RSX

MULTI-TASKER

October 1984 Issue

Printed in the U.S.A.

RSX MULTI TASKER

Table of Contents

ONLY A FEW WEEKS TO VOTE ON THE RSX-11M SIG FALL 1984 MENU

The RSX-11M Special Interest Group conducts a yearly balloting to determine the most important issues facing the users of Digital Equipment Corporation RSX-11M operating systems. It is now time for the 1984 Menu voting. Attached is the list of menu items and the menu ballot to be returned. Due to delays in preparing the menu for publication you have only a short time to read the menu and return the ballot. In order to have the results for Anaheim, we must ask you to return the ballot no later than November 9, 1984.

This menu and ballot is one step in a continuing process. At past DECUS symposia, the results of previous menus were discussed and Digital responded to the top items. The complete menu results and the Digital response has been published in the Multi-Tasker.

SIG members are periodically invited to input their suggestions for the RSX menu, both at the Symposium and through the Multi-Tasker.

The results of the votes received from this ballot will be compiled and announced at the Anaheim symposium in December, 1984, and the next menu will begin. A menu submission form for 1985 will appear in an upcoming issue of the Multi-Tasker.

The menu voting is open to all RSX sites. Each site is allowed to cast one ballot for each RSX-11D, RSX-11M, RSX-11M-Plus, and RSX-11S operating system license it has purchased. This will typically mean one ballot for each RSX system. If a site is eligible to cast multiple ballots, please make a copy of the ballot and use one ballot for each system. If there are multiple users on a system, please meet together and return a consolidated ballot.

The menu items below have been accumulated by Alan Bennett of Clark Systems for the RSX SIG.

The menu ballot has had a very favorabvle impact on Digital in the past, and a large number of the items on past ballots have been implemented. In order for this process to be effective in the future, we must continue to receive the large number of responses that we have received in the past. We must also continue to take this process seriously. This menu will be as effective as you make it.

Tabulating the menu is a large task for our volunteers, so please fill out the ballot very carefully. Please avoid some of the common mistakes made in the past:

* Make sure you submit separate ballots for each system you have. In the past, we know many sites have submitted only one ballot when they have had ten systems.

* Please use a separate ballot for each system. We cannot handle a single ballot that represents more than one system.

If the menu is to have any effect on Digital, a substantial number of the RSX systems installed must respond. The RSX menu is the primary method for users to supply input to Digital on their needs for the future. Please take the time to vote now. If you have any comments on the menu items or the menu process, please include them on a second sheet.

HELP REACH ALL USERS

This menu is included only in the Multi-Tasker. Not all RSX sites receive the Multi-Tasker (alas!), so if you know of sites in your organization or your area that do not receive it, please forward a copy of the menu and ballot to them. Your help in this will insure the menu reaches the majority of RSX sites in the world.

Working Group News

Jeff Hamilton
Working Group Coordinator
(214)457-4175

Date of this report: 14SEP84

The working group chairmen are as follows:

RSX-11M Unsupported Versions:
Bill Burton
Texas Research Institute
1300 Moursand
Houston, Texas 77030

System Performance and Accounting
Roy S. Maull
U. S. Air Force

HQ SAC / AD1AE
Offutt AFB, Ne 68113

DECUS Library
Bruce Zielinski
RCA
Marne Highway M/S 138-2
Moorestown, N. J. 08057

SIG Tape Collection
Glenn Everhart
RCA Government Systems Division
Route 38
Cherry Hill, New Jersey 08358

SRD
Bob Turkelson
NASA/Goddard Space Flight Center
Mail Code 614
Greenbelt, Maryland 20771

RSX Realtime
Ed Cetron
University of Utah
Center for Biomedical Engineering
3168 Merrill Engineering Building
Salt Lake CIty, Utah 84112

Runoff
Chuck Spalding
Adept Technology Inc.
1202 Charleston Rd.
Mountain View, Calif. 94043

Cheap Networks
Evan Kudlajev
Philadelphia Electric Company
P.O. Box 8699
Phildelphia, Pa. 19101

The Unsupported Versions working group continues in its efforts to provide support for "old" versions of RSX. People presenting solutions to problems in previous releases of RSX will be submitting articles to Multitasker, covering changes to retro-fit Bonner lab Runoff, SRD, etc. on old versions of RSX. An article was sent to the Multi-tasker in regard to using BRU with MT: tape drives. Bill Burton will be unable to attend the fall symposium but Jeff Hamilton will chair the unsupported versions working group session.

The System Performance and Accounting working group is continuing its work in preparing the index of the past RSXSIG tapes in regards to System performance and accounting. There is much interest in continuing the development of performance measuring tools. Further contact with DEC in regards to the working groups goals is being pursued. An article is being prepared for the multitasker. Further work is being done in the area of consolidating the working group's division of labor.

A restructoring of the tree for distribution of the DECUS library tape has been done. The people on the tree are being used in the evaluation of a sample 2400' 800 bpi tape that will be submitted to the DECUS library. The tape was sent to the volunteers the Friday before the Spring Symposium. Evaluation is being done on the tape at this time. A couple of responses have been returned. A lot of the tree evaluation team have failed to return the tape with their evaluation and some have even disappeared with their tapes. Bruce requests those he has not been able to get in touch with to please help him by either returning their tapes or finishing their evaluation.

The SIG Tape working group reports that the spring 84 tape will be distributed around the 18th of September. As of August 21st the midwest tree has been finalized and the eastern and western trees are almost finalized. The spring tape is in it's final form because of the lack of space on the tape to include anything more.

The SRD working group has started making changes to SRD for Micro/RSX and POS (basically for named directories). Hopefully they will have the changes made by the next symposium. A minor bug fix is being worked up for the Fall symposium tape. The SIG data base is being used in the evaluation of different SRD and SRD command file procedures. The SRD Working group has submitted a seperate article to this issue of the Multi-Tasker describing corrections for SRD V6.4.

The Real-time and Industrial Automation working group is working on the pre-symposia seminar that they are sponsoring. They are also working on a mailing about the role of the Real-time working group versus the LABS SIG. Other work is being done in the are of volunteers and objectives. Anyone with ideas, suggestions, or extra time and energy; please let Ed know.

The Runoff group has continued its effort to consolidate desirable features of several versions of Runoff into an "official" version. Chuck, due to work efforts, is finding the time to upkeep the working group more and more difficult. If there is anyone willing to help in this position, please get in touch with Chuck.

The Cheap Networks working group got off the ground with the volunteer effort of Evan Kudlajev who volunterred to be the chairman of this group. An evaluation of the current free software

for networking applications is the group's first task. Further work in the area of layering a mail application on this software was discussed. Evaluation of the current version of RSX KERMIT (2.17) is continuing. Distribution of the KERMIT kit is being done by KERMIT transfers and other virtual terminal transfer programs. An article is being prepared for the Multitasker.

The formation of the computer aided instruction working group still awaits the appointment of a working group chairperson.

If you are interested in providing information to a special working group concerning problems or ideas in that area, please get in touch with the working group chairman of that group.

SRD Corrections

Submitted by the SRD Working Group

SRD V6.4 found on the Fall 1983 RSX SIG Tape in [352,4] has a bug which shows up when searching by date if a date stored in a file header is corrupt. SRD reports the problem, but then erroneously reports corrupt dates for every file which satisfies the date selection criteria.

Another problem showed up recently for ODS1 disks restored on VMS using BACKUP/IMAGE. For files having dates with a single digit day of month (for example, 4-OCT-84), VMS BACKUP stores the date with a blank preceding the digit (" 4OCT84"). RSX stores the date with a leading zero ("04OCT84"). SRD V6.4 does not expect the blank and reports that the date is corrupt.

The following correction files fix these problems. Follow this procedure to generate an updated SRD.TSK:

- Create SRDATA.COR, SRDLST.COR, and SRDSUB.COR.
- Rename the corresponding V6.4 source files to version 1:

```
PIP *.*;1/RE=SRDATA.MAC;2,SRDLST,SRDSUB
```

- SLP @SRDATA.COR
  SLP @SRDLST.COR
  SLP @SRDSUB.COR
- @SRD

If the *.OBJ files exist from previously building

SRD V6.4, only SRDATA, SRDLST, and SRDSUB need to be assembled by SRD.CMD.

```
; SRDATA.COR
;
; This file updates SRDATA.MAC;1 from SRD V6.4.
SRDATA.MAC;2/AU=SRDATA.MAC;1
®
-/.IDENT/,.
   .IDENT  -6.4A-          ; AUG-84
-/for-selective-delete/
;
;
;  VERSION 6.4A - Aug-84           (;WG001)
;
;
%
-,,/;WG001
-/SRDVER::/,.
SRDVER::.ASCII     <12><15>"SRD -- Version WG-6.4A, Modified August 19
/


; SRDLST.COR
;
; This file updates SRDLST.MAC;1 from SRD V6.4.
SRDLST.MAC;2/AU=SRDLST.MAC;1
®
-/.IDENT/,.
   .IDENT  -6.4A-          ; Aug-84
-/three times/
;
;
;  VERSION 6.4A - Aug-84           (;WG001)
;
;
;          Correct bug so that diagnostic message for corrupted date
;            in file header is not repeated for files with proper date
;
;  ;       ;
%
-,,/;WG001/
-/LSTE1:/
-/12$:/
-/DIAG...BADDH/
   BIC     #DATERR,FLAGS$  ; Clear file-date-corrupt flag
/


; SRDSUB.COR
;
```

```
; This file updates SRDSUB.MAC;1 from SRD V6.4.
SRDSUB.MAC;2/AU=SRDSUB.MAC;1
®
-/.IDENT/,.
   .IDENT  -6.4A-          ; Aug-84
-/address ending/
;
;
;  VERSION 6.4A - Aug-84           (;WG001)
;
;          In the CVDATE routine (in SRDSUB), skip over the first
;            byte of the date field if it is blank.  When
;            VMS BACKUP/IMAGE restores a Structure 1 disk, a file
;            date with a single digit day of month is stored in
;            the file header with a blank preceding the digit,
;            rather than a zero character as is done in RSX.
;            (For example, " 4OCT84" rather than "04OCT84".)
;
%
-,,/;WG001/
-/CVDATE::/
-/$SAVRG/
   CMPB    #<' >,(R1)      ; Is first byte of date field a blank?
   BNE     5$              ; NE - no
   INC     R1              ; Skip over the blank
5$:
/
```

THE DAARC SIDE OF THE FORCE

A New Name for the LABS SIG
Announcment from DECUS

For some time, the Steering Committee of the LABS SIG has been concerned about reaching a hitherto unreached portion of the DECUS membership. The unreached portion consisted of those involved in real time process automation and control. At the same time, it was thought that a gathering place for those involved in the analysis of data was also indicated. The line of reasoning went something like this:

1. The practice and craft of real time data acquisition has similarities whether the practitioner is acquiring data in a laboratory on a small RT based system or controlling an industrial process with a network of VAXes.

2. Outside problems (noise, grounding, physical hazards, etc.) are faced by the laboratory worker as well as his or her industrial automation counterpart.

3. Once one has acquired data, one has to do something about it, usually analysis of some sort, whether it be to find the peak of a spectra or establish the setpoint for a controller.

Therefore, the Steering Committee resolved to provide a place in DECUS where, in an operating system free environment, these items could be freely discussed and shared.

Toward the aforementioned ends, the committee has opened up the LABS SIG to embrace DATA ACQUISITION, ANALYSIS, RESEARCH and CONTROL. Henceforth, the LABS SIG will be known as DAARC and will endeavor to provide a forum for a valuable exchange of knowledge between a broader spectrum of the DECUS membership and also to provide DIGITAL with a SIG interested in products catering to those areas of interest.

The DAARC steering committee wishes to stress that DAARC is an applications oriented SIG open to participation by all DECUS members. DAARC will also relay real time based needs to the proper operating system SIGs via working groups.

So, if you're using a computer in a laboratory or trying to interface a system to an industrial process, come join some kindred souls in DAARC. DAARC will be sponsoring several sessions at the Anaheim Symposium this December, why not join us there?

Contact: Jim Deck, DAARC SIG Chairman
c/o DECUS, 249 Northboro Road, BP02
Marlboro, Massachusetts 01752

TTDRV "BUG" in Update D RSX-11M-PLUS

There is a change being released in update D of 11M-PLUS related to the terminal driver and remote lines. In previous releases, DTR was always high unless the line was being hung up. With the Update D, DTR is not raised until it detects a ring signal.

There are two ways to change this behaviour to the old DTR handling. The first is before a SYSGEN is done, you can edit SYSCM.MAC. Find the line:

```
     $TTPRM::.WORD    1                ;DEFAULT TERMINAL DRIVER BEHAVIOR ;
DD187
```

Change the value from '1' to '2'.

If a SYSGEN is already done, you can find the location in RSX11M.MAP, and OPEN or ZAP the location.

It is not really a bug. It should work just as well whether we look at Ring and then bring up DTR and look for CD, or always have DTR high and just wait for CD.

$TTPRM in SYSCM now controls modem behavior. It only affects 11M+, since that Update was out before the "problem" was noticed. It affects all terminal driver devices.

STATEMENT OF MENU ITEMS

There were many new items added to the RSX menu as a result of the submissions received at the Spring 1984 DECUS Symposium. To encourage submissions, DEC developers agreed to examine the list of items to respond to items that could be answered quickly and to indicate that the remaindor were items that the RSX users needed to submit their ballots for.

The following is a list of the response by DEC sent by Gary Oden. DEC's RSX Development Engineering responses are in brackets ([]).

1. At one time, typeahead (3.2) was allowed on slaved or attached terminals. The decision was made to support only on attached. Some of our applications did not allow for attaching, but the terminals were slaved, and typeahead worked for us. Digital did publish a patch for 4.0.

   [What we'd like to do is as follows: we don't want to change the function back, nor do we want to keep publishing the feature patch. What we'll do is put documentation into the driver source on how to change the behavior. It's a one line edit, zap, or conditional. We'll throw in the disabling of output buffering feature patch, for good measure.]

2. RMS is only able to provide bucket-level locking (rather than record-level locking as on VAX VMS). Provide a means for the RMS developers to be able to implement record locking under RSX.

   [Not likely to be done: Architecturally, RSX knows nothing about records, and we believe the changes to be extensive to implement record locking. VMS also has a much larger amount of shared database amongst the accessors.]

3. Provide support in the the FORTRAN/MACRO Symbolic Debugger for I- and D- space tasks, perhaps distributed as a separate debugger for M-Plus systems.

   [planned]

4. Fix ABORT so that a task which is marked for abort will not wait infinitely for impossible completion of I/O.

   [No, no, a thousand times no!]

5. Allow use of (Q-Bus) RS02 without restriction on 22 bit systems.

   [Fixed in last release - 2.1/4.1 updates "B"]

6. Change Sysgen so that when running from a saved answer file it is possible to specify which answers have been changed since last assembly, and use that information to do conditional assembly of only those modules affected by the change.

   [Yecch! - We think this would be an accident often finding a place to happen.]

7. For many processes, disk block caching would help performance.

   [Already planned]

8. Continuation line support in GCML$

   [We've tentatively planned this for 2.2. If it isn't in there it will be because it wasn't feasible.]

9. Allow the CON OFFLINEing of an IP: subsystem and subsequent powering down of said IP: system without crashing the PDP-11 and occasionally wiping out home blocks.

   [The RSX group doesn't own the IP driver anymore, but we'll forward the request.]

10. Add PLAS directives to AME (simpler cluster libraries would work).

[planned]

11. To make the Receive Region by Reference directive useful for us, we need the additional capability of Receive Region by Reference or STOP.

    [We're trying to get this into V2.2 of RSX-11M-PLUS.]

12. Autoconfigure / parameter-driven startup command files should be made available on full kit-currently only on pregenerated kits. also MCR set/crash-device =N: command.

    [Unlikely. Only some devices have been converted to allow on-line configure/set crash support. The general case is orders of magnitude harder.]

13. Convert PDP-11 PASCAL to use RMS instead of FCS.

    [No commitment yet, but being planned.]

14. Rewrite all utilities to use RMS instead of FCS, so they can use transparent DECnet support.

    [Unlikely. Scope is enormous, but may do selected utilities.]

15. Datatrieve-11 should use I/D space, supervisor mode RMS, plas overlays, RMS DAP support for DECnet, be built multi-user (/MU) to share memory.

    [Unlikely.]

The following is the list of menu items for which the RSX users need to submit their votes.

Indicate on the last page your votes for the RSX menu ballot. Remove that page and mail it to the address below, so that we receive it on or before November 9, 1984:

DECUS RSX Menu
248 Northboro Road, BP02
Marlboro, Mass. 01752
Attn. Shelly

1. Support of BREAK function of terminal interfaces (DL-11, DZ-11, etc.) for communication with foreign systems.

2. Provide mechanism to specify which checkpoint file to use for installed tasks.

3. Add a switch to DMO to dismount the disk even if checkpointing space is allocated (after migrating any checkpointed tasks elsewhere, of course).

4. Terminals spooled by QMG should display (through DEV or SET) all terminal characteristics such as speed, type, etc.

5. QMG/SHR INIT or STOP should not wait if the device is attached by another task.

6. Enhance ACS with switch to list all tasks checkpointed to a given file

7. Enhance ACS to allow forcing something checkpointed to become not checkpointed

8. Support for Digital DF03 modem is sorely lacking. Need ability to control automated facilities of modem.

9. Double-check with user before BAD and INI.

10. Provide memory virtual disk driver support as an official part of the RSX-11M Plus system.

11. It's hard for a first time user to understand what RMS is/does/can/cannot do.
Provide better introductory macro programming documentation for both FCS and RMS.

12. Modify QMG package and PRT... to allow record lengths in excess of 132. characters.

13. Transparent spooling over DECnet! (FTS does not qualify)

14. Give LPP (line printer despooler) the smarts to recognize the XOFF/XON state of a spooled terminal device (such as a LA100, LQP02, LA180, etc.) and treat a long-term XOFF in the same manner as a line printer being off-line.

15. Allow update to patch a choice of 1 volume on RL02 distribution kit.

16. Please supply a way to alter the print flag page.

17. Modify sysgen procedure to automatically include devices NL: and CO:, at least NL:.

18. Change default in RSXBLD.CMD to assume that loadable user written drivers have a loadable data base, to eliminate the necessity of editing RSXBLD.CMD to perform the GBLDEF of $USRTB.

19. Do not have 3-letter names for things in [1,54] and [3,54] which should not be flying-installed by TDX, e.g. BOO,LDR,MCR,TKN.

20. Allow more of the layered software products to use I/D space (e.g. FMS, DECnet, DPM).

21. Provide driver for DMF-32
[Unlikely. This isn't a software issue as much as diagnostics, service, etc.]

22. Allow M+ I/D space non-multiprocessor systems to use APR 00 for pool.

23. For M+ I/D space non-multiprocessor systems turn INITL into real pool, not ICB pool.

24. ALlow RSX networks tasks to be run under VMS compatibility mode.

25. Implement a "command line editor", to allow recall and editing of the last one or several MCR commands.

26. Save worst case pool stats when changing pages in RMD and then returning to "M" page.

27. There is no way to tell which ERRLOG.LST came from which physical processor when there are two 11/70 processors, one serving as a hot back-up for the other. The disks are dual-ported and follow the o/s's; they do not stay with the processor's.

28. Allow an optional number of trailing form feeds to be put on print jobs using the new Queue Manager System.

29. Have the ability to set the system name at start-up (or possibly with VMR) rather than early in sysgen.
[DEC specifically did not allow changing the name with MCR because of DECnet, but VMR might be possible.]

30. Modify the ACP to collect internal performance statistics which can be dumped into the accounting file.

31. Task resource accounting selective by task, instead of current all-or-nothing accounting.

32. Drop remote lines (DZ11) if no activity or no login present.

33. Would love to have FLX read/write RSTS file structure disk.

34. Allow easily determining whether a device is mounted at TI:

35. Make RSX-11M/M+ Fortran-77 compatible with VAX-11 Fortran-77 (with the obvious exceptions disregarded).

36. Provide a software tool(s) like VAX CMS and MMS for RSX. (CMS is a Code Management System that provides control over program development. MMS is a Module management system which automates software system building.)

37. Develop FMS-11 2.0 for 11M+, Micro/RSX, P/OS.

38. Supervisor mode FMS resident library and supervisor mode common run time library along lines of VMS common RTL.

39. Allow ASCII form names for queue manager, generic escape sequence for print spooler.

40. Make DAPRES (RMS DECnet support resident library) run in supervisor mode like RMSRES.

41. Settable upper and lower memory limits for RMD memory page.

42. Supply DTE program from Micro/RSX on 11M+ kits.

43. Provide tuning documentation on tasks supplied with an increment and what they use the extra memory for (so you can figure how much more you want to give them), which could be rebuilt in I/D space, with supervisor mode.

44. EDT should support the delete upon exit EDX command like EDI.

45. Allow unsolicited escape sequences to be sent to the appropriate CLI.

46. Allow OPE to use a command file (e.g. "OPE @CMD") or a one line open-patch command for use by indirect command files, complete with link tracing.

47. Files 11 provides no way or determining that a file which is opened for read regularly but not updated is not an old file. Provide a "file-last-accessed-when" mechanism for determining this.

48. Enhance the level of security offered by the ACNT program.

49. Allow (easy) configuration of foreign (non-DEC) terminals to TT: driver and editors.

50. For the crash memory dump device, SYSGEN currently asks you for the device name and its controller's CSR address. It parses out the unit number from the device name you supply and assumes that the data will go to that unit on that controller. If you have specified a disk or tape that has multiple controllers, and the crash dump device is not on the first controller, CRASH may wind up dumping memory to a different drive than that which the operator is requested to mount the scratch media in, with possibly catastrophic results.

    If one selects a disk or tape as the memory dump device, SYSGEN should ask for (1) the device's name; (2) its CSR address; and (3) the unit number of that device on the controller; then configure the CSR, unit, and device name in the message from these answers. (DEC please refer to SPR #11-68478)

Indicate on this sheet your votes for the RSX menu ballot. Return on or before 11/9/84.

Site:_____

Operating System:_____

CPU type:_____

1.

/strongly agree /agree /neutral /disagree /strongly disagree

2.

/strongly agree /agree /neutral /disagree /strongly disagree

3.

/strongly agree /agree /neutral /disagree /strongly disagree

4.

/strongly agree /agree /neutral /disagree /strongly disagree

5.

/strongly agree /agree /neutral /disagree /strongly disagree

6.

/strongly agree /agree /neutral /disagree /strongly disagree

7.

/strongly agree /agree /neutral /disagree /strongly disagree

8.

/strongly agree /agree /neutral /disagree /strongly disagree

9.

/strongly agree /agree /neutral /disagree /strongly disagree

10.

/strongly agree /agree /neutral /disagree /strongly disagree

11.

/strongly agree /agree /neutral /disagree /strongly disagree

12.

/strongly agree /agree /neutral /disagree /strongly disagree

13.

/strongly agree /agree /neutral /disagree /strongly disagree

14.

/strongly agree /agree /neutral /disagree /strongly disagree

15.

/strongly agree /agree /neutral /disagree /strongly disagree

16.

/strongly agree /agree /neutral /disagree /strongly disagree
17.

/strongly agree /agree /neutral /disagree /strongly disagree
18.

/strongly agree /agree /neutral /disagree /strongly disagree
19.

/strongly agree /agree /neutral /disagree /strongly disagree
20.

/strongly agree /agree /neutral /disagree /strongly disagree
21.

/strongly agree /agree /neutral /disagree /strongly disagree
22.

/strongly agree /agree /neutral /disagree /strongly disagree
23.

/strongly agree /agree /neutral /disagree /strongly disagree
24.

/strongly agree /agree /neutral /disagree /strongly disagree
25.

/strongly agree /agree /neutral /disagree /strongly disagree
26.

/strongly agree /agree /neutral /disagree /strongly disagree
27.

/strongly agree /agree /neutral /disagree /strongly disagree
28.

/strongly agree /agree /neutral /disagree /strongly disagree
29.

/strongly agree /agree /neutral /disagree /strongly disagree
30.

/strongly agree /agree /neutral /disagree /strongly disagree
31.

/strongly agree /agree /neutral /disagree /strongly disagree
32.

/strongly agree /agree /neutral /disagree /strongly disagree
33.

34. /strongly agree /agree /neutral /disagree /strongly disagree

35. /strongly agree /agree /neutral /disagree /strongly disagree

36. /strongly agree /agree /neutral /disagree /strongly disagree

37. /strongly agree /agree /neutral /disagree /strongly disagree

38. /strongly agree /agree /neutral /disagree /strongly disagree

39. /strongly agree /agree /neutral /disagree /strongly disagree

40. /strongly agree /agree /neutral /disagree /strongly disagree

41. /strongly agree /agree /neutral /disagree /strongly disagree

42. /strongly agree /agree /neutral /disagree /strongly disagree

43. /strongly agree /agree /neutral /disagree /strongly disagree

44. /strongly agree /agree /neutral /disagree /strongly disagree

45. /strongly agree /agree /neutral /disagree /strongly disagree

46. /strongly agree /agree /neutral /disagree /strongly disagree

47. /strongly agree /agree /neutral /disagree /strongly disagree

48. /strongly agree /agree /neutral /disagree /strongly disagree

49. /strongly agree /agree /neutral /disagree /strongly disagree

50. /strongly agree /agree /neutral /disagree /strongly disagree

MULTITASKER

/strongly agree /agree /neutral /disagree /strongly disagree

Using a Second Terminal on a PC350 with the Native Toolkit

Forrest Foor
419 W. 56th St.
N.Y., N.Y. 10019

One of the interesting features of the PC350 is the ability to attach a second terminal to the printer port with a console cable (BCC14-10). The console cable shorts pins 8 and 9, allowing the hardware to determine that a terminal, rather than a printer, is attached to the printer port. The second terminal port has the device name of TT2.

This terminal was intended as a debugging and testing feature. It is particularly useful when debugging programs which utilize graphics and forms, since the debugger dialogue can be redirected to the second terminal and will not cause the display to scroll off the screen.

The second terminal is also useful, if the PC is being used for application development with the Native Toolkit. The DCL interpreter can be installed under a second task name and spawned to TT2. This can be accomplished by running a simple macro program (see below).

Running the Toolkit from two terminals has certain advantages. For instance, two users can be editing, compiling, and linking programs at the same time, or a single user can compile several BASIC PLUS-2 modules and task build them, while editing other files. (At the present time there does not appear to be a way to spawn compilations in background mode with the BASIC PLUS-2 compiler.)

Performance is somewhat degraded, when running tasks simultaneously. However, the time savings for the programmer can still be substantial. Compilation times with the BASIC PLUS-2 compiler (V2.2-00) with a representative module were compared under the following conditions:

1)	running alone	3.8 min
2)	with EDT (V3.00-14), medium speed typing	4.7 min

(The most noticeable degradation in EDT performance occurred during writes to the journaling file, and when searching for text. Start up times were also affected.)

3) compiling the same module
on TT1 and TT2 6.4 min

(Both compilations finished at essentially the same time.)

One might have expected that memory demand would lead to extensive checkpointing under these conditions; however, addition of an extra 256k memory module (above the standard 512k) did not improve performance. The BASIC PLUS-2 compiler and EDT have not yet been written to take full advantage of the extra memory available on the PC350. Instead the compiler and and the text editor utilize work files on the disk extensively. Most of the degradation in performance is probably attributable to competition for disk I/O.

DCL commands can be run simultaneously from both terminals, since the utility programs are installed under different names. For instance, when you enter the DIRECTORY command from TT1, PIP is installed as PIPT1, while from TT2, it is installed as PIPT2. The same is also true for such commands as BASIC (BP2T1 vs. BP2T2), EDIT (EDTT1 vs. EDTT2), SHOW TASKS (CA2T1 vs. CA2T2), SHOW MEMORY (RMDT1 vs. RMDT2), etc. An exception to this, and an inconvenient one, is INDRCT.TSK, which is run with the task name "INDRCT" from both terminals. Using the DCL "@" command on TT2, while INDRCT is active on TT1, causes DCL to hang up on TT2. If this happens, enter ABORT TKT2 from TT1, and respawn DCL on TT2 (see below).

To run utilities simultaneously with the RUN command, you must first install them under different names, since the RUN command installs all tasks as "TT1", regardless of the terminal from which it is issued. For instance, you can use the following commands to run PAB (Pro Application Builder) from TT2, when PAB is already running on TT1 as task "TT1":

```
$ INSTALL /TASK:PABT2 APPL$DIR:PAB
$ RUN PABT2
```

It is possible to run P/OS utilities and applications with DCL. If the terminal on TT2 supports eightbit characters and has the LK201 keyboard, e.g., a VT220, then P/OS utilities, and applications which utilize menu routines and function keys, can also be run on TT2. The command RUN C$DUTL, for instance, will run disk services on either terminal. The P/OS utility task files reside in [ZZSYS] and are always installed. A second copy can also be installed under another name.

P/OS Utility	File name	Task name
File Services	CFUTL.TSK	C$FUTL
Disk Services	CDUTL.TSK	C$DUTL
Print Services	CPUTL.TSK	C$PUTL
View Message/Status	CVUTL.TSK	C$VUTL

Use the DCL command CLEAR to clean up the screen after running these utilities. You will find File Services is of limited usefulness, when DCL is available. When a second terminal is attached, Print Services cannot be used (see below). Disk Services, however, provides functions which are not available with DCL alone.

To run an application, it is necessary to determine in which [ZZAPnnnnn] directory the P/OS install utility has placed it. It may also be necessary to install certain libraries and, rarely, to define the logical names APPL$MENU, APPL$HELP, or APPL$HFRAME. Refer to the ZZAPnnnnn.INS file for this information. Command files for running applications are particulary useful, but, as mentioned above, can cause problems, when there are two terminals.

If the task uses any menu, help, or message files, it may be necessary to redefine APPL$DIR, while running the application. This effectively prevents running two such applications simultaneously from different terminals and may cause other problems. Be sure to reassign APPL$DIR to the DCL directory after running the application. Note that running communications (CCMAIN.TSK) does not require redefining APPL$DIR.

Also beware of changing the default directory name, since it is changed simultaneously for both terminals. If task builder or compiler command files use full device and directory specifications, this is less of a problem.

Before running the Toolkit on TT2, you may need to set the terminal characteristics. The settings depend on the type of terminal which is attached. For a VT100, for example, you might need to enter:

```
$ SET TERMINAL:TT2:/VT100/SPEED=(4800:4800)/NOEIGHTBIT
```

If you use EDT on TT2 and you are using a VT100, you will need to enter the EDT line editing command SET TERMINAL NOEDIT.

Use ABORT TKT2 to stop the Toolkit running on TT2. If you happen to use the SHOW MEMORY command on TT2, you will probably find you can't exit with ctrl/Z or ctrl/C, and will have to use ABORT RMDT2 issued from TT1. Ctrl/C trapping is task dependent, since TT2 interrupts are are not normally enabled.

You can also print, if the attached terminal has a printer. Set the TT2 terminal to printer copy mode and TYPE the file. If Toolkit DCL is not running on TT2, set the TT2 terminal to printer copy and use commands issued from TT1, such as COPY FILE.DAT TT2: or DIR/OUT=TT2:. The P/OS print utility will not work, since it detects no printer on the port (an inconvenience).

Toolkit DCL does not appear to support running another copy of DCL on TT2 with commands alone. However, this can be accomplished by running the following macro program:

```
   .TITLE SPWNTK
;
; Program function:
;
;  Installs DCL.TSK with the NOREMOVE option  and with the task name
;  specified as TKT2 to differentiate it from NATVTK on TT1.
;  The program uses the PROTSK routine in the POSSUM library.
;  Be sure to task build with the option LIBR = POSSUM:RO.

;  TKT2 is then spawned establishing TT2: as the task's physical
;  terminal.  Task build with the switch /PR:0 on the output
;  file.  (The task must be privileged to specify a new TI:.)
;
   .MCALL  EXIT$S, SPWN$S
;
; Local data
;
STAT:       .BLKW   8.                  ;  POSSUM status block

INSTL:      .WORD   1.+32.+64.          ;  Request argument:
                                        ;    1 = install the task
                                        ;   32 = task name specified
                                        ;   64 = noremove

TSKNAM:     .RAD50  /TKT2  /

FILNAM:     .ASCII  /DCL.TSK/           ;  The default directory should be set
                                        ;  to APPL$DIR, when this task is run.
   LEN = . - FILNAM
   .EVEN
FILLEN:     .WORD   LEN

ARGLST:     .BYTE   5, 0                ; No. of arguments in the call
   .WORD    STAT                ; Status block address
   .WORD    INSTL               ; Request value address
   .WORD    TSKNAM              ; Address of RAD50 task name
   .WORD    FILNAM              ; Address of task file name
   .WORD    FILLEN              ; Address of length of file name

; Code:

SPWNTK::

; Call the POSSUM routine PROTSK

   MOV      #ARGLST, R5
   JSR      PC, PROTSK

; Spawn TKT2 on TT2:
```

```
    SPWN$S  #TSKNAM,,,,,,,,,,#2,#"TT

    EXIT$S

    .END SPWNTK
```

In summary, although P/OS is not a multi-user operating system, it can support an additional terminal on the printer port. The lack of multi-user support can lead to problems, but most of these are not serious, and can be worked around. The addition of a second terminal may be advantageous for debugging and application development with the Native Toolkit.

Tracking Down Programs That Depend on a Subroutine

A. Randall Barron
Gas Turbine Laboratory
Massachusetts Institute of Technology
Cambridge, MA 02139

When you make serious revisions to a subprogram, changing its argument list for instance, you might well want to identify all the tasks in your directory that use the subprogram -- in case they stand in need of revision. The indirect command command file TSKBLT will do this for you, provided you have observed certain common conventions in naming task files and TKB command files. If you don't use this convention with the aid of BOZO.CMD, you may be able to modify the routine to suit your own convention.

```
.;+TSKBLT  -- LISTS TASKS THAT WERE BUILT WITH SPECIFIED OBJECT FILES
.;+ V1.2 -- Written, 6-SEP-84, by A. R. Barron, MIT Gas Turbine Lab.
.;
.;  CALL:
.;              @TSKBLT FOO,BAR,BLETCH,...
.;
.;  The routine works by checking all TKB command files in the default
.;  UFD for occurrences of the specified object filenames.  Tasks are
.;  identified by the extension "TSK", and the corresponding TKB commar
.;  files must have the same names, but extension "CMD" -- otherwise,
.;  the routine will ignore them.
.;
        .ENABLE QUIET,SUBSTITUTION
        .DISABLE DISPLAY
        .ONERR GAG      ! ESTABLISH TRANSFER TARGET FOR SYNTAX ERRORS
```

```
        .SETS SLASH "/"          ! THE DELIMITER FOR TKB OPTIONS
        .OPEN TI:

.;  Parse the input command string into a list of object file names.

        .PARSE COMMAN " " DUMMY LIST
        .TEST LIST
        .IF <STRLEN> EQ 0 .DATA TYPICAL CALL: @TSKBLT FOO,BAR,BLETCH
        .IF <STRLEN> EQ 0 .EXIT <WARNIN>
        .SETN NCNT 0
        .SETS REMS LIST
.Q:
        .PARSE REMS "," TEMPS REMS
        .TEST TEMPS
        .IF <STRLEN> EQ 0 .GOTO TEST
        .INC NCNT
        .SETS STR'NCNT' "'TEMPS%C'"     ! STRIP OFF OUTSIDE BLANKS
.TEST:
        .TEST REMS
        .IF <STRLEN> EQ 0 .GOTO NDQ
        .GOTO Q
.NDQ:
        .SETN NOBS NCNT

.;  Make a temporary listing of tasks in the default UFD.

        .SETF <ERSEEN>
        PIP TMP.DIR = *.TSK/BR
        .IFT <ERSEEN> .GOTO GAG

.;  Open the directory listing file and read the header.

        .OPENR#1 TEMP.DIR
        .IF <FILERR> NE <SUCCES> .GOTO GAG
        .SETS TMPDIR <FILSPC>
        .READ#1 DUMMY
        .IFT <EOF> .GOTO GAG

.;  Open the output listing file and write the header.

        .OPEN#2 TMP.LST
        .IF <FILERR> NE <SUCCES> .GOTO GAG
        .SETS TMPLST <FILSPC>
        .DATA#2 DIRECTORY '<LOGDEV>':'<UIC>', '<DATE>'
        .DATA#2 Tasks built with object files: 'LIST'

.;  Scan through the directory file

        .SETN COUNT 0
.SCAN:
        .READ#1 ENTRY
        .IFT <EOF> .GOTO NDSCAN
        .SETS ENTRY ENTRY[3:*]          ! STRIP OFF LEADING <CR><LF>
```

```
        .TEST ENTRY
        .IF <STRLEN> EQ 0 .GOTO NDSCAN  ! THE LAST LINE IS JUST <CR><LF
        .PARSE ENTRY "." TEMPS DUMMY
        .TESTFILE 'TEMPS'.CMD
        .IF <FILERR> NE <SUCCES> .GOTO SCAN

.;  If a task has a corresponding TKB command file, open it for reading

        .SETS F$NAME <FILSPC>           ! THE ROUTINE BARFS ON THE OPEN
        .OPENR#3 'F$NAME'               ! IF THE FILE DOES NOT EXIST
        .IF <FILERR> NE <SUCCES> .GOTO GAG

.;  Run down the TKB command file

.TOP:
        .READ#3 LINE
        .IFT <EOF> .GOTO BOT
        .IF SLASH EQ LINE [1:1] .GOTO BOT  ! SKIP THE LIST OF TKB OPTIO

.;  Loop through the items in the list of object files

        .SETN NCNT 1
.LOOP:
        .IF NCNT GT NOBS .GOTO NDLOOP
        .TEST LINE STR'NCNT'
        .IF <STRLEN> GT 0 .GOTO GOT
        .INC NCNT
        .GOTO LOOP

.NDLOOP:
        .GOTO TOP
.GOT:
        .DATA#2 'ENTRY'
        .INC COUNT
.BOT:
        .CLOSE#3
        .GOTO SCAN
.NDSCAN:
        .IF COUNT EQ 0 .DATA#2 ****NONE FOUND****
        .CLOSE#1
        .CLOSE#2
        PIP TI:='TMPLST'
        PIP 'TMPDIR'/DE/NM
        .ASK [<FALSE>] SAVE SAVE LISTING FILE 'TMPLST' [N]?
        .IFF SAVE .XQT PIP 'TMPLST'/DE/NM
        .EXIT <SUCCES>
.GAG:
        .IFNDF ERRMSG .SETS ERRMSG "TSKBLT -- FATAL ERROR"
        .IFDF ENTRY .SETS ERRMSG ERRMSG+" WHILE PROCESSING "+ENTRY
        .DATA 'ERRMSG'
        .DATA           CHECK TMP.DIR,TMP.LST
        .EXIT <SEVERE>
```

VIRTUAL Arrays in Common.....Revisited

Gary L. Maxwell

United States Geological Survey
345 Middlefield Road, MS 977
Menlo Park, CA 94025

In a previous issue of the Multi-Tasker (Vol. 15, No. 10, May/June 1982), Chris Doran described a method of patching the Task Builder so that all Fortran Virtual arrays from different modules could be overlaid, thus creating a single, unnamed "common block" consisting of all the Virtual Arrays used by the program. This feature allows one or more Virtual Arrays to be shared among subroutines without having to pass the arrays in argument lists.

Some time ago, I was disassembling the V4.1/V2.1 Task Builder (I forget why -- temporary insanity, I suppose), when I discovered that Chris' patches to the Task Builder had been included in standard TKB in their entirety by the RSX Developers!

I soon discussed my discovery with an RSX Developer, who expressed in no uncertain terms that this IS and WILL BE AN UNSUPPORTED FEATURE of TKB. However, the development team was apparently satisfied with leaving this feature in, due to the low overhead involved.

To use this feature, perform the following steps:

1. Define all Virtual arrays used by the program EXACTLY the same way in each program module which needs to access them (as though the VIRTUAL statement were really a COMMON statement). The easiest way of doing this is to create a definition file and INCLUDE it in the necessary modules.

2. Compile the program modules as you normally would.

3. In your Task Builder command file, include the option:

   VARRAY=OVR

   which forces the Task Builder to assign the OVR attribute to the PSECT $VIRT. By default, the Task Builder always assigns the CON attribute to this PSECT.

Note that the usual tricks you can play with COMMON blocks will not work with this feature. For example, defining VIRTUAL A(100) in one module and VIRTUAL B(50),C(50) does NOT equivalence C(1) to A(51).

The "CON" keyword can also be used with the VARRAY option, which concatenates all Virtual arrays (the default action).

The following simple example shows how the mechanism works.

Main Program (TEST.FTN):

```
      PROGRAM TEST
C
      VIRTUAL A(10), I(10)        ! Define arrays
C
      CALL SUBR                   ! Manipulate arrays
C
      WRITE (5,100) A
  100 FORMAT (2(1X,5F7.1,/))
C
      WRITE (5,200) I
  200 FORMAT (2(1X,5I7,/))
C
      CALL EXIT
      END
```

Subroutine (SUBR.FTN):

```
      SUBROUTINE SUBR
C
      VIRTUAL A(10), I(10)        ! Define arrays
C
      DO 1000 J=1,10
         A(J)=FLOAT(J)
 1000    I(J)=-J
C
      RETURN
      END
```

Compile, Task Build, and Execution Sequence:

```
>F77 TEST,TEST=TEST
>F77 SUBR,SUBR=SUBR
>TKB
TKB>TEST/FP/CP,TEST=TEST,SUBR,LB:[1,1]F77OTS/LB
TKB>/
Enter Options:
TKB>VARRAY=OVR
TKB>//
>
>RUN TEST
    1.0    2.0    3.0    4.0    5.0
    6.0    7.0    8.0    9.0   10.0

     -1     -2     -3     -4     -5
```

-6 -7 -8 -9 -10

>

Hows and Whys of ASTs in RSX

Gary L. Maxwell
United States Geological Survey
Menlo Park, California

Like all other operating systems, RSX-11 implements an interface between programs that run on the system and any external event which affects the execution of the program. As stated in the Executive Reference Manual, there are only two means of notifying a task on RSX that an external event has occurred: event flags, and Asynchronous System Traps, or ASTs.

Event flags are relatively simple to implement into any program, but they provide event response "passively." A task must wait for, stop for, or actively poll the event flags. These methods effectively reduce the parallelism of operations within a task.

ASTs provide "dynamic" event response. When an event occurs, the affected task's current processing is "interrupted" so that the event is handled responsively by an "AST service routine." The idea of a task "interrupt" is completely analogous to an operating system interrupt when a peripheral device requires attention. This allows a task to perform any background processing at "task level" while events are handled at "AST level." Furthermore, several types of events are only available through the AST mechanism.

The major obstacle to applying ASTs is the difficulty to conceptualize and implement them into a task. An additional difficulty is the lack of full support for ASTs in Fortran. Fortunately, structured HLLs (such as DECUS C) either support ASTs or easily adapt to their use.

This paper will address the following issues regarding the mechanisms and applications of ASTs:

1. How ASTs are generated and handled by the Executive, and how a task which services ASTs is affected by the AST mechanism.

2. The classes of ASTs in RSX, and the individual ASTs within each class.

3. The rules which should be followed in implementing an AST service routine in order to avoid common pitfalls and obscure programming bugs.

4. Features and restrictions of DEC supported AST services from Fortran programs, plus an example of unsupported AST usage in a Fortran environment.

1.0 HOW ASTS WORK

We can see how the Executive processes ASTs and how ASTs affect task operation by following the flow of an AST through the system.

1.1 "Arming" Of An AST By A Task

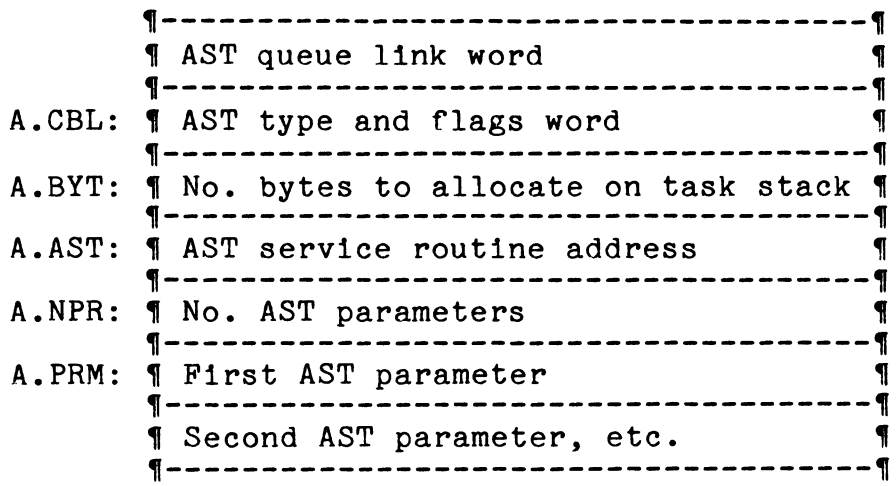
A task "arms" (or "disarms") an AST by means of an argument to a system directive. Either the AST argument is the address of the AST service routine which will field the AST (arming the AST), or the argument is zero, indicating that no AST is desired. When a task is initiated, all ASTs are disarmed.

The RSX Executive effectively arms the AST by allocating an "AST control block" from the Dynamic Storage Region and initializing some offsets within the control block.

1.2 Executive Declares The AST

An armed AST becomes a "declared" AST when the event associated with the AST occurs. The Executive fills any event-related information into the AST control block associated with the event, and enters the control block into a queue of declared AST control blocks for the task.

An AST control block has the following general format:



The AST control block is queued at the end of a first-in first-out list of other declared ASTs for the task. Therefore, ASTs are ordered by event time history for each task on the system. The AST queue is located in the TCB (Task Control Block) at offset T.ASTL. (We will see later how one class of ASTs, the Kernel ASTs, have their own queueing rules.)

1.3 Executive Dispatches The AST

The next time the task is scanned by the RSX Executive scheduler, the AST queue is examined. If the queue is not empty and AST recognition for the task is enabled, then the first queue entry is dispatched, even if the task is in a stopped or waitfor state. Dispatching the AST involves the following steps:

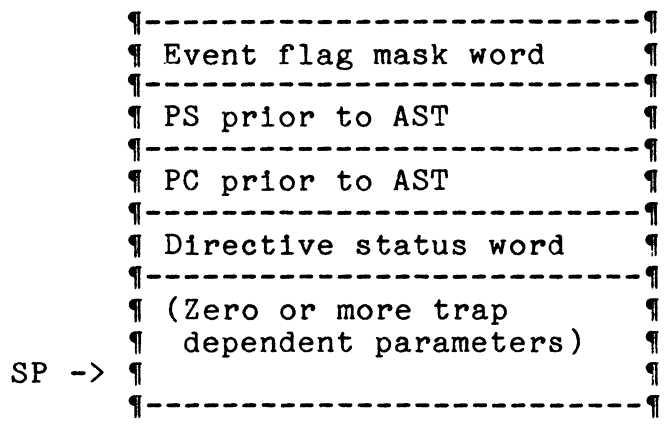
1. Four words of information regarding the task's previous context are pushed on the task's stack: the event flag mask (which identifies the event flag last used in a STOP or WAIT directive), the program counter, the processor status word, and the Directive Status word.

2. Any parameters associated with the event, called "trap dependent parameters," are pushed on the stack.

3. The Executive deallocates the AST control block, and transfers control to the AST service routine. The service routine, like the task itself, still competes with other tasks on the system based on the task's priority. The previous state of the task (stopped or "waitfor") is saved.

1.4 Task AST Service Routine Executes

The AST service routine is entered with the task's stack in the following state:



None of the general purpose or hardware Floating Point registers are preserved; therefore the AST service routine saves and restores these as necessary. The service routine is free to execute most of the system directives and perform regular computations. While the AST service routine is executing, all other declared ASTs are deferred (e.g., AST "interrupts" are disabled), guaranteeing serialization of ASTs.

The service routine exits by removing any trap-dependent parameters from the stack (always leaving the first four words pushed by the Executive) and by issuing an AST Service Exit (ASTX$S) directive. The Executive restores the previous context of the task by popping the DSW, PS, PC, and waitfor mask from the stack. If the AST queue is empty, then the task's context and previous state is restored. If another declared AST is pending, then it is dispatched at this time.

2.0 AST CLASSES AND THEIR DIRECTIVES

Although all ASTs are handled in a similar manner, there are actually four distict classes of ASTs that the Executive handles. Each of these classes has its own unique personality, and some have their own special rules.

For each type of AST within each class, there is a corresponding system directive which arms or disarms the AST. This section contains a description of all AST classes and their members, the AST-related system directives for each member, and the conditions which cause each AST to be declared.

2.1 Miscellaneous AST Directives

The following directives do not belong to any partiular class of ASTs. They provide support for manipulating the task's AST environment and perform special AST functions.

ASTX$ - AST Service Routine Exit. This directive is required to be the last instruction in an AST service routine. When this directive is issued, the task must be at AST state, and all trap dependent parameters must have been removed from the stack (leaving the four context words pushed by the Executive).

DSAR$ -
IHAR$ - Disable (or Inhibit) AST Recognition. Issuing this directive prevents the Executive from dispatching any ASTs for the task. This means that ASTs can be declared and queued up for the task, but no AST service routines are ever entered until AST recognition is enabled. This directive is used to place a "guard" around critical sections of code which cannot be guaranteed to execute correctly should an AST be dispatched. AST service routines can issue this directive so that no further ASTs are dispatched with the current service routine exits.

ENAR$ - Enable AST Recognition. This directive instructs the Executive to dispatch any ASTs presently declared for the task and clears a previously issued DSAR$ or IHAR$ directive condition. If an AST is already declared for the task when this directive is issued, the AST is immediately dispatched. When a task starts, AST recognition is enabled. AST service routines can issue this directive (but it is hardly ever necessary to do so).

CMKT$ - Cancel One or More Mark Time requests. If an AST service routine is specified with this directive, then all Mark Time requests that were issued with this AST service routine are cancelled.

CRVT$ - Create Virtual Terminal (M-Plus systems only). This directive creates a VT: unit for the issuing task (the parent), allowing the arming of three AST service routines. When an offspring task issues an I/O request to the VT: unit created by the parent, an AST is declared for the parent. The three AST service routines are used to separately service input, output, and attach/detach requests to the VT: unit by offspring tasks. The ASTs remain armed until the VT: unit is eliminated by the ELVT$ directive. This directive is supported by DEC in Fortran programs.

2.2 Class 1: Dynamic ASTs

The most common form of an external event occurs when an operation initiated by a task has completed. Examples of this class are I/O operations and timers. The ASTs which can be declared when such events occur are called "dynamic" ASTs, simply because only one AST is declared for every operation initiated by a task.

The following directives for dynamic ASTs are divided into two groups. The first group of directives do not support ASTs in Fortran, and include:

QIO$ - Queue I/O Function. The I/O operation specified in the argument list is initiated, and control returns to the program. The AST is declared when the I/O operation is finished.

QIOW$ - Queue I/O Function and Wait. Identical to QIO, except the task is placed in a waitfor state. The waitfor condition is cleared and the AST is declared when the I/O operation completes.

MRKT$ - Mark Time. The system is instructed to initialize a timer for the task and return control to the program. When the time period expires, the Executive declares a significant event, sets an event flag, and declares the AST.

The second group of directives supports ASTs in Fortran, and all of the directives are part of the Executive's parent-offspring tasking support. In all cases, the AST for these directives is declared for the parent task when the offspring task exits or emits status. These directives are:

CNCT$ - Connect to task. This directive connects the issuing task to another task (the offspring) which is already active.

SDRC$ - Send Data, Request, and Connect. This directive allows the issuing task to queue a standard data packet to, request (if inactive), and connect to an offspring task.

SPWN$ - Spawn. This directive spawns (requests), connects to, and optionally queues a command line to an offspring task,

VSRC$ - Variable Send Data, Request, and Connect (M-Plus systems only). Same as SDRC$, except the size of the data packet is under program control.

2.3 Class 2: Specified ASTs

There is a class of ASTs that addresses the need to handle certain events when and if they MIGHT occur. The family of "Specified ASTs" enables the task to identify to the system which events should be caught and passed on to the task using the AST mechanism. Most of the Specified ASTs provide a task with the ability to recover from certain "disasters," such as power failures.

None of the events available through Specified ASTs are available through the event flag mechanism. Note that only the SPRA$ (power recovery) and SREA$/SREX$ (requested exit) Specified ASTs are supported in Fortran programs.

The family of Specified ASTs and the function of each member is:

SCAA$ - Specify Command Arrival AST. For CLI (Command Line Interpreter) tasks only, this directive causes an AST to be declared when a command line is queued to the task. The command line is subsequently read with the GCCI$ directive.

SFPA$ - Specify Floating Point Processor Exception AST. This directive causes an AST to be declared when the hardware Floating Point Processor traps through the floating point exception vector. This directive is normally issued by the runtime system of any HLL program as part of a standard error recovery procedure.

SPEA$ - Specify Parity Error AST (M-Plus systems only). This directive causes an AST to be declared when a memory parity error is detected in one of the task's regions. On M-Plus systems, the Fixer task (FXR...) uses this AST to determine the range of bad memory within a partition and make the bad memory inaccessible to the system by creating an unusable region within the partition.

SPRA$ - Specify Power Recovery AST. This directive causes an AST to be declared when the operating system performs a power down/up sequence as a result of a power failure.

SRDA$ - Specify Receive Data AST. This directive causes an AST to be declared when a data packet is queued to the issuing task. The data packet can be dequeued using one of the Receive Data directives.

SREA$ -
SREX$ - Specify Requested Exit AST. Perhaps the best of the directives added to RSX in Version 4.0, these directives cause an AST to be declared when an attempt is made to abort the issuing task by the ABRT$ directive or the MCR/DCL ABORT command. This allows a task to clean up and exit gracefully instead of being rudely aborted.

SRRA$ - Specify Receive by Reference AST. Similar to SRDA$, this directive causes an AST to be declared when a receive-by-reference packet is queued to the issuing task. The task can dequeue the packet by issuing the RREF$ directive.

There are some special rules for Specified ASTs. They are:

1. A Specified AST is armed by issuing the appropriate directive with the address of the AST service routine. The AST remains armed until the same directive is issued with a value of zero for the service routine address, or until the task exits. There are two exceptions. The SPEA$ AST is always disarmed after the AST is dispatched. In processing the SREA$/SREX$ AST for non-privileged tasks, the Executive disarms the AST after the AST is dispatched, and any attempts by the task to re-specify the AST are disallowed.

2. None of the Specified AST directives can be issued from an AST service routine.

3. When an event causes a Specified AST to be declared, then that Specified AST is effectively disarmed until the AST service routine is dispatched. As soon as the service routine is entered, the Specified AST is rearmed. This rule implies that there is not a one-to-one correspondence between the number of events and the number of ASTs declared.

   This rule is very important, especially when the SRDA$ (Specify Receive Data AST) or SRRA$ (Specify Receive by Reference AST) directives are used. The program must assume that one or more additional events may occur between the time the AST is declared and the time the service routine is dispatched.

The Executive must perform some bookkeeping to keep track of the Specified ASTs that have been armed by a task. When a task arms a Specified AST, the Executive allocates the AST control block for the AST and links the block into a list of other armed Specified ASTs for the task in the TCB (at offset T.SAST).

When an event occurs for which there may be a Specified AST, the Executive searches the list of armed Specified ASTs in the TCB. If the correct control block exists, it is removed from the list at T.SAST and queued into the declared AST list at T.ASTL, declaring the Specified AST. Once the Specified AST has been dispatched to the service routine, the AST control block is reentered into the Specified AST list at T.SAST, rearming the AST. This procedure forms the basis for Rule 3 above.

2.4 Class 3: "Customized" ASTs

Support is provided for system processes (such as I/O drivers) to define "customized" ASTs and perform specialized actions when the AST is dispatched. Executive level processing for a customized AST is almost identical to other ASTs. The best (and only?) example of a customized AST is the Full Duplex Terminal Driver's Unsolicited Input Character AST.

A task arms the unsolicited character AST by attaching the terminal with a special attach function, IO.ATA. The terminal driver allocates an AST control block, and initializes two offsets which define a special routine in the driver to be executed when the AST is dispatched. When the terminal driver receives an unsolicited character, the AST is declared for the task.

When the AST is dispatched, the Executive sets up the task's stack, and calls the terminal driver's special routine. This routine recovers the AST control block, and scans the typeahead buffer for another unsolicited character. If one is already present, the driver declares the AST once again, (which will be deferred until the task services the original AST). The driver returns control to the Executive, which dispatches the AST to the task.

2.5 Class 4: Kernel ASTs

A Kernel AST, as the name implies, is declared, dispatched, and serviced within the Executive, and is completely transparent to all tasks on the system. Although Kernel ASTs use the regular AST mechanism, they have their own set of rules and procedures. Since the Kernel AST is a very unique and interesting feature of RSX, we will devote some discussion to them.

Just as a task uses ASTs to perform post-event processing, the Executive uses Kernel ASTs to perform certain Executive-level post-event processing. For example, an event may occur which requires the Executive to store some words of data into a task's address space. If the task is checkpointed, then the Executive cannot store the information until the task is reloaded into memory. A Kernel AST serves as a "place marker" for the Executive to complete some event processing when a task's address space and context are loaded and mapped.

A Kernel AST is declared by the Executive when event processing requires a particular task's context to be known and loaded. A Kernel AST is declared in the same manner as a regular AST, except the AST control block is entered at the BEGINNING of the declared AST queue, before any other ASTs. This is required so that the Executive's AST processing precedes the task's AST processing.

As with regular ASTs, a Kernel AST is dispatched when the Scheduler finds a declared AST entry for a task which is granted the CPU. Note that when an AST is dispatched, the Executive has already loaded the task's context. The Kernel AST control block is dequeued, and the Executive dispatches the Kernel AST to a particular service routine within the Executive.

The Executive's AST service routine performs the deferred processing specified by the Kernel AST, which completes the Executive's responsibility for handling the event. When the service routine completes, the scheduler is reinvoked to grant the CPU to the task with the highest priority.

The following subsections describe each type of Kernel AST. With the exception of the Load Region Kernel AST, all of the Kernel ASTs follow the above rules for declaring, dispatching, and servicing the Kernel AST.

2.5.1 Buffered I/O Kernel AST -

On heavily loaded multi-user systems, the checkpointing or shuffling of tasks that have outstanding terminal input requests can improve system performance. The ability to swap out tasks with outstanding I/O requires the Executive to store the input data in temporary buffers. On RSX, this is called Intermediate I/O Buffering, and this feature is used by the Full Duplex Terminal Driver.

When a task issues a read request to the terminal driver, and the region containing the task's I/O buffer is checkpointable, the driver allocates an intermediate buffer from pool and makes the task region eligible for checkpointing. The driver then supervises the I/O operation, using the intermediate buffer to store input characters as they are received. When the read request is completed, the driver cannot finish the I/O processing since the task region may have been shuffled or checkpointed. The driver declares a Kernel AST (with code AK.BUF) by calling the $QUEBF Executive routine.

The Kernel AST is serviced by a routine ($FINBF) which copies the contents of the intermediate buffer to the I/O buffer in the region, and the intermediate buffer is deallocated. The service routine calls $IOFIN to complete the I/O request in the regular manner.

A slightly similar Kernel AST is provided for more generalized intermediate I/O buffering, allowing the device driver to implement its own buffering scheme. Upon I/O completion, the driver declares the Kernel AST with a code of AK.GBI. The Kernel AST control block contains the entry point of a routine within the driver which will copy the intermediate buffer to the task buffer. When the Kernel

AST is dispatched (to the $GENBF routine), a call is made to this driver subroutine, which completes the Kernel AST processing.

2.5.2 Debugger Kernel AST (M-Plus Only) -

A handy new MCR command on M-Plus systems is "DEBUG taskname," which causes an executing task to trap to its debugging aid (provided it was built with one). This allows the user to force a runaway program to return control to the debugging aid.

The DEBUG command is implemented using a Kernel AST. When MCR receives a DEBUG command, it verifies the privilege of the user issuing the command, makes sure the target task is active, and then declares a Kernel AST on behalf of the target task with a code of AK.TBT.

When the target task is scanned by the Scheduler, the Kernel AST is dispatched to the $DBTRP routine.

The $DBTRP routine first checks the target task to verify that it actually contains a debugging aid. If it doesn't, then TKTN is requested to issue an error message on the terminal which issued the DEBUG command. Otherwise, $DBTRP sets the T-bit in the saved PS (Processor Status word) for the task in its task header, and the service routine exits. When the task is finally granted the CPU, the T-bit trap is immediately effected, and the debugging aid gains control of the task.

2.5.3 Finish Offspring Exit Kernel AST -

When an offspring task exits, several actions may be performed for the parent task. An event flag (possibly group global) may be set, an AST can be declared, an exit status block can be copied to a buffer in the parent's task address space, and the OCB (Offspring Control Block) must be deallocated. Since the parent task may be checkpointed, the exit status block cannot be copied until the parent task is loaded. Therefore, the Executive declares a Kernel AST with code AK.OCB for the parent task when an offspring task exits. When the Kernel AST is dispatched to the $FINXT routine, all of the above actions are performed, completing the offspring task rundown.

2.5.4 Group Global Rundown Kernel AST -

When a task exits with a pending Receive-by-Reference packet, a Kernel AST with code AK.GGF is declared for the sender task. When the Kernel AST is dispatched to the $GGFRN routine, a rundown on the sender task's Group Global event flags is performed if the Receive-by-Reference was synchronized with a Group Global event flag.

2.5.5 Delayed I/O Kernel AST (M-Plus Only) -

This Kernel AST is declared (with code AK.DIO) when a non-transfer I/O request is completed, and the target task has been checkpointed or shuffled. This is similar to the Intermediate I/O Buffering Kernel AST, except only the I/O status block in the task has become unmapped (there are no I/O buffers to copy). The Kernel AST is dispatched to the $FINDI routine to copy the two words of I/O status into the task's address space.

2.5.6 Region Load Kernel AST (M-Plus Only) -

As the name indicates, a Region Load Kernel AST deals with activities performed when a region is loaded from a checkpoint file. But the logical flow of a Region Load Kernel AST differs from all other Kernel ASTs, primarily because this type of Kernel AST is declared by another Kernel AST, the Buffered I/O Kernel AST.

When a Buffered I/O Kernel AST is dispatched, the $FINBF routine first checks to guarantee that the region containing the I/O buffer is memory resident. If it is, then the buffer is subsequently copied. If the buffer is not resident, then $FINBF must defer its processing until the region is loaded. This scenario only occurs when a task issues a buffered I/O request to a region and subsequently unmaps from the region. Since the region is not mapped when the scheduler grants the CPU to the task, the region is not required to reside in memory.

The region is loaded by forcing the task which issued the I/O request to access the region, which places the region into the Loader's queue and blocks the task from executing. The Buffered I/O Kernel AST control block is then inserted into the PCB (Partition Control Block) for the region at offset P.SWSZ (which normally contains the swapping size of the region), and a bit (PS.AST) is set in the PCB's status word.

The next action is taken by the Loader when the region is loaded back into memory. Seeing that there is a Kernel AST queued into the region's PCB, the Loader immediately dispatches the Kernel AST,

which causes the I/O buffer to be copied immediately. The I/O operation is completed, and the region is deaccessed from the task (since it is still unmapped).

3.0 HINTS AND KINKS IN WRITING AST SERVICE ROUTINES

The primary challenge of ASTs is to write an effective AST service routine. Because of obscure side effects or programming oversights, AST bugs can be extremely difficult to diagnose. An admirable goal is to correctly program ASTs into a task on the first try. Careful adherence to the following sets of rules for AST service routines can prevent many common AST programming problems.

Except in trivial applications, careful planning is required before actually writing an AST service routine. The scope of the service routine should be well-defined. An AST service routine is meant to perform time-critical operations in response to an event; therefore, it is wise to defer any processing that is not time-critical to normal task level.

3.1 Rules Governing All AST Service Routines

The following rules are basically "hard" rules, which means that breaking one of the rules will probably result in an incorrect AST service routine (which may work correctly some of the time). These rules apply to any AST application, whether the application is written in MACRO assembler or in a HLL.

1. A task's context, except for the four context words pushed on the stack by the Executive, is never preserved when a service routine is entered. A task's context includes the contents of the general purpose and Floating Point Processor registers, all mapping assignments to regions, and any loaded and/or mapped overlay segments. The AST service routine must save and restore any portion of the task's context, if required.

2. As inferred by the above rule, AST service routines must be located in the root segment of an overlaid task. This rule can be stretched by logically guaranteeing that the service routine will always be mapped when the AST is dispatched, but performing such "tricks" is not recommended.

3. Any subroutines which are commonly used at both task level and AST level must be reentrant. Such subroutines cannot have any static variables which can be overwritten should an AST occur. If common subroutines are not reentrant, then they must be guarded by a critical section, using the IHAR$

and ENAR$ directives at task level.

4. As previously mentioned, all trap-dependent parameters must be popped from the stack before the AST service routine exits. The task will be aborted by the Executive if the stack is incorrect when the service routine exits.

5. Some directives are illegal when issued from an AST service routine. These are the Specified AST directives and any directives which would cause the task to enter a stopped state within the service routine. Otherwise, an AST service routine can issue any system directive.

6. The state of the task at task level does not change when an AST is dispatched. This means that if a task is stopped, and an AST is dispatched, the task will remain stopped when the service routine exits. Therefore, if the task has stopped itself (or is waiting) in anticipation of the event handled by the AST, the AST service routine must manually unstop the task.

3.2 Rules Governing Supported Fortran AST Service Routines

The following rules apply to Fortran AST routines supported by DEC. Currently, these supported ASTs are the parent-offspring directives, the "requested exit" AST, the Create Virtual Terminal directive, and the power recovery AST. The rules also apply to specialized (and unsupported) AST applications in Fortran.

1. Since the Fortran I/O subsystem is not re-entrant, the AST service routine cannot issue any Fortran I/O commands. One workaround to this problem is to place critical sections around any Fortran I/O commands performed at task level, but this can be cumbersome and inefficient. Note that the QIO directive is not a part of Fortran I/O, so the service routine may issue QIO directives.

2. The Floating Point Processor registers are not saved before the AST service routine is entered, so no floating point operations (including any math library functions) can be performed within the service routine. A simple workaround is to use the SAVFPP routine in Appendix A, which is a coroutine that will save the FPP registers and restore them when the service routine exits.

3. A Fortran AST service routine should not use the STOP or CALL EXIT commands if output files are currently open. Since the Fortran I/O subsystem may be in an indeterminate state when the AST is dispatched, exiting the program at AST state may corrupt the output files.

4. The service routine cannot access any Fortran virtual arrays, since the AST can interrupt the virtual array subsystem at an indeterminate state.

5. The AST service routine must be compiled with traceback disabled (/NOTRACE). Otherwise, the AST service routine will corrupt the program's traceback. Additionally, any runtime error that occurs in an AST service routine may produce either an erroneous error message or other unpredictable results.

6. The Fortran AST interface imposes a finite limit on the number of outstanding armed ASTs a Fortran task can have. Currently, a task may have up to 24 outstanding parent-offspring ASTs pending and up to 24 simultaneous Virtual Terminal units. Exceeding these limits produces a directive status of IE.UPN (insufficient pool). Raising or lowering these limits requires a patch to the SYSLIB module SPTBL.

7. Fortran subroutines are not re-entrant, since they contain static variables. Therefore, Fortran subroutines cannot be used in common between task and AST levels. Fortunately, the Fortran Exucutive directive calls are re-entrant, so they do not pose any problems when used within service routines.

3.3 Example: Fortran Requested Exit AST Usage

There are always occasions when a Fortran task must be aborted. However, this causes any output files to be locked, and usually the last few output records are lost because they were not written out to disk before the task aborted.

In another situation, a long-running task (such as a modelling program) may be aborted due to system considerations, or because the user wants to examine intermediate results produced by the task. It would be beneficial to be able to resume the task from the point where it was aborted. Previously, programming such a capability required user interaction at the terminal or a clever scheme using Send and Receive directives.

Using the Specify Requested Exit AST, a task becomes aware that it is being aborted, and it can take appropriate action to complete any partial output results, perhaps save its current data environment, and gracefully exit. Fortunately, since this AST is supported in a Fortran program, implementation is relatively simple.

In our example, we have two source files, MAIN.FTN, which contains the main program code, and ABORT.FTN, which contains the AST service routine. (It is wise to segregate AST service routines

from other sections of the program, since the AST service routines must be compiled with the /NOTRACE option.) The service routine communicates with the main program through a COMMON block. When the AST occurs, the service routine sets a flag in the COMMON block. When the main program checks the flag, it can then orderly shut down.

A skeletal outline of the code required might be:

File MAIN.FTN:

```
      Program MAIN
C
      External ABORT          ! Define AST Service Routine
      Common /ABOCOM/ Iflag
C
      Iflag = 0               ! Reset flag
      Call SREA(ABORT)        ! Arm the abort AST
C
C     The main loop follows.
C
 1000 Continue                ! Perform normal processing
C
C...  (Main loop contents)
C
      If (Iflag .eq. 0) Goto 1000  ! Check abort flag
C
C     Abort request fielded. Break out of loop
C
      Close (Unit=1)          ! Close possible output file
      Call Exit
      End
```

File ABORT.FTN:

```
      Subroutine ABORT
C
      Common /ABOCOM/ Iflag
C
      Iflag = 1               ! Set the Abort flag!
      Return                  ! And return from AST state
      End
```

To compile and build the program, the command sequence might be:

```
>F77 MAIN,MAIN=MAIN
>F77 ABORT,ABORT=ABORT/NOTR
>TKB MAIN=MAIN,ABORT,LB:[1,1]F77OTS/LB
```

3.4 Example: Unsolicited Character Input AST From Fortran

We perform various types of interactive signal processing on our PDP-11/70 in Menlo Park, using graphics terminals driven at 9600 baud. Typically, a set of traces are drawn on the screen in a 30 second period. We felt it was necessary to allow users to interrupt the display process to change parameters and redraw the screen. The mechanism we used to provide this capability is the Unsolicited Input Character AST mechanism of the terminal driver.

Since all of the analysis programs are written in Fortran, a special set of Macro routines were written to allow Fortran programs to arm and disarm the unsolicited input AST mechanism. These routines are listed in Appendix B.

A program arms the unsolicited input character by calling the TTYATA routine with the logical unit number assigned to the user's terminal and with the address of a Fortran AST service routine which is invoked when a character is intercepted. The TTYATA routine issues a QIO$ directive to the terminal with the IO.ATA function, which attaches the terminal.

The program can perform any Fortran READ or WRITE statements to the terminal while the AST is armed. The AST will not be dispatched, since a READ statement is a solicited read request. However, if the user should enter an unsolicited character when no READ statement is active, then the AST is dispatched. (Note that the terminal driver does not echo the unsolicited input character.)

The user's Fortran AST service routine is entered with two arguments: the ASCII value of the unsolicited character, and the task's waitfor mask. In our own applications, the AST service routine compares the input character with a set of legal input characters. If a match is found, the character is stored in a COMMON block. The service routine then exits via a RETURN statement.

The main program periodically examines the COMMON block while the AST is armed. When an input character is detected, the program takes appropriate actions based on the particular character entered. When the AST mechanism is no longer required, the TTYDET routine is called, which detaches the terminal and disarms the AST.

Examining the Macro code in Appendix B shows how the AST is handled and passed to the Fortran service routine. The IO.ATA QIO$ actually specifies a service routine within the Macro routines. This "primary" service routine saves the general purpose and FPP registers, sets up a standard Fortran argument list pointing to the character and the waitfor mask, and calls the Fortran service routine. When control returns to the Macro service routine, the registers are restored and the ASTX$ directive is issued.

3.5 Tips For Writing Specialized Fortran ASTs

To write an application in Fortran using an "unsupported" AST interface, the same procedures can be used that are shown in the Unsolicited Character AST example above. In fact, the same procedures are used by Fortran in implementing "supported" ASTs, such as the parent-offspring directives. The basic procedures to follow are:

1. Define the Fortran subroutine interface to be used to arm the AST mechanism.

2. Define the necessary trap-dependent AST parameters which will be passed to the Fortran AST service routine in the service routine's argument list.

3. Write the Macro-level, Fortran-callable AST arming mechanism routine.

4. Write the Macro "primary" AST service routine which actually fields the AST when it is dispatched. The service routine will save the general purpose and FPP registers, set up the Fortran argument block, and call the Fortran service routine. Upon return, the Macro routine restores the registers, performs any cleanup operations, and issues the ASTX$ directive.

The example listing in Appendix B provides a good example of the details involved. Further information regarding the Fortran OTS can be obtained from the Fortran Object Time System Manual supplied with the compiler.

4.0 CONCLUSIONS

The use of ASTs can be beneficial in a wide variety of applications. When real-time response is critical, ASTs provide the best event mechanism when a task is responsible for managing a large event-oriented environment.

Care should be used, however, in avoiding a difficult AST implementation where the benefits may be marginal. From my own experience, a programmer should ask the following questions before journeying into the world of ASTs:

1. Does my program spend a great deal of time waiting for event flags when it could be doing something else?

2. Do I need to be notified of an event for which event flags are useless, such as power fail recovery?

3. Is it worth the programming time to implement ASTs when event

flags will perform adequately?

Answering "yes" to any of the questions provides a good argument for using ASTs. By combining careful planning and programming, ASTs could be an important performance improvement to your application.

Appendix A

SAVFPP Routine - Save FPP Registers

The following routine must be called by a Fortran AST service routine if the service routine contains any floating point arithmetic or math library calls. The contents of all the FPP hardware registers are saved on the task's stack. Note that a coroutine linkage is used; when the service routine exits, the FPP registers are transparently restored.

The SAVFPP routine may be assembled in a straightforward manner, e.g.:

```
>MAC SAVFPP,SAVFPP/-SP=SAVFPP
```

The listing for the file SAVFPP follows:

```
        .TITLE  SAVFPP -- Save FPP Registers (AST Use)
        .IDENT  /01/
        .ENABL  LC
;+
;
;  SAVFPP -- Save FPP registers for calling routine and
;            restore registers as a coroutine.
;
;  Usage:
;
;          Call SAVFPP
;
;  This routine must be called by Fortran AST service routines
;  that perform floating point operations or call math library
;  routines.
;
;  When the routine calling SAVFPP issues a RETURN statement
;  (Fortran or Macro), the previous contents of the FPP
;  registers are restored.
;
;  This routine requires 27 decimal words of stack space.
;
;-

; Define FPP registers

F0=%0
F1=%1
F2=%2
F3=%3
F4=%4
F5=%5
```

```
SAVFPP::
        STFPS   -(SP)           ; Save FPP status
        SETD                    ; Set double precision
        STD     F0,-(SP)        ; Save FPP registers
        STD     F1,-(SP)
        STD     F2,-(SP)
        STD     F3,-(SP)
        LDD     F4,F0
        STD     F0,-(SP)
        LDD     F5,F0
        STD     F0,-(SP)
        MOV     62(SP),-(SP)    ; Get caller's address
        CALL    @(SP)+          ; Call the caller back

        SETD                    ; Restore - set double
        LDD     (SP)+,F0        ; Pop off registers
        STD     F0,F5
        LDD     (SP)+,F0
        STD     F0,F4
        LDD     (SP)+,F3
        LDD     (SP)+,F2
        LDD     (SP)+,F1
        LDD     (SP)+,F0
        LDFPS   (SP)+           ; Load FPP status
        TST     (SP)+           ; Pop stale address
        RETURN                  ; Return to caller's caller

        .END
```

Appendix B

TTYATA -- Fortran Callable Unsolicited Character AST

The following Macro listing contains the routines required to implement unsolicited character input ASTs from a Fortran program. The procedures can be generalized to other specialized Fortran AST applications.

Note that the module can be assembled to use either the Fortran OTS register save routines or a manual set of register save/restore routines.

```
        .TITLE  TTYATA
        .IDENT  /V2.0/
        .ENABL  LC
;
;+
; Subroutines TTYATA, TTYDET, ASTSV$.
;
; Subroutine TTYATA.
;
; TTYATA is a Fortran callable subroutine to link a Fortran
; completion routine to the unsolicited input character AST
; mechanism.
;
; Since AST service routines are serialized, the completion
; routine may not use any features that depend on the AST
; mechanism, such as I/O.
;
; Only one lun may be attached through this subroutine at a
; time, due to the nature of the Fortran linkage.  To change
; TTY's, you must first call TTYDET (see below) to clean up
; for the next call to TTYATA.  TTYATA marks itself "in use"
; if $DSW = IS.SUC, regardless of the I/O status returned.
;
;  Call TTYATA (lun,subr,iosb,ierr)
;
;    lun = Preassigned logical unit number for TTY attachment
;          with unsolicited input character AST, and the event
;          flag number used in the I/O calls.  The high order
;          byte may be used to specify any special subfunction
;          bits to be set, in addition to TF.AST, such as
;          TF.NOT for use with the full duplex terminal
;          driver. See section 2.3.1 in the I/O Drivers
;          Manual.
;
;    subr= Name of the Fortran completion routine. It must be
;          declared in an External specification statement,
;          and will be called by the AST service routine with
```

```
;           two arguments:  the unsolicited character (except
;           when left in the typeahead buffer) and the task's
;           waitfor mask (see section 2.3.3 in the Executive
;           Reference Manual), e.g.,
;
;              FORTRAN main program:
;
;                 External SUBR
;                     :
;                 Call TTYATA (5,SUBR,iosb,ierr)
;                     :
;                 Stop
;                 End
;
;              FORTRAN AST service routine:
;
;                 Subroutine SUBR (char, mask)
;                 Logical*1  char
;                 Integer*2  mask
;                     :
;                 Return
;                 End
;
;              Macro level AST service routine (ASTSV$):
;
;                     :
;                 Call SUBR (char,mask)
;                     :
;                 ASTX$S
;
;    iosb = Two word I/O status block, filled in at the
;           completion of the attach QIO.
;    ierr = Error return code:
;                0 = No errors
;                1 = Subroutine already in use
;               <0 = Directive status error code
;
; The QIO is issued synchronously with notification through
; event flag "lun".  The issuing task must first check the
; directive status word (ierr) before checking the I/O status
; block for errors.
;
;-
;
; *** Assembly Instructions ***
;
; TTYATA may be assembled to produce two versions:
;
;   1)  If the symbol NO$$F4 is defined, TTYATA performs its own
;       register saves and restores.  You must prefix the system
;       configuration file to determine whether the floating
;       point hardware is present:
;
```

```
;           >MAC TTYATA,TTYATA=[200,200]RSXMC/PA:1,[G,M]TTYATA
;
;       The completion routine will be called using the
;       standard Fortran linkage.
;
;   2)  If the symbol NO$$F4 is undefined, TTYATA assumes the
;       Fortran OTS routines are available to save and restore
;       all required registers:
;
;           >MAC TTYATA,TTYATA=TTYATA
;
        .MCALL  FILIO$,SPCIO$,IOERR$,QIOW$S
;
; Define I/O Functions and Error Codes
;
        FILIO$
        SPCIO$
        IOERR$

        .PSECT  CODE,RO,I       ; Pure code
        .ENABL  LSB

TTYATA::
        CLR     @10(R5)         ; ierr = 0
        TST     FORLUN          ; Check if in use already
        BNE     10$             ; If ne yes, fatal
        MOV     2(R5),R2        ; Get addr of LUN
        MOVB    (R2)+,R0        ; Pick up unit no.
        MOVB    (R2),R1         ; Pick up any subfunction bits
        BIS     _#IO.ATA,R1      ; Set function code for AST

        QIOW$S  R1,R0,R0,,6(R5),,<_#ASTSV$> ; Issue Attach QIO
        BCS     20$             ; If CS QIO failed
        MOV     R0,FORLUN       ; Mark subroutine in use
        MOV     4(R5),FORSUB    ; Save address of the FORTRAN
                                ;    completion routine
EXATA:  RETURN                  ; Return to caller

10$:    INC     @10(R5)         ; ierr = 1
        BR      EXATA

20$:    MOV     $DSW,@10(R5)    ; ierr = <Directive status word>
        BR      EXATA

        .PSECT  IDATA,RW,D      ; Impure data

FORLUN: .WORD   0               ; "in use" flag, contains LUN
FORSUB: .WORD   0               ; Fortran completion routine

        .DSABL  LSB
;+
;
; Subroutine TTYDET.
```

```
;
; TTYDET detaches TTY from lun and marks subroutine TTYATA as
; available for use again.
;
;    Call TTYDET(iosb,ierr)
;
;         iosb    = Same as above for TTYATA.
;         ierr    = Same as above for TTYATA, except ierr = 1
;                   indicates that no lun is currently attached.
;
;-
;

         .PSECT  CODE,RO,I        ; Pure code
         .ENABL  LSB

TTYDET::
         CLR     @4(R5)           ; ierr = 0
         MOV     FORLUN,R0        ; Get LUN in use
         BEQ     30$              ; If eq, fatal - not in use
         QIOW$S   #IO.DET,R0,R0,,2(R5) ; Detach tty from FORLUN
         BCS     40$              ; If CS QIO failure
         CLR     FORLUN           ; Mark TTYATA available

EXDET:   RETURN                   ; Return to caller

30$:     INC     @4(R5)           ; ierr = 1
         BR      EXDET

40$:     MOV     $DSW,@4(R5)      ; ierr = <Directive status word>
         BR      EXDET

         .DSABL  LSB
;+
;
; Subroutine ASTSV$.
;
; ASTSV$ is the TTY unsolicited input character
; AST service routine.
;
; On entry, the stack contains:
;
;         SP+10  Event flag mask word
;         SP+06  PS of the task prior to AST
;         SP+04  PC of the task prior to AST
;         SP+02  Task's Directive status word
;         SP+00  The unsolicited character in the low byte
;                (if TF.NOT not set); optional parameter 2
;                in the high byte (not used)  (See section
;                2.3.2.1 in the I/O Drivers Manual.)
;
; Before exit, the unsolicited character must be popped
; off the stack.
```

```
;
; To exit, invoke ASTX$S:
;
;           TST     (SP)+
;           ASTX$S
;
; The Fortran completion subroutine is called with the
; unsolicited input character as the first argument, and the
; event flag mask as the second argument (to replace the value
; on the stack before dismissing the AST).
;
;   *** Fortran OTS Routines Used ***
;
;   MODULE:  FPPUTI    ENTRY:   $SVFPP    FPP register save
;            SAVRG              $SAVPO    General register save
;
;-

        .MCALL  ASTX$S
        .PSECT  CODE,RO,I       ; Pure code

ASTSV$::
        MOV     SP,ARGLST+2     ; Move char ptr into arg list
        MOV     SP,ARGLST+4     ; Move EF mask ptr into arg list
        ADD     _#10,ARGLST+4    ; Correct for stack offset

        .IF DF  NO$$F4          ; Manual register save

        CALL    F4PSR$          ; Save R0-R5, F0-F5

        .IFF    ; DF NO$$F4     ; Fortran OTS save

        MOV      _#EXAST,-(SP)   ; Force coroutine return to us
        CALL    $SAVPO          ; Save general registers
        CALL    $SVFPP          ; Save FPP registers

        .IFTF   ; DF NO$$F4

        MOV      _#ARGLST,R5     ; Point to arg list
        CALL    @FORSUB         ; Call user AST routine

        CALL    @(SP)+          ; Coroutine call to restore regs

        .IFF    ; DF NO$$F4

        RETURN                  ; OTS coroutine restore of R0-R5
                                ;   (Returns control to EXAST)

        .ENDC   ; DF NO$$F4

EXAST:  TST     (SP)+           ; Pop character off stack
        ASTX$S                  ; Exit AST service routine
```

```
        .PSECT  IDATA,RW,D      ; Impure data

ARGLST: .WORD   2               ; 2 arguments for Fortran
        .WORD   0               ; 1st points to unsolicited char
        .WORD   0               ; 2nd points to event flag mask

        .IF DF  NO$$F4          ; Register save for no OTS

        .PSECT  CODE,RO,I       ; Pure code

F4PSR$: MOV     R0,-(SP)        ; Save registers for Fortran use
        MOV     R1,-(SP)
        MOV     R2,-(SP)
        MOV     R3,-(SP)
        MOV     R4,-(SP)
        MOV     R5,-(SP)

STKDEP=14

        .IF DF  F$$LPP

F0=%0
F1=%1
F2=%2
F3=%3
F4=%4
F5=%5

        STFPS   -(SP)           ; Save FPP status and registers
        SETD
        STD     F0,-(SP)
        STD     F1,-(SP)
        STD     F2,-(SP)
        STD     F3,-(SP)
        LDD     F4,F0
        STD     F0,-(SP)
        LDD     F5,F0
        STD     F0,-(SP)

STKDEP=STKDEP+62

        .ENDC   ; DF F$$LPP

        MOV     STKDEP(SP),-(SP) ; Get callers return address
        CALL    @(SP)+          ; Coroutine return to caller

        .IFT    ; DF NO$$F4

        MOV     (SP)+,STKDEP(SP) ; Load return address in slot

        .IF DF  F$$LPP

        SETD                    ; Restore FPP regs and status
```

```
        LDD     (SP)+,F0
        STD     F0,F5
        LDD     (SP)+,F0
        STD     F0,F4
        LDD     (SP)+,F3
        LDD     (SP)+,F2
        LDD     (SP)+,F1
        LDD     (SP)+,F0
        LDFPS   (SP)+

        .ENDC   ; DF F$$LPP

        MOV     (SP)+,R5        ; Restore general registers
        MOV     (SP)+,R4
        MOV     (SP)+,R3
        MOV     (SP)+,R2
        MOV     (SP)+,R1
        MOV     (SP)+,R0

        RETURN

        .ENDC   ; DF NO$$F4

        .END
```

DECUS

DECUS SUBSCRIPTION SERVICE
DIGITAL EQUIPMENT COMPUTER SOCIETY
249 NORTHBORO ROAD, (BPO2)
MARLBORO, MA 01752

Bulk Rate
U.S. Postage
PAID
Permit No. 18
Leominster, MA
01453

MOVING OR REPLACING A DELEGATE?

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

( ) Change of Address
( ) Delegate Replacement

DECUS Membership No.: _____
Name: _____
Company: _____
Address: _____
_____
State/Country: _____
Zip/Postal Code: _____
Phone No.: _____

Mail to: **DECUS - Attn: Subscription Service**
249 Northboro Road, (BPO2)
Marlboro, MA 01752 USA