```
;---------------------------------------------------------------
; Mesad.Mu - Xfer, State switching, process support, Nova interface
; Last modified by Levin - August 1, 1978  3:26 PM
;---------------------------------------------------------------


;---------------------------------------------------------------
; F r a m e   A l l o c a t i o n
;---------------------------------------------------------------


;---------------------------------------------------------------
; Alloc subroutine:
;       allocates a frame
;       Entry conditions:
;         frame size index (fsi) in T
;       Exit conditions:
;         frame pointer in L, T, and frame
;         if allocation fails, alternate return address is taken and
;           temp2 is shifted left by 1 (for ALLOC)
;---------------------------------------------------------------
!1,2,AllocSub,ALLOClarge;                                       for ALLOC byte code
!1,2,ALLOCr,XferGr;                                             subroutine returns
!1,2,ALLOCrf,XferGrf;                                           failure returns
!3,4,Alloc0,Alloc1,Alloc2,Alloc3;                               dispatch on pointer flag
;       if more than 2 callers, un-comment the following pre-definition:
; !17,1,Allocx;                                                 shake IR← dispatch

AllocSub:       L←avm1+T+1, TASK, :Allocx;                      fetch av entry

Allocx:         entry←L;                                        save av entry address
                L←MAR←entry;
                T←3;                                            mask for pointer flags
                L←MD AND T, T←MD;                               (L←MD AND 3, T←MD)
                temp←L, L←MAR←T;                                start reading pointer
                SINK←temp, BUS;                                 branch on bits 14:15
                frame←L, :Alloc0;

;
; Bits 14:15 = 00, a frame of the right index is queued for allocation
;

Alloc0:         L←MD, TASK;                                     new entry for frame vector
                temp←L;                                         new value of vector entry
                MAR←entry;                                      update frame vector
                L←T←frame, IDISP;                               establish exit conditions
                MD←temp, :ALLOCr;                               update and return

;
; Bits 14:15 = 01, allocation list empty:  restore argument, take failure return
;

Alloc1:         L←temp2, IDISP, TASK, :Alloc1x;                 restore parameter
Alloc1x:        temp2←L LSH 1, :ALLOCrf;                        allocation failed

;
; Bits 14:15 = 10, a pointer to an alternate list to use
;

Alloc2:         temp←L RSH 1, :Allocp;                          indirection: index←index/4

Allocp:         L←temp, TASK;
                temp←L RSH 1;
                T←temp, :AllocSub;

Alloc3:         temp←L RSH 1, :Allocp;                          (treat type 3 as type 2)
```

```
;---------------------------------------------------------------
; Free subroutine:
;       frees a frame
;       Entry conditions: address of frame is in 'frame'
;       Exit conditions: 'frame' left pointing at released frame (for LSTF)
;---------------------------------------------------------------

!3,4,RETr,FREEr,LSTFr,;                                 FreeSub returns
!17,1,Freex;                                            shake IR← dispatch

FreeSub:        MAR←frame-1;                            start read of fsi word
Freex:          NOP;                                    wait for memory
                T←MD;                                   T←index
                L←MAR←avm1+T+1;                         fetch av entry
                entry←L;                                save av entry address
                L←MD;                                   read current pointer
                MAR←frame;                              write it into current frame
                temp←L, TASK;
                MD←temp;                                write!
                MAR←entry;                              entry points at frame
                IDISP, TASK;
                MD←frame, :RETr;                        free
```

```
;-------------------------------------------------------------
; ALLOC - allocate a frame of size specified by <TOS> (popped)
;       if <TOS> < maxallocslot, <TOS> is a frame size index (fsi)
;       otherwise, <TOS> is requested frame size in words.
;-------------------------------------------------------------
!1,1,Savpcinframe;                                          (here so ALLOCrf can call it)
!7,10,XferGT,Xfer,Mstopr,PORTOpc,LSTr,ALLOCrfr,,;           return points for Savpcinframe
!1,2,doAllocTrap,XferGfz;                                   used by XferGrf

ALLOC:          L←ret7, TASK, :Xpopsub;                     returns to ALLOCrx
ALLOCrx:        L←maxallocslot-T-1;
                L←T, SH<0;                                  L:fsi
                temp2←L LSH 1, IR←msr0, :AllocSub;
ALLOCr:         :pushTB;                                    successful allocation

;
;  Allocation failed - save mpc, undiddle lp, push fsi*4 on stack, then trap
;

ALLOCrf:        IR←sr5, :Savpcinframe;                      failure because lists empty
ALLOCrfr:       L←temp2, TASK, :doAllocTrap;                pick up trap parameter

ALLOClarge:     L←temp2, TASK, :Alloc1x;                    failure because too large

;
;  Inform software that allocation failed
;

doAllocTrap:    ATPreg←L;                                   store param. to trap proc.
                T←sAllocListEmpty, :Mtrap;                  go trap to software


;-------------------------------------------------------------
; FREE - release the frame whose address is <TOS> (popped)
;-------------------------------------------------------------

FREE:           L←ret10, TASK, :Xpopsub;                    returns to FREErx
FREErx:         frame←L, TASK;
                IR←sr1, :FreeSub;
FREEr:          :next;
```

```
;-------------------------------------------------------------
; D e s c r i p t o r   I n s t r u c t i o n s
;-------------------------------------------------------------


;-------------------------------------------------------------
; DESCB - push <<gp>+gfi offset>+2*alpha+1 (masking gfi word appropriately)
;       DESCB is assumed to be A-aligned (no pending branch at entry)
;-------------------------------------------------------------

DESCB:          T←gp;
                T←ngpoffset+T+1, :DESCBcom;                     T:address of frame

DESCBcom:       MAR←gfioffset+T;                                start fetch of gfi word
                T←gfimask;                                      mask to isolate gfi bits
                T←MD.T;                                         T:gfi
                L←ib+T, T←ib;                                   L:gfi+alpha, T:alpha
                T←M+T+1, :pushTA;                               pushTA because A-aligned


;-------------------------------------------------------------
; DESCBS - push <<TOS>+gfi offset>+2*alpha+1 (masking gfi word appropriately)
;       DESCBS is assumed to be A-aligned (no pending branch at entry)
;-------------------------------------------------------------

DESCBS:         L←ret15, TASK, :Xpopsub;                        returns to DESCBcom
```

```
;---------------------------------------------------------------
; T r a n s f e r   O p e r a t i o n s
;---------------------------------------------------------------


;---------------------------------------------------------------
; Savpcinframe subroutine:
;       stashes C-relative (mpc,ib) in current local frame
;       undiddles lp into my and lp
;       Entry conditions: none
;       Exit conditions:
;         current frame+1 holds pc relative to code segment base (+ = even, - = odd)
;         lp is undiddled
;         my has undiddled lp (source link for Xfer)
;---------------------------------------------------------------
; !1,1,Savpcinframe;                                            required by PORTO
; !7,10,XferGT,Xfer,Mstopr,PORTOpc,LSTr,ALLOCrfr,,;             returns (appear with ALLOC)
!7,1,Savpcx;                                                    shake IR← dispatch
!1,2,Spcodd,Spceven;                                            pc odd or even

Savpcinframe:   T←cp, :Savpcx;                                  code segment base
Savpcx:         L←mpc-T;                                        L is code-relative pc
                SINK←ib, BUS=0;                                 check for odd or even pc
                T←M, :Spcodd;                                   pick up pc word addr

Spcodd:         L←0-T, TASK, :Spcopc;                           - pc => odd, this word
Spceven:        L←0+T+1, TASK, :Spcopc;                         + pc => even, next word

Spcopc:         taskhole←L;                                     pc value to save
                L←0;                                            (can't merge above - TASK)
                T←npcoffset;                                    offset to pc stash
                MAR←lp-T, T←lp;                                 (MAR←lp-npcoffset, T←lp)
                ib←L;                                           clear ib for XferG
                L←nlpoffset+T+1;                                L:undiddled lp
                MD←taskhole;                                    stash pc in frame+pcoffset
                my←L, IDISP, TASK;                              store undiddled lp
                lp←L, :XferGT;
```

```
;-----------------------------------------------------------------
; Loadgc subroutine:
;       load global pointer and code pointer given local pointer or GFT pointer
;       Entry conditions:
;         T contains either local frame pointer or GFT pointer
;         memory fetch of T has been started
;         pending branch (1) catches zero pointer
;       Exit conditions:
;         lp diddled (to framebase+6)
;         mpc set from second word of entry (PC or EV offset)
;         first word of code segment set to 1 (used by code swapper)
;       Assumes only 2 callers
;-----------------------------------------------------------------

!1,2,XferOr,Xfer1r;                                             return points
!1,2,Loadgc,LoadgcTrap;
!1,2,LoadgcOK,LoadgcNull;                                       good global frame or null
!1,2,LoadgcIn,LoadgcSwap;                                       in-core or swapped out

Loadgc:         L←lpoffset+T;                                   diddle (presumed) lp
                lp←L;                                           (only correct if frame ptr)
                T←MD;                                           global frame address
                L←MD;                                           2nd word (PC or EV offset)
                MAR←cpoffset+T;                                 read code pointer
                mpc←L, L←T;                                     copy g to L for null test
                L←gpoffset+T, SH=0;                             diddle gp, test for null
                T←MD, BUSODD, :LoadgcOK;                        check for swapped out

LoadgcOK:       MAR←T, :LoadgcIn;                               write into code segment

LoadgcIn:       gp←L, L←T;                                      set global frame pointer
                cp←L, IDISP, TASK;                              set code pointer
                MD←ONE, :XferOr;

;
;       picked up global frame of zero somewhere, call it unbound
;
!1,1,Stashmx;
LoadgcNull:     T←sUnbound, :Stashmx;                           BUSODD may be pending

;
;       swapped code segment, trap to software
;
LoadgcSwap:     T←sCsegSwappedOut, :Stashmx;

;
;       destination link = 0
;
LoadgcTrap:     T←sControlFault, :Mtrap;
```

```
;-----------------------------------------------------------------
; CheckXferTrap subroutine:
;       Handles Xfer trapping
;       Entry conditions:
;         IR: return number in DISP
;         T: parameter to be passed to trap routine
;       Exit conditions:
;         if trapping enabled, initiates trap and doesn't return.
;-----------------------------------------------------------------
!3,4,Xfers,XferG,RETxr,;                                    returns from CheckXferTrap
!1,2,NoXferTrap,DoXferTrap;
!3,1,DoXferTrapx;

CheckXferTrap:  L←XTSreg, BUSODD;                           XTSreg[15]=1 => trap
                SINK←DISP, BUS, :NoXferTrap;                dispatch (possible) return

NoXferTrap:     XTSreg←L RSH 1, :Xfers;                     reset XTSreg[15] to 0 or 1

DoXferTrap:     L←DISP, :DoXferTrapx;                       tell trap handler which case
DoXferTrapx:    XTSreg←L LCY 8, L←T;                        L:trap parameter
                XTPreg←L;
                T←sXferTrap, :Mtrap;                        off to trap sequence
```

```
;---------------------------------------------------------------
; Xfer open subroutine:
;       decodes general destination link for Xfer
;       Entry conditions:
;         source link in my
;         destination link in mx
;       Exit conditions:
;         if destination is frame pointer, does complete xfer and exits to Ifetch.
;         if destination is procedure descriptor, locates global frame and entry
;           number, then exits to 'XferG'.
;---------------------------------------------------------------

!3,4,Xfer0,Xfer1,Xfer2,Xfer3;                                   destination link type

Xfer:           T←mx;                                           mx[14:15] is dest link type
                IR←0, :CheckXferTrap;
Xfers:          L←3 AND T;                                      extract type bits
                SINK←M, L←T, BUS;                               L:dest link, branch on type
                SH=0, MAR←T, :Xfer0;                            check for link = 0. Memory
;                                                               data is used only if link
;                                                               is frame pointer or indirect

;---------------------------------------------------------------
; mx[14-15] = 00
;       Destination link is frame pointer
;---------------------------------------------------------------

Xfer0:          IR←msr0, :Loadgc;                               to LoadgcNull if dest link = 0
Xfer0r:         L←T←mpc;                                        offset from cp: - odd, + even

;
; If 'brkbyte' ~= 0, we are proceeding from a breakpoint.
;     pc points to the BRK instruction:
;       even pc => fetch word, stash left byte in ib, and execute brkbyte
;       odd pc => clear ib, execute brkbyte
;
!1,2,Xdobreak,Xnobreak;
!1,2,Xfer0B,Xfer0A;
!1,2,XbrkB,XbrkA;
!1,2,XbrkBgo,XbrkAgo;

                SINK←brkbyte, BUS=0;                            set up by Loadstate
                SH<0, L←0, :Xdobreak;                           dispatch even/odd pc

;
; Not proceeding from a breakpoint - simply pick up next instruction
;

Xnobreak:       :Xfer0B;

Xfer0B:         L←MAR←cp+T, :nextAdeafa;                        fetch word, pc even
Xfer0A:         L←MAR←cp-T, SH=0, :nextXBdeaf;                  fetch word, pc odd (L=0)

;
; Proceeding from a breakpoint - dispatch brkbyte and clear it
;

Xdobreak:       ib←L, :XbrkB;                                   clear ib for XbrkA

XbrkB:          IR←sr20;                                        here if BRK at even byte
                L←MAR←cp+T, :GetalphaAx;                        set up ib (return to XbrkBr)

XbrkA:          L←cp-T;                                         here if BRK at odd byte
                mpc←L, L←0, BUS=0, :XbrkBr;                     ib already zero (to XbrkAgo)

XbrkBr:         SINK←brkbyte, BUS, :XbrkBgo;                    dispatch brkbyte

XbrkBgo:        brkbyte←L RSH 1, T←0+1, :NOOP;                  clear brkbyte, act like nextA
XbrkAgo:        brkbyte←L, T←0+1, BUS=0, :NOOP;                 clear brkbyte, act like next
```

```
;-----------------------------------------------------------------
; mx[14-15] = 01
;       Destination link is procedure descriptor:
;         mx[0-8]: GFT index (gfi)
;         mx[9-13]: EV bias, or entry number (en)
;-----------------------------------------------------------------

Xfer1:          temp←L RSH 1;                                   temp:ep*2+garbage
                count←L MLSH 1;                                 since L=T, count←L LCY 1;
                L←count, TASK;                                  gfi now in 0-7 and 15
                count←L LCY 8;                                  count:gfi w/high bits garbage
                L←count, TASK;
                count←L LSH 1;                                  count:gfi*2 w/high garbage
                T←count;
                T←1777.T;                                       T:gfi*2
                MAR←gftm1+T+1;                                  fetch GFT[T]
                IR←sr1, :Loadgc;                                pick up two word entry into
;                                                               gp and mpc
Xfer1r:         L←temp, TASK;                                   L:en*2+high bits of garbage
                count←L RSH 1;                                  count:en+high garbage
                T←count;
                T←enmask.T;                                     T:en
                L←mpc+T+1, TASK;                                (mpc has EV base in code seg)
                count←L LSH 1, :XferG;                          count:ep*2


;-----------------------------------------------------------------
; mx[14-15] = 10
;       Destination link is indirect:
;         mx[0-15]: address of location holding destination link
;-----------------------------------------------------------------

Xfer2:          NOP;                                            wait for memory
                T←MD, :Xfers;

;-----------------------------------------------------------------
; mx[14-15] = 11
;       Destination link is unbound:
;         mx[0-15]: passed to trap handler
;-----------------------------------------------------------------

Xfer3:          T←sUnbound, :Stashmx;
```

```
;-------------------------------------------------------------
; XferG open subroutine:
;       allocates new frame and patches links
;       Entry conditions:
;         'count' holds index into code segment entry vector
;         assumes lp is undiddled (in case of AllocTrap)
;         assumes gp (undiddled) and cp set up
;       Exit conditions:
;         exits to instruction fetch (or AllocTrap)
;-------------------------------------------------------------

;
; Pick up new pc from specified entry in entry vector
;

XferGT:         T←count;                                        parameter to CheckXferTrap
                IR←ONE, :CheckXferTrap;
XferG:          T←count;                                        index into entry vector
                MAR←cp+T;                                       fetch of new pc and fsi
                T←cp-1;                                         point just before bytes
;                                                               (main loop increments mpc)
                IR←sr1;                                         note: does not cause branch
                L←MD+T;                                         relocate pc from cseg base
                T←MD;                                           second word contains fsi
                mpc←L;                                          new pc setup, ib already 0
                T←377.T, :AllocSub;                             mask for size index
;
; Stash source link in new frame, establishing dynamic link
;
XferGr:         MAR←retlinkoffset+T;                            T has new frame base
                L←lpoffset+T;                                   diddle new lp
                lp←L;                                           install diddled lp
                MD←my;                                          source link to new frame

;
; Stash new global pointer in new frame (same for local call)
;
                MAR←T;                                          write gp to word 0 of frame
                T←gpoffset;                                     offset to point at gf base
                L←gp-T, TASK;                                   subtract off offset
                MD←M, :nextAdeaf;                               global pointer stashed, GO!

;
;  Frame allocation failed - push destination link, then trap
;
; !1,2,doAllocTrap,XferGfz;                                     (appears with ALLOC)

XferGrf:        L←mx, BUS=0;                                    pick up destination, test = 0
                T←count-1, :doAllocTrap;                        T:2*ep+1

;       if destination link is zero (i.e. local procedure call), we must first
;       fabricate the destination link

XferGfz:        L←T, T←ngfioffset;                              offset from gp to gfi word
                MAR←gp-T;                                       start fetch of gfi word
                count←L LSH 1;                                  count:4*ep+2
                L←count-1;                                      L:4*ep+1
                T←gfimask;                                      mask to save gfi only
                T←MD.T;                                         T:gfi
                L←M+T, :doAllocTrap;                            L:gfi+4*ep+1 (descriptor)
```

```
;-------------------------------------------------------------
; Getlink subroutine:
;       fetches control link from either global frame or code segment
;       Entry conditions:
;         temp: - (index of desired link + 1)
;         IR: DISP field zero/non-zero to select return point (2 callers only)
;       Exit conditions:
;         L,T: desired control link
;-------------------------------------------------------------
!1,2,EFCgetr,LLKBr;                                          return points
!1,2,framelink,codelink;
!7,1,Fetchlink;                                              shake IR← in KFCB

Getlink:        T←gp;                                        diddled frame address
                MAR←T←ngpoffset+T+1;                         fetch word 0 of global frame
                L←temp+T, T←temp;                            L:address of link in frame
                taskhole←L;                                  stash it
                L←cp+T;                                      L:address of link in code
                SINK←MD, BUSODD, TASK;                       test bit 15 of word zero
                temp2←L, :framelink;                         stash code link address

framelink:      MAR←taskhole, :Fetchlink;                    fetch link from frame
codelink:       MAR←temp2, :Fetchlink;                       fetch link from code

Fetchlink:      SINK←DISP, BUS=0;                            dispatch to caller
                L←T←MD, :EFCgetr;
```

```
;---------------------------------------------------------------
; EFCn - perform XFER to destination specified by external link n
;---------------------------------------------------------------
; !1,1,EFCr; implicit in EFCr's return number (23B)

EFC0:           IR←ONE, T←ONE-1, :EFCr;                         0th control link
EFC1:           IR←T←ONE, :EFCr;                                1st control link
EFC2:           IR←T←2, :EFCr;                                  . . .
EFC3:           IR←T←3, :EFCr;
EFC4:           IR←T←4, :EFCr;
EFC5:           IR←T←5, :EFCr;
EFC6:           IR←T←6, :EFCr;
EFC7:           IR←T←7, :EFCr;
EFC8:           IR←T←10, :EFCr;
EFC9:           IR←T←11, :EFCr;
EFC10:          IR←T←12, :EFCr;
EFC11:          IR←T←13, :EFCr;
EFC12:          IR←T←14, :EFCr;
EFC13:          IR←T←15, :EFCr;
EFC14:          IR←T←16, :EFCr;
EFC15:          IR←T←17, :EFCr;


;---------------------------------------------------------------
; EFCB - perform XFER to destination specified by external link 'alpha'
;---------------------------------------------------------------
!1,1,EFCdoGetlink;                                              shake B/A dispatch (Getalpha)

EFCB:           IR←sr23, :Getalpha;                             fetch link number
EFCr:           L←0-T-1, TASK, :EFCdoGetlink;                   L:-(link number+1)

EFCdoGetlink:   temp←L, :Getlink;                               stash index for Getlink
EFCgetr:        IR←sr1, :SFCr;                                  for Savpcinframe; no branch


;---------------------------------------------------------------
; SFC - Stack Function Call (using descriptor on top of stack)
;---------------------------------------------------------------

SFC:            IR←sr1, :Popsub;                                get dest link for xfer
;                                                               now assume IR still has sr1
SFCr:           mx←L, :Savpcinframe;                            set dest link, return to Xfer
```

```
;---------------------------------------------------------------
; KFCB - Xfer using destination <<SD>+alpha>
;---------------------------------------------------------------
; !1,1,KFCr; implicit in KFCr's return number (21B)
!1,1,KFCx;                                              shake B/A dispatch (Getalpha)
; !7,1,Fetchlink;       appears with Getlink

KFCB:           IR←sr21, :Getalpha;                     fetch alpha
KFCr:           IR←avm1, T←avm1+T+1, :KFCx;             DISP must be non zero
KFCx:           MAR←sdoffset+T, :Fetchlink;             Fetchlink shakes IR← dispatch


;---------------------------------------------------------------
; BRK - Breakpoint (equivalent to KFC 0)
;---------------------------------------------------------------

BRK:            ib←L, T←sBRK, :KFCr;                    ib = 0 <=> BRK B-aligned


;---------------------------------------------------------------
; Trap sequence:
;       used to report various faults during Xfer
;       Entry conditions:
;         T: index in SD through which to trap
;         Savepcinframe has already been called
;       entry at Stashmx puts destination link in OTPreg before trapping
;---------------------------------------------------------------
; !1,1,Stashmx; above with Loadgc code

Stashmx:        L←mx;                                   can't TASK, T has trap index
                OTPreg←L, :Mtrap;

Mtrap:          T←avm1+T+1;
                MAR←sdoffset+T;                         fetch dest link for trap
                NOP;
Mtrapa:         L←MD, TASK;                             (enter here from PORTO)
                mx←L, :Xfer;
```

```
;-------------------------------------------------------------
; LFCn - call local procedure n (i.e. within same global frame)
;-------------------------------------------------------------
!1,1,LFCx;                                                  shake B/A dispatch

LFC1:           L←2, :LFCx;
LFC2:           L←3, :LFCx;
LFC3:           L←4, :LFCx;
LFC4:           L←5, :LFCx;
LFC5:           L←6, :LFCx;
LFC6:           L←7, :LFCx;
LFC7:           L←10, :LFCx;
LFC8:           L←11, :LFCx;
LFC9:           L←12, :LFCx;
LFC10:          L←13, :LFCx;
LFC11:          L←14, :LFCx;
LFC12:          L←15, :LFCx;
LFC13:          L←16, :LFCx;
LFC14:          L←17, :LFCx;
LFC15:          L←20, :LFCx;
LFC16:          L←21, :LFCx;

LFCx:           count←L LSH 1, L←0, IR←msr0, :SFCr;         stash index of proc. (*2)
;                                                           dest link = 0 for local call
;                                                           will return to XferG


;-------------------------------------------------------------
; LFCB - call local procedure number 'alpha' (i.e. within same global frame)
;-------------------------------------------------------------

LFCB:           IR←sr22, :Getalpha;
LFCr:           L←0+T+1, :LFCx;
```

```
;-------------------------------------------------------------
; RET - Return from function call.
;-------------------------------------------------------------
!1,1,RETx;                                                  shake B/A branch

RET:            T←lp, :RETx;                                local pointer

RETx:           IR←2, :CheckXferTrap;
RETxr:          MAR←nretlinkoffset+T;                       get previous local frame
                L←nlpoffset+T+1;
                frame←L;                                    stash for 'Free'
                L←MD;                                       pick up prev frame pointer
                mx←L, L←0, IR←msr0, TASK;                   mx points to caller
                my←L, :FreeSub;                             clear my and go free frame
RETr:           T←mx, :Xfers;                               xfer back to caller


;-------------------------------------------------------------
; PUSHX - push destination link of previous Xfer
;-------------------------------------------------------------

PUSHX:          T←mx, :pushTB;


;-------------------------------------------------------------
; LLKB - push external link 'alpha'
;       LLKB is assumed to be A-aligned (no pending branch at entry)
;-------------------------------------------------------------

LLKB:           T←ib;                                       T:alpha
                L←0-T-1, IR←0, :EFCdoGetlink;               L:-(alpha+1), go call Getlink
LLKBr:          :pushTA;                                    alignment requires pushTA
```

```
;-----------------------------------------------------------------
; P o r t   O p e r a t i o n s
;-----------------------------------------------------------------


;-----------------------------------------------------------------
; PORTO - PORT Out (XFER thru PORT addressed by TOS)
;-----------------------------------------------------------------

PORTO:          IR←sr3, :Savpcinframe;                          undiddle lp into my
PORTOpc:        L←ret5, TASK, :Xpopsub;                         returns to PORTOr
PORTOr:         MAR←T;                                          fetch from TOS
                L←T;
                MD←my;                                          frame addr to word 0 of PORT
                MAR←M+1;                                        second word of PORT
                my←L, :Mtrapa;                                  source link to PORT address


;-----------------------------------------------------------------
; PORTI - PORT In (Fix up PORT return, always immediately after PORTO)
;       assumes that my and mx remain from previous xfer
;-----------------------------------------------------------------
!1,1,PORTIx;
!1,2,PORTInz,PORTIz;

PORTI:          MAR←mx, :PORTIx;                                first word of PORT

PORTIx:         SINK←my, BUS=0;
                TASK, :PORTInz;

PORTInz:        MD←0;
                MAR←mx+1;                                       store it as second word
                TASK, :PORTIz;
PORTIz:         MD←my, :next;                                   store my or zero
```

```
;---------------------------------------------------------------
; S t a t e   S w i t c h i n g
;---------------------------------------------------------------


;---------------------------------------------------------------
; Savestate subroutine:
;       saves state of pre-empted emulation
;       Entry conditions:
;         L holds address where state is to be saved
;         assumes undiddled lp
;       Exit conditions:
;         lp, stkp, and stack (from base to min[depth+2,8]) saved
;---------------------------------------------------------------

; !1,2,DSTr1,Mstopc; actually appears as %1,1777,776,DSTr1,Mstopc; and is located
; in the front of the main file (Mesa.mu).

!17,20,Sav0r,Sav1r,Sav2r,Sav3r,Sav4r,Sav5r,Sav6r,Sav7r,Sav10r,Sav11r,DSTr,,,,,;
!1,2,Savok,Savmax;

Savestate:      temp←L;
Savestatea:     T←-12+1;                                        i.e. T←-11
                L←lp, :Savsuba;
Sav11r:         L←stkp, :Savsub;
Sav10r:         T←stkp+1;
                L←-7+T;                                         check if stkp > 5 or negative
                L←0+T+1, ALUCY;                                 L:stkp+2
                temp2←L, L←0-T, :Savok;                         L:-stkp-1

Savmax:         T←-7;                                           stkp > 5  => save all
                L←stk7, :Savsuba;

Savok:          SINK←temp2, BUS;                                stkp < 6 => save to stkp+2
                count←L, :Sav0r;

Sav7r:          L←stk6, :Savsub;
Sav6r:          L←stk5, :Savsub;
Sav5r:          L←stk4, :Savsub;
Sav4r:          L←stk3, :Savsub;
Sav3r:          L←stk2, :Savsub;
Sav2r:          L←stk1, :Savsub;
Sav1r:          L←stk0, :Savsub;
Sav0r:          SINK←DISP, BUS;                                 return to caller
                T←-12, :DSTr1;                                  (for DST's benefit)

;   Remember, T is negative

Savsub:         T←count;
Savsuba:        temp2←L, L←0+T+1;
                MAR←temp-T;
                count←L, L←0-T;                                 dispatch on pos. value
                SINK←M, BUS, TASK;
                MD←temp2, :Sav0r;
```

```
;---------------------------------------------------------------
; Loadstate subroutine:
;       load state for emulation
;       Entry conditions:
;         L points to block from which state is to be loaded
;       Exit conditions:
;         stkp, mx, my, and stack (from base to min[stkp+2,8]) loaded
;           (i.e. two words past TOS are saved, if they exist)
;       Note:  if stkp underflows but an interrupt is taken before we detect
;         it, the subsequent Loadstate (invoked by Mgo) will see 377B in the
;         high byte of stkp.  Thinking this a breakpoint resumption, we will
;         load the state, then dispatch the 377 (via brkbyte) in Xfer0, causing
;         a branch to StkUf (!)  This is not a fool-proof check against a bad
;         stkp value at entry, but it does protect against the most common
;         kinds of stack errors.
;---------------------------------------------------------------
!17,20,Lsr0,Lsr1,Lsr2,Lsr3,Lsr4,Lsr5,Lsr6,Lsr7,Lsr10,Lsr11,Lsr12,,,,,;
!1,2,Lsmax,Ldsuba;
!1,2,Lsr,BITBLTdoner;

Loadstate:      temp←L, IR←msr0, :NovaIntrOn;                   stash pointer
Lsr:            T←12, :Ldsuba;
Lsr12:          my←L, :Ldsub;
Lsr11:          mx←L, :Ldsub;
Lsr10:          stkp←L;
                T←stkp;                                         check for BRK resumption
                L←177400 AND T;                                 (i.e. bytecode in stkp)
                brkbyte←L LCY 8;                                stash for Xfer
                L←T←17.T;                                       mask to 4 bits
                L←-7+T;                                         check stkp > 6
                L←T, SH<0;
                stkp←L, T←0+T+1, :Lsmax;                        T:stkp+1
Lsmax:          T←7, :Ldsuba;
Lsr7:           stk7←L, :Ldsub;
Lsr6:           stk6←L, :Ldsub;
Lsr5:           stk5←L, :Ldsub;
Lsr4:           stk4←L, :Ldsub;
Lsr3:           stk3←L, :Ldsub;
Lsr2:           stk2←L, :Ldsub;
Lsr1:           stk1←L, :Ldsub;
Lsr0:           stk0←L, :Xfer;

Ldsub:          T←count;
Ldsuba:         MAR←temp+T;
                L←ALLONES+T;                                    decr count for next time
                count←L, L←T;                                   use old value for dispatch
                SINK←M, BUS;
                L←MD, TASK, :Lsr0;
```

```
;---------------------------------------------------------------
; DST - dump state at block starting at <LP>+alpha, reset stack pointer
;       assumes DST is A-aligned (also ensures no pending branch at entry)
;---------------------------------------------------------------

DST:            T←ib;                                           get alpha
                T←lp+T+1;
                L←nlpoffset1+T+1, TASK;                         L:lp-lpoffset+alpha
                temp←L, IR←ret0, :Savestatea;
DSTr1:          L←my, :Savsuba;                                 save my tool
DSTr:           temp←L, L←0, TASK, BUS=0, :Setstkp;             zap stkp, return to 'nextA'


;---------------------------------------------------------------
; LST - load state from block starting at <LP>+alpha
;       assumes LST is A-aligned (also ensures no pending branch at entry)
;---------------------------------------------------------------

LST:            L←ib;
                temp←L, L←0, TASK;
                ib←L;                                           make Savpcinframe happy
                IR←sr4, :Savpcinframe;                          returns to LSTr
LSTr:           T←temp;                                         get alpha back
                L←lp+T, :Loadstate;                             lp already undiddled


;---------------------------------------------------------------
; LSTF - load state from block starting at <LP>+alpha, then free frame
;       assumes LSTF is A-aligned (also ensures no pending branch at entry)
;---------------------------------------------------------------

LSTF:           T←lpoffset;
                L←lp-T, TASK;                                   compute frame base
                frame←L;
                IR←sr2, :FreeSub;

LSTFr:          T←frame;                                        set up by FreeSub
                L←ib+T, TASK, :Loadstate;                       get state from dead frame
```

```
;-------------------------------------------------------------
; E m u l a t o r   A c c e s s
;-------------------------------------------------------------


;-------------------------------------------------------------
; RR - push <emulator register alpha>, where:
;       RR is A-aligned (also ensures no pending branch at entry)
;       alpha:  1 => wdc, 2 => XTSreg, 3 => XTPreg, 4 => ATPreg,
;               5 => OTPreg, 6 => clockreg, 7 => cp
;-------------------------------------------------------------
!7,10,RR0,RR1,RR2,RR3,RR4,RR5,RR6,RR7;

RR:             SINK←ib, BUS;                                   dispatch on alpha
RR0:            :RR0;

RR1:            T←wdc, :pushTA;
RR2:            T←XTSreg, :pushTA;
RR3:            T←XTPreg, :pushTA;
RR4:            T←ATPreg, :pushTA;
RR5:            T←OTPreg, :pushTA;
RR6:            T←clockreg, :pushTA;
RR7:            T←cp, :pushTA;


;-------------------------------------------------------------
; WR - emulator register alpha ← <TOS> (popped), where:
;       WR is A-aligned (also ensures no pending branch at entry)
;       alpha: 1 => wdc, 2 => XTSreg
;-------------------------------------------------------------
!7,10,WR0,WR1,WR2,,,,,;

WR:             L←ret3, TASK, :Xpopsub;
WRr:            SINK←ib, BUS;                                   dispatch on alpha
WR0:            TASK, :WR0;

WR1:            wdc←L, :nextA;
WR2:            XTSreg←L, :nextA;


;-------------------------------------------------------------
; JRAM - JMPRAM for Mesa programs (when emulator is in ROM1)
;-------------------------------------------------------------

JRAM:           L←ret2, TASK, :Xpopsub;
JRAMr:          SINK←M, BUS, SWMODE, :next;                     BUS applied to 'nextBa' (=0)
```

```
;---------------------------------------------------------------
; P r o c e s s / M o n i t o r  S u p p o r t
;---------------------------------------------------------------
!1,1,MoveParms1;                                        shake B/A dispatch
!1,1,MoveParms2;                                        shake B/A dispatch
!1,1,MoveParms3;                                        shake B/A dispatch
;!1,1,MoveParms4;                                       shake B/A dispatch


;---------------------------------------------------------------
; ME,MRE - Monitor Entry and Re-entry
;---------------------------------------------------------------

ME:             IR←3, :MoveParms1;                      1 parameter
MRE:            IR←4, :MoveParms2;                      2 parameters


;---------------------------------------------------------------
; MXW,MXD - Monitor Exit and either Wait or Depart
;---------------------------------------------------------------

MXW:            IR←5, :MoveParms3;                      3 parameters
MXD:            IR←6, :MoveParms1;                      1 parameter


;---------------------------------------------------------------
; NOTIFY,BCAST - Awaken process(es) from condition variable
;---------------------------------------------------------------

NOTIFY:         IR←7, :MoveParms1;                      1 parameter
BCAST:          IR←10, :MoveParms1;                     1 parameter


;---------------------------------------------------------------
; REQUEUE - Move process from queue to queue
;---------------------------------------------------------------

REQUEUE:        IR←11, :MoveParms3;                     3 parameter


;---------------------------------------------------------------
; Parameter Transfer for Nova code linkages
;       Entry Conditions:
;         T: 1
;         IR: dispatch vector index of Nova code to execute
;---------------------------------------------------------------

;MoveParms4:    L←stk3;                                 if you uncomment this, don't
;               AC3←L, T←0+T+1;                         forget the pre-def above!
MoveParms3:     L←stk2;
                AC2←L, T←0+T+1;
MoveParms2:     L←stk1;
                AC1←L, T←0+T+1;
MoveParms1:     L←stk0;
                AC0←L;

                L←stkp-T, TASK;                         decrement stkp
                stkp←L;
                T←DISP+1, :STOP;
```

```
;---------------------------------------------------------------
; M i s c e l l a n e o u s   O p e r a t i o n s
;---------------------------------------------------------------


;---------------------------------------------------------------
; CATCH - an emulator no-op of length 2.
;       CATCH is assumed to be A-aligned (no pending branch at entry)
;---------------------------------------------------------------

CATCH:          L←mpc+1, TASK, :nextAput;                       duplicate of 'nextA'


;---------------------------------------------------------------
; STOP - return to Nova at 'NovaDVloc+1'
;       control also comes here from process opcodes with T set appropriately
;---------------------------------------------------------------
!1,1,GotoNova;                                                  shake B/A dispatch

STOP:           L←NovaDVloc+T, :GotoNova;


;---------------------------------------------------------------
; STARTIO - perform Nova-like I/O function
;---------------------------------------------------------------

STARTIO:        L←ret4, TASK, :Xpopsub;                         get argument in L
STARTIOr:       SINK←M, STARTF, :next;
```

```
;-------------------------------------------------------------
; BLT - block transfer
;       assumes stack has precisely three elements:
;         stk0 - address of first word to read
;         stk1 - count of words to move
;         stk2 - address of first word to write
;       the instruction is interruptible and leaves a state suitable
;         for re-execution if an interrupt must be honored.
;-------------------------------------------------------------
!1,1,BLTx;                                                  shakes entry B/A branch
!1,2,BLTintpend,BLTloop;
!1,2,BLTnoint,BLTint;
!1,2,BLTmore,BLTdone;
!1,2,BLTeven,BLTodd;
!1,1,BLTintx;                                               shake branch from BLTloop

BLT:            stk7←L, L←T, TASK, :BLTx;                   stk7=0 <=> branch pending
BLTx:           temp←L, :BLTloop;                           stash source offset (+1)

BLTloop:        L←T←stk1-1, BUS=0, :BLTnoint;
BLTnoint:       stk1←L, L←BUS AND ~T, :BLTmore;             L←0 on last iteration (value
;                                                           on bus is irrelevant, since T
;                                                           will be -1).

BLTmore:        T←temp-1;                                   T:source offset (0 or cp)
                MAR←stk0+T;                                 start source fetch
                L←stk0+1;
                stk0←L;                                     update source pointer
                L←stk2+1;
                T←MD;                                       source data
                MAR←stk2;                                   start dest. write
                stk2←L, L←T;                                update dest. pointer
                SINK←NWW, BUS=0, TASK;                      check pending interrupts
                MD←M, :BLTintpend;                          loop or check further

BLTintpend:     SINK←wdc, BUS=0, :BLTloop;                  check if interrupts enabled

;       Must take an interrupt if here (via BLT or BITBLT)

BLTint:         SINK←stk7, BUS=0, :BLTintx;                 test even/odd pc
BLTintx:        L←mpc-1, :BLTeven;                          prepare to back up

BLTeven:        mpc←L, L←0, TASK, :BLTodd;                  even - back up pc, clear ib
BLTodd:         ib←L, :Intstop;                             odd - set ib non-zero

;       BLT completed

BLTdone:        SINK←stk7, BUS=0, TASK, :Setstkp;           stk7=0 => return to 'nextA'


;-------------------------------------------------------------
; BLTC - block transfer from code segment
;       assumes stack has precisely three elements:
;         stk0 - offset from code base of first word to read
;         stk1 - count of words to move
;         stk2 - address of first word to write
;       the instruction is interruptible and leaves a state suitable
;         for re-execution if an interrupt must be honored.
;-------------------------------------------------------------
!1,1,BLTCx;                                                 shake B/A dispatch

BLTC:           stk7←L, :BLTCx;                             if BLT were odd, we could use:
BLTCx:          L←cp+1, TASK, :BLTx;                        BLTC:   T←cp+1, TASK, :BLT;
```

```
;-------------------------------------------------------------
; BITBLT - do BITBLT using ROM subroutine
;       If BITBLT A-aligned, B byte will be ignored
;-------------------------------------------------------------
!1,1,BITBLTx;                                                   shake B/A dispatch
!7,1,DoBITBLTx;                                                 shake IR← dispatch
!3,4,Mstop,,NovaIntrOff,DoBITBLT;                               includes NovaIntrOff returns

BITBLT:         stk7←L, :BITBLTx;                               save even/odd across ROM call
BITBLTx:        L←stk0, TASK;
                AC2←L;                                          stash descriptor table
                L←stk1, TASK;
                AC1←L;
                SINK←wdc, BUS=0;                                check if Mesa interrupts off
                IR←sr3, :NovaIntrOff;                           if so, shut off Nova's
DoBITBLT:       L←BITBLTret, SWMODE, :DoBITBLTx;                get return address
DoBITBLTx:      PC←L, L←0, :ROMBITBLT;                          L←0 for Alto II ROM0 "feature"

BITBLTdone:     IR←sr1, :NovaIntrOn;                            ensure Nova interrupts are on
BITBLTdoner:    brkbyte←L, BUS=0, TASK, :Setstkp;               don't bother to validate stkp

BITBLTintr:     L←AC1, TASK;                                    pick up intermediate state
                stk1←L, :BLTint;                                stash instruction state
```

```
;-------------------------------------------------------------
; M e s a / N o v a   C o m m u n i c a t i o n
;-------------------------------------------------------------


;-------------------------------------------------------------
; Subroutines to Enable/Disable Nova Interrupts
;-------------------------------------------------------------
; !3,4,Mstop,,NovaIntrOff,DoBITBLT;                          appears with BITBLT
; !1,2,Lsr,BITBLTdoner;                                      appears with LoadState
!7,1,NovaIntrOffx;                                           shake IR← dispatch

NovaIntrOff:    T←100000;                                    disable bit
NovaIntrOffx:   L←NWW OR T, TASK, IDISP;                     turn it on, dispatch return
                NWW←L, :Mstop;

NovaIntrOn:     T←100000;                                    disable bit
                L←NWW AND NOT T, IDISP;                      turn it off, dispatch return
                NWW←L, L←0, :Lsr;


;-------------------------------------------------------------
; IWDC - Increment Wakeup Disable Counter (disable interrupts)
;-------------------------------------------------------------
!1,2,IDnz,IDz;

IWDC:           L←wdc+1, TASK, :IDnz;                        skip check for interrupts


;-------------------------------------------------------------
; DWDC - Decrement Wakeup Disable Counter (enable interrupts)
;-------------------------------------------------------------
!1,1,DWDCx;

DWDC:           MAR←WWLOC, :DWDCx;                           OR WW into NWW

DWDCx:          T←NWW;
                L←MD OR T, TASK;
                NWW←L;
                SINK←ib, BUS=0;
                L←wdc-1, TASK, :IDnz;

;  Ensure that one instruction will execute before an interrupt is taken

IDnz:           wdc←L, :next;
IDz:            wdc←L, :nextAdeaf;


;-------------------------------------------------------------
; Entry to Mesa Emulation
;       AC0 holds address of current process state block
;       Location 'PSBloc' is assumed to hold the same value
;-------------------------------------------------------------

Mgo:            L←AC0, :Loadstate;
```

```
;-------------------------------------------------------------
;  N o v a   I n t e r f a c e
;-------------------------------------------------------------
$START          $L004020,0,0;                                   Nova emulator return address


;-------------------------------------------------------------
;  Transfer to Nova code
;       Entry conditions:
;         L contains Nova PC to use
;       Exit conditions:
;         Control transfers to ROM0 at location 'START' to do Nova emulation
;         Nova PC points to code to be executed
;         Except for parameters expected by the target code, all Nova ACs
;           contain garbage
;         Nova interrupts are disabled
;-------------------------------------------------------------

GotoNova:       PC←L, IR←msr0, :NovaIntrOff;                    stash Nova PC, return to Mstop


;-------------------------------------------------------------
;  Control comes here when an interrupt must be taken.  Control will
;  pass to the Nova emulator with interrupts enabled.
;-------------------------------------------------------------

Intstop:        L←NovaDVloc, TASK;                              resume at Nova loc. 30B
                PC←L, :Mstop;

;-------------------------------------------------------------
;  Stash the Mesa pc and dump the current process state,
;  then start fetching Nova instructions.
;-------------------------------------------------------------

Mstop:          IR←sr2, :Savpcinframe;                          save mpc for Nova code
Mstopr:         MAR←CurrentState;                               get current state address
                IR←ret1;                                        will return to 'Mstopc'
                L←MD, :Savestate;                               dump the state

; The following instruction must be at location 'SWRET', by convention.

Mstopc:         SWMODE;
                :START;                                         off to the Nova ...
```