Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-001
Date: February 22, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

The iscan utility is an interactive software tool used to observe the internal state of individual boards on the C2. It is generally used in to a system hang or hard crash.

The utilities can be entered by entering the following instructions from the console:

1) cd /hw/cputest
2) iscn <utility name>

The utility will then inform you of the modules being loaded and some basic commands that can be used to access various registers. the prompt will then be displayed as follows:

"iscn68==>"

If you wish to change the head or mcm to be monitored it can be accomplished as follows:

1) :900=<Head #>
2) :901=<MCM #>

If you wish to examine another screen on another utility it is possible to load the the desired utility by the following:

"iscn68==> in "<utility name>"

This is the general instructions for entering the iscan utilities and move around with a minimum of problems.

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-002
Date: February 22, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

Although "mm" is not actually a part of the iscan utilities it runs very much like one and has much useful information. This utility can be run by simply typing mm from the console. As always you must be in the directory /hw/cputest in order to load this utility.

In order to successfully use this utility, it will be necessary to run a cpureg first. The PC, CIR and TID values will be required to successfully run mm. The initial prompt will appear as follows:

mm(0,0): where the first 0 indicates the CIR value and the second 0 indicates the current TID.

Initially the CIR and TID will default to 0, but can easily be changed by the command:

c <CIR 0-7> or t <TID 0-7>

The CIR is the value of the Communication Index Register and the TID is the Thread index Register. The TID being set to anything but 0 will indicate that the head was attempting a parallel operation. Again these can be obtained from the cpureg and must be changed for every head as these values will change the SDR's (Segment Descriptor Registers) which will change the physical memory reference.

Most usage of this utility will involve the PC as this is the pointer that indicates the location of the memory that is in use. As the default for mm is physical memory it will be necessary to change to logical as the PC is a logical value. This can be accomplished by the command

"mm(0,0): s log".

So let's do a quick recap and see how it looks, up to this point:

1) cd /hw/cputest
2) cpureg > file name
3) Get value of PC, CIR and TID from heads to be examined
4) mm
5) mm(0,0): s log
6) mm(0,0): c x
7) mm(x,0): t y
8) mm(x,y):

. . . continued on following page

It will now be possible to examine the instruction stream that the CPU was executing at the time of the crash. To start off, it will be necessary to take the PC value and subtract some value as we would like to see what the CPU was doing prior to the crash. I generally take the arbitrary number of hex 12 and subtract this from the PC. This now gives us PC-12. It also necessary to decide on the number of instructions to examine. I usually use the value 20, since this will place the actual PC near the middle of the stream to be examined.

What we now wish to do is display this on the screen and can be accomplished as follows:

```
mm(x,y): i pc-12,20
```

What follows will be an example display:

```
0x000016a0    put.l    s2,0(a3)
0x000016a6    sub.w    s3,s3
0x000016a8    ld.b     0(a5),s3
0x000016ac    ulk      0(a3)
0x000016b2    eq.w     #0x0,s3
0x000016b6    brs.t    0x16be
0x000016b8    lck      0(a3)
0x000016be    add.w    #0x1,a3
```

At this point the address preceeding the PC value will be the actual instruction be performed at the time of the crash. More can be determined by looking this instruction up in the Architecture Manual. We also have a list of all C2 instructions and a list of boards that are involved in appendix D of the C2 processor Diagnostics Manual. These two sources will give much info about what is occuring at the time of the crash or hang.

In addition to examining instruction flow at the time of the crash it is also possible to examine physical memory and write different patterns to suspect memory. This must be done from physical memory. As said previously, the default is physical, but if "s log" has been entered you can get back to physical by entering "s phys".

The pattern can be used to fill memry locations by the following command:

```
f a1 a2 patt   Where a1 is the first location and a2 is the
               ending memory address.
```

The memory can also be displayed by use of the command di:

```
di a1,count   Where count is the number of memroy addresses to
              display.
```

As stated earlier, this is to be a quick overview. If you wish to know more, all material discussed is detailed in the CONVEX Scan Script Manual.

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-003
Date: February 22, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

The utilities "mcm_func" and "mcm_scr" are designed to examine and interrogate the gate arrays of the mcm's. It is very easy to run and offers a great deal of useful information in the troubleshooting process of the C2. There are several handy commands to be used with these utilities and they permit the examination of the win queue, the XBAR read que, the individual ports as well as memory related error information.

The win que is probably the most widely known and used function of these utilities. the command "dump_win_q" enables the most recent 16 transactions of memory to be viewed on each mcm. The information gained through this command are the active port at the time of the crash and the banks being accessed.

EXAMPLE OUTPUT

"dump_win_q"

```
 Log_char returning EOF-- current register contents for MCM[0] --
win_q_adr     qpx_win     qm_start     rpx_win     rm_start
    c             0           2            0           4

-- contents of the win queue for MCM[0] --
     adr      win     start
      b        0        4
      a        0        1
      9        0        1
      8        0        2
      7        0        2
      6        0        4
      5        0        1
      4        0        1
      3        0        2
      2        0        2
      1        0        4
      0        0        1
      f        0        2
      e        0        4
      d        0        1
      c        0        1
```

This would be repeated for each mcm in the system. As you can see from the above output head 0 was the active head. The memory banks are in the column labeled start. The start column is an 8 bit field where bit 0-7 corresponds to bank 0-7. The adr column indicates the order of the transaction with the top item being the most recent.

. . . continued on following page . . .

. . . continued from previous page

As you can see the win que can be helpful in determining the active head if other attempts fail at determining this.

The next helpful command with this utility is "dump_rd_q". This command will enable you to examine the XBAR read que. This can be helpful in finding some memory read errors.

EXAMPLE

rd_ptr	wr_ptr	rd_ptr	rd_dat
40	8	f	ffffffff
20	8	e	1b840054
10	8	d	2c345679
8	8	9	50513456
4	8	9	50512345
2	8	4	73ffff00
1	8	4	73ff0000
80	8	f	ffffffff

As you can see the crossbar read que is 8 deep and contains memory read data.

The "mcm_err" function will display any hard or soft errors present on any MCM. If errors are found it will display the slot, bank and type of error. If there are no errors then nothing is returned. It is possible for memory errors to show up here that were previously unreported.

The iscan utility mcm_scr is an extension of mcm_func. The difference being that mcm_scr presents more information. In most cases the additional info is unnecessary and only serves to confuse, but in some cases of memory related problems it is quite useful.

The mcm_scr function can be entered by "iscn mcm_scr" if entering directly, or by "in "mcm_scr"" if entering from in another iscan script. The most useful commands in this utility are "port" and "all_ports".

. . . continued on following page . . .

. . . continued from previous page

The "port" command displays the info for the given port on the desired mcm. The mcm can be changed by use of the 901 register, as described in the intro and the port can be changed by issuing another port, for example "port 1". The "all_ports" entry will display all ports for the given mcm.

The fields on these displays are, for the most part self explanatory. The info consists of the complete address including bank and row, parity information, zone (correspond to active byte) and active memory data.

In addition to the above displays, a complete error printout is available for the given mcm. This can be viewed by entering "error" under "mcm_scr" and is quite valuable in finding memory backplane problems. The only time that this display would be used if a positive response is received from "mcm_err" from "mcm_func".

The error screen will display a complete error listing and in most cases get the error to a bit. The display will give the error type, the memory address, the data written and read, the zone and ecc info. If you can get the error to a specific address it is the useful to run "mm" and try to fill that physical address with the same data pattern that failed. This can be used to verify the fault.

As previously stated, "mcm_scr" displays a lot of information and in most cases mcm_func will supply all the info that is necessary for a particular problem.

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-004
Date: February 22, 1990
Subject: Iscan Troubleshooting
Submitted By: Al Haddix

Although cpureg is not actually a part of the Scan Script family it is very necessary information needed to begin to use the Scan Scripts effectively. The utility "cpureg" is very easy to use and can be dumped by entering "cpureg". It is generally quite helpful to redirect the output to a file so that the information is not lost and can be referred to later. It is important to keep in mind that some of the Scan Utilities can cause the pc to advance and so it is important to save the cpu registers before beginning any other activity. This would include hwdump as it is not uncommon for the contents of the pc to be different after the hwdump begins than it was when the crash occured.

It is also standard to receive a cpu register dump after a diagnostic failure, but in the case of a multi-head systems will quite frequently dump the wrong head. So, it might be valuable to run "cpureg" after a diagnostic failure.

****NOTE****

The Scan Utilities are just as effective with a diagnostic failure as they can be with a system crash or hang. Please don't ignore this valuable tool when troubleshooting with diagnostics. I would also like to take this opportunity to point out that no diagnostic should be taken seriously immediately following a crash. This is because a crash can leave the system in an unnatural state that quite often, can only be cleared by removing power and rebooting. So, when troubleshooting, please bear this in mind.

EXAMPLE

```
Register dump for cpu: 0
a0: 00057488 s0: 00000001 00000008 t0: 00003c17 pc: 000016be
a1: 000000ab s1: 0000000d 00000329 t1: 00057436 psw: 80000040
a2: 00002000 s2: ffffffff 00000000 t2: 00001624 ipc: bfd
a3: 00003c17 s3: eaeaea32 00000fff t3: 88000002 ccr: 600040
a4: 00000000 s4: d0d124d5 44444444 t4: 00000be4 cir: 0 tir: 0
a5: 00010000 s5: 00000000 ffffffff t5: 00000000 vl: 80 vs: 00000000
a6: 84888000 s6: c0d54c00 66666555 t6: 00008000 vs_u: 00000000 00000000
a7: 00057689 s7: d6400028 77777777 t7: 00001678 vs_l: 00000000 00000000
global_int_enab: 00 local_int_enab: ff int_mode: 00 target_CPU: 0 ION: 0
```

. . . continued on next page . . .

...continued from previous page

As can be seen by the example, the register dump consists of the dump of all register file locations for the ASP, the pc, the interrupt pc and all process status info. It also includes vector merge info, which can be used to determine if the head was in the middle of a vector operation when the fault occured.

As a helpful hint, if the PC value of the suspected head is 00000000, this may mean that the head failed while attempting a cold start. While this will not likely occur during routine operation, it quite frequently will occur during mminit or sysreset operations. If the head was attempting a coldstart the contents of register t5 will generally contain the contents of 1014. This is the physical start of cold start. If this occurs the ipc will indicate the actual point at which the cpu halted in cold start. A partial dump of cold start can be seen in the C2 CPU Debug Manual on page 2-6. After finding the actual failure, the system prints can be used to locate the full path and isolate the area of the fault.

A similar path can be followed if the system fails to perform a sysreset -l2. The ipc again will contain the address of the code being performed and can be examined further on page 2-1 of the C2 CPU Debug Manual.

If the contents of the IPC is BFD, this will always indicate the head that has stopped. The location BFD indicates a complete shutdown of a head. This will frequently occur during a diagnostic failure. This can be a quick way of spotting the offending head.

The psw (processor Status Word) and ccr (Cpu Control Register) can be helpful in determining, quickly, what the CPU was involved in. Both of these registers are explained fully in chapter 7 of the CPU Debug Manual and so will not be repeated here.

The cir and tir have been mentioned in part II (Scan-002 pg.1) of this discussion and so it is not necessary to repeat this explanation.

This has been a cursory overview of the contents of this valuable display. It is only intended to give you some valuable clues and point you in the right direction. With use and reading of the architecture manual, it will soon be your most useful troubleshooting tool and best friend during a system crash.

For more information on this utility please contact the TAC. But remember, please start all troubleshooting efforts with a cpureg; You'll not be sorry you did.

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-005
Date: February 26, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

I'd like to take this opportunity to sum up some of the topics that have been discussed up to this point. We have discussed some of the more useful scan tools, displays that can and should be used after any crash to get a good picture of what occurred and allow the FE to make some intelligent decisions on how to proceed after a crash or hang.

The utilities mcm_func, mm and cpureg are very valuable and should always be among the first utilities used. I would recommend that the following order be used directly following a crash or hang:

1) cpureg > filename (This should always be first, because as previously discussed, the pc can advance under hwdump.)
2) hwdump (this is necessary in case something goes wrong, so that you do have some information to fall back on.)
3) mcm_func (To get the head and possible memory errors)
4) mm (To determine what the head was doing at the time of the crash.

There are far more scan utilities and they will be discussed in future releases under this topic, but using the utilities discussed, so far, provides valuable data for making troubleshooting decisions. You are not going to want to run all of the scan scripts, so these basics will help in determining which ones need to be run next and in what order. This is why a script can't be written to run the proper utilities for you, as decisions must be made based on the data received after each script is run.

Never forget about the foreplane connectors in your thought process. These connectors include most head internal busses and thus are a physical part of the boards that they connect. Using prints it is sometimes easy to spot potential problems with these connectors, but this is the exception rather than the rule. It is always a good idea to mark the connectors and return them to their starting location after moving for troubleshooting. There are different pins used at each station and so it is possible for a foreplane connector to work in one location and not another.

If you cannot initialize memory from one head and it is a multi head system try forcing an mminit from another head. The head performing the initialization is always the first head listed in the file /mnt/boot_db under the entry "procnum". Generally this will be the first physical head such as A. This is a great way of determining if the problem is localized or a general system problem. The procedure to force mminit on a head has been discussed in previous allfe mail.

. . . **continued on next page** . . .

. . .continued from previous page

As discussed in previous allfe mail, if you have an application related problem it is possible to force this on a single head to check for hardware problems. This can be accomplished by the online utilities cpuconf <-e,-d> and mpa -c<head number 0-3>.

Although the above suggestions do not involve the scan utilities directly, they are valuable tools and should be considered in conjunction with the scan scripts.

Always remember that the scan utilities are just as valuable after a diagnostic failure, an mminit failure, or a sysreset failure. Also remember that after any system reset, the information is lost, so it is important that customers understand the importance of not immediately rebooting. Even if the FE or TAC is not available, it is very easy to leave instructions for the customer on performing a cpureg and hwdump. It is even possible to write a script so that the customer need only enter a simple command at the spu.

Most of the problems that we see are of an intermittent nature, so it is extremely important that we get all of the information quickly and on the very first failure. There are very few "glitches" or transients that will crash a C2, so all crashes should be considered real.

As with anything, the scan utilities are only as valuable as the level of understanding permits. It will prove quite helpful by using the Architecture Manual and the Theory of Operation when initially using the scan utilities. The more you understand about the system operation, the more helpful the scan utilities will be.

As usual, if you have any questions or problems concerning scan utilities, please contact the TAC for assistance.

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-006
Date: February 28, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

The utility fu_mmqueue is designed to examine the most recent memory return que contents. As we all know the memory return que is used to match memory returned data with the module that requested the data. This script will display the module making the most recent requests, size of the request and to which ASP register it is destined for. It will also indicate whether the request has been completed or was still waiting

EXAMPLE

(iscn fu_mmqueue)

```
MMQUEUE      [Head 1]
NUM                          POP        part qinh  MOx_EN  MEx_EN  CU
ITEMS _qpush empty full      q qq qqq      2  sxv  3 2 1 0 3 2 1 0 EN
----- ------ ----- ----      -------    ---- ----  ------- ------- --
    0      1     1    0      0  0  0       0    0  0 0 0 1 0 0 0 1  0

o loc or bd qa0 qa1       req                          full op      mmq cu sxv  inh
k ptr  # 01 <<3 <<3       01* sz rot unit  rtn          upd rdy vm act en dual ust
- ------ -- --- -------- --- -- --- ----- ------ ---- --- -- --- -- ---- ---
  wp-> 5 (next position to be written)
N qw-> 5 00 0a0 000360a0 11   3   0 2 IP   05 A5      1   1  1   0  0    0   0
Y rp-> 5 00 030 00036030 00   3   0 2 IP   02 A2      1   1  1   0  0    0   0
N qr-> 5 00 030 00036030 00   3   0 2 IP   02 A2      1   1  1   0  0    0   0
N      4 00 098 00036098 00   3   0 2 IP   10 S0      1   1  1   0  0    0   0
N      3 00 090 00036090 00   3   0 2 IP   10 S0      1   1  1   0  0    0   0
N      2 00 088 00036088 00   3   0 2 IP   10 S0      1   1  1   0  0    0   0
N      1 00 a08 00000a00 10   0   6 0 ASc  10 S0      0   1  1   0  0    0   0
N      0 00 080 00036080 00   3   0 2 IP   01 A1      1   1  1   0  0    0   0
N      f 00 078 00036078 00   3   0 2 IP   01 A1      1   1  1   0  0    0   0
N      e 00 070 00036070 00   3   0 2 IP   01 A1      1   1  1   0  0    0   0
N      d 00 068 00036068 00   3   0 2 IP   01 A1      1   1  1   0  0    0   0
N      c 00 060 00036060 00   3   0 2 IP   00 A0      1   1  1   0  0    0   0
N      b 00 058 00036058 00   3   0 2 IP   06 A6      1   1  1   0  0    0   0
N      a 00 000 00010000 01   0   1 3 ASn  08 T0      0   1  1   0  0    0   0
N      9 00 050 00036050 00   3   0 2 IP   06 A6      1   1  1   0  0    0   0
N      8 00 048 00036048 00   3   0 2 IP   07 A7      1   1  1   0  0    0   0
N      7 00 040 00036040 00   3   0 2 IP   02 A2      1   1  1   0  0    0   0
N      6 00 038 00036038 00   3   0 2 IP   02 A2      1   1  1   0  0    0   0
```

As can be seen from the example, this display contains a lot about the destination of data and the device to receive it. As you know from reading the Architecture manual the example is nothing but a dump of the memory return queue, located on the SFU/EFU. This display can be viewed by entering dump_mmq after entering this script.

. . . continued on next page . . .

. . . continued from previous page

The MOx_EN and MEx_EN indicates the memory pair involved in the data transfer. This can be helpful when a memory problem is suspected. This is a 4 bit field and displays the result of a Read Ack assertion.

The column labeled ok indicates whether the request has been satisfied. If a Y is in this column, this indicates the point at which the system failed and that this request was never satisfied. If a Y exists or for the first request remaining to be satisfied, the component that made the request is indicated under the column "unit". In addition to this the "rtn" column indicates the register on the ASP that will be used to stage the data before transfer to the requesting device.

A quick check of the return que can be obtained by examining the "empty" column at the very top of the display. If this entry is a 1, it means that the return que is empty and there will be no useful info in this display.

The "rot" column indicates how the data is to be rotated. In other words which byte was actually requested. Again this is only useful if suspecting a memory problem. And the "sz" field indicates the size of the transfer.

The "cu en" when asserted indicates data from the CPX/CUE. This can be helpful in tracking a parity error.

As you can see a lot of useful information can be obtained from this utility. It is included in the hardware dump, so it isn't necessary to run it independently, unless you are just trying to run some quick tests. This utility would always be the fifth utility examined in any debug operation, preceeded by:

1) cpureg
2) hwdump
3) mm
4) mcm_func

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-007
Date: February 28, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

The scan script **hard** is useful in determining the hard error status of all boards after a system crash. The script will display the hard error state for the ASP, SFU/EFU, EDC/DCU, VPC, VPD, CPX/CUO/CUE, PIA and the first memory pair.

Most of the fields in this display are self explanatory as can be seen in the example below. The SFU and EFU display different fields and this should be considered when examining this display. It is normal for the SFU fields to contain a 7 in the fields par_b0_ok, par_b1_ok, par_e0_ok and par_e1_ok. These are the Dcache even and odd tags.

For the EFU qpar_err is nothing more than a logical OR of the other error fields. So if nothing is set in this field then no error exists with the EFU.

EXAMPLE
(hard)

```
ASP: ucode_err wcs_par_err sram_par_err sram_synd cbus_synd dbus_synd
           0           0           0        7f        00        00
IPP: mbus_par_err icache_val_err la_val_err icache_par_err la_par_err
            00           0           0          0000           0
EFU: qpar_err ri_adr_pe ri_data_pe ram_par_err _upd_par_ok
           0        3f        36          0          0
DCU: hard_error pte_par_err val_par_err tag_par_err fu_par_err
           0           0           0           0           0
VPC: hard_err vm_par_err softerr ul_par_err ua_par_err um_par_err vd_par_err
           0           0       0           0           0           0         0
VPD: hard_err vre_par_err vro_par_err is_par_err os_par_err vm_par_err
           0           0           0          0          0          0
CPX: hard_err cregerr (synd) _par_adr pcmmerr tiperr tiaerr pcmrerr pcmaerr
           0       0     00      07b       0       0       0       0       0
     start_err mx_cyl mx_wrda mx_wrpar pcmstart rmstart tistart mx_uadr mx_ad
           0       0    0000         3        0       0       0     000 00000
ME0: hard_err par_err ecc_err multi_bit
           0      00       00        00
MO0: hard_err par_err ecc_err multi_bit
           0      00       00        00
PIA: hard_error ia_harderr re_byte_perr rdata_hi rdata_lo r_par (pcm) (idle)
             0          0           00 0000000e 00000018     f7      0       0
```

The ucode field for the ASP will contain a 1 after any microcode related error, such as an ASP104 error. This field indicates that the microcode encountered a problem and jumped to a micro instruction that caused a halt.

If a hard error is set for any board, it is extremely helpful to refer to the prints to get the flow thru the board and the location of parity generators and checkers. Knowledge of these will help in isolating the location of most hard errors.

Remember, just because a board has the hard error field set, does not mean that the problem is with this board. It is not at all uncommon for a parity check circuit to be located on one board and almost the entire data path located on other boards. But this error bit will give you an idea where the data path for the error is located.

After detecting a hard error on a board it will then be helpful to visit the iscan display for the board in question. These displays will give far more insight into the error that has been encountered. These will be discussed in future installments of the ISCAN discussion.

Unfortunately, only hard errors occuring on the first memory pair are logged in this utility. To verify all of memory it will be necessary to use mem_func.

The display for the PIA can be helpful with certain memory errors, as well as I/O or ebus errors.

In order to examine this diplay for each head it is necessary to change heads by use of the :900=n (where n=0,1,2,3) command.

After running iscn hard the display can be viewed by entry of the command **hard** at the prompt.

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-008
Date: February 28, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

The hang utility can be useful in examining the state of the CPU after a system hang has occurred. An example of a system hang during CPU diagnostics would be a subtest timeout. This is roughly equivalent to a total system hang while running OS.

The hang display can be viewed by running iscn hang under /hw/cputest at the spu. When the prompt is returned, it is necessary to enter the command "hang". This will produce the display that can be seen in the example below.

As can be seen, this display produces a lot of useful information. It is important to remember that this utility is of very little use after a crash and should only be examined after a system hang.

EXAMPLE

(hang)

```
          hang            Cpu  1

IPP:      (pc)      (ba)      (na)       ja as_pend vp_pend inst_rdy
     0003605e 0003605c 00036068 0003603c        0        0        1
           la la_maxed pur_state
     000360a0        1         1

ASP: ipc upc ui_valid ui_lvl_1 ip_inh mem_idle fu_busy vc_idle
     65f 660         f        0      1        1       0       1

SFU: vl       vs vg_busy1 vg_dbusy1 vg_vlc vg_dvlc vg_qmop vg_qsiz
     0e 00000004         0         0      d       e       15       2
     mmqempty     vl_rdy_as     vl_rdy_vc  vmq_ptr  vpq_ptr
            1             0             0        0        0

DCU: qreq qfault qlab_full qrd_cyc qwr_cyca qex_cyc      qla1
        0      0          0       1        0       1      000360a0

VPC: (disp) ip_disp (ep) ul_active_1 ul_uia ua_active ua_uia um_active um_uia
           0       0  1f5           0    069         0    028         0    030
     lbd_state dlp_state os_state_reg vmo_state_reg
             0         0            0            0
```

IPP: ba- Branch Address
Ja- Jump Address
as_pend- When an instruction is to be dispatched to ASP and VPC and the VPC has been dispatched.
vp_pend- Opposite of as_pend
inst_rdy- Indicates the IPP has cracked the next instruction
la- The address of the IPP lookahead being requested at time of hang.
pur_state- Icache purge controller state. Determines state of validity columns. See iscn manual page hang-2 for meaning of contents.

ASP: ipc- Interrupt program counter
upc- ipc+1
ui_valid- Indicates valid microinstruction if contents is f.
ip_inh- Inhibits IPP memory requests. This happens during micro fault conditions.
mem_idle- Memory queue is empty and no requests pending.
fu_busy- Asserted when any logical function unit is active.
vc_idle- Indicates vector unit is idle.

SFU: vl- vector length
vs- vector stride
vg_busy1- VAG active with vector request
vg_dbusy1- VAG active with dual vector request
vg_vlc- VAG vector length counter
vg_dvlc- VAG vector length counter in dual
vg_siz- Size of vector data request.0=byte,1=half word,2=word,3=long
vl_rdy_as- Indicates new vl. Inhibits ASP loading vl again.
vl_rdy_vc- VPC available for new vl.

DCU: qreq- Memory request in memory controller.
qfault- Page fault, pte miss etc.
qlab_full- Lookaside buffer is full and will supply next memory request.
qrd_cyc- Current memory cycle is read.
qwr_cyca- Current memory cycle is write
qex_cyc- Current memory cycle is read for IPP.

VPC: ip_disp- IPP ready to transfer vector instruction
ep- Entry point address from IPP
ul_active_1- Means load/store controller is active.
ul_uia- Micro address of the load/store controller.
ua_active- ALU micro controller is active
um_active- Multiplier microcontroller is active.
lbd_state- State of load back door controller.
see iscn manual page hang-4 for status of this field.
os_state_reg- State of the output staging controller.
vmo_state_reg- VM output stage controller.

By examining the individual fields above it is possible to obtain a very good overview of the functional boundaries between the individual components in a head.

CONTROL X

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-009
Date: February 28, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

The Iscn utility, pi2_func is very helpful in determining the cause of I/O related data errors. These errors would include ebus parity errors as well as 600 series errors.

It is important to remember that this utility cannot be used with a standard PIA and therefore can only be used on the wide body systems.

This display is very similar to the options we have with the mcm displays, in that we can examine things such as the arbq which is similar to the win que. This arbq display will give us the last 5 masters which controlled the ebus. These possible masters include the PIA and SPU.

The ebus display will will list the CCU involved as the Pbus master. The display is pretty much self explanatory. For more info on individual fields please refer to page pi2_func-2 in the Scan Script Manual.

EXAMPLE
(pi2_func)

"ebus" screen

```
m   ebus master    request     PBUS master    PBUS cycle    sp4 read
v          10           1               4             0           0
s  PIO               sp4          ccu0            idle         no
m
v  Memory   INT   memory bd   Bank   Row    ram_addr    ebus_addr    maddr
s           1       1100      2      2       422f6      7ffffff8      0
m  8 way           M80/M00
v
s  pstate   estate  io_transfer  last_req  cycle_ack
m      0       80             0         1          1
v   idle     idle     no             read       yes
s
m  hard_err    md_perr    pcm_pe   ia_harderr    arb_err
v         1          1         0            0          0
s
m  soft_err    ea_soft    npm_berr   ill_berr  wpar_berr  ccu_hard
v         1          0          0          0          1        00
s                                                          none
```

This display can be obtained by entering "e ebus" at the iscn prompt. This display can be very helpful in following a data transfer from memory all the way to the pbus. The display lists hard and soft errors and even gives a memory origination address. This display also tells whether the error encountered was hard or soft and details the type of error.

Remember; To select different pi2's it is necessary to change by using set_pi(slot #). The default is slot 0.

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-010
Date: February 28, 1990
Subject: Iscan Utilities
Submitted By: Al Haddix

Most of the remaining Scan Utilities are associated with a specific board and are designed to supply specific information for given boards when the problem has been isolated to a specific area. These displays are especially useful when attempting to track an elusive backplane problem.

In the case of asp_func; this utility is used to examine data flow within the head and between a head and memory. The individual diplays include screens for the dfw gate array, the register file, hazard bits and the control stores.

A lot of the information will be of little use when attempting to isolate the area of the problem, but will be helpful in later stages. The exception to this, is the hazard bits. This display can give needed info as to what the head was doing at the time of the crash. This display can be viewed by entering haz at the iscn prompt. From reading the architecture manual it can be determined that the hazard bits act as a hardware version of a lock bit and is active when any registers are in active use. This is to prevent 2 simultaneous operations to the same register.

The registers can be displayed by entering rega, regs, or regt, but this info is identical to that obtained by a cpureg or a hwdump. It is however possible to write to these registers and can be useful in testing specific data paths to the register file. The format of the write is wreg <select>,<data> where the select is:

```
00-07 = A0-A7 respectively
08-0F = T0-T7 respectively
   10 = S0
   11 = S0 upper
   12 = S1
   13 = S1 upper
        .
        .
        .
   1F = S7 upper
```

By the use of the wreg it is now possible to write any data pattern into the register file.

For more information on the other functions, please read the asp_func entries in the Scan Script Manual.

There is little functionality with the utility cpx_func except the dumping of the communication registers, which is accomplished by the utility commreg. So... I will spend no time on this. I will however discuss the cuo_func and cue_func utilities, under this topic.

These two utilities can only be used with widebody systems, but supply a wealth of information on errors reported by the CUE and CUO.

The CUE monitors several error nets directly can supply much needed information about these errors when they occur. Anytime that an hard error is reported by the CUE it is possible to examine it by use of the Iscn utility cue_func. After returning the scn prompt it is only necessary to enter "cue_err".

EXAMPLE
"cue_err"

```
CUE: hard_err C.IO_MS_ADDR_ERR ti_err C.PCM_ERR ARB.ERR_START C.RM_64K_ADDR_ERR
          1                1       0          0                0                0
   FUNCTION:     io_err     ti_err    pcm_err    arb_err     rm_err
```

Additional information about each one of these errors can be obtained by simply entering the error name after the prompt. In the above example, we see that we have encountered an io_err. By now entering "io_err" after the prompt a more thorough explanation of this error can be obtained.

As a note, if the hard_err field is not set to one, then no other field entries are valid. All optional displays that can be obtained are listed in the example after FUNCTION. The entries on the top line is the actual net that is being monitored for this error.

In addition, we also have a cuo_func utility which helps in monitoring still other error conditions. In order to display these errors it is necessary to enter cuo_err after the iscn prompt. An example of this display follows:

Output of "cuo_err"

```
CUO: hard_err pcm_mem_err pcm_ram_err comreg_err
          1           1           0           0
FUNCTION:          mem_err     ram_err     comreg
```

As can be seen from the example above; the format is identical to that of cue_func, with the additional error displays functioning in the same manner. Again all information is invalid unless a "1" exists in the hard_err field. As can be seen in the example above, a pcm_mem_err exists and can be examined further by entering "mem_err" at the prompt.

These displays are extremely easy to understand and supply information that cannot be obtained with any other utility. These error displays can be extremely helpful in finding specific defective nets.

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-012
Date: April 10, 1990
Subject: Scan Utilities
Submitted By: Al Haddix

The utility mmap although not strictly an iscn utility, is quite useful in running down the physical location of memory errors encountered by iscn and system hard errors.

The utility mmap was released to the field with the 3.2.1 release of diagnostics. It is located in the file /bin and can be entered by entering "mmap" at the spu prompt. This will result in the display below:

EXAMPLE:
(mmap)

```
Memory configuration
size          = 256K
Rows per bank = 2
Board pairs   = 1
Interleave    = 8

Physical address = 0x00000000

Board select = me0
Bank select  = 0x0
Row select   = 0x0
Ram address  = 0x000000

ADDR<31..3>  = 0x00000000
MADDR<2..0>  = 0x0
ROW_SEL<1>   = 0x0
```

This utility is extremely easy to use and requires, only, that you know the physical memory address to be decoded and the number of memory board pairs.

With this info then after entering "mmap" is is only necessary to move to the entry at the top, labeled board pairs and overwrite the 1 with the proper value. It is possible to move around in the display by the use of the arrow keys.

After changing the number of pairs it will be necessary to move to the entry labeled physical address and enter the address for decoding. As the address is entered it will be possible to view the change in the physical location below, under the labels of board select, bank select, row select and ram address.

When the physical address is fully entered the full physical location of the error will be displayed.

This utility is great to use when another iscn utility points you to a particular area of memory, or when a system hard error displays a physical address. It is also handy to use when verifying memory locations in the Softlog.

CONVEX

Field Support Tech Tip

Product: C-1
Tech Tip Number: Scan-013
Date: May 10, 1990
Subject: Iscan Utility 'mbus'
Submitted By: Al Haddix

We have a new version of hardware dump that is designed to examine failure information after receiving an mbus parity error. This new utility is released in version 3.3 of the diag data base and is called mbus. This utility is executed exactly like a hardware dump and is very useful in getting the specific information necessary to locate the source of the mbus parity error.

To take the mbus dump proceed as follows:

1) cd /hw/cputest
2) mbus <filename> - This will take approximately half of the time that the hwdump will normally take on a given system.

Even if the system does not have mbus loaded it is still possible to use the mbus file as a guide to the proper way to proceed when chasing the mbus parity error.

The script proceeds as follows:

1) Take date and cop.out
2) Execute hard_logger
3) Executes iscn to take necessary dumps
4) Determines configuration and executes halt disable so clocks will be stopped in the event of a hard error.
5) Determines CPU that pulled error and dumps state.
6) Dumps scalar registers using asp_func; rega, regs and regt.
7) Dumps DFW gate array with asp_func; ga. This is done to locate bad data in the DFW ureg.
8) Dumps the memory return queue with fu_mmqueue; dump_mmqueue. This is done to find which mcm's were providing data to the head at the time of the crash.
9) Set halt disable for all existing mcm's
10) Dump read queue with mcm_func; dump_rd_q. This is done to examine the data and parity coming from memory at the time of the failure. ****NOTE**** Parity for the rd_que is odd
11) Dump crossbar read and write pointers with mcm_func; xbar_ptr. This is of no use while troubleshooting this problem, but will assist in component level repair.
12) Examine Icache for parity errors with icache -e.

This will supply all data necessary to locate the cause of the mbus parity error. Again, even if the mbus utility does not exist, it is still possible to follow this procedure and obtain the information.

The halt disable referred to in the procedure refers to the iscn utility halt_off. This can be executed by running iscn halt_of and assigning the head with the register :900 and executing "halt_off". This command will stop clocks on all effected boards of a head when a hard error is encountered.

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-014
Date: May 10, 1986
Subject: Iscan Utilities 'edc_func'
Submitted By: Al Haddix

The **edc_func** utility enables the examination of the tag rams and Dcache on a particular head. It is important to remember that the **rd_edc** function is identical to the dump created by entering dcache utility. The difference between these two (2) utilities is that the edc_func allows a specific address range to be dumped and the dcache utility does not. Because of the similarities between the two, both will be discussed here.

EXAMPLE
(edc_func)

```
Data Cache
Left Parity bit => Left data byte, Left validity bit => Left data byte.
ADDR   DATA      P TAG_A   P TAG_BC P TAG_DE P   VBS   VCS   VDS   VES   UPD
0200   00000000 f 0ad6b0 4 0ad6bc 4 0ad6bc 4   0 0   0 0   0 0   0 0   00
ODD    00000000 f 0ad6b0 4 0ad6bc 4 0ad6bc 4   0 0   0 0   0 0   0 0   00
0208   00000000 f 0ad6b0 4 0ad6bc 4 0ad6bc 4   0 0   0 0   0 0   0 0   00
ODD    00000000 f 0ad6b0 4 0ad6bc 4 0ad6bc 4   0 0   0 0   0 0   0 0   00
0210   50515253 9 0002c0 5 0002cc 5 0002cc 5   1 0   1 0   1 0   1 0   ff
ODD    54555657 6 0002c0 5 0002cc 5 0002cc 5   1 0   1 0   1 0   1 0   33
```

The utility edc_func can be entered with the command **iscn edc_func**. The above display can be achieved with **rd_edc <start addr>,<end addr>**. As usual the head can be specified with **:900=n** (where n = 0,1,2,3).

The ADDR field indicates the address of the cache, with the DATA being the contents at that address. The P columns, in all cases, indicate parity and is not rotated so the ms bit of parity indicate the ms byte of data. In all cases the parity used is odd.

The TAG_x columns indicate the contents of the Tag rams, located on the EDC. Every word in the Dcache has a tag that is 20 bit data. The trailing 0 in all Tags are used to make it easier to read parity. The Dcache maintains three (3) identical copies of TAGS that are used for other heads and the I/O system's use. All Tags are updated when data is written from the cache. The trailing c is used in all the Tags but A and is hard coded. This trailing c should be the only difference between all copies of the Tags. If a difference exists, this generally indicates a soft error has occurred within the gate array.

. . . continued on next page. . .

...continued from previous page

The columns labeled VxS are copies of the validity ram data plus respective syndrome bits. The first column under each indicates the validity of the data with one bit for each byte. All copies must be set to one for a byte to be valid, as indicated in the example. The S column is the syndrome bit for the validity rams. If any bits are set to one, this indcates a parity error in this copy of validity rams.

UPD is the contents of the Update Rams on the EFU. Both nibbles (4 bits) should be equal to one another.

As stated earlier, this information is identical to the information obtained from a dcache dump except that the rd_edc function has a start and end address option.

When taking a Dcache dump the proper command will be:

dcache -c <head> -dh -e

(where <head> = 0,1,2,3) and then redirected to a file. The -dh option converts to hex format. With 3.3 rev diag data base and above, there is a -e option for the dcache function. This -e only displays the dcache entry that is in error. This can be a great time saver, but 3.3 must be loaded to use this option.

This should enable much more thorough analysis, in the field, of the 201 and 904 error codes. As always, please address any questions to the TAC.

CONVEX

Field Support Tech Tip

Product: C-2
Tech Tip Number: Scan-015
Date: May 12, 1990
Subject: Iscan Utility 'efu_func'
Submitted By: Al Haddix

The purpose of the Remote Invalidation circuit is to monitor remote processer writes and purge entries of the local cache after a remote write to memory that corresponds to entries in local cache. The Remote Invalidate system examines writes from other CPU's and I/O and compares these to the copies it maintains in the TAG Rams as explained in the discussion on **edc_func** (TT Scan-014). If the write is contained in the local cache then remote invalidate signals and validity ram addresses are dispatched to the EDC to invalidate this entry.

In order to accomplish this remote invalidation port E addresses are bussed to all CPU's and the other ports are daisy chained between heads in the following sequence:

A -> C -> D -> B -> A

Three addresses are transferred during one system clock cycle. Because of this a special "triple" clock exists to drive the RITA system. Each clock cycle is labeled PHI2, PHI1 or PHI0 with PHI2 being the first cycle.

The iscn utility **efu_func** can be quite helpful in isolating problems within the RITA logic. The two commands under this utility enable the RITA gate arrays and the output staging registers to be examined. Although this utility is utilized extensively by development, it can also be very important when troubleshooting net problems in this circuit.

This logic can be tested by the use of the diagnostic misc4000 or the iscn utility **ri_test**. These tests are identical in functionality with ri_test allowing interaction and control.

The command **dump_ritas <head #>** allows display of the scan state of the RITA Gate Arrays. The fields of this dump are defined on page 2 of this Tech Tip.

. . . continued on following page . . .

. . . continued from previous page

RITA Gate Array Dump Fields

enb_wr_on_dmode - Allows tag rams to be written during DMODE HOLD clocks.
DMODE = 1 SC_CTL = 3

enb_bad_phi - RITA keeps copies of the PHI state and if they stop matching or multiple clock phases are asserted simultaneously then a parity error will occur.

enb_ram_par - Enables parity error detection on tag rams.

enb_ri_par - Enables parity errors on the daisy chained bus

enb_<b,c,d or e> - Enables remote invalidates on the designated port.
Port a is default - others must be selected.

qpar_err - General parity error bit. Set when any others are set.

ram_par_err - A parity error occurred while reading the tag rams.

ri_adr_pe - A parity error occured on bit <11...3> of the daisy chained bus.

ri_dat_pe - A parity error occurred on bit <31...12> of the daisy chained bus.

. . . continued on following page . . .

. . . **continued from previous page**

The command **dump_daisy** will dump the input and output staging registers for odd and even Remote Invalidates. When the dump occurs the Ritas will be stopped at PHI2. Below is an example of the output and a table to display register position of the data:

EXAMPLE

```
Remote Invalidation Daisy Chain Bus
RI_PAR<3> (odd par) => RI<31..24>
RI_PAR<2> (odd par) => RI<23..16>
RI_PAR<1> (even par) => RI<15..12>, WRITE_REQ
RI_PAR<0> (odd par) => RI<11..3>
```

CPU STAGE	RIO << 3	WRT RQ0	RIO PAR	RI1 << 3	WRT RQ1	RI1 PAR
A RITA	01314330	0	3	01314330	0	3
EXTERN	009ac330	0	d	009ac330	0	d
C RITA	01c7c330	0	1	01c7c330	0	1
EXTERN	00044330	0	b	00044330	0	b
D RITA	009ac330	0	d	009ac330	0	d
EXTERN	01314330	0	3	01314330	0	3
B RITA	00044330	0	b	00044330	0	b
EXTERN	01c7c330	0	1	0c17c330	0	1

NOTE

Notice that the example is in the order of the daisy chain sequence.

TABLE

	DAISY CHAIN ADDRESS TRANSFER							
TIME	CPU A		CPU B		CPU C		CPU D	
	IN	OUT	IN	OUT	IN	OUT	IN	OUT
PHI2	D	B	C	D	B	A	A	C
PHI1	B	A	D	B	A	C	C	D
PHI0	C	D	A	C	D	B	B	A

CONVEX

Field Support Tech Tip

Product: C-1
Tech Tip Number: Scan-016
Date: June 19, 1990
Subject: Iscan Utility - ipp_func
Submitted By: Al Haddix

ipp_func

The scan utility **ipp_func** is quite useful for examining memory data registers, the Icache and analyzing ipp related hangs. There are four (4) usable commands in this utility: **m_dat, rd_ic, wr_ic and ipp_hang.** The **m_dat** command is used to examine memory data arriving in the pre-crack portion of the ipp. The **rd_ic** and **wr_ic** commands are used to display and write to designated areas of the Icache. The **ipp_hang** command enables the user to examine the state of the ipp after an apparent hang.

The **rd_ic** and **wr_ic** commands can be used to view and write to the Icache. In both cases the display can be controlled with the start and end address. The command **wr_ic** will also allow writing to the memory data registers. Below is the example of the display from **rd_ic:**

EXAMPLE
(rd_ic)

```
          inst_0    inst_1    inst_2    inst_3          err  tag    stat a b
800020b8 = fffc 1 0 3638 2 0 fffc 1 0 1188 2 0  HIT  0000 40001 0002 0 1
                    -- Lookahead --               HIT  0    40001      0 1
800020c0 = 0000 2 0 3640 3 0 8008 1 0 1168 3 0  HIT  0000 40001 0002 0 1
                    -- Lookahead --               HIT  0    40001      0 1
800020c8 = 3240 3 0 8006 1 0 1168 3 0 5101 1 0  HIT  0000 40001 0002 0 1
                    -- Lookahead --               HIT  0    40001      0 1
800020d0 = 1981 2 0 0003 2 0 3638 2 0 fff4 1 0  HIT  0000 40001 0002 0 1
                    -- Lookahead --               HIT  0    40001      0 1
800020d8 = 5108 1 0 1488 2 0 0078 3 0 1489 2 0  HIT  0000 40001 0002 0 1
                    -- Lookahead --               HIT  0    40001      0 1
800020e0 = 0098 2 0 3638 2 0 fff8 1 0 3639 2 0  HIT  0000 40001 0002 0 1
                    -- Lookahead --               HIT  0    40001      0 1
800020e8 = fff0 1 0 3341 3 0 8014 1 0 a5a8 1 0  HIT  0000 40001 0002 0 1
                    -- Lookahead --               HIT  0    40001      0 1
800020f0 = 2a3d 2 0 fff8 1 0 3368 3 0 800e 1 0  MISS 0000 40001 0002 0 1
                    -- Lookahead --               MISS 0    40001      0 1
800020f8 = 14a0 2 0 3738 2 0 ffe0 1 0 3368 3 0  MISS 0000 40001 0002 0 1
                    -- Lookahead --               MISS 0    40001      0 1
```

The first column is the logical address. The utility hwdump will dump this info 5 locations either side of the pc in cpureg. The column labeled HIT or MISS indicates if the data at that address is located in the cache. A HIT would indicate that the data was in the Icache. The err column indicates a parity error in each of the 4 halfwords for that location.

...continued from previous page

The stat column indicates the purge state of that entry and displays which validity column is written during writes, which column is used on reads and which column is purged. This column refer to the columns a and b which are the Icache validity columns. The state codes are defined below:

```
0000 Both columns purged and unused
0001 Using A and B is marked for purge
0010 Using B and A is marked for purge
0101 Using A and purging B
0110 Using B and purging A
1101 A waiting for purge and purging B
1110 B waiting for purge and purging A
```

The command **m_dat** displays the memory data and parity transferred form the ASP. The data longword is displayed as four halfwords plus parity. This data can be compared to the data displayed by **ga** in **asp_func.**

The command **ipp_hang** provides the scan fields necessary to examine the state of instruction dispatching within the IPP. These fields will provide the state of the icache, instructions and individual scalar and vector components.

TECHNICAL BULLETIN

CONVEX

TECHNICAL BULLETINS

Fourth Edition

TECHNICAL BULLETIN

EDITION INFORMATION

First Edition	Released July, 1991
Second Edition	Released November, 1991
Third Edition	Released March, 1992
Fourth Edition	Released January, 1993