```
;       FILE MESA-NOVA1.ASM
;       R. JOHNSSON
;       LAST MODIFIED August 10, 1978  9:47 AM
        .TITL MesaNova1
        .TXTM B
;
        .ENT EMLOOP
        .ENT MesaNova1
        .ENT MesaNovaSize1
        .ENT AC0Ptr
        .ENT AC1Ptr
        .ENT AbsoluteTXV
        .ENT CleanUpQueueUser
        .ENT RequeueSubUser
        .ENT WakeHeadImplementer
        .ENT STOPUser
        .ENT AdvanceTimerPtr
        .ENT processTrapPtr
        .ENT firstProcessPtr
        .ENT lastProcessPtr
        .ENT firstStateVectorPtr
;
        .SREL
EMLOOP: EMLP-TXV        ;WHERE TO START RUNNING
MesaNova1: TXV
MesaNovaSize1: END-TXV
AC0Ptr: currentState
AC1Ptr: WASAC1
AbsoluteTXV: X  ; To check compatibility
CleanUpQueueUser: CleanUpQueue
RequeueSubUser: RequeuePtr
WakeHeadImplementer: WakeHead-TXV+X
STOPUser: pSTOP
AdvanceTimerPtr: AdvanceTimer-TXV+X
processTrapPtr: processTrap
firstProcessPtr: firstProcess
lastProcessPtr: lastProcess
firstStateVectorPtr: firstStateVector
;
        .NREL
        .GET "Mesa-NovaDefs.asm"

X = 25          ; The absolute location of TXV at run time


TXV:    EMLP-TXV+X      ; Microcode expects this to be in location 25
        MSTOP-TXV+X
        77400           ; BLK
        77400           ; CONVERT
        ME-TXV+X
        MRE-TXV+X
        MXW-TXV+X
        MXD-TXV+X
        NOTIFY-TXV+X
        BCAST-TXV+X
        REQUEUE-TXV+X

CVArray:        ; This will be location 40
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0
        0

MSTOP:  JMP @pSTOP
```

```
pSTOP:  0


;
; Locations for saving accumulators
WASAC1: 0
trapmask: abortPending;+timeoutPending ; make timeout not trap
processTrap: 0

; Monitor ReEntry(@Monitor,cvptr)
MRE:
        sta     1 cvptr
        jsr     Enter
        mov     0 0 snc         ; if failed
          jmp   EMLP1           ; then return
        lda     0 cvptr
        jsr     @CleanUpQueue
        lda     2 currentPSB
        sub     0 0
        sta     0 cleanUpLink,2 ; currentPSB.cleanUpLink ← Clean
        lda     0 trapmask
        lda     1 bitsandpriority,2
        and#    0 1 snr         ; if no timeout or abort
          jmp   EnterTrue       ;   then return
        lda     2 currentState  ; cause a trap in the current state
        lda     1 dest,2
        sta     1 source,2      ; make the dest be the source
        lda     1 @processTrap
        sta     1 dest,2        ; and cause a call to SD[sProcessTrap]
        jmp     EMLP

Enter:          ;(@Monitor) returns carry=1 iff success
        mov     0,2             ; copy pointer
        lda     0 lockqueue,2   ; get lock and queue word
        movzl   0 0             ; lock bit to carry
        movl    1 1             ; store carry in low-order AC1
        movzr   0 0             ; set lock bit to 0 = locked
        sta     0 lockqueue,2   ; store value back
        movzr   1 1 szc         ; test and remove lock bit
          jmp 0,3               ; done, carry=1
EnterFail:
        sta     3 EXWreturn
        mov     2 1             ; assumes queue=0
        lda     2 currentPSB
        lda     0 bitsandpriority,2
        movl    0 0
        movor   0 0
        sta     0 bitsandpriority,2 ; set enterfailed ← TRUE
        lda     0 pReadyList
        jsr     RequeueSub;(pReadyList,@Monitor.queue,currentPSB)
        movz    0 0             ; set carry=0
        jmp     @EXWreturn

REQUEUE:        ;(@from,@to,p)
        jsr     RequeueSub
        jmp     EMLP1

TabortPending: abortPending
ReSchedulePending: 0
clStateVector: lStateVector

; Monitor Exit and Depart(@Monitor)
MXD:
        jsr     Exit
        jmp     EMLP1


NoneReady:
        EIR
EMLP:   DIR
EMLP1:  LDA 1 WASAC1
        LDA 0 currentState
        LDA 2 ReSchedulePending
        SNZ 2 2
          61010 ; JMPRM
ReSchedule:
```

```
        lda     2 SDC
        sz      2 2             ; scheduling disabled?
          61010 ; JMPRM ;       jmp     EMLP
        lda     3 readyList
        snz     3 3
          jmp   NoneReady
        sta     2 ReSchedulePending
        lda     3 link,3        ; pReadyList↑↑.link
        lda     2 currentPSB
        ;sne    2 3
          ;jmp  EMLP
        sta     3 currentPSB    ; store new PSB
        lda     3 currentState
        lda     0 dest,3
        sta     0 frame,2
        lda     3 currentPSB
        lda     1 bitsandpriority,3
        movzl   1 0             ; carry ← enterfailed
        cycle   17
        sta     0 bitsandpriority,3 ; enterfailed ← FALSE
        lda     0 mPriority
        and     0 1             ; isolate priority
        lda     0 @firstStateVector
        lda     2 clStateVector
        mul                     ; AC1 ← priority*lStateVector+firstStateVector
        sta     1 currentState
        mov     1 2
        lda     1 frame,3
        sta     1 dest,2
        mov     0 0 snc
          jmp   EMLP1
        jmp     EnterFalse

mPriority: priority
firstStateVector: 0

; Monitor Exit and Wait(@Monitor,cvptr,timeout)
MXW:
        sta     1 cvptr
        sta     2 savetimeout
        jsr     Exit
        lda     0 cvptr
        jsr     @CleanUpQueue
        lda     2 currentPSB
        lda     1 TabortPending
        lda     0 bitsandpriority,2
        and#    1 0 szr         ; if not abortPending
          jmp   EMLP1
        lda     1 @cvptr
        movzl   1 1 snc         ; test ww bit
          jmp   MXWq
        sta     1 @cvptr        ; ww set, clear it and don't stop
        jmp     EMLP1

MXWq:   lda     1 TwaitingOnCV
        add     1 0
        lda     1 FtimeoutAllowed
        and     1 0             ; timeoutAllowed ← FALSE
        lda     3 savetimeout
        sz      3 3
          adc   1 0             ; timeoutAllowed ← TRUE
        sta     0 bitsandpriority,2
        lda     0 timer
        add     0 3
        sta     3 timeout,2     ; currentPSB.timeout ← timer+timeout
        lda     0 pReadyList
        lda     1 cvptr
        jsr     RequeueSub;(pReadyList,cvptr,currentPSB)
        jmp     EMLP1

CleanUpQueue:   0       ;(q) returns (q)

RequeueSub:     sta     2 ReSchedulePending
        jmp     @RequeuePtr
RequeuePtr:     0       ; (q1,q2,p)
```

```
EXWreturn: 0

cvptr:  0
savetimeout: 0
TwaitingOnCV: waitingOnCV
FtimeoutAllowed: 0-timeoutAllowed-1
pReadyList: readyList

; Monitor Entry(@Monitor)
ME:
        jsr     Enter
        mov     0 0 snc         ; if failed
          jmp   EMLP1   ;   then reschedule
EnterTrue:
        subzl   0 0 skp
EnterFalse:
        sub     0 0
        lda     2 currentState
        sta     0 stk0,2        ; store return value
        mkone   0 0
        sta     0 stkp,2        ; stkp ← 1
        jmp     EMLP

NOTIFY:         ;(cvptr)
        jsr     @CleanUpQueue
        jsr     WakeHead
        jmp     EMLP1

BCAST:          ;(cvptr)
        jsr     @CleanUpQueue
        sta     0 cvptr
BCASTloop:
        lda     0 cvptr
        jsr     WakeHead
        sz      2 2             ; = process notified or 0 if none
          jmp   BCASTloop
        jmp     EMLP1

Exit:           ;(@Monitor) returns carry=1 iff requeue was called
        sta     0 saveq
        sta     3 EXWreturn
        lda     1 @saveq
        mov     1 2 snr         ; if mq↑ = NIL
          jmp   exitstore       ; low-order zero will go to carry

        lda     1 pReadyList
        ; 0  correct from above
        lda     2 link,2        ; p ← mq↑↑.link
        jsr     RequeueSub;(@Monitor.queue,pReadyList,Monitor.queue.link)
        lda     1 @saveq
        movol   1 1             ; low-order one will go to carry
exitstore:
        movor   1 1             ; set lock to 1 = unlocked
                                ; carry ← 1 iff requeue happened
        sta     1 @saveq
        jmp     @EXWreturn

saveq:  0

WakeHead:       ;(cvptr)
        sta     3 EXWreturn
        mov     0 3
        lda     2 0,3           ; cvptr↑
        movzl#  2 2 snr         ; test for process = 0, ignore ww
          jmp   @EXWreturn
        lda     2 link,2        ; cvptr↑↑.link
        lda     1 bitsandpriority,2
        lda     3 FwaitingOnCV
        and     3 1
        sta     1 bitsandpriority,2
        lda     1 pReadyList
        jsr     RequeueSub;(cvptr,pReadyList,cvptr↑)
        jmp     @EXWreturn

FwaitingOnCV: 0-waitingOnCV-1
```

```
timer:  0
timeflag: -3
timeconst: -3

AdvanceTimer:
        isz     timeflag
          bri
        lda     0 timeconst
        sta     0 timeflag
        isz     timer
          nop
        lda     2 @firstProcess
loop:   lda     0 timermask
        lda     1 bitsandpriority,2
        and     0 1
        seq     0 1
          jmp   notready
        lda     1 timeout,2
        lda     0 timer
        seq     0 1
          jmp   notready
;       lda     0 FtimeoutPending
;       lda     1 bitsandpriority,2
;       and     0 1
;       adc     0 1             ; p.timeoutPending ← TRUE
        lda     0 FwaitingOnCV
        lda     1 bitsandpriority,2
        and     0 1             ; p.waitingOnCV ← FALSE
        sta     1 bitsandpriority,2
        sub     0 0
        lda     1 pReadyList
        jsr     RequeueSub;(NIL,pReadyList,p) -- returns p in AC2
notready:
        lda     1 @lastProcess
        sne     1 2
          bri
        lda     1 clPSB
        add     1 2
        jmp loop

timermask: state+timeoutAllowed+waitingOnCV
FtimeoutPending: 0-timeoutPending-1
firstProcess: 0
lastProcess: 0
clPSB: lPSB

END:    jmp     TXV+200         ; generate error if too big

;
;
        .END
```