Cross Reference

```
[2324]          4
[2340]          22
[2341]          17
[2342]          7,22
[2350]          22
[2364]          11:     11
[2420]          22
[2472]          4:      6,6,6,7
[2516]          6
[2522]          6
[2526]          6
[2532]          6:      7
[2]             6,20
[3000]          21
[3400]          7
[376]           7,21:   23
[3]             4:      4,7
[4000]          17:     23
[440]           10
[4]             5
[6000]          18
[7000]          19
[72]            11
[73777]         7
[777]           21
```

Assembly Statistics

Assembled: Thu Aug 19 13:34:54 1982

129 Seconds run time

No Errors

```
EEEEEE  H     H    AA     H     H  N     N
E       H     H   A  A    H     H  NN    N
EEEEE   HHHHHHH  A    A   HHHHHHH  N N   N
E       H     H  AAAAAA   H     H  N  N  N
E       H     H  A    A   H     H  N   NN
EEEEEE  H     H  A    A   H     H  N    N
```

Wed Oct 27 21:08:56 1982

Listing file Pipe output for Eric Hahn at bbns.

Listing file Pipe output for Eric Hahn at bbns.

Listing file Pipe output for Eric Hahn at bbns.

Wed Oct 27 21:08:56 1982

```
EEEEEE  H     H    AA     H     H  N     N
E       H     H   A  A    H     H  NN    N
EEEEE   HHHHHHH  A    A   HHHHHHH  N N   N
E       H     H  AAAAAA   H     H  N  N  N
E       H     H  A    A   H     H  N   NN
EEEEEE  H     H  A    A   H     H  N    N
```

```
EEEEEE  H     H     AA     H     H  N     N
E       H     H    A  A    H     H  NN    N
EEEEE   HHHHHHH   A    A   HHHHHHH  N N   N
E       H     H  AAAAAA    H     H  N  N  N
E       H     H  A    A    H     H  N   NN
EEEEEE  H     H  A    A    H     H  N    N
```

Wed Oct 27 21:08:56 1982

Listing file Pipe output for Eric Hahn at bbns.

Listing file Pipe output for Eric Hahn at bbns.

Listing file Pipe output for Eric Hahn at bbns.

Wed Oct 27 21:08:56 1982

```
EEEEEE  H     H     AA     H     H  N     N
E       H     H    A  A    H     H  NN    N
EEEEE   HHHHHHH   A    A   HHHHHHH  N N   N
E       H     H  AAAAAA    H     H  N  N  N
E       H     H  A    A    H     H  N   NN
EEEEEE  H     H  A    A    H     H  N    N
```

```
                .title "ARPANET System 4310"
004310          .vers.=4310
                .wide



;**  THE INTERFACE MESSAGE PROCESSOR PROGRAM
;**
;**  DEVELOPED BY BOLT BERANEK AND NEWMAN INC.
;**
;**  UNDER CONTRACT NO. DAHC15-69-C-0179
;**  SPONSORED BY THE ADVANCED RESEARCH PROJECTS AGENCY
;
;    Nowadays, of course, the bills are paid by the Defense Communication
;    Agency (DCA).
```

```
.comment
Edit history for NMFS IMP 4310.  Enter all edits like this one:

LJS/EAH, 20-Mar-82
(example)
        Initial version.  As of this date, the NMFS IMP has the new LJS
        init logic (INI.C30).  Fixed missing "STATE.IDLE" bit set in
        INI's creation of fake hosts, removed extra ERA for real hosts.

KDL, 1-Sep-82
        Initial version 4310, incorporating IMP 4305 plus PAT4305.

KDL, 17-Sep-82
        Still for version 4310.  Fixed JREG/CHKSUM stuff, fixed Off Page
        Reference in I2H, and changed I2HXDV to ALWAYS call the sw host, for
        ANY xdv code.

.endcomment
```

```
                .nmfsops
;page and section definitions
        000000      pg0 =0                  ;Pages 1,2 are the loader
                                            ;Page   3   is the configurat
        000003      noc =3
        000004      pg4 =4
        000005      pg5 =5
        000006      pg6 =6
        000007      pg7 =7
        000010      pg10=10
        000011      pg11=11
        000012      pg12=12
        000013      pg13=13
        000014      pg14=14
        000015      pg15=15
        000016      pg16=16
        000017      pg17=17
        000020      pg20=20
        000021      pg21=21
        000022      pg22=22
        000023      pg23=23
        000024      pg24=24
        000025      pg25=25
        000026      pg26=26
        000027      pg27=27
        000030      pg30=30
        000031      pg31=31
        000032      pg32=32
        000033      pg33=33
        000034      pg34=34
        000035      pg35=35
        000036      pg36=36
        000040      heap=40
;set up pc in each section
                .section pg0
        000000      .=pg0 *1000
                .section noc
        003000      .=3000                  ;noc area begins at 3000
                .section pg4
        004000      .=pg4 *1000
                .section pg5
        005000      .=pg5 *1000
                .section pg6
        006000      .=pg6 *1000
                .section pg7
        007000      .=pg7 *1000
                .section pg10
        010000      .=pg10*1000
                .section pg11
        011000      .=pg11*1000
                .section pg12
        012000      .=pg12*1000
```

```
                        .section pg13
        013000          .=pg13*1000
                        .section pg14
        014000          .=pg14*1000
                        .section pg15
        015000          .=pg15*1000
                        .section pg16
        016000          .=pg16*1000
                        .section pg17
        017000          .=pg17*1000
                        .section pg20
        020000          .=pg20*1000
                        .section pg21
        021000          .=pg21*1000
                        .section pg22
        022000          .=pg22*1000
                        .section pg23
        023000          .=pg23*1000
                        .section pg24
        024000          .=pg24*1000
                        .section pg25
        025000          .=pg25*1000
                        .section pg26
        026000          .=pg26*1000
                        .section pg27
        027000          .=pg27*1000
                        .section pg30
        030000          .=pg30*1000
                        .section pg31
        031000          .=pg31*1000
                        .section pg32
        032000          .=pg32*1000
                        .section pg33
        033000          .=pg33*1000
                        .section pg34
        034000          .=pg34*1000
                        .section pg35
        035000          .=pg35*1000
                        .section pg36
        036000          .=pg36*1000
                        .section heap
        040000          .=heap*1000
                        heapbeg:

;now assemble imp
```

```
                .stitl DEFINITIONS
                .INCLUDE def.m4


;NMFS definitions

        000010      %rdclok = 10                    ;read 100us clock
        000011      %lites = 11                     ;A=low 16 bits of lights, X=h
        000012      %memhi = 12                     ;read addr of highest macrome
        000000      %crash = 0                      ;?? crash application

        000020      %25.6ms = 16.                   ;16 ticks, 1.6ms each
        000620      %640ms = 25. * %25.6ms          ;25 ticks, 25.6ms each
        044476      %30sec = 18750.                 ;18750 ticks, 1.6ms each
        022237      %15sec = %30sec / 2
        111174      %60sec = %30sec * 2

        000000      q.forw = 0                      ;queue forward ptr
        000001      q.back = 1                      ;back
        000002      q.hedr = 2                      ;ptr to header
        000003      q.size = 3                      ;length (header only)
        000004      qhedr.size = q.size+1           ;size of a queue header
        000003      qcell.size = q.hedr+1           ;size of a queue member cell


        000003      pcb.pri     = 3                 ;pcb's priority and state wor
        000004      pcb.goad    = 4                 ;used by microcode
        000005      pcb.type    = 5                 ;type (0=software, n=device +
        000006      pcb.pc      = 6                 ;macroprogram conext - progra
        000007      pcb.a       = 7                 ;                      accumu
        000010      pcb.x       = 10                ;                      index
        000011      pcb.f       = 11                ;                      flags
        000012      pcb.b       = 12                ;                      B regi
        000013      pcb.sp      = 13                ;                      SP reg
        000014      pcb.r1      = 14                ;                      IREG
        000015      pcb.r2      = 15                ;                      JREG
        000016      pcb.time    = 16                ;wakeup time (set by TPR)
        000017      pcb.runh    = 17                ;runtime high
        000020      pcb.runl    = 20                ;runtime low
        000021      pcb.rb      = 21                ;ptr to microcode register bl
        000021      t0pcb.size = pcb.runl + 1       ;size of type 0 pcbs (non-I/O

        000022      pcb.iocb    = 22                ;ptr to current IOCB (optiona
        000023      pcb.get     = 23                ;ptr to GET queue (optional)

        000024      pcb.put     = 24                ;ptr to PUT queue (optional)


        000025      pcb.num     = 25                ;logical number of this devic
        000026      pcb.ff      = pcb.num + 1       ;first free word on non-type0

        000003      iocb.size   = 3                 ;size of transfer (bits)
        000004      iocb.addr   = 4                 ;start addr of transfer
        000005      iocb.flags  = 5                 ;flags (completion codes, etc
        000006      biocb.size = iocb.flags + 1     ;size of basic IOCB (no devic

;Flags in PCB.STATE (PCB.PRI)

        100000      state.idle = 100000             ;flag saying this PCB is idle
```

```
;Flags in IOCB.FLAGS
```

```
100000      %ioflags.always = 100000        ;interrupt always on this IOC
000017      %ioflags.cc = 17                ;completion code for the tran

;Definitions for CM1 device

000000      %cm1.nop =      0               ;no operation
000001      %cm1.reset =    1               ;reset & enable
000003      %cm1.iloop =    3               ;loop interface
000004      %cm1.mloop =    4               ;loop modem
000005      %cm1.unloop =   5               ;unloop and reset/enable mode

;XDV codes

000000      %c18.reset = 0                  ;reset abort current IOCB
000001      %c18.raise = 1                  ;raise IMP ready
000002      %c18.lower = 2                  ;lower IMP ready
000003      %c18.status = 3                 ;read 1822 CSR
000004      %c18.loop = 4                   ;loop
000005      %c18.unloop = 5                 ;unloop

;Bits in IOCB.FLAGs:

000200      %c18.eom = 200                  ;this IOCB marks EOM

;Bits in CSR

002000      %c18csr.down = 2000             ;Host ready line is down
004000      %c18csr.flap = 4000             ;Host ready line has flapped

000002      %c18csr.impr =    2             ;Imp ready line is down
000004      %c18csr.loop =    4             ;(sw only) host is looped
```

```
                .stitl system parameters, flags, bits, & codes

000020      nh=16.                      ;no of real hosts
000004      fh=4                        ;no of fake hosts
000024      th=nh+fh                    ;no of total hosts
000005      bh=5.                       ;no of back hosts
000016      ch=14.                      ;no of trunk lines
000010      nach=8.                     ;ch * nach = tot logical chan
000015      minf=ch-1                   ;min number of free bufs+1
000070      nmb=56.                     ;no of message blocks
000044      ntsb=36.                    ;no of transaction blocks
000040      nreab=32.                   ;no of reassembly blocks
000074      ptck=60.                    ;number of ticks to prop rout
000004      nspd=4                      ;# of different line spds we
000200      nnodes=128.                 ;spf- number of nodes
000540      nlines=352.                 ;spf- number of lines (simpl
000040      bufbcl=32.                  ;2*# buffers ranges that can
000020      npaks=16.                   ;maximum number of packages
000017      octmax = 15.                ;max octet number (128 chann

;noc area equates

003440      crshp=3440                  ;addr of saved crash PC addre
003443      crsha=3443                  ;addr of saved crash A-reg co
003425      crshcd=3425                 ;addr of saved crash code

;words in buffer

000000      ptrc=0                      ;chain pointer (q.forw)
000001      revp=ptrc+1                 ;reverse pointer (q.back)
000002      topp=revp+1                 ;queue top ptr (q.hedr)
000003      artm=topp+1                 ;arrival time for delay
000004      wrdc=artm+1                 ;word/use count
000005      inch=wrdc+1                 ;input channel/output timeout
000006      neth=inch+1                 ;*header - 8 words
000007      typh=neth+1                 ;*packet type
000010      chkh=typh+1                 ;*software checksum
000011      srch=chkh+1                 ;*source imp
000012      seqh=srch+1                 ;*sequence control
000013      pkth=seqh+1                 ;*packet control
000014      dsth=pkth+1                 ;*destination imp
000015      midh=dsth+1                 ;*message id
000010      hdrl=midh-neth+1            ;number of header words
000016      data=midh+1                 ;*beginning of data
000103      datal=67.                   ;maximum number of data words
000077      odatal=63.                  ;max no 316/plur compat data
000121      itst=data+datal             ;input time/sent time (tracin
000004      minpl=seqh-neth             ;minimum packet length
000113      maxpl=hdrl+datal            ;maximum packet length
000006      hedr=neth                   ;start pointer for input
000121      bufend=data+datal           ;end pointer for input
000122      bufl=bufend+1               ;buffer length
000122      rbufl=bufl
```

```
;bits in wrdc
        177400      pktsiz=177400                   ;packet size (header plus dat
        000400      pktsi1=400                      ;low bit of pktsiz
        000200      trcpkt=200                      ;1=this packet being traced
        000100      trcrrt=100                      ;1=this packet was rerouted
        000070      unused=70
        000007      usecnt=7                        ;use count for buffer, 0=free
;bits in inch
;for modem input and host input
        100000      hstmod=100000                   ;1=from host, 0=from modem
        000017      inpchn=17                       ;input source (0>ch-1 for mod
                                                    ; 0>th+bh-1 hosts)
;after task processing:
;for host output, the time (in terms of slow ticks)
;when packet will be too old - 30 secs
;for simp output, the logical output line
;for modem output, the retransmit counter
;bits in neth
        100000      odeven=100000                   ;channel odd/even ack sequenc
        040000      endbit=40000                    ;1=pkt from hi numbered imp o
        020000      dscpkt=20000                    ;ack & discard this packet
        010000      unused=10000
        007400      chn8=7400                       ;channel number for old forma
        000377      chn128=377                      ;channel number for new forma
        007400      octet=7400                      ;octet number for new format

        000376      unused=376
        000001      magicm=1                        ;1=for magic modem
        177400      sernum=177400                   ;routing serial # (routing ms
        000400      sernm1=400                      ;low bit of sernum
;bits in typh
        140000      pactyp=140000                   ;packet type:
        000000        dattyp=0                      ;0- data (message,rfnm,etc.)

        040000        nettyp=40000                  ;1- netwrk ctrl(get-a-block,r
        100000        swttyp=100000                 ;2- switch control (routing,n
        140000        sptty=140000                  ;3- special(pkt core,demand r
        020000      compat=20000                    ;odd/even compatibility for p
        000000        datcpt=0
        000000        netcpt=0
        000000        swtcpt=0
        000000        sptcpt=0
        010000      prirty=10000                    ;1=high priority
        004000      trcflg=4000                     ;1=trace this packet
        003400      lflags=3400                     ;leader flags (type 0, codes
        002000      octprt=2000                     ;1=tty octal(type 0, code 0,1
        002000      allusd=2000                     ;1=alloc8 used(type 1, code 1
        001000      hacrej=1000                     ;1=host access rejected
                                                    ; (type 1, code 15 only)
        000400      blkrej=400                      ;1=getblk rejected(type 1,cod
        001400      swtsub=1400                     ;switch type subtype field
        003000      rutcom=3000                     ;routing compat #(type 2,rout
        000000      rutcvl=0                        ;current rutcom value
        000400      nulbit=400                      ;0=routing, 1=null
        003400      sptcod=3400                     ;type 3 (special) code:
        000000        sptpkc=0                      ;0- packet core
        000400        sptdmr=400                    ;1- demand reload
        001000        sptrlr=1000                   ;2- reload request
        000377      ackbts=377                      ;8 acknowledge bits, 1/channe
```

; (type 0,1, and null only)

```
;bits in seqh
       177400     mesnum=177400          ;message number (type 0,1 onl
       000400     mesnm1=400             ;low bit of mesnm1
       000377     blknum=377             ;foreign block number (type 0
       177400     dsthst=177400          ;dest. host (uncontrolled, ge
       000377     srchst=377             ;source host(uncontrolled, ge

;bits in pkth
       100000     mltpkt=100000          ;1=multi-packet
                                         ;(type 0,code 0-3,type 1,code
       040000     lstpkt=40000           ;1=last packet (type 0,codes
       030000     unused=30000
       007400     usenum=7400            ;foreign use #(type 0,codes 0
       000360     pktnum=360             ;packet number (type 0, codes
       000020     pktnm1=20              ;low bit of pktnum
       000017     pktcod=17              ;packet code(0-7=>type 0, 10-
       000000       msgcod=0             ;0- message
       000001       reqcod=1             ;1- request
       000002       gvbcod=2             ;2- giveback
       000003       inccod=3             ;3- incomplete message
       000004       rfncod=4             ;4- rfnm
       000005       allcod=5             ;5- rfnm/allocate
       000006       dedcod=6             ;6- dead rfnm
       000007       irfcod=7             ;7- incomplete rfnm
       000010       qercod=10            ;10- incomplete query
       000011       gtbcod=11            ;11- get-a-block
       000012       rstcod=12            ;12- reset
       000013       unasgn=13
       000014       oorcod=14            ;14- out-of-range
       000015       gtrcod=15            ;15- get-a-block reply
       000016       rsqcod=16            ;16- reset request
       000017       rsrcod=17            ;17- reset reply
       177400     handtp=177400          ;handling type (packet core o
       000377     desths=377             ;destination host (packet cor

;bits in midh
       177760     linkno=177760          ;link=msg id(type 0, packet c
       000017     subtyp=17              ;sub-type (type 0 only):
       000000       subreg=0             ;0- regular message
       000001       subref=1             ;1- refusable message
       000002       unasgn=2
       000003       subunc=3             ;3- uncontrolled packet
       170000     hndtyp=170000          ;local handling type (type 1
       007400     usenum=7400            ;local use number (type 1 onl
       000377     blknum=377             ;local block number (type 1 o
```

```
;words in transaction block (parallel table)
        000000      lch=0*ntsb                  ;time sent/chain ptr
        000044      lms=1*ntsb                  ;message identification
        000110      lcd=2*ntsb                  ;message code
        000154      ldi=3*ntsb                  ;destination imp
        000220      lid=4*ntsb                  ;message id/subtype
        000264      lpk=5*ntsb                  ;packet ptr/dead host stats
        000006      tsbl=6                      ;transaction block size

;bits in lms (for outstanding messages)
        177400      mesnum=177400               ;message number
        000377      blknum=377                  ;local block number (never 0)
;bits in lms (for control message queue)
        177400      msgtyp=177400               ;message type (never 0)
        000377      blknum=377                  ;(always 0)

;bits in lcd (for outstanding messages)
        177400      tsbcod=177400               ;message code:
        000000        tsbms1=0                  ;0- single-pkt msg
        000400        tsbms8=400                ;1- multi-pkt msg
        001000        tsbrq1=1000               ;2- single-pkt req
        001400        tsbrq8=1400               ;3- multi-pkt req
        002400        tsbgvb=2400               ;5- giveback
        000377      desths=377                  ;destination host
;bits in lcd (for control message queue)
        177400      handtp=177400               ;handling type
        000377      desths=377                  ;destination host

;bits in tsex and rsex
        000377      acks.=377

;words in message block
        000000      mbimp=0*nmb                 ;foreign imp (0 if free)
        000070      mbhst=1*nmb
        000160      mbfor=2*nmb
        000250      mbtim=3*nmb
        000340      mbmes=4*nmb                 ;(transmit blocks only)
        000005      tmbl=5                      ;transmit block size
        000340      mbsta=4*nmb                 ;rstate (receive blocks only)
        000430      mbtyp=5*nmb                 ;rtype (receive blocks only)

        000006      rmbl=6                      ;receive block size
```

```
;bits in mbhst
        177400      mbfhst=177400                   ;foreign host
        000377      mblhst=377                      ;local host

;bits in mbfor
        170000      mbhand=170000                   ;handling type
        007400      mbfrnu=7400                     ;foreign use number
        000377      mbfrnb=377                      ;foreign block no

;bits in mbtim
        177400      mesnum=177400                   ;message number
                                                    ;transmit - latest message #
                                                    ;receive - oldest incomplete
        000400      mesnm1=400                      ;low bit of messno
        000360      mbusno=360                      ;local use number
        000020      mbuse1=20                       ;low bit of mbusno
        000017      mbage=17                        ;age counter

;bits in mbmes
        100000      mbrst=100000                    ;1=reset
        040000      mbinit=40000                    ;1=initializing
        020000      mbstp=20000                     ;1=wait to clear pipe
        016000      mballc=16000                    ;# of allocates we have
        002000      mball1=2000                     ;low bit of mballc
        001400      mbinct=1400                     ;incomplete t/o
        000400      mbinc1=400                      ;low bit of mbinct
        000377      mesbts=377                      ;message bits- 0=message sent
                                                    ;a,b,c, ... where a:msg-1, b:

;bits in mbsta and mbtype
;state and type words contain 8 2-bit entries, numbered
;i=1,...,8 (left to right), where entry #i corresponds to
;either message number rmess-i (previous window, *p*), or
;to message number rmess+8-i (current window, *c*), depending
;on the values of state and type. the meanings of the various
;combinations, along with the window (*p* or *c*), is given
;by the following table:
;
;state->   idle        request      message       reply
;           00          01           10           11
;type !
;     v
;  00    rfnm sent    req1 rcvd    all1 sent/    rfnm1 to
;          *p*          *c*        msg rcvd      be sent
;                                  *c*,*p*         *p*
;
;                                                rfnm8 to
;  01    all8 sent    req8 rcvd    gvb rcvd      be sent
;          *p*          *c*          *c*           *p*
;
;                     all1 to                    dead to
;  10    dead sent    be sent      dead rcvd     be sent
;          *p*          *c*          *c*           *p*
;
;                     all8 to                    inc to
;  11    inc sent     be sent      inc rcvd      be sent
;          *p*          *p*          *c*           *p*
```

```
;physical, logical fake host numbers
        000020      fhptty=nh+fh-4
        000374      fhltty=374                  ;teletype
        000021      fhpddt=nh+fh-3
        000375      fhlddt=375                  ;ddt
        000022      fhptrc=nh+fh-2
        000376      fhltrc=376                  ;trace
        000022      fhpprm=nh+fh-2
        000376      fhlprm=376                  ;parameter change
        000022      fhphtm=nh+fh-2
        000376      fhlhtm=376                  ;htm
        000022      fhppkc=nh+fh-2
        000376      fhlpkc=376                  ;packet core
        000023      fhpsts=nh+fh-1
        000377      fhlsts=377                  ;statistics
        000023      fhpdis=nh+fh-1
        000377      fhldis=377                  ;discard

;words in reassembly block (parallel table)
        000000      rch=0*nreab                 ;chain ptr
        000040      rms=1*nreab                 ;zero if free
        000100      rct=2*nreab
        000003      reasl=3                     ;reas block size

;bits in rms
        177400      rmes=177400                 ;mess no
        000377      rblk=377                    ;receive block number

;bits in rct
        177400      rmax=177400                 ;pkts max
        000377      rsf=377                     ;pkts so far
```

```
;leader formats - new leader
;first leader word
        170000      unused=170000
        007400      ldrnlf=7400                 ;new leader flag
        000377      unused=377                  ;for network-later
;second leader word
        170000      unused=170000
        004000      ldrtfl=4000                 ;trace
        003400      ldrlfl=3400                 ;leader flags
        002000              ldroct=2000         ;octal print
        001000              ldrrut=1000         ;1=route by spf
        000400              ldrtag=400          ;1=tag packet with impno and
        000377      ldrmty=377                  ;message type
        000000              lcreg=0             ;hi regular
        000001              lcerwo=1            ;hi error w/o id
        000002              lchgdn=2            ;hi host going down
        000004              lcnop=4             ;hi nop
        000010              lcerwi=8.           ;hi error with id
;third leader word
        177400      ldrhnd=177400               ;handling type
        100000              ldrhpr=100000       ;priority
        007400              ldrhln=7400         ;max length
        000377      ldrhst=377                  ;destination host
;fourth leader word
        177777      ldrimp=177777               ;destination imp
;fifth leader word
        177760      ldrmid=177760               ;message-id
        000017      ldrsty=17                   ;subtype
        000000              lcnorm=0            ;type 0 normal
        000001              lcrefs=1            ;type 0 refusable
        000002              lcgetr=2            ;type 0 get ready
        000003              lcunct=3            ;type 0 uncontrolled
;sixth leader word
        177777      ldrlen=177777               ;message length

;ih codes - for transaction blocks
        000000      creg=0                      ;reg
        000400      cerrld=400                  ;error in hi leader
        000000              cerr32=0            ;error in first 32 bits
        000001              cshort=1            ;less than 32 bits in message
        000002              cillgl=2            ;illegal hi code
        001000      cimpdn=1000                 ;imp going down
        140000              dwn.wy=140000       ;why going down
        036000              dwn.wn=036000       ;when going down
        001777              dwn.lg=001777       ;how long will be down
        002000      cnop=2000                   ;nop
        002400      crfnm=2400                  ;rfnm
        003000      chstst=3000                 ;dead host status
        003400      cdestd=3400                 ;destination dead
        000000              cimpd=0             ;dest imp dead
        000001              chstd=1             ;dest host dead
        000003              cnhac=3             ;host access rejected
        004000      cerrdt=4000                 ;error in hi data
        004400      cinctr=4400                 ;incomplete trans
        000000              cslowd=0            ;dest host took >30 secs
        000001              clong=1             ;more than 8095 bits
        000002              cslows=2            ;source host took >15 secs
        000003              clost=3             ;lost in subnet
        000004              cblock=4            ;source imp took > 15 secs
```

```
                                ;goes before tsb is assigned
       000004          cerror=4                ;error bit set (becomes cerrd
       000005          cdmcer=5                ;like hi data error-- dmc scr
       005000    creset=5000                   ;imp-to-host reset - ready li

 ;number of hstat words (for trouble reports)
       000004    hstatsize=(nh-1)/4+1

 ;bits in hdown (host down data)
       160000    hstday=160000
       017400    hsthr=17400
       000360    hstmin=360
       000017    hstwhy=17

 ;Values for HOST(num) - the public host state (see I2H for more info)
       000000    hstoff=0                      ;not configured or down
       000001    hstup=1                       ;up
       000002    hsttdy=2                      ;tardy
       000003    hstigd=3                      ;imp-going-down

 ;Interrupt levels
       000061    m2i=61                        ;modem-to-imp
       000062    i2m=m2i+1                     ;imp-to-modem
       000063    i2h=i2m+1                     ;imp-to-host
       000064    h2i=i2h+1                     ;host-to-imp
       000065    t.o=h2i+1                     ;timeout
       000066    spf=t.o+1                     ;routing
       000067    tsk=spf+1                     ;task
       000070    dpy=tsk+1                     ;display driver
       000071    bck=dpy+1                     ;background

 ;level 0
       000060    ini=60                        ;initialization
       000060    tty=60                        ;teletype interrupt
       000060    all=60                        ;uninterruptable code like dx
       000060    fre=60                        ;references to free list and

 ;Miscellaneous levels
       000126    var=126                       ;variables
       000103    con=103                       ;constants
       000125    und=125                       ;undefined

 ;config and init values
       000001    m2ipri=1                      ;m2i priority
       000002    i2mpri=m2ipri+1               ;i2m priority
       000003    i2hpri=i2mpri+1               ;i2h priority
       000004    h2ipri=i2hpri+1               ;h2i priority

       003000    configtrunk=3000              ;address of first trunk confi
       003200    confighost=3200               ;address of first host config
       003400    config.num=3400               ;address of imp number
       003416    hacchk=3416                   ;address of HACCHK word
       003417    hacsum=3417                   ;address of HACSUM word
       000237    basicm2isize=slots

       000010    cf.size=10                    ;size of a config block
       000000    cf.trtype=0
       000001    cf.nchan=1
       000002    cf.delay=2
```

```
        000005      cf.speed=5

        000000      cf.htype=0
        000001      cf.haccom=1
        000002      cf.hacmem=2

        000006      cf.typerb=6
        000077      cf.typemask=77
        177700      cf.rbmask=177700

;NCC Commands
        000000      ncc.null=0                  ;nothing to do, idle
        000001      ncc.lmod=1                  ;loop modem
        000002      ncc.imod=2                  ;loop interface
        000003      ncc.umod=3                  ;unloop modem/interface
        000004      ncc.lhst=4                  ;loop host [interface]
        000005      ncc.uhst=5                  ;unloop/reset host [interface
        000006      ncc.stop=6                  ;nice-stop/%crash
        000007      ncc.boot=7                  ;re-boot from tape
        000010      ncc.rest=10                 ;re-start IMP
        000011      ncc.reld=11                 ;panic-stop/%crash

;package bit equates
        000001      pbit00=1                    ;vdh package
        000002      pbit01=2
        000004      pbit02=4
        000010      pbit03=10                   ;stats package
        000020      pbit04=20                   ;trace package
        000040      pbit05=40                   ;tty package
        000100      pbit06=100                  ;ddt package
        000200      pbit07=200                  ;hdlc package
        000400      pbit10=400                  ;hdh package
        001000      pbit11=1000
        002000      pbit12=2000
        004000      pbit13=4000
        010000      pbit14=10000
        020000      pbit15=20000
        040000      pbit16=40000
        100000      pbit17=100000               ;experimental package
```

```
                        .stitl PAGE 0 USAGE
                        .INCLUDE pg0.m4


                        .section pg0
        000001          . = 1                           ;skip INDEX

;IREG and JREG are global registers (much like A, B, X) which are
;saved by NMFS across interrupts. Note that JREG doubles as the entry point
;to the checksum routine.

                        .lev var
000001 000000 V         ireg:   0                       ;IREG and JREG are temps
                        jreg:                           ;JREG is also entry point to
000002 000000 V         chksum: 0
000003 102004 V                 jmp .+1 i
000004 031221 V                  chksm2

                        .lev con
;then come some global constants (.pzcons)

000005 000077 C         c77:    .pzcon 77
000006 000000 C         zero:   .pzcon 0
000007 000377 C         maskh:  .pzcon 377
000010 177400 C         lefth:  .pzcon 377<<8.
000011 000003 C         three:  .pzcon 3
000012 000005 C         five:   .pzcon 5
000013 000007 C         seven:  .pzcon 7
000014 177777 C         minus1: .pzcon -1
000015 177776 C         minus2: .pzcon -2
000016 177775 C         minus3: .pzcon -3
000017 177774 C         minus4: .pzcon -4
000020 177773 C         minus5: .pzcon -5
000021 177772 C         minus6: .pzcon -6
000022 177770 C         min10:  .pzcon -10
000023 177700 C         min100: .pzcon -100
000024 000011 C         c11:    .pzcon 11
000025 000012 C         c12:    .pzcon 12
000026 000014 C         c14:    .pzcon 14
000027 000015 C         c15:    .pzcon 15
000030 177771 C         minus7: .pzcon -7
000031 000006 C         six:    .pzcon 6
000032 177760 C         min20:  .pzcon -20
000033 000777 C                 .pzcon 777
000034 001400 C                 .pzcon 1400
000035 160000 C                 .pzcon 160000
000036 000070 C                 .pzcon 70
000037 140000 C                 .pzcon 140000
000040 050000 C                 .pzcon 50000
000041 003400 C                 .pzcon 3400
000042 036000 C                 .pzcon 36000
000043 100017 C                 .pzcon 100017
000044 037777 C                 .pzcon 37777
000045 007777 C                 .pzcon 7777
000046 040500 C                 .pzcon 40500
000047 037400 C                 .pzcon 37400
000050 070000 C                 .pzcon 70000
000051 007000 C                 .pzcon 7000
```

```
                        bittab:
000052 000001 C         one:    .pzcon 1
000053 000002 C         two:    .pzcon 2
000054 000004 C         four:   .pzcon 4
000055 000010 C         ten:    .pzcon 10
000056 000020 C                 .pzcon 20
000057 000040 C                 .pzcon 40
00006C 000100 C         c100:   .pzcon 100
000061 000200 C                 .pzcon 200
000062 000400 C                 .pzcon 400
000063 001000 C                 .pzcon 1000
000064 002000 C                 .pzcon 2000
000065 004000 C                 .pzcon 4000
000066 010000 C                 .pzcon 10000
000067 020000 C                 .pzcon 20000
00007C 040000 C                 .pzcon 40000
000071 100000 C         sign:   .pzcon 100000

000072 000017 C         maskq:  .pzcon 17
000073 000360 C                 .pzcon 360
000074 007400 C                 .pzcon 7400
000075 170000 C                 .pzcon 170000
000076 177721 C         m30sec: .pzcon -47.             ;number of 640ms ticks in 30

;fixed pg0 NOC area
        000100          . = 100
                        .lev var
                        nocpg0b:
000100          V       ncca:   .block 1                ;arg word
000101          V       nccc:   .block 1                ;command word
000102          V       paks:   .block 1                ;packages in this IMP
000103          V       gopoke: .block 1                ;non-zero = PCB ptr of someon
000104          V       pkcrld: .block 1                ;>0 => pass "setup send" on t
000105          V       vers:   .block 1                ;version number
000106          V       mine:   .block 1                ;my imp number
000107          V       view:   .block 1                ;non-zerp = address to put in
                        nocpg0e:

                        .lev con

                                                        ;address constants
00011C 012000 C         jam:    gam                     ;give a word from fake host t
000111 014111 C         doze:   bkx                     ;jam wait
000112 013621 C         suck:   suk                     ;get a word for fake host fr
000113 014117 C         wait:   bkw                     ;suck wait
000114 014134 C         sleep:  bkz                     ;back host wait
000115 031125 C         flushi: flush                   ;pntr to subr to free buffer

000116 031157 C         getfri: getfre                  ;ptr to subr to get free buf

000117 031176 C         maxchi: maxchk                  ;ptr to subr to check reassem
000120 031040 C         hltjst: .hltjs                  ;ptr to subr to call hltncc
000121 031061 C         getqi:  getq                    ;ptr to subr to get pkt off q
000122 031103 C         mgetqi: mgetq                   ;ptr to subr to get pkt off m
000123 022267 C         killii: killin                  ;ptr to subr to kill line
000124 024157 C         toret:  to6                     ;pointer to t.o loop return a
000125 036054 C         fndni:  fndn                    ;find neighbor subroutine
000126 027124 C         hltncc: hltwrd                  ;pntr to halt reporting subr
```

```
;indirect table pointers
;the next two must be in order for htpsub ?? IHSP, HISP

000127 042334 C   ltbi:   ltb                    ;spf line table
000130 043074 C   ltbei:  ltb+nlines             ;spf ptr: neg index into line
000131 043074 C   spfrti: spfrut                 ;spf new routing table
000132 043275 C   spfrei: spfrut+nnodes+1        ;spf pointer for negative inc
```

```
;NMFS pointers, etc.

000133 045135 C   idlqi:  idleq                 ;ptr to idle queue (and whol
000134 045351 C   bckpci: bckpcb                ;BCK's PCB
000135 045372 C   dpypci: dpypcb                ;DPY's PCB
000136 045413 C   tskpci: tskpcb                ;TSK's PCB
000137 045434 C   spfpci: spfpcb                ;SPF's PCB
000140 045455 C   timpci: timpcb                ;TIM's PCB
000141 046172 C   m2ipci: m2ipct                ;pointer to PCB pointer tabl
000142 046210 C   i2mpci: i2mpct
000143 046226 C   i2hpci: i2hpct
000144 046252 C   h2ipci: h2ipct

;package hooks area - for extra hooks besides inixx and bckxx

;trace package hooks
000145 005651 C   trace.m2idone:  jstm2i
000146 010333 C   trace.hitrch:   jsth2i
000147 014311 C   trace.fh2:      jstbck
000150 016265 C   trace.task:     jsttsk

;Page zero zeroed area -- cleared on restart.

                  p0zb:                         ;first section is useful NCC

                  .lev var
000151        V   time:   .block 1              ;time in fast ticks (permane
000152        V   times:  .block 1              ;time in slow ticks
000153        V   sync:   .block 1              ;time for stat routines
000154        V   ludflg: .block 1              ;line up/down flag for bck
000155        V   sp306:  .block 1              ;send 306 type msg flag for b
000156        V   maxs:   .block 1              ;max S/F limit (IMPDEFed)
000157        V   maxr:   .block 1              ;max R.A. limit (#buffs minus
000160        V   tsklck: .block 1              ;TASK vs. TIMEOUT RESET/R int
000161        V   reqstk: .block 1              ;stack of out-of-order REQs
000162        V   i2ddt:  .block 1              ;character window from imp ->
000163        V   hltloc: .block 1              ;trap cache
000164        V   hlta:   .block 1
000165        V   hltx:   .block 1
000166        V   this:   .block 1              ;current buffer
000167        V   thisb:  .block 1              ;current message block
000170        V   blkflg: .block 1              ;block check flag (0=ok)
000171        V   rngflg: .block 1              ;range check flag (0=ok)
000172        V   source: .block 1              ;source imp for this packet
000173        V   seqnum: .block 1              ;seq no for this packet or n
000174        V   mestb1: .block 1              ;ptr to mess #
000175        V   mestb2: .block 1              ;ptr to mesbits or rstate
000176        V   lochst: .block 1              ;which physical host we are
000177        V   tot:    .block 1              ;timeout routines temp
000200        V   diagq:  .block 1
000201        V   mbl:    .block 1              ;modem block pointer for m2i

000202        V   ombl:   .block 1              ;modem block pointer for i2m

000203        V   ourmb:  .block 1              ;modem block pointer for task
000204        V   zmbl:   .block 1              ;modem block pointer for line

000205        V   endptd: .block 1              ;destination node
```

```
000206          V   endpts: .block 1                ;source node
000207          V   deltad: .block 1                ;change in distance over linc
```

```
000210          V   update: .block 1                ;new distance for line
000211          V   naylpe: .block 1                ;ending index for neighbor se
000212          V   naylpx: .block 1                ;current index for neighbor s
000213          V   rtgdwn: .block 1

000214          V   nutemp: .block 2                ;temp storage for NU and pack

;Don't move these without checking routines like JUQC in timeout, also
;BACK4 depends on NSF and NRE having the same relationship as SRRQ and
;SRPQ.

                counta:
000216          V   nfa:    .block 1                ;free list
000217          V   nsfa:   .block 1                ;store-and-forward count
000220          V   nrea:   .block 1                ;reassembly count
000221          V   nala:   .block 1                ;allocate count

                counts:
000222          V   nfs:    .block 1
000223          V   nsfs:   .block 1
000224          V   nres:   .block 1
000225          V   nals:   .block 1

;init variables
000226          V   pcbaddr: .block 1
000227          V   pcblen:  .block 1
000230          V   cfblock: .block 1
000231          V   cfword:  .block 1

                p0ze:                               ;end of zeroed page 0
```

```
                .stitl NOC and CONFIG area
                .INCLUDE noc.m4


                .section noc
003000          V   noc.trunk0:     .block 10
003010          V   noc.trunk1:     .block 10
003020          V   noc.trunk2:     .block 10
003030          V   noc.trunk3:     .block 10
003040          V   noc.trunk4:     .block 10
003050          V   noc.trunk5:     .block 10
003060          V   noc.trunk6:     .block 10
003070          V   noc.trunk7:     .block 10
003100          V   noc.trunk10:    .block 10
003110          V   noc.trunk11:    .block 10
003120          V   noc.trunk12:    .block 10
003130          V   noc.trunk13:    .block 10
003140          V   noc.trunk14:    .block 10
003150          V   noc.trunk15:    .block 10
003160          V   noc.trunk16:    .block 10       ;currently unavailable
003170          V   noc.trunk17:    .block 10       ;currently unavailable

003200          V   noc.host0:       .block 10
003210          V   noc.host1:       .block 10
003220          V   noc.host2:       .block 10
003230          V   noc.host3:       .block 10
003240          V   noc.host4:       .block 10
003250          V   noc.host5:       .block 10
003260          V   noc.host6:       .block 10
003270          V   noc.host7:       .block 10
003300          V   noc.host10:       .block 10
003310          V   noc.host11:       .block 10
003320          V   noc.host12:       .block 10
003330          V   noc.host13:       .block 10
003340          V   noc.host14:       .block 10
003350          V   noc.host15:       .block 10
003360          V   noc.host16:       .block 10
003370          V   noc.host17:       .block 10

003400          V   noc.rbconfig:   .block 20
003420          V   noc.rbstate:    .block 20
003440          V   noc.rbmain:     .block 20
003460          V   noc.rbclock:    .block 20
003500          V   noc.rbioos:     .block 20
003520          V   noc.rbbase:     .block 20

003540          V                   .block 40       ;currently unused
003600          V   noc.neighb:     .block 20
003620          V   noc.line:       .block 20
003640          V   noc.host:       .block th

003664          V                   .block 40-th    ;currently unused

003700          V   noc.lecs:       .block 20       ;line error counters

;individual nco locations
;tbdi
        000120      noc.zsize = . - noc.neighb
```

```
;layout of noc.rbconfig
        003400      .=noc.rbconfig

003400          V   noc.mine:       .block 1        ;imp number
003401          V                   .block 1        ;imp serial number
003402          V                   .block 1        ;ucode version number
003403          V                   .block 1        ;usys version
003404          V   noc.spfabt:     .block 1        ;low bit=1 => crash on spfabt
003405          V   noc.nocimp:     .block 1        ;imp for noc host
003406          V   noc.nochst:     .block 1        ;host for noc host
003407          V   noc.noclnk:     .block 1        ;link for noc host
```

```
                .stitl IMP INITIALIZATION
                .INCLUDE ini.m4
;NMFS notes:



;This is the top of IMP initialization. We come here from LOD after the
;transfer is complete. First, we reset NMFS and the I/O world. Then, we
;Clear all of the HEAP storage and do the IMPDEFs, setting up, among other
;things, the BCK PCB. We then start NMFS running with the BCK PCB which
;will in turn start all the other processes so indicated by the config
;area.

;The priorities for the NMFS IMP are:
;
;  0     unused
;  1     M2I
;  2     I2M
;  3     I2H
;  4     H2I
;  5     TIM
;  6     SPF
;  7     TSK
; 14     DPY
; 15     BCK
; 9-31   unused

                .section pg4
                .lev bck
                .lck all

004000 140040 9 0 init:   cra                         ;turn off NMFS
004001 000030 9 0         nmfs                        ;by doing a nmfs(0)

004002 005560 9 0         lda [ heapend - heapbeg ] ;get size of heap area
004003 072070 9 0         ldx [ heapbeg ]           ;get ptr to first word to cle
004004 021161 9 0         jst i.zero                ;zero the HEAP

004005 005561 9 0         lda [ p0ze - p0zb ]       ;clear page 0 zero area
004006 073562 9 0         ldx [ p0zb ]
004007 021161 9 0         jst i.zero

004010 005563 9 0         lda [paraml]              ;zero parameter table
004011 073564 9 0         ldx [paramt]
004012 021161 9 0         jst i.zero

;now clear noc areas
004013 004055 9 0         lda [ nocpg0e - nocpg0b ]
004014 072060 9 0         ldx [ nocpg0b ]
004015 021161 9 0         jst i.zero
004016 005565 9 0         lda [ .vers. ]            ;and set vers at 105
004017 010105 9 0         sta vers

004020 005566 9 0         lda [ noc.zsize ]
004021 073567 9 0         ldx [ noc.neighb ]
004022 021161 9 0         jst i.zero

;next, do impdefs
```

```
004023 005570 9 0          lda [ (defbeg - defend) / 2 ] ;get minus size of def
004024 010001 9 0          sta ireg
004025 073571 9 0          ldx [ defbeg ]                ;get ptr to first

004026 044000 9 0 i.def:   lda 0 x                       ;get address
004027 101040 9 0          snz
004030 000000 9 0           %crash                      ;i.def: IMPDEF of address 0
004031 010002 9 0          sta jreg                      ;save it
004032 044001 9 0          lda 1 x                       ;get value
004033 110002 9 0          sta jreg i                    ;plant it
004034 024000 9 0          irs 0                         ;step to next entry
004035 024000 9 0          irs 0                         ;(each is 2 words)
004036 024001 9 0          irs ireg                      ;more?
004037 003026 9 0           jmp i.def                    ;yes - do them

;Now init all the NMFS queues. This is done here because doing it with
;the DEF mechanism (above) is too costly. Note that in the I.NMFQ loop,
;we don't bother to clear Q.LENG. Since the queue headers are in HEAP
;storage, they are already zero.

004040 005572 9 0          lda [ -( 1+1+1+32. ) ]        ;set IREG = number of NMFS qu
004041 010001 9 0          sta ireg                      ; (IDLE + READY + TIMING + 32

004042 072133 9 0          ldx idlqi                     ;get pointer to idle queue
004043 004000 9 0 i.nmfq:  lda 0                         ;get this queue's address
004044 050000 9 0          sta q.forw x                  ;point forward, back and hedr
004045 050001 9 0          sta q.back x
004046 050002 9 0          sta q.hedr x
004047 014054 9 0          add [ qhedr.size ]            ;then step X to next header
004050 010000 9 0          sta 0
004051 024001 9 0          irs ireg                      ;more to go?
004052 003043 9 0           jmp i.nmfq                   ;yes

;Now init the IMP queue end pointers in the heap.

004053 073573 9 0          ldx [ queueb - queuee ]
004054 032001 9 0          stx ireg
004055 073574 9 0          ldx [ queuee ]
004056 005575 9 0          lda [ queueb ]                ;start A with address of star

004057 050000 9 0 i.qhh:   sta 0 x                       ;set end pointer back to star
004060 141206 9 0          aoa
004061 024000 9 0          irs 0
004062 024001 9 0          irs ireg
004063 003057 9 0           jmp i.qhh

;Having made the NMFS queue headers and all the non-device PCBs (BCK, DPY,
; TIM, SPF, TSK) - ENQ the BCK PCB onto the IDLEQ and start NMFS there.

004064 004133 9 0          lda idlqi                     ;get ptr to NMFS idle queue
004065 072134 9 0          ldx bckpci                    ;get ptr to BCK's PCB
004066 000002 9 0          enq
004067 000030 9 0          nmfs                          ;using IDLQI still in A, star
004070 000000 9 0           %crash                      ;I.NMFQ: returned from NMFS(id

                           ;"fall" into BCKINI, next page
```

```
                        ;"from" previous page

 ;This is the first instruction to be executed under NMFS. It is actually
 ;the top of the BCK (background) process. At this point, appropriate HEAP
 ;memory is zero, and the DEFs have all been done. We now ENQ all the non-IO
 ;PCB left (DPY, TIM, SPF, TSK) and then start on the IO processes.

                    .lev bck

004071 001001 9   bckini: .inh all
004072 072135 9 0         ldx dpypci              ;DPY
004073 021415 9 0         jst enqapr

004074 072140 9 0         ldx timpci              ;TIM
004075 021415 9 0         jst enqapr

004076 072137 9 0         ldx spfpci              ;SPF
004077 021415 9 0         jst enqapr

004100 072136 9 0         ldx tskpci              ;TSK
004101 021415 9 0         jst enqapr

 ;Now make the 4 fake hosts I2H and H2I processes

004102 072056 9 0         ldx [ nh+0 ]
004103 021424 9 0         jst fakapr

004104 073576 9 0         ldx [ nh+1 ]
004105 021424 9 0         jst fakapr

004106 073577 9 0         ldx [ nh+2 ]
004107 021424 9 0         jst fakapr

004110 073600 9 0         ldx [ nh+3 ]
004111 021424 9 0         jst fakapr

 ;Now begin to process the config area.  First, copy imp number into MINE.

004112 104041 9 0         lda [config.num] i      ;get our imp number
004113 010106 9 0         sta mine                ;put it on page zero

004114 000012 9 0         %memhi                  ;get higest memory address av
004115 010226 9 0         sta pcbaddr

004116 140040 9 0         cra                     ;i.chns and i.hosts used to
                                                  ;calculate maxs and maxr
004117 011475 9 0         sta i.chns              ;total number of channels
004120 011474 9 0         sta i.hosts             ;number of hosts

004121 005601 9 0         lda [-ch]
004122 011473 9 0         sta i.temp              ;ireg counts modems

004123 005602 9 0         lda [configtrunk]       ;address of first trunkblock

004124 010230 9 0         sta cfblock

 ;throughout the following two loops (one for modems, one for hosts),
 ;cfblock contains the base address of the current trunk/host block and
```

;cfword contains the address of a specific word of the current trunk/host

```
;block (whichever word is being dealt with at the moment)

004125 010231 9 0 i.mpcb: sta cfword
004126 104231 9 0         lda cfword i 0&cf.trtype
004127 101040 9 0         snz                          ;does modem exist?
004130 003133 9 0          jmp i.mpc0                  ;no
004131 021172 9 0         jst i.min                    ;make modem in pcb
004132 021242 9 0         jst i.mout                   ;make modem out pcb
004133 004230 9 0 i.mpc0: lda cfblock                  ;move to next trunk block
004134 014055 9 0         add [cf.size]
004135 010230 9 0         sta cfblock
004136 025473 9 0         irs i.temp
004137 003125 9 0          jmp i.mpcb                  ;do next modem

004140 004032 9 0         lda [-nh]
004141 010001 9 0         sta ireg                     ;ireg counts hosts
004142 005603 9 0         lda [configihost]            ;first host config block
004143 010230 9 0         sta cfblock
004144 010231 9 0 i.hpcb: sta cfword
004145 104231 9 0         lda cfword i 0&cf.htype
004146 101040 9 0         snz                          ;does this host exist?
004147 003153 9 0          jmp i.hpc0                  ;no
004150 025474 9 0         irs i.hosts                  ;count another host
004151 021271 9 0         jst i.hin                    ;make host in pcb
004152 021331 9 0         jst i.hout                   ;make host out pcb
004153 004230 9 0 i.hpc0: lda cfblock                  ;move to next host block
004154 014055 9 0         add [cf.size]
004155 010230 9 0         sta cfblock
004156 024001 9 0         irs ireg
004157 003144 9 0          jmp i.hpcb                  ;do next host

004160 003450 9 0         jmp i.cioe                   ;go carve up buffers


;subroutine to zero an area
;enter with a=length, x=start address
004161 000000 9 0 i.zero: 0
004162 140407 9 0         tca
004163 010001 9 0         sta ireg
004164 140040 9 0         cra
004165 050000 9 0 i.zer1: sta 0 x
004166 024000 9 0         irs 0
004167 024001 9 0         irs ireg
004170 003165 9 0          jmp i.zer1
004171 103161 9 0         jmp i.zero i


;subroutine to init an M2I PCB
004172 000000 9 0 i.min:  0
004173 024231 9 0         irs cfword 0&cf.nchan        ;modem in pcb size equals
004174 104231 9 0         lda cfword i                 ;number of channels
004175 010001 9 0         sta ireg
004176 040475 9 0         lgr 3                        ;plus number of octets
004177 014001 9 0         add ireg
004200 015604 9 0         add [basicm2isize]           ;plus size of fixed part
004201 010227 9 0         sta pcblen                   ;size of this pcb
004202 021400 9 0         jst zeropcb
004203 004226 9 0         lda pcbaddr                  ;address of previous pcb
```

```
004204 016227 9 0          sub pcblen              ;  minus size of this pcb
004205 010226 9 0          sta pcbaddr             ;  equals size of this pcb
004206 072226 9 0          ldx pcbaddr             ;x register will contain the
                                                   ;pointer for the rest of i.mi
004207 104231 9 0          lda cfword i 0&cf.nchan
004210 050027 9 0          sta m2ipcb.nchan x      ;initialize channel field in
004211 015475 9 0          add i.chns              ;add to running total
004212 011475 9 0          sta i.chns

004213 024231 9 0          irs cfword
004214 104231 9 0          lda cfword i 0&cf.delay
004215 050030 9 0          sta m2ipcb.delay x      ;initialize delay field of pc

004216 024231 9 0          irs cfword              ;skip over next 2 words of tr
004217 024231 9 0          irs cfword
004220 024231 9 0          irs cfword

004221 104231 9 0          lda cfword i 0&cf.speed
004222 050026 9 0          sta m2ipcb.speed x      ;initialize speed field of pc

004223 005473 9 0          lda i.temp
004224 015605 9 0          add [ch]
004225 050025 9 0          sta pcb.num x           ;modem number

004226 005606 9 0          lda [m2ipri + state.idle] ;modem in's priority
004227 050003 9 0          sta pcb.pri x           ;and idle bit

004230 005607 9 0          lda [m2iini-1]          ;make modem in start at m2ii
004231 050006 9 0          sta pcb.pc x

004232 004000 9 0          lda 0                   ;put pcb pointer in directory
004233 073473 9 0          ldx i.temp
004234 151610 9 0          sta [m2ipct+ch] ix
004235 072226 9 0          ldx pcbaddr

004236 024231 9 0          irs cfword              ;move cfword to the in-type/r
004237 021360 9 0          jst typerb              ;set up type and rb fields

004240 021415 9 0          jst enqapr              ;enq pcb on idle queue
                                                   ;and create process
004241 103172 9 0          jmp i.min i             ;return

004242 000000 9 0 i.mout:  0
004243 005611 9 0          lda [i2mpcb.size]       ;size of a modem-out pcb
004244 010227 9 0          sta pcblen
004245 021400 9 0          jst zeropcb
004246 004226 9 0          lda pcbaddr             ;address of previous pcb
004247 016227 9 0          sub pcblen              ;  minus size of this pcb
004250 010226 9 0          sta pcbaddr             ;  equals address of this pcb
004251 072226 9 0          ldx pcbaddr             ;x reg will contain the pcb a
                                                   ;throughout i.mout
004252 005473 9 0          lda i.temp
004253 015605 9 0          add [ch]
004254 050025 9 0          sta pcb.num x           ;modem number

004255 005612 9 0          lda [i2mpri + state.idle] ;modem out priority
004256 050003 9 0          sta pcb.pri x           ;and idle bit

004257 005613 9 0          lda [i2mini-1]          ;make modem out start at i2mi
```

```
004260 050006 9 0          sta pcb.pc x

004261 004000 9 0          lda 0                    ;put pcb pointer in directory
004262 073473 9 0          ldx i.temp
004263 151614 9 0          sta [i2mpct+ch] ix
004264 072226 9 0          ldx pcbaddr

004265 024231 9 0          irs cfword               ;cfword is address of out-typ
004266 021360 9 0          jst typerb               ;set up type and rb fields

004267 021415 9 0          jst enqapr               ;enq pcb on idle queue
                                                    ;and create process
004270 103242 9 0          jmp i.mout i             ;return

004271 000000 9 0 i.hin:   0
004272 005615 9 0          lda [h2ipcb.size]        ;size of a host-in pcb
004273 010227 9 0          sta pcblen
004274 021400 9 0          jst zeropcb
004275 004226 9 0          lda pcbaddr              ;address of previous pcb
004276 016227 9 0          sub pcblen               ;  minus size of this pcb
004277 010226 9 0          sta pcbaddr              ;  equals address of this pcb
004300 072226 9 0          ldx pcbaddr              ;x register will contain the
                                                    ;address throughout i.hin
004301 024231 9 0          irs cfword
004302 104231 9 0          lda cfword i 0&cf.haccom
004303 050045 9 0          sta h2ipcb.haccom x

004304 024231 9 0          irs cfword
004305 104231 9 0          lda cfword i 0&cf.hacmem
004306 050044 9 0          sta h2ipcb.hacmem x

004307 024231 9 0          irs cfword               ;skip next 3 words of host bl
004310 024231 9 0          irs cfword
004311 024231 9 0          irs cfword

004312 004001 9 0          lda ireg
004313 014056 9 0          add [nh]
004314 050025 9 0          sta pcb.num x            ;host number

004315 005616 9 0          lda [ state.idle + h2ipri ] ;host-in priority
004316 050003 9 0          sta pcb.pri x

004317 004045 9 0          lda [h2iini-1]           ;make h2i start at h2iini
004320 050006 9 0          sta pcb.pc x

004321 004000 9 0          lda 0                    ;put pcb address in directory
004322 072001 9 0          ldx ireg
004323 151617 9 0          sta [h2ipct+nh] ix
004324 072226 9 0          ldx pcbaddr

004325 024231 9 0          irs cfword               ;cfword is the address of the
                                                    ;in-type/rb field in the pcb

004326 021360 9 0          jst typerb               ;set up type and rb fields of

004327 021415 9 0          jst enqapr               ;put pcb on the idle queue an
                                                    ;create the process
004330 103271 9 0          jmp i.hin i              ;return
```

```
004331 000000 9 0 i.hout: 0
```

```
004332 005620 9 0          lda [i2hpcb.size]
004333 010227 9 0          sta pcblen
004334 021400 9 0          jst zeropcb
004335 004226 9 0          lda pcbaddr
004336 016227 9 0          sub pcblen
004337 010226 9 0          sta pcbaddr
004340 072226 9 0          ldx pcbaddr              ;x reg contains pcb pointer
                                                    ;throughout i.hout

004341 004001 9 0          lda ireg
004342 014056 9 0          add [nh]
004343 050025 9 0          sta pcb.num x            ;host number

004344 005621 9 0          lda [ state.idle + i2hpri ] ;host-out priority
004345 050003 9 0          sta pcb.pri x

004346 005622 9 0          lda [i2hini-1]           ;make i2h start at i2hini
004347 050006 9 0          sta pcb.pc x

004350 004000 9 0          lda 0                    ;put pcb address in directory
004351 072001 9 0          ldx ireg
004352 151623 9 0          sta [i2hpct+nh] ix
004353 072226 9 0          ldx pcbaddr

004354 024231 9 0          irs cfword               ;cfword is the address of the
                                                    ;in-type/rb field in the pcb

004355 021360 9 0          jst typerb               ;set up type and rb fields of

004356 021415 9 0          jst enqapr               ;put pcb on the idle queue an
                                                    ;create the process
004357 103331 9 0          jmp i.hout i             ;return


004360 000000 9 0 typerb:  0
004361 104230 9 0          lda cfblock i 0&cf.typerb ;1 in first cfblock word me
004362 016052 9 0          sub [1]                  ;regular modem or host
004363 101040 9 0          snz
004364 003374 9 0           jmp regulr
004365 004006 9 0          lda zero                 ;vdh and hdh hosts get type 0
004366 050005 9 0          sta pcb.type x
004367 104231 9 0 rbpart:  lda cfword i 0&cf.typerb
004370 006023 9 0          ana [cf.rbmask]          ;get rb
004371 040472 9 0          lgr 6
004372 050021 9 0          sta pcb.rb x
004373 103360 9 0          jmp typerb i

004374 104231 9 0 regulr:  lda cfword i 0&cf.typerb
004375 006005 9 0          ana [cf.typemask]        ;get type
004376 050005 9 0          sta pcb.type x
004377 003367 9 0          jmp rbpart               ;get rb and return

004400 000000 9 0 zeropcb:  0
004401 004226 9 0          lda pcbaddr
004402 016227 9 0          sub pcblen
004403 010002 9 0          sta jreg
004404 004227 9 0          lda pcblen
004405 140407 9 0          tca                      ;negate to use as counter
004406 010000 9 0          sta 0
004407 140040 9 0          cra
```

```
004410 110002 9 0 zerot:  sta jreg i                ;zero word pointed to by jreg
```

```
004411 024002 9 0          irs jreg
004412 024000 9 0          irs 0
004413 003410 9 0           jmp zerot
004414 103400 9 0          jmp zeropcb i              ;return

;Enter with X=PCB pointer. ENQ's, APR's and GPR's the process.
004415 000000 9 0 enqapr:  0
004416 004133 9 0          lda idlqi                  ;get ptr to IDLE queue
004417 000002 9 0          enq
004420 101000 9 0           nop
004421 000003 9 0          apr                        ;create it
004422 000043 9 0          gpr                        ;and poke it
004423 103415 9 0          jmp enqapr i               ;then return
```

```
;Subroutine to create fake host #n's host processes. Enter with fake host
;internal number (NH, NH+1, .. NH+FH-1) in X.

004424 000000 9 0 fakapr: 0
004425 032001 9 0         stx ireg                    ;save host number
004426 145614 9 0         lda [i2hpct] ix             ;get pointer to I2H PCB
004427 026000 9 0         ima 0                       ;put into X, get host number
004430 050025 9 0         sta pcb.num x               ;set up the number
004431 005622 9 0         lda [i2hini-1]
004432 050006 9 0         sta pcb.pc x
004433 005621 9 0         lda [ state.idle + i2hpri ]
004434 050003 9 0         sta pcb.pri x               ;leave type=0 (software)
004435 021415 9 0         jst enqapr                  ;make it happen
004436 072001 9 0         ldx ireg                    ;get host number back
004437 145624 9 0         lda [h2ipct] ix             ;get pointer to H2I PCB
004440 026000 9 0         ima 0                       ;put into X, get host number
004441 050025 9 0         sta pcb.num x               ;set up the number
004442 004045 9 0         lda [h2iini-1]
004443 050006 9 0         sta pcb.pc x
004444 005616 9 0         lda [ state.idle + h2ipri ]
004445 050003 9 0         sta pcb.pri x               ;leave type=0 (software)
004446 021415 9 0         jst enqapr                  ;make it happen
004447 103424 9 0         jmp fakapr i                ;return to caller
```

```
;Here when all the processes have been created. Get the highest
;memory address available (where the pcb's begin) and set LAST pointing to
;it. Then carve up memory from BUF0 as far as possible, skipping those
;places with stolen buffers. This loop is similar to the one in pre-4300
;IMPs, but should probably be re-written.

004450 140040 9 0 i.cioe: cra
004451 011476 9 0         sta i.bufs                ;count buffers

004452 004040 9 0         lda [ buf0 ]
004453 011473 9 0 i.crv:  sta i.temp                ;i.temp is "proposed buffer a
004454 004226 9 0         lda pcbaddr
004455 017473 9 0         sub i.temp
004456 101400 9 0         smi
004457 023625 9 0          cas [bufl]               ;room for this buffer?
004460 101000 9 0           nop                    ;yes, plenty
004461 100000 9 0           skp                    ;yes, just enough
004462 003477 9 0            jmp i.bfrs            ;no, stop when 0<=rem<bufl
004463 005473 9 0         lda i.temp
004464 121626 9 0         jst [bchsch] i            ;search bench: is buffer stol
004465 003453 9 0          jmp i.crv                ;yes, resume past stolen rang
004466 121627 9 0         jst [buffre] i            ;no, add buffer to free queue
004467 025476 9 0         irs i.bufs                ;count buffers
004470 005473 9 0         lda i.temp                ;compute next address
004471 015625 9 0         add [bufl]
004472 003453 9 0         jmp i.crv

004473       9 0 i.temp: .block 1
004474       9 0 i.hosts:.block 1                   ;counts hosts
004475       9 0 i.chns: .block 1                   ;total number of channels
004476       9 0 i.bufs: .block 1                   ;number of buffers

;set maxr and maxs.  each host gets a minimum of 10 buffers.  if it
;is possible to give store and forward one buffer per channel (i.chns)
;and still have at least 10 buffers per host left over for reassembly,
;set maxs to i.chns and set maxr to the remainder (i.bufs-i.chns).
;otherwise, give reassembly the minimum (10*number of hosts) and
;give store and forward the rest.
004477 005474 9 0 i.bfrs: lda i.hosts               ;count fake hosts too
004500 014054 9 0         add [4]                   ;now multipy by 10
004501 041477 9 0         lgl 1                     ;i.hosts*2
004502 010001 9 0         sta ireg                  ;save it
004503 041476 9 0         lgl 2                     ;i.hosts*8
004504 014001 9 0         add ireg                  ;plus i.hosts*2
004505 010001 9 0         sta ireg                  ;10*i.hosts
004506 005476 9 0         lda i.bufs                ;number of buffers
004507 017475 9 0         sub i.chns                ;are there enough to give i.c
004510 022001 9 0         cas ireg                  ;..to sf and still have 10h f
004511 003521 9 0          jmp i.maxsf              ;yes, give sf max
004512 003521 9 0          jmp i.maxsf
004513 004001 9 0         lda ireg                  ;no, give reassembly the mini
004514 010157 9 0         sta maxr                  ;10*i.hosts
004515 005476 9 0         lda i.bufs                ;and give sf the remainder
004516 016157 9 0         sub maxr
004517 010156 9 0         sta maxs
004520 003524 9 0         jmp i.add                 ;do add chain

004521 010157 9 0 i.maxsf: sta maxr                 ;maxr=i.bufs-i.chns
004522 005475 9 0         lda i.chns                ;give store-and-forward the m
```

```
004523 010156 9 0          sta maxs                    ;and fall into add chain

 ;Now init the ADD CHAIN. This can be removed when all the checksum functions
;have been converted to use the NMFS CHK instruction.
 004524 073630 9 0 i.add:  ldx [-maxpl]                ;init add chain
 004525 005631 9 0         lda [add maxpl+hedr-1 x]
004526 151632 9 0 i.addl: sta [addtop+maxpl] ix
 004527 016052 9 0         sub one
 004530 024000 9 0         irs 0
004531 003526 9 0          jmp i.addl


; now, call each package (or the default hook)

 004532 004032 9 0         lda [-npaks]
004533 011473 9 0          sta i.temp                  ; i.temp is loop ctr
 004534 005473 9 0 i.paks: lda i.temp
 004535 015633 9 0         add [ini00.hook+npaks]
004536 010001 9 0          sta ireg                    ; compute address of next hoo
 004537 104001 9 0         lda ireg i                  ; use it to fetch next hook
 004540 010001 9 0         sta ireg
004541 120001 9 0          jst ireg i                  ; now "jst" through that hook
 004542 025473 9 0         irs i.temp
 004543 003534 9 0          jmp i.paks

 ;Having done all that, wait a while ??, fire off a trouble report, enable
 ;and begin the IMP.

 004544 000010 9 0 i.wait: %rdclok
 004545 101400 9 0         smi
004546 003544 9 0          jmp .-2
 004547 000010 9 0         %rdclok
 004550 100400 9 0         spl
004551 003547 9 0          jmp .-2
 004552 000010 9 0         %rdclok
 004553 101400 9 0         smi
004554 003552 9 0          jmp .-2

 004555 121634 9 0         jst [swch] i                ;fire off trouble report
004556 000401 9 0          .enb bck                    ;start IMP!
 004557 103635 9           jmp [backst] i              ;init becomes background
```

```
                .section pg6
                .lev bck
                .lck all

;The following area of memory is called the DEF area (previously known
;as IMPDEF). It consists of pairs of values: the first is an address
;and the second is a value. The value is planted where the address points.

                defbeg:


;First, set up the NMFS processes BCK, TIM, SPF and TSK:

006000 045463 9 0       .word timpcb+pcb.pc, timini-1
006001 024102 9 0
006002 045460 9 0       .word timpcb+pcb.pri, 5. + state.idle
006003 100005 9 0

006004 045442 9 0       .word spfpcb+pcb.pc, spfini-1
006005 034056 9 0
006006 045437 9 0       .word spfpcb+pcb.pri, 6. + state.idle
006007 100006 9 0

006010 045421 9 0       .word tskpcb+pcb.pc, tskini-1
006011 015777 9 0
006012 045416 9 0       .word tskpcb+pcb.pri, 7. + state.idle
006013 100007 9 0

006014 045400 9 0       .word dpypcb+pcb.pc, dpyini-1
006015 023345 9 0
006016 045375 9 0       .word dpypcb+pcb.pri, 14. + state.idle
006017 100016 9 0

006020 045357 9 0       .word bckpcb+pcb.pc, bckini ;no need for -1 because i
006021 004071 9 0
006022 045354 9 0       .word bckpcb+pcb.pri, 15. + state.idle
006023 100017 9 0

;Now set up the fake and back host addresses in JAM, SUCK and SLEEP as
;well as the PCB directory entries for the fake hosts

006024 014266 9 0       .word dztb+0, nojam     ;TTY
006025 014114 9 0
006026 014267 9 0       .word dztb+1, ddjini    ;DDT
006027 025001 9 0
006030 014270 9 0       .word dztb+2, trjini    ;TRACE
006031 027177 9 0
006032 014271 9 0       .word dztb+3, stjini    ;STAT
006033 026000 9 0

006034 014272 9 0       .word wttb+0, nosuck    ;TTY
006035 014122 9 0
006036 014273 9 0       .word wttb+1, ddsini    ;DDT
006037 025251 9 0
006040 014274 9 0       .word wttb+2, prsini    ;PARAMETER CHANGE
006041 030000 9 0
006042 014275 9 0       .word wttb+3, dssini    ;DISCARD
006043 027144 9 0
```

```
006044 045542 9 0       .word 0*h2ipcb.size+fhipcb+h2ipcb.hacmem, 177774 ;TTY
006045 177774 9 0
006046 045612 9 0       .word 1*h2ipcb.size+fhipcb+h2ipcb.hacmem, 100000 ;DDT
006047 100000 9 0
006050 045662 9 0       .word 2*h2ipcb.size+fhipcb+h2ipcb.hacmem, 000002 ;PKC
006051 000002 9 0
006052 045732 9 0       .word 3*h2ipcb.size+fhipcb+h2ipcb.hacmem, 177777 ;MGM
006053 177777 9 0

006054 046272 9 0       .word h2ipct+nh+0, 0*h2ipcb.size+fhipcb
006055 045476 9 0
006056 046273 9 0       .word h2ipct+nh+1, 1*h2ipcb.size+fhipcb
006057 045546 9 0
006060 046274 9 0       .word h2ipct+nh+2, 2*h2ipcb.size+fhipcb
006061 045616 9 0
006062 046275 9 0       .word h2ipct+nh+3, 3*h2ipcb.size+fhipcb
006063 045666 9 0

006064 046246 9 0       .word i2hpct+nh+0, 0*i2hpcb.size+fhopcb
006065 045736 9 0
006066 046247 9 0       .word i2hpct+nh+1, 1*i2hpcb.size+fhopcb
006067 046005 9 0
006070 046250 9 0       .word i2hpct+nh+2, 2*i2hpcb.size+fhopcb
006071 046054 9 0
006072 046251 9 0       .word i2hpct+nh+3, 3*i2hpcb.size+fhopcb
006073 046123 9 0

006074 014276 9 0       .word sltb+0, bk0ini    ;RFNMS, allocates, etc.
006075 014354 9 0
006076 014277 9 0       .word sltb+1, bk1ini    ;Tx block monitor
006077 015030 9 0
006100 014300 9 0       .word sltb+2, bk2ini    ;Rx block monitor
006101 015030 9 0
006102 014301 9 0       .word sltb+3, bk5ini    ;Give backs, etc. ?? why is t
006103 015340 9 0
006104 014302 9 0       .word sltb+4, bk4ini    ;Re-route
006105 015162 9 0

;These words get 0

006106 005660 9 0       .word dmndcp, 0
006107 000000 9 0
006110 030074 9 0       .word pkcbff, 0
006111 000000 9 0
006112 027526 9 0       .word pkcast, 0
006113 000000 9 0
006114 015107 9 0       .word b1cutc, 0
006115 000000 9 0
006116 015110 9 0       .word b2cutc, 0
006117 000000 9 0
006120 015622 9 0       .word back5i, 0
006121 000000 9 0
006122 014641 9 0       .word b0try,  0
006123 000000 9 0
006124 024605 9 0       .word mbtfre, 0
006125 000000 9 0
006126 014306 9 0       .word modloc, 0
006127 000000 9 0
006130 031436 9 0       .word bufrq1, 0
```

```
006131 000000 9 0
006132 031437 9 0          .word bufrq2, 0
006133 000000 9 0
006134 031440 9 0          .word bufbsy, 0
006135 000000 9 0
006136 031433 9 0          .word bufflu, 0
006137 000000 9 0
006140 047017 9 0          .word trcnts, 0
006141 000000 9 0
006142 033225 9 0          .word rupsnd, 0
006143 000000 9 0
006144 033144 9 0          .word iniflg, 0
006145 000000 9 0

;These words get -1

006146 005657 9 0          .word dmndcd, -1
006147 177777 9 0
006150 014634 9 0          .word b0pass, -1
006151 177777 9 0
006152 000314 9 0          .word smoclk, -1
006153 177777 9 0
006154 014000 9 0          .word sumtim, -1
006155 177777 9 0
006156 026521 9 0          .word thrupc, -1
006157 177777 9 0

;Now some other assorted values

006160 040163 9 0          .word b5hisp, b5typh-typh ;for back host 5 (sigh)
006161 015601 9 0
006162 046615 9 0          .word addbot,(jmp chksum i) ;the IMP's add chain retu
006163 102002 9 0
006164 006651 9 0          .word ovrhed,5          ;overhead toggle 5/6 = 5.5
006165 000005 9 0
006166 021103 9 0          .word blkslc,100000     ;software lock for BLKSRC in
006167 100000 9 0
006170 023254 9 0          .word cxndx,cxtab       ;backgound checksummer index

006171 023255 9 0

       006172      defend = .

                   .lev var
006172 000000 V    jstini: 0                       ; dummy init routine
006173 103172 V            jmp jstini i
```

```
                .stitl MODEM TO IMP (M2I)
                .INCLUDE m2i.m4
;NMFS notes:
```

```
;This is the format for the M2I PCB application words:

000026      m2ipcb.speed = pcb.ff              ;speed (00=50kb, 01=230, 10=1
000027      m2ipcb.nchan = m2ipcb.speed + 1 ;number of channels (8,16,24
000030      m2ipcb.delay = m2ipcb.nchan + 1 ;prop delay in 800us units
000031      m2ipcb.getqh = m2ipcb.delay + 1 ;get queue header
000035      m2ipcb.putqh = m2ipcb.getqh + qhedr.size ;put queue header
000041      m2ipcb.iocb1 = m2ipcb.putqh + qhedr.size ;IOCB #1
000047      m2ipcb.iocb2 = m2ipcb.iocb1 + biocb.size ;IOCB #2
000055      tsex = m2ipcb.iocb2 + biocb.size ;odd-even bit to use for nex
000065      rsex    = tsex+(octmax+1)/2        ;last odd-even bit received
000075      chfree  = rsex+(octmax+1)/2        ;bit=1 => channel is free
000105      chcounter = chfree+(octmax+1)/2 ;number of channels in use
000106      maxoctet  = chcounter+1            ;min(neighbor's cfoctet,our c
000107      cfoctet = maxoctet+1               ;max octet number we're confi
000110      snull   = cfoctet+1                ;which octets have acks to se
000111      octtab  = snull+1                  ;pointer to first word of oct
000112      octend  = octtab+1                 ;pointer to the first word af
                                               ;octet table
000113      sendoct = octend+1                 ;pointer to next octet to sen
000114      putoct  = sendoct+1                ;pointer to next place to put
000115      slt     = putoct+1                 ;non-zero to do demand core
000116      neighb  = slt+1                    ;number of neighbor
000117      line    = neighb+1                 ;0=up,-=silent,1=dead,2=comin
000120      lnei    = line+1                   ;last neighbor seen
000121      thrupw  = lnei+1                   ;word throughput(modem in)
000122      thrupt  = thrupw+1                 ;packet throughput(modem in)

000123      i2mtab  = thrupt+1                 ;pointer to first channel slo
000124      i2mend  = i2mtab+1                 ;pointer to word after last c
000125      retrqi  = i2mend+1                 ;pointer to retransmission qu
000126      retrqh  = retrqi+1                 ;retransmission queue header

000132      i2mwtn  = retrqh+qhedr.size        ;counter of words sent out (m
000133      i2mwt2  = i2mwtn+1                 ;double precision
000134      i2mpkc  = i2mwt2+1                 ;packet core output completio
000135      i2mrup  = i2mpkc+1                 ;saved routing update pointer
000136      i2mrtt  = i2mrup+1                 ;nominal retransmit time
000137      i2mbsy  = i2mrtt+1                 ;non-zero means i2m is busy
000140      i2mrar  = i2mbsy+1                 ;rare event flag for i2m
000141      i2mrem  = i2mrar+1                 ;reset modem flag for i2m
000142      i2mnxt  = i2mrem+1                 ;address of packet currently
000143      i2mack  = i2mnxt+1                 ;non-zero if packet on the li
000144      nulptr  = i2mack+1                 ;pointer to null area
000145      nullarea = nulptr+1                ;null area (for nulls and z p
000152      smq     = nullarea+seqh-hedr+1     ;modem queue front pointer
000153      smpq    = smq+1                    ;modem priority queue front p
000154      emq     = smpq+1                   ;modem queue end pointer
000155      empq    = emq+1                    ;modem priority queue end poi
000156      zrcv    = empq+1                   ;copied TYPH word of latest h
000157      zoct    = zrcv+1                   ;copied NETH word of latest h
000160      zsend   = zoct+1                   ;non-zero tells i2m to send h
000161      zncnt   = zsend+1                  ;n consecutive counter
000162      zkcnt   = zncnt+1                  ;k consecutive counter
000163      zkofn   = zkcnt+1                  ;k-of-n counters (7 words)
000172      zk      = zkofn+7                  ;value of k
000173      zn      = zk+1                     ;value of n
000174      zzcmup  = zn+1                     ;ticks to come up
000175      zmastr  = zzcmup+1                 ;high bit 1 if master, else 0
```

```
000176      lnclkp  = zmastr+1              ;pointer to line protocol clo
000177      rtrcvd  = lnclkp+1              ;number of ihy's received (hc
```

```
       000200     rtssnt  = rtrcvd+1              ;number of hellos sent (ticks
       000201     rup4to  = rtssnt+1              ;flag tells t.o to poke i2m
       000202     retflg  = rup4to+1              ;retransmission flag, set by
       000203     pkcnbr  = retflg+1              ;dead neighbor table for pack
       000204     sndrup  = pkcnbr+1              ;tells i2m to send routing up
       000205     rtptrs  = sndrup+1              ;pointer to first word of ret
       000206     rtpend  = rtptrs+1              ;pointer to word after last w
       000207     rtclkp  = rtpend+1              ;pointer to retransmission cl
       000210     rtrcnt  = rtclkp+1              ;retransmission counter
       000211     delsum  = rtrcnt+1              ;sum of delays
       000212     delsmo  = delsum+1              ;overflows from delsum
       000213     delcnt  = delsmo+1              ;count of transmitted packets
       000214     delcur  = delcnt+1              ;current average delay
       000215     delbas  = delcur+1              ;base average delay(last repo
       000216     rettimrs = delbas+1             ;first retransmission timer w
       000237     slots = rettimrs+(nnodes/8.+1)  ;first channel slot - must fo
                                                  ;retransmission timers (see m

       000002     mqueuel = emq-smq

 ;This is Modem-to-Imp (M2I). It has two halves: pre-dispatch, and post-
 ;dispatch. The first half is concerned with very low-level functions:
 ;doing physical input, timing out stale I/O and managing the "dummy
 ;input area" (below). The post-dispatch section (so named because M2I
 ;has deciphered the frame type at that point) runs lower-level protocol
 ;functions like ACK processing, demand relode packets, etc.

 ;The "dummy input area" is used in place of a buffer for catching
 ;input that is to be discarded. When, for example, no input
 ;has occured for a long time (10 seconds), we flush the current input
 ;and force the dummy buffer to be used for the next one. If the
 ;line has been physically disconnected from the modem, etc., then
 ;input may not occur for a very long time -- we wouldn't want to
 ;tie up a buffer for the duration.

 ;The following is an simplification of M2I's handling of an input
 ;completion interrupt:
 ;
 ;   INTERRUPT
 ;       0. previous input have an error? yes - goto step 4
 ;       1. previous input was into dummy area? yes - goto step 5
 ;       2. is LINE state = silent? yes - goto step 4
 ;       3. perform dispatch and post-dispatch processing, return.
 ;       4. if last input was into buffer, flush buffer
 ;       5. put up a new buffer if LINE<>silent, dummy if LINE=silent, return.

 ;If a NMFS timeout interrupt occurs (meaning the last input was stale) we:
 ;
 ;   TIMEOUT
 ;       0. flush previous transfer
 ;       1. put up dummy area, return

                    .section pg5
                    .lev m2i
 ;This is M2I initialization. Note that we tend to set up many parameters which

 ;are trunk-related because this is the highest priority modem process.

 005000 021233 1    m2iini: jst m2inmf              ;do NMFS-related initializat
 005001 000020 1            pcb
```

```
005002 032201 1          stx mbl                    ;save modem block pointer
```

```
005003 044025 1            lda pcb.num x          ;get our modem number
005004 010232 1            sta mp                 ;save it, too
005005 004000 1            lda 0                  ;initialize i2mtab to point
005006 015663 1            add [rettimrs]         ;first retr counter word
005007 050205 1            sta rtptrs x
005010 015664 1            add [nnodes/8.+1]      ;channel slots follow the
005011 050206 1            sta rtpend x           ;retransmission timers
005012 050123 1            sta i2mtab x           ;first channel slot
005013 044027 1            lda m2ipcb.nchan x     ;calculate highest octet numb
005014 040475 1            lgr 3
005015 016052 1            sub [1]
005016 050107 1            sta cfoctet x          ;highest octet number accordi
005017 050106 1            sta maxoctet x         ;must be set up for octini
005020 021275 1            jst octini             ;set up i2mtab and octtab for
005021 004000 1            lda 0                  ;retrqi is address of queue h
005022 015665 1            add [retrqh]
005023 050125 1            sta retrqi x
005024 021261 1            jst m2inmq             ;init queue header

005025 121666 1   m2iind:  jst [setnk] i          ;set up line up/down paramete
005026 120123 1            jst killi i            ;and declare them down
005027 000020 1   m2iioc:  pcb                    ;set up MIOCB for M2INEW
005030 044024 1            lda pcb.put x
005031 000022 1            deq
005032 003036 1             jmp m2idb             ;when we've done them all, st
005033 032234 1            stx miocb
005034 021156 1            jst m2inew             ;put up something for input
005035 003027 1            jmp m2iioc             ;do all the IOCBs
```

```
;Come to M2IDB to debreak until poked or output completes.

005036 000020 1   m2idb:  pcb                     ;get pointer to our PCB
005037 005667 1           lda [ %30sec / 3 ]     ;get 10 seconds
005040 000203 1           tpr                    ;set a timeout
005041 000103 1           spr                    ;debreak
005042 003130 1            jmp m2ito             ;on NMFS timeouts, flush and

;All modem interrupt entrances converge here. First, we set up MP=modem number

;and try to acquire the presumably just-filled input IOCB setting up M2ISP to
;point to the buffer hanging from it.

005043 000020 1   m2iint: pcb                     ;set up MP = M2IPCB.NUM
005044 032201 1           stx mbl                ;save modem block pointer
005045 044025 1           lda pcb.num x
005046 010232 1           sta mp
005047 044024 1           lda pcb.put x          ;try to get at the IOCB which
005050 000022 1           deq
005051 003036 1            jmp m2idb             ;see commentary in I2M about
005052 032234 1           stx miocb              ;save ptr to IOCB itself for

005053 044005 1   m2i0:   lda iocb.flags x       ;step 0 -- check for errors
005054 006072 1           ana [%icflags.cc]      ;mask off the completion code
005055 100040 1           sze                    ;non-zero?
005056 003120 1            jmp m2i6              ;if got an error, flush and r
005057 044004 1   m2i1:   lda iocb.addr x        ;get transfer address
005060 017670 1           sub [m2idmy]           ;to the dummy area?
005061 101040 1           snz
005062 003126 1            jmp m2i5              ;could be M2I4 but we know th
005063 017671 1           sub [ neth - m2idmy ]  ;fix SUB M2IDMY and subtract
005064 010233 1           sta m2isp              ;save the buffer address for
005065 044003 1           lda iocb.size x        ;get size
005066 040474 1           lgr 4.                 ;in words, not bits, please
005067 141240 1           icr 0&pktsiz
005070 141206 1           aoa 0&usecnt&trcpkt&trcrrt ;use count of one
005071 072233 1           ldx m2isp
005072 050004 1           sta wrdc x
005073 072201 1           ldx mbl                ;fix index
005074 044117 1   m2i2:   lda line x             ;get LINE state
005075 100400 1           spl                    ;silent state?
005076 003125 1            jmp m2i4              ;flush this buffer, will even

;Knowing that the previous input completed okay AND having M2ISP pointing
;to it, we are now ready to put up a new input.

005077 021156 1   m2i3:   jst m2inew              ;put up new input

;We are now ready to further process the just-completed input. The IOCB has
;been re-used (with a new buffer) for input. The old buffer, however, is
;pointed to by M2ISP (as it will continue to be throughout M2I).

005100 072233 1           ldx m2isp
005101 000010 1           %rdclok
005102 050121 1           sta itst x             ;save input time - 100 us cl
005103 020002 1           jst chksum             ;compute checksum
005104 003374 1            jmp pktch3            ;bad length
005105 100040 1           sze                    ;is the checksum good?
005106 003372 1            jmp pktch1            ;no, report as an error
```

005107 004232 1 lda mp 0&inpchn&hstmod

```
005110 050005 1           sta inch x              ;save input modem no
005111 044007 1           lda typh x              ;get pkt type + compat bit
005112 041675 1           alr 3
005113 006013 1           ana seven 0&pactyp&compat
005114 010000 1           sta 0
005115 045310 1           lda m2idsp x            ;pick out dispatch address
005116 010001 1           sta ireg
005117 102001 1           jmp ireg i              ;dispatch through it

;This is step #4 of M2I -- flush old (whatever it is).

005120 000020 1   m2i6:   pcb                     ;come here if input error
005121 044025 1           lda pcb.num x
005122 010000 1           sta 0
005123 165370 1           irs stats.lecs ix
005124 101000 1            nop
005125 021144 1   m2i4:   jst m2ifls               ;flush old input
005126 021156 1   m2i5:   jst m2inew              ;put up new input
005127 003043 1           jmp m2iint              ;try for more

;Here on timeout. Flush old and put up dummy.

005130 000020 1   m2ito:  pcb                     ;get ptr to PCB
005131 044025 1           lda pcb.num x
005132 010232 1           sta mp
005133 004052 1           lda [ %cm1.reset ]      ;reset the input
005134 000403 1           xdv                     ;poof
005135 044024 1           lda pcb.put x           ;get ptr to put queue
005136 000022 1           deq                     ;get punted IOCB off
005137 000000 1            %crash                 ;XDV of CM1.RESET didnt get ba
005140 032234 1           stx miocb
005141 021144 1           jst m2ifls              ;flush old
005142 021207 1           jst m2ipud              ;put up dummy
005143 003043 1           jmp m2iint              ;and do anything else

;Subr to flush old input, whatever it was.

005144 000000 1   m2ifls: 0
005145 072234 1           ldx miocb               ;where is the input for?
005146 044004 1           lda iocb.addr x
005147 017670 1           sub [m2idmy]            ;dummy area?
005150 101040 1           snz
005151 103144 1            jmp m2ifls i           ;if to dummy, skip flush
005152 017671 1           sub [ neth - m2idmy ]   ;compute address
005153 010000 1           sta 0
005154 120115 1           jst flushi i            ;flush the buffer
005155 103144 1           jmp m2ifls i

;Subroutine to put up new input depending on LINE.

005156 000000 1   m2inew: 0
005157 072201 1           ldx mbl
005160 044117 1           lda line x              ;check LINE state
005161 072234 1           ldx miocb               ;get IOCB pointer for later
005162 100400 1           spl
005163 003205 1            jmp m2ind              ;if LINE=silent, put up dummy
005164 121672 1           jst [gfree] i           ;get a buffer without checkin
005165 003201 1            jmp m2int              ;no buffers, trap and put up
005166 010000 1           sta 0                   ;make X point to buffer
```

```
005167 004052 1            lda [1]
005170 050004 1            sta wrdc x                ;set up use count of 1
005171 004000 1            lda 0                     ;get buffer ptr back
005172 072234 1            ldx miocb                 ;and IOCB pointer into X
005173 014031 1            add [neth]                ;offset into first REAL word
005174 050004 1            sta iocb.addr x           ;set up size
005175 005673 1            lda [ 16.*(bufend-neth) ] ;get size in bits
005176 050003 1            sta iocb.size x
005177 021217 1            jst m2ienq                ;enq it for input
005200 103156 1            jmp m2inew i              ;return

005201 072232 1   m2int:   ldx mp
005202 165371 1            irs stats.bufti ix        ;**stats**
005203 101000 1             nop
                                  ;defhlt(51. no buffers for input)
005204 120120 1            jst hltjst i 0&m2i        ;since we're highet priority
005205 021207 1   m2ind:   jst m2ipud                ;put up dummy
005206 103156 1            jmp m2inew i              ;and return

;Subroutine to put up dummy area

005207 000000 1   m2ipud:  0
005210 072234 1            ldx miocb
005211 005670 1            lda [m2idmy]              ;get dummy pointer
005212 050004 1            sta iocb.addr x           ;set up address
005213 004056 1            lda [ 16. ]               ;and size
005214 050003 1            sta iocb.size x
005215 021217 1            jst m2ienq                ;enq it
005216 103207 1            jmp m2ipud i              ;return

;Subroutine to ENQ the IOCB (pointed to by MIOCB) onto the GET queue of the
;current PCB. Destroys A,X.

005217 000000 1   m2ienq:  0
005220 000020 1            pcb                       ;get PTR to GET queue
005221 044023 1            lda pcb.get x
005222 072234 1            ldx miocb                 ;get back IOCB addr
005223 066005 1            ima iocb.flags x          ;save PCB.GET, set %IOFLAGS.A
005224 140500 1            ssm 0&%ioflags.always
005225 066005 1            ima iocb.flags x          ;restore AC to PCB.GET, plant
005226 000002 1            enq                       ;place it on the GET queue
005227 101000 1             nop
005230 000020 1            pcb
005231 001003 1            pdv                       ;poke the device
005232 103217 1            jmp m2ienq i              ;and return to caller
```

```
;Subrcutine to do NMFS-related M2I initialization. This is a subroutine only
;for clarity (it has only one caller, near M2IINI). We set up the get and
;put queue pointers and headers, then create the IOCB (and put it on the
;PUT queue).

005233 000000 1   m2inmf: 0
005234 000020 1           pcb                          ;get PCB pointer
005235 004000 1           lda 0                        ;get base
005236 015674 1           add [ m2ipcb.getqh ]         ;get ptr to queue header
005237 050023 1           sta pcb.get x
005240 021261 1           jst m2inmq                   ;initialize the queue header,
005241 014054 1           add [ m2ipcb.putqh - m2ipcb.getqh ] ;to PUT queue, no
005242 050024 1           sta pcb.put x
005243 021261 1           jst m2inmq                   ;initialize the queue header,
005244 014054 1           add [ m2ipcb.iocb1 - m2ipcb.putqh ] ;to IOCB
005245 026000 1           ima 0                        ;put IOCB ptr in X, PCB in A

005246 015675 1           add [ m2ipcb.putqh ]         ;compute address of put queue
005247 000002 1           enq                          ;equivelant to LDA PCB.PUT X
005250 101000 1            nop
005251 000020 1           pcb                          ;get PCB ptr back
005252 004000 1           lda 0                        ;get PCB ptr into A
005253 015676 1           add [ m2ipcb.iocb2 ]         ;compute ptr to IOCB #2
005254 010000 1           sta 0                        ;in X
005255 015677 1           add [ m2ipcb.putqh - m2ipcb.iocb2 ] ;compute ptr to P
005256 000002 1           enq
005257 101000 1            nop                         ;?? can't crash here because
005260 103233 1           jmp m2inmf i                 ;return

;Subrcutine (called from M2INMF and M2IINI) to init a queue header pointed
;to by A.Preserves A, X.

005261 000000 1   m2inmq: 0
005262 032001 1           stx ireg                     ;save caller's Index
005263 010002 1           sta jreg                     ;  ... and AC
005264 010000 1           sta 0                        ;put address of queue in X
005265 050000 1           sta q.forw x                 ;point it at self
005266 050001 1           sta q.back x
005267 050002 1           sta q.hedr x
005270 140040 1           cra                          ;length = 0, too
005271 050003 1           sta q.size x
005272 072001 1           ldx ireg                     ;fix index
005273 004002 1           lda jreg                     ;and AC
005274 103261 1           jmp m2inmq i                 ;return

;octini is called by m2i init, and by line up/down when the first
;line up/down packet comes in on a line (so we know how many
;octets to set up for).  m2ipcb.nchan and maxoctet must be set up
;before entering this routine.
005275 000000 1   octini: 0
005276 044123 1           lda i2mtab x
005277 054027 1           add m2ipcb.nchan x
005300 050124 1           sta i2mend x                 ;points one word beyond last
005301 050111 1           sta octtab x                 ;octtab follows channel slots
005302 050113 1           sta sendoct x                ;sendoct=putoct when table em
005303 050114 1           sta putoct x
005304 054106 1           add maxoctet x
005305 141206 1           aoa
005306 050112 1           sta octend x                 ;points one word beyond last
```

```
005307 103275 1          jmp octini i          ;..word of octet table
```

```
;This dispatch table is used by code above

005310 005334 1   m2idsp: m2ipkt 0&dattyp&datcpt  ;data
005311 005433 1           m2ifr1 0&dattyp
005312 005334 1           m2ipkt 0&nettyp&netcpt  ;network control
005313 005433 1           m2ifr1 0&nettyp
005314 005320 1           m2iswt 0&swttyp&swtcpt  ;switch control
005315 005320 1           m2iswt 0&swttyp         ;experimental protocols
005316 005440 1           m2ispc 0&sptcpt&sptcpt  ;special
005317 005433 1           m2ifr1 0&sptcpt

;Here to further dispatch on type 2 packets (switch control).

005320 004034 1   m2iswt: lda [swtsub]            ;use full subtype mask
005321 072233 1           ldx m2isp
005322 046007 1           ana typh x              ;get switch type subtype
005323 141140 1           icl 0&swtsub
005324 010000 1           sta 0
005325 045330 1           lda m2isds x            ;pick up subtype dispatch add
005326 010001 1           sta ireg
005327 102001 1           jmp ireg i

005330 005433 1   m2isds: m2ifr1                  ;subtype 0 (unused)
005331 005412 1           m2inul                  ;subtype 1 null
005332 022312 1           m2izpk                  ;subtype 2 hello/ihy
005333 033000 1           m2irup                  ;subtype 3 spf routing update
```

```
;Here from the dispatch mechanims (above) on DATA and NETWORK CONTROL
;packets. If the line is down, be sure to fire off a TRAP. If the packet
;is from ourselves (looped line?), throw the packet away. If all looks
;well, process the ACKs from the packet and pass it to TASK.

005334 072201 1   m2ipkt: ldx mbl                    ;get low-end bit
005335 044117 1          lda line x
005336 101040 1          snz                        ;is this line down?
005337 003343 1           jmp m2ipk1                ;no
                         ;defhlt(/9. accepting packet on dead line)
005340 072232 1          ldx mp
005341 120120 1          jst hltjst i               ;trap to ncc
005342 072201 1          ldx mbl
005343 044175 1   m2ipk1: lda zmastr x
005344 072233 1          ldx m2isp
005345 052006 1          era neth x
005346 006070 1          ana [endbit]
005347 101040 1          snz                        ;is this pkt from us?
005350 003434 1           jmp m2ifre                ;yes, throw away pkt
005351 021462 1          jst m2iack                 ;process acks

;Now, hand the packet off to TASK. This consists of placing it on the TASK queue
;and GPR'ing the TASK process.

005352 072233 1   m2t0:   ldx m2isp
005353 000010 1          %rdclok                    ;get arrival time
005354 050003 1          sta artm x                 ;for delay measurement
005355 105700 1          lda [etq] i
005356 010001 1          sta ireg
005357 132001 1          stx ireg i                 ;put on task queue
005360 133700 1          stx [etq] i
005361 072136 1          ldx tskpci                 ;get ptr to TASK PCB
005362 000043 1          gpr                        ;poke it

005363 072232 1   m2idun: ldx mp
005364 165367 1          irs stats.mtoti ix         ;**stats
005365 003043 1           jmp m2iint                ;try for more
005366 003043 1          jmp m2iint                 ;try for more

                         stats.mtoti:
005367 040000 1                 stats.dummy
                         stats.lecs:
005370 003700 1                 noc.lecs
                         stats.bufti:
005371 040000 1                 stats.dummy

                         .section heap
                         .lev var
                         stats.dummy:
040000          V               .block ch
```

```
                        .section pg5
                        .lev m2i

005372 072232 1   pktch1: ldx mp
                         ;defhlt(/15. software checksum error in packet)
005373 021401 1          jst pktchk

005374 072232 1   pktch3: ldx mp                    ;report "dmc" error
                         ;defhlt(/16. m2i bad packet length for adder)
005375 021401 1          jst pktchk

;modem to imp packet-error stuff

005376 000000 1   pktch0: 0                         ;trap, flush, and quit
005377 120126 1          jst hltncc i
005400 103701 1          jmp [m2ifr1] i

005401 000000 1   pktchk: 0                         ;address of pkt error
005402 120126 1          jst hltncc i               ;report trap to ncc
005403 072233 1   pktchc: ldx m2isp
005404 121702 1          jst [diagpt] i             ;put on diag queue
005405 003363 1          jmp m2idun                 ;dismiss

005406 000000 1   pktch2: 0
005407 120126 1          jst hltncc i
005410 120123 1          jst killi i                ;kill lines for spurious acks
005411 003403 1          jmp pktchc                 ;continue
```

```
;Here through the dispatch mechanism when a NULL packet is received.

005412 072233 1   m2inul: ldx m2isp
005413 044011 1           lda srch x          ;got a null packet of acks
005414 012106 1           era mine            ;compare with mine
005415 100040 1           sze                 ;ignore acks if line looped
005416 021462 1            jst m2iack         ;process acks first
005417 003433 1           jmp m2ifr1

;Here when a demand reload is received.

005420 044012 1   m2idmc: lda seqh x          ;password ok?
005421 013653 1           era paswrd
005422 100040 1           sze
005423 021436 1            jst m2idm1         ;no
005424 044011 1           lda srch x          ;panic demand?
005425 101400 1           smi
005426 000000 1            %crash            ;received demand relode
005427 004232 1           lda mp               ;no, use our modem no
005430 010100 1           sta ncca             ;and nice-stop reload, save m
005431 004031 1           lda [ncc.stop]
005432 010101 1           sta nccc            ;force a stop command
005433 072233 1   m2ifr1: ldx m2isp           ;and free pkt
005434 120115 1   m2ifre: jst flushi i
005435 003363 1           jmp m2idun

005436 072232 1   m2idm1: ldx mp
                          ;defhlt(/19. reload demand with bad password)
005437 021401 1           jst pktchk
```

```
;special-type pkt

005440 072233 1   m2ispc: ldx m2isp
005441 044007 1           lda typh x              ;get special type code
005442 141140 1           icl
005443 100100 1           slz 0&sptcod
005444 003420 1            jmp m2idmc 0&sptdmr    ;demand reload
005445 004053 1           lda two 0&sptpkc        ;packet core
005446 120117 1           jst maxchi i            ;room in reassembly?
                          ;defhlt(/10. no room for packet core)
005447 021376 1           jst pktch0              ;no, trap and ignore
005450 105703 1           lda [host+fhppkc] i     ;yes, is packet core up?
005451 012052 1           era [hstup]
005452 100040 1           sze
005453 003433 1            jmp m2ifr1             ;no, discard packet
005454 024220 1           irs nrea                ;yes, count in reassembly
005455 105704 1           lda [epcq] i
005456 010001 1           sta ireg
005457 132001 1           stx ireg i 0&ptrc       ;add to packet core queue
005460 133704 1           stx [epcq] i
005461 003363 1           jmp m2idun
```

```
005462 000000 1   m2iack: 0
005463 072201 1          ldx mbl
005464 044106 1          lda maxoctet x          ;highest legal octet number

005465 010001 1          sta ireg
005466 072233 1          ldx m2isp
005467 101040 1          snz                     ;old format line?
005470 003641 1           jmp m2i8ch             ;yes
005471 044006 1          lda neth x              ;no, get octet number from pa
005472 006074 1          ana [octet]
005473 141340 1          ica
005474 010237 1          sta m2ioct              ;save octet number
005475 022001 1          cas ireg                ;is it a legal octet number
005476 000000 1           %crash                 ;no
005477 101000 1           nop                    ;yes
005500 040477 1          lgr 1                   ;divide octet number by 2
005501 014201 1   m2iak1: add mbl                ;offset modem block pointer t
005502 010002 1          sta jreg                ;access correct octet
005503 044007 1          lda typh x              ;get ack bits
005504 140401 1          cma                     ;tsex = next bit to send, so

005505 141050 1          cal                     ;complement rsex bits from t
005506 010001 1          sta ireg                ;save typh
005507 072002 1          ldx jreg                ;get saved offset modem block
005510 004237 1          lda m2ioct
005511 100100 1          slz                     ;is it an odd octet?
005512 003643 1           jmp m2iodd             ;yes, get right half of tsex
005513 004001 1          lda ireg                ;no, get left half of tsex w
005514 052055 1          era tsex x
005515 141050 1          cal                     ;keep old left half
005516 052055 1   mboth:  era tsex x
005517 066055 1          ima tsex x              ;store new tsex word and
005520 052055 1          era tsex x              ;see if anything changed
005521 101040 1          snz                     ;any acks?
005522 103462 1           jmp m2iack i           ;no acks, so quit
005523 011661 1          sta ackt                ;save acks to process
005524 046075 1          ana chfree x            ;should be no acks on free s
005525 100040 1          sze
                         ;defhlt(/17. spurious ack)
005526 021406 1           jst pktch2
005527 005661 1          lda ackt
005530 052075 1          era chfree x            ;free up acked channels
005531 050075 1          sta chfree x
005532 140040 1          cra
005533 010235 1          sta numfre              ;use numfre to count acks
005534 072201 1          ldx mbl
005535 004237 1          lda m2ioct
005536 006015 1          ana [177776]            ;if octet number is odd, subt
005537 041475 1          lgl 3                   ;multiply by 8
005540 054123 1          add i2mtab x            ;add to address of first chan
005541 010000 1          sta 0                   ;to get address of first poss
005542 005661 1          lda ackt                ;slot for these acks
005543 100000 1          skp
005544 024000 1   acklop:  irs 0                 ;next channel slot
005545 040477 1          lgr 1
005546 101001 1          ssc                     ;was this channel acked?
005547 003544 1           jmp .-3                ;no, try next
005550 011661 1          sta ackt                ;yes,save ackt for later
005551 032236 1          stx slotaddr            ;save channel slot address to
```

```
005552 140040 1          cra
005553 066000 1          ima 0 x                ;check packet pointer
```

```
005554 072201 1            ldx mbl
005555 101400 1            smi
005556 022064 1             cas [2000]              ;real buffer address?
005557 003562 1              jmp ackgud             ;yes (minus or >2000)
005560 101000 1              nop                    ;no
                           ;defhlt(/18. quasi-impossible spurious ack)
005561 021406 1            jst pktch2              ;debug spurious ack
005562 010001 1   ackgud: sta ireg                  ;save packet pointer
005563 024235 1            irs numfre              ;one more channel freed
005564 044125 1            lda retrqi x            ;get retr queue pointer
005565 072001 1            ldx ireg
005566 000042 1            rmq                     ;remove packet from retr queu
005567 101000 1             nop                    ;return here if queue now emp
005570 004001 1            lda ireg                ;restore packet pointer
005571 072201 1            ldx mbl
005572 062142 1            cas i2mnxt x            ;is this i2m's current packe
005573 100000 1             skp                    ;no
005574 003631 1              jmp acksyn            ;yes, we cant free it yet
005575 010000 1            sta 0
005576 120115 1            jst flushi i            ;flush it
005577 024223 1            irs nsfs

                           ;fall into ACKS1
```

```
                        ;from previous page

005600 121650 1  acks1:  jst stats.retr i        ;** stats
005601 004201 1          lda mbl                 ;delay takes packet pointer i
                                                 ;modem block pointer in A
005602 121705 1          jst [delay] i           ;add packet delay
005603 044004 1          lda wrdc x
005604 141340 1          ica
005605 100400 1          spl 0&trcpkt            ;tracing?
005606 003634 1           jmp actr2              ;yes, go do it
005607 141050 1  actr1:  cal 0&pktsiz            ;get # wrds in the pkt
005610 072201 1          ldx mbl
005611 054121 1          add thrupw x
005612 100400 1          spl
005613 004071 1           lda sign               ;mark overflow
005614 064122 1          irs thrupt x
005615 050121 1          sta thrupw x
005616 005661 1  acktr3: lda ackt
005617 101040 1          snz                     ;have we processed all the ac
005620 003623 1           jmp chcheck            ;yes, so quit
005621 072236 1          ldx slotaddr            ;continue from this slot
005622 003544 1          jmp acklop

005623 044105 1  chcheck:lda chcounter x
005624 016235 1          sub numfre              ;subtract number of channels
005625 050105 1          sta chcounter x
005626 101040 1          snz                     ;are all channels free now?
005627 024217 1           irs nsfa               ;yes, nullify one nsfs
005630 103462 1          jmp m2iack i            ;and return

005631 050143 1  acksyn: sta i2mack x            ;set i2mack to show ack'ed
005632 010000 1          sta 0
005633 003600 1          jmp acks1

005634 011662 1  actr2:  sta m2isiz
005635 004232 1          lda mp
005636 120145 1          jst trace.m2idone i     ;** trace packet
005637 005662 1          lda m2isiz
005640 003607 1          jmp actr1

005641 010237 1  m2i8ch: sta m2ioct              ;m2ioct is 0 for 8 channel li
005642 003501 1          jmp m2iak1

005643 004001 1  m2iodd: lda ireg                ;get ack bits from packet
005644 141340 1          ica                     ;in left half of a, since odd
005645 052055 1          era tsex x              ;get tsex's right half
005646 141044 1          car
005647 003516 1          jmp mboth
```

```
005650 005651 1   stats.retr: jstm2i

005651 000000 1   jstm2i: 0
005652 103651 1           jmp jstm2i i

                          ;defplc(/demand reload password)
005653 051664 1   paswrd: 51664                    ;keep next locations in order
005654 000000 1   dmndcr: 0                       ;neth
005655 140400 1           sptty+sptcpt+sptdmr     ;typh
005656         1          .block 1                ;checksum
005657 177777 1   dmndcd: -1                      ;reload code
                                                  ;-1 for directed nice stop du
                                                  ;0 for cyclic panic dumper
                                                  ;1-ch for directed panic dump
005660 000000 1   dmndcp: 0                       ;password put here manually

                  .lev var
005661         V  ackt:   .block 1
005662         V  m2isiz: .block 1

                  .section pg0
                  .lev var
000232         V  mp:     .block 1
000233         V  m2isp:  .block 1
000234         V  miocb:  .block 1
000235         V  numfre:  .block 1               ;number of channels freed by
000236         V  slotaddr: .block 1              ;save channel slot address fo
000237         V  m2ioct:  .block 1               ;octet number for ack process

                  .section heap
040016         V  m2idmy:  .block 1               ;single word used for dummy i
```

```
                .stitl IMP TO MODEM (I2M)
                .INCLUDE i2m.m4
;NMFS notes:

;I2MNUL (and ZPK) seem to want to do output but never do it!



;I2M's PCB has only the get/put headers and one IOCB

        000026      i2mpcb.getqh = pcb.ff               ;get queue header
        000032      i2mpcb.putqh = i2mpcb.getqh + qhedr.size ;put queue header
        000036      i2mpcb.iocb = i2mpcb.putqh + qhedr.size ;IOCB
        000044      i2mpcb.size = i2mpcb.iocb + biocb.size ;size of an M2I PCB
```

```
                .section pg6
                .lev i2m

;Modem-out interrupts (both software and hardware) converge here.
;First, we try to acquire the presumably-free IOCB on our PUT queue.
;Throughout I2M, "OCHN" contains the modem number (0 to CH-1) and
;"OIOCB" points to the IOCB which we should use for outputting.

006174 021632 2   i2mini: jst i2mnmf              ;do NMFS initialization
006175 000020 2           pcb                     ;get modem number
006176 044025 2           lda pcb.num x
006177 010000 2           sta 0
006200 145665 2           lda [mblock] ix         ;get modem block pointer
006201 010000 2           sta 0
006202 015666 2           add [nullarea]          ;initialize nulptr to point
006203 050144 2           sta nulptr x            ;to first word of null area
006204 014012 2           add [smq-nullarea]      ;initialize queue end pointer
006205 050154 2           sta emq x               ;point to queue front pointer
006206 014052 2           add [smpq-smq]          ;same for priority queue
006207 050155 2           sta empq x

006210 000020 2   i2mC:   pcb                     ;get ptr to our process
006211 044025 2           lda pcb.num x           ;get modem number
006212 010243 2           sta ochn                ;save it for easy reference
006213 004101 2           lda nccc                ;is there a command to be dor
006214 101040 2           snz
006215 003242 2            jmp i2mnc              ;nope
006216 022011 2           cas [ncc.umod]          ;is it codes 1,2,3?
006217 003242 2            jmp i2mnc              ;nope
006220 003240 2            jmp i2mul              ;3 - unloop
006221 101100 2           sln 0&ncc.imod          ;code 1 or 2?
006222 003236 2            jmp i2mil              ;2 - interface loop
006223 004054 2   i2mml:  lda [%cm1.mloop]
006224 010001 2   i2mdo:  sta ireg                ;save the command
006225 004243 2           lda ochn                ;is it for us?
006226 012100 2           era ncca
006227 100040 2           sze
006230 003242 2            jmp i2mnc              ;nope, skip it
006231 010101 2           sta nccc                ;yes, clear the command word

006232 004001 2           lda ireg
006233 000020 2           pcb
006234 000403 2           xdv                     ;do the operation
006235 003242 2           jmp i2mnc

006236 004011 2   i2mil:  lda [%cm1.iloop]
006237 003224 2           jmp i2mdo

006240 004012 2   i2mul:  lda [%cm1.unloop]
006241 003224 2           jmp i2mdo
```