# RSX

# MULTI-TASKER

JUNE 1985

# RSX MULTITASKER

## Table of Contents

# From the Editor

The Multitasker will undergo a format change next month. All SIG newsletters published by DECUS will be combined into a single monthly publication. Each SIG will have its own section. The RSX SIG will have a monthly Multitasker "section" in the combined DECUS newsletter.

The combined newsletter will be partially subsidized by DECUS. The yearly subscription fee will be $35.00. All current subscribers to the Multitasker will receive for the remainder of their subscription the combined newsletter at no extra charge. When renewal time arrives, the $35.00 fee will entitle the subscriber to twelve issues of the combined newsletter which will contain material from all SIGs (based on their publication schedule). As stated, the RSX SIG will continue to publish material monthly.

Since many people subscribe to more than one newsletter, the combining of newsletters will save many for most.

The procedure to contribute to the Multitasker remains the same. Machine readable copy can be sent to DECUS or directly to me at the address below. All media will be returned. I can accept the following media and formats:

| | | |
|---|---|---|
| Magnetic Tape: | 800, 1600, 6250 BPI | - PIP, BRU, FLX |
| | 1600, 6250 BPI | - VMS BACKUP |
| Floppy Disk: | RX01, RX02 | - ODS-1, ODS-2 |
| TU58 Tape: | | |

Dominic DiNollo
Loral Electronic Systems
Engineering Computer Center
Ridge Hill
Yonkers, New York 10710

(914) 698-2500 ext. 2210

# SIG News

## Nominations for the RSX SIG Executive Committee

The RSX SIG Executive Committee consists of five elected members, three appointed members, a representative from the DECUS staff and a representative from the RSX Product Management. Nominations are now open for the five elected positions. RSX SIG members may be nominated for the elected positions by submission of a petition containing the nominee's name and address. The petition must be signed by at least five RSX SIG members. Please submit any nominations to:

Elizabeth Bailey
Tennessee Valley Authority
222 CEB
Muscle Shoals, Ala., 35660

Nominations should be received by August 1st.

A ballot containing a brief summary of each candidate's qualifications and a photograph of each candidate will be mailed to all members of the RSX SIG in August or September. The new members of the executive committee will be announced at the fall DECUS symposium. The term of office is two years, with the new committee members assuming office on January 1st.

# The RSX System Manager

## Undeleting RSX Files and File Security

In last month's column, there was one issue regarding undeleting files I neglected to mention. In particular, the UND program performs no file protection checks when a file is recovered.

This can pose some serious system security problems when UND is permanently installed. Specifically, non-privileged users can effectively resurrect any deleted file, including restricted files (e.g., MAIL files). There are two approaches to remedy this

problem, if your system deals with sensitive information.

First, don't install UND. Since the program is privileged, it cannot be invoked (installed) by non-privileged users. This is probably the most satisfactory solution, since it may be unwise to allow any user to try and undelete his/her files. The disadvantage is that a system manager or system programmer with privileged access will have to be located when a file needs to be recovered.

Second, UND can be modified to check the file protection bits in the file header of a deleted file, and disallow recovery of a file which does not have read access allowed to the user who invoked UND. This is the more general solution, but I know of no implementation at this time.

## Some Useful Print Despooler Modifications

The main topic this month deals with modifications to the RSX-11M/M-Plus line printer despooler task, LPP, that we have made over the years at USGS.

Since LPP is a privileged task, and the modifications discussed here can potentially corrupt or crash a running system, the United States Geological Survey, DECUS, Digital Equipment Corporation, and I disclaim any and all responsibility for the correctness or reliabilty of the modifications discussed herein. (Of course, I would like to be notified of any errors.) The modifications discussed here apply to RSX-11M-Plus Version 2.1 and RSX-11M Version 4.1.

Following brief descriptions of the modifications are the .COR files and the commands for incorporating the modifications into RSX-11M-Plus, followed by the .PAT files and commands for incorporation under RSX-11M. Be sure to follow the appropriate instructions for your system.

### Modification 1: Customized Flag Page Headings

Many requests dealing with the print spooler are for modifying the text that appears in the "corners" of the job and file flag pages. As distributed, these headings are composed of the ASCII string "01234567..." and the name of the operating system.

Modifying and customizing the flag page headings is relatively straightforward. The strings used for these headings are used as input to the System Library routine, $EDMSG, to format the informational lines on the flag pages. These strings are located in two modules: JBINI (for job flag pages) and FLINI (for file

flag pages). For M-Plus systems, one can modify the MACRO-11 source files for these modules in UIC [121,10] on the distribution disk, reassemble and relink LPP. For RSX-11M, the object files for these routines must be patched with alternative strings.

Another desirable modification deals with narrow forms. LPP normally chops off the left part of the left header string (TXTFMT) when the output is going to a narrow printer. With the addition of a substitute string (SHFMT), an abbreviated header string can be used instead. When modified, the SHFMT string will be used if the buffer size of the output device is less than 132 characters.

The length of these strings can vary slightly, but remember that you have 132 columns at most to work with. The headers we use have the following lengths:

| | |
|---|---|
| TXTFMT (Left side) | 42 characters |
| SHFMT (Left side) | 17 characters |
| ENDFMT (Right side) | 22 characters |

Since you will want to put your own flag page headings into the despooler, the .COR and .PAT files contain the strings we use at USGS, to serve as an example. You will probably have to do some experimenting with these strings.

## Modification 2: Checking for Bad Filetypes

It's amazing how fast a box of printer paper disappears when a task image file is spooled to the printer. At USGS, we grew tired of reminding our users about the dangers of a wildcard PRINT command, when non-text files would be spooled along with everything else.

In the FILPRO module of LPP, the next file to be printed is prepared for spooling. This module has been modified so that a check is performed to reject files with particularly nasty filetypes. Obvious filetypes that should be rejected out-of-hand are .TSK, .OBJ, and .OLB. Since different systems may have additional data types which should not be allowed, the correction to FILPRO is driven off an expandable table.

When a file is found to be among the class of illegal filetypes, LPP will still print the job and file flag pages, but will follow with an error message on both the printer and the console terminal with an error code of -40 (illegal record size). The print job will continue normally if more files remain to be printed.

When adding filetypes to the illegal filetype table, remember that all files with that filetype will be rejected. The only workaround, once the patch is installed, is to copy the file using a different filetype.

Modification 3: Console Logging of Print Jobs

There are many advantages to keeping a running log of print jobs on the console terminal. A system manager may want to locate the person who left the line printer offline, or it may be desirable to have a record of users who make heavy use of the printer.

Modifying LPP to print a message on the console after each print job turns out to be relatively easy. For M-Plus systems, the user and jobname of each print job is saved in a buffer in the root segment; for 11M systems (and in some cases in M-Plus), information needs to be collected and saved when the job is started. Modifications to the JBINI and JOBEND modules, and a modification to the [1,20]LPPBLD.BLD file accomplishes this function. The modification to LPPBLD.BLD, which in turn modifies [1,24]LPPBLD.ODL, is required to bring a console logging routine into the overlay segment containing the JOBEND routine.

The console message consists of a time stamp, the jobname, the user or account name, and the name of the printer device.

How to Incorporate the Modifications

To incorporate all of the modifications described above, simply follow all of the steps outlined for your system type (11M or M-Plus). If you wish to incorporate only one or two modifications, use the following table to determine which LPP modules you will have to modify:

| | FILPRO | FLINI | JBINI | JOBEND | LPPBLD |
|---|---|---|---|---|---|
| Custom Flag Pages | | X | X | | |
| Filetype Checks | X | | | | |
| Console Logging | | | X | X | X |

Try to use the RSX UIC conventions for patching and correcting system components. Correction files should be placed in [*,40] directories. Be sure to create a new version of the LPP object library to work on, and use PIP to protect the original copies of the source and object files.

* * * * * Corrections for RSX-11M-Plus LPP Despooler * * * * *

Correction file [121,40]FILPRO.COR:

```
FILPRO.MAC;2/AU/-BF=[121,10]FILPRO.MAC;1
-/.IDENT/,.
        .IDENT  /1.04A/
-/VERSION/,.
; VERSION 1.04A
```

```
-/LOCAL...DATA/
-/DELETE/

; Illegal filetype table.
; This table contains filetypes whose files typically drive
; printers crazy (e.g., any non-text file). The table can
; be extended to any length, as long as it is terminated by
; a null word.

ILLTAB: .RAD50  /TSK/   ; Don't print .TSK files
        .RAD50  /OBJ/   ; Don't print .OBJ files
        .RAD50  /OLB/   ; Don't print .OLB files
        .WORD   0       ; Terminate the list

-/LOA:/
-/N.FTYP(R1)/+1

        MOV     #ILLTAB,R5      ; Get illegal file types
20$:    TST     (R5)            ; Reached end of table?
        BEQ     30$             ; If eq yes - file passes
        CMP     (R5)+,N.FTYP(R1) ; Is this file a no-no?
        BNE     20$             ; If ne no, try next one
        MOVB    #IE.RBG,F.ERR(R0) ; Return bad record size
        SEC                     ; Set error return
        BR      LOADED          ; And finish
30$:                            ; Reference label
/


Correction file [121,40]FLINI.COR:

FLINI.MAC;2/AU/-BF=[121,10]FLINI.MAC;1
-/.IDENT/,.
        .IDENT  /03.01A/
        .ENABL  LC
-/TXTFMT:/,/.ENDC/

; The following appears on the left side of the banner header.
; This must be ASCIZ and should end with "%4S"

TXTFMT: .ASCII  /*** National Strong Motion Data Center ***/
        .ASCIZ  /%4S/

; The following alternate to TXTFMT is used on narrow forms.
; This must be ASCIZ and should end with "%4S"

SHFMT:  .ASCII  /*** N S M D C ***/
        .ASCIZ  /%4S/

-/ENDFMT:/,/SPCFD:/

; The following appears on the right side of the banner header.
; It must be .ASCIZ and should begin with "%2S"
```

```
ENDFMT: .ASCII  /%2S/
        .ASCIZ  /Menlo Park, California/

; The following strings are lower case equivilents of
; standard LPP flag page strings

COPFMT: .ASCIZ  /Copy %D of %D/
DELFMT: .ASCIZ  /Deletion %VA/
NOT:    .ASCII  /not /
SPCFD:  .ASCII  /specified/
-/#RSXFMT/,.
        MOV     #SHFMT,R1       ;Shorten the line - small printer
/


Correction file [121,40]JBINI.COR:

JBINI.MAC;2/AU/-BF=[121,10]JBINI.MAC;1
-/.IDENT/,.
        .IDENT  /02.01A/
        .ENABL  LC
-/TXTFMT:/,/MNYFMT:/

; The following appears on the left side of the banner header.
; Note that this must be an .ASCIZ string and should end with
; "%4S"

TXTFMT: .ASCII  /*** National Strong Motion Data Center ***/
        .ASCIZ  /%4S/

; The following alternate to TXTFMT is used on narrow forms.
; It too, must be .ASCIZ and should end with "%4S"

SHFMT:  .ASCII  /*** N S M D C ***/
        .ASCIZ  /%4S/

; The following appears on the right side of the banner header.
; It must be .ASCIZ and should begin with "%3S"

ENDFMT: .ASCII  /%3S/
        .ASCIZ  /Menlo Park, California/

; The following are upper/lower case equivalents of the
; standard banner page information lines.

TIMFMT: .ASCIZ  /%3S%Y%3S%3Z/

JOBFMT: .ASCIZ  /%VA%3R - Limit:%D pp./
NOLIM:  .ASCIZ  /%VA%3R - No page limit/

FORYES: .ASCIZ  /Form #%D - %D lines per page/
FORNO:  .ASCIZ  /Form #%D - Normal hardware forms/
IMPYES: .ASCIZ  /Form feed implied after %D lines/
```

```
IMPNO:  .ASCIZ  /No implied form feed/
FILFMT: .ASCIZ  /%X/
FIDFMT: .ASCIZ  "/FI:%P:%P"
MNYFMT: .ASCIZ  /Job contains %D filespecs/

; The following three lines are required if console
; logging of print jobs will be performed.

SAVSTR: .ASCIZ  /%1O<%VA%1O>%6A/
        .EVEN
SAVARG: .BLKW   3

-/#RSXFMT/,.
        MOV     #SHFMT,R1       ; Narrow forms - shorter line
-/70$:/

; The following code saves the account and jobname fields in
; a save area in the root, which will be used later to output
; a message to the console terminal

        MOV     #SAVSTR,R1      ; Get address of input string
        MOV     #FLBUF,R0       ; Get save area
        MOV     #SAVARG,R2      ; Get argument area
        MOV     UICDSD,(R2)     ; Save size of account string
        MOV     R4,2(R2)        ; Save address of account string
        MOV     R4,4(R2)        ; Copy into next word
        ADD     (R2),4(R2)      ; Get address of jobname
        CALL    $EDMSG          ; Save it all for later
/


Correction file [121,40]JOBEND.COR:

JOBEND.MAC;2/AU/-BF=[121,1O]JOBEND.MAC;1
-/.IDENT/,.
        .IDENT  /02.01A/
        .ENABL  LC
-/.MCALL/+1
        .MCALL  GLUN$

$COLUN  ==      2               ; Console LUN
$COEFN  ==      9.              ; Console event flag

-/USTP:/

GLUN:   GLUN$   LPLUN,GLUNBF    ; Get spooled device info
GLUNBF: .BLKW   6               ; GLUN$ buffer

; $EDMSG Input string and argument block

COARG:  .WORD   FLBUF+1O.       ; Address of jobname
        .WORD   GLUNBF+G.LUNA   ; Address of device name
        .WORD   GLUNBF+G.LUNU   ; Address of device unit
```

```
        .WORD   FLBUF           ; Address of user name

COSTR:  .ASCII  /Print job - %6A%4S/
        .ASCIZ  /completed on %2A%B: User %10A/

COBUF:  .BLKB   60.             ; Console message buffer

-/40$:/
        DIR$    #GLUN           ; Get output device info
        MOV     #COBUF,R0       ; Point to output buffer
        MOV     #COSTR,R1       ; Point to input string
        MOV     #COARG,R2       ; Point to argument block
        CALL    $EDMSG          ; Format the output string
        MOV     R1,-(SP)        ; Push length of output string
        MOV     #COBUF,-(SP)    ; Push beginning address
        CALL    $PRCO           ; Send message to the console
/
```

Correction file [1,40]LPPBLD.COR:

```
LPPBLD.BLD;2/-AU=[1,20]LPPBLD.BLD;1
-/.DATA IDENT=/,.
.DATA IDENT=03.02A
-/.DATA J:/,.
.DATA J:        .FCTR   '$LI'LPP/LB:INIT:JOBSTR:JOBEND:IOPRT-J1
.DATA J1:       .FCTR   '$LI'QMG/LB:PRCO
/
```

To incorporate these correction files into the M-Plus LPP despooler, follow instructions in the Release Notes, or use the following command sequence:

```
>SET /UIC=[121,10]                  ! Correct source files
>SLP @[121,40]FILPRO.COR
>SLP @[121,40]FLINI.COR
>SLP @[121,40]JBINI.COR
>SLP @[121,40]JOBEND.COR
>SET /UIC=[121,24]                  ! Assemble sources
>MAC FILPRO=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[121,10]FILPRO
>MAC FLINI =[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[121,10]FLINI
>MAC JBINI =[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[121,10]JBINI
>MAC JOBEND=[1,1]EXEMC/ML,[11,10]RSXMC/PA:1,[121,10]JOBEND
>SET /UIC=[1,24]
>PIP LPP.OLB/NV/CO=LPP.OLB;1       ! Update library
>LBR LPP/RP/-EP=[121,24]FILPRO,FLINI,JBINI,JOBEND
Module "FILPRO" replaced

Module "FLINI " replaced
```

```
Module "JBINI " replaced

Module "JOBEND" replaced

>SET /UIC=[1,20]
>SLP @[1,40]LPPBLD.COR            ! Modify build file
```

Now either invoke SYSGEN to rebuild privileged tasks, or manually modify the current .CMD and .ODL files for LPP in [1,24] and rebuild LPP. After task building, test the new version of LPP (be wary of crashes!), and then replace the old version by using VMR.

* * * * * Corrections for RSX-11M LPP Despooler * * * * *

Correction file [121,40]FILPRO.PAT:

```
        .TITLE  FILPRO
        .ENABL  LC
;
;       Original   file checksum: 042135
;       Correction file checksum: (depends on filetype table)
;
; Check for filetypes that strictly aren't text files,
; and reject them with an illegal record error.
;
        .IDENT  /1.04A/

        .PSECT
.$ = .

. = .$ + 410
        JMP     $PATB           ; Go check for illegal filetypes

        .PSECT  $PATBD,RO,D

ILLTAB: .RAD50  /OBJ/           ; Table of illegal filetypes
        .RAD50  /TSK/
        .RAD50  /OLB/
        .WORD   0               ; Terminate table with a zero!

        .PSECT  $PATBI,RO,I

$PATB:  MOV     #ILLTAB,R5      ; Point to table
10$:    TST     (R5)            ; End of table?
        BEQ     20$             ; If eq yes
        CMP     (R5)+,N.FTYP(R1) ; Match an illegal file type?
        BEQ     30$             ; If eq yes, return with CC=1
        BR      10$             ; Check next entry

20$:    CALL    .FIND           ; Lookup the file
```

```
        BR      40$             ; Return with .FIND status
30$:    MOVB    #IE.RBG,F.ERR(R0) ; Return bad record size
        SEC                     ; Set CC=1
40$:    RETURN                  ; And return to caller

        .END
```

Correction file [121,40]FLINI.PAT:

```
        .TITLE  FLINI
        .ENABL  LC
;
;       Original   file checksum: 016554
;       Correction file checksum: (depends on header text)
;
        .IDENT  /03.01A/
        .PSECT
.$ = .

. = .$ + 236
        MOV     #TXTFMT,R1      ; Get text for left side of flag page

. = .$ + 252
        MOV     #SHFMT,R1       ; Get left side text for narrow forms
. = .$ + 464
        MOV     #ENDFMT,R1      ; Get right side text

        .PSECT  $PATAD,RW,D

        .NLIST  BEX

; The following appears on the left side of the banner header.
; This must be ASCIZ and should end with "%4S"

TXTFMT: .ASCII  /*** National Strong Motion Data Center ***/
        .ASCIZ  /%4S/

; The following alternate to TXTFMT is used on narrow forms.
; This must be ASCIZ and should end with "%4S"

SHFMT:  .ASCII  /*** N S M D C ***/
        .ASCIZ  /%4S/

; The following appears on the right side of the banner header.
; It must be .ASCIZ and should begin with "%2S"

ENDFMT: .ASCII  /%2S/
        .ASCIZ  /Menlo Park, California/

        .EVEN
        .END
```

Correction file [121,40]JBINI.PAT:

```
        .TITLE  JBINI
        .ENABL  LC
;
;       Original   file checksum: 075724
;       Correction file checksum: (depends on string option)
;
; Customize job flag page for site installation
;
        .IDENT  /02.01A/
        .PSECT
.$ = .
INFO = .$ + 770                 ; Routine called from patch

. = .$ + 464
        MOV     #TXTFMT,R1      ; Get left side of flag page

. = .$ + 500
        MOV     #SHFMT,R1       ; Get left side: narrow forms

. = .$ + 672
        MOV     #ENDFMT,R1      ; Get right side text

; The following two lines are required for console logging

. = .$ + 730
        CALL    $PATC           ; Save account/job info

        .PSECT  $PATAD,D,RW

; The following three lines are required if console
; logging of print jobs will be performed.

SAVSTR: .ASCIZ  /%10<%VA%10>%6A/
        .EVEN
SAVARG: .BLKW   3

; The following appears on the left side of the banner header.
; Note that this must be an .ASCIZ string and should end with
; "%4S"

TXTFMT: .ASCII  /*** National Strong Motion Data Center ***/
        .ASCIZ  /%4S/

; The following alternate to TXTFMT is used on narrow forms.
; It too, must be .ASCIZ and should end with "%4S"

SHFMT:  .ASCII  /*** N S M D C ***/
        .ASCIZ  /%4S/

; The following appears on the right side of the banner header.
; It must be .ASCIZ and should begin with "%3S"
```

```
ENDFMT: .ASCII  /%3S/
        .ASCIZ  /Menlo Park, California/
        .EVEN

; The following code saves the account and jobname fields in
; a save area in the root, which will be used later to output
; a message to the console terminal

        .PSECT  $PATCI,RO,I

$PATC:  CALL    INFO            ; Format the other lines
        MOV     #SAVSTR,R1      ; Get address of input string
        MOV     #FLBUF,R0       ; Get save area
        MOV     #SAVARG,R2      ; Get argument area
        MOV     UICDSD,(R2)     ; Save size of account string
        MOV     R4,2(R2)        ; Save address of account string
        MOV     R4,4(R2)        ; Copy into next word
        ADD     (R2),4(R2)      ; Get address of jobname
        CALL    $EDMSG          ; Save it all for later
        RETURN                  ; Return from patch code

        .END


Correction file [121,40]JOBEND.PAT:

        .TITLE  JOBEND
        .ENABL  LC
;
;       Original   file checksum: 112617
;       Correction file checksum: 044570
;                  (MACRO-11 Version Y05.00d)
;
; Output message to console at end of job
;
        .IDENT  /02.01A/

        .PSECT
.$ = .

. = .$ + 302
        JMP     $PATA           ; Go print message

        .PSECT  $PATAD,D,RW
        .MCALL  GLUN$,DIR$

$COLUN  ==      2               ; Console LUN
$COEFN  ==      9.              ; Console event flag

GLUN:   GLUN$   LPLUN,GLUNBF    ; Get spooled device info
GLUNBF: .BLKW   6               ; GLUN$ buffer

; $EDMSG Input string and argument block
```

```
COARG:  .WORD   FLBUF+10.       ; Address of jobname
        .WORD   GLUNBF+G.LUNA   ; Address of device name
        .WORD   GLUNBF+G.LUNU   ; Address of device unit
        .WORD   FLBUF           ; Address of user name

COSTR:  .ASCII  /Print job - %6A%4S/
        .ASCIZ  /completed on %2A%B: User %10A/

COBUF:  .BLKB   60.             ; Console message buffer
        .EVEN

        .PSECT  $PATAI,I,RO

$PATA:: DIR$    #GLUN           ; Get output device info
        MOV     #COBUF,R0       ; Point to output buffer
        MOV     #COSTR,R1       ; Point to input string
        MOV     #COARG,R2       ; Point to argument block
        CALL    $EDMSG          ; Format the output string
        MOV     R1,-(SP)        ; Push length of output string
        MOV     #COBUF,-(SP)    ; Push beginning address
        CALL    $PRCO           ; Send message to the console
        RETURN                  ; Return from JOBEND
        .END
```

```
Correction file [1,40]LPPBLD.COR:

LPPBLD.BLD;2/-AU=[1,20]LPPBLD.BLD;1
-/.DATA J:/,.
.DATA J:        .FCTR   '$LI'LPP/LB:INIT:JOBSTR:JOBEND:IOPRT-J1
.DATA J1:       .FCTR   '$LI'QMG/LB:PRCO
/
```

To incorporate the changes, follow the instructions in the Release Notes for performing object level modifications to system components, or use the following command sequence as a guide:

```
>SET /UIC=[121,24]
>MAC FILPRO.POB=[121,40]FILPRO.PAT       ! Assemble patches
>MAC FLINI .POB=[121,40]FLINI .PAT
>MAC JBINI .POB=[121,40]JBINI .PAT
>MAC JOBEND.POB=[121,40]JOBEND.PAT
>LBR FILPRO.OBJ;1=[1,24]LPP;1/EX:FILPRO   ! Extract originals
>LBR FLINI .OBJ;1=[1,24]LPP;1/EX:FLINI
>LBR JBINI .OBJ;1=[1,24]LPP;1/EX:JBINI
>LBR JOBEND.OBJ;1=[1,24]LPP;1/EX:JOBEND
>PAT FILPRO.OBJ;2=FILPRO.OBJ;1/CS:042135,FILPRO.POB
>PAT FLINI .OBJ;2=FLINI .OBJ;1/CS:016554,FLINI .POB
>PAT JBINI .OBJ;2=JBINI .OBJ;1/CS:075724,JBINI .POB
>PAT JOBEND.OBJ;2=JOBEND.OBJ;1/CS:112617,JOBEND.POB/CS:44570
>SET /UIC=[1,24]
```

```
>PIP LPP.OLB/NV/CO=LPP.OLB;1                ! Update library
>LBR LPP/RP=[1211,24]FILPRO,FLINI,JBINI,JOBEND/-EP
Module "FILPRO" replaced

Module "FLINI " replaced

Module "JBINI " replaced

Module "JOBEND" replaced

>SET /UIC=[1,20]
>SLP @[1,40]LPPBLD.COR                      ! Correct build file
```

Now either manually modify [1,24]LPPBLD.ODL and rebuild LPP, or reinvoke SYSGEN2 to rebuild the privileged tasks. Be sure to try out these modifications, keeping in mind that they may crash your system. Good luck!

**********

Please send questions, comments, ideas and submissions for this column to the following address:

Gary Maxwell
U.S.G.S. M/S 977
345 Middlefield Road
Menlo Park, CA 94025

## The Bag of Tricks: MACRO-11

Bruce R. Mitchell
Machine Intelligence and Industrial Magic
PO Box 601
Hudson, WI 54016

This column covers MACRO-11 bag-of-tricks routines, as stated in previous issues of the Multi-Tasker. encouraged to submit their favorite routines to the Multi-Tasker so that these useful, interesting, or just plain bizarre tricks can be put out before the SIG in general for the admiration and edification of all.

In this month's column, we have something which is related to the previous processor identification routine - a routine to determine if the host CPU posesses a floating point processor (FPP).

Again, this routine lets those of us who do commercial coding ride herd on persons of questionable morality who would make unauthorized copies of our children who are out in the big cold world. If you license a program to run on a machine with floating point, then by damn, you can use this to make sure it runs on a machine with floating point and nothing else.

Another use, of course, is to ensure the presence of an FPP before starting up a mainline program which uses FPP instructions. It is always embarrassing to have a program trap out nastily for no apparent reason.

This routine was written using the previous processor ID routine as a template. It should be noted that any attempt to use it with a debugging tool such as ODT will give very, very strange results.

Because the author tends to separate data and code structures in his programs, the labels have been made so that the data and code can be readily separated within a single source file.

The output from TSTFPP is an integer in R0 which flags the presence or absence of an FPP. This can be used directly in the mother program to compare against a stored value.

```
    .SBTTL  TSTFPP  Test for FPP


;   TTTTTTTTTT   SSSSSSSSS  TTTTTTTTTT  FFFFFFFFFF  PPPPPPPPP   PPPPPPPPP
;   TTTTTTTTTT   SSSSSSSSS  TTTTTTTTTT  FFFFFFFFFF  PPPPPPPPP   PPPPPPPPP
;       TT      SS              TT      FF          PP      PP  PP      PP
;       TT      SS              TT      FF          PP      PP  PP      PP
;       TT       SSSSSSS        TT      FFFFFFFF    PPPPPPPPP   PPPPPPPPP
;       TT       SSSSSSS        TT      FFFFFFFF    PPPPPPPPP   PPPPPPPPP
;       TT             SS       TT      FF          PP          PP
;       TT             SS       TT      FF          PP          PP
;       TT             SS       TT      FF          PP          PP
;       TT             SS       TT      FF          PP          PP
;       TT       SSSSSSS        TT      FF          PP          PP
;       TT       SSSSSSS        TT      FF          PP          PP


;   TSTFPP - Test Host System for Floating Point Processor
;
;   This subroutine attempts to determine if which the host system
;   processor is equipped with a floating point processor (FPP option).
;
;   Inputs:  None
;
;   Outputs:  R0 - 1 for no FPP, 0 for FPP present
;             Carry clear on success, carry set on failure
;
```

```
;  Register dispositions:  R0 destroyed
;
;  Variable dispositions:  TRPTBL modified


;  SST trap table

TRPTBL:     .WORD   0, 0, 0, 0, RSRVED, 0, 0, 0
TRTLEN = . - TRPTBL


;  Trap catcher

RSRVED:     INC     R0
   RTT


;  Subroutine code

TSTFPP:     CLR     R0                      ; Clear the trap flag register
   SVTK$S   #TRPTBL, #TRTLEN          ; Set up trapping

;  We shall attempt to copy the floating condition codes into the PSW

   CFCC                               ; Copy floating condition codes

;  Clear the trap catcher and return to the caller showing success

   CLR      TRPTBL+8.                 ; Clear reserved instruction vector
   SVTK$S   #TRPTBL, #TRTLEN          ; Clear trapping

   RETURN                             ; Return to the caller without prejudice


   .END
```

# More Useful Extensions to FORTRAN

Terry Medlin
Survey Sampling, Inc.

At our shop, we manipulate lots of direct access files. More often than not, we need to know how many records are in the file so we can append information or simply process the records in a loop. As you may know, FORTRAN does not support an END= option on direct

access reads.

The way we accomplish this is through a routine called LUNRNO which is treated as an INTEGER*4 function in our F77 code. This routine works for FIXED length FCS files that do NOT have the NOSPANBLOCK attribute. An example FORTRAN shell would be:

```
   INTEGER*4 LRECL                  !NUMBER OF RECORDS
   INTEGER*4 LUNRNO                 !FUNCTION
C
   OPEN(UNIT=1,NAME=.....
C
   LRECL=LUNRNO( 1 )                ! 1 IS THE LUN
```

Note that if you open the file TYPE='UNKNOWN',ACCESS='APPEND' then you can create the file and keep adding records via continued use of a program.

The code for the routine follows:

```
   .TITLE  LUNRNO - GET NO. OF RECORDS IN A FILE
   .IDENT  /X01/
;+
;  FUNCTION TO GET NO. OF RECORDS IN A FILE.
;  RETURNS A DOUBLE WORD INTEGER IN FORTRAN
;
;  N=LUNRNO(LUN) WHERE LUN IS LOGICAL UNIT NO.
;
;  DOES NOT CURRENTLY WORK FOR FILES WITH NOSPANBLOCK SET
;  ONLY WORKS CORRECTLY WITH FIXED LENGTH RECORD FILES
;
; MODIFICATION LOG
;
;  TPM 29-AUG-84
;
;  REPAIRED CODE TO RETURN 0 RECORDS RATHER THAN -1 FOR EMPTY FILE
;
;  MADE CODE USEFUL FOR VARIABLE LENGTH FILES AS FOLLOWS:
;          EMPTY FILES RETURNS A ZERO
;          NON-EMPTY RETURNS A ONE (1)
;
;  THUS CAN BE USED TO SEE IF FILE IS EMPTY FOR ALL FILES
;-
   .PSECT  $$LBCD,RO,I,CON,REL
   .GLOBL  $FCHNL,$OTSV
LUNRNO::
   MOV     @2(R5),R2        ;GET LUN
   MOV     @#$OTSV,R3       ;GET WORK AREA ADDRESS
   JSR     PC,$FCHNL        ;FIND FDB ADDRESS
   MOV     R0,R3
   ADD     #14,R3           ;GET TO RSX FDB
   MOV     F.EFBK+2(R3),R1  ;GET LOW ORDER COUNT
```

```
        MOV     F.EFBK(R3),R0   ;GET HIGH ORDER COUNT
        DEC     R1
        TST     R0              ; SEE IF WE HAD AN EMPTY FILE
        BNE     10$             ; BR IF NOT
        TST     R1
        BNE     10$
        TST     F.FFBY(R3)
        BNE     10$
;
; WE HAVE AN EMPTY FILE
;
        CLR     R0              ; RETURN ZERO RECORDS
        CLR     R1
        BR      30$
10$:
        ASHC    #9.,R0
        ADD     F.FFBY(R3),R1
        ADC     R0
        MOV     R1,R2
        MOV     R0,R1
        MOV     F.RSIZ(R3),R0
        BNE     20$
;
; IF ZERO THEN WE HAVE A VARIABLE LENGTH FILE
;
        INC     R0      ; RETURN A 1
        CLR     R1
        BR      30$
20$:
        INC     R0      ;ROUND TO EVEN WORD SIZE
        ASH     #-1,R0
        ASH     #1,R0
        CALL    $DDIV
        MOV     R2,R0
30$:
        RETURN
        .END
```

Now what about files with NOSPANBLOCK. For fixed length record files, you know how many records will be in each block except the last. When, you open a file FORTRAN remembers the last block in use and the byte within that block. The following routine can be used to obtain this information and then you can calcualate yur file size in records. Note this routine is also useful for tracking record placement if you plan to use .POINT style access for pseudo-direct access of variable length records.

The code follows:

```
        .TITLE FCBSIZ - ROUTINE TO SNATCH BLOCK & FCBSIZER
        .IDENT /X01/
;+
```

```
;  CALL FCBSIZ (ARG1,ARG2,ARG3)
;  INTEGER*2 ARG1          !LUN
;  INTEGER*4 BLOCK         !LAST USED BLOCK
;  INTEGER*2 BYTE          !BYTE POSITION IN LAST BLOCK
;
;  ARG1 = FORTRAN CHANNEL NO.
;  ARG2 = BLOCK # WITH EOF
;  ARG3 = FIRST FREE BYTE
;-
FCBSIZ::
   MOV     @2(R5),R2       ;SET CHNL NO.
   JSR     PC,$FCHNL       ;GET FDB IN R1
   MOV     R0,R1
   ADD     #12.,R1
   MOV     4(R5),R4        ;2ND ARG ADDR.
   MOV     F.EFBK+2(R1),(R4)+
   MOV     F.EFBK(R1),(R4)
   MOV     F.FFBY(R1),@6(R5)
   RETURN
   .END
```

As you can see, FORTRAN can be made more useful and a lot more elegant by accessing information in the FDB and by using some of the FCS routines.

Bruce Mitchell's column on MACRO can help you understand the MACRO side. We will try to cover some more FCS material in the future. If you run into a snag, see if Ed's Q/A column can help.

## Recursive FORTRAN Programming

Dale D. Lutes
Cessna Aircraft Company
Wichita, KS

There are many programming problems which can be greatly simplified by using recursive subroutines. Although programming languages which support recursive programming are available, some installations may not have a compiler for such a language. Here is a method for writing recursive FORTRAN routines for use on a PDP-11. These ideas can be modified for use with other languages and other machines.

There are two problems to be overcome when writing recursive routines in FORTRAN:

1. FORTRAN compilers will not allow a subroutine to call itself.

2. Because each successive call to the routine uses the same memory space for local variables, intermediate results will be lost for all but the final call to the subroutine.

The first problem may be solved by using an alias for the recursive routine. For instance, if some program makes use of the recursive subroutine CHKPAR, instead of coding the recusive call like this:

```
SUBROUTINE CHKPAR
   .
   .
CALL CHKPAR
   .
   .
RETURN
```

Write it like this:

```
SUBROUTINE CHKPAR
   .
   .
CALL CHK
   .
   .
RETURN
```

This satisfies the compiler, but we get an undefined reference for CHK at task build time. There are two ways to establish the alias. One is to get a copy of the memory allocation map provided by TKB and note the address of CHKPAR. Task build again using the GBLDEF option:

```
GBLDEF=chk:address of CHKPAR
```

This makes CHK an alias for CHKPAR as they both refer to the same address. The second solution uses a short MACRO-11 subroutine to define CHK:

```
        .TITLE  CHK
CHK::   JSR     PC,CHKPAR
        RTS     PC
        .END
```

The second method has the slight disadvantage of placing an extra subroutine call on the stack. The first method has the disadvantage of requiring two task builds each time the program is modified. For large programs involving time-consuming task builds, the second method is much more attractive.

The second problem is usually a little more difficult to solve. If our routine, CHKPAR, includes a loop like this:

```
        DO 100, K_=1,5
            IF (IARRAY(K).NE.0) CALL CHK
100     CONTINUE
```

and the variable K has reached, let's say 3, before CHK is called, there is only a very slight chance that K will still be equal to 3 when CHK returns. Clearly, variables like K that should remain unchanged by more deeply nested calls to the recursive routine must be saved somewhere. The obvious place is on the stack. We can write MACRO routines to push the variables on the stack and then pop them after the recursive call. To save our variable, K, from the previous example, we can use the following:

```
        FROM THE MAIN PROGRAM...

        DO 100, K_=1,5
            CALL PUSH (K)
            IF (IARRAY(K).NE.0) CALL CHK
            CALL POP (K)
100     CONTINUE


        MACRO SUBROUTINE...

        .TITLE  CHK
CHK::   JSR     PC,CHKPAR       ;Make recursive call
        RTS     PC
;
PUSH::  MOV     (SP)+,TEMP1     ;Save PUSH's return...
        MOV     (SP)+,TEMP2     ;...address _& link
        MOV     @2(R5),-(SP)    ;Push K
        MOV     TEMP2,-(SP)     ;Restore return...
        MOV     TEMP1,-(SP)     ;...address _& link
        RTS     PC
;
POP::   MOV     (SP)+,TEMP1     ;Save POP's return...
        MOV     (SP)+,TEMP2     ;...address _& link
        MOV     (SP)+,@2(R5)    ;Pop K
        MOV     TEMP2,-(SP)     ;Restore return...
        MOV     TEMP1,-(SP)     ;...address _& link
        RTS     PC
;
TEMP1:  .WORD
```

```
TEMP2:  .WORD
```

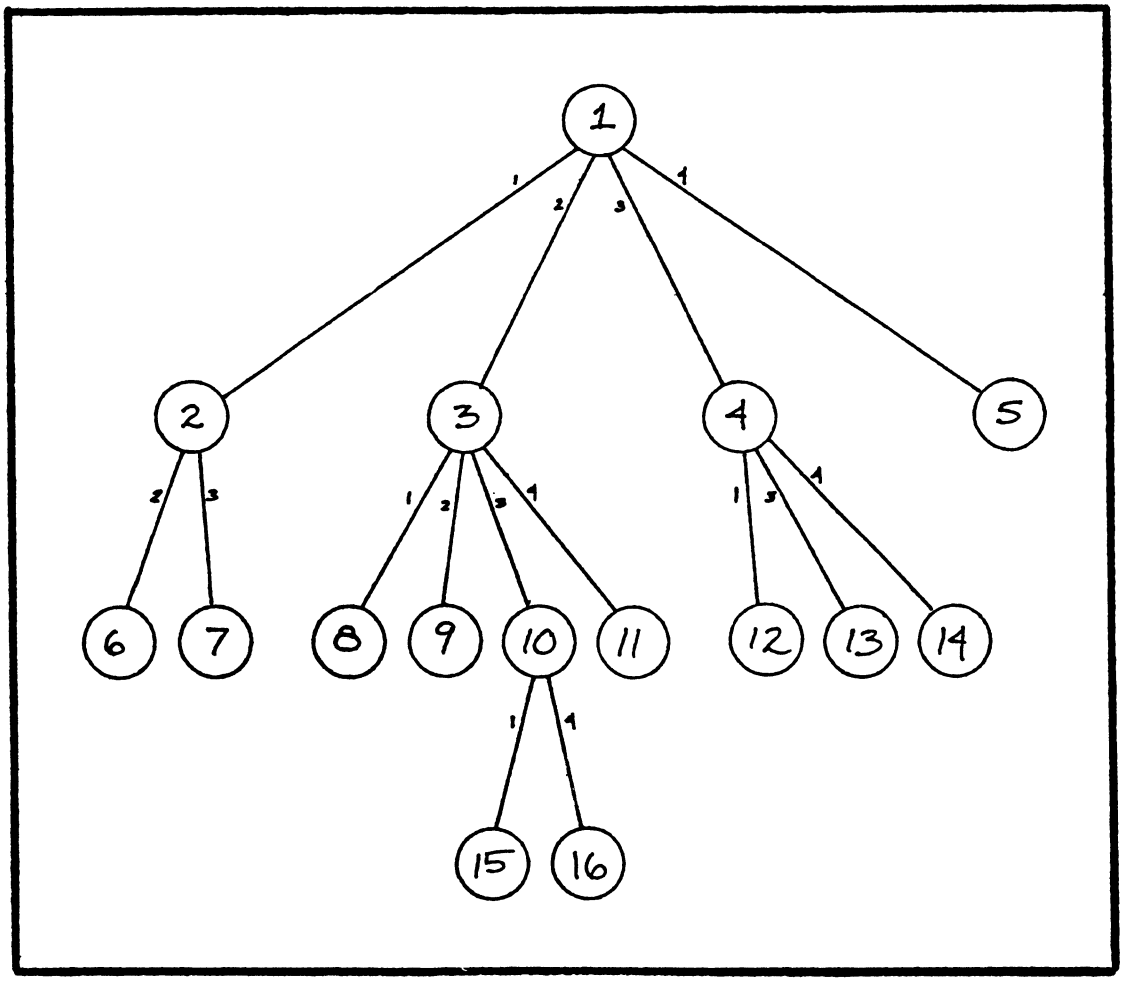
FORTRAN puts the contents of R5 on the stack before loading it with the address of the parameter table. The subroutine call puts the return address on top of that. These two items must remain on the top of the stack during pushes and pops. The previous example temporarily stores these values at addresses TEMP1 and TEMP2 then returns them to the top of the stack after the push or pop.

The demonstration program that accompanies this article shows one application of recursive programming. The program does the postorder traversal of the 4-ary tree shown on the next page. Postorder traversal means that all of a given node's children are visited before the node itself. The tree is set up as a 16x4 array of integers representing the four children of each of the sixteen nodes in the tree.

The program's first action is to print a table listing the nodes and their children. A zero in this table indicates no child on that link. Processing of the tree begins with the call to TRAVRS from the main program, RECURS.

TRAVRS contains a DO loop that checks each of the current node's links in turn. If a child is found, the current node and link number are pushed onto the stack and processing continues with the child. When all of a node's links have been processed, that node is visited.

The program prints the names of each of the nodes in the order in which they are visited. If you would like to follow the program through the series of recursive calls, uncomment the first three write statements in the subroutine TRAVERS.




Tree traversed by demonstration program, RECURS
NOTE: Small numbers near links indicate child number.

```
C=====================================================================
C
C    FROM DR1:[4,120]RECURS.FTN - FORTRAN recursive programming
C                                 demonstration.
C
C    COMPILE INFO:  F77 RECURS,RECURS=RECURS
C
C    TASK BUILD INFO:       TKB RECURS,RECURS=RECURS,TRV
C
C    WRITTEN:       DALE D. LUTES    11-MAR-85
C
C=====================================================================
C
     PROGRAM RECURS
C
```

```
      INTEGER TREE(16,4)      !4-ary tree to be traversed
C
      DATA TREE / 2, 0, 8,12, 0, 0, 0, 0, 0,15, 0, 0, 0, 0, 0, 0
     1          , 3, 6, 9, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
     1          , 4, 7,10,13, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
     1           , 5, 0,11,14, 0, 0, 0, 0, 0,16, 0, 0, 0, 0, 0, 0/
C
      COMMON  /ALL/TREE
C
C
C.......FORMAT STATEMENTS
C
10  FORMAT(2X, 5(I2, 7X))
11  FORMAT(///' NODE ', 4(3X, 'CHILD', I1))
12  FORMAT(//)
C
C
C.......MAIN
C
      WRITE (6,11) 1,2,3,4    !Print the tree in table format
      DO 100, I=1,16
          WRITE (6,10) I,(TREE(I,J), J=1,4)
100         CONTINUE
      WRITE (6,12)
C
      CALL TRAVRS(1)          !Begin traversal at root
      END
C
C
C
C=============================================================
C
C  Recursive subroutine to do a postorder traversal of a
C       4-ary subtree
C
C=============================================================
C
      SUBROUTINE TRAVRS (NODE)
C
      INTEGER TREE(16,4),     !Global variable - see RECURS
     1        NODE            !Node at which to begin traversal
C
      COMMON  /ALL/TREE
C
C
C.......FORMAT STATEMENTS
C
10  FORMAT('0Beginning node ', I2)
11  FORMAT(5X, 'Node ', I2, ', child ', I1, ' = ', I2)
12  FORMAT('0Returning to node ', I2)
14  FORMAT('0VISITING ', I2)
C
C
```

```
  C.......MAIN
  C
  C  WRITE (6,10) NODE
     DO 900, I=1,4
  C       WRITE (6,11) NODE, I, TREE(NODE,I)
          IF (TREE(NODE,I).NE.0) THEN    !Non-empty subtree, traverse it
              CALL PUSH(NODE,I)          !Save local variables
              CALL TRV (TREE(NODE,I))    !Make recursive call
              CALL POP (NODE,I)          !Restore local variables
  C           WRITE (6,12) NODE
          END IF
  900         CONTINUE
  C
     WRITE (6,14) NODE        !Finished with subtrees, visit node
     RETURN
     END


  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;
  ;    FROM DR0:[4,120]TRV.MAC - This subroutine allows TRAVRS to call itself.
  ;    Also included are routines to push/pop two integers on/from the stack.
  ;
  ;    CALLING SEQUENCE:       CALL TRV (NODE)
  ;          WHERE:    NODE = The single parameter normally passed in
  ;                           CALL TRAVRS (NODE)
  ;
  ;    ASSEMBLE INFO: MAC TRV,TRV=TRV
  ;
  ;    WRITTEN BY:             DALE D. LUTES    11-MAR-85
  ;
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;
     .TITLE  TRV
     .IDENT  /031185/
  ;
  TRV::      JSR     PC,TRAVRS        ;Process CHILD
     RTS     PC
  ;
  ;
  ;
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;
  ;    CALLING SEQUENCE:       CALL PUSH (I1, I2)
  ;                            or
  ;                    CALL POP  (I1, I2)
  ;          WHERE:    I1 = 1st integer variable to be pushed/popped
  ;                    I2 = 2nd integer variable to be pushed/popped
  ;
  ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
  ;
  PUSH::     MOV     (SP)+,TEMP1      ;Get return address & link register
     MOV     (SP)+,TEMP2
```

```
        MOV     @2(R5),-(SP)    ;Push PARENT on the stack
        MOV     @4(R5),-(SP)    ;Push INDEX on the stack
        MOV     TEMP2,-(SP)     ;Put return address & link on new
    ;                                   top of stack
        MOV     TEMP1,-(SP)
        RTS     PC
    ;
    POP::       MOV     (SP)+,TEMP1     ;Get return address & link register
        MOV     (SP)+,TEMP2
        MOV     (SP)+,@4(R5)    ;Restore INDEX from the stack
        MOV     (SP)+,@2(R5)    ;Restore PARENT from the stack
        MOV     TEMP2,-(SP)     ;Put return address & link on new
    ;                                   ;top of stack
        MOV     TEMP1,-(SP)
        RTS     PC
    ;
    TEMP1:      .WORD
    TEMP2:      .WORD
        .END
```

Printed in the U.S.A.




DECUS SUBSCRIPTION SERVICE
DIGITAL EQUIPMENT COMPUTER SOCIETY
219 BOSTON POST ROAD, (BP02)
MARLBORO, MA 01752

## STATUS CHANGE

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

( ) Change of Address
( ) Please Delete My Membership Record
(I Do Not Wish To Remain A Member)

DECUS Membership No: _____

Name: _____

Company: _____

Address: _____

_____

State/Country: _____

Zip/Postal Code: _____

Mail to: DECUS - ATTN: Subscription Service
219 Boston Post Road, BP02
Marlboro, MA 01752
USA