliseconds. The main routine begins by making a call to routine TIMEI that gets the current time-of-day from the system. Next the SINTST routine is called and the FORTRAN and MACRO sine functions are evaluated. After 10,000 sine comparisons have been completed, the time-of-day clock is again read and the difference between the beginning and ending times is the program execution time. This time is printed and the test is over. Listing 1 shows the main test routine.

6.2 Timing routines -- TIMEI and TIMEE

The TIMEI, or time-initial, routine gets the current time-of-day from the system using the GTIM request. This value is saved in a COMMON block for later use. The TIMEE, or time-end, routine is called at the end of the timing loop. It also uses the GTIM request to get the current time-of-day. Then the starting and ending times are subtracted using the JSUB, or two word subtract routine, and then converted to seconds. As noted before, the error on the line frequency clock is plus or minus one clock tick or 16.6 milliseconds. I could have used a more precise time base, but I was really only interested in a qualitative time measurement and not the exact time required to execute the test routine. Listing 2 shows both timing routines.

6.3 Sine test routine -- SINTST

The heart of the test routine is in SINTST. Basically, the routine

generates a random floating-point number, and then evaluates its sine with the FORTRAN sine function. Next a MACRO language version of the FORTRAN sine routine is called with the same random floating-point value. The absolute difference between the two results is then taken and if the difference is not zero, a message is printed. Ten thousand numbers are evaluated in this manner before the program ends.

There are two versions of the SINTST routine in order to allow me to check-out virtual arrays. The first SINTST routine in Listing 3 uses a real array for the storage of the random numbers, while the SINTST routine in Listing 4 stores the random numbers in virtual arrays. There are three array accesses per loop in the routine -- one to store the random number, one to call the FORTRAN sine routine, and one to call the MACRO sine routine. Because the loop is executed ten thousands times, and since there are three array accesses per loop, a total of thirty thousand array accesses will be made. With this many accesses, we should be able to get some good statistics on how much time is required to access a virtual array element as compared to a non-virtual array access.

6.4 MACRO sine function -- SIN

The code for the FORTRAN support routines does not directly utilize the FPP (Floating Point Processor) or FIS (Floating-point Instruction Set for LSI-11 or 11/35) floating-point units. Instead, a call is made to a general floating-point operations routine. Then, when the FORTRAN library is built, the proper floating-point routine to support the machine's hardware configuration is inserted into the library with the proper name. For example, the floating-point multiply routine may be called MULF, but MULF may contain an FPP coded routine, an FIS coded routine or a NHD (No Hardware Dependent) routine. This scheme is great for the person who writes the support routines, because he or she does not have to remember how to use the FPP or FIS; just a general calling sequence is used. The problem is that the code takes longer to execute because we have to call a subroutine instead of executing in-line code for the particular hardware unit. In one of my applications, I wanted to speed up a series of sine calculations, and so I modified the FORTRAN sine function to use in-line code. The result is the routine in Listing 5. From the results of this test routine, I have found that it gives the same answer as the FORTRAN sine function.

6.5 Random number generator -- RANDU

The random number generator in Listing 6 is nothing more than a call to the system random number generator RAN. The seeds to the random number generator are saved in a COMMON block so that they will not change if the subroutine is overlaid. The function RANDU returns an floating-point number between 0 and 1.

7.0 Linking Considerations

Simple programs have no trouble fitting into the 28K words of user space available, but as program complexity increases, so does program size. There are two techniques available to the user in RT-11 to circumvent the memory shortage -- chaining and overlaying.

In order to chain programs together, it is first necessary to divide the job to be done into many separate programs, that individually accomplish a portion of the overall task. Then, the last thing that a program does before exiting is to call or schedule the next program in the chain. This technique has the advantage that each sub-program can be checked out individually and then fit into the larger chain. There are numerous disadvantages, such as the fact that files are closed between programs and have to be reopened. Also, any communication or data that is common to all the routines has to be stored in a file, or in a system area that can be accessed by the next job when it is started. Finally, if any of the programs are modified, there is a possibility that they may become too large to fit into available memory. Then that program will have to be broken up into chained programs or overlaying could be used.

When programs are overlaid, part of the program resides on secondary storage (usually a disk), and is read into memory when required. It is possible to have parts of the program share memory space or regions and when one segment is read into memory, it 'overlays' or is copied over an existing segment that is no longer needed. The user has to be very careful in his overlay scheme to insure that the return path for subroutine calls is always in memory and that the return path has not been overlaid by another segment.

The Linker sets up the overlaying mechanism at link time under the direction of the user. When the program is run, a run-time overlay handler is called whenever an overlaid segment is called. This run-time overlay handler reads in the required segment (in the case of disk or /O overlays) or maps to the segment (in the

case of memory resident overlays or /V overlays) and then branches to that section of code. In all cases, the program will take longer to execute due to the fact that parts of the program reside on secondary storage and have to be read into main memory, or because parts of the program are in extended memory and have to be mapped.

8.0 Test Configurations

Now I will discuss the test routine configurations. There are fifteen configurations and their execution times are summarized in Table 2. The first five configurations use low memory data arrays while the last ten configurations use virtual arrays for the storage of data. For each configuration I will discuss how it was linked, how fast it executed under the various RT-11 monitors, how it compared to other test configurations, and then remark on any observed anomalies.

8.1 Non-virtual array configurations

Configurations 1 and 2 will run under all the operating systems while configurations 3, 4 and 5 will only execute under the XM monitor.

8.1.1 Configuration 1 -- Base-line -

This test configuration was tested in all the operating system environments and provides the base-line program execution time that we will use for comparison purposes. Link 1 shows how the program was linked--all parts of the program are memory resident and there are no overlay segments.

One of the results of the first paper was that background jobs will execute in the same amount of time when run under either the SJ or FB monitors. Because of this fact, I only ran the test routines under the FB monitor and assumed that they would require the same amount of time under SJ.

The program execution time for the VM handler is the same as either the FB or SJ monitor with good reason. If we think of the VM device as a very fast disk and if we remember that for non-overlaid programs only two disk accesses are required to bring in the routine (one directory access to locate the program on disk and one to actually read in the program), then the time it takes the program to execute is totally dependent upon the operating system. Because you can only use the VM handler with either the SJ or FB monitors, it follows that program execution time is the same as for the SJ and FB monitors for this configuration.

The XM monitor time is slightly longer than the FB time. The XM monitor is conditionally assembled from the FB monitor, which implies that the internals of both XM and FB are similar. Therefore, the only major difference between the XM and FB monitors must be the fact that the KT-11 is always enabled.

I ran the base-line program under TSX and found that it took longer to execute than under XM. In order to be fair, the test routine was the only job running under TSX and I was the only user. Because TSX is a true multi-user operating system, I can understand why programs will take longer to execute. This time was included only to show potential users the time penalties they would pay in a multi-user system.

So, in summary, the test program executed the fastest FB. The use of the VM handler had no effect on program execution time--the program executed in the same amount of time as FB test routine. The use of the XM monitor cost 1.21 seconds in additional time while TSX required an additional 2.44 seconds of time. Unless you require extended memory support, then either the SJ or FB monitor is the one to use for the fastest program execution times.

8.1.2 Configuration 2 -- Disk overlays -

This test configuration uses disk resident overlays and is shown in Link 2. The two routines SIN and RANDU are disk resident and the test routine alternates execution of those two routines. First a random number is returned from RANDU and then its sine is evaluated by SIN. There is a lot of disk I/O time with this particular configuration and the times in Table 2 reflect this.

The difference between the FB and XM times shows how the effect of the disk being used for overlaying. I believe that I can explain the 161.38 second difference by the fact that FB allows the program to execute the fastest so that the next disk overlay is right under the head when it is called for by the overlay handler. When the same program is run under XM, it executes slower and the disk overlay handler has to wait a full rotation before being able to read in the segment. Another possibility is that one overlay segment spans a track boundary in the XM version, and a head positioning is required before the segment can be read in. Either explanation supports the fact that disk overlayed programs are very dependent upon the characteristics of the disk drive. The

test results that I gathered in the first paper also support the two explanations. With the first paper, I was using slower RK05 disk drives, and the execution time of the FB and XM test configurations was identical.

The real surprise with this configuration is the VM handler time. Again, it is not surprising when we consider that the VM handler is a fast disk. In the next configuration we will want to compare the execution times of virtual overlays with disk overlays using the VM handler.

The TSX execution time is about equal to the XM time because the environments are very similar when only one job is running under TSX. TSX may provide the slowest running environment, but it does provide a multi-user environment where total system throughput and utilization of system resources and time are high.

So, for this test configuration as compared with the baseline configuration, program execution time is between 8 and 12 times greater except when using the VM handler. The additional time required for execution is all due to the disk drive where the overlay segments reside. I can conclude that if you have a very heavily overlaid program that requires a lot of I/O time, you can cut down on the program execution time by using the VM handler.

8.1.3 Configuration 3 -- Virtual overlays -

This configuration is similar to configuration 2 except for the fact that all /O's are changed to /V's as shown in Link 3. This configuration should run faster than disk resident overlays.

The execution time for this configuration is much less than the disk resident configuration because there is no disk I/O time required. Instead of reading the next segment off of disk, the run-time overlay handler maps to the extended memory segment and program execution continues. I had expected to find that the virtual overlays would be faster than disk resident overlays because the overlay segments were always memory resident and only a re-map would be required to access the segment; then I looked at the VM handler time. I was surprised to find that the VM handler time was shorter than the virtually overlaid configuration. This led me to investigate how the run-time overlay handler worked and the result was configuration 4.

Notice also that there is a big time difference between the old virtual overlay handler and the new patched version. By fixing an old problem, DEC has also speeded up the virtual overlay handler so that the VM handler is only a little bit faster than a virtually overlaid program.

8.1.4 Configuration 4 -- Re-map segments -

The VM handler has to be used with disk resident overlays and it works by actually transferring the required overlay segment from extended memory to low memory. When virutal overlays are used, there is no transfer of data; only a re-map is required because the overlay segment is already in memory. Note that the old run-time overlay handler uses the .CRAW programmed request that, in effect, defines a virtual address window and maps it into a physical memory region. I understand that the new patched run-time overlay handler now uses a .MAP request and so should be faster than the old version in configurations where you are bouncing between already created segments. Link 4 shows the test configuration I used to investigate the run-time overlay handler.

The overlay handler exists in two versions. There is the regular version that handles only disk resident overlays and then there is the new Version 4 overlay handler that handles both /V and /O overlay constructs. The proper version is included with the routine when it is linked. Only the regular version is required for disk resident overlays and it is smaller than the overlay handler that supports /V and /O overlay types.

I constructed configuration 4 so that there was only one /V segment. Based upon what I knew about /O segments, I figured that there would only be one map required because there was only one virtually overlaid segment. That was certainly the case for disk resident overlays. I figured that the program execution time would then be close to the XM base-line time. Table 2 shows otherwise and I had to dig into the book for an answer.

From the source code for the run-time overlay handler, I discovered that, unlike the run-time overlay handler for disk resident overlays, the virtual overlay handler always does a .CRAW for every segment and does not check to see if the segment is already mapped! I was quite surprised but there was my answer. Given the fact that the base-line test configuration required 41.53 seconds to execute and configuration 4 required 66.94 seconds, I can assume that the difference in time is due to the run-time overlay handler and the .CRAW programmed request. From the code of the test routines, I know that 10,000 sines were evaluated by SIN (the only routine in the /V section). That

means that the run-time overlay handler and the .CRAW programmed request requires about 2.54 ([66.94 - 41.53] / 10000) milliseconds per usage on an 11/23. That is a lot of time, equivalent to about 508 instructions (assuming 5 microseconds per instruction on the average), but significantly less than the time required to read an overlay segment off of disk.

If we now examine the patched version of the overlay handler, we see that the change from a .CRAW to a .MAP program request, allows the program to execute in about the same amount of time as the base-line configuration, which is what I expected in the first place.

8.1.5 Configuration 5 -- One /V segment -

Link 5 shows test configuration 5. In this configuration all the routines are combined into one virtual overlay segment. That means that only one map is required and the entire program would execute in the one extended memory overlay segment. The execution time for configuration 5 is about the same as for the base-line configuration which says two things. Because we are always executing in an extended overlay segment, the KT-11 memory management unit must always be enabled. Secondly, the old overlay used .CRAW's, which were very expensive. The new, patched, overlay handler uses .MAP's which are considerably faster.

That concludes the discussion on the test configurations that do not use virtual arrays. The overall conclusion that I have reached to this point is that the FB monitor and the VM handler are the best combination to use for large programs. No change is required to your programming style to take advantage of extended memory--again think of the VM handler as a fast, but small disk drive. I think that the limitations imposed upon the programmer in order to use virtual overlays (such as taking 4K words of address space, at a minimum, for each different extended overlay segment) are not worth the time when a faster and easier solution exits (i.e. VM handler). Of course, this is not to say that virtual overlays do not have a place in the RT-11 programmer's bag of tricks. One of the purposes of this paper is to point out the alternatives that are available to the programmer.

8.2 Extended memory based virtual array tes

The following 5 test configurations all use virtual arrays that are stored in extended memory. The virtual array support libraries from DEC exist in two forms--one for the XM monitor and one for the SJ and FB environment. For configurations 6 and 7, I had to use the different libraries and so created separate link command files. Configuration 6 is really the base-line test routine for virtual arrays. Note that only the FB and XM monitors can be tested--TSX and the VM handler do not support virtual arrays stored in extended memory.

8.2.1 Configuration 6 -- Virtual array base-line -

This configuration is similar to configuration 1 except that the data is stored in virtual arrays instead of low memory arrays. Link 6 was used for this configuration. From the source code it can be seen that there are three virtual array accesses per program loop. One access is used to store the random number, another to compute its sine with the FORTRAN sine function, and one more to compute the sine with the MACRO language routine. And since the program looped 10,000 times with 3 virtual array accesses per loop, there were a total of 30,000 accesses per program. I can make some estimates as to the time required to access virtual array elements in the various operating system environments.

The difference in execution time from configuration 1 to configuration 6, for the FB monitor, is 2.41 seconds which we can attribute totally to virtual array accesses. That means that each virtual array access requires 80.3 microseconds (on an 11/23) or about 16 instruction times (again assuming about 5 microseconds per instruction). With the XM monitor, we have a difference of 4.49 seconds between configuration 1 and configuration 6, which translates into 149.6 microseconds per access. From the data I can conclude that the XM environment does not provide a more efficient (i.e. less time) mapping of virtual array elements as was suggested to me at the Fall 80 DECUS convention.

8.2.2 Configuration 7 -- Disk overlays -

Configuration 7 (see Link 7) is really nothing more than configuration 2 with virtual arrays used. Here again we can see the effect of the disk on execution times. In the FB environment, virtual arrays are requiring more time, so that the next disk overlay segment is not positioned exactly under the head when it is called for. The test routine has to wait for more disk latency time than under the first configuration as its total execution time has increased.

Notice that the total execution time has actually decreased for the XM version of the test. I believe that this

is due to again to how the overlay segment is positioned on the disk itself and how long the program has to wait until the required segment comes around under the read/write head.

8.2.3 Configurations 8,9 and 10 -- Virtual overlays -

Configurations 8, 9 and 10 (shown in Link 8, Link 9 and Link 10) do nothing more than confirm the fact that virtual array accesses require about 150 usec. for each element.

8.3 Disk based virtual array test configura

The final 5 test configurations also use virtual arrays, but instead of being stored in extended memory, the array elements are stored on disk. Because the arrays do not use extended memory, I could test this configuration under all the operating systems.

8.3.1 Configuration 11 -- Disk based base-line -

From the times in table 2 we can see that an additional 10 seconds is required for the test routines (see Link 11) that are running under FB, XM, and TSX. The test routine running with the VM handler only required about 6 seconds more, which only proves that the VM "disk" is faster than the winchester disk I was using for the other tests. With 30,000 virtual array accesses per test, it requires about 333 usec. per virtual array element under FB, XM and TSX. Using the VM handler, it only requires 200 usec. per array element. These times are considerably slower than the extended memory virtual arrays, but it is the only way that allows you to use virtual arrays with TSX.

8.3.2 Configuration 12 -- Disk overlays -

In this configuration, I used disk overlays and virtual arrays that were stored on disk as shown in Link 12. Now, we have the times being about equal for FB, XM, and TSX. The FB time was 166 seconds slower that the time of configuration 1 and I think that it is due to the rotational delay in the disk and the fact that the virtual arrays are also stored on the same disk. The execution time for the FB test has gotten progressively slower as I forced the routine to do more processing before calling the next overlay segment.

The real performer for this configuration is the VM handler. Its time of 88.17 seconds is faster than XM configuration 8 that used virtual overlays and virtual arrays that were stored in extended memory. It is something to think about when using the XM monitor.

8.3.3 Configurations 13,14 and 15 -- Virtual overlays -

The final three configurations (shown in Link 13, Link 14 and Link 15) were only run to support the fact that about 10 seconds of time was added to execution time when virtual arrays were stored on disk. Note also that the new patched virtual overlay handler provides for faster program execution than the old virtual overlay handler.

9.0 Normalized times

I have included Table 3, which contains the times of Table 2 normalized to the base-line time of Configuration 1, for reference. It truly shows the relative performance of each of the test configurations in all the RT-11 operating environment. Table 3 can be used to choose the proper RT-11 operating environment for your program or system once you know which options you will be using. For example, if you are going to use only virtual arrays, Table 3 shows that the SJ or FB monitor delivers the best performance with virtual arrays. From Table 3, it can also be seen that the VM handler and the FB monitor, will allow you program to execute the fastest if it is heavily overlaid.

10.0 Conclusions and Recommendations

With all the above data in mind, I have come to the following conclusions.

1. In my opinion the FB monitor has more services and is worth the extra memory. For all types of jobs the XM monitor provides the slowest execution time environment of all the RT-11 monitors.

2. Virtual overlays are about 84% faster than disk resident overlays, depending upon the speed of the disk.

3. Virtual overlays may be optimized by executing within a segment because of the .MAP required to go to the next segment.

4. Programs executing under TSX-Plus take 5% longer than under FB or SJ

and only 2% longer than under XM.

5. Virtual arrays execute the fastest under the SJ and FB environments--XM is less efficient.

6. The use of the VM handler and disk resident overlays is about 3% faster than virtual overlays and XM. I believe that the use of the VM handler with the FB monitor will provide the best enviroment for users with large programs that are heavily overlaid. In addition, I think that the disk resident overlay scheme is easier to use than the virtual overlay scheme because you do not have to worry about PAR/PDR usage and the fact that 4K words is required, as a minimum, for each virtually overlaid region.

7. The patched virtual overlay handler is between 8% and 35% faster than the original overlay handler.

8. Disk based virtual arrays are slower than extended memory stored virtual arrays, but allow virtual arrays to be used with TSX-Plus.

9. The VM handler is still the overall speed champion, but not by much.

Configuration	Monitor SJ/FB	(execution time in seconds) XM	XM+	TSX	VM	Comments
1 (Base-line)	41.53	42.74		43.97	41.53	No virtual arrays
2 (/O segs.)	345.98	507.36		507.42	81.52	No virtual arrays
3 (/V segs.)		90.77	83.69			No virtual arrays
4 (re-map seg)		66.94	43.77			No virtual arrays
5 (one /V seg)		42.52	42.61			No virtual arrays
6 (Vir. array)	43.94	47.23				Virtual array-mem
7 (/O segs.)	426.59	504.80				Virtual array-mem
8 (/V segs.)		95.86	88.34			Virtual array-mem
9 (re-map seg)		71.51	48.27			Virtual array-mem
10 (one /V seg)		47.21	47.18			Virtual array-mem
11 (Disk array)	51.32	52.83		54.20	47.84	Virtual array-disk
12 (/O segs.)	512.43	511.13		511.67	88.17	Virtual array-disk
13 (/V segs.)		101.00	93.63			Virtual array-disk
14 (re-map seg)		76.78	48.36			Virtual array-disk
15 (one /V seg)		52.51	52.69			Virtual array-disk

Table 2

Configuration	Monitor SJ/FB	(execution time in seconds) XM	XM+	TSX	VM	Comments
1 (Base-line)	1.00	1.02		1.06	1.00	No virtual arrays
2 (/O segs.)	8.33	12.22		12.22	1.96	No virtual arrays
3 (/V segs.)		2.19	2.02			No virtual arrays
4 (re-map seg)		1.61	1.05			No virtual arrays
5 (one /V seg)		1.02	1.03			No virtual arrays
6 (Vir. array)	1.06	1.14				Virtual array-mem
7 (/O segs.)	10.27	12.16				Virtual array-mem
8 (/V segs.)		2.31	2.13			Virtual array-mem
9 (re-map seg)		1.72	1.16			Virtual array-mem
10 (one /V seg)		1.14	1.14			Virtual array-mem
11 (Disk array)	1.24	1.27		1.31	1.15	Virtual array-disk
12 (/O segs.)	12.34	12.31		12.32	2.12	Virtual array-disk
13 (/V segs.)		2.43	2.25			Virtual array-disk
14 (re-map seg)		1.85	1.16			Virtual array-disk
15 (one /V seg)		1.26	1.27			Virtual array-disk

Australian RT11 SIG
Librarian

71 Galatea St,
Charleville,
Australia 4470.

Dear Ken,

Herewith the Wish List from the 1981 Australian DECUS. Sorry to be so slow in getting it to you. All discussion, rude remarks, or other interest generated will be counted as well worth while!

A note re RUNOFF: a later version has been submitted to DECUS by this SIG. It has all the patches a la Bob Denny (Miami SIG tape) plus further bug fixes, RT11 double buffered I/O, and the ability to send index information to an external file for people who like long indices. No doubt it still has bugs and limitations, and the more feedback we can get the better.

Regards,
Chester Wilson.

****

RT-11 WISH LIST DECUS AUSTRALIA 1981

****

INTERCOMMUNICATION

. All other DEC systems to be able to read and write RT11 format magnetic tapes!

****

RT-11 SYSTEM

MONITOR ITSELF:

. Power Fail entry point for handlers and/or means to tell RMON there has been a powerfail.

. RMON/cusps as system virtual jobs or something so we can have "underground" (They would have to be "privileged virtual").

. "Completion routine" version of .GTLIN, .TTYIN etc.

. Option to have RMON .CHAIN to a user DECODE routine if an undefined command is found. Alternatively, do-it-yourself documentation for RMON command additions and RMON overlays.

. Transparent print spooler (direct from LP: rather than via a PRINT command).

. Date format on RMON commands to be dd-mmm-yy rather than dd:mmm:yy (RMON can translate to ":" for system programmes).

. RENAME should assume same device on both filespecs.

. Control/R feature like RSTS and RSX (to see what you've typed).

. ? Possibility of allowing command editing eg with cursor keys.

. Delete to take out Caret (^) character if rubbing out control characters.

. RUN command on loaded/resident device should not need to use swap blocks (as R does not), thereby enhancing speed for loading large programmes.

. R command to allow argument lists as per RUN command.

. Better documentation on Virtual Jobs please.

. Fix up the quality control in Software Dispatches.

. DECNET to allow communication directly with VAX asynchronously.

. Separation of TT: handler & terminal drivers from RMON sources, or at least SET TT NOCRLF, TAB in SJ monitor.

. RMON .EXIT (with bit 4000 set) for passing commands should merely go down a level in the indirect file processor rather than wiping out all active indirect files.

. Some way of knowing whether or not a programme is being run via an indirect command file.
(Note that INDFL etc cannot tell if the last line of the command file has been read till it tries for the NEXT line of input).

. SET option to be able to take octal OR decimal value depending upon whether the user has typed 8, 9, or a terminating ".".

. Location in start of .SAV file to indicate DEBUG start address, with a monitor DEBUG command to start at this address.

. Allow ability to have more than 40 characters in the RMON terminal output buffer.

. Indirect command files are read by KNOW block by block, with shuffling taking place between each block's being placed in memory. This is very slow on floppies. It should be possible to find the size of the file, shuffle sufficient space, read it in in one slurp, reorganize as necessary, and possibly have a second shuffle to minimize space taken up by the command file.

. DELETE/QUERY to remain the default option (unlike VAX, RSTS).

. MRRT-11 available in Australia.

. Produce separate command for unloading jobs as against handlers. Alternatively, if the xx in UNLOAD xx is not found to be a job, try the list of handlers next rather than pranging. (That essential ":" is inconsistent and a bloody nuisance.

DIRECTORY OPERATIONS:

. Support for .LOOKUP, .ENTER, .RENAME to have access to extra directory words (plus count of them, for checking!) in a documented manner.

. Date of last access in word in directory (optional, but supported by monitor and system programmes).

. Creation time in directory (ibid. Would need flags in directory header to state that these words were dedicated to this pur and not just additional user-defined words.)

PIP:

. /SINCE, /BEFORE for PIP (very useful for backup with COPY).

. COPY/SETDATE:DD MMM-YR to set date on output file without affecting date on input file or transferring that date to the output.

. Option to confirm transfer if target file already exists.

. COPY/WAIT "files copied" message placed AFTER the "Mount -- volume in device - Continue" messages, and ability to ^C out of COPY/WAIT.

. Copying a file to or from a serial device (eg X.TMP=TT:) leads to an additional line-feed stuck on the end. Can this at least be optional (eg included only with /A)?

. Copying one file to itself only sets the date on that file. Can we have an option to actually copy the file, so as to shift it on the disc (useful with unreliable blocks, and to partly compact a disc). On sequential I/O devices with no date information, the copy should actually be performed (eg TT:=TT: should work).

. Copying to magnetic tape should preserve the old date unless the SETDATE option is used. Otherwise loss of dates on data files results, and can lead to confusion.

KED:

. KED as a virtual (CRUNnable) job.

. Storable macros and/or a startup command file.

. Named macros so that there can be several of them.

. "Hold Screen" and "Seeall" modes as per TECO.

. Aux files to include learned command string.

. (Also other programmes) - Keyboard LED L1 to be lit while in alternate keypad mode.

allowing:

. Command to convert to normal keypad for numeric input, L1 to go out; and vice versa. (then eg single <esc> to return to keypad mode).

. Multiple save areas (dare we call them q-registers? sorry Jim!)

. Multiple paste buffers, even if some of them were limited in size, so that one can go through a file pasting in a single pass. This is probably the same as the above.

. Show command for
1) SET options
2) Paste buffer(s)
3) Delete buffers

. Wild card characters for the search command, even if it IS difficult.

. Multiple search facility: such that a single pass through an (always large) file will pick up any of several strings to be edited.

MACRO, LINK:

. Option for MACRO to output Local Symbols to .OBJ files

. LINK "undefined global" to tell which file(s) referenced the global - should be possible in second pass.

(or) System programme (like GLOB on DEC-10) which provides a cross-referenced list of globals referenced by a series of .OBJ files.

. LINK: change the message which is output when there is not enough disc space for the output file (LINK -Writing text beyond high address) as it isn't very relevant.

DUP, FILEX:

. COPY/WAIT/INTERCHANGE for RX01 only systems.

. Files-11 format in FILEX (definitely RSX, ??? VAX) (wow!)

. SQUEEZE to be able to copy to a disc-image file (thereby of minimum size) for a later COPY/DEVICE/FILE.

. SQUEEZE to be able to append files from one device onto another device already containing file

SYSLIB etc:

. Wild card/character subroutine in SYSLIB (for adding +,% CCI capabilities easily). [Realize this one is most difficult!]

. SYSLIB, FORLIB documentation: an index by function (as well as by name) as can be hard to find what is available.

. SYSLIB support for .SDTIM ENT.

****

FORTRAN IV

. Good INTEGER*4 support.

. INCLUDE-filename for common and data blocks.

. FORTRAN 77 - or if that's too hard, FORTRAN IV PLUS. (Please!)

. Support for virtual arrays on disc files (for little tiny machines!).

. Ability to check for variables which appear only on the right-hand side of expressions

. Ability to have errors and warnings sent to the terminal or to th output file without all the rest of the listing (? /LIERR).

****

Miscellaneous

. Super Star Trek: Missing file called PLAQUE.SAV! (Anyone in DECU who can put me in touch with the writers of this magnificen programme? It would be much appreciated.)

. PASCAL (ISO standard, naturally!).

. C

. BLISS RUNNING on PDP 11's (albeit slow!)

RT-11 WISH LIST

LOS ANGELES 1981 DECUS SYMPOSIUM

Before you read the Wish List from the Los Angeles DECUS, I want to remind you that Wishes are accepted all year round, not just at the biyearly symposia. All you have to do is to send me your suggestions for enhancements that you would like to see on RT. Just write me:

Marilyn Runyon
39 Locust Point Road
Locust, NJ 07760

1. Special Directory-Structured Devices and RT-11 Magtape:

In the current (4.0) and previous (3x) versions of RT-11 magtape was supposedly supported as a special directory structured device via the FSM module. In fact, this is not the case. Large amounts of device dependent code are included in PIP, DUP and DIR to support the file structure on magtape (MT, MM, and MS) and casette tape (CT). The following is a proposal on a method to support special directories under RT-11.

1) Support a form of the .LOOKUP request which returns the file name of the opened file. The form used would be:

.LOOKUP area,chan,dblk,seqnum

where the file name is null and the seqnum argument is >0. The file name would be returned in the dblk area in RAD50. This data could then be used by the utilities to do wild-card lookups and directories.

2) Support a reserved .SPFUN call to initialize the device. In the case of RT-11 magtape this would write the VOL1 header record and LEOT. Data for the volume label (Vol-ID and Owner) could be passed through the data buffer.

3) In handlers with the FILST$ bit set, allow support of

.DELETE amd .rename requests.

The fallout of these changes would create the following features:

1) An alternate FSM module for the magtape handlers (FSMDOS?) to support DOS-11 format magtapes. This would allow RT-11 user to read the Structured Languages SIG Tape would invonveniencing our RSX and RSTS friends. (Also FSMRSX and FSMRST ??)

2) Alternate device handlers for other media to allow (among others) working with IBM Interchange format diskettes without FILES.

(Kenneth Bell, Cucamonga, CA)

2. Need Shareable XM regions. One job can use an XM region for mailbox instead of MQ and reads and writes.

An option in the linker to provide Virtual common for FORTRAN. Now virtual arrays are not shareable but it would be nice.

Option to fix location of an XM region for special devices that can use it for data acquisition, i.e., A/D converters. OM virtual.

(Ned W. Rhodes)

4. KED would be nicer if it told you the NAME and SIZE of the file edited upon exit (like EDT).

4. I need RT V4.0 Sysgened baseline and RX02 - is this a possible combination. Also need VT-11 support and multi-terminal support as a combination.

5. We want RT-11 FORTRAN-77..... Actually what we really must have is a FORTRAN that generates efficient code for the FPU, and perhaps the FPF11 accelerator board for the 11/23. What are the specific problems with bringing FORTRAN-77 UP on RT? Is there anything we can do to get it implemented?

(Robert Walraven, Univ of Calif, Davis)

6. We need an option for DIFF/PRI, DUMP/PRI, PRI, DIR/PRI,

/TDSTAMP DIFF/PRI/TOSTAMP DUMP/TOSTAMP, PRI/TOSTAMP, DIR/PRI/TOSTAMP will place date and time on first line. Also, a way to place directories of many devices on one device and search for which devices contain file.typ. Need support for Directory annotation. Also a means to force reread of directory segment to detect a swapped floppy disk, for instance, SET for .ENTER .CLOSE, also programmed requests.

PIP Copy/wait would permit swapping to DX1: from input,output without disturbing DX0:=SY:. Make LF at end of print optional. It is NOT now possible to print 2 files with a trailing FF and print the next file at the same place on the page without backing up the line prints manually. COPY/QUERY/COMPARE - COMPARE would place on screen a line as follows with length and date:

DEV:FILE.TYP LEN DATE to DEV:FILE.TYP LEN DATE
(if file exists on output device, place marker here
and show existing file)

This would simplify decision to copy or not which presently requires composing and marking to alpha behind directories.

RENAME/SETDATE:DD:MON:yy - place date optionally as parameter to command rather than system date.

Make trailing formfeed optional for compatibility with PIP. Like a SET QUEUE (NO) FORM 2. We place a trailing FF in our text files which

allows us to print multiple copies with

SET LP:NO FORM0. Queue gives a blank page between each copy of the file.

Option to have Queue pause before a selected job printout for 2 paper change. Option for narrow or wide banner page. Option for one line header at top of first page of printout, QUEUE DEV:file.typ len file-date real-date real-time. Command to purge queue of jobs. Provide queue service for DIFF, DIR, DUMP etc. .DIR/QUE .DIFF/QUE

.DUMP/QUE, auto open dump file and place name in the queue.

Support for RA80 as 8 logical devices on SOME new winchester storage! [] control character to exit program without swap auch as D. [] control character to exis programs and save status so that one can continue from where he left off - a proceed, so to speak.

(P.F. Fitts, Innovatek, Millerton, NY)

7. What are the differences between FOFTRAN-PLUS and FORTRAN-77? I would like to see PLUS on RT and have been told that it is not supported. I am a first-time DECUS attendee and am sure that it has been asked before. Is a version of FORTRAN with PLUS features possible or available?

8. I wish a means of defeating the device" full" restriction when adding a file with a size greater than one/half the available remaining space.

(Thomas G. Barnum, Bradley Corp,
Menomonee Falls, WI)

9. Make some utilities VIRTUAL jobs on Version 5, for, example, KED.

10. FORTRAN WISH LIST (RT-11 FORTRAN) 1) Virtual arrays changeable from FORTRAN subroutines. 2) Commons as virtual array entities. 3) Better integer*4 support (as genuine data type).

11. Tell us enough about STATWD bits to ascertain whether another line of input exits in a command file. This would allow a program to receive a variable amount of input from an indirect command file and then switch to user-friendly CRT prompts and special TT mode to solicit input from the user when no more input exists in the command file. (i.e., no echoing). Please do the same in the FB monitor.

(Bob Natale, International Computing Co.)

12. Want FILES-11 (RSX/VAX) support in FILEX. DOS magtape support.

(Ian Hammond, Hammond Software)

13. Want support in FORTRAN (READ-WRITE) for multi-terminal.

14. This is a rewrite of one probably thrown away on Monday. Instead of just writing the device full message when a file to be written is larger than 1/2 the remaining space on the volume, allow the user to decide whether to force the action.

(An explanation of the above is necessary for those of you

who were not at Los Angeles. The clean up crew at the

hotel did their job so well on the first night

that our Wish List box was in the RT-11 campground - it disappeared!

This happened to all other such Wish List boxes, too, so we were

not unique. Fortunately, I

had emptied it at 10:30 that evening, so all was not lost.

Needless to say, it was replaced. M.R.)

15. In the System User Guide, command section, put back in the command names in the upper corner of EVERY page. You had it in V3, but it was lost in V4 and makes the manual mich harder to use.

Make it possible to assign a terminal as LP: (an LS:-type driver) in a multi-terminal system.

Make it possible to obtain control in a multi-terminal system on an arbitrary terminal. If we need another option than having to FIND the active terminal and issue SEE TT CONSOLE=n on it.

FILES-11 support from FILEX!!!

16. Would like to see 9 track mag tape DOS format support in FILEX.

(Steve Macha, Consultant, Stafford, TX)

17. If not multitasking, why not two tasks only - not an unusual need for a single user or use a FORTRAN program in background of FB type moniter.

In KED enter a select field as a find model, set left margin.

Want FORTRAN FF on RT-11, C and/or PASCAL for RT-11, and BASIC debugger for V4.

18. On RT-11 magtape handlers, PIP copies put the CURRENT date on the tape. Since I use the file dates for documenting revision levels, I NEED to have the file date as stored on the disk transferred to the tape. My use for a /SETDATE option here is minimal

(Glenn Bever, NASA DFRF, Edwards, CA)

19. How about efficient mag tape use under RT-11, ie, to back up large disks. (Program similar to ROLLIN.)

20. MQ handler should use RAD50 names so programs can use STANDARD I/O calls. Need an entry point in FORTRAN OTS to tell us how much memory is available before we call IGETSP.

21. In SET TT:SCOPE mode, please handle deletion of control characters correctly. Control characters display as two graphic characters (including "") but rubout only removes the alphabetic character.

22. Need user written duplex serial device drivers and ability in PIP to handle unlabeled (or in general, non-RT-file structured) mag tape.

23. How about a DEC supported C compiler and support library for RT (now that DEC is working on the same for VMS). Change ODT to use .TTYIN etc instead of hard I/O so that when in a multi-terminal system and the console has been changed, one can still use ODT.

24. Provide an option to pass all keyboard input to the application (including S, Q, C, O, etc).

Editor's Note:
The RT-11 SIG would like the wish list to be representative of all RT-11 users (not just RT-11 users at Symposia). Therefore, if you have any comments concerning either RT-11 wish list, please send them to me. I will forward all comments to the RT-11 development group.

SYMPOSIUM TAPE INFORMATION

The following changes are a fix to the SFGL70 Graphics Package which I submitted to the RT-11 SIG Tape in Los Angeles. The problem only occurred if you build the package for a CPU that does not have floating point hardware. Edit the file DISTIC.MAC and make the following changes.

Insert the following SUB instruction between lines 65 & 66:

```
65  XSPTR:  .WORD   0
            SUB     #4.,R4          ;leave R4 pointing to xstart
66          CALL    S$AVARG         ;save regs
```

Change lines 130 & 131 from:

```
130         .WORD   MOF$MM,X$RANGE,M$XRAN   ;set up max arg
131         .WORD   MOI$IS,X$MXCOR          ;get max range
```

to:

```
130         .WORD   MOF$MM,Y$RANGE,M$XRAN   ;set up max arg
131         .WORD   MOI$IS,Y$MXCOR          ;get max range
```

Additionally, change any occurrence of M$OVCUR to **MVCURI** in the routines FLTXT.MAC,FLTXTI.MAC,PLTSYM.MAC,TXTGRD.MAC, & TXTINT.MAC.

I have just finished implementing the necessary changes to SFGL70 to allow it to be used with DEC BASIC. I will be submitting this new package to the Spring 1982 tape in Atlanta. If you are a BASIC language user and would like to try the package before May please contact me. I need the test sites. I'm sure there must be many people using BASIC who would like to have graphics capability on their Tektronix or VT100 terminals.

Ken Demers
203 727-7527

DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
ONE IRON WAY, MR2-3/E55
MARLBORO, MASSACHUSETTS 01752

BULK RATE
U.S. POSTAGE
PAID
PERMIT NO. 129
NORTHBORO, MA
01532

MOVING OR REPLACING A DELEGATE?

Please notify us immediately to guarantee continuing receipt of DECUS literature. Allow up to six weeks for change to take effect.

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: _____

Name: _____

Company: _____

Address: _____

_____

State/Country: _____

Zip/Postal Code: _____

Mail to: DECUS - ATT: Membership
One Iron Way, MR2-3
Marlboro, Massachusetts 01752 USA

Affix mailing label here. If label is not available, print old address here. Include name of installation, company, university, etc.