```
comment set_interrupt and set_code;

; set_interrupt is used to set the interrrupt conditions by which may
; get an MINT message back

set_interrupt:
        skipn   inited
        pushj   p,init
        pushj   p,newptr
        movei   a,MCONOB                ; cono-b command
        movem   a,@b
        pushj   p,newptr
        setzm   @b
        hrli    b,p_int                 ; the high order bit actually
        move    a,-2(p)                 ; enable/disable
        dpb     a,b
        move    a,-1(p)                 ; viva la causa
        hrli    b,p_cause
        dpb     a,b
        jrst    exit3

; set_code is used to transmit a code word via the command stream

set_code:
        skipn   inited
        pushj   p,init
        pushj   p,newptr
        movei   a,MDTA                  ; Datao-a command
        movem   a,@b
        pushj   p,newptr
        move    a,-1(p)                 ; the code word
        movem   a,@b
        jrst    exit2
```

```
subttl  Get procedure

; Called: get(id,number[optional])
; Returns: identifer

get:    skipn   inited          ; has it been inited?
         pushj  p,init          ; go ahead and do it
        move    c,-2(p)         ; pop id
        andi    c,7             ; remove optional mask
        caig    c,id_maximum
         skipge c
          jrst  [
                movei   al,[asciz /Get: Identifier type out of range./]
                pushj   p,box_error
                jrst    .+1
                ]
        move    b,max_id(c)     ; the maximum identifier for each unit
        move    a,max_id-1(c)
        aos     a               ; a is now the minimum identifier for each unit
        move    d,-2(p)         ; check type of sum memory
        andi    d,777770
        cain    c,id_sum_memory
         jrst   getsum

        jumpn   d,[
                movei   al,[asciz /Get: Memory quadrant illegal without memory id./]
                pushj   p,box_error
                jrst    .+1
                ]
        jrst    getunit

getsum: lsh     d,3             ; multiply by 64 after normalizing
        add     a,d             ; to get the right offset
        movei   b,=63           ; maximum number of sum memory/quadrant-1
        add     b,a             ; b is now the maximum unit generator
getunit:
        skipge  d,-1(p)         ; the specific number
         jrst   getloop         ; not a specific unit, branch to search loop
        add     a,d
        camle   a,b             ; requested unit out of bounds?
         jrst   getfai          ; a losing request!
        move    b,a             ; set both bounds to specific unit
getloop:
        skipn   u_gen(a)
         jrst   [
                setom   u_gen(a)
                jrst    exit3
                ]
        camge   a,b
         aoja   a,getloop       ; and another...
getfai: outstr  [asciz /Get: Warning- can't get requested element./]
        seto    a,              ; Out of them... return -1
        jrst    exit3
```

```
subttl  Give and Decode and Relative

; give(identifier) releases the identifier

give:   skipl   a,-1(p)         ; get the unit generator number
         caile  a,max_unit      ; is it out of range
          jrst  [
                movei   al,errmes
                pushj   p,errlst
                %erstr,,[asciz /Give: Identifier out of range - /]
                %eroct,,a
                %erchr,," "
                %erdec,,a
                %erchr,,"."
                %ercrlf,,0
                -1
                movei   al,errmes
                pushj   p,box_error
                jrst    exit2
                ]
        setzm   u_gen(a)
        jrst    exit2

; decode(identifier) returns: (in a)
; 0 if a generator
; 1 if a modifier
; 2 if a sum memory location
; 3 if a delay memory port
; -1 if not any of the above
; b will contain the relative unit number

decode: move    a,-1(p)
        skipl   b,a
         caile  a,max_unit      ; completely out of range check
          jsp   a,decod1
        caig    a,=256-1        ; generator top limit
         jsp    a,decod1
decod0: caig    a,=256+=128-1   ; modifier top limit
         jsp    a,decod1
        caig    a,=256+=128+=256-1
         jsp    a,decod1
        movei   a,3             ; it MUST be a delay port
        jrst    decod2
decod1: hrrei   a,-decod0(a)
        ash     a,-1
decod2: sub     b,max_id-1(a)
        soja    b,exit2

; Relative is just decode except a and b are swapped
; (i.e., it returns the number of the identifier)

relative:
        push    p,-1(p)
        pushj   p,decode
        exch    a,b
        jrst    exit2
```

```
subttl PutCmd - put an arbitrary word in code stream

; Called: PutCmd(word)
; ***** Warning! *****
; * PutCmd does not update internal info about state of
; * box, so screwy things can happen if you don't know
; * what you're doing.
; ********************

PutCmd: pushj   p,GetPtr        ;get a home for word
        pop     p,a             ;get return address
        pop     p,(b)           ;put word in its new home
        jrst    (a)             ;return
```

```
subttl  Bind procedure (bind_field too)

; Called: bind(identifier,name,value)
; sets up acs as follows:
; a = value (what to set "name" to)
; b = "name" (index into function)
; c = identifier type
; d = identifier relative number
; r is the pointer to the parameter descriptor block

bind:   skipn   inited            ; seen the init routine?
         pushj  p,init            ; there's always a first time
        move    a,-3(p)           ; the identifier
        push    p,a               ; push the id
        pushj   p,decode          ; find out what kind of identifier it is
        jumpl   a,[
                movei   al,[asciz /Bind: Identifier out of range./]
                pushj   p,box_error
                jrst    bind_exit
                ]
        move    c,a               ; copy unit type
        move    d,b               ; copy relative unit number
        skipl   b,-2(p)           ; get the name and
         camle  b,max_table(a)    ; test against table maximum
          jrst  range
        move    t,id_table(a)
        add     t,b               ; id_table(type)[name]
        move    a,-1(p)           ; don't forget the value
        skipe   r,(t)             ; is the address 0?
         jra    r,(r)             ; setup r with name&descr, go to approp. routine

range:  movei   al,errmes
        pushj   p,errlst
        %erstr,,[asciz /Bind: Illegal parameter type for /]
        %ertyp,,c
        %erchr,," "
        %eroct,,d
        %erchr,,"."
        %ercrlf,,0
        -1
        movei   al,errmes
        pushj   p,box_error

bind_exit:
exit4:  sub     p,[4,,4]
        jrst    @4(p)

bind_field:
        push    p,packmode        ; save the current packing mode
        move    x,-2(p)           ; get temp mode
        movem   x,packmode        ; use it while doing a bind
        push    p,-5(p)           ; copy id, name and val to top of stack
        push    p,-5(p)
        push    p,-5(p)
        pushj   p,bind            ; do a bind with temp mode set
        pop     p,packmode        ; restore old mode
        sub     p,[5,,5]          ; clear our args from stack
        jrst    @5(p)             ; return
```

```
SUBTTL Vectors to routines and parameters (Bind)

id_table:
        b_generator
        b_modifier
        b_sum_memory
        b_delay

max_table:
        e_generator             ; generator size
        e_modifier              ; modifier size
        e_sum_memory            ; sum memory
        e_delay                 ; delay ports

count ←← 0

b_generator:
        relocate(sum_memory,<[[asciz /sum/],,%GSUM],,g_sum_memory>)
        relocate(osc_mode,<[[asciz /osc_mode/],,%GMODE],,g_osc_mode>)
        relocate(mode,<[[asciz /mode/],,%GMODE],,SHARED>)
        relocate(sweep,<[[asciz /sweep/],,%GO],,EASY>)
        relocate(frequency,<[[asciz /frequency/],,%GJ],,LONG>)
        relocate(angle,<[[asciz /angle/],,%GK],,ZEROED>)
        relocate(ncosines,<[[asciz /ncosines/],,%GN],,SHARED>)
        relocate(scale,<[[asciz /scale/],,%GM],,SHARED>)
        relocate(rate,<[[asciz /rate/],,%GI],,EASY>)
        relocate(exponent,<[[asciz /exponent/],,%GQ],,LONG>)
        relocate(asymptote,<[[asciz /asymptote/],,%GL],,SHARED>)
        relocate(fm,<[[asciz /fm/],,%GFM],,g_fm>)
        relocate(run_mode,<[[asciz /run_mode/],,%GMODE],,g_run_mode>)
        relocate(envelope,<[[asciz /env_mode/],,%GMODE],,g_envelope>)
e_generator ← count

        reloc   b_generator+count+1

count ←← 0

b_modifier:
        relocate(sum_memory,<[[asciz /sum/],,%MSUM],,m_sum_memory>)
        relocate(function,<[[asciz /mode/],,%MMODE],,SHARED>)
        relocate(mode,<[[asciz /mode/],,%MMODE],,SHARED>)
;;;     relocate(mode,<[[asciz /mode&scales/],,%MHACK],,m_mode>)
        relocate(coeff0,<[[asciz /coeff0/],,%MM0],,LONG>)
        relocate(coeff1,<[[asciz /coeff1/],,%MM1],,LONG>)
        relocate(term_0,<[[asciz /term0/],,%ML0],,ZEROED>)
        relocate(term_1,<[[asciz /term1/],,%ML1],,ZEROED>)
        relocate(a_in,<[[asciz /a_in/],,%MIN],,m_in>)
        relocate(b_in,<[[asciz /b_in/],,%MRM],,m_in>)
        relocate(a_scale,<[[asciz /a_scale/],,%MSCALE],,m_a_scale>)
        relocate(b_scale,<[[asciz /b_scale/],,%MSCALE],,m_b_scale>)
        relocate(replace_sum_memory,<[[asciz /replace_sum/],,%MSUM],,m_replace_sum_memory>)
        relocate(invoke_delay_unit,<[[asciz /dly_unit/],,%MRM],,m_dly_unit>)
e_modifier ← count

        reloc   b_modifier+count+1

b_sum_memory:
e_sum_memory ← .-b_sum_memory

count ←← 0
```

```
b_delay:
        relocate(base_address,<[[asciz /base/],,%DLYX],,SHARED>)
        relocate(mode,<[[asciz /mode/],,%DLYZP],,dly_mode>)
        relocate(delay_length,<[[asciz /size/],,%DLYZP],,dly_length>)
        relocate(scale,<[[asciz /scale/],,%DLYZP],,dly_scale>)
        relocate(index,<[[asciz /index/],,%DLYY],,dly_index>)
e_delay ← count

        reloc   b_delay+count+1
```

```
subttl  Parameter descriptor blocks

; A parameter descriptor block is passed to the routines that follow
;  (EASY, LONG, SHARED, ZEROED) to guide them in their labors.  A block
;  looks like this:

; Block indices
        ; used by all
Cmnd←←0         ; All command bits - not unit, data or extend, but with disables
Data←←1         ; POINT <n data bits>,d,<data lsb>
Negate←←2       ; If signed data then -1 else 0
Width←←3        ; If signed data then ¬<2↑<n-1>-1> else ¬<2↑n-1>
        ; used by SHARED
GetOld←←4       ; H{L/R}Z b,Table(d)    Gets entry from share table
PutOld←←5       ; HR{L/R}M b,Table(d)   Puts entry into share table
Enable←←6       ; Word containing disable bit to clear
        ; used by ZEROED
OldTab←←4       ; Z Table(d)    Address of share&clear table entry
Ccmnd←←5        ; Clear command using clear bit, others disabled
        ; used by LONG
Shift←←4        ; -<#DX bits>


; Width definition macros
define signed(n)
        {-1
        ¬<<1⊛<=n-1>>-1>}
define unsigned(n)
        {0
        ¬<<1⊛<=n>>-1>}
```

```
; GENERATOR parameters

%GQ:    1b26                    ;LONG
        point 20,d,23
        signed(24)
        0,,-=4
%GJ:    2b26                    ;LONG
        point 20,d,23
        signed(28)
        0,,-=8
%GP:    6b27                    ;EASY
        point 20,d,23
        signed(20)
%GN:    1b5 ! 7b27              ;SHARED
        point 11,d,19
        unsigned(11)
        hlrz b,gnm(d)
        hrlm b,gnm(d)
        1b4
%GM:    1b4 ! 7b27              ;SHARED
        point 4,d,23
        unsigned(4)
        hlrz b,gnm(d)
        hrlm b,gnm(d)
        1b5
%GL:    1b5 ! 10b27             ;SHARED
        point 12,d,17
        signed(12)
        hrrz b,glsum(d)
        hrrm b,glsum(d)
        1b4
%GSUM:  1b4 ! 10b27             ;SHARED
        point 6,d,23
        unsigned(6)
        hrrz b,glsum(d)
        hrrm b,glsum(d)
        1b5
%GK:    11b27                   ;ZEROED
        point 20,d,23
        signed(20)
        d,,gkclr
        1b4 ! 1b5 ! 1b6 ! 12b27
%GMODE: 1b5 ! 12b27             ;SHARED
        point 10,d,16
        unsigned(10)
        hlrz b,gmodfm(d)
        hrlm b,gmodfm(d)
        1b4
%GFM:   1b4 ! 12b27             ;SHARED
        point 7,d,23
        unsigned(7)
        hlrz b,gmodfm(d)
        hrlm b,gmodfm(d)
        1b5
%GO:    13b27                   ;EASY
        point 20,d,23
        signed(20)
```

```
; MODIFIER parameters

%MM0:   30b28                   ;LONG
        point 20,d,23
        signed(30)
        0,,-=10
%MM1:   31b28                   ;LONG
        point 20,d,23
        signed(30)
        0,,-=10
%ML0:   34b28                   ;ZEROED
        point 20,d,23
        signed(20)
        d,,ml0clr
        1b4 ! 1b5 ! 1b6 ! 1b7 ! 36b28
%ML1:   35b28                   ;ZEROED
        point 20,d,23
        signed(20)
        d,,ml1clr
        1b4 ! 1b5 ! 1b6 ! 37b28
%MMODE: 1b5 ! 1b7 ! 36b28       ;SHARED
        point 5,d,12
        unsigned(5)
        hlrz b,mmdscsm(d)
        hrlm b,mmdscsm(d)
        1b4
%MSCALE:1b4 ! 1b5 ! 36b28       ;SHARED
        point 4,d,16
        unsigned(4)
        hlrz b,mmdscsm(d)
        hrlm b,mmdscsm(d)
        1b7
%MSUM:  1b4 ! 1b7 ! 36b28       ;SHARED
        point 7,d,23
        unsigned(7)
        hlrz b,mmdscsm(d)
        hrlm b,mmdscsm(d)
        1b5
%MRM:   1b5 ! 37b28             ;SHARED
        point 8,d,15
        unsigned(8)
        hlrz b,mrmin(d)
        hrlm b,mrmin(d)
        1b4
%MIN:   1b4 ! 37b28             ;SHARED
        point 8,d,23
        unsigned(8)
        hlrz b,mrmin(d)
        hrlm b,mrmin(d)
        1b5

%MHACK: 1b5 ! 36b28             ;SHARED
        point 9,d,16
        unsigned(9)
        hlrz b,mmdscsm(d)
        hrlm b,mmdscsm(d)
        1b4 ! 1b7
```

```
; DELAY parameters

%DLYX:  1b28 ! 0b30             ;SHARED
        point 16,d,19
        unsigned(16)
        hlrz b,dxy(d)
        hrlm b,dxy(d)
        0
%DLYY:  1b28 ! 1b30             ;ZEROED
        point 16,d,19
        unsigned(16)
        dyclr(d)
        0
%DLYZP: 1b28 ! 2b30             ;SHARED
        point 20,d,23
        unsigned(20)
        hlrz b,dzp(d)
        hrlm b,dzp(d)
        0

REPEAT 0,<              ; Unused for now
%DLYZ:  1b28 ! 2b30             ;SHARED
        point 16,d,19
        unsigned(16)
        hlrz b,dzp(d)
        hrlm b,dzp(d)
        0
%DLYZT: 1b28 ! 2b30             ;SHARED
        point 4,d,19
        unsigned(4)
        hlrz b,dzp(d)
        hrlm b,dzp(d)
        0
%DLYP:  1b28 ! 2b30             ;SHARED
        point 4,d,23
        unsigned(4)
        hlrz b,dzp(d)
        hrlm b,dzp(d)
        0
>;END REPEAT 0
```

```
subttl  EASY routine - for simple cases

; works for all the easy cases - no optimizing/sharing/clearing/extending

; acs are setup as follows:
; a is the value
; b is the "name" of the function (as passed to bind)
; c is the identifier type
; d is the relative number of the identifier
; r is the pointer to the parameter descriptor block

easy:   skipge  z,a             ; get width
         xor    z,Negate(r)     ; maybe signed
        tdne    z,Width(r)      ; make sure bits fit
         pushj  p,TooBig        ; else complain
        pushj   p,GetPtr        ; get a seat on the bus
        ior     d,Cmnd(r)       ; invent command
        dpb     a,Data(r)       ; put in data
        movem   d,(b)           ; stash in buffer
        jrst    bind_exit
```

```
subttl  LONG routine - for parameters using DX

; acs are setup as follows:
; a is the value
; b is the "name" of the function (as passed to bind)
; c is the identifier type
; d is the relative number of the identifier
; r is the pointer to the parameter descriptor block

DXcmnd←←40

long:   skipge  z,a             ; get width in z
         xor    z,Negate(r)     ; maybe signed
        move    t,packmode      ; split on packing mode
        caie    t,full_word
         jrst   Unfull
        tdne    z,Width(r)      ; make sure ALL the bits fit
         pushj  p,TooBig
        tlon    z,-2            ; less than 20. bits?
         jrst   DoData          ; will be right loaded
        setzb   b,z             ; (z=0) means load left
        lshc    a,@Shift(r)     ; high 20. bits in a, DX in b
        jumpe   b,DoData        ; assume DX is already zero, no command needed
        lsh     b,-4            ; put bits where they belong (4-23)
        iori    b,DXcmnd        ; invent a DX command
        push    p,b             ; b gets clobbered
        pushj   p,GetPtr        ; find a seat in the buffer
        pop     p,(b)           ; sit there
DoData: pushj   p,GetPtr        ; all together now
        ior     d,Cmnd(r)       ; invent a command
        dpb     a,Data(r)       ; dump in data bits (20. bits)
        skipn   z               ; loading left or right
         tro    d,1b27          ; make a load left command
        movem   d,(b)
        jrst    bind_exit

Unfull: tloe    z,-2            ; normal width is only 20. bits
         pushj  p,TooBig        ; and this don't make it
        caie    t,right_justified       ; right loading?
         setz   z,              ; nope, need to indicate that
        jrst    DoData
```

```
subttl  SHARED routine - for parameters with friends

; acs are setup as follows:
; a is the value
; b is the "name" of the function (as passed to bind)
; c is the identifier type
; d is the relative number of the identifier
; r is the pointer to the parameter descriptor block

; SHARED is for parameters loaded by multiple function commands.

shared: skipn   optim           ; optimizing?
         jrst   easy            ; nope, no hair
        skipge  z,a             ; usual width check
         xor    z,Negate(r)
        tdne    z,Width(r)      ; does it fit?
         pushj  p,TooBig
        xct     GetOld(r)       ; either [HLRZ b,Table(d)] or [HRRZ b,Table(d)]
        move    z,CodPtr        ; fix gruesome bug in optimizing        *OPT*
        cain    b,-1(z)         ; only merge adjacent commands          *OPT*
         jumpn  b,ShrOld        ; share old command if available
        pushj   p,GetPtr        ; else have to find a new seat
        xct     PutOld(r)       ; either [HRLM b,Table(d)] or [HRRM b,Table(d)]
        tdoa    d,Cmnd(r)       ; invent a command
ShrOld:  move   d,(b)           ; get the old command
        tdz     d,Enable(r)     ; clear the disable load bit for this parameter
        dpb     a,Data(r)       ; dump in the data
        movem   d,(b)
        jrst    bind_exit
```

```
subttl  ZEROED routine - for parameters having clear bit

; acs are setup as follows:
; a is the value
; b is the "name" of the function (as passed to bind)
; c is the identifier type
; d is the relative number of the identifier
; r is the pointer to the parameter descriptor block

; ZEROED is for parameters which may be cleared by a bit in a shared command.

zeroed: skipn   optim           ; optimizing?
         jrst   easy            ; no, no sweat
        skipge  z,a             ; get width
         xor    z,Negate(r)     ; maybe signed
        tdne    z,Width(r)      ; do bits fit?
         pushj  p,TooBig
        move    b,@OldTab(r)    ; get <clr,,old>
        move    z,CodPtr        ; fix gruesome bug in optimizing        *OPT*
        cain    z,1(b)          ; only merge adjacent commands          *OPT*
;       trne    b,-1            ; is there an old command?              *OPT*
         jrst   UseOld          ; yes, recycle it
        jumpn   a,NotClr        ; clearing?
        hlrz    b,b             ; get address of clear command          *OPT*
        cain    z,1(b)          ; only merge adjacent commands          *OPT*
         jumpn  b,SetClr        ; just set clear bit
        pushj   p,GetPtr        ; get a place to sit
        hrlm    b,@OldTab(r)    ; remember to share it
        ior     d,Ccmnd(r)      ; invent clear command
        movem   d,(b)           ; (no data)
        jrst    bind_exit

NotClr: pushj   p,GetPtr        ; get a seat
        hrrm    b,@OldTab(r)    ; for future reference
        tdoa    d,Cmnd(r)       ; invent command
UseOld:  move   d,(b)           ; get old command
        dpb     a,Data(r)       ; put in (new) data
        movem   d,(b)           ; sit down
        jrst    bind_exit

SetClr: ;hlrz   b,b             ; get address                           *OPT*
        movsi   d,(<1b6>)       ; this is the clear bit
        iorm    d,(b)           ; fixup command
        jrst    bind_exit
```

```
subttl  Generator bind routines

g_sum_memory:
        pushj   p,smquad
        caie    b,this_mod_pass⊗-3
         cain   b,last_mod_pass⊗-3
          pushj p,BadMem
        movem   a,sum_g
        jrst    shared

g_fm:
        pushj   p,chksm
        trze    a,200           ; coerce <genLast,modLast,modThis,genThis>
         trc    a,100           ;   to   <genLast,modLast,modLast,genLast>
        jrst    shared

g_run_mode:
        caile   a,<1⊗4>-1       ; too many bits?
         pushj  p,TooBig
        lsh     a,-runlsb
        dpb     a,[p_g_run_mode,,mode_g(d)]
        move    a,mode_g(d)
        jrst    shared

g_envelope:
        caile   a,<1⊗2>-1       ; too many bits?
         pushj  p,TooBig
        lsh     a,-envlsb
        dpb     a,[p_g_envelope,,mode_g(d)]
        move    a,mode_g(d)
        jrst    shared

g_osc_mode:
        caile   a,<1⊗4>-1       ; too many bits?
         pushj  p,TooBig
        dpb     a,[p_g_osc_mode,,mode_g(d)]
        move    a,mode_g(d)
        jrst    shared
```

```
subttl  Modifier bind routines

m_replace_sum_memory:
        pushj   p,smquad
        tro     a,100           ; the replace bit
        jrst    m_memory

m_sum_memory:
        pushj   p,smquad
        trz     a,100
m_memory:
        caie    b,this_gen_pass●-3
         cain   b,last_gen_pass●-3
          pushj p,BadMem
        move    t,a
        tro     t,100           ; form full memory address
        movem   t,sum_m(d)      ;  (i.e. in mod sum quad)
        jrst    shared

m_in:
        pushj   p,chksm
        jrst    shared

m_dly_unit:
        push    p,a
        pushj   p,decode
        caie    a,id_delay
         pushj  p,BadDly
        move    a,b             ; a now has the relative unit number
        jrst    shared

m_a_scale:
        caile   a,3             ; too many bits?
         pushj  p,TooBig
        dpb     a,[p_a_scale,,scl_m(d)]
        move    a,scl_m(d)
        jrst    shared

m_b_scale:
        caile   a,3             ; too many bits?
         pushj  p,TooBig
        dpb     a,[p_b_scale,,scl_m(d)]
        move    a,scl_m(d)
        jrst    shared

repeat 0,<
m_mode: ; box doesn't do this!
        aosg    mdmesp
        outstr  [asciz / Warning: "mode" will soon equal "function" in bind.
/]
        dpb     a,[point 4,scl_m(d)]    ; remember scale
>;repeat 0
        jrst    shared
```

```
subttl  Delay bind routines

dly_mode:
        caile   a,<1⊗4>-1           ; too many bits?
         pushj  p,TooBig
        dpb     a,[p_d_mode,,mode_d(d)]
        move    a,mode_d(d)
        jrst    shared

dly_length:
        caile   a,<1⊗=16>-1         ; too many bits?
         pushj  p,TooBig
        dpb     a,[p_d_size,,mode_d(d)]
        move    a,mode_d(d)
        jrst    shared

dly_scale:
        caile   a,<1⊗4>-1           ; too many bits?
         pushj  p,TooBig
        dpb     a,[p_d_scale,,mode_d(d)]
        move    a,mode_d(d)
        jrst    shared

; acs are setup as follows:
; a is the value
; b is the "name" of the function (as passed to bind)
; c is the identifier type
; d is the relative number of the identifier
; r is the pointer to the parameter descriptor block

dly_index:
        skipn   optim               ; optimizing?
         jrst   easy                ; no, no sweat
        skipge  z,a                 ; get width
         xor    z,Negate(r)         ; maybe signed
        tdne    z,Width(r)          ; do bits fit?
         pushj  p,TooBig
        move    b,@OldTab(r)        ; get <clr,,old>
        hlrz    z,b                 ; get clr by itself
        caile   z,(b)               ; is there an old load not followed by clear?
         jrst   OldInd              ; yes, recycle it
        caile   z,(b)               ; is there an old clear not followed by load?
         jumpe  a,bind_exit         ; loading base cleared index, do nothing
        pushj   p,GetPtr            ; get a seat
        hrrm    b,@OldTab(r)        ; for future reference
        tdoa    d,Cmnd(r)           ; invent command
OldInd:  move   d,(b)               ; get old command
        dpb     a,Data(r)           ; put in (new) data
        movem   d,(b)               ; sit down
        jrst    bind_exit
```

```
subttl  GetPtr, NewPtr

; Called internally only, getptr() returns a pointer to the command
; list. Note that if the command buffer is full, then the procedure
; will flush the buffer according to the command stream and then return
; a pointer to the first word

getptr: move    t,codcnt        ; get the buffer count
        caml    t,codmax        ; at the end?
         pushj  p,flshcd        ; flush if full
        move    b,codptr        ; load the buffer pointer
        aos     codptr          ; increment it too
        aos     codcnt          ; and the count
        aos     t,update_tick   ; one more update tick
        camg    t,max_update_ticks
         popj   p,              ; return in any case
        pushj   p,clrdp         ; clear dual pointers because update ticks
                                ; are all over
        movei   t,1
        movem   t,update_tick
        aos     pass            ; increment current pass number
        popj    p,              ; return in any case after reseting counters

; Called internally only as well, newptr() returns a pointer to the
; mtape command buffer.

newptr: move    t,comcnt        ; the same as above
        caml    t,commax        ; at the end?
         pushj  p,flshcm        ; flush if full
        move    b,comptr        ; load the buffer pointer
        aos     comptr          ; increment it too
        aos     comcnt          ; and the count
        popj    p,              ; return in any case
```

```
subttl  Flush and its subroutines

; flush should be called before exiting the user program. Failure
; to do so results in missing words in the end.

flush:  pushj   p,flshcd        ; flush code
        pushj   p,flshcm        ; and commands
        popj    p,

flshcd: push    p,x
        movei   x,code_stream
        pushj   p,flushc        ; flush code and
        pop     p,x
        popj    p,              ; return

flshcm: push    p,x
        movei   x,command_stream
        pushj   p,flushc        ; flush commands
        pop     p,x
        popj    p,

; flush Code of any kind; stream type is in x

flushc: skipn   bufcnt(x)       ; there are really words, right?
        popj    p,              ; might as well return
        push    p,a             ; save ac's
        push    p,b
        push    p,c
        push    p,d
        push    p,t
        skipe   a,bufprc(x)     ; is there a user defined procedure?
        jrst    ubfprc          ; yep, prepare to call it
        movn    t,bufcnt(x)     ; -number of words currently in the buffer
        hrls    t               ; -words,,-words
        hrr     t,buffer(x)     ; -words,,buffer-1
        sos     t
        movem   t,olist
        skipg   a,outcnt(x)
        jrst    [
                outstr  [asciz /Flush: No output channels!
/]
                jrst flshrn
                ]
        sos     a               ; first channel is at addr+0, not addr+1
        move    b,outchn(x)
        hrli    b,a             ; index off buffer pointer by offset
cflush:
ife nouuo,<
        move    d,[point 4,outc,12]
        move    c,@b            ; <outchn(x)>(a) gets channel number
        dpb     c,d             ; put the channel in
        xct     outc            ; and execute it
>; ife nouuo
        sojg    a,cflush

flshrn: pushj   p,clrdp         ; clear dual pointers out
        setzm   bufcnt(x)       ; start all over again
        move    a,buffer(x)     ; the buffer pointer starts again too
        movem   a,bufptr(x)
        setzm   (a)             ; don't forget to wash our hands!
        hrli    a,(a)
        move    b,bufmax(x)
        addi    b,-1(a)
        aos     a
        blt     a,(b)           ; there, now it's all clean
        pop     p,t             ; restore ac's
        pop     p,d
        pop     p,c
        pop     p,b
        pop     p,a
        popj    p,

clrdp:  setzm   dp_begin        ; prepare to clear!
        move    t,[dp_begin,,dp_begin+1]
        blt     t,dp_end        ; zero dual pointer area
        popj    p,

; Here we call a user defined procedure. It's address is in r
; and the stream is in x

ubfprc: movem   16,savblk+16    ; save ac's
        movei   16,savblk
        blt     16,savblk+15
        move    16,savblk+16    ; (if x=16 or a=16 this would matter)
        push    p,bufary(x)     ; push array pointer
        push    p,bufcnt(x)     ; push size
        pushj   p,(a)           ; call procedure
        movsi   16,savblk       ; restore ac's
        blt     16,16
        jrst    flshrn          ; and continue with the flush
```

```
subttl  Chksm, Smquad

; chksm (called internally only) checks to see if an argument to the bind
; procedure is a sum memory location; if not it checks the appropriate
; sum memory storage jar.

chksm:  push    p,a             ; for the call to decode
        pushj   p,decode
        xct     chksms(a)
        popj    p,

; Table corresponding to decode
        pushj   p,NotMem        ; unknown type
chksms: move    a,sum_g(b)      ; generator's sum memory address
        move    a,sum_m(b)      ; modifier's sum memory address
        move    a,b             ; b contains the relative sum memory location
        pushj   p,NotMem        ; delay unit

; smquad (called internally only) checks to see if an argument to the bind
; procedure is a sum memory location, then returns the quadrant in b and the
; address within the quadrant in a

smquad: push    p,a             ; for call to decode
        pushj   p,decode
        caie    a,id_sum_memory
         pushj  p,NotMem
        setz    a,
        rotc    a,-6            ; isolate quadrant, address
        rot     a,6             ; right justify address
        popj    p,
subttl  Init

; Clears all the constants and tables and then returns

initialize:
init:   setzm   data_begin      ; prepare to clear
        move    a,[data_begin,,data_begin+1]
        blt     a,data_end      ; clear out
        movei   x,max_stream    ; beginning stream number
stmini: move    a,buffer(x)     ; start of the command list
        movem   a,bufptr(x)     ; and the buffer pointer
        sojge   x,stmini        ; next stream
        move    a,[out 0,olist]
        movem   a,outc          ; the flush command
        movei   a,codsiz
        movem   a,codmax        ; store the instruction buffer size
        movei   a,comsiz
        movem   a,commax        ; and the command size too
        setom   inited          ; it's been done
        popj    p,              ; and return
```

```
subttl Error handling - Box_error, Length, Set_procedure

; Called internally only, box_error attempts to return control to a
; user specified procedure. Failing this, usererr will be called

box_error:
        pushj   p,pushacs
        push    p,al            ; save the message address
        push    p,al            ; push the message address for the pointer
        pushj   p,length        ; calculate the length
        skipe   errorl          ; has the user set a return location
         jrst   jmp_error       ; aha! So it was expected, eh?
        push    sp,a            ; push the length
        pop     p,a
        push    p,[0]           ; usererr(0,0,string)
        push    p,[1]
        hll     a,[point 7,0]
        push    sp,a            ; push the pointer
        push    sp,[0]          ; from 0 to
        push    sp,-2(sp)       ; the length
        pushj   p,usererr
        pushj   p,popacs
        popj    p,
        exit                    ; exit just in case

jmp_error:
        push    sp,a            ; push length
        pop     p,a             ; restore address
        hll     a,[point 7,0]
        push    sp,a
        pushj   p,@errorl       ; and return control
        pushj   p,popacs
        popj    p,

; set_procedure allows the user to choose a home grown procedure
; for error recovery. Called set_procedure(procedure name)

set_procedure:
        pop     p,a             ; dump the return address
        pop     p,errorl        ; and save it
        jrst    @a              ; and return

; length (called internally), takes an address to an asciz string and
; calculates the length

length: move    b,-1(p)
        hll     b,[point 7,0]   ; null pointer
        setz    a,
strlen: ildb    c,b
        jumpe   c,exit2
        aoja    a,strlen

; save and restore the ac's

pushacs:exch    0,(p)
        adjsp   p,17
        movem   0,(p)
        movei   0,-16(p)
        hrli    0,1
        blt     0,-1(p)
        move    0,-17(p)
        popj    p,

popacs: movsi   16,-17(p)
        blt     16,16
        adjsp   p,-20
        jrst    @20(p)
```

```
subttl  - TooBig, BadMem, NotMem, BadDly

; a     - contains offending data
; d     - contains offending unit #
; r     - left points to parm name, right points to descr. block
; p     - assume [test ↔ pushj] sequence, so is 2 more than losing test

TooBig: movei   a1,errmes
        pushj   p,errlst
        %ercrlf,,0
        %erstr,,[asciz /Bind: Too many bits for /]
        %erpar,,0
        %erstr,,[asciz / of /]
        %ertyp,,c
        %erchr,," "
        %eroct,,d
        %erchr,,"."
        %ercrlf,,0
        %erstr,,[asciz /      Value = /]
        %eroct,,a
        %erstr,,[asciz /, /]
        %erdec,,a
        %erchr,,"."
        %ercrlf,,0
        -1
        movei   a1,errmes
        jrst    box_error

BadMem: movei   a1,errmes
        pushj   p,errlst
        %ercrlf,,0
        %erstr,,[asciz /Bind: Attempt to write /]
        -1
        movei   a2,[asciz /ModSum /]
        caie    c,id_generator
         movei  a2,[asciz /GenSum /]
        pushj   p,errstr
        pushj   p,errlst
        %eroct,,a
        %erstr,,[asciz / from /]
        %ertyp,,c
        %erchr,," "
        %eroct,,d
        %erchr,,"."
        %ercrlf,,0
        -1
        movei   a1,errmes
        jrst    box_error

NotMem: movei   a1,errmes
        move    a2,b
        add     a2,max_id-1(a)
        addi    a2,1
        movem   a2,ErrTmp
        pushj   p,errlst
        %ercrlf,,0
        %erstr,,[asciz /Bind: Attempt to use /]
        %eroct,,ErrTmp
        %erstr,,[asciz /, /]
        %erdec,,ErrTmp
        -1
        jumpl   a,NotMel
        pushj   p,errlst
        %erchr,,"("
        %ertyp,,a
        %erchr,," "
        %eroct,,b
        %erchr,,")"
        -1
NotMel: pushj   p,errlst
        %erstr,,[asciz / as /]
        %erpar,,0
        %erstr,,[asciz / for /]
        %ertyp,,c
        %erchr,," "
        %eroct,,d
        %erchr,,"."
        %ercrlf,,0
        -1
        movei   a1,errmes
        jrst    box_error

BadDly: movei   a1,errmes
        move    a2,b
        add     a2,max_id-1(a)
        addi    a2,1
        movem   a2,ErrTmp
        pushj   p,errlst
        %ercrlf,,0
        %erstr,,[asciz /Bind: Attempt to use /]
        %eroct,,ErrTmp
        %erstr,,[asciz /, /]
        %erdec,,ErrTmp
        -1
        jumpl   a,BadDl1
        pushj   p,errlst
        %erchr,,"("
        %ertyp,,a
        %erchr,," "
        %eroct,,b
        %erchr,,")"
        -1
BadDl1: pushj   p,errlst
        %erstr,,[asciz / as delay unit for modifier /]
        %eroct,,d
        %erchr,,"."
        -1
        movei   a1,errmes
        jrst    box_error

        [asciz /unknown/]
Types:  [asciz /generator/]
        [asciz /modifier/]
        [asciz /sum memory/]
        [asciz /delay unit/]
```

```
subttl - errcrlf, errstr, erroct, errdec, errend

; a1    - byte pointer into message buffer
; a2    - info to be appended, if any
; if a purported byte pointer has a zero left half, <point 7,0> is filled in

errtmp: 0
errmes: block   40      ; 160. characters worth of space

errcrlf:tlnn    a1,-1           ; append a cr-lf
         hrli   a1,440700
        movei   a2,15           ; cr
        idpb    a2,a1
        movei   a2,12           ; lf
        idpb    a2,a1
        popj    p,

errstr: tlnn    a1,-1           ; append string pointed to by a2 into buffer
         hrli   a1,440700
        tlnn    a2,-1
         hrli   a2,440700
errst1: ildb    a3,a2
        jumpe   a3,errst2
        idpb    a3,a1
        jrst    errst1
errst2: popj    p,

erroct: skipa   a4,[=8]         ; append number in a2 printed as signed octal
errdec:  movei  a4,=10          ; append number in a2 printed as signed decimal
errnum: tlnn    a1,-1
         hrli   a1,440700
        movei   a3,"-"          ; in case of negative
        skipge  a2
         idpb   a3,a1
        movms   a2              ; get absolute value now
        jumpl   a2,errde2       ; for minus infinity
        movei   a3,"'"          ; in case of octal
        cain    a4,=8
         idpb   a3,a1
errde1: idiv    a2,a4           ; chop off low digit
        hrlm    a3,(p)          ; save that digit
        skipe   a2              ; continue if bits left
         pushj  p,errde1        ; recurse
        hlrz    a3,(p)
        addi    a3,"0"          ; make a digit
        idpb    a3,a1           ; append it
        popj    p,
errde2: movei   a3,"∞"          ; not usually handled, but we do!
        idpb    a3,a1
        popj    p,

errchr: tlnn    a1,-1           ; append character in a3
         hrli   a1,440700
        idpb    a3,a1
        popj    p,

errpar: hlrz    a2,r            ; append parameter name found thru LH of r
        jrst    errstr

errtyp: move    a2,Types(a2)    ; append type name selected by index in a2
        jrst    errstr
```

```
errend: tlnn    a1,-1           ; just append a zero byte to make good asciz
         hrli   a1,440700
        setz    a2,
        idpb    a2,a1
        popj    p,
subttl - errlst

; errlst expects a call of the form:
;       pushj   p,errlst
;       <code>,,<value>
;       <code>,,<value>
;            ...
;       -1
;
; where <code> indicates the interpretation to be given to <value>.
; good codes to use are:
;   %erstr - <value> is address of string
;   %erchr - <value> is single ascii character
;   %eroct - <value> is address of integer to print in octal (with ')
;   %erdec - <value> is address of integer to print in decimal
;   %ercrlf - <value> is ignored. appends crlf
;   %erpar - <value> is ignored. appends string pointed to by LH of r
;   %ertyp - <value> is address of index into Types. type is appended
;
; the resulting text is appended to the string pointed to by A1.  it is left
; as asciz (that is, a zero byte is forced at the end, but not in the string).

errlst: aos     a4,(p)          ; increment return address
        skipge  a2,-1(a4)       ; get next code-value pair
         jrst   errlsl          ; hit -1. that's all
        hlrz    a4,a2           ; get code
        hrrzs   a3,a2           ; isolate value
        skipge  a4,errdis(a4)   ; need contents of address?
         move   a2,(a2)         ; yes
        pushj   p,(a4)          ; call appropriate routine
        jrst    errlst
errlsl: push    p,a1            ; save string pointer
        pushj   p,errend
        pop     p,a1            ; get unincremented pointer
        popj    p,

errdis: 0,,errstr       ;%erstr
        0,,errchr       ;%erchr
        -1,,erroct      ;%eroct
        -1,,errdec      ;%erdec
        0,,errcrlf      ;%ercrlf
        0,,errpar       ;%erpar
        -1,,errtyp      ;%ertyp
```

```
comment Stream procedures - Set_output and Set_stream and Unset_output;

; set_output allows the user to set a channel for either command
; or code output to the box. The first argument will be used as an
; offset into the various tables.

set_output:
ife nouuo,<
        skipn   inited
        pushj   p,init
        move    x,-2(p)         ; which channel? ; dgl - which STREAM!?!
        skipl   x               ; if < 0 or
        caile   x,max_stream    ; > maximum then
        jrst    [
                movei   al,[asciz /Set_output: Not code or command stream/]
                pushj   p,box_error
                jrst    .+1
                ]
        skipl   a,-1(p)         ; sail channel
        caile   a,17            ; make sure it's legit
        jrst    [
                movei   al,[asciz /Set_output: Illegal channel/]
                pushj   p,box_error
                jrst    .+1
                ]
        move    z,outcnt(x)     ; number of channels
        cail    z,=15           ; more than the poor buffer?
        jrst    [
                movei   al,[asciz /Set_output: Too many channels/]
                pushj   p,box_error
                jrst    .+1
                ]
        aos     outcnt(x)       ; one more time...
        add     z,outchn(x)     ; offset start of list
        movem   a,(z)           ; stash this channel there
>;ife nouuo
exit3:  sub     p,[3,,3]
        jrst    @3(p)

; Unset_output(stream,channel) allows the user to remove a channel
; from the output list. It doesn't close the channel however, that is
; the user's responsibility.

unset_stream:
        skipn   inited
        pushj   p,init
        move    x,-2(p)         ; the stream
        skipl   x
        caile   x,max_stream    ; test for a bad stream
        jrst    [
                movei   al,[asciz /Unset_stream: bad stream number/]
                pushj   p,box_error
                jrst    .+1
                ]
        move    b,-1(p)         ; channel number
        move    a,outchn(x)     ; pointer to the array of output channels
        move    c,outcnt(x)     ; and the number of channels
        sos     c               ; from 0 to n-1
        add     a,c             ; compute the address of the last one
findoc: camn    b,(a)           ; compare against a channel
        jrst    fndoc           ; found it!
        sos     a               ; the address is now one less, going backward
        sojg    c,findoc        ; try again with one less
        jrst    exit3           ; not found, ignore

fndoc:  move    b,a
        hrls    a               ; copy the address into the left half
        aobjn   .+1             ; x is (a+c+1),,(a+c+1)
        subi    a,1             ; now it is outchn(x)+c+1 to outchn(x)+c
        sos     outcnt          ; one less output channel
        add     b,outcnt        ; stop blt at outchn(x)+outcnt-1
        blt     a,(b)           ; move everybody up by one
        jrst    exit3           ; done!

; set_stream allows the user to control several stream specific
; parameters, including the buffer flush routine

set_stream:
        skipn   inited
        pushj   p,init
        move    x,-3(p)         ; the stream
        skipl   x               ; if < 0
        caile   x,max_stream    ; or > maximum stream then
        jrst    [
                movei   a1,[asciz /Set_stream: Bad stream number/]
                pushj   p,box_error
                jrst    .+1
                ]
        move    b,-2(p)         ; the field
        skipl   b               ; the usual bounds check
        caile   x,max_sfield
        jrst    [
                movei   a1,[asciz /Set_stream: Bad field/]
                pushj   p,box_error
                jrst    .+1
                ]
        move    a,-1(p)         ; the value to set the field to
        move    z,strmtb(b)     ; get the pointer to the field
        add     z,x             ; add the stream
        movem   a,(z)           ; and store it!
        jrst    exit4
```

```
subttl  The End!
patch:  block 20
        end
```