NMFS CONSOLE COMMANDS

Command		Example	Function
A		A	Open the emulated A register (203R)
nD		1076D	Open the specified dispatch memory cell
E		E	Give the console to the macrocode application
nG		2000G	Reset then start application at specific addres
H		H	Halt application
nI		2I	Open the specified I/O address
nJ		26J	Jump to specific microcode address
nM		105M	Open the specified macromemory address
N		N	Open the current NMFS process register (206R)
P		P	Open the emulated PC register (200R)
nR		200R	Open the specified microregister
S		S	Open the emulated SP register (211R)
nU		20667U	Open the specified microcode URAM word
nW	*	1234W	Trap if macromemory location 1234 is altered
W		W	Cancel previous memory watch
X		X	Open the emulated X register (205R)
@		@	Has the value of last typed number
^		^	Open previous address
<lf>		<lf>	Open next address
<cr>		<cr>	Close any open address
n<cr>		7<cr>	If there's an open addr, deposit n, close it
ctrl-N		ctrl-N	Give the console back to the MBB (see "E" cmd)

* Note: It is possible to instruct the "W" command to watch for changes in only part of a given address by placing the desired bit mask value in microregister 243. The default value for 243R is all ones, meaning watch for any bit change. If one set 243R to 100003, for example, only chanes in the sign or the low two bits would be considered a watch hit. Location 243 is not reset to ones unless the microcode is reloaded - this is left to the user.

NMFS OPCODES, etc.

OPCODES (* indicates function not yet implemented)

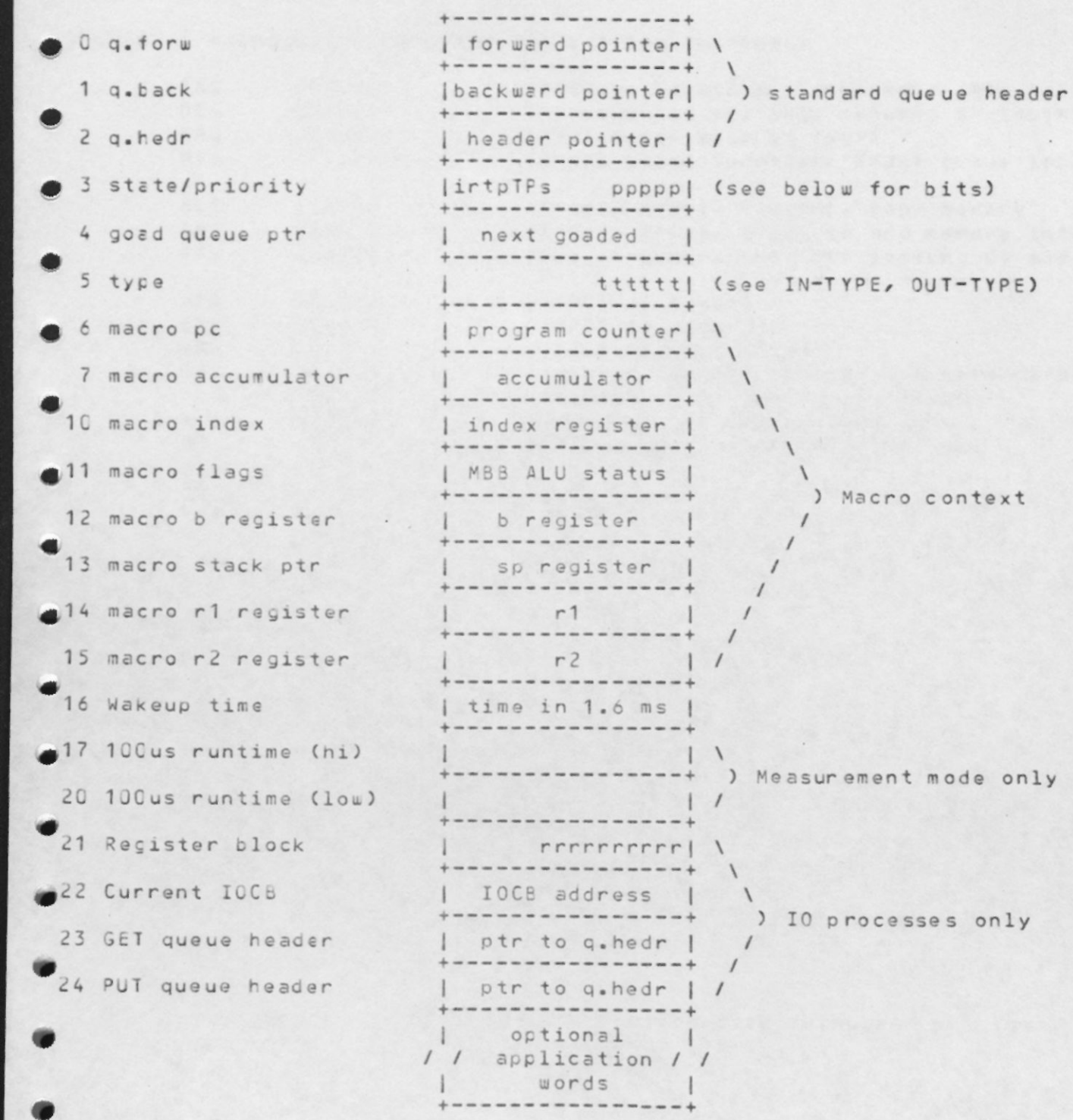
```
        ENQ     000002          ;A=queue location, X=element, skip len>1
        DEQ     000022          ;A=queue header, skip returns X=element
        RMQ     000042          ;X=element, skip if len>1
        MVQ     000102 *        ;A=src queue location, X=dst queue location

        BLT     000031          ;X=src, A=dst, B=count, copy memory
        CHK     000032          ;X=src, A=sum, B=count, add memory into A
        FFO     000033          ;in: A=number, out: A=# leading 0s and skips

        APR     000003          ;X=PCB to create
        DPR     000023          ;X=PCB to destroy
        GPR     000043          ;X=PCB to goad (poke)
        SPR     000103          ;suspend current process, skp=poked, else t.o.
        TPR     000203          ;A=interval (1.6ms), sets timeout
        XDV     000403          ;A=opcode, X=PCB, perform device function
        PDV     001003          ;X=PCB, poke micro-device driver

        RDCLOK  000010          ;get 100us clock into A
        LITES   000011          ;A=low 16 lights, X=hi 4 (req. MII in slot#1)
        MEMHI   000012          ;A=highest allowed macromemory addr
        PCB     000020          ;get current PCB addr into X
        NMFS    000030          ;A=0, turn off NMFS, A=IDLEQ, start NMFS there

        PUSH e  MR OP #14       ;(SP)=e, SP=SP-1
        CALL e  MR OP #16       ;(SP)=PC, SP=SP-1, PC=e
        POP  e  MR OP #17       ;SP=SP+1, e=(SP)
        RETN    100002          ;SP=SP+1, PC=(SP)
        SRETN   100003          ;SP=SP+1, PC=1+(SP)
        IRETN   100010          ;(SP+1)=1+(SP+1)
        CASP    100011          ;SP=A
        CSPA    100012          ;A=SP
        CSPX    100013          ;X=SP
        PUSHA   101002          ;(SP)=A, SP=SP-1
        POPA    101003          ;SP=SP+1, A=(SP)
```

C18 (MII 1822) XDV OPCODES, etc.

```
        %C18.RESET      000000  ;XDV: abort current IOCB with
        %C18.RAISE      000001  ;XDV: raise IMP ready line
        %C18.LOWER      000002  ;XDV: lower IMP ready line
        %C18.STATUS     000003  ;XDV: read 1822 CSR into A register (below)
        %C18.LOOP       000004  ;XDV: loop 1822 interface
        %C18.UNLOOP     000005  ;XDV: unloop 1822 interface

        %C18.EOM        000200  ;IOCB.FLAGS: end-of-message indicator

        %C18CSR.IMPR    000002  ;CSR: state of IMP ready line
        %C18CSR.XPAT    000020  ;CSR: state of interface loop (1=looped)
        %C18CSR.FLAP    002000  ;CSR: host ready was down (flapped)
        %C18CSR.DOWN    004000  ;CSR: host ready line is down
```

```
CM1 (MII 2651, 24-bit CRCs) XDV OPCODES, etc.

        %CM1.NOOP       000000  ;XDV: no-operation
        %CM1.RESET      000001  ;XDV: reset hardware and enable I/O
        %CM1.DESET      000002  ;XDV: reset hardware and disable I/O
        %CM1.ILOOP      000003  ;XDV: loop interface
        %CM1.MLOOP      000004  ;XDV: loop modem
        %CM1.UNLOOP     000005  ;XDV: unloop both interface and modem
```

NMFS DATA STRUCTURES - PCB

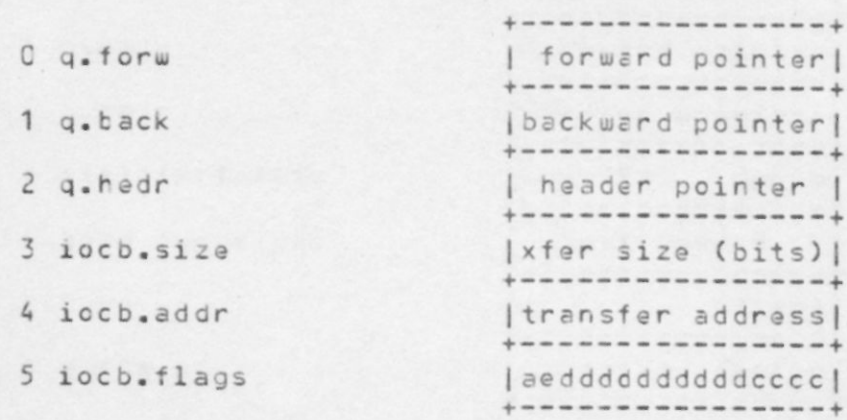


Bits in State/Priority word (3):

i = process is idle	r = process is ready
t = process is timing	p = process is pending
T = process has been timed	P = process has been poked

s = process wants to skip pppp = priority (0=highest, 37=lowest)

An I/C Control Block (IOCB) has the following format:



Format of IOCB flags:

Bits	Meaning
a	Interrupt always (whenever this IOCB is placed on the PUT queue)
e	Interrupt on non-zero completion code (see cccc bits)
cccc	Completion code for this IO (generally, 0=okay, non-0=error)
d .. d	Device-specific bits

NMFS CONFIGURATION AREA

All numbers are octal unless otherwise noted.

```
3000 - 3007     trunk #0 config block
3010 - 3017     trunk #1 config block
3020 - 3027     trunk #2 config block
                .
                .
                .
3150 - 3157     trunk #15 config block
3160 - 3167     trunk #16 config block *
3170 - 3177     trunk #17 config block *

3200 - 3207     host #0 config block
3210 - 3217     host #1 config block
3220 - 3227     host #2 config block
                .
                .
                .
3350 - 3357     host #15 config block
3360 - 3367     host #16 config block
3370 - 3377     host #17 config block

3400 - 3417     system config block
3420 - 3437     copy of RB.STATE (20) after last crash
3440 - 3457     copy of RB.MAIN (200)
3460 - 3477     copy of RB.CLOCK (40)
3500 - 3517     copy of RB.IOOS (300)
3520 - 3537     copy of register block located at BASE in RB.STATE

3600 - 3617     IMP Neighbor Table, 16 (decimal) trunks
3620 - 3637     IMP Trunk State Table, 16 trunks (see below)
3640 - 3657     IMP Host State Table, 16 hosts (see below)
```

Values for IMP Trunk and Host state tables:

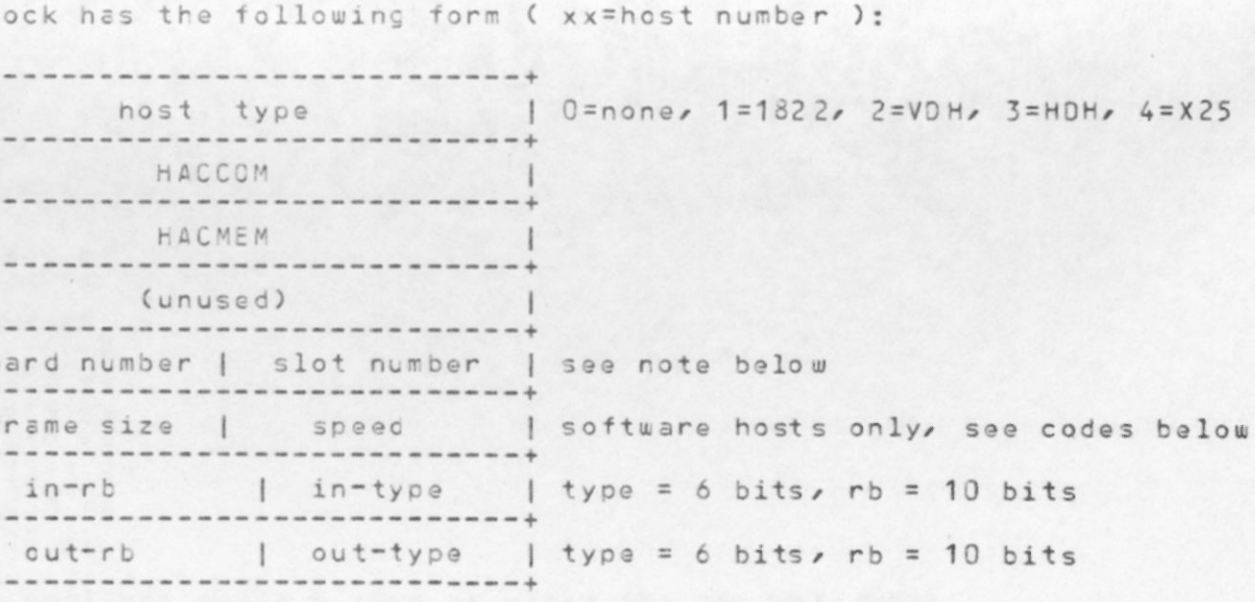
	Trunk (3620+n)	Host (3640+n)
0	up	up
1	down	down
2	coming up	tardy
3	N.A.	not configured
4	waiting to come up	not initialized
-n	silent	N.A.

(*) Due to limitations in the NMFS IMP, trunks #16 and #17 are not available for use. Their config blocks are reserved for future expansion.

Each trunk block has the following form (xx=modem number):

```
        +--------------------------------+
3xx0:   |          trunk type            | 0=none, 1=normal trunk
        +--------------------------------+
3xx1:   |  number of IMP-IMP channels    | 8. to 128. channels, by 8.
        +--------------------------------+
3xx2:   |     propagation delay          | in 800 microsecond units
        +--------------------------------+
3xx3:   |   unused (12 bits)   | logno   | logical line assignment, 0 to CH-1
        +--------------------------------+
3xx4:   | board number |  slot number    | see note below
        +--------------------------------+
3xx5:   |             speed              | see codes below
        +--------------------------------+
3xx6:   |     in-rb      |  in-type      | type = 6 bits, rb = 10 bits
        +--------------------------------+
3xx7:   |     out-rb     |  out-type     | type = 6 bits, rb = 10 bits
        +--------------------------------+
```

Each host block has the following form (xx=host number):



Notes:

1. Board number codes, and slot number codes

The board number refers to which C/30 I/O board contains the configured device. Board numbers run from 1 to n, with the lowest board being board 1 We are trying to keep boards in the following order:

1. MII boards - lowest boards
2. MTI boards - next highest boards
3. Msynch boards - above MTI and MII boards

The slot number indicates which interface slot is used on the indicated board. For boards with heterogenous interfaces (MII, MTI), the in-type field must be used to tell to which type of interface the slot number refers.

2. Frame size codes (software hosts only)

Maximum size frame

Index	VDH	HDH	X.25 L3 data bytes	
0	vdh	pktmod	128	(defaults)
1			32	
2			64	
3		pktmod	128	
4			256	
5			512	
6		msgmod	1024	

3. Line speed codes

note: trunks only use codes 0 thru 3, hosts can use any speed

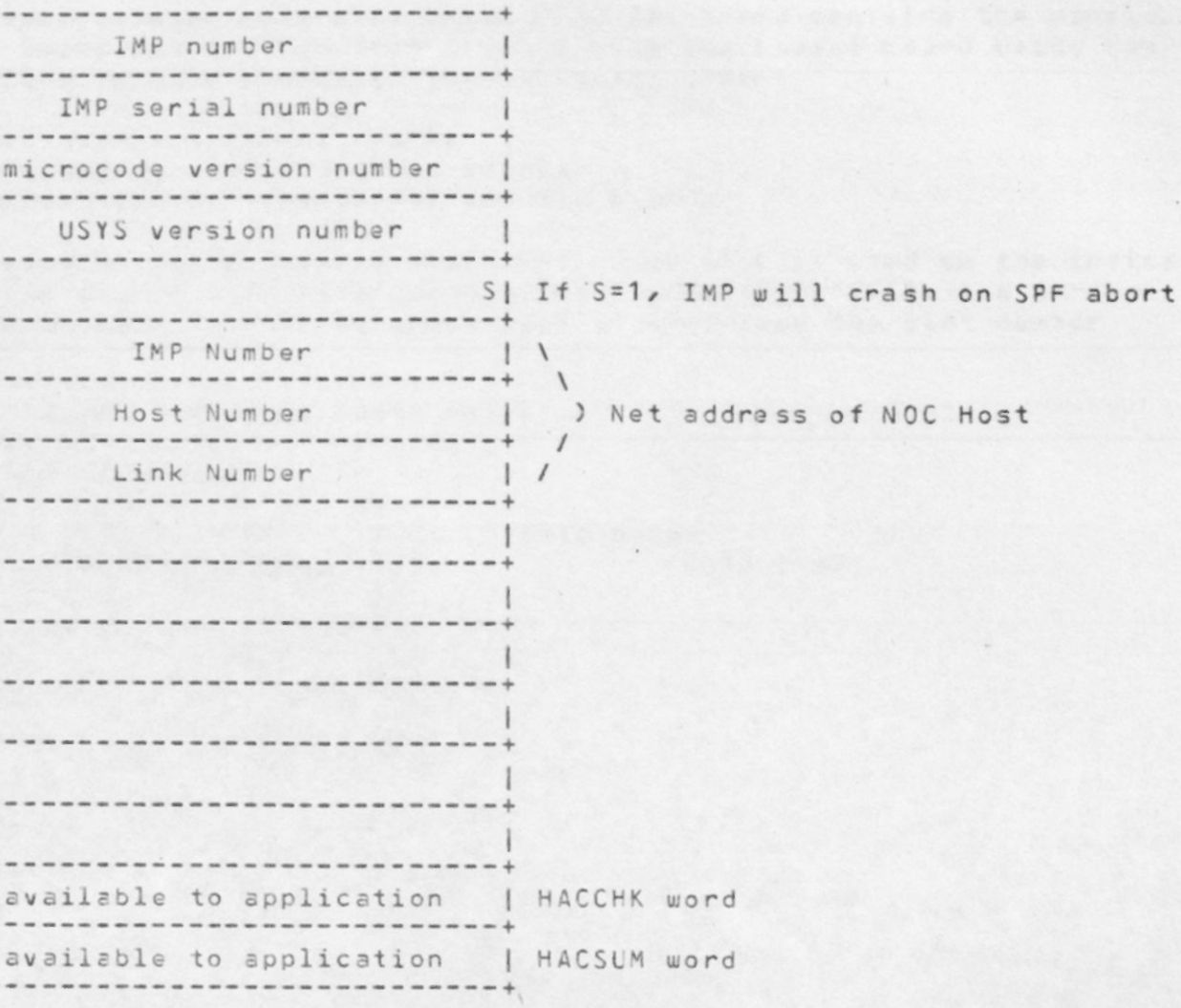
0 = 50 KB
1 = 230 KB
2 = 19.2 KB
3 = 9.6 KB
4 = 4.8 KB
5 = 2.4 KB
6 = 1.2 KB
7 = 56 KB
8 = 64 KB
9 = 100 KB

4. Choose in-type and out-type from the following list:

Code	Function
0	software devices only, never configured
1	MII board 2651 driver, 24 bit CRC, input
2	MII board 2651 driver, 24 bit CRC, output
3	MII board 1822 driver, input
4	MII board 1822 driver, output

```
    5           MBP console/cassette, input
    6           MBP console/cassette, output
    7           MTI board 2652 driver, HDLC, input
   10           MTI board 2652 driver, HDLC, output
   11           MTI board 1822 driver, input
   12           MTI board 1822 driver, output
   13           MSYNC 2652 driver, HDLC, input
   14           MSYNC 2652 driver, HDLC, output
```

The system config block has the following format:

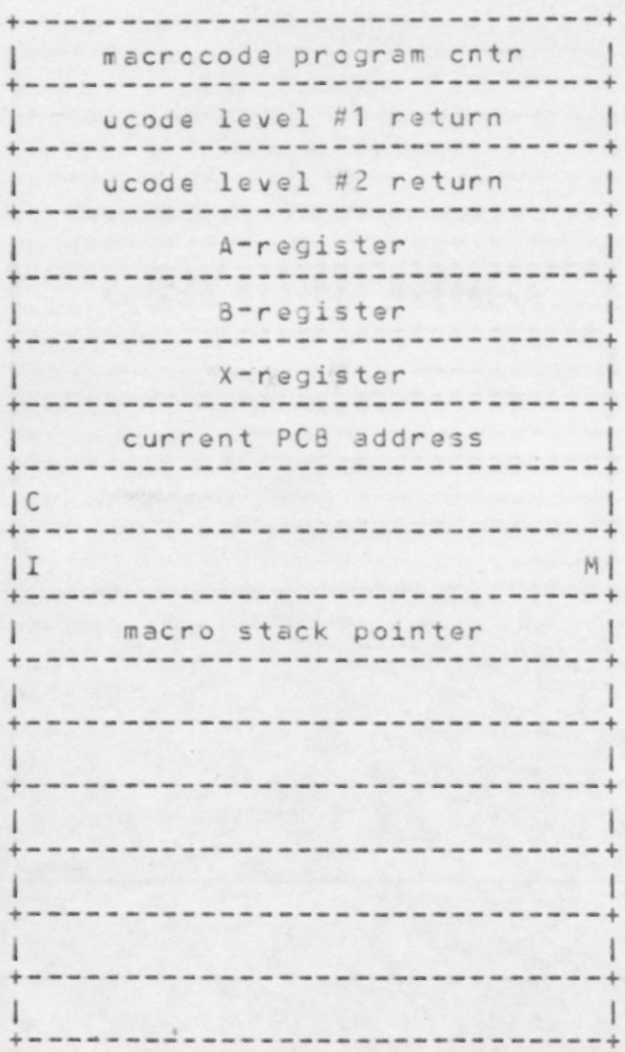


RB.STATE, RB.MAIN, RB.CLOCK, RB.IOOS and a register block determined dynamically are saved in macromemory from 3420 - 3537. The contents of each is given below. Further documentation can be found in the microcode source itself.

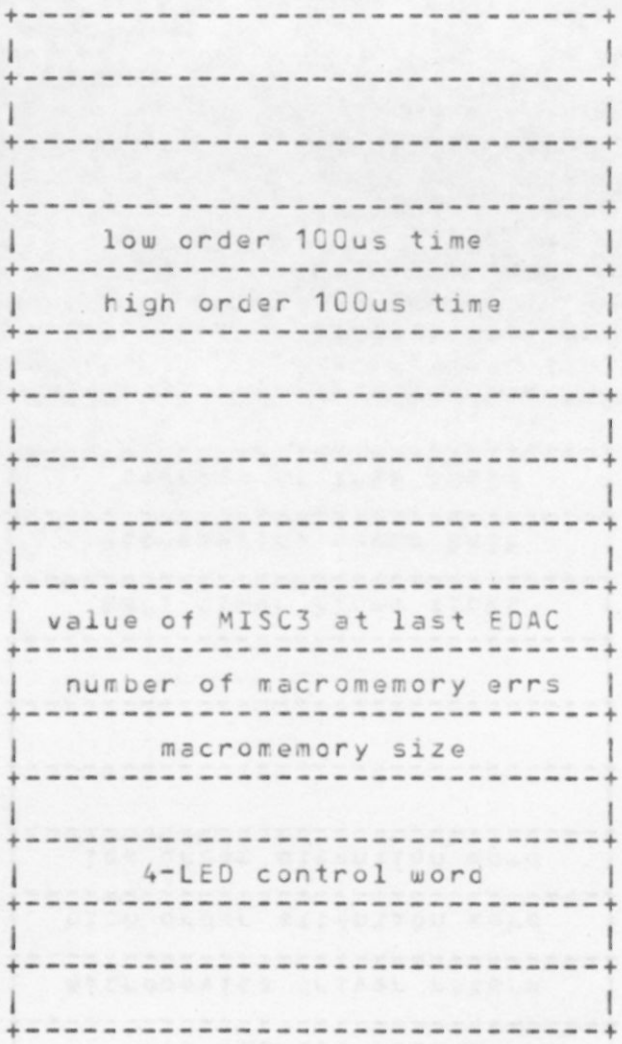
RB.STATE (20):

3420:	BASE	(used to determine 3520-3537)
3421:		
3422:		
3423:	MAR	
3424:	MBR	
3425:	crash code / TEMP	(holds trap number after a trap)
3426:	INTS	(vector addr of next microinterrupt)
3427:	ALUST	(ALU statis register)
3430:	SN	(MBP serial, same as 3401)
3431:	MISC	(MBB "miscellaneous register")
3432:	MISC2	(MBB "miscellaneous register")
3433:	MISC3	(MBB "miscellaneous register")
3434:	MIR	
3435:		
3436:		
3437:	reload disable password	set to 333 if tape reload inhibited

RB.MAIN (200):


3440:
macrocode program cntr
3441:
ucode level #1 return
3442:
ucode level #2 return
3443:
A-register
3444:
B-register
3445:
X-register
3446:
current PCB address
0 if NMFS turned off
3447:
C
C=carry bit (sign)
3450:
I
M
I=INHibit, M=Measurement mode
3451:
macro stack pointer
3452:
3453:
3454:
3455:
3456:
3457:

RB.CLOCK (40):


3460:
3461:
3462:
3463:
low order 100us time
3464:
high order 100us time
3465:
3466:
3467:
3470:
3471:
value of MISC3 at last EDAC
3472:
number of macromemory errs
3473:
macromemory size
3474:
3475:
4-LED control word
3476:
3477:

RB.IOCS (300):

3500:	address of PENDING queue	
3501:	microcode level #1 return	
3502:	microcode level #2 return	
3503:	microcode level #3 return	
3504:	address of READY queue	
3505:	address of TIMING queue	
3506:	PCB address pointer	(all PCB operations act on this one)
3507:	microdevice driver return	
3510:	high order attention word	(sign=priority 0, lsb=priority 15.)
3511:	low order attention word	(sign=priority 16., lsb=priority 31.)
3512:		
3513:		
3514:		
3515:	NMFS timer 1.6ms ticks	
3516:	microdevice saved BASE	
3517:	address of IDLE queue	

NMFS MICROCODE REGISTER USAGE

Register Blocks	Use
0000 - 0177	USYS (see top of USYS-PROM.MIC for definitions)
0200	RB.MAIN, main instruction emulation (see 316.MIC)
0220	RB.MSHIFT, used during 316's shift instructions
0240	RB.SOFTINT, used with MBB's software interrupt
0250	RB.TID, shared with RB.SOFTINT, used for timing heap
0260	RB.BITS, table of bits, used throughout
0300	RB.IOOS, main NMFS IO base, see PM.MIC
0320	RB.DEV, pointers to NMFS micro device drivers
0340 - 0377	-- reserved for 316/NMFS expansion
0400 - 1777	-- for configuration-specific device drivers

NMFS NCC COMMAND WORDS IN NCCC/NCCA

```
ncc.null=0                      ;nothing to do, idle
ncc.lmod=1                      ;loop modem
ncc.imod=2                      ;loop interface
ncc.umod=3                      ;unloop modem/interface
ncc.lhst=4                      ;loop host [interface]
ncc.uhst=5                      ;unloop/reset host [interface]
ncc.stop=6                      ;nice-stop/reload, NCCA=modem
ncc.boot=7                      ;nice-stop/re-boot from tape (halt)
ncc.rest=10                     ;nice-stop/re-start IMP
ncc.reld=11                     ;panic-stop/reload, NCCA=modem
ncc.igd=12                      ;imp-going-down, reason in ncca
```

NMFS MICROCODE TRAPS

Number Meaning

1 HALT. The macrocode executed a HLT instruction, opcode 000000. Look at register 0 of RB.MAIN (L0.PC) to determine the address of the halt. Register L6.CURR in RB.MAIN has the current PCB address, if that helps.

2 Instruction not implemented yet. The program attempted to execute an opcode which is defined, but not implemented (e.g., MVQ). Debug it as if it executed an illegal instruction or a halt (i.e. look at L0.PC in RB.MAIN, etc.).

3 Watch hit. The user has executed a W command and the associated memory value has changed.

4 Illegal instruction. The macrocode executed an unassigned opcode. debug just like traps #1 or #2, above.

5 Illegal stack pointer (SP). The macrocode executed a stack instruction with an illegal value (0) in the SP register. Check the SP register contents to find out its current value.

6 JST, JMP, or CALL to location 0. The macrocode did a JST, JMP or CALL which would have gotten it to location 0. In NMFS, this instruction is treated as an illegal instruction with L0.PC, etc. being preserved. Debug it like any other illegal instruction.

7-17 Should never occur.

20 Illegal forward pointer. This results when a queueing primitive detects an inconsistent forward pointer somewhere (pointing to 0, for example). It is generally caused by one of two culprits:

i) A macrocode ENQ/DEQ/RMQ with bad args, or,
ii) A microcode manipulation of some NMFS data which was "stepped on" by the macrocode (clobbered).

The first thing you should do is look at register 0 of RB.STATE (which has a saved copy of BASE). If BASE=200 (RB.MAIN), then the machine was probably executing user instructions and you should proceed to find the bad ENQ/DEQ/RMQ located at the current instruction (check L0.PC in RB.MAIN, etc.). If BASE<>200, it's probably 700, which is the NMFS base. This case is harder because you'll have to figure out what NMFS was trying to do at the time the queueing primitives bombed out. Generally, it will be manipulating a PCB or an IOCB. The best thing to do is to write down the octal values in L1, L2 & L3 of RB.IOOS and look them up in the microcode listing, in order (L1 first). Whatever's in L1 is generally the address of whoever called the queuing primative that failed. L2

has whoever call THAT routine. Occaisionally, L3 is used, too. By looking up the subroutines' addresses, you can see what it was trying to do and what went wrong. Generally, RB.IOOS holds all the answers (you'll usually end up tracking this down to some macromemory bogusness caused by a failing macro program).

21 Illegal back pointer. Just like trap #20, different bad word.

22 Illegal header pointer. Just like trap #20, different bad word.

23 Illegal length. Similar to #20-#22, but it means that the length in the queue header didn't agree with the contents of the queue. You should proceed through the procedure for #20, but this type of problem is generally more subtle, having to do with macrocode which either smashes the queue's length word or play very tricky games with the contents without using ENQ/DEQ/RMQ, etc.

24-27 Should never occur.

30 PCB at zero. The macrocode tried to manipulate a PCB which is supposedly at location 000000. Find the offending instruction by tracking down the current macro PC (see trap #1, etc.), then look at the A (L3.A in RB.MAIN) and X (L5.X in RB.MAIN) registers to see what's going on.

31 Should never occur.

32 XDV of idle processes. Macrocode tried to XDV a process which was idle (not APR'ed).

33 RMQ.IDLE, not idle state. Someone (almost certainly the macrocode in this case) tried to remove a PCB from the idle heap (presumably the only place this is done is during an APR) but the PCB didn't have the idle bit set in its state word. Track this down like #31.

34 RMQ.IDLE, not idle queue. Similar to #33. The PCB claims to be in the idle state (has the idle bit set in the state word) but its queue header pointer doesn't seem to point to the idle queue. Clearly someone is confused. Track this down like #31.

35,36 RMQ.READY, not ready state/queue. Like #33/#34 except referring to the ready queue, not the idle queue. #33 and #34 are almost always caused directly by macrocode instructions, #35 and #36 can also happen as a result of some NMFS manipulations at microcode level (RMQ.READY is the routing to remove a PCB from the READY queue.. this occurs whenever a process is GOADed, etc.). The first thing you should do is find out what PCB is in trouble. See L6.PCB in RB.IOCS. Then look at its state word and header pointer. You should see the trouble.

37,40 RMQ.TIMING, not timing state/queue. Same as #35/#36, but timing.

41,42 RMQ.PENDING, not pending state/queue. Same as #35/#36, but pending.

43 RMQ.PENDING, didn't see attention bit set. This means that the machine took some PCB off the pending queue and noticed that it was the last (the pending queue is now empty). Since the pending queue went from non-empty to empty, NMFS turns off the "attention bit" associated with that priority level, but found it was already

off (implying that the PCB shouldn't have been on the queue in the first place). The attention bits (which live in L10.ATTN1 and L11.ATTN2 of RB.IOOS) are set for each priority that has at least one process which wants to run. This helps NMFS find the highest priority process to run when it needs to context switch. This is almost always caused by the macrocode DEQ or RMQ'ing PCBs from a pending queue (clearly this is illegal).

44 ENQ.PENDING, didn't see attention bit clear. Similar to #43 but means that when the first PCB was queued on an otherwise empty pending queue, the corresponding attention bit was already set indicating that it shouldn't have been empty. Usually cause by the macrocode ENQ'ing onto a pending queue directly.

45 Pending queue was empty but attention bit set. NMFS went to schedule a process associated with the highest-order attention bit but found no process at that priority. Probably macrocode DEQ'ed or RMQ'ed the process off the pending queue which left the attention bit set.

46 Illegal PCB.TYPE. The pcb in question (see L6.PCB in RB.IOOS) had an illegal type. Type 0 is software, others are chosen from the same list as in-type and out-type elsewhere in this document.

47 Nothing to run. NMFS could find no processes to run (the attention bits were all zero). Macrocode must always supply at least one runnable process (hopefully the lowest priority one).

50,51 Should never occur.

52 Empty idle queue at start-up. The macrocode did a NMFS instruction (opcode 30) setting up the addresses of the idle, ready, timing and pending queues. NMFS went to automatically APR and GPR the first process on the idle queue for you (to get things rolling) but found nothing there. Fix your program.

53 PCB.RB messed up. When you create (APR) a process which has a non-zero type, you are creating an I/O process. The PCB.RB field of your PCB determines what register block the microcode should use to do I/O for that device, 1600(8), for example, must correspond to modem #0. The microcode device driver for the device records the address of the associated PCB in one of these registers. Whenever you reference the device (XDV for example), the device driver makes sure your PCB matches the one it was bound to by NMFS. This trap results if that comparison fails. The macrocode has probably stepped on PCB.RB after APR'ing the process. The current PCB is probably the culprit. BASE (as saved in L0.BASE of RB.STATE) is the register block that the device driver was called with (will match PCB.RB of the PCB).

54,55 Bad XDV args. These codes are returned when an XDV is done with illegal argument(s). For example, if you try to do function number 100 on a ceivce which supports functions 0-5, you'll get one of these. Find the bogus XDV at the current PC and look at L3.A and L5.X in RB.MAIN for details.

56 Illegal IOCB.SIZE. An IOCB had an illegal transfer size word. Most commonly this is a value of 0 or one which is not a multiple of the I/O byte size. Find the PCB which owns the IOCB in L6.PCB of RB.IOOS (not necessarily the current one!) and then look at PCB.IOCB to find the address of the offending PCB.

57 Transfer count when negative. This should never happen. It is a serious microcode bug which says that a device driver output or input more than the IOCB said to.

60 Illegal IOCB address. The microcode device driver saw an illegal IOCB address (probably 0). Find the current PCB at L6.PCB in RB.IOOS, find the current IOCB at PCB.IOCB, etc.

61 uDD attempted to GET second IOCB ("uDD" = microcode device driver) The microcode attempted to set up more than one current IOCB. Find the current PCB at L6.PCB in RB.IOOS, then locate the address of the old IOCB in PCB.IOCB. This trap is given from IODD.GET in NMFS (see IO.MIC). Look at L7.DDUPC in RB.IOOS to find out the microcode address of whoever call IODD.GET. This can also be caused by the macrocode setting PCB.IOCB non-zero itself.

62 uDD attempted to PUT non-existant IOCB. Similar to trap #61, above. Track down the confused microcode device driver through L7.DDUPC, etc.

63 uDD couldn't shake IOCB. Serious microcode bug indicating the failure of IODD.PUT to get rid of the current IOCB in use by the microcode device driver.

64 Illegal sequence of APRs. Some devices insist on having one direction be in existance before they'll let you APR the other direction. Generally, the input side comes first (for those devices which care). Find the offending APR at the current PC.

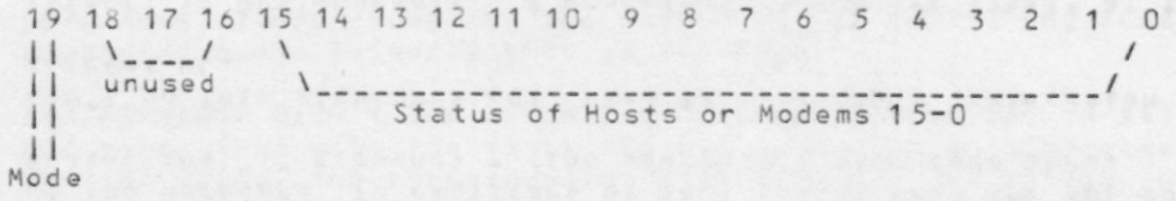
65 Attempt to SPR while INHibited. The macrocode attempted to debreak from the current process (SPR) but left the machine INHibited. This is clearly illegal (it makes no sense, either). Find the offending SPR at the current PC.

66 uDD saw unexpected interrupt. This is a serious microcode bug which says that the microcode got an interrupt from a device which shouldn't be interrupting. You can generally identify which device from the saved BASE in L0.BASE of RB.STATE. If possible, you'll want to look at the hardware I/O registers of that device from the MBB console (using the "I" command) - find yourself a microcode guru.

67 Can't do this with NMFS off. Find the offending instruction at the current PC.

70 Illegal to DPR yourself. A process tried to DPR itself at the current PC.

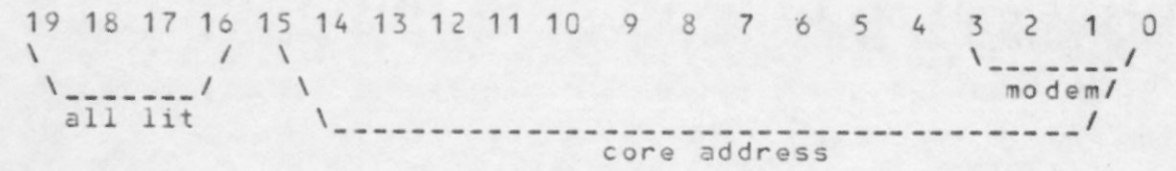
NMFS IMP LIGHTS DISPLAY



If Mode = 0, bits 0-15 refer to the modem status. If a given modem's bit is off, that modem is down. If it is on, that modem is up, if it is flashing, that modem is up and looped.

If Mode = 1, bits 0-15 refer to the host status. If a given host's bit is off, that hosts is down. If it is on, that host is up, if it is flahsing, that host is tardy.

NMFS LOADER LIGHTS DISPLAY



When running the loader (a.k.a. IMPLOD, a.k.a. LOD) lights 16-19 are lit and lights 0-15 have two possible displays: if the loader is not transferring data, positions 0-3 indicate which trunk the loader is using (0=first trunk, 2=trunk #2, etc.). When the loader has a transfer going, lights 0-15 display the last "piece address" which generally will count up, but does not necessarily reflect the memory address the loader is operating on.

STARTING NMFS and IMPLOD-4300

To start the NMFS Loader/Dumper (IMPLOD-4300) cycling through modems 0-3:

```
1000G
```

To start the loader locked on modem "n" (n=0,1,2,3):

```
A  xxxxxx  n <cr>          eg:   A  120342 3 <cr>
1001G                            1001G
```

To cause the loader to demand-reload from the neighbor, use 100000+n:

```
A  xxxxxx 100000+n <cr>  eg:  A   120342 100003 <cr>
1001G                         1001G
```

To start or restart the IMP program:

```
2000G
```

To halt the IMP program or IMPLOD, use "H".

Although the loader will automatically start the IMP after an IMP-IMP reload, it is sometimes necessary to tell IMPLOD to jump to a specific address from the NCC. To do this, place the desired address in location 2001 via packet core (U/NU DDT, or BROADCAST). Deposit a 2000 to start/restart the IMP or 2002 to cause IMPLOD to boot from tape.

```
 5653    /demand reload password
11506    /hi - waiting for tsb slot for host reply
12640    /nop here to turn off i2h checksum check
14276    /dztb - goes with jam
14302    /wttb - goes with suck
14306    /sltb - goes with sleep [back hosts]
15111    /free transmit, receive block counters
17645    vha - task hook
26657    /parameters table
26733    /message generator number
30214    /nop to disable loading other imps
30324    cassette package hook
27547    cassette package hook
43074    /spfrut
43275    receive message block table
44425    transmit message block table
45411    tsbtab - transaction block table
45741    reastb - reassembly block table
```

```
EEEEEE  H    H    AA    H    H  N    N
E       H    H   A  A   H    H  NN   N
EEEEE   HHHHHH  A    A  HHHHHH  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 20:58:18 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 20:58:18 1982

```
EEEEEE  H    H    AA    H    H  N    N
E       H    H   A  A   H    H  NN   N
EEEEE   HHHHHH  A    A  HHHHHH  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
```

```
    0  pg0
  232  m2i
  240  i2m
  243  i2m
  245  h2i
  272  i2h
  306  tsk
  310  spf
  316  con-b
  316  con-e
 3000  noc
 3410  con-b
 3410  con-e
 4000  ini
 4560  con-b
 4643  con-e
 5000  m2i
 5372  m2i
 5663  con-b
 5713  con-e
 6000  ini
 6174  i2m
 6665  con-b
 6705  con-e
 7000  i2m
 7537  con-b
 7603  con-e
10000  h2i
10645  con-b
10745  con-e
11000  h2i
11744  con-b
12000  h2i
12001  con-e
12135  i2h
12721  con-b
13000  i2h
13033  con-e
13721  con-b
13764  con-e
14000  bck
14652  con-b
14717  con-e
15000  bck
15623  con-b
15674  con-e
16000  tsk
16657  con-b
16714  con-e
17000  tsk
17701  con-b
17742  con-e
20000  tsk
20542  con-b
20577  con-e
21000  tsk
```

```
 21332  con-b
 21355  con-e
 22000  zud
 22703  con-b
 22740  con-e
 23000  tim
 23203  utl
 23346  dpy
 23657  con-b
 23727  con-e
 24000  zud
 24103  tim
 24713  con-b
 24777  con-e
 25000  fak
 25337  con-b
 25363  con-e
 26000  fak
 26735  con-b
 27000  con-e
 27000  fak
 27172  fak
 27630  con-b
 27704  con-e
 30000  fak
 30534  con-b
 30565  con-e
 31000  utl
 31671  con-b
 31712  con-e
 32000  h2i
 32317  con-b
 32370  con-e
 33000  spf
 33512  con-b
 33512  con-e
 34000  spf
 34705  con-b
 34705  con-e
 35000  spf
 35672  con-b
 35672  con-e
 36000  spf
 36600  con-b
 36600  con-e
 40000  m2i
 40016  m2i
 40017  h2i
 40761  h2i
 41005  i2h
 41021  i2h
 41457  tsk
 41530  utl
 46101  heap
 47770  con-b
 47770  con-e
```

```
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 20:58:36 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 20:58:36 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 20:58:36 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 20:58:36 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
```

```
 4032    i.def: IMPDEF of address 0
 4072    I.NMFQ: returned from NMFS(idleq) or ENQ skipped
 5141    XDV of CM1.RESET didnt get back stale IOCB
 5430    received demand relode
 6342    I2MIDL: saw extra IOCB
 6347    I2MIDL: NMFS timeout
 6533    M2I: saw extra IOCB
 6652    I2MNMF: found extra IOCB
 7123    I2M: got poked while waiting for packet core to ??
10036    HIR2: got poked
10063    HICHEW: timeout waiting for EOM
10575    HI2TSK: NMFS timeout
10601    HI2TSK: awoke but TSKFLG still zero
32074    HI8TSK: NMFS timeout
32100    HI8TSK: Awoke but TSKFLG=0
11020    H2INMF: found extra IOCB
11064    HIWSPR: timed out
11156    HIEOM: called without an IOCB setup
11206    HIDEQ: cant DEQ IOCB
11222    HIENQ: extra IOCB?
12065    SUCK: IOCB disapeared during xfer
12102    SUCK: saw extra IOCB
12145    I2HINI: got poked
12243    IHT: Fake host went tardy
12404    IH: IH queue foulup
12667    IHFLDQ: bad function
12672    IHFLDQ: bad function
13052    IHWSPR: timed out
13172    IHDEQ: cant DEQ IOCB
13206    IHENQ: extra IOCB?
13244    I2HNMF: found extra IOCB
13672    SUCK: IOCB disapeared during xfer
13701    SUCK: saw extra IOCB
14142    spurious background return
14177    background saw timeout stop
14265    BCK: NMFS timeout
15223    buffer word count smashed - background
15262    GIVTSK: TASK didnt take packet
16004    TSKDB: task awakened from NMFS timeout
16652    tagged packet checksum smashed
20244    buffer word count smashed - task
24125    TODB: got poked
23023    add chain broken
23033    add chain return jump broken
23037    software wdt fired
25324    DDSGIV: mailbox overrun
30530    packet core buffer smashed
31053    JUQC: queue counter went negative
31137    buffer flushed too many times
31352    BUFCHK: past it: crash
23243    background found code checksum broken
23523    DPYVEW: got poked
23652    DPYSUB: got poked
33043    yes! another imp with same number!
34261    routing queue broken
35044    spf cas error
```

```
35163     spf cas error
35205     spf cas error
35260     spf cas error
35354     spf cas error
35442     spf cas error
35505     spf cas error
36107     SPF: abort
36112     SPF: abort
36146     SPFDB: timed out
36173     spf cas error
36312     spf cas error
36327     spf cas error
36416     spf cas error
36444     spf cas error
```

```
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 N
E       H     H  A    A   H     H  N   NN
EEEEEE  H     H  A    A   H     H  N    N
```

Wed Oct 27 20:58:50 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 20:58:50 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 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   HHHHHH   A    A  HHHHHH  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 20:58:50 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 20:58:50 1982

```
EEEEEE  H    H     AA    H    H  N    N
E       H    H    A  A   H    H  NN   N
EEEEE   HHHHHH   A    A  HHHHHH  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
```

```
 5205    51. no buffers for input
 5341    /9. accepting packet on dead line
 5374    /15. software checksum error in packet
 5376    /16. m2i bad packet length for adder
 5440    /19. reload demand with bad password
 5450    /10. no room for packet core
 5527    /17. spurious ack
 5562    /18. quasi-impossible spurious ack
 7201    /75. i2m - routing update with bad length
 7203    /76. i2m - routing update with intra-imp xsum error
 7331    /23. i2m bad length packet for adder
 7332    /24. modem out detected intra-imp checksum error
 7433    /25. too many retransmissions of discarded packet
 7450    /78. i2m: 32 retransmissions
10354    55. HI18FW: long wait for buffer
10444    30. HI1GMW: host blocked waiting messno for single packet msg
10451    57. HI1GMW: long wait messno for single packet msg
10461    60. HITSBW: long wait for REQ1 TSB
10613    65. HI2TSK/HITOOK: long wait for TASK to take a data packet
32003    31. HI8GMW: host blocked waiting for messno for REQ8/MESS8
32010    59. HI8GMW: long wait for messno for REQ8/MESS8
32025    60. HI8GGW: long wait for TSB for MESS8
32037    61. HI8GQW: long wait for TSB to queue a REQ8
32102    62. HI8TSK: long wait for TASK to take REQ8
32105    47. HI8G8W: host block waiting for REQ8
32123    63. HI8TKW: long wait for ALL8
32126    48. HI8G2W: host blocked waiting for messno for MESS8
32135    64. HI8G2W: long wait for messno for MESS8
32150    29. HI1TO: first packet timed out
32165    32. HI8TO: middle packet timed out
11504    56. HILPUT: long wait for TSB for host reply
11736    /27. hi bad packet length, a=length/host, x=pkt
12412    /33. IH packet too long for adder
12414    /34. IH detected intra-IMP checksum err
15567    /3. sending inc? to imp=x, messno=a
16067    /73. invalid source imp number discovered in task
16315    /53. throwing away pkt for dead imp=x
17030    /5. recvd transmission from dead imp
17117    /45. no reasblk for mesg inc, inc?, or gvb
17124    /37. no reas block for msg from imp=x
17341    /49. sending duplicate reply to imp=x, messno=a
17346    /66. ignoring inc?
20157    /50. sending inc reply to imp=x, messno=a
20204    /13. sending out of range to imp=x, messno=a
20311    /67. bad transaction block for imp=x, messno=a
20315    /68. missing transaction block for imp=x, messno=a
20542    /42. got out of range from imp=x, messno=a
21204    /14. rcvd non-dup getblk from imp=a for active blk=x
22123    /72. line killed in zdedl, x=neighbor number
22256    83. lowering to A channels on line X
22404    /79. master kills line due to slave says hello
22435    /71. slave kills line due to master down
22447    /4. neighbor imp changed
22460    /77. killing line X due to channel mismatch
22661    /74. line up, a=ticks to come up, x=neighbor no.
27024    /80. line up with too few channels
```

```
30427     /40. oversize packet core on line=a from imp=x
27241     /41. got "setup send" with pg 1 checksum=a on line=x
27272     /81. got setup send: no dump required
27533     /43. packet core abort recieved
27564     /44. sending packet core abort
34520     /82. spf: gen. retransmission w/bad packet length
```

```
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 N
E       H     H  A    A   H     H  N   NN
EEEEEE  H     H  A    A   H     H  N    N
```

Wed Oct 27 20:59:12 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 20:59:12 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 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 20:59:12 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 20:59:12 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 21:00:50 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 21:00:50 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   HHHHHH  A    A  HHHHHH  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 21:00:50 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 21:00:50 1982

```
EEEEEE  H    H    AA    H    H  N    N
E       H    H   A  A   H    H  NN   N
EEEEE   HHHHHH  A    A  HHHHHH  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
```

```
004300      sa.ver=4300                    ;Loader/Dumper for NMFS-power

                .insym imp.sym

            ;Known bugs/deficiencies:
            ;
            ;       %memhi logic replaced by fixed 30K load to get around
            ;       coretype is currently NMFS (2)
            ;       "page 1 checksum" isn't implemented
            ;       BAK is not background, OUT is - fix documentation, et

            ;Edit history:
            ;
            ;  Who    When              What
            ;
            ;  EAH  13-Nov-81        Made PCB's bigger to hold PCB.F, .R1-
            ;  EAH   9-Feb-82        Moved to pages 1 & 2, restored cyclin
            ;  EAH  25-Feb-82        Added Lights display support, auto-st
            ;  EAH  13-Mar-82        Renumbered modems to 0-3
            ;  EAH  20-Mar-82        lengthened IOCBs, added delay-on-send
            ;  EAH  28-Mar-82        Added auto-start/boot word, set sign
            ;  EAH  10-Apr-82        Increased IOCB.SIZE for input IOCBs t
            ;  EAH  26-Apr-82        Made to work with new IMP, added SA.S
            ;  ARH  8-Jun-82         New subr sa.tct (allow type 1 & 2 cor
            ;  ARH  11-Jun-82        Disable 13 second setup xmit if non-i
            ;  ARH  2-Aug-82         Clean up 13 second disable code
            ;  ARH  4-Aug-82         Replace sa.srf with sa.cti as cycle t
            ;  ARH  18-Aug-82        Allow only NMFS coretypes
            ;
            ;Basic Layout:
            ;
            ;This implementation of the loader/dumper (LD) is build aroun
            ;three NMFS process: IN, OUT, and BAK. IN processes modem inp
            ;OUT, modem output, and BAK is a general backround housekeepi
            ;process.
            ;
            ;The BAK process monitors the top-level state of the LD perfo
            ;"master clears" when it feels things are "stuck". It also co
            ;the OUT process to send periodic "SETUP" messages.
            ;
            ;The OUT process is basically a slave to the other two. It wa
            ;flags to determine when it's time for it to send a given typ
            ;message. SA.STF being non-zero, for example, will cause the
            ;process to send a SETUP. If the OUT process has no explicit
            ;it might send a piece of core, if there is a send transfer i
            ;progress.
```

```
;There are some things which the reader might find confusing.
;since this code wishes to refer to ARPAnet-style packets, it
;to read the IMP symbol table. It is then possible to say "NE
;get the offset into an ARPAnet packet associated with that s
;name. Unforetunately, the IMP defines these offsets to  the

;IMP BUFFER HEADER which precedes them (chain pointer, use co
;Thus, if the LD whish to reference NETH, it will do so by sa
;NETH-HEDR.
```

```
                .pzoff                          ;constants must be on page

000024          s.pcb=t0pcb.size + 3            ;basic PCB + PCB.IOCB + GET +
000006          iocb.app = biocb.size           ;first word avail to us is a

000020          t25.6=16.                       ;25.6ms in  1.6ms NMFS ticks

000773          t13=507.                        ;13.0s  in 25.6ms ticks
004747          t65=5*t13                       ;65.0s  in 25.6ms ticks
```

```
001000          . = 1000                        ;the L/D starts here

                ;Enter at 1000 to cycle, 1001 if A has code to use.

                sa.beg:                         ;top of loader immune memory

001000 003012   sa.go:  jmp sa.goc              ;here to cycle
001001 073664           ldx [0]
001002 033450           stx sa.cyc              ;wihtout hurting A
001003 033451           stx sa.imd
001004 033452           stx sa.lin
001005 100400           spl                     ;is this an immediate reload
001006 025451            irs sa.imd             ;yes, set immediate flag
001007 007665           ana [3]                 ;take off low bits for line
001010 011452           sta sa.lin              ;start there, not at zero
001011 003026           jmp sa.zap              ;then enter the ZAP code

001012 140040   sa.goc: cra                     ;cycling
001013 011451           sta sa.imd              ;never immediate when cycling
001014 025450           irs sa.cyc              ;always cycling
001015 011452           sta sa.lin              ;starting with line 0
001016 003026           jmp sa.zap

                ;This is the "top" of the loader/dumper cycle loop. Initially
                ;various conditions thereafter, we come here to clear the NMF
                ;state and set up to run on a given modem.

001017 005450   sa.nxt: lda sa.cyc              ;cycling?
001020 101040           snz
001021 003026            jmp sa.zap             ;no, skip increment
001022 005452           lda sa.lin              ;get line
001023 141206           aoa
001024 007665           ana [3]                 ;increment mod 4 (0,1,2,3)
001025 011452           sta sa.lin

001026 140040   sa.zap: cra
001027 000030           nmfs                    ;turn off NMFS
001030 111666           sta [sa.ato] i          ;also clean up after previous

001031 073667           ldx [-sa.zrl]           ;clear variables
001032 151670   sa.zll: sta [sa.zra+sa.zrl] ix
001033 033461           stx sa.stc              ;init send counter to -1
001034 024000           irs 0                   ;(notice that x=-1 on last pa
001035 003032            jmp sa.zll

001036 073452           ldx sa.lin              ;get line number
001037 145671           lda [sa.rbi] ix         ;get input rb
001040 111672           sta [sa.ipcb + pcb.rb] i
001041 145673           lda [sa.rbo] ix         ;get output rb
001042 011516           sta sa.opcb + pcb.rb

                ;Init the queue headers

001043 073674           ldx [sa.idlq]           ;init the queues
001044 005675           lda [-sa.qnm]           ;number of queues
001045 011462           sta sa.tm1              ;save in a temp
001046 004000   sa.qlp: lda 0
001047 050000           sta q.forw x
001050 050001           sta q.back x
```

```
001051 050002                sta q.hedr x
```

```
001052 140040            cra
001053 050003            sta q.size x
001054 005676            lda [qhedr.size]        ;bump
001055 014000            add 0
001056 010000            sta 0
001057 025462            irs sa.tm1
001060 003046             jmp sa.qlp

                         ;fall into init the backround PCB
```

```
                        ;from previous page

                    ;init the backround pcb (priority 1)

001061 073677               ldx [sa.bpcb]           ;get ptr to backround pcb
001062 005700               lda [100000 + 1]        ;state=idle, priority=1
001063 050003               sta pcb.pri x
001064 140040               cra                     ;clear type
001065 050005               sta pcb.type x
001066 050021               sta pcb.iocb x          ;and current IOCB
001067 050022               sta pcb.get x           ;and get/put pointers
001070 050023               sta pcb.put x
001071 005701               lda [sa.bak]            ;get pc
001072 050007               sta pcb.pc x
001073 005674               lda [sa.idlq]           ;put it on the idle queue
001074 000002               enq
001075 101000                nop

                    ;init the input process

001076 073670               ldx [sa.ipcb]
001077 005702               lda [100000 + 0]        ;satate=idle, priority=0
001100 050003               sta pcb.pri x
001101 005703               lda [1]                 ;type=1
001102 050005               sta pcb.type x
001103 140040               cra
001104 050021               sta pcb.iocb x          ;no current IOCB
001105 005704               lda [sa.igq]
001106 050022               sta pcb.get x           ;get and put queues, too
001107 005705               lda [sa.ipq]
001110 050023               sta pcb.put x
001111 005706               lda [sa.ine-1]          ;start right at INE, even tho
001112 050007               sta pcb.pc x            ;SPR'ed by SA.BAK
001113 005674               lda [sa.idlq]
001114 000002               enq
001115 101000                nop

                    ;init the output process

001116 073707               ldx [sa.opcb]
001117 005700               lda [100000 + 1]        ;state=idle, priority=1
001120 050003               sta pcb.pri x
001121 005710               lda [2]                 ;type
001122 050005               sta pcb.type x
001123 140040               cra
001124 050021               sta pcb.iocb x          ;no current IOCB
001125 005711               lda [sa.ogq]
001126 050022               sta pcb.get x
001127 005712               lda [sa.opq]
001130 050023               sta pcb.put x
001131 005713               lda [sa.out-1]          ;see comment regarding SA.INE
001132 050007               sta pcb.pc x
001133 005674               lda [sa.idlq]
001134 000002               enq
001135 101000                nop

                        ;fall into IOCB initialization
```

```
                        ;from previous page

                    ;Here to initialize the two IOCBs (IIOCB and OIOCB) by fillin
                    ;fields and placing them on the IPCB and OPCB in the "empty"

001136 073714               ldx [sa.iiocb]
001137 005702               lda [100000]            ;set FLAGS.IALWAYS
001140 050005               sta iocb.flags x        ;no need to ENQ input IOCB be
001141 073715               ldx [sa.oiocb]          ;put output IOCB on output pu
001142 050005               sta iocb.flags x
001143 005712               lda [sa.opq]
001144 000002               enq
001145 101000                nop

                    ;Prepare the EMPTY SETUP message at SA.SET.

001146 073716               ldx [sa.set]            ;get pointer to setup message
001147 005717               lda [22]                ; ... needed, init size
001150 050000               sta sa.set-sa.set x
001151 005447               lda sa.why              ;get reload code
001152 140500               ssm                     ;set sign bit for "nmfs loade
001153 041474               lgl 4                   ;set typh with it and
001154 013451               era sa.imd              ;immediate flag
001155 141050               cal
001156 013720               era [sptty!sptcpt!sptpkc]
001157 050001               sta sa.hed-sa.set x

001160 105721               lda [config.num] i      ;pick up IMP number
001161 140500               ssm                     ;set sign minus to indicate N
001162 050003               sta sa.idn-sa.set x
001163 050013               sta sa.lck-sa.set x     ;set up checksum to equal IMP

001164 005722               lda [ctnmfs<<11]        ;init local coretype
001165 050014               sta sa.lcm-sa.set x
001166 005723               lda [132000]
001167 050010               sta sa.frm-sa.set x     ;init as setup
001170 005703               lda [1]
001171 050020               sta sa.sdf-sa.set x     ;setup "send"
001172 005664               lda [0]
001173 011455               sta sa.srf              ;set process idle
001174 005665               lda [jreg+1]            ;start address expected (skip
001175 050015               sta sa.adr-sa.set x
                    ;        %memhi
001176 005724               lda [73777]             ;&& ask for 30K load regardle
001177 056015               sub sa.adr-sa.set x     ;transfer size expected
001200 050016               sta sa.siz-sa.set x
001201 005725               lda [376]               ;0&fhlpkc
001202 050011               sta sa.lch-sa.set x     ;init local (neighbor) host

001203 005726               lda [-t65]              ;init 65 second piece timer
001204 011456               sta sa.pti
001205 011460               sta sa.cti              ;init cycle timer
001206 005727               lda [-t13]              ;init 13 second SETUP timer
001207 011457               sta sa.t13
001210 011453               sta sa.stf              ;also turn on send setup flag
001211 005674               lda [sa.idlq]           ;master clear and start
001212 000030               nmfs                    ;turn on NMFS
```

```
                        ;This is code is the first thing to run under NMFS after mast
                        ;It starts the two modem processes.

001213 073670   sa.bak: ldx [sa.ipcb]
001214 000003           apr
001215 000043           gpr                     ;note that the process will r
001216 073707           ldx [sa.opcb]
001217 000003           apr
001220 000043           gpr                     ;ditto
001221 005452           lda sa.lin              ;put the modem number in the
001222 141206           aoa                     ;number of left shifts + 1
001223 140407           tca
001224 010000           sta 0
001225 005703           lda [1]
001226 100000           skp
001227 041477           lgl 1
001230 024000           irs 0
001231 003227            jmp .-2
001232 073730           ldx [17]                ;and light all high order lig
001233 000011           %lites


                        ;This is the loader/dumper "backround" loop. It runs every 25
                        ;and "master clears" if we have not commenced a transfer in e
                        ;In addition, it runs the two receive-mode down-timers, SA.PT
                        ;pieces) and SA.T13, a 13 second timer used to output periodi

001234 005731   sa.blp: lda [t25.6]              ;get 25.6 milliseconds
001235 000020           pcb
001236 000203           tpr                     ;set up our sleep time
001237 000103           spr                     ;debreak for that long
001240 101000            nop

001241 005455           lda sa.srf              ;get send/receive flag
001242 100040           sze                     ;busy?
001243 003250           jmp sa.b2               ;yes
001244 025460           irs sa.cti              ;check cycle timer
001245 100000            skp
001246 003017             jmp sa.nxt            ;yes, step
001247 003260             jmp sa.b3             ;run 13 second timer
                        ;Here when it's been 25.6 milliseconds and we're receiving. C
                        ;SA.PTI (piece timer). If it gets to zero, master clear becau
                        ;been a long time between pieces. Also count down SA.T13, a 1
                        ;timer. If it goes to zero, tell the output process to send a
                        ;SETUP by setting the SA.STF flag and poking it.

                        sa.b2:
001250 101400           smi                     ;sending?
001251 003266            jmp sa.bls             ;yes, skip count down
001252 025456           irs sa.pti              ;run the piece timer
001253 100000            skp
001254 003026             jmp sa.zap            ;if it goes to zero, master c
001255 005727           lda [-t13]              ;reset timer
001256 011457           sta sa.t13
001257 003234            jmp sa.blp             ;debreak if busy
001260 025457   sa.b3:  irs sa.t13              ;13 seconds up?
001261 003234            jmp sa.blp             ;no, debreak for 25.6 more
001262 005727           lda [-t13]              ;yes, re-set timer
001263 011457           sta sa.t13
```

```
001264 011453           sta sa.stf          ;and set send setup flag to r
001265 003234           jmp sa.blp          ;done for a while

                ;Here when its been 25.6 ms and we're sending.  Increment thr
                ;so that the output side will see to send a piece.

001266 001001   sa.bls: inh
001267 005461           lda sa.stc          ;bump counter
001270 100400           spl                 ;alredy run out?
001271 141206           aoa                 ;no, bump it
001272 011461           sta sa.stc
001273 000401           enb
001274 003234           jmp sa.blp          ;all done
```

```
001275 000020     sa.odb: pcb
001276 000043             gpr                     ;fix BCK time to run
001277 000103     sa.od2: spr                     ;(enter here from below)
001300 101000              nop                    ;don't care which type of wak

001301 005453     sa.out: lda sa.stf              ;does BACKROUND want to send
001302 100040             sze
001303 003312              jmp sa.osu             ;if so, go do it
001304 005455             lda sa.srf              ;are we busy?
001305 101040             snz
001306 003275              jmp sa.odb             ;idle - wait until sending
001307 101400             smi
001310 003324              jmp sa.ot2             ;sending - go try for a peic
001311 003275             jmp sa.odb              ;and debreak

                  ;Here to output the canned SETUP message and clear the SA.STF
                  ;We try to DEQ the output IOCB from the modem's output PUT qu
                  ;we can't get it, we leave SA.STF on and try later. If we get
                  ;we point it at the SETUP area (SA.SET) and output it. This c
                  ;also use by SA.ABT (send an abort). In that case, the canned
                  ;will contain an ABORT which will be sent by us instead of th

001312 005533     sa.osu: lda sa.opcb + pcb.put
001313 000022             deq
001314 003275              jmp sa.odb             ;try again ASAP
001315 005716             lda [sa.set]            ;get ptr to SETUP
001316 050004             sta iocb.addr x         ;set IOCB pointing there
001317 005732             lda [22*16.]            ;get size (in bits)
001320 050003             sta iocb.size x
001321 140040             cra
001322 011453             sta sa.stf              ;clear "please send a setup"
001323 003413             jmp sa.oeq              ;and ENQ this [setup] for out
```

```
001324 005461     sa.ot2: lda sa.stc              ;okay to send?
001325 100400             spl
001326 003275              jmp sa.odb             ;nope, wait

001327 005533             lda sa.opcb + pcb.put   ;get put queue pointer
001330 000022             deq                     ;get something from it
001331 003275              jmp sa.odb             ; last output never completed

001332 073675             ldx [sa.set-sa.frm-1]   ;copy header
001333 145733             lda [sa.frm+1] ix
001334 051553             sta sa.oiocb+iocb.app+sa.frm+1-sa.set x
001335 024000             irs 0
001336 003333              jmp .-3
001337 073734             ldx [sa.oiocb + iocb.app] ;buf ptr to application are
001340 013735             era [133000?132000]     ;make it 'core' instead of 's
001341 050010             sta data-hedr x
001342 105736             lda [sa.siz] i          ;update transfer size
001343 017737             sub [72]                ;by max # of core words in pk
001344 100400             spl                     ;last piece?
001345 140040              cra                    ;yes, make remainder 0
001346 101040             snz                     ;last piece?
001347 011455              sta sa.srf             ;yes, set flag to idle
001350 127736             ima [sa.siz] i          ;update transfer size
001351 117736             sub [sa.siz] i          ;get old-new (>0)
001352 140407             tca                     ;negative count
001353 011462             sta sa.tm1              ;of words to send
001354 140407             tca                     ;positive size
001355 113740             era [sa.iiocb+iocb.app+data+4-hedr] i ; coretype
001356 050012             sta data+2-hedr x
001357 113740             era [sa.iiocb+iocb.app+data+4-hedr] i ;now, undo core
001360 015741             add [data-neth+4]       ;plus overhead
001361 100100             slz                     ;odd?
001362 141206              aoa                    ;yes, make even
001363 050000             sta neth-hedr x         ;set size in neth
001364 041474             lgl 4.                  ;convert to bits
001365 011537             sta sa.oiocb+iocb.size  ;and store it in IOCB
001366 105742             lda [sa.adr] i          ;start addr
001367 050011             sta data+1-hedr x

001370 105742     sa.sc2: lda [sa.adr] i          ;copy data from mem
001371 011463             sta sa.ti1              ;(SA.TM1 is holding copy coun
001372 105463             lda sa.ti1 i
001373 050013             sta data+3-hedr x       ;to buf
001374 125742             irs [sa.adr] i
001375 024000             irs 0
001376 025462             irs sa.tm1
001377 003370              jmp sa.sc2
001400 140040             cra
001401 050013             sta data+3-hedr x       ;last word=0
001402 005455             lda sa.srf              ;reset counter for next piece
001403 140407             tca
001404 011461             sta sa.stc
001405 073743             ldx [ 13 ]              ;set lites for dump
001406 105742             lda [sa.adr] i          ;get address for lights
001407 000011             %lites
001410 073715             ldx [sa.oiocb]          ;get iocb ptr again
001411 005734             lda [sa.oiocb+iocb.app] ;fill in ADDR
001412 050004             sta iocb.addr x
```

```
;fall into SA.OEQ
```

```
                ;Here to checksum and output OIOCB.

                                                ;enter at OEQ from send SETUP
001413 073540   sa.oeq: ldx sa.oiocb + iocb.addr ;get ptr to data part
001414 005537           lda sa.oiocb + iocb.size ;get size
001415 040474           lgr 4.                   ;div 16.
001416 140407           tca
001417 011462           sta sa.tm1               ;save in counter
001420 114000   sa.ock: add 0 i                  ;compute this part
001421 024000           irs 0
001422 025462           irs sa.tm1
001423 003420            jmp sa.ock              ;compute whole checksum
001424 073540           ldx sa.oiocb + iocb.addr ;get pointer to address agai
001425 140407           tca                      ;compute checksum word value

001426 054002           add chkh-hedr x
001427 050002           sta chkh-hedr x
001430 073715           ldx [sa.oiocb]           ;prepare to ENQ output IOCB
001431 005532           lda sa.opcb + pcb.get    ;get pointer to get queue
001432 000002           enq                      ;place it
001433 101000            nop
001434 000020           pcb                      ;poke it
001435 001003           pdv
001436 003275           jmp sa.odb               ;debreak for a while
```

```
                        ;Register block tables for CM1I/O micro device driver

001437 001200   sa.rbi: 1200
001440 001220           1220
001441 001240           1240
001442 001260           1260

001443 001340   sa.rbo: 1340
001444 001360           1360
001445 001400           1400
001446 001420           1420
```

```
001447          sa.why: .block 1                ;reload code
001450          sa.cyc: .block 1                ;cycle flag (1=yes)
001451          sa.imd: .block 1                ;immediate reload flage (=1),
001452          sa.lin: .block 1                ;line to load from
001453          sa.stf: .block 1                ;our send setup flag, >0 => s
001454          sa.sab: .block 1                ;out send abort flag, >0 => s
001455          sa.srf: .block 1                ;our send/receive flag, =0 =>
                                                ;                       <0 =>
                                                ;                       >0 =>
001456          sa.pti: .block 1                ;piece timer (receive only)
001457          sa.t13: .block 1                ;13-second periodic SETUP tim
001460          sa.cti: .block 1                ;cycle timer
001461          sa.stc: .block 1                ;25.6 ms counter for sending
001462          sa.tm1: .block 1                ;temp for first page
001463          sa.ti1: .block 1                ;another temp for first page

001464          sa.bpcb:  .block s.pcb
001510          sa.opcb:  .block s.pcb
001534          sa.oiocb: .block biocb.size + bufl
```

```
001664 000000          .constants
001665 000003
001666 002001
001667 177755
001670 002316
001671 001437
001672 002324
001673 001443
001674 002472
001675 177767
001676 000004
001677 001464
001700 100001
001701 001213
001702 100000
001703 000001
001704 002516
001705 002522
001706 002250
001707 001510
001710 000002
001711 002526
001712 002532
001713 001300
001714 002342
001715 001534
001716 002273
001717 000022
001720 140000
001721 003400
001722 002000
001723 132000
001724 073777
001725 000376
001726 173031
001727 177005
001730 000017
001731 000020
001732 000440
001733 002304
001734 001542
001735 001000
001736 002311
001737 000072
001740 002364
001741 000014
001742 002310
001743 000013
```

```
        002000      . = 2000

  002000 103536     sa.st:  jmp [init] i            ;start the IMP if user typed
  002001            sa.ato: .block 1                ;deposit address in here and
  002002 000000     sa.boo: hlt

  002003 000020     sa.idb: pcb                     ;make sure microdevice driver
  002004 001003             pdv
  002005 000103             spr                     ;debreak until more input
  002006 101000              nop

  002007 105537     sa.in:  lda [sa.ipcb+pcb.put] i ;try to get at the input IOCB
  002010 000022             deq
  002011 003003              jmp sa.idb             ;nothing to read - try later


  002012 044005             lda iocb.flags x        ;look at completion code
  002013 007540             ana [17]
  002014 100040             sze                     ;must be zero
  002015 003251              jmp sa.ine             ;or we ignore the IOCB

  002016 073346             ldx sa.iiocb+iocb.addr
  002017 044001             lda typh-hedr x         ;is it packet core?
  002020 013274             era sa.hed
  002021 141044             car                     ;ignore lower half
  002022 100040             sze
  002023 003251              jmp sa.ine             ;no

  002024 005345             lda sa.iiocb+iocb.size  ;compute size
  002025 040474             lgr 4.                  ;in words, not bits
  002026 140407             tca                     ;make it negative
  002027 011315             sta sa.cnt
  002030 114000     sa.ich: add 0 i                 ;add in this one
  002031 024000             irs 0
  002032 025315             irs sa.cnt
  002033 003030              jmp sa.ich
  002034 100040             sze                     ;ok?
  002035 003251              jmp sa.ine             ;no

  002036 073346             ldx sa.iiocb+iocb.addr
  002037 044004             lda seqh-hedr x         ;for me?
  002040 013276             era sa.idn
  002041 100040             sze
  002042 003251              jmp sa.ine             ;no

  002043 044003             lda srch-hedr x         ;from right neighbor?
  002044 013277             era sa.stn
  002045 101040             snz
  002046 003056              jmp sa.in2             ;yes
  002047 005277             lda sa.stn              ;no, was neighbor specified?

  002050 100040             sze
  002051 003251              jmp sa.ine             ;yes, reject

                            ;fall into SA.IN1
```

```
                        ;from previous page

                    ;Here when we get a good message from the right neighbor. Pos
                    ;directed and then dispatch on packet type. A is known to be

002052 111541               sta [sa.cyc] i          ;clear the cycle flag
002053 044003               lda srch-hedr x         ;save him
002054 011277               sta sa.stn              ;as our neighbor
002055 011305               sta sa.lci              ;and local imp in our setup m
002056 005307       sa.in2: lda sa.lcm              ;save neighbor line #
002057 052010               era data-hedr x         ;(merged with core type)
002060 007542               ana [177000]
002061 052010               era data-hedr x
002062 011307               sta sa.lcm
002063 044010               lda data-hedr x         ;get packet core type
002064 007542               ana [177000]
002065 013543               era [133000]            ;core?
002066 101040               snz
002067 003123                jmp sa.crr             ;yes
002070 013544               era [132000?133000]     ;no, setup?
002071 101040               snz
002072 003077                jmp sa.sup             ;yes
002073 013545               era [134000?132000]     ;no, abort?
002074 100040               sze
002075 003251                jmp sa.ine             ;no
002076 103546               jmp [sa.nxt] i          ;yes, abort transaction
```

```
                        ;Come here if "setup" received

002077 044011   sa.sup: lda data+1-hedr x       ;copy foreign host
002100 011300           sta sa.frh
002101 044012           lda data+2-hedr x       ;and foreign imp
002102 011301           sta sa.fri
002103 044013           lda data+3-hedr x       ;and link
002104 011302           sta sa.frl
002105 044014           lda data+4-hedr x       ;and line #
002106 011315           sta sa.cnt              ; store away for safe keeping
002107 021146           jst sa.tct              ; test coretype
002110 013547           era [132000]            ;+setup code
002111 011303           sta sa.frm
002112 044015           lda data+5-hedr x       ;copy core addr
002113 011310           sta sa.adr
002114 044016           lda data+6-hedr x       ;and size
002115 011311           sta sa.siz
002116 044017           lda data+7-hedr x       ;and send setup flag
002117 111550           sta [sa.stf] i
002120 044020           lda data+10-hedr x      ;and send/receive flag
002121 111551           sta [sa.srf] i
002122 003251           jmp sa.ine

                        ; come here if "core" message received

002123 005300   sa.crr: lda sa.frh              ;from right source?
002124 052005           era pkth-hedr x
002125 141050           cal                     ;0&desths
002126 015301           add sa.fri
002127 052006           era dsth-hedr x
002130 100040           sze
002131 003251            jmp sa.ine             ;no
002132 044012           lda data+2-hedr x       ;get coretype
002133 011315           sta sa.cnt              ; store it away for safety
002134 021146           jst sa.tct              ; test coretype
002135 105551           lda [sa.srf] i          ; receiving?
002136 101400           smi
002137 003172            jmp sa.cr1             ;no, accept any core
002140 005310           lda sa.adr              ;core addr we are expecting
002141 062011           cas data+1-hedr x       ;vs what we are getting
002142 003251            jmp sa.ine             ;<expecting, ignore
002143 003172            jmp sa.cr1             ;=expecting, ok
002144 125550           irs [sa.stf] i          ;>expecting, send setup send

002145 003251           jmp sa.ine              ;for missing piece

002146 000000   sa.tct: 0                       ;core type test
002147 007552           ana [7000]              ;isolate type
002150 100040           sze                     ;is it type 0?
002151 023553           cas [2000]              ;type 2?
002152 003160            jmp sa.abt             ;zero or >2 , abort
002153 100000           skp                     ; type 2 ok
002154 003160            jmp sa.abt             ; type 1 abort
002155 005315           lda sa.cnt              ; restore A
002156 141050           cal                     ;clear type and MM
002157 103146           jmp sa.tct i            ; exit
```

```
;Come here from Input Processor when we need to queue an ABOR
```

```
002160 005315     sa.abt: lda sa.cnt          ;get A from storage
002161 141050             cal               ;clear coretype and MM
002162 013554             era [134000]      ;mark as abort
002163 011303             sta sa.frm        ;set abort code
002164 005555             lda [pkcsbc]      ;with status of bad coretype

002165 011304             sta sa.frm+1
002166 140040             cra               ;back to idle state
002167 111551             sta [sa.srf] i    ;trigger send setup code to s
002170 125550             irs [sa.stf] i    ;set SEND SETUP for area we j
002171 003251             jmp sa.ine        ;abort this input, start agai
```

```
002172 024000      sa.cr1: irs 0                   ;next start addr
002173 044010              lda data-hedr x
002174 101040              snz                     ;done?
002175 003231               jmp sa.crd             ;yes
002176 011314              sta sa.ptr
002177 024000              irs 0
002200 044010              lda data-hedr x         ;next size
002201 007556              ana [777]               ;isolate size from coretype
002202 101040              snz                     ;zero length?
002203 003172               jmp sa.cr1             ;yes, no core
002204 140407              tca
002205 011315              sta sa.cnt
002206 024000      sa.cr2: irs 0

                   ;Check to see who is sending this piece. If it's the packet c
                   ;host, this is an IMP-IMP reload and we must ignore those par
                   ;data which would destroy ourselves. If the piece is from som
                   ;source (DDT, TENEX, NU), we will blindly allow any addresses
                   ;there is a human at the other end of the transfer.

002207 005300              lda sa.frh              ;foreign host
002210 141050              cal
002211 013557              era [fhlpkc]            ;is it the packet core fake h
002212 100040              sze
002213 003223               jmp sa.cr5             ;no, skip checks (real host)

002214 005314              lda sa.ptr              ;get address

                   ;Make sure the address (in A and SA.PTR) is not inside the lo
                   ;address space, or in the config area

002215 017544              sub [sa.beg]            ;before the beginning?
002216 100400              spl
002217 003223               jmp sa.cr5             ;yes, it's okay
002220 017560              sub [4000-sa.beg]       ;after the beginning - before
002221 100400              spl
002222 003225               jmp sa.cr3             ;in between the two - skip th

002223 044010      sa.cr5: lda data-hedr x         ;from pkt
002224 111314              sta sa.ptr i            ;into core
002225 025314      sa.cr3: irs sa.ptr
002226 025315              irs sa.cnt
002227 003206               jmp sa.cr2
002230 003172              jmp sa.cr1

002231 073561      sa.crd: ldx [ 0 ]               ;clear X for load
002232 005314              lda sa.ptr              ;get addr
002233 000011              %lites
002234 105551              lda [sa.srf] i          ;receiving?
002235 101400              smi
002236 003251               jmp sa.ine             ;no

002237 005314              lda sa.ptr              ;yes, adjust addr
002240 027310              ima sa.adr
002241 017314              sub sa.ptr
002242 015311              add sa.siz
002243 011311              sta sa.siz              ;and size
002244 101400              smi
002245 101040               snz                    ;are we done with transfer?
```

```
002246 003262          jmp sa.tst              ;yes, test for startup
```

```
002247 005562           lda [-t65]              ;no, renew timeout (65 second
00225C 111563           sta [sa.pti] i

                ;come here to start or restart input also initially

002251 073564   sa.ine: ldx [sa.iiocb]
002252 005565           lda [(buf1-1)*16.]      ;size of the IOCB
002253 050003           sta iocb.size x
002254 005566           lda [sa.iiocb+iocb.app] ;addr
002255 050004           sta iocb.addr x
002256 105567           lda [sa.ipcb + pcb.get] i
002257 000002           enq
00226C 101000            nop
002261 003003           jmp sa.idb              ;debreak
```

```
                        ;Here at the end of the transfer.  If this xfer was from pac
                        ;host, then this must be an IMP-IMP reload, therefore we shou
                        ;IMP by jumping to 2000 which will jump to wherever this IMP

002262 005300           sa.tst: lda sa.frh              ;foreign host
002263 141050                   cal
002264 013557                   era [fhlpkc]            ;is it the packet core fake h
002265 101040                   snz
002266 103536                    jmp [init] i           ;start the IMP if this was a

                        ;Now check the auto-start/boot word:

002267 005001                   lda sa.ato              ;look at it
002270 101040                   snz
002271 003251                    jmp sa.ine             ;was zero, continue with inpu
002272 103001                   jmp sa.ato i            ;was non-zero, use it as an a

                        ;reload request/setup send message

                        sa.zra:                         ;this area cleared to zero ea
002273                  sa.set: .block 1                ;size (init to 22)
002274                  sa.hed: .block 1                ;typh (init to 140000 or 1400
002275                  sa.sck: .block 1                ;checksum
002276                  sa.idn: .block 1                ;(init to our imp no)
002277                  sa.stn: .block 1                ;satellite neighbor
002300                  sa.frh: .block 1                ;foreign host
002301                  sa.fri: .block 1                ;foreign imp
002302                  sa.frl: .block 1                ;foreign link
002303                  sa.frm: .block 1                ;foreign line (init to 132000
002304                  sa.lch: .block 1                ;lcl (neighbor) host (init to
002305                  sa.lci: .block 1                ;local (neighbor) imp
002306                  sa.lck: .block 1                ;lcl (neighbor) link (init to
002307                  sa.lcm: .block 1                ;lcl (neighbor) line and core
002310                  sa.adr: .block 1                ;current core address (init t
002311                  sa.siz: .block 1                ;current transfer size (init
002312                          .block 1                ;send setup flag = 0
002313                  sa.sdf: .block 1                ;send/receive flag (init to 1

002314                  sa.ptr: .block 1                ;temp pointer
002315                  sa.cnt: .block 1                ;temp count
        000023          sa.zrl=.-sa.zra                 ;end of zeroed area

002316                  sa.ipcb:  .block s.pcb
002342                  sa.iiocb: .block biocb.size + bufl ;make the IOCBs

                        ;Queues

                        sa.qbg:
002472                  sa.idlq: .block qhedr.size
002476                  sa.rdyq: .block qhedr.size
002502                  sa.timq: .block qhedr.size
002506                  sa.p0q:  .block qhedr.size
002512                  sa.p1q:  .block qhedr.size
002516                  sa.igq:  .block qhedr.size      ;input get queue
002522                  sa.ipq:  .block qhedr.size
002526                  sa.ogq:  .block qhedr.size      ;output get queue
002532                  sa.opq:  .block qhedr.size
        000011          sa.qnm = (.-sa.qbg)/qhedr.size  ;number of queue headers to i
```

```
        002535      sa.end = .-1                    ;end of L/D

  002536 004000             .constants
  002537 002341
  002540 000017
  002541 001450
  002542 177000
  002543 133000
  002544 001000
  002545 006000
  002546 001017
  002547 132000
  002550 001453
  002551 001455
  002552 007000
  002553 002000
  002554 134000
  002555 000002
  002556 000777
  002557 000376
  002560 003000
  002561 000000
  002562 173031
  002563 001456
  002564 002342
  002565 002420
  002566 002350
  002567 002340

                            .outsym lod.sym
        000000              start
```

%LITES	11	SA.CRR	2123	SA.SAB	1454
.	2570	SA.CTI	1460	SA.SC2	1370
BIOCB.SIZE	6	SA.CYC	1450	SA.SCK	2275
BUFL	122	SA.END	2535	SA.SDF	2313
CHKH	10	SA.FRH	2300	SA.SET	2273
CONFIG.NUM	3400	SA.FRI	2301	SA.SIZ	2311
CTNMFS	2	SA.FRL	2302	SA.SRF	1455
DATA	16	SA.FRM	2303	SA.ST	2000
DSTH	14	SA.GO	1000	SA.STC	1461
FHLPKC	376	SA.GOC	1012	SA.STF	1453
HEDR	6	SA.HED	2274	SA.STN	2277
INIT	4000	SA.ICH	2030	SA.SUP	2077
IOCB.ADDR	4	SA.IDB	2003	SA.T13	1457
IOCB.APP	6	SA.IDLQ	2472	SA.TCT	2146
IOCB.FLAGS	5	SA.IDN	2276	SA.TI1	1463
IOCB.SIZE	3	SA.IGQ	2516	SA.TIMQ	2502
JREG	2	SA.IIOCB	2342	SA.TM1	1462
NETH	6	SA.IMD	1451	SA.TST	2262
PCB.GET	22	SA.IN	2007	SA.VER	4300
PCB.IOCB	21	SA.IN2	2056	SA.WHY	1447
PCB.PC	7	SA.INE	2251	SA.ZAP	1026
PCB.PRI	3	SA.IPCB	2316	SA.ZLL	1032
PCB.PUT	23	SA.IPQ	2522	SA.ZRA	2273
PCB.RB	6	SA.LCH	2304	SA.ZRL	23
PCB.TYPE	5	SA.LCI	2305	SEQH	12
PKCSBC	2	SA.LCK	2306	SPTCPT	0
PKTH	13	SA.LCM	2307	SPTPKC	0
Q.BACK	1	SA.LIN	1452	SPTTYP	140000
Q.FORW	0	SA.NXT	1017	SRCH	11
Q.HEDR	2	SA.OCK	1420	TOPCB.SIZE	21
Q.SIZE	3	SA.OD2	1277	T13	773
QHEDR.SIZE	4	SA.ODB	1275	T25.6	20
S.PCB	24	SA.OEQ	1413	T65	4747
SA.ABT	2160	SA.OGQ	2526	TYPH	7
SA.ADR	2310	SA.OIOCB	1534		
SA.ATC	2001	SA.OPCB	1510		
SA.B2	1250	SA.OPQ	2532		
SA.B3	1260	SA.OSU	1312		
SA.BAK	1213	SA.OT2	1324		
SA.BEG	1000	SA.OUT	1301		
SA.BLP	1234	SA.POQ	2506		
SA.BLS	1266	SA.P1Q	2512		
SA.BOC	2002	SA.PTI	1456		
SA.BPCB	1464	SA.PTR	2314		
SA.CNT	2315	SA.QBG	2472		
SA.CR1	2172	SA.QLP	1046		
SA.CR2	2206	SA.QNM	11		
SA.CR3	2225	SA.RBI	1437		
SA.CR5	2223	SA.RBO	1443		
SA.CRC	2231	SA.RDYQ	2476		

```
%LITES              -       8,11,21
.                   3,16:   8,11,23,23,23
BIOCB.SIZE          -       3,15,23
BUFL                -       15,22,23
CHKH                -       13,13
CONFIG.NUM          -       7
CTNMFS              -       7
DATA                -       11,11,11,11,11,11,11,11,18,18,18,19,19,19,19,19,19,
                            19,19,19,19,19,21,21,21
DSTH                -       19
FHLPKC              -       21,23
HEDR                -       11,11,11,11,11,11,11,11,13,13,17,17,17,18,18,18,18,
                            19,19,19,19,19,19,19,19,19,19,19,19,21,21,21
INIT                -       17,23
IOCB.ADDR           -       10,11,13,13,17,17,22
IOCB.APP            3:      11,11,11,11,11,22
IOCB.FLAGS          -       7,7,17
IOCB.SIZE           -       10,11,13,17,22
JREG                -       7
NETH                -       11,11
PCB.GET             -       6,6,6,13,22
PCB.IOCB            -       6,6,6
PCB.PC              -       6,6,6
PCB.PRI             -       6,6,6
PCB.PUT             -       6,6,6,10,11,17
PCB.RB              -       4,4
PCB.TYPE            -       6,6,6
PKCSBC              -       20
PKTH                -       19
Q.BACK              -       4
Q.FORW              -       4
Q.HEDR              -       4
Q.SIZE              -       5
QHEDR.SIZE          -       5,23,23,23,23,23,23,23,23,23,23
S.PCB               3:      15,15,23
SA.ABT              20:     19,19
SA.ADR              23:     7,7,11,11,11,11,19,19,21
SA.ATC              17:     4,23,23
SA.B2               8:      8
SA.B3               8:      8
SA.BAK              8:      6
SA.BEG              4:      21,21
SA.BLP              8:      8,8,9,9
SA.BLS              9:      8
SA.BOC              17
SA.BPCB             15:     6
SA.CNT              23:     17,17,19,19,19,20,21,21
SA.CR1              20:     19,19,21,21
SA.CR2              21:     21
SA.CR3              21:     21
SA.CR5              21:     21,21
SA.CRC              21:     21
SA.CRR              19:     18
SA.CTI              15:     7,8
SA.CYC              15:     4,4,4,18
SA.END              23
SA.FRH              23:     19,19,21,23
SA.FRI              23:     19,19
```

```
SA.FRL          23:     19
SA.FRM          23:     7,11,11,11,19,20,20
SA.GO           4
SA.GOC          4:      4
SA.HED          23:     7,17
SA.ICH          17:     17
SA.IDB          17:     17,22
SA.IDLQ         23:     4,6,6,6,7
SA.IDN          23:     7,17
SA.IGQ          23:     6
SA.IICCB        23:     7,11,11,17,17,17,22,22
SA.IMD          15:     4,4,4,7
SA.IN           17
SA.IN2          18:     17
SA.INE          22:     6,17,17,17,17,17,18,19,19,19,19,20,21,23
SA.IPCB         23:     4,6,8,17,22
SA.IPQ          23:     6
SA.LCH          23:     7
SA.LCI          23:     18
SA.LCK          23:     7
SA.LCM          23:     7,18,18
SA.LIN          15:     4,4,4,4,4,4,8
SA.NXT          4:      8,18
SA.OCK          13:     13
SA.OD2          10
SA.ODB          9:      10,10,10,11,11,13
SA.OEQ          13:     10
SA.OGQ          23:     6
SA.OICCB        15:     7,11,11,11,11,11,13,13,13,13
SA.OPCB         15:     4,6,8,10,11,13
SA.OPQ          23:     6,7
SA.OSU          10:     10
SA.OT2          10:     10
SA.OUT          10:     6
SA.POQ          23
SA.P1Q          23
SA.PTI          15:     7,8,22
SA.PTR          23:     21,21,21,21,21,21,21
SA.QBG          23:     23
SA.QLP          4:      5
SA.QNM          23:     4
SA.RBI          14:     4
SA.RBC          14:     4
SA.RDYQ         23
SA.SAB          15
SA.SC2          11:     11
SA.SCK          23
SA.SDF          23:     7
SA.SET          23:     7,7,7,7,7,7,7,7,7,7,7,7,7,10,11,11
SA.SIZ          23:     7,11,11,11,19,21,21
SA.SRF          15:     7,8,10,11,11,19,19,20,21
SA.ST           17
SA.STC          15:     4,9,9,10,11
SA.STF          15:     7,8,10,10,19,19,20
SA.STN          23:     17,17,18
SA.SUP          19:     18
SA.T13          15:     7,8,8,8
SA.TCT          19:     19,19,19
SA.TI1          15:     11,11
```

```
SA.TIMQ          23
SA.TM1           15:      4,5,11,11,13,13
SA.TST           23:      21
SA.VER           0
SA.WHY           14:      7
SA.ZAP           4:       4,4,4,8
SA.ZLL           4:       4
SA.ZRA           23:      4,23
SA.ZRL           23:      4,4
SEQH             -        17
SPTCPT           -        7
SPTPKC           -        7
SPTTYP           -        7
SRCH             -        17,18
TOPCB.SIZE       -        3
T13              3:       3,7,8,8
T25.6            3:       8
T65              3:       7,21
TYPH             -        17
[0]              4,21:    7
[100000]         6:       7
[100001]         6:       6
[1000]           11,18:   21
[1017]           18
[1213]           6
[1300]           6
[132000]         7,19
[133000]         18
[134000]         20
[13]             11
[140000]         7
[1437]           4
[1443]           4
[1450]           18
[1453]           19:      19,20
[1455]           19:      19,20,21
[1456]           22
[1464]           6
[14]             11
[1510]           6:       8
[1534]           7:       11,13
[1542]           11:      11
[173031]         7,21
[177000]         18:      18
[177005]         7:       8,8
[177755]         4
[177767]         4:       11
[17]             8,17
[1]              6:       7,8
[2000]           7,19
[2001]           4
[20]             8
[2250]           6
[2273]           7:       10
[22]             7
[2304]           11
[2310]           11:      11,11,11
[2311]           11:      11,11
[2316]           4:       6,8
```