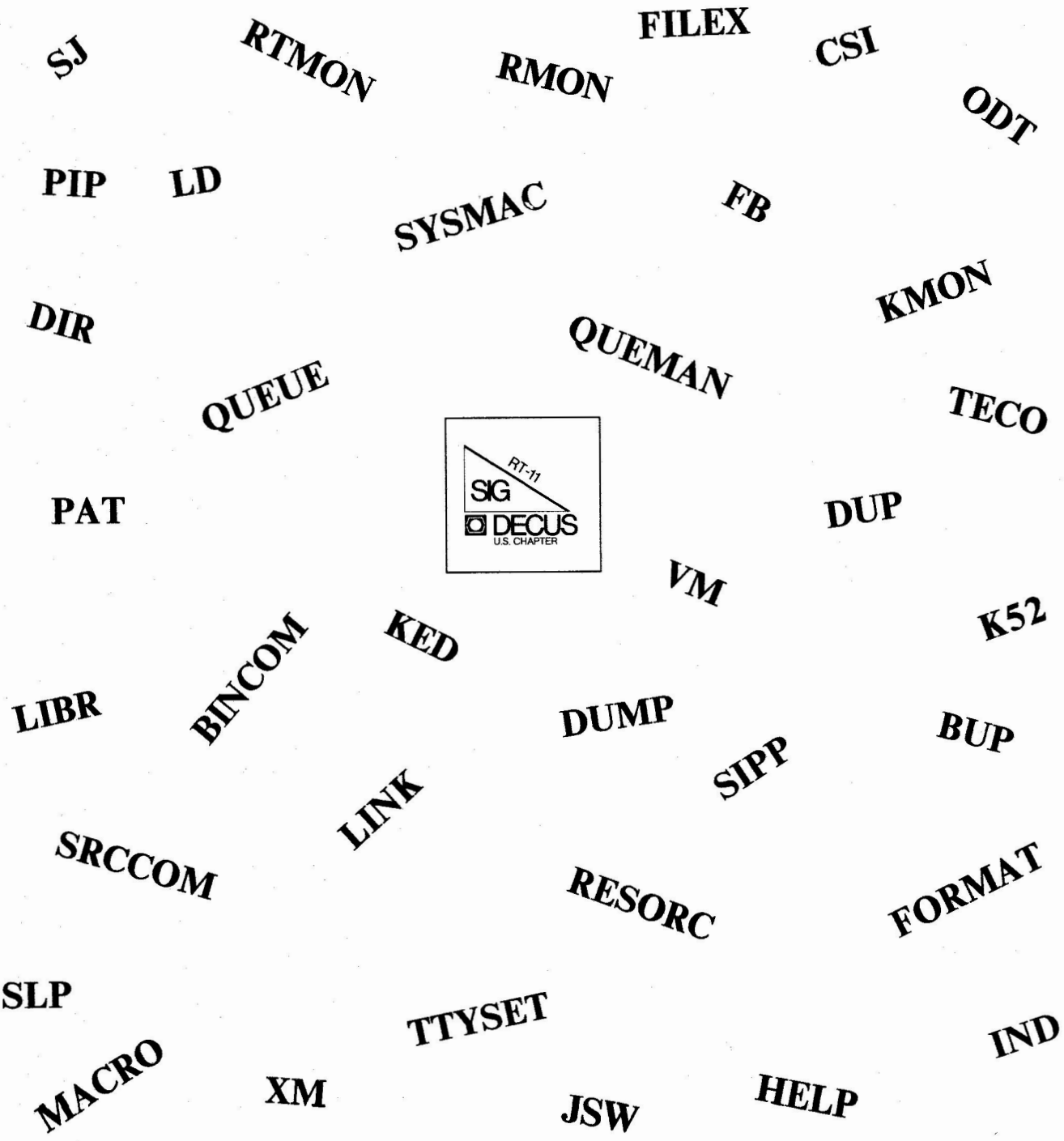
THE mini-tasker

DECUS
RT-11 SIG NEWSLETTER

May 1984 Volume 10, Number 3



*Printed in the U.S.A.*

THE mini·tasker

DECUS
RT-11 SIG NEWSLETTER

MAY 1984 VOLUME 10, NUMBER 3

Contributions to the newsletter should be sent to:

Ken Demers
Adaptive Automation
5 Science Park
New Haven, CT
06511
(203) 436-1029

Other communications can be sent to:

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

or

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

# TABLE OF CONTENTS

# USER INPUT

```
        .TITLE UCL

; This program is a UCL program for RT-11 V5.
; It provides a command parser for simply running programs residing
; on other disks then SY: or DK:
; H. H. NKG-AZG

        .MCALL .PRINT, .EXIT, .CHAIN
        .ENABL LC

        $CLEN   = 510
        $CTEXT  = 512
        $JSW    = 44
        MX.LEN  = 38.
        CHAIN$  = 400


UCL::   BIT     #CHAIN$, @#$JSW         ;Have we been chained to?
        BNE     UCL1
        .PRINT  #IDENT
        BR      UCLEX                   ;No, take exit

UCL1:   CMP     $CLEN,#MX.LEN           ;Yes, can print
        BGT     UCL2                    ;       all error text on 1 line?
;
; See if command matches to one in the command list:
;
        MOV     #COMNDS,R0              ;Command list pointer
        MOV     #PROG50,R3              ;Program list    "
```

Enclosed you find a User Command Language program (V5) with a command parser for executing .RU DEV:NAME by simply typing NAME in response to the monitor dot. The nice thing is that it works independent of the assignments SY: and DK:. The program NAME may also be a little program which chains to another program or indirect command file after e.g. menu selection.

I hope to see a lot of other/better UCL programs from RT-11 users in the MINITASKER!

Yours sincerely,

H. Haenen
Dept. Clin. Neurology AZG
P.O. Box 30.001
9700 RB GRONINGEN / The Netherlands

```
;
NEXT:   MOV     @#510,R1                ;Nr. chars in input command
        MOV     #512, R2                ;Pointer to command string
COMPAR: CMPB    (R0)+,(R2)+
        BEQ     OK
1$:     TSTB    (R0)+                   ;No match: skip rest of string
        BNE     1$                      ;+Skip null byte
        TSTB    (R0)                    ;A second null byte? -> end of all
        BEQ     NOTHNG
        ADD     #6,R3                   ;Point to next program
        BR      NEXT
;
OK:     TSTB    (R0)                    ;Null byte?
        BNE     COMPAR                  ;If not -> keep checking
        MOV     (R3)+,@#500             ;Store program in comm area
        MOV     (R3)+,@#502
        MOV     (R3)+,@#504
        MOV     EXT  ,@#506
        .CHAIN

NOTHNG: .PRINT  #ERRMS1
        MOV     #$CTEXT,R0
        ADD     $CLEN,R0
        MOVB    #200,(R0)               ;SET "NO CRLF" at end of input string.
        .PRINT  #$CTEXT
        .PRINT  #ERRMS2
        BR      UCLEX

UCL2:   .PRINT  #RUDE
UCLEX:  .EXIT

    ; ---------------- DATA AREA'S ----------------

    AREA:   .BLKW   10

    COMNDS: .ASCIZ  /LISDAT/        ;Command list
            .ASCIZ  /STUDAT/
            .ASCIZ  /TABDAT/
            .ASCIZ  /PARDAT/
            .ASCIZ  /CHECK/
            .ASCIZ  /SPELL/
            .ASCIZ  /LIJST/
            .BYTE   0               ;Terminator and End of command list
            .EVEN
    ;-----------------------------------------------------------
    PROG50: .RAD50  /DBSLISDAT/     ;Corresponding program list
            .RAD50  /DBSSTUDAT/
            .RAD50  /DBSTABDAT/
            .RAD50  /DBSPARDAT/
            .RAD50  /DBSCHECK /
            .RAD50  /DL1SPELL /
            .RAD50  /QN5LIJST /
    EXT:    .RAD50  /SAV/

    IDENT:  .ASCIZ  <15><12>/UCL V02.00/
    ERRMS1: .ASCII  <15><12>/?UCL-F-The command "/<200>
    ERRMS2: .ASCIZ  /" has no meaning at NKG-AZG!/
    RUDE:   .ASCIZ  <15><12>/All that typing... and it's wrong/

            .END UCL
```

I read the mini-tasker each time with a lot of pleasure. In this letter I want to react on an article in the last newsletter (January 1984 Vol. 10, nr. 1) and additionally I have a contribution for you.

First my reaction on the article "Method for shortening link maps" on pages 5 to 7. I think that the idea of shortening link maps is good, although it should be better if changes will be made in the link program. A wish that I have submitted already in Zürich, September 1983, but it was not understood by Digital; "a linkmap is never longer than two pages" was the answer. They don't write Fortran! So I made my own procedure but I did it in TECO. Below you see my solution and it has several advantages above that in your newsletter:

1. It runs much easier:

```
.MUNG SHMAP
Long .MAP-file : dd:xxxxxx.MAP
Short map-file : dd:xxxxxx.SMP
```

2. It runs faster.

3. The line with transfer address and program size has been removed to the top of the list. Mostly you are only interested in these numbers.

4. It shows all routine names in the $CODE section.

```
! macro S -- read string and put it in Q9
             0MS => input length unlimited
             nMS => read not more then n characters
             Q8=-1 => ctrl-Z entered
!
@^US#
[0 [1 [2 [3 U3 Q3"E 256U3 ' ETU0 .U1 40ET <-^T;> 14ET 0U9 0,0X9 0U8
<^TU2 13-Q2"E Q1,.X9 :Q9U9 0; '
 127-Q2"E .-Q1"G 1ET 8^T 32^T 8^T 14ET -D ' @0!A! '
 26-Q2"E 26^T -1U8 0; '
 32-Q2"G @0!A! '
 Q3-(.-Q1)"G Q2@I// Q2^T '
 !A!
> 13^T 10^T Q1,.D Q0ET Q9 ]3 ]2 ]1 ]0
#

@EI## 2ED 134ET
!AGAIN!
@^A#Long .MAP-file : # 14MS
@:ER#^EQ9#"U @^A#File not found# 13^T 10^T @0!AGAIN! '
@_#Transfer address# -L 2XA
@ER#^EQ9#
@^A#Short map-file : # 14MS
@EW#^EQ9#
Y 3L B,.P B,.K GA 13@I## 10@I## B,.P
<@:_#Segment size#; 0L B,.K ZJ A 3K J 3L B,.P
 @:_#$CODE#; L B,.K ZJ A 3K J @:_#OTS$0#; 0L B,.P
>
EF HKEX
```

My contribution is a method to simulate include-statements in FORTRAN-IV programs concerning common areas.
If you write FORTRAN programs with a lot of routines you always have trouble if you have to change one or more common areas. You have to edit each routine seperately and probably you make one or more mistakes. These problems are now history! In my solution I use two indirect command files and a TECO procedure. (The TECO procedure is essential).

1. UPDCOM.COM starts the TECO procedure and recompiles the routines changed. Command: @UPDCOM
2. F.COM is my standard FORTRAN compilation procedure command: @F file
3. UPDCOM.TEC changes the common areas and creates an indirect command file "UPDATE.COM" for recompilation. (If a routine has been changed more than once because of changes in more than one common area, the file will be recompiled once.)

Restrictions:

1. Every common area must be placed in a file commonname.CMN. If the common area must be changed, edit only this file. Do not use command-lines in this file!
   Define the common area with the statement "tab COMMON/name/" with no blanks between COMMON and / and between name and /.

2. Enclose the common area in your routines by "C<CR>" lines.

Working of the TECO-procedure:

1. It asks "name of COMMON :" . Give it. CTRL-Z is stop.
2. The file DK:name.CMN will be searched for.
3. In all .FOR files of DK: is searched for the string: tab COMMON/name/
4. If found the text between the C(CR) before and C(CR) after will be deleted and the contents of file name.CMN will be inserted. The name of the routine changed is stored internally.
5. A recompile command file is generated.

Example:

file AREA.CMN

```
      LOGICAL*1  FLAG
      COMMON/AREA/A,B,I,FLAG,
      1           D,E,F
```

FORTRAN routines:

```
      SUBROUTINE   R
C
C
      COMMON/AREA/A,B,I,
      1           D,E,F
C
```

```
                    COMMON/AREA2/....
                .
                .
                .
                C
                .
                .
                .
```

To change all routines with common AREA:

```
@ UPDCOM
Name of COMMON: AREA
etc.
```

```
.ENABLE QUIET
MUN/NOINI SY:UPDCOM
;
; Compile the updated files
;
$@UPDATE
DEL UPDATE.COM
```

Listing UPDCOM.COM

```
.ENABLE QUIET
DEL/NOQ 0:'P1'.OBJ,L:'P1'.LST
RUN SY:FORTRA 0:'P1'[-1],L:'P1'[-1]='P1'/S
```

Listing: F.COM

```
! macro S -- read string and put it in Q9
             0MS => input length unlimited
             nMS => read not more then n characters
             Q8=-1 => ctrl-Z entered
!
@^US#
[0 [1 [2 [3 U3 Q3"E 256U3 ' ETU0 .U1 40ET (-^T;) 14ET 0U9 0,0X9 0U8
(^TU2 13-Q2"E Q1,.X9 :Q9U9 0; '
 127-Q2"E .-Q1"G 1ET 8^T 32^T 8^T 14ET -D ' @0!A! '
 26-Q2"E 26^T -1U8 0; '
 32-Q2"G @0!A! '
 Q3-(.-Q1)"G Q2@I// Q2^T '
 !A!
) 13^T 10^T Q1,.D Q0ET Q9 ]3 ]2 ]1 ]0
#

@EI## 2ED 134ET 0,0XU
!AGAIN! 13^T 10^T @^A#Name of COMMON : # 6MS
1+Q8"E @0!EXIT! '
@:ER#Q9.CMN#"U @^A#UPDCOM-F-COMMON file "# :G9 @^A#.CMN" not found#
                @0!AGAIN! '
@^A#  updated in file :# 10^T 13^T
@EN#*.FOR#
```

```
(:@EN##; G* 13@I## 10@I##)
J 1XF K HXD HK Y HXC HK
(@ER#QF# 0UF
  ( :@_#<TAB>COMMON/Q9/#"S HK @ER#^EQF# @EW#^EQF#
                                  (:@N#<TAB>COMMON/^EQ9/#; -1UF -@S#
C
#
                                   .U0 @S## -1L Q0,.K GC
                                  )
                                | 0;'
  ) EC QF"S @^A#     # GF 2R0T J :XU HK '
  GD -Z; 0J 1XF K HXD HK
)
HK @O!AGÀIN!

!EXIT! HK @EW#UPDATE.COM# GU
16ED J ( .-Z; 1XF .U0 2:@S#QF#"S Q0J K | L ' )
J (.-Z; 3D @I#@F# L2R-4DL )
EX
<ESC><ESC>
```

SY:F

Listing: UPDCOM.TEC

I hope you can place these contributions in one of the following MINI-TASKERS.

Greetings,

Akzo Pharma bv
SYSTEM DEVELOPMENT & AUTOMATION

Ronald Beetz
Akzo Pharma bv
Weth. v. Eschstraat 1
P.O. Box 20
5340 BH Oss
The Netherlands.

---

## RT-11 V5 Report

In about October 1982, after the last Australian DECUS Symposium, I was asked if I wanted to be a Field Test site for RT-11 V5. Filled with a desire for the latest and best from DEC, I naturally accepted at once. In due course, after signing a number of licences, non-disclosure agreements and H-P agreements, a box of floppy discs and a pile of grotty photocopies arrived. It was . . . RT-11 Y05.02. For those who have not met this aspect of DEC's code-numbering, the V stands for "Official Release Version", (supposedly free of bugs), a Y stands for "Field Test Version", (shouldn't have too many bugs, and the users will help find those), and an X stands for "Very Unofficial Highly Experimental Version" (in-house use only, and provided they can get it to link). So I took the photocopies home and read them, and loaded the software onto a couple of my systems at work and booted them. Some of the users still bear (bare) the scars.

DEC provided me with a form called a QAR: Quality Assurance Report. Fortunately, it was in machine-readable form, because we sent off nearly thirty of the things. And quality "Assurance" has nothing to do with it: the function of the form is "Bug Report". I didn't bother with QAR'ing the manuals, as the number of typographical errors was immense, but I had some confidence that DEC could sort that out themselves. (In fact, they did a superb job.)

In due course of time, Y05.08 arrived. One wonders where the intervening versions got to, or what happened to them. Y05.08 certainly removed a few bugs, but it added a few new ones. Somewhere along the line the Linker lost the ability to ^C, so the only way out of it when it got hung (which it did frequently) was to reboot! And the new SL handler managed to get the entire system into Gordian knots. Actually, some of my "testers" developed quite ingenious ways around some of the bugs.

Finally came the message: Field Test is over. Panic: you can't release it in that state! But released it was, and I got my version very early by way of compensation for the testing. Someone at work suggested that it wasn't so much Field Testing as Character Testing. However, on inspection of the finished product (manuals too), I was very agreeably surprised. RT-11 V5 is a smooth, reliable and well-documented product. To be sure, there are a few small bugs (latest count is 24 SPR's, but some were due to a faulty SYSGEN), but it's a very great pleasure to work with. And when I compare it with V2B...

Anyhow, it was great (grate?) fun, and a real education. The twin marvels, of course, were that "they" managed to get so many bugs into the system, and that "they" managed to get almost all of them out again.

## Future RT-11 Features
## or, Is There Life After Death?

We had lots of discussion about this at the last symposium. The DEC visitor for RT-11 was Greg Adams, the new RT-11 Product Manager. To a large degree, Greg is the one who sets RT-11 policy, so he was a very good bloke to lean on. He had lots of ideas to try out on us, and the welcome news that RT is not a dying breed at Maynard (as it was for a while). He was also very easy to get on with - as all RT users seem to be! We definitely enjoyed having him along. The following summary of the magic sessions is from my notes and memories.

First of all, let it be noted that all RT users at the Symposium were very sure of the need for continuing development of RT-11, despite its present (V5) very nice state and the downwards thrust of VAX hardware. The following comments were presented to Greg in that light.

### TSX+ or Multi-Tasking

There is VERY wide support for the merging of RT and TSX+. The ability to "switch terminals" under TSX+ is very popular. The most common use of this seems (to me, RNC) to be the ability to interrupt an editing session to do something else, such as running a spelling checker. The use of KEX in the foreground and compilers etc in the background is a partial step in this direction, but we lack monitor access from the foreground still with this approach. With the increasing use of the 11/23(+) with at least 1/4Mb, not to mention the new 11/73 (J-11 chip set), there seems to be a real demand for a multi-tasking form of RT. The alternative of using RSX did not seem to meet with any form of popular or even polite support. Interestingly, quite a few users had had experience with all three, and did not wish to go to RSX.

RTEM

Greg gave a talk on this, an RT-11 (FB) emulator under RSX. He was very confident about the quality of this product, and assured us that even the RT-11 development group had been using it without prangs for some time. It does not really provide the same functionality as TSX+, and does require a larger system (and a going RSX, which isn't all that easy to find), but it does present one avenue for users for expansion. Only a few EMT's don't work (are not supported), and those ones aren't all that common.

RT11SJ

The possibility of dropping this Monitor was discussed, and met with little objection, provided that a version of FB of a similar small size becomes available. That is, the only real value seen in the SJ monitor was its small size. The nuisance value of the differences between SJ and FB was seen as significant both to the users and to DEC. DEC have to support two different monitors, essentially, for no real gain. The users have to go to all sorts of devious and annoying machinations to try to make programs behave the same under either monitor. (My personal opinion is that SJ is a menace!)

RT11XM

Few people use this monitor at present (which was mildly embarrassing, as Greg more or less wrote it!). This is at least partly due to the reputation it acquired on it's first release, when it was a bit wonky. The availability of KEX may help to alter this. That is, since most users have managed without XM in the past, there is a certain inertia acting against any change. Given good reason, such as KEX, this could slowly change. The question remains however whether the present XM is what is really wanted. Many (most) saw a great demand for a revamped version as outlined above under TSX+.

The New RT11.SYS

Out of all the above comes a proposal for a new form of RT-11. It would feature only one monitor, called RT11.SYS. This would be like the FB monitor, but with many tuning options. A small version of limited functionality for those users with very large SJ-type run-time packages would be essential. An average version for the typical FB user would be the middle-of-the-road offering, presumably to be known as Good Old RT (GORT). Only half the GORT users might ever actually use the foreground. At a somewhat larger core-size, we could have a virtual version of RT which offered much increased functionality. That is, the user would (probably?) be in page 0, but most of the monitor routines would be in virtual memory. This should lead to a very smart and fast system. Switching to virtual memory would be done only by the monitor (via EMT's) for its own purposes, so that there should be very few bugs and user software should stay simple. Alternately, we could have a more sophisticated linker which would put overlays in extended memory to the extent of the space available and the rest on disc automatically: overlay changing would then be then just a matter of juggling the PAR's most of the time. This would have remarkable effects on the speed of the Fortran compiler (about 24 overlays) and the linker itself (10 about overlays). Finally, at the high end, we would have the multi-tasking (and why not multi-user) version of RT/TSX+. This might well, in the long run, eclipse all the others. The problem here is that, while conceptually quite different from the internals of RSX, it would effectively be competing with it (unless all RSX users migrate quickly to VMS, which seems quite on the cards).

The key point here is the radical change in memory prices: there is little difference between a 64Kb memory board and a 256Kb board (MSV11-DD @ A$842 and MSV11-LK @ A$1755), relative to either disc prices or current "A-class" software licences. This means that the old ways of thinking (RT in 8Kb) are now irrelevant. That's NOT to say that we want the monitor to grow to RSX/UNIX size! The "visible" (ie in page 0) part MUST stay as small as possible. But the extended memory could well be used to immense advantage. I foresee a problem in retaining compatability between 16-bit (LSI11/2) and 18/22-bit systems, but this could surely be handled with the options mentioned above, or with a clever bit of "paging" code, which either loads from disk or switches to extended memory. This code would be in the resident monitor, and would be used by the monitor as well as CUSP's and user programs.

I can see that somewhere along the line a degree of imcompatibility will eventually be introduced. I see it, I regret it, and I welcome it. The basic GORT would hopefully hide it, while the more powerful version would show it. But if advantages outweigh the disadvantages, I believe that it would be worth while. What do you think, and what would you want for it to be worth while?

Non-SYSGEN'ed Standard System Package

Numerous comparisons were drawn between RT and CP/M-MS/DOS. The virtues of RT were not in dispute, but the price of a full licence was seen as a major problem. As many users don't do a SYSGEN, the possibility of a standard 1 or 2 Floppy distribution kit with a single small manual, along the lines of the Rainbow and Decmate software kits, was brought up. This would run quite happily on packaged systems (including the PC350 if possible), and at (say) $300 would sell like hot cakes. This idea partly presumes the single-monitor concept outlined above. The monitor provided would probably be GORT.

If such an idea could be pushed through, there would need to be another "full" kit available, with complete documentation and fully commented sources. This source kit need not contain any compiled files (apart from MACRO and LINK, if not included in the basic kit), as the basic kit would be quite sufficient to get you up and going. A price of $1000 for the full kit would be reasonable. The kit of the full range of manuals would be a separate item again, priced at whatever is (un)reasonable (as usual).

These ideas were kicked around by quite a few people on several occasions, and Greg was an very keen participant in the discussions. I got the message that Greg was very interested in finding out just what the users wanted: how many radical changes he can push through may be another matter, of course, but at least DEC is listening.

Roger Caffin
Australian RT-11 SIG Chairman

The Australian RT-11 SIG Tape!

This long awaited Oeuvre approaches gestation. It has changed form a bit along the way, for a number of reasons. Many of the SIG tapes contain a lot of repetition from year to year: this is unnecessary and even undesirable. Some of the programs put on the tapes have been Library items, and there has been some debate over these. The conclusion we seem to have come to is as follows. It is quite appropriate for a program submitter to bring his program to the Symposium and make it available to all attendees: it is his, after all. After the Symposium however it is taken off the SIG tape as it is not in the SIG's interest to give it away when the Library is relying on the revenue.

Anyhow, this is the index to the collections. Note that all "files" fit on a single RX01, even if they look bigger. The files are actually floppy images, and can be accessed via LD: or XD:, or may be copied to a floppy and accessed normally. The "extra" space is the invisible space (directory, etc) at the front of the floppy. The whole tape has been submitted to the DECUS library.

| "File" | Size | Date | Description |
|---|---|---|---|
| AR .DSK | 40P | 10-Aug-83 | ARchives NEW versions of files to a backup |
| BANLIF.DSK | 80P | 11-Aug-83 | Banner and Life (old?) |
| BASEXT.DSK | 419P | 06-May-83 | Peek/Poke type extensions to Basic |
| CALHAE.DSK | 27P | 06-May-83 | Decus paper on interupts in Fortran |
| CCHIT1.DSK | 446P | 01-Sep-83 | Utilities and C extensions |
| CCHIT2.DSK | 435P | 01-Sep-83 | " |
| CCHIT3.DSK | 469P | 01-Sep-83 | " |
| CDUMP .DSK | 140P | 07-Sep-83 | Disc Dump Utility |
| CGW1 .DSK | 494P | 01-Sep-83 | CB, CSPLM foreground spooler, CSPOOL, ... |
| CGW2 .DSK | 418P | 01-Sep-83 | LR/LW Comms handlers, C bits and games |
| CHESS .DSK | 230P | 11-Aug-83 | Chess game + DOC |
| CHITL1.DSK | 494P | 29-Jun-83 | Utilities and Fortran extensions |
| CHITL2.DSK | 494P | 29-Jun-83 | " |
| COMM .DSK | 200P | 20-Oct-83 | Terminal, Net, Comms: Communications pkg's |
| COMPAG.DSK | 84P | 13-Jul-82 | Multi-column post-RUNOFF processor |
| CP .DSK | 40P | 10-Aug-83 | Universal Copy Program |
| DECODE.DSK | 70P | 22-Jul-82 | PDP-11 dissembler |
| DECUS1.DSK | 449P | 12-May-83 | Sundry bits |
| DEMO .DSK | 800P | 31-Aug-83 | VT125 pictures |
| DXSCAN.DSK | 134P | 06-May-83 | Floppy disc analyser |
| EXFILE.DSK | 68P | 22-Jul-82 | Interprocessor Communications Program |
| FGMON .DSK | 72P | 07-Sep-83 | Foreground Monitor for running several progs |
| FLECS1.DSK | 455P | 11-Aug-83 | Fortran Language w Extended Control Structures |
| FLECS2.DSK | 300P | 11-Aug-83 | - - - - - - |
| FLECS3.DSK | 435P | 11-Aug-83 | " |
| FLECS4.DSK | 120P | 11-Aug-83 | " |
| FLECS5.DSK | 285P | 11-Aug-83 | " |
| FLOPPY.DSK | 494P | 30-Aug-83 | Sundry (comms), inc Screen |
| GAMES .DSK | 471P | 12-May-83 | Various Games |
| GRAPH1.DSK | 340P | 10-Aug-83 | Graphics Software |
| GRAPH2.DSK | 398P | 10-Aug-83 | " |
| KB .DSK | 58P | 22-Jul-82 | KB device handler |
| LSTPRO.DSK | 60P | 12-May-83 | List Processor for letters |
| ODDS .DSK | 500P | 28-Jun-83 | Various Games |
| PICAX .DSK | 266P | 12-May-83 | Program for Interactive Control and eXperiment |
| RATFOR.DSK | 380P | 11-Aug-83 | RATFOR; Rational Fortran |
| RAYB .DSK | 110P | 30-Aug-83 | XNET, PO.MAC |
| RESEQ .DSK | 190P | 10-Aug-83 | Fortran Line Number Resequencer |
| RHODES.DSK | 166P | 22-Jul-82 | Sundry |
| SFOC .DSK | 125P | 07-Sep-83 | Structured Focal Extensions |
| SORT .DSK | 341P | 29-Aug-83 | Sort Utility |
| SPOOL .DSK | 87P | 22-Jul-82 | Greg Adam's Transparent Spooler |
| STAGE2.DSK | 75P | 11-Aug-83 | Stage 2 Macro Processor |
| STUFF .DSK | 494P | 02-Sep-83 | Greg Adam's UCL.SAV, and other bits |
| TSTE .DSK | 240P | 10-Aug-83 | Time Share Terminal Emulator |
| TTLIB .DSK | 432P | 29-Aug-83 | VT100 Library |
| UPDATE.DSK | 65P | 10-Aug-83 | Archive disk Updating program |
| VARRAY.DSK | 79P | 12-May-83 | C extensions for virtual arrays |
| VIRTAL.DSK | 40P | 10-Aug-83 | Fortran Ext'ns for Disk-based Virtual Arrays |
| XASM .DSK | 50P | 11-Aug-83 | Cross assemblers for 8080 & 6800 |
| XD .DSK | 150P | 10-Aug-83 | Logical device handler |

For a limited time for Australian DECUS members, the individual DSK components of this tape may be obtained in the normal manner. You send enough floppies, in a plastic bag, in reinforcing such as masonite or metal, in a Jiffy bag, with a return address label and return postage, to RNC. You include a list of what files you would like. NOTE: a single "file" request will probably be restored to the floppy as a directly readable disk, but multiple requests will usually be copied as files, packed on floppies, and you will have to unpack them. For those who haven't done this before, the following should do it with standard RT-11:

```
COPY/FILE/DEV dva:NAME.DSK dvb:
```

This copies a file on dva: to the device dvb: itself. That is, the device dvb: is written on starting at block 0. This means that the directory structure in the file is transferred to the dvb: too. The device size will probably then appear less than it should. In V5 this can be fixed by SQUEEZEing the disc.

Alternate media supported are mag tape (800 or 1600 bpi, but SPECIFY!) and RL02 disks. Tape should be sent packaged like floppies. What you do with RL02 disks is up to you, but remember that CSIRO can't cover any costs.

Turn-around time for single floppies should be short: only a few days. For requests for "lots of floppies" it will be longer. Mag tape will also take a bit longer, although with 2400' tape the fastest is to just dump the lot (COPY DL1: MT:). RL02 disks will also take a few days, and again the fastest is to dump the lot.

## Sundry files in the SIG tape

Many of the "files" listed on the SIG tape are obviously part of a system, and so are not listed separately. However, there are also many little bits, in several collections. These are listed below:

| Program | Size | Date | "File" | Description |
|---|---|---|---|---|
| AD .MAC | 4 | 01-Jul-82 | RHODES | |
| ADTEST.FOR | 1 | 01-Jul-82 | RHODES | |
| ANIMAL.DAT | 32 | 21-Aug-83 | CGW2 | Data for CELL |
| BANNER.SAV | 52 | 22-Sep-80 | BANLIF | Prints large text banners |
| BANNER.SAV | 52 | 17-Dec-79 | GAMES | Probably the same |
| BB .MAC | 83 | 28-Dec-81 | CGW1 | BB/CB package |
| BB .SAV | 13 | 28-Dec-81 | CGW1 | BB/CB package |
| BL .MAC | 36 | 28-Dec-81 | CGW1 | BB/CB package |
| BL .SAV | 7 | 28-Dec-81 | CGW1 | BB/CB package |
| CALC .SAV | 32 | 05-Feb-81 | STUFF | "Calculator" on screen (very simple) |
| CB .MAC | 34 | 22-Dec-81 | CGW1 | BB/CB package |
| CB .RNO | 109 | 28-Dec-81 | CGW1 | BB/CB package |
| CB .SAV | 10 | 22-Dec-81 | CGW1 | BB/CB package |
| CDUMP .SAV | 48P | 06-Sep-78 | STUFF | CGW's special dump program |
| CELL .C | 48 | 24-Aug-83 | CGW2 | Picture drawing program, uses ANIMAL |
| CELL .COM | 1 | 20-Jul-83 | CGW2 | CELL.C |
| CELL .SAV | 26 | 24-Aug-83 | CGW2 | CELL.C |
| CLIB .OBJ | 81 | 28-Mar-82 | CGW2 | C library |
| CLIST .C | 10 | 20-Aug-83 | CGW2 | C Pretty Printer |
| CLIST .COM | 1 | 21-Aug-83 | CGW2 | CLIST |
| CLIST .SAV | 15 | 21-Aug-83 | CGW2 | CLIST |
| CLK100.MAC | 4 | 01-Jul-82 | RHODES | Clock display |
| COM .MAC | 32 | 21-Jul-82 | ODDS | |
| COM .MAC | 32 | 21-Jul-82 | DECUS1 | |
| COM .MAC | 38 | 24-Aug-83 | CGW2 | Interprocessor Comms prog, uses LR, LW |

| | | | | | |
|---|---|---|---|---|---|
| COM | .SAV | 7 | 24-Aug-83 | CGW2 | COM |
| CONSOL | .MAC | 1 | 01-Jul-82 | RHODES | |
| CSPLM | .MAC | 27 | 05-Jul-80 | CGW1 | CGW's Foreground Spooler (good) |
| CSPLM | .TXT | 7 | 05-Aug-79 | CGW1 | " |
| CSPOOL | .COM | 1 | 03-Dec-78 | CGW1 | " |
| CSPOOL | .INI | 1 | 02-Dec-78 | CGW1 | " |
| CSPOOL | .MAC | 34 | 03-Dec-78 | CGW1 | Greg Adam's Foreground Spooler |
| CSPOOL | .REL | 7 | 03-Dec-78 | CGW1 | " |
| CSPOOL | .TXT | 10 | 04-Dec-78 | CGW1 | " |
| DECUS | .DOC | 6 | 18-Jul-82 | CALHAE | Interrupts in Fortran (paper) |
| DECUS | .FOR | 13 | 24-Jul-82 | CALHAE | " |
| DIREC | .MAC | 12 | 02-Dec-78 | CGW1 | Fortran access to directory entries |
| DISASM | .SAV | 17 | 04-Mar-82 | STUFF | Disassembler |
| DISC | .SAV | 9 | 19-Aug-81 | STUFF | |
| DX | .C | 27 | 20-Aug-83 | CGW2 | Accesses individual sectors, (+RDSEC) |
| DX | .COM | 1 | 21-Aug-83 | CGW2 | DX.C |
| DX | .SAV | 19 | 21-Aug-83 | CGW2 | DX.C |
| FLAG | .SAV | 17 | 16-Mar-83 | STUFF | Banner heading on listing |
| GO | .FOR | 8 | 01-Jul-82 | RHODES | |
| HANDLE | .SAV | 7 | 18-May-82 | STUFF | Decodes block 0 of %%.SYS to TT: |
| HS | .MAC | 28 | 01-Jul-82 | RHODES | |
| HSPOOL | .MAC | 4 | 01-Jul-82 | RHODES | |
| KB2 | .MAC | 33 | 01-Jul-82 | RHODES | |
| LIFE | .SAV | 8 | 22-Sep-80 | BANLIF | Conroy's LIFE |
| LIFE | .SAV | 11 | 12-Oct-79 | GAMES | " |
| LIFE | .TXT | 3 | 22-Sep-80 | BANLIF | " |
| LPC | .MAC | 34 | 05-Dec-78 | CGW1 | Alternate LP handler |
| LPC | .TXT | 7 | 05-Aug-79 | CGW1 | Alternate LP handler |
| LPCF | .MAC | 34 | 12-May-79 | CGW1 | Alternate LP handler |
| LR | .MAC | 12 | 03-Jun-82 | CGW2 | LR/LW Comms Handlers |
| LR | .SYS | 3 | 03-Jun-82 | CGW2 | LR/LW Comms Handlers |
| LW | .MAC | 6 | 03-Jun-82 | CGW2 | LR/LW Comms Handlers |
| LW | .SYS | 2 | 03-Jun-82 | CGW2 | LR/LW Comms Handlers |
| MACLIB | .MAC | 38 | 01-Jul-82 | RHODES | |
| MAILER | .MAC | 17 | 28-Apr-81 | DECUS1 | Types mailing labels from list |
| MAILER | .SAV | 6 | 18-Aug-81 | DECUS1 | MAILER.MAC |
| MC | .MAC | 4 | 04-Dec-78 | CGW1 | Odd macros |
| MTCON | .FOR | 7 | 01-Jul-82 | RHODES | |
| NET | .MAC | 60 | | COMM | Communications |
| NZPAT | .MAC | 98 | 23-Jul-82 | DECUS1 | Versatile Patch program |
| NZPAT | .SAV | 14 | 23-Jul-82 | DECUS1 | NZPAT.MAC |
| NZPAT | .TXT | 6 | 22-Aug-82 | DECUS1 | NZPAT.MAC |
| PEEK | .FOR | 1 | 01-Jul-82 | RHODES | |
| PLANE | .SAV | 48 | 18-May-80 | GAMES | 747 landing game |
| PO | .MAC | 35 | 25-Aug-83 | RAYB | RT-11/TSX+ DLV-11J Port Handler |
| POKE | .FOR | 1 | 01-Jul-82 | RHODES | |
| PROGRA | .SAV | 6 | 17-Aug-82 | STUFF | Decodes block 0 of *.SAV to TT: |
| PUSSY | .CAT | 103 | 05-Dec-80 | ODDS | LP Picture File |
| RDSEC | .MAC | 1 | 19-Aug-83 | CGW2 | Subroutine for DX.C |
| RDSEC | .OBJ | 1 | 19-Aug-83 | CGW2 | RDSEC.MAC |
| READ | .ME | 3 | 21-Aug-83 | CGW2 | Various explanatory text files |
| READ | .ME | 3 | 24-Jul-82 | DECUS1 | " |
| READ | .ME | 3 | 24-Jul-82 | ODDS | " |
| RHODES | .RME | 2 | 01-Jul-82 | RHODES | " |
| RTMON | .MAC | 33 | 12-May-83 | STUFF | "Monitor" for multi-terminal |
| SCAN | .DOC | 2 | 02-Sep-83 | STUFF | Compares contents of two discs |
| SCAN | .SAV | 24 | 25-Jul-83 | STUFF | SCAN |
| SCREEN | .MAC | 4 | 01-Jul-82 | RHODES | |
| SETUP | .SAV | 7 | 25-Aug-83 | STUFF | Sends ESC string to VT100 to set it up |
| SLIDE | .SAV | 27 | 02-Dec-82 | STUFF | Automated "slide" maker (on VT100) |

| | | | | | |
|---|---|---|---|---|---|
| SNAKE | .FOR | 20 | 28-Aug-82 | CGW2 | Game: worm on screen |
| SNAKE | .SAV | 31 | 28-Aug-82 | CGW2 | Snake.for |
| SP | .SYS | 2 | 06-Jun-83 | STUFF | Part of SPOOL |
| SPACE | .DAT | 1 | 24-Jul-82 | ODDS | Space Invaders (11-510) |
| SPACE | .FOR | 66 | 22-Jul-82 | ODDS | " |
| SPACE | .SAV | 60 | 24-Jul-82 | ODDS | " |
| SPACHK | .MAC | 2 | 22-Aug-82 | ODDS | " |
| SPOOL | .DOC | 2 | 02-Sep-83 | STUFF | Greg Adams Transparent Spooler |
| SPOOL | .REL | 8 | 22-Aug-83 | STUFF | SPOOL |
| SPOOLX | .REL | 7 | 11-Aug-83 | STUFF | SPOOL |
| SPX | .SYS | 2 | 10-Aug-83 | STUFF | |
| SPY | .FOR | 1 | 01-Jul-82 | RHODES | |
| SST | .DOC | 118 | 09-Jan-78 | GAMES | Super Star Trek |
| SST | .SAV | 184 | 09-Jan-78 | GAMES | " (partial, lacks PLAQUE.SAV) |
| STDIO | .H | 7 | 16-Mar-82 | CGW2 | C Library |
| SUPORT | .OBJ | 1 | 28-Mar-82 | CGW2 | C routines |
| TAB | .MAC | 10 | 18-Aug-81 | DECUS1 | Compresses <TAB>/<SP> to DEC standard |
| TAB | .SAV | 5 | 18-Aug-81 | DECUS1 | TAB.MAC |
| TERMNL | .MAC | 22 | | COMM | Communications |
| TICTAC | .SAV | 10 | 14-Sep-80 | GAMES | 3-D Tic-Tac-Toe (?) |
| TO | .MAC | 14 | 23-Aug-83 | CGW2 | TT Interceptor, Shows control ch's |
| TO | .SAV | 3 | 23-Aug-83 | CGW2 | TO.MAC |
| TRACE | .SAV | 17 | 19-Apr-83 | STUFF | Execution tracer. Put on SY:. Details? |
| UCL | .SAV | 11 | 13-Aug-83 | STUFF | User Command Linkage, G Adams |
| V50VER | .SLI | 8 | 02-Sep-83 | STUFF | Slides to go with SLIDE: shows format |
| VM | .MAC | 21 | 01-Jul-82 | RHODES | DECUS VM handler (?) |
| WORM1 | .FOR | 3 | 02-Oct-81 | GAMES | Patterns on screen |
| WORM1 | .SAV | 17 | 01-Oct-81 | GAMES | " |
| WORM3 | .FOR | 3 | 21-May-82 | GAMES | " |
| WORM3 | .SAV | 17 | 21-May-82 | GAMES | " |
| XFIL | .MAC | 7 | 03-Dec-78 | CGW1 | File transfer, channel to channel |
| XNET | .MAC | 67 | 09-Aug-83 | RAYB | Comms Package (uses PO) |

Not all files have descriptions. Some are obvious, some are not. The latter lack a description for that reason: RNC doesn't know what they do. Help would be appreciated. You might also notice some duplicates: the clean-up hasn't been finished yet. But for what it is worth.....

Please note: requests for files from this list will not be filled. If you want one, request the .DSK file.

R N Caffin
Australian RT-11 SIG Chairman

# DECUS LIBRARY

TSXLIB Updated

Like RT-11, TSX-Plus offers the MACRO programmer a number of system services via programmed requests or EMTs. RT-11 makes its system services available to the FORTRAN programmer through the system subroutine library, SYSLIB. TSXLIB makes the TSX-Plus EMTs available to the FORTRAN programmer as a library of callable routines. The package includes the MACRO source modules for all the routines, a user's manual in machine readable form, a cross reference chart, an indirect command file to build the library, and the implemented library.

The library has been updated to include all EMTs through TSX-Plus V5.0. It is available from the DECUS Program Library (order #11-490) on RX01 Floppy Diskette (KA) and 600-ft Magtape (MA), both in RT-11 format. The address for the DECUS Program Library is:

DECUS
One Iron Way
Marlboro, MA 01752

With this release, maintenance of TSXLIB has been assumed by NAB Software Services, Inc. of Albuquerque, NM.

N. A. Bourgeois, Jr.
3-Apr-84

C Language System (Binary Version) for RT-11, for the Professional-300 Series

Version: December 1983

Author: Robert Denny, Martin Minnow, David Conroy, Charles Forsythe

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrail Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: MACRO

Memory Required: 24KW Minimum

"C" is a general purpose programming language well suited for professional usage. The DECUS "C" distribution contains a complete "C" programming system including:

- A compiler for the "C" language. The entire language is supported except for floating-point, macros with arguments, bit fields and enumeration.
- A common runtime library (standard I/O library) for "C" programs running uder the RT-11 operating system. By using this library, "C" programs may be developed on one operating system for eventual use on another.
- Several "C" programs, including a cross-referencer lister for "C" programs, a lexical analyser program generator, cross-assemblers for several microcomputers, and several games.
- Extensive documentation for the compiler and runtime library.

Note: Sources are not included. All software is distributed in Binary format. For sources users should order DECUS No. 11-SP-18, which is a more complete offering.

Restrictions: Documentation in RUNOFF format only.

Sources are not included. Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskettes (JB)

Format: RT-11

Keywords: Structured Languages, PRO - 300 Series, RT-11 - PRO
Operating System Index: RT-11

RUNOFF M02.4H for RSTS, RSX and RT, for the Professional - 300 Series

Version: December 1983

Author: Chester Wilson

Submitted By: Ian Calhaem Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 or higher for Professional-300 Series support,
P/OS V1.7 or higher as RSX files need FCS support

Source Language: MACRO

Memory Required: 8KW to 10KW

Document preparation is greatly aided by RUNOFF. Automatic line fill, right margin justification, hyphenation, pagination, index creation and decimal notation sectioning are among the facilities provided. This program is an updated and enhanced version of RUNOFF (DECUS No. 11-530). This release supports the RT and P/OS operating systems with a common baseline.

This version of RUNOFF has a modified hyphenation algorithm, conforming to the UNIX V7 table and digram threshold values. It also has support for transparent printer control strings which are passed directly to the output without affecting the fill and justify processing. This facility makes it possible to use special features found on many letter quality printers, as well as implementation of specialized pre-processor programs which can provide mathematical typesetting, graphics, etc.

Restrictions: To assemble under RT-11 V5.1 set KMON NOINT.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT-11

Keywords: RUNOFF, RT-11 - PRO, Text Manipulation, PRO - 300 Series
Operating System Index: RT-11, P/OS

TTLIB: VT100 Library, for the Professional - 300 Series

Version: December 1983

Author: Chester Wilson

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: MACRO

TTLIB is a library of programs to conveniently control a VT100 type terminal in ANSI mode. Routines allow drawing boxes and lines, cursor positioning, screen appearance, video attributes, screen and line clearing, screen and keyboard behavior, graphics facilities, assorted heights and widths, tab settings and clearings, and reporting cursor position.

Routines are provided for FORTRAN and MACRO calling programs. As TTLIB is a library, only the routines which are actually used are loaded into the user's programs.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT-11

Keywords: VT100 Routines, RT-11 - Libraries, RT-11 - PRO, PRO - 300 Series
Operating System Index: RT-11

COMPAG: Combine Pages for the Professional - 300 Series

Version: December 1983

Author: Chester Wilson

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: MACRO

COMPAG provides the ability to easily combine pages so they may be spread across a printer form rather than taking up a page each. It can cope with up to 8 columns within the output page, the size of each column being determined by the width of the page. Alternatively a series of left margins may be specified, one for each column in sequence.

Tab conversion is usually performed on the output file, to reduce size. This uses the DEC standard for hardware tabs. Underscoring in the input file will be retained in the output.

COMPAG was designed to be used with RUNOFF output files, but will work with any files.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT-11

CVLLIB: General Purpose Library for RT MACRO and FORTRAN, for the Professional - 300 Series

Version: December 1983

Author: Chester Wilson

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: MACRO

Special Hardware Required: FIS or FPU to make use of floating point routines.

CVLLIB is a personal general-purpose library for the RT-11 MACRO and FORTRAN programmer. Routines cover facilities such as reading and writing decimal (up to triple precision) and octal (up to double precision) integers, money formats (double and triple precision), dates and times, filenames and RAD50 formats.

The library has been split into three segments:

| | | |
|---|---|---|
| General/MACRO portion | CVGLIB | Manual: CVLLIB |
| Real Number portion | CVRLIB | Manual: included in CVLLIB |
| FORTRAN portion | CVFLIB | Manual: CVFLIB |

The author has provided excellent documentation with this library and it should be a valuable addition to every user's program collection.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskettes (JC)

Format: RT-11

SORT for RT-11, for the Professional-300 Series

Version: December 1983

Author: Chester Wilson and Darrell Whimp

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: FORTRAN IV, MACRO-11

SORT is a general purpose high speed RT-11 memory/disc sort/merge utility program, capable of coping with files as large as RT-11 can manage. Sorting may be ASCII or alphanumeric, and considerable trouble was taken by the original author to enhance the speed of the sorting.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT 11

INDEX: FORTRAN Cross-Referencer, for the Professional-300 Series

Version: V5.6, December 1983

Author: Michael Levine

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: MACRO-11

Memory Required: 24KW

INDEX is a cross-referencing program that does for FORTRAN what CREF does for MACRO. A source program run through INDEX will be checked for all of its variable name and label usage. The names and labels used in the program, the lines on which they were used, and how they were used. If needed, the variables from the specified programs can be saved along with those of other programs and later printed out as a super index giving variable names and the names of all the programs it was used in. Also included is the capability to exclude from the index listing all variables that appear only once in a program in a common block or type declaration but are not used elsewhere (or list only those if wanted). The user can also list only those variables that are global (defined in a common block) or those that are local.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT-11

TECO V36 for RT-11, for the Professional-300 Series

Version: December 1983

Author: Andrew Goldstein, Mark Barnhall and Ian Calhaem

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: MACRO

Memory Required: 16KW

TECO is a powerful text editor that runs under most RT-11 operating systems. It is suitable for editing almost any form of text file including programs, manuscripts, correspondence, and the like. TECO is a character oriented editor, and as such is free from many of the inconveniences associated with many line oriented editors. In addition, TECO has most of the facilties

found in a programming language, such as arithmetic loops, conditional execution, GOTO's etc., allowing the user to write editing programs to alphabetize lists, reformat tables, renumber statement labels, and much, much more.

This version of TECO includes support for PRO-300 Series terminals, which do not respond to the usual tests for VT100 type terminals in ANSII mode. It will therefore run under RT-11 version 5.1 or later.

Note: From RT-11 version 5.1 or later TECO is not supplied with RT-11 distribution. It can only be obtained through DECUS.

Restrictions· Although complete sources are not included .OBJ files are provided and the source of the terminal driver module, so the user can configure TECO for special use. Command files are provided to assemble both a background and a virtual (system job) version of TECO.

Associated Documentation· To obtain the TECO manual see DECUS No. 11-450.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format RT 11

RENUM: FORTRAN Renumbering Program, for the Professional-300 Series

Version: V4.01, December 1983

Author: Eric Morton

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: FORTRAN-IV-PLUS

RENUM is intended to provide two useful services for the FORTRAN programmer:

1. Replace the existing statement number in a FORTRAN program with a new (an equivalent) series of sequential statement numbers,

2. Produce a cross-referenced table of all variable names used in the source code showing the line numbers where all references to each variable name occur.

The line number used for cross-referencing agree with those on a compiler-generated listing. Either or both of these services, along with a listing of the source code, can be produced; the two functions are completely independent. Control is by means of terminal-entered command strings following the standard RT-11 rules. Input files (up to six per command line) cna be on any disk device, and output files can be directed to any device. When renumbering, a new disk file (by the original name) is created for the renumbered source code, and the original file is changed to .OLD. RENUM version 4 adds several new useful features including; the ability to specify the starting new statement numbers and the increment between new statement numbers; a reformatted source

listing; flagging the range of DO loops; and improved variable name scanning, especially in logical IF statements.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT-11

CVLLIB: General Purpose RT-11 Library

Version: V3A, July 1983

Author: Chester Wilson

Operating System: RT-11 V3 or later

Source Language: MACRO-11

Special Hardware Required: To use floating point routines you must have an EIS.

CVLLIB is a (personal) general purpose library for the RT-11 MACRO, FORTRAN or C programmer. Routines cover facilities such as reading and writing decimal (up to triple precision) and octal and hexidecimal (up to double precision) integers, money format (double or triple precision), dates and times, filenames and Radix50 formats. Numerous convenience routines are included. Multiple precision arithmetic routines are included, none of which requires an EIS or FIS. Real (floating point) routines are included, but these require either a KEV11 or a floating-point hardware unit (FPU).

This 3A comprises the MACRO, FORTRAN and Real number sections of CVLLIB. The C portion is to be released separately at a later date.

Changes and Improvements: Reorganized internally, bug fixes and manuals updated.

Restrictions: Interfaces for DECUS 'C' will be released later as a separate submission.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskettes (KD),
2400' Magtape (PA)

Format: RT-11

**Indirect Command File Processing for RT-11 V4.0**

**Version: February 1983**

**Author: Russell L. Morrison II, Plessey Peripheral Systems, Irvine, CA**

**Operating System: RT-11 V4.0**

**Other Software Required: RT-11 V4.0 Autopatch Revision D**

**Special Hardware Required: Line Time Clock**

This manual consists of a description of a patch to the RT-11 V4.Ø Indirect Command File Processor from Autopatch Revision D. Users may patch IND.SAV which is found in the RT-11 Autopatch Revision D in the manner described in the manual. IND.SAV provides RT-11 with RSX-like command files, having features such as parameter substitution, terminal input ("ASK"), limited math, and flow control (".GOTO" and ".GOSUB").

Note: The media contains a manual only.

Restrictions: The patch described in this manual is operating system dependent. It will only work with IND.SAV from RT-11 V4.Ø Autopatch Revision D. In order for the patch to be effective, RT-11 must be SYSGENed with timer support in the Single Job (SJ) monitor.

Media (Service Charge Code): Floppy Diskette (KA),
600' Magtape (MA)

Format: RT-11

**BRUCE: A Backup and Restore Utility with Consolidation and Enhancement**

**Version:** V01.1, October 1983

**Author:** Bruce D. Sidlinger, Alcor Inc., San Antonio, TX

**Operating System:** RT-11 V5.Ø or later

**Source Language:** RT-11 IND

**Other Software Required:** IND.SAV (included with RT-11 V5.Ø)

BRUCE, a "Backup and Restore Utility with Consolidation and Enhancement", is submitted as both a useful utility program and as a demonstration of what can be done with the INDirect command file processor included with RT-11 Version 5.

BRUCE copies all of the files from a disk onto another (scratch) disk or tape of equal or larger capacity. The files appear on the output device in EXT, FILNAM sorted order. If there were no errors, BRUCE then initializes the input disk and copies the files back. The result is a "squeezed" (Consolidated) disk with all of the .SYS files adjacent (hence the performance Enhancement), and with a "neat-looking" (unsorted) directory.

Restrictions: BRUCE cannot restructure the current system disk (SY:), but the Backup phase is still applicable. BRUCE also requires some space on SY: for its work files and the output volume must already be initialized.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
600' Magtape (MA)

Format: RT-11

FLECS: FORTRAN Language with Extended Control Structures, for the Professional - 300 Series

Version: V28.02, December 1983

Author: T. Beyer, L. Yarborough and Ian Calhaem

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1 (PRO support requires V5.1 or later)

Source Language: FLECS

Memory Required: 28KW

FLECS is an extension of the FORTRAN language which provides the control structures necessary to support recent concepts of structured programming. Currently implemented as a translator which converts FLECS programs to FORTRAN, the system is written in FLECS and is easily adaptable to new machines and systems. The entire system including source code and documentation has been placed in the public domain by the author. The purpose of making the system available is to convince as many members of the Fortran Community as possible that structured programming when properly supported by a language is quite natural and requires substantially less support than programming in standard Fortran.

This release supports the PRO-300 Series computers, but many restrictions make it difficult to compile two of the source files on a PRO. For this reason the distribution includes these .OBJ files as well as full source code.

Note: The source has been updated to reflect the FORTRAN 2.6 compiler.

Restrictions: The Professional-300 Series memory restricts the compiling of some modules. To avoid this situation .OBJ files have been included for two modules which otherwise give dynamic memory overflow.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT-11

Airplane Landing Simulation Game, for the Professional-300 Series

Version: December 1983

Author: Bill Green, Les Parent and Ian Calhaem

Submitted By: Ian Calhaem, Ph.D., Dept. of Scientific and Industrial Research, Wellington, New Zealand

Operating System: RT-11 V5.1

Source Language: FORTRAN-IV

Memory Required: 16KW

This program is an airplane landing simulation game. It provides a psuedo graphic display of an aircraft instrument panel with real time updates at one second intervals. The program simulates a real instrument landing approach from an altitude of 25000 feet to the runway, with instructions from ground radar control. Aircraft climbs, dives, and stalls are properly simulated. An off airport landing as well as go-around for a missed approach are both possible.

Source code is supplied for both VT100 compatible and VT52 compatible terminals, and command files are supplied to enable versions to be produced for background, foreground and system job.

Documentation on magnetic media.

Media (Service Charge Code): 5 1/4" Floppy Diskette (JA)

Format: RT-11

new
11-687

FORTRAN Callable Subroutines Package for Fast Continuous A/D on the MINC

Version: V1.0, November 1983

Submitted By: Digital Equipment Corporation

Operating System: RT-11 V5.0

Source Language: FORTRAN IV, MACRO-11

Memory Required: 280 (decimal) Words

Other Software Required: MACRO-11 Assembler, RT-11 O/S, FORTRAN IV compiler

Special Hardware Required: MNCAD-MINC A/D Module, MNCKW-MINC Clock Module

Assembly routines, ADCONT, WAITFD, and STOPIT, constitute a FORTRAN callable package capable of providing dedicated, continuous (buffer management with transfer to peripheral storage) analog-to-digital acquisition at rates two to eight times faster than the maximum rates provided by REAL-11 routines in a fraction of the memory space. The interface is also less complex and was modeled after MINC BASIC. Experienced users should be able to modify the sources for use with K and V series logic running under RT-11. A programmable clock and A/D modules are required.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA),
600' Magtape (MA)

Format: RT-11

new
11-688

Monitor Commands for Namelists Package

Version: V1.Ø, November 1983

Author: John Alexander, Shiva Associates, Sepulveda, CA

Operating System: RT-11

Source Language: TECO-11

Other Software Required: TECO-11

This is a group of keyboard monitor "Executives" that is intended to allow the user to utilize a "Namelist" file to perform keyboard monitor commands, and others on a group of modules. The "executives" call up TECO files to perform the work. The TECO executive gets the namelist of modules, or single files, and creates further com lines that operate on the specified files.

This can be very useful to the user that has 5 to 1Ø5 modules that make up a major program. A typical executive is one that will merge files into a single file to allow global edits. When finished the user may then utilize split .N to split them out again. To execute any of these "executives", type in "@" in front of the executive name and a carriage return. The executive's name will prompt the user for inputs.

For example: To create an alphabetized name list of files type in @NAM.N To see a directory of the available executives type in @D.N. To copy a group of modules in a namelist to a device type in @COPY.N, etc.

In general any executives that end in "E" are more general executives, e.g., COPYE.N allows the user to specify a different file type to copy than the module names have in the name list file. RENAME.N allows a source namelist and a destination namelist, etc.

Restrictions: Dependent upon RT-11 Executives that utilize TECO.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up (AA), Floppy Diskette (KA), 6ØØ' Magtape (MA)

Format: RT-11

new
11-69Ø

File Manipulation Commands

Version: V1.Ø, November 1983

Author: John Alexander, Shiva Associates, Sepulveda, CA

Operating System: RT-11

Source Language: TECO-11

Other Software Required: TECO-11

This is a group of file manipulation "Executives" that is intended to allow the user to make "global" changes to all lines of a file. The "executives" call up TECO files to perform the work. The TECO executive gets the name of modules and creates further com lines that operate on the specified files. This can be very useful to the user that desires to utilize the code that has already been written and write supporting documentation, or to manipulate data files, etc. Some examples of "operations" that can be performed are: 1) Number all lines of a file, 2) Move a set of columns to a new column location for all lines, 3) Pad out the end of line to a given column, 4) Cut excessively long lines to a given column, 5) Fix all lines (long or short) to a given column), 6) Eliminate all "tabs" from a file, 7) Replace spaces with tabs, where possible, 8) Shuffle two files together, 9) Combine columns of one file with columns of another file, 10) Strip comments out of code, and 11) Strip code out of comments, etc., etc. To execute any of these "executives", type in "@" in front of the exec name and a carriage return. The executive's will prompt the user for input.

Restrictions: Dependent upon RT-11 Executives that utilitze TECO.

Documentation on magnetic media.

Media (Service Charge Code): Write-Up (AA), Floppy Diskettes (KB), 600' Magtape (MA)

Format: RT-11

revision 11-494

DIBOL '83 Screen Handler Package

Version: V4.0, February 1984

Author: David L. Wyse, Projects Unlimited, Inc., Dayton, OH

Operating System: RSTS/E V8, RSX-11M-PLUS V2.1, Micro/RSX V1.0, RT-11 V5.0, CTS-300 V8.0, CTS-500 V5.0, VAX/VMS V3.2

Source Language: DIBOL

Other Software Required: DIBOL '83 Compiler

This is a Screen Handling package written in DIBOL '83 and is transportable across all DIBOL supported operating systems. The package consists of three DIBOL '83 subroutines: "DISPL", a screen display subroutine which allows full use of VT100 type advanced video, line and special character drawing features and will format numeric fields with decimal points; the "ACCPT" which accepts input from a VT100 type terminal including decimal point numeric fields, using the optimized IO features of DIBOL '83; and "CENT" which uses the DISPL subroutine to display a literal in the center of a line with the capability of using the advanced and special video features of VT100 type terminals.

Changes and Improvements: Subroutines have been updated to use DIBOL '83 features including improved screen IO and faster execution times.

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskette (KA) Format: RT-11,
600' Magtape (MA) Format: DOS-11

new
11-694

WORD: Document Spelling Checker/Corrector

Version: June 1983

Author: R. R. DiMarco, South East Old Electricity Board,
Brisbane, Australia

Operating System: RSTS/E, RT-11, TSX-PLUS

Source Language: MACRO-11

Memory Required: 28KB

Special Hardware Required: EIS

The WORD package consists of a 15000 word dictionary file and a number of simple programs that allow the dictionary to be used to in the correction of spelling errors in documents. The major components of the kit are:

WORDS.DIC...dictionary file
WORDS.SAV...flags possible spelling errors
WORDE.SAV...corrects spelling errors
SPELL.SAV...finds correct spelling from abbreviation
SOUND.SAV...finds correct spelling from sound

Documentation on magnetic media.

Media (Service Charge Code): Floppy Diskettes (KB),
600' Magtape (MA)

Format: RT-11

# Past Symposium Information

## HOW TO WRITE RT-11 DEVICE HANDLERS

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

ABSTRACT

Device handlers for RT-11 are easy to develop if the proper design methodology is followed. First the user must get to know the device and its associated registers. Next a wait loop routine should be written to check out the basic operation of the device. Then the dynamic characteristics should be verified with an interrupt service routine. Finally the full device handler can be developed based upon both the wait loop and interrupt service routines.

### 1. INTRODUCTION

In the real-time environment, the system is usually called upon to control external devices or to collect data from external devices. Given that a hardware interface exists between the computer and the external device, software is required to actually control and command the device.

The ultimate goal of this paper is to provide a method to follow to develop device drivers. The real key to their development is to break the problem (handler) down into small, easily understood steps and then to develop the handler in an incremental fashion. A possible method is suggested by the three types of software routines used to control external devices. If a wait loop routine is developed first, the structure of an interrupt service routine follows easily. Similarly, once the interrupt service routine has been developed, the formal device driver is easily implemented. The key is to experiment and to learn about the device using progressively more complex routines as the handler is developed.

Although the focus of this paper is on the ADV-11, analog-to-digital (A/D) converter for the QBUS, there is an equivalent A/D converter for the UNIBUS, and everything that is said about the QBUS A/D converter will pertain to the UNIBUS version. In fact, most devices that have QBUS and UNIBUS interfaces are software compatible. This means that the software developed for the QBUS version will run without change on a UNIBUS machine.

### 2. GENERAL DESCRIPTION OF DEVICE CONTROL ROUTINES

#### 2.1 Wait Loop Routines

The wait loop is the simplest of the software routines to program. The theory behind the wait loop is that the program will start an I/O operation and then sit in a tight loop that does nothing more than test the "done" bit on the device interface. The routine is called a wait loop by virtue of the fact that the computer is "waiting" for the external device or interface to complete the operation.

This type of routine wastes computer resources because the computer is tied up in the tight wait loop and is therefore unavailable to do other work. To solve this problem, wait loops or "do nothing" loops can be converted to "do something" loops. This is accomplished by testing the done bit as usual, but then, instead of looping back to test the bit again, a useful piece of code or subroutine call is performed next. Once that is completed, the bit test is again performed. While this technique allows the computer to do additional work while the I/O is continuing, it suffers from the fact that the device may finish its task and be ready to perform another while the computer is off in the "do something" loop. This could lead to lost or missed data. In time-critical applications this may be a problem.

#### 2.2 Interrupt Service Routines

An additional level of sophistication is added when the wait loop routine is converted to an interrupt service routine. Here, the I/O is initiated as in the wait loop, and then the interrupt is enabled on the device interface card. Now, when the device has completed the I/O, the setting of the done bit will "vector" the computer to an interrupt service routine. The computer is now free to do other tasks while the external device is performing the I/O; the computer will be interrupted after the I/O operation is completed.

This additional functionality is not added without cost. This type of routine must properly set up the interrupt vector and ensure that the interrupt enable bit is set so that an interrupt will be generated. Failure to properly set up the interrupt vector may lead to a system crash when the interrupt is generated because the vector does not point to the interrupt services routine.

## 2.3 Device Driver Routine

There are potential problems with the interrupt service routine approach that can be solved by rewriting the routine in the framework of a device handler. First of all, the interrupt service routine would have to be linked with every routine that wants to use the device, and by nature of the fact that only one interrupt service routine can be pointed to by the interrupt vector, simultaneous access to a device would be prohibited.

In a device handler, the mechanics of the interrupt service routine are formalized. Usually the handler framework is supplied as a part of the operating system, and the user merely adapts his interrupt service routine to this framework. With a device driver, access to devices can be shared, and a copy of the routine does not have to be included with each routine that wants to use the device; it resides in system space, available for use by any program.

# 3. ADV-11 A/D CONVERTER

Before any code can be written to control an external device, you have to take some time to get to know the device. In order to program the device, you have to know how many registers the device uses and how each bit in each of the registers is used.

The ADV-11 is a 16-channel analog-to-digital converter that samples analog data at specific user-defined rates, and stores the 12-bit digital equivalent for further processing. A/D conversions can be initiated by program control, clock overflow, or external events. For the purposes of this paper, we will examine how to acquire data under program control.

## 3.1 Device Registers

Two registers on the ADV-11 are used to command and control the A/D converter. The CSR or command and control register is used to select the desired channel for data conversion, to enable interrupts and to start data conversion. The other register is the data buffer register, and it holds the result of the A/D conversion.

3.1.1 Command and Control Register. Figure 1 shows the format of the command and control register. Bit 15, the high order bit of the CSR is the error bit. It is set whenever an error condition is detected by the device. Bit 14 is the interrupt enable bit for the error conditions. If bit 14 is set and an error condition is detected that will also set bit 15, then an interrupt will be generated. For the purposes of this paper, the error bits (14 and 15) will be ignored.

Bits 12 and 13 are not used while bits 11-8 will contain the address of the analog channel that is currently addressed for data conversion.

Bits 1-5 are special bits that control how the device is commanded externally and in the maintenance mode. Again, for the purposes of this paper, these bits will be ignored.

Bits 0, 6, and 7 usually have the same function on every QBUS or UNIBUS device. Bit 0 is usually considered the "GO" bit. Once all the registers in the device are set up, the I/O transfer is usually started by setting the "GO" bit. This is true with the ADV-11. Once the channel address is loaded in the CSR, setting the "GO" bit will initiate the A/D conversion for that channel.

Bit 7 is usually the "DONE" bit. It indicates that the I/O transfer has completed. Setting bit 6 will cause an interrupt to be generated when the "DONE" bit (bit 7) is set; no interrupt will be generated unless bits 6 and 7 are set.

3.1.2 Data Buffer Register. The other register of importance on the ADV-11 is the data buffer register and it is detailed in Figure 2. Unlike the CSR, the data buffer register has different functions depending upon whether the computer is reading or writing to the register. When the data register is read, bits 0-11 will contain the digital representation of the analog voltage. When data is written into the register, the offset value of the analog data can be changed. In the read mode, bits 13-15 are unused and bit 12 will be set at the end of a data conversion if bit 3 of the CSR is also set.

As mentioned above, the converted data occupies only the 12 bits that correspond to decimal values of 0-4095. Negative values are in the range of 0-2047 and positive values are in the range of 2048 to 4095. In order to convert the digital values to the proper 16-bit two's complement values, 2048 must be subtracted from the 12-bit quantities. This data conversion can take place either in the software routine that acquires the data, or in the analysis routine. The software routines that will be presented in this paper will convert all the data to the proper two's complement values.

## 3.2 Device Bus Address

On PDP-11's and LSI-11's, the upper 4K words of address space are reserved for device addresses. When external devices are installed in QBUS or UNIBUS systems, each device must have a unique address on the bus so that the software routines can address that device. The net result is that the device registers look like memory addresses to the software routines and no special I/O instructions are required of the computer.

Many devices have "standard" addresses that have been established by the manufacturer. The use of these standard addresses for certain types of devices means that the software need not be reconfigured every time a new device is added to the system because the software can expect the device to answer to a fixed address.

The standard address for the ADV-11 is 770400 (octal). This address is set on the card by a row of DIP switches and the address corresponds to the address of the CSR register. The address of the data buffer register follows the CSR and is 770402. All of the example programs that follow will use these standard addresses.

# COMMAND AND CONTROL REGISTER

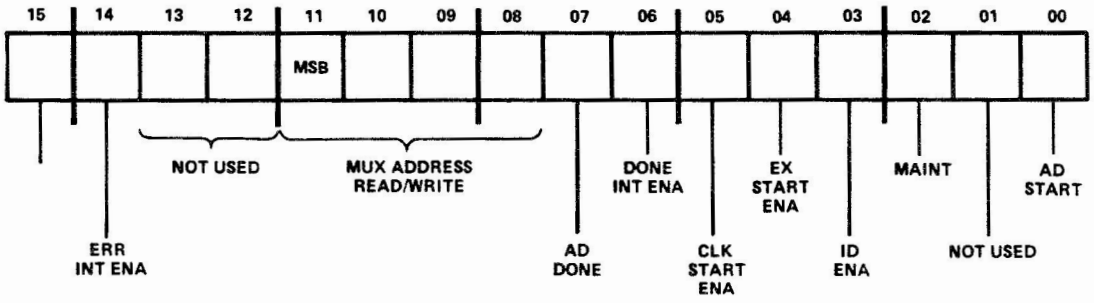



Figure 1

# DATA BUFFER REGISTER

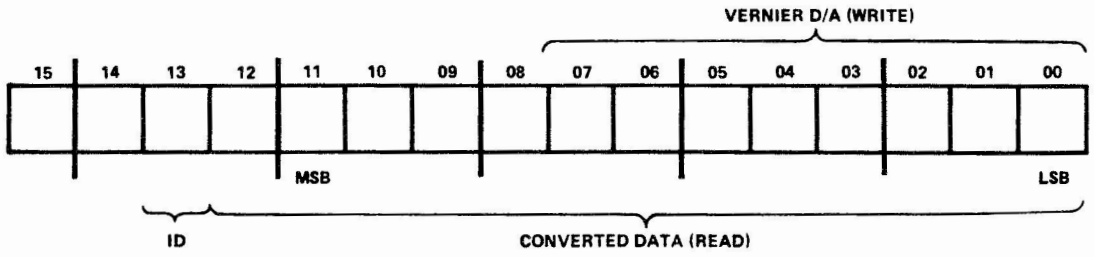



Figure 2

### 3.3 Device Vector Address

Besides the bus address of the interface card, the other item that has to be set is the device interrupt vector. The interrupt vector is important when the device will be programmed to use interrupts. When a device requests an interrupt, the CPU will save the current processor state on the system stack and "vector" or begin executing an interrupt service routine whose address is stored at the vector address.

On PDP-11 systems, vector addresses are contained in memory from about 60-500 (octal). Vector addresses contain two pieces of information and occupy two words. The first word of the interrupt vector contains the address of the interrupt service routine while the second word contains the processor status word (PSW) for that interrupt service routine. The major field of interest in the PSW is the processor priority bits. These bits will determine the executing priority of the interrupt service routine.

When an interrupt is taken, the CPU stores the current processor state which consists of the current program counter and the current processor status word. Then it loads the program counter with the address contained in the first word of the interrupt vector and the PSW from the second word of the vector. The loading of the program counter from the interrupt vector is equivalent to jumping to that address, which is the address of the interrupt service routine. The RTI instruction (ReTurn from Interrupt) restores or reloads the old PSW and program counter from the stack which resumes the execution of the program where it was interrupted.

### 3.4 Device Bus Grant Level

The other setable parameter on a device interface card is the bus grant level. The bus grant level is concerned with the order in which interrupts are serviced in the computer. For the purposes of this paper, only the bus grant level is of interest and does not affect how the software routines are developed. Additional information about the bus grant level is contained in literature that describes the theory and operation of the QBUS and UNIBUS.

Very simply, there are four bus grant levels available and the bus arbitration logic on the PDP-11 will only grant an interrupt to a device whose bus grant level is greater than the current execution priority of the CPU. For example, the CPU normally operates at priority 0. If an interface requests an interrupt and its bus grant level is 5, then the interrupt will be granted and an interrupt service routine will begin execution. The priority of the interrupt service routine is contained in the second word of the interrupt vector and is set when the interrupt service routine is entered. If the interrupt service routine is operating at priority 5 and another device at bus grant level 5 requests an interrupt, the request is queued and held off until the processor priority falls below 5. This lowering of the priority can occur under software control, or by the RTI instruction at the end of the interrupt service routine.

## 4. DESIGN METHOD FOR DEVELOPING DEVICE DRIVERS

Now, this paper will address how device drivers can be easily developed. As was suggested before, the easiest path to a device driver involves writing two routines before the actual device driver itself. First a wait loop routine should be designed and tested. Then, based upon what is learned from that routine, an interrupt service routine should next be developed. Finally, the interrupt service routine should be formalized and incorporated into a full device driver. It has been my experience that, if this development path is followed, the device driver is easy to develop and its design will follow logically from the wait loop and interrupt service routines.

The examples that follow all implement a way to acquire data from the ADV-11. In each example, two pieces of information are required from the user. First of all, since the A/D converter can convert up to 16 channels of information, the user must provide the number of channels that must be converted. In order to make things easier, all the routines will assume that the channels are connected in sequential order so that the individual channel numbers will not be required. That means that if three channels are requested, then the routine knows that channels 0, 1 and 2 are being requested. Although random access of the individual channels is supported with this device, the examples will not implement that particular feature.

## 5. WAIT LOOP

Listing 1 shows an implementation of a software routine that collects data from the A/D converter using the wait loop approach. Because the routine is FORTRAN callable, arguments are passed to the routine using the standard parameter block (Figure 3). In the parameter block, the first parameter contains the number of parameters being passed, followed by the addresses of the parameters. Register 5 points to this parameter block upon entry to the routine.

The first few instructions merely set up the parameters for the routine. The number of channels to acquire is stored in register 1 and the address of the buffer to receive the converted values is stored in register 2. The next instruction clears the A/D CSR register which has the effect of setting the channel address (bits 8-11) to zero. The instruction that increments the CSR register will set the "GO" bit and will begin the conversion for channel 0.

The wait loop itself consists of the two instructions at the label "LOOP:". Here the routine does a byte test on the lower half (bits 0-7) of the CSR register. When the conversion is done, bit 7 will be set. If we consider only the lower eight bits of the CSR as a two's complement number, bit 7 corresponds to the sign bit. Therefore, the branch positive (BPL) will be taken if bit 7 is not set or the conversion is not

ready. Note that a bit test instruction could have been used to explictly test for the done bit instead of the test byte instruction.

The next few instructions read in the data from the A/D converter and start the next conversion. First, the number of channels to acquire (R1) is decremented. If it is 0, then the desired number of conversions has been performed and the branch is taken to "DONE:", where the data is read from the data register and stored in the user's buffer. If there are more channels to acquire, the channel address is incremented by one with the INCB instruction. This instruction affects the top byte of the CSR which will increment the channel address. Next the converted data is read from the data register and stored in the user's buffer. The number 2048 is subtracted from the data to put it in the proper two's complement form and the A/D converter is started again by the INC instruction which sets the "GO" bit. Now the routine returns to the wait loop to wait for the conversion to finish.

The routine exits when the last channel is converted and the data is stored in the user buffer. This type of routine is an excellent way to start handler development and it is easy to debug such an application because a debugger can be used in the wait loops and print statements can be used to follow the program execution. Because interrupts are not used in this routine, any programming errors that are encountered are usually non-fatal.

## 6. INTERRUPT SERVICE ROUTINE

Once the basic operation of the device is established through the use of the wait loop routine, the interrupt service routine approach can be addressed. In the interrupt service routine, the basic structure of the wait loop will be changed so that the software can take advantage of the fact that the device can generate interrupts.

Listing 2 shows the form of the interrupt service routine for the ADV-11. The interrupt service routine usually contains three separate parts. The first part, which starts at "INIT:", sets up the interrupt vector and the PSW for the interrupt service routine. Note that once the device generates an interrupt, the routine "ADISR" will be entered and the priority of the the interrupt service routine will be 7 as indicated by the 340 (octal) in the interrupt vector. In addition to setting up the interrupt vector, the A/D status register is cleared to halt any operations that may have been started earlier.

With an interrupt service routine such as this one, the basic philosophy is to initiate the I/O operation and then perform another task while the I/O is being performed. The problem that the software designer then needs to address is how to determine when the I/O operation is complete so that other I/O operations can be started or that analysis of the acquired data can begin. If the device generates an interrupt at the completion of all the I/O, then that interrupt can be used. If, on the other hand, an interrupt is generated at the end of each word of the I/O transfer, another method must be used to indicate that the entire operation is completed.

Many routines use a flag or semaphore to indicate that an operation is totally complete. In that case, the I/O operation can be started, and when the flag is set, the main calling routine knows that all the I/O has finished.

The second portion of the interrupt service routine is the initiation section which starts at "START:" in Listing 2. The function of the initiation section is to initially start the I/O and clear the user's done flag. In this case, once the I/O is started, the interrupt service routine portion of the routine will continue to acquire data until the requested number of channels have been read.

The initiation section begins by storing the number of channels to convert in the memory location "LENGTH". Next the data buffer address and the address of the done flag are stored in other memory locations. Memory locations have to be used in this case, because the contents of the general purpose registers will not be known when the interrupt portion of the routine is entered. The user's flag is then cleared to indicate that the transfer is not completed and then the A/D converter is started.

The way in which the A/D converter is started deserves further attention. Because the CSR register was initially cleared in the initiation section, the channel address was also set to zero. In order to begin the I/O transfer, the software routine must set the "GO" bit. In order to allow the computer to respond to the interrupt, the "interrupt enable" bit on the interface card must also be set. The interrupt enable bit is usually bit 6 and so loading a 101 (octal) into the A/D CSR register performs the dual function of setting the "GO" bit and enabling the device interrupt.

Now, all the rest of the work will be performed by the interrupt service portion of the routine. The interrupt service portion begins at label "ADISR:". The code in the interrupt service routine looks very similar to the code in the wait loop routine except that memory locations are used instead of general purpose registers. First, the number of channels to acquire is decremented. If the last channel was converted, the routine jumps to the completion section at label "DONE:". If this was not the last channel, then the channel address in the CSR is incremented by one, the data is moved from the A/D data buffer register and stored in the user's buffer using an indirect store instruction (the memory location "BUFFER" points to the user's buffer). Now, the next A/D conversion is started again by setting the "GO" bit on the interface card. There is no need to set the interrupt enable bit again as it is already set from the initiation section.

A few clean-up items must be performed at this point. First, the converted data must be unbiased and put into the proper 16-bit two's complement form by subtracting 2048. Finally the buffer

# FORTRAN PARAMETER BLOCK

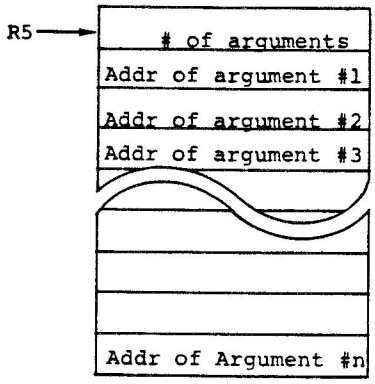



Figure 3

# QUEUE ELEMENT

| NAME | OFFSET | CONTENTS |
|---|---|---|
| Q.Link | Ø | Link to next Queue element |
| Q.CSW | 2 | Pointer to channel status word |
| Q.BLKN | 4 | Block number |
| Q.FUNC<br>Q.UNIT<br>Q.JNUM | 6 | reserved (1 bit) \| Job number (4 bits) \| UNIT number (3 bits) \| Special function code 8 bits |
| Q.BUFF | 10 | User buffer address |
| Q.WLNT | 12 | Word count |
| Q.COMP | 14 | completion routine code<br>Ø = wait<br>1 = asynchronous<br>even = completion routine addr |
| Q.PAR | 16 | Programmable address reg. 1 value Par 1 |
| | 20 | RESERVED |
| | 22 | RESERVED |

Figure 4

address stored in "BUFFER" must be adjusted to point to the next memory location in the user's buffer. Because the PDP-11 is byte addressable, two must be added instead of one. The interrupt service routine now completes with the RTI instruction and the computer resumes execution where it was interrupted.

The completion section of the interrupt service routine reads in the last channel converted and subtracts 2048 to unbias the data. Now, the user's flag is set to indicate that the I/O is completed, the CSR is cleared to reset the interrupt enable bit and to reset the channel address to 0, and the interrupt service routine is exited with the RTI instruction.

Debugging this type of routine is significantly harder than the wait loop routine because most debuggers will not work within an interrupt service routine and the routine cannot use RT-11 system services within an interrupt service routine without performing some synchronization with the operating system. This routine will allow the I/O to continue while other calculations are being performed. The calling routine is free to sit in a tight loop waiting for the flag to be set, or to do other work and check the done flag periodically.

## 7. DEVICE DRIVER

In order to incorporate the interrupt service routine into the operating system, it has to be formalized and placed in the framework called the device driver. In general, device drivers have three parts. The first is the initiation section where the I/O is started. The second section is the interrupt service portion or the continuation section. Finally, when the I/O request is completed, the completion section is used. Listing 3 contains an implementation of a handler to acquire data from the ADV-11. Besides the three sections mentioned, this paper will discuss the macros that RT-11 provides to help make handler development easier and the data structure known as the queue element. Further explanation of these macros and the structure of device drivers can be found in the Software Support Manual for the RT-11 Operating System.

### 7.1 RT-11 Queue Element

One of the reasons to develop an operating system is to adopt standard conventions in the area of system communications and input and output to external devices. The queue element is the data structure used by RT-11 to initiate I/O operations. Its structure is shown in Figure 4. I/O works as follows under RT-11. First a program decides that it will perform I/O to a particular device. Using RT-11 system calls, the routine will associate a channel number with a particular device. From then on, all I/O to a device is referred to by channel number. Now, the software routine invokes an RT-11 macro to perform the I/O. In order to service the request, the system "buys" a queue element from the list of available queue elements and transfers the parameters of the I/O operation to the queue element. Next, the queue element is placed in a queue for the particular device handler. If the handler is currently busy with another I/O request, the queue element remains on the queue and will be processed when the previous I/O has completed. If the handler is not busy, the operating system calls the handler at the initiation section. Note that the handler can always assume that the current queue element is the first one on the queue for the handler.

### 7.2 RT-11 Device Macros

In what is known as the preamble section of the handler, various options and constants are set up. RT-11 provides a set of macros that define the fields in the queue element (.QELDF) and sets up the various communications paths to the operating system (.DRBEG, .DRAST, .DRFIN, .DREND, .FORK). This section also contains conditional assembly parameters for the various options supported by the operating system such as Extended Memory, Error Logging and Timeout support. In addition, various status words and constants can be set up in the preamble section. These types of parameters and constants are well documented in the Software Support Manual for the RT-11 Operating System and will not be repeated here as they do not help considerably in understanding how device drivers operate.

### 7.3 Initiation Section

This section of the device driver initiates or starts the I/O operation and is called the "header" section in RT-11. The macro .DRBEG sets up the entry point to the initiation section and provides information about the device registers, name, size and status.

Upon entry to the initiation section, the device must validate the I/O request. For example, write-only devices will not perform read requests and read-only devices will not perform write requests. This validation of the type of request is performed by examining the word count field of the queue element. Normally, the address of the queue element is held in R4 and all other references to the queue element are made as symbolic offsets to the address in R4.

By convention, a word count of 0 is a seek request and may be used to position a mechanical device to a particular area. In the case of this device, a seek has no meaning and the I/O request can be completed right away. A write request is signified by a negative word count and since this is a read-only device, write requests will abort and return an error code. Read requests have a positive word count field in the queue element.

In order to start the I/O, the handler merely needs to set the channel address to zero, set the "GO" bit and enable the interrupt. The operating system is responsible for loading the interrupt vector when the handler is loaded. The MOV instruction loads a one into the CSR register, sets the "GO" bit and clears out the channel address bits at the same time. The interrupt is enabled with the Bit Set (BIS) instruction. The initiation section is now finished and the RETURN instruction will return

control to the operating system while the I/O proceeds.

### 7.4 Interrupt Section

The .DRAST macro sets up the entry to the continuation section of the handler. The other two parameters to the macro set the priority of the interrupt service routine, while the last parameter is the address of the abort I/O routine. The abort I/O routine will be entered whenever a job is aborted by either a control-C or error. Its purpose is to allow the handler to stop any I/O that is in progress in a controlled manner.

The first thing that the continuation section must do is point to the queue element as all the information about the I/O in progress is stored there. Now, the routine is ready to read the data from the device and store it in the user's buffer. The address of the user's buffer is stored in the queue element, but this address is different depending upon the RT-11 monitor being used. In the unmapped environment (SJ and FB) the address in the queue element is the direct address of the user's buffer. In the mapped environment (XM) the user's address consists of page references and offsets within the page. In order to make it easier to store and retrieve information, RT-11 provides hooks into the operating system for the computation of the proper physical address from the user's virtual address for the mapped environment.

In this case, the address stored at offset "BUFF" of the queue element contains the proper address in the unmapped environment. In that case, the data need only be read from the data register and stored directly in the user's buffer. Then the data can be unbiased and the buffer address in the queue element can be incremented to point to the next item in the buffer. In the case of the mapped environment, the monitor routine "$PTWRD" will take the top item from the stack, store it in the user's buffer, and increment the address in the queue element. The routine assumes that R4 points to the queue element and that the data to be stored is on the top of the stack.

The saving of the data in the mapped environment is accomplished by reading the data register and storing the data in R0 (R0 and R4 are available for use in interrupt service routines). The data is then unbiased and stored on the stack. The routine "$PTWRD" is then called to actually move the data to the user's buffer.

The final task that the continuation section must perform is either to end the data transfer if all the data has been transferred, or start the next transfer if it has not. As with the other routines mentioned above, the word count is decremented. If it is zero then the transfer is complete and the routine branches to the completion section. If the transfer is not complete, the channel address is incremented by one and the "GO" bit is set by adding 401 (octal) to the contents of the CSR. Adding 400 (octal) will increment the top byte of the CSR register which will increment the channel address and adding 1 will set the "GO" bit in the lower byte of the CSR. The routine can now exit the continuation section and wait for the next data conversion to complete.

### 7.5 Completion Section

The I/O completion, as the name implies, is entered at the completion of the I/O. In this section, the routine must leave the device in the proper termination condition. In the case of the ADV-11, all that is required is to disable the interrupt and then exit. In the case of other devices, additional tasks may be required. The macro .DRFIN performs the task of placing the current queue element on the monitor's completion queue and restarting the handler if there is another queue element on the handler's queue. Once on the completion queue, the monitor will take the proper action depending on whether the I/O was one with wait, no wait or completion routine requested.

### 7.6 Abort Section

The ".DRAST" macro has a parameter that indicates the address of an abort routine if one is wanted. In the case of this handler, the abort section of the handler is the same as the completion section. If the I/O is aborted, all that is required is to disable the interrupt and return--the data will not be used in this case.

## 8. CONCLUSIONS

This paper has presented a method of design to follow in order to simplify the development of device handlers for RT-11. In order to develop device handlers, the user should first get to know the device. That implies that he should study each of the registers used to control a device, and the purpose of each of the bits in the registers. Next, armed with an understanding of how the device works, the user should write a wait loop routine to verify the user's understanding of the device. Once the wait loop routine is fully operational and debugged, an interrupt service routine should be the next step. The interrupt service routine will give the user confidence that he truly understands how the device will operate under interrupt conditions.

Once the interrupt service routine is completed, the full device handler can easily be developed. The handler normally consists of the preamble portion that sets up the handler data structures, the initiation section that starts the I/O operation, the continuation section that reinitiates the I/O until the requested number of words have been transferred, the completion section that terminates the I/O, and the abort section that handles abnormal termination of the I/O operation.

During development of the device handler, there are various timing considerations to think about. The wait loop, although the most wasteful of computer resources, will respond the fastest to changes in the device. This is due to the fact that the computer is constantly monitoring the device in a tight loop. The interrupt service routine will give good performance, but it has a property known as interrupt latency. The latency is the time required for the computer to detect the interrupt,

save its current status, and vector to the interrupt service routine. In time-critical applications, this time period may be too long.

The handler, while the most general of the routines, is probably the slowest to respond. In addition to the interrupt latency, additional time may be spent in the operating system forwarding the request to the handler itself. The net result is that one should consider all the advantages and disadvantages of the various software methods as they apply to your application before choosing the particular implementation necessary to solve your particular problem.

LISTING 1

```
        .TITLE  SCAN
;
;
;       This routine implements data collection from an ADV-11
;       A/D converter using a wait loop technique
;
;
        ADSTAT=170400           ;CSR REGISTER
        ADBUFF=ADSTAT+2         ;DATA REGISTER
;
;
SCAN::  MOV     (R5)+,R0        ;PASS # PARAMETERS
        MOV     (R5)+,R1        ;ADDRESS OF LENGTH OF SCAN
        MOV     (R1),R1         ;LENGTH OF SCAN
        MOV     (R5)+,R2        ;ADDRESS OF BUFFER
        CLR     @#ADSTAT        ;CLEAR A/D STATUS REG
        INC     @#ADSTAT        ;START A/D
LOOP:   TSTB    @#ADSTAT        ;DONE?
        BPL     LOOP            ;BRANCH BACK IF NOT DONE
        DEC     R1              ;IS THE SCAN DONE?
        BEQ     DONE            ;BRANCH TO DONE IF YES.
        INCB    @#ADSTAT+1      ;INCREMENT MUX NUMBER
        MOV     @#ADBUFF,(R2)   ;MOVE VALUE FROM A/D BUFF TO DATA BUFF
        SUB     #2048.,(R2)+    ;UNBIAS THE DATA
        INC     @#ADSTAT        ;START THE A/D AGAIN
        BR      LOOP            ;GET MORE DATA
DONE:   MOV     @#ADBUFF,(R2)   ;READ THE DATA
        SUB     #2048.,(R2)+    ;UNBIAS THE DATA
        RTS     PC              ;RETURN
        .END
```

LISTING 2

```
        .TITLE  SCAN -- Scan the A/D converter once with interrupts
;
;
;       This routine implements data collection from an ADV-11
;       A/D converter using an interrupt service routine.
;
;
        ADSTAT=170400           ;STATUS REGISTER
        ADBUFF=ADSTAT+2         ;DATA REGISTER
        ADVECT=340              ;INTERRUPT VECTOR
;
;
;       Initialization routine -- Set up the interrupt vector
;
;
INIT::  MOV     #ADISR,@#ADVECT ;INTERRUPT SERVICE ROUTINE
        MOV     #340,@#ADVECT+2 ;PRIORITY 7
        CLR     @#ADSTAT        ;START OUT AT A ZERO
        RETURN
;
;
;       Start the conversion, flag will be set nonzero
;       when it completes
;
;
START:: MOV     @2(R5),LENGTH   ;LENGTH OF SCAN
        MOV     4(R5),BUFFER    ;ADDRESS OF BUFFER
        MOV     6(R5),FLAG      ;ADDRESS OF DONE FLAG
        CLR     @FLAG           ;CLEAR IT
        MOV     #101,@#ADSTAT   ;ENABLE INTERRUPTS AND START IT OFF
        RETURN                  ;AND RETURN TO CALLER
;
;
;       A/D interrupt service routine
;
```

```
ADISR:: DEC     LENGTH          ;IS THE SCAN DONE?
        BEQ     DONE            ;BRANCH TO DONE IF YES.
        INCB    @#ADSTAT+1      ;INCREMENT MUX NUMBER
        MOV     @#ADBUFF,@BUFFER;MOVE VALUE FROM A/D BUFF TO DATA BUFF
        INC     @#ADSTAT        ;START THE A/D AGAIN
        SUB     #2048.,@BUFFER  ;UNBIAS THE DATA
        ADD     #2,BUFFER       ;BUMP THE ADDRESS
        RTI                     ;INTERRUPT RETURN
;
;
;       Get the last channel and set the flag
;
;
DONE:   MOV     @#ADBUFF,@BUFFER;READ THE DATA
        SUB     #2048.,@BUFFER  ;UNBIAS THE DATA
        INC     @FLAG           ;SET THE FLAG
        CLR     @#ADSTAT        ;DISABLE INTERRUPT AND CLEAR CHANNEL
        RTI                     ;INTERRUPT RETURN
;
;
LENGTH:         .WORD 0         ;SCAN LENGTH
BUFFER:         .WORD 0         ;BUFFER ADDRESS
FLAG:           .WORD 0         ;SCAN COMPLETE FLAG
        .END
```

LISTING 3

```
        .TITLE  AD-11 DRIVER
;
;
;       This handler will scan the A/D once.
;       The length of the scan is determined by the number
;       of words requested in the I/O queue element.
;
;
        .IDENT /V04.0/
;
;
        .SBTTL PREAMBLE SECTION
;
;
        .MCALL  .QELDF,.DRBEG,.DRAST,.DRFIN,.DREND,.FORK
;
;
;       SYSGEN DEFAULT DEFINITIONS
;
;
        MMG$T   = 1
        .IIF    NDF,MMG$T       MMG$T = 0
        .IIF    NDF,ERL$T       ERL$G = 0
        .IIF    NDF,TIM$IT      TIM$IT = 0
;
;
;       DEVICE UNIBUS ADDRESSES
;
;
        .IIF    NDF,AD$VEC      AD$VEC=340      ;A/D VECTOR
        .IIF    NDF,AD$CSR      AD$CSR=170400   ;A/D CSR

        ADBUFF = AD$CSR + 2                     ;BUFFER REGISTER
        HDERR = 1                               ;HARD ERROR ON CHANNEL
;
;
;       DEVICE STATUS INFORMATION
;
;
        ADDSIZ = 0                              ;DEVICE BLOCK SIZE
        ADSTS = 40370                           ;DEVICE STATUS WORD
;
;
;       DEFINITION OF Q ELEMENT SYMBOLICS
;
;
        .QELDF
        WCNT = Q.WCNT - Q.BLKN
        BUFF = Q.BUFF - Q.BLKN
;
;
        .SBTTL  SET OPTIONS
;
;
;       NO SET OPTIONS
```

LISTING 3 (Continued)

```
;
;
        .SBTTL  HEADER SECTION
;
;
        .DRBEG  AD,AD$VEC,ADDSIZ,ADSTS
;
;
;       ENTRY POINT FROM QUEUE MANAGER
;
;
        MOV     ADCQE,R4                ;POINT TO CURRENT QUEUE ELEMENT
        TST     WCNT(R4)                ;WHAT DO YOU WANT??
        BEQ     ADDONE                  ;SEEK COMPLETES RIGHT AWAY
        BMI     ADERR                   ;WE DON'T DO WRITES
;
;
;       ASSUME A READ
;
;
        MOV     @#ADBUFF,R0             ;CLEAR OUT A/D FLAG
        MOV     #1,@#AD$CSR             ;START A/D AT CHANNEL 0
RET:    BIS     #100,@#AD$CSR           ;ENABLE INTERRUPT
        RETURN                          ;BYE
;
;
        .SBTTL  INTERRUPT SERVICING
;
;
        .DRAST  AD,4,ADDONE
        MOV     ADCQE,R4                ;POINT TO ELEMENT
        .IF     EQ,MMG$T
        MOV     @#ADBUFF,@BUFF(R4)      ;READ THE WORD
        SUB     #2048.,@BUFF(R4)        ;UNBIAS THE DATA
        ADD     #2,BUFF(R4)             ;BUMP BUFFER ADDRESS
        .IFF
        MOV     @#ADBUFF,R0             ;READ THE WORD
        SUB     #2048.,R0               ;UNBIAS THE DATA
        MOV     R0,-(SP)                ;STACK IT
        CALL    @$PTWRD                 ;GIVE TO THE USER
        .ENDC
        DEC     WCNT(R4)                ;ONE LESS WORD
        BEQ     ADDONE                  ;WE ARE DONE
        ADD     #401,@#AD$CSR           ;START NEXT CONVERSION
        RETURN                          ;RETURN
;
;
        .SBTTL  I/O COMPLETION SECTION
;
;
ADERR:  BIS     #HDERR,@Q.CSW-Q.BLKN(R4)
                                        ;SET ERROR BIT IN CHANNEL
ADDONE: BIC     #100,@#AD$CSR           ;DISABLE INTERRUPT
        .DRFIN  AD
        .DREND  AD
        .END
```