TO: MSPM Distribution
FROM: D. R. Widrig
SUBJECT: BF.20.12
DATE: 12/01/67

Minor editorial changes have been made in the interest of greater clarity. Also, the "compressed status word" description has been expanded to more fully describe individual status bits.

Published: 12/01/67
(Supersedes: BF.20.12 07/19/67)

Identification

GIM - Interrupt Handling and Status Requests
D. R. Widrig and S. D. Dunten

Purpose

This section is part 4 of the complete description of the GIM: see BF.20.02.

Interrupt Handling - gioc_stat$int

Upon receiving an interrupt from a GIOC, the Multics Interrupt Interceptor will mask all lower priority interrupts and call the GIOC Interface Module's interrupt handler with the following call:

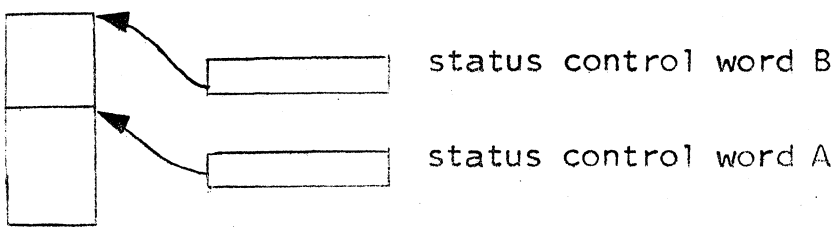
```
call gioc_stat$int (giocno, statno, timep)
```

where the arguments are declared as follows:

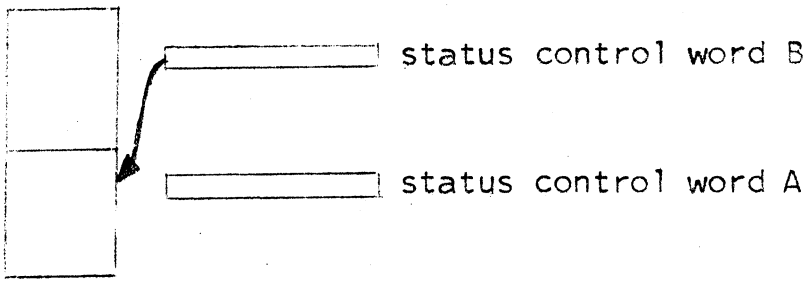
```
(giocno                 /* GIOC number causing interrupt */
statno) fixed bin(17)   /* status channel causing interrupt */
timep ptr               /* pointer to calendar-time of
                           interrupt */
```

The GIM performs very few actions on interrupts in accordance with a policy of swift response to interrupts as outlined in BK.0, MSPM. Pointers to the GIOC base and the appropriate status channel's LCT are obtained via a call to check$statusp. Errors returned from this routine indicates probable machine malfunctions. Malfunctions are not dealt with in the GIM at the current time so a return is made upon detection of errors. A short discussion of the GIM's hardware status queue discipline is now in order. Thorough knowledge of the GIOC handling of Status Control Words (SCWs) is assumed.

The hardware queue for a given status queue handled by the GIM is, in reality, two status sub-queues. Each sub-queue is used by a single SCW. Thus, a particular hardware status queue may appear as follows:

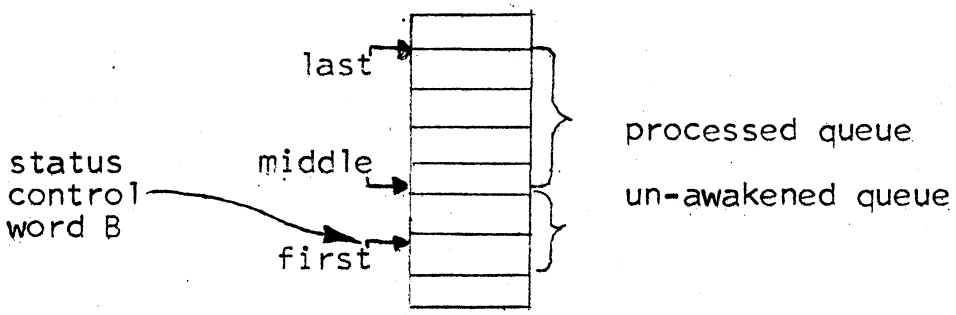


After exhausting the queue controlled by SCWB, the GIOC exchanges SCWB and SCWA. Thus, if the prime queue is exhausted, one might expect to see the following arrangement:



To put it another way, the current upper bound of the queue is always obtained by inspecting SCWB. The role of SCWA is discussed later in Moving of Hardware Status Words.

One can consider the hardware status queues to be one logical queue as follows:



The "un-awakened" queue represents status words not yet processed by the GIM. As such, the device manager processes interested in these status words have yet to be notified. The "processed" queue represents status words that have been processed by the GIM and have been brought to the attention of the appropriate device manager process.

The GIM's task upon being called by the Interrupt Interceptor is to awaken all device manager processes associated with status words in the "unawakened" queue. Clearly, if the "middle" pointer and the "first" pointer are identical, all status words have been processed. If "first" is less than "middle", the queue has wrapped around. To simplify processing, the upper bound is set to the end of the queue for a wrap-around case. Otherwise, the upper bound is set to contain the area between "middle" and "first".

Having established the boundaries of the status words to be inspected, the following actions are performed for each status word. First, the time of the interrupt is stored in a software analog to the hardware status storage. In effect, the software augments the hardware and simulates the effect of the storing of the time of an interrupt as well as the status storing. Having stored the time, the channel number stored in the status word is extracted and tested. If the channel is a user's channel, the Channel Assignment Table (CAT) is referenced to obtain the device index of the charnel. A call to dstm$set_dev_signal (see BQ.6.01) is made to wake up the device manager process. Upon being awakened, the device manager process should call the GIM for status information via the request$status call.

If the channel is a connect channel, a call to the GIM entry connect$int is made so as to restart the channel should there be some more CCWs to process.

In a manner similar to the queueing strategy discussed in List or Connect Channel Activation, the "middle" pointer is moved up to the end of the current "un-awakened" queue (thus expanding the "processed" queue) unless the end pointer is off the end of the queue. If the end pointer has run off the end, the "middle" pointer is reset to the base of the queue. The entire loop described above is then repeated until there are no new status words to be processed. The routine then returns to the Multics Interrupt Interceptor.

Inquiring as to Device Status - request$status

Whenever a DIM wishes to discover the status of its associated lists and retrieve hardware status relevant to the active device, the following call is made:

```
call request$status (device_index, cur_stat, rtn_stat,
     [stats])
```

where the arguments are defined as follows:

```
device_index fixed bin (17)  /* user device tag */
cur_stat bit (1)             /* ON if current status
                                desired */
rtn_stat bit (36)            /* standard GIM error return
                                word */
```

The optional argument(s), "stats", is a structure declared as follows:

```
dcl 1 stats,                          /* status frame */
    2 filled bit (1),                 /* ON if frame has data in
                                         it */
    2 active bit (1),                 /* ON if channel is active */
    2 status_waiting bit (1),         /* ON if more status waiting */
    2 started bit (1)                 /* ON if channel has started */
    2 int_id bit (24),                /* id list for this frame */
    2 int_idx fixed bin (12),         /* index of item in list */
    2 tally fixed bin (12),           /* current DCW tally, if
                                         applicable */
    2 time bit (52),                  /* time of interrupt or
                                         status */
    2 stat_length fixed bin (17), /* length of status array */
    2 stat (stat_length) fixed bin (24); /* breakdown of
                                            status */
```

The remainder of this section discusses the setting of every item within a status frame and illustrates possible errors encountered by the GIM.

The GIM begins processing a request$status call by calling a general utility, check$device_index, to verify the user device tag, "device_index". Possible errors from check$device_index include an illegal device index, "baddev", and no Logical Channel Table (LCT) associated with this device index, "lctnf".

Having validated the user device tag and having obtained a pointer to the caller's LCT, the GIM calls gioc_stat$move to move all hardware status relevant to this user into the user's work areas. Specifically, all relevant hardware status data is moved into the free storage area within the user's LCT. (See the later section, Moving of Hardware Status Words for further details of the status moving mechanism.)

After moving the status data, a call to getargs will collect pointers to the caller's status frames, "stats". If there are no status frames to be filled, i.e. the caller supplied none, request$status immediately returns. Note that the main effect of a call with no supplied status frames is to simply collect the status for later use. The status is not lost.

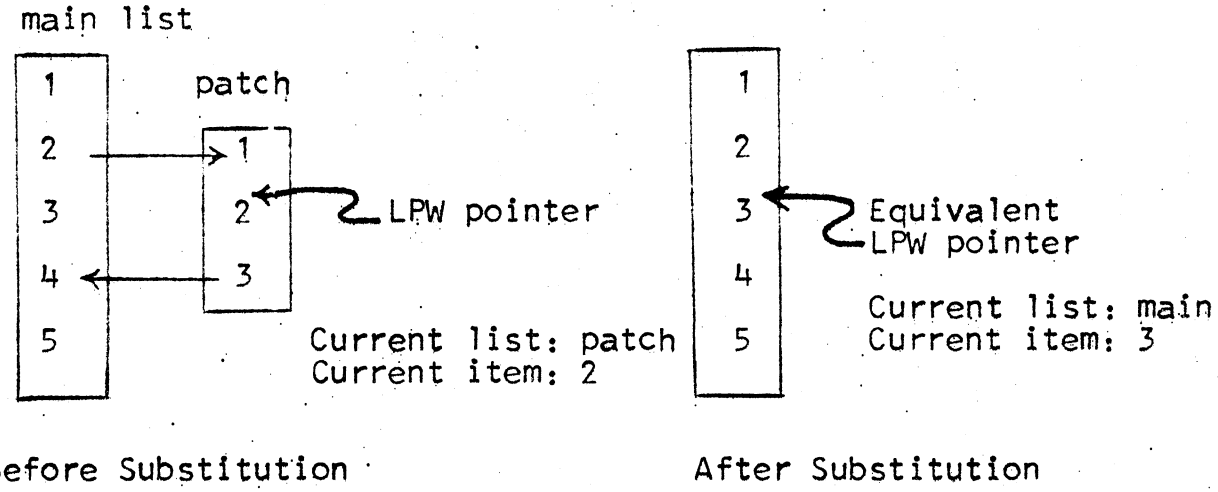
Assuming that the caller did supply one or more status frames, the GIM now proceeds to fill them. A call to check$gioc and lpw$active will collect a pointer to the GIOC mailbox associated with the caller and determine current channel activity. Possible errors from the above two calls include illegal GIOC number, "badcall", and GIOC not available, "giocnf".

A check of "cur_stat" is now made in order to determine the order of status being returned. If "cur_stat" is ON, the caller expects to receive the current list status in the first (and possibly, only) status frame supplied in the calling sequence. The following discussion assumes that "cur_stat" is ON.

For active lists, the GIM must construct a list status which is relevant to the actual GIOC processing of the caller's DCW lists. For an active list, the GIM collects the current LPW and DCW mailboxes via calls to double$load, a small EPLBSA routine which gets mailbox areas via double-word instructions. Double-word instructions are necessary to insure the consistency of the data gotten from the mailboxes. The LPW mailbox is examined and the equivalent list id and item index derived by a call to lpw$fnd. The only error possibility is a system or machine error, "syserr". Lpw$fnd also determines whether the LPW has moved since the connect$list call by matching the current LPW with the LPW saved at the time of the connect$list call. If the LPW has moved since the connect$list call, "stats.started" in the caller's status frame is set ON.

Having matched the LPW, a test is made to determine whether the LPW is pointing at a patch list. If the LPW is pointing at a patch list, the list data for the list which is being patched is substituted. The substitution insures that the patch is transparent to the user. (See Patching Live Lists.)

EXAMPLE



Having gotten the list ID and item index, "stats.int_idx", and "stats.int_id" are inserted into the caller's status frame. The DCW tally from the DCW mailbox is copied into "stats.tally". Note that for an inactive channel as noted by the earlier call to lpw$active, the list ID, item index, tally, etc. are all set to zero.

Having collected the list data, the current time is noted in "stats.time", the channel activity is marked in "stats.active" and the "filled" entry is set ON. This completes processing of the first status frame if current status was requested. The GIM continued by translating relevant hardware status words into items required by the caller for all caller-supplied status frames except the current status frame (if one was requested).

Recalling that the earlier call to gioc_stat$move has moved all relevant status into the caller's Logical Channel Table (LCT) the GIM examines the start of the status thread, "lct.fststat". If the start of the thread is zero, no status data currently exists for this caller. In the case of a zero thread, request$status simply sets "stats.filled" and "stats.status_waiting" to zero in all caller-supplied status frames. That is, the GIM announces there is no status to return.

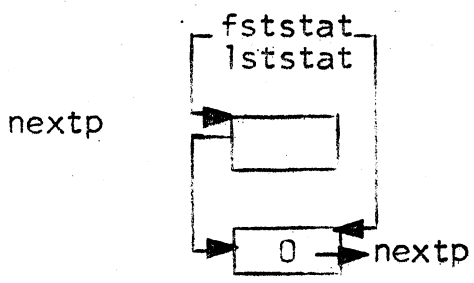
For a non-zero thread, the status data pointed at by "lct.fststat" is unthreaded and prepared for use. The first item in the chain is the oldest status data; the last item is the most recent.

The declaration for each status data block is

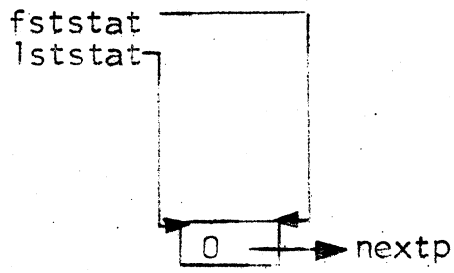
```
/* Declarations for chained status frame */

  dcl 1 sfrm based(fststat),   /* chained status */
      2 nextp bit(18),         /* offset of next frame */
      2 time bit(54),          /* time at interrupt */
      2 statwd bit(72);        /* interrupt status word */
```

To unthread the first item, one simply resets "lct.fststat" to be equal to "sfrm.nextp", thus re-linking the chain around the first status block

EXAMPLE



Before unthreading



After unthreading

A check is then made to see if the end of the chain is reached. If the end has been reached, "lct.lststat" is zeroed. The user-supplied status frame is updated with the "status_waiting" switch by simply setting the switch ON if the thread has not been reduced to the empty thread.

By matching the LPW tally stored in the hardware status word with the LPW which was saved in "lct.stlpw" at the time of the connect$list call, the GIM determines whether the LPW has moved since it was set up. The proper entry in the caller's status frame, "stats.started", is set accordingly.

Request$status continues by matching the LPW tally with the tally base and length of all currently defined lists in order to find the list and item associated with the LPW tally stored in the hardware status word. Reference to the section entitled Generation of DCWs gives the mechanism which insures a unique match.

If a match is found, the list ID and item index are posted in "stats.int_id" and "stats.int_idx", respectively. If no match with any currently defined list is found, the ID and index are set to zero. In addition, a no-match case will cause "stats.started" to be set OFF, by convention.

Having posted the list identity, the DCW tally residue and time of interrupt are copied from the status block into "stats.tally" and "stats.time", respectively. Request$status now calls statfil, a routine which takes a hardware status word and processes it according to the instructions found in the appropriate Class Driving Table. Possible errors from statfil include CDT not found, "cdtnf", illegal field-action code in CDT, "illfld", GIOC not available "giocnf", bad GIOC number, "badcall", and too many lists, "tmlst".

The role and actions of statfil is more completely discussed in a later section, Status Word Translations. We note now, however, that statfil updates the channel activity flag to reflect whether the channel is still active. This flag is then posted by request$status into "stats.active" for the caller's convenience.

As final actions for the caller-supplied status frame, the "filled" switch is set ON, the unthreaded status block is freed and the next caller-supplied status frame is selected. Upon exhausting the status frames, request$status returns.

Moving of Hardware Status Words - gioc_stat$move

At certain times, it is necessary to move the contents of a hardware status queue into a larger, pageable area. Such moves generally occur as a result of a DIM calling the GIM for status through the request$status entry. To move status words, the GIM makes the following call:

```
call gioc_stat$move (lgch, lctp, polsw)
```

where the arguments are defined as follows:

```
lgch fixed bin (12),   /* logical channel number */
lctp ptr               /* pointer to LCT */
polsw bit (1)          /* always "0"b in current
                          implementation */
```

In order to move the status words, the GIM first moves all relevant status words on all GIOCs into a large, pageable, data base known as the Channel Status Table (CST).

The move to the CST is accomplished in the following manner.

For each GIOC, a pointer to the GIOC base is extracted via a check$gioc call. Then, the following actions are performed for each status channel on the selected GIOC. A pointer to the selected status channel's LCT is extracted via a call to check$statusp. The "last" and "middle" pointers for the status queue are extracted and tested. If "last" and "middle" are identical, there are no un-processed status words so no further action is needed for this status channel. However, if "last" is less than "middle", then all the status words from "last" to "middle" must be moved into the CST. If "last" is greater than "middle", the queue has wrapped around the end and only the status words from "last" to the end of the queue should be moved. Having selected the boundaries, the following action occurs for each status word.

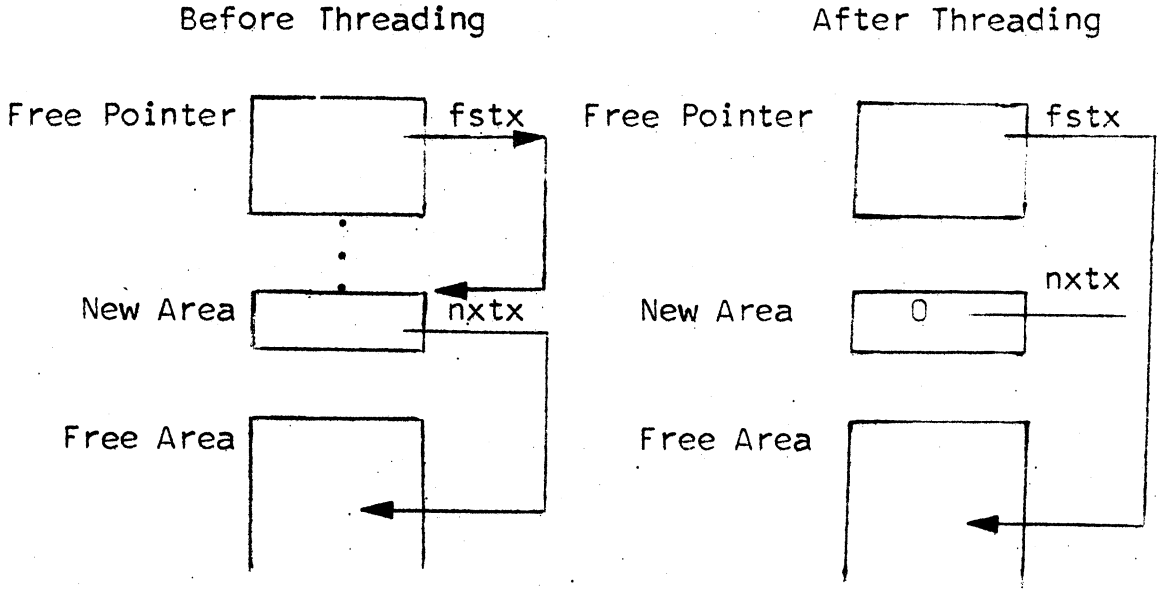
Until the model B GIOC arrives, a call to "fake72" is made to transform each 36-bit status word into a format approximating the 72-bit status word stored by the model B GIOC. The GIM is oriented around a 72-bit status word so that installation of the model B GIOC requires only the removal of the fake72 call.

The physical channel number is extracted from the hardware status word and tested to see if it is a user channel or a connect channel. Assuming a user channel, the following actions are performed.

For a user channel with a defined Logical Channel Table (LCT), the GIM attempts to insert the hardware status data into the Channel Status Table (CST). The storage is done by inspecting the free storage chain originating in slot 0 of the index table of the CST. A chain pointer of 0 indicates no free storage is available. For a 0 chain pointer, a new slot must be extracted by reference to the highest index currently allocated. This index is incremented and the appropriate CST entry, cst.hix, is updated. It is hoped that careful re-use of free storage will tend to keep the CST compacted and alleviate paging necessity as much as possible. Each status block has the following declaration:

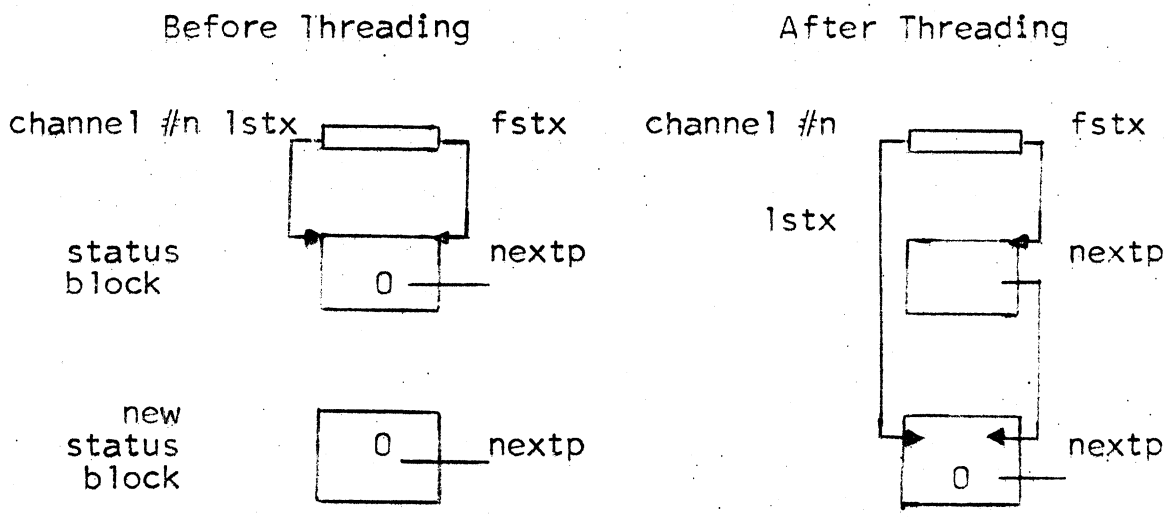
```
/* Declarations for chained status frame */

dcl 1 sfrm based(fststat),      /* chained status */
    2 nextp bit (18),           /* offset of next frame */
    2 time bit (54),            /* time at interrupt */
    2 statwd bit (72);          /* interrupt status word */
```

Having selected a slot for storing the status data, the free pointer is threaded to point to the area indicated by the forward pointer of the new area. The forward pointer of the new free area is then reset to indicate no forward chain. The chaining up to this point is illustrated in the following diagram:



Free area discipline

The status word is now inserted in the status block entry, "statwd". The time of the interrupt is copied into the new status block entry. The status block is then threaded into the other status blocks relevant to the logical channel under inspection. A simple demonstration of the threading of a status block is shown in the following diagram:



To express the threading in words, the CST is a number of single-threaded chains in which each separate chain corresponds to a particular logical channel's unclaimed status words.

After threading in the status data, a check is made to determine whether a hardware status sub-queue boundary was passed. If a boundary was passed, SCWA of the appropriate status channel is reset to re-use the sub-queue just emptied. If the mid-boundary was passed, SCWA is reset to point to the lower sub-queue. If the upper boundary was reached, SCWA is reset to point at the upper sub-queue.

Having moved the status data into the CST, the "last" pointer is updated to the end of the status words that were moved. If the upper end of the queue is reached, "last" is reset to the first item in the queue. After processing all status queues on all GIOCs, the second phase of the status block moving commences.

In the second phase, status data relevant to the logical channel indicated in the call to gioc_stat$move must be moved from the CST into the user's work area within the Logical Channel Table (LCT) for this channel.

Moving status blocks from the CST to the indicated LCT consists mainly of the allocation and threading of a status block within the user's LCT and the filling in of the status block from data contained within some particular status block in the CST. The actions are performed in the following manner.

The first and last pointers for the proper logical channel's thread in the CST are extracted and saved. If the first pointer is zero, there are no entries in the thread. For a zero thread, an immediate return is made.

A non-zero thread pointer indicates one or more status blocks exist and must be moved into the proper LCT. The following actions are performed until all of the chain elements have been moved into the proper LCT.

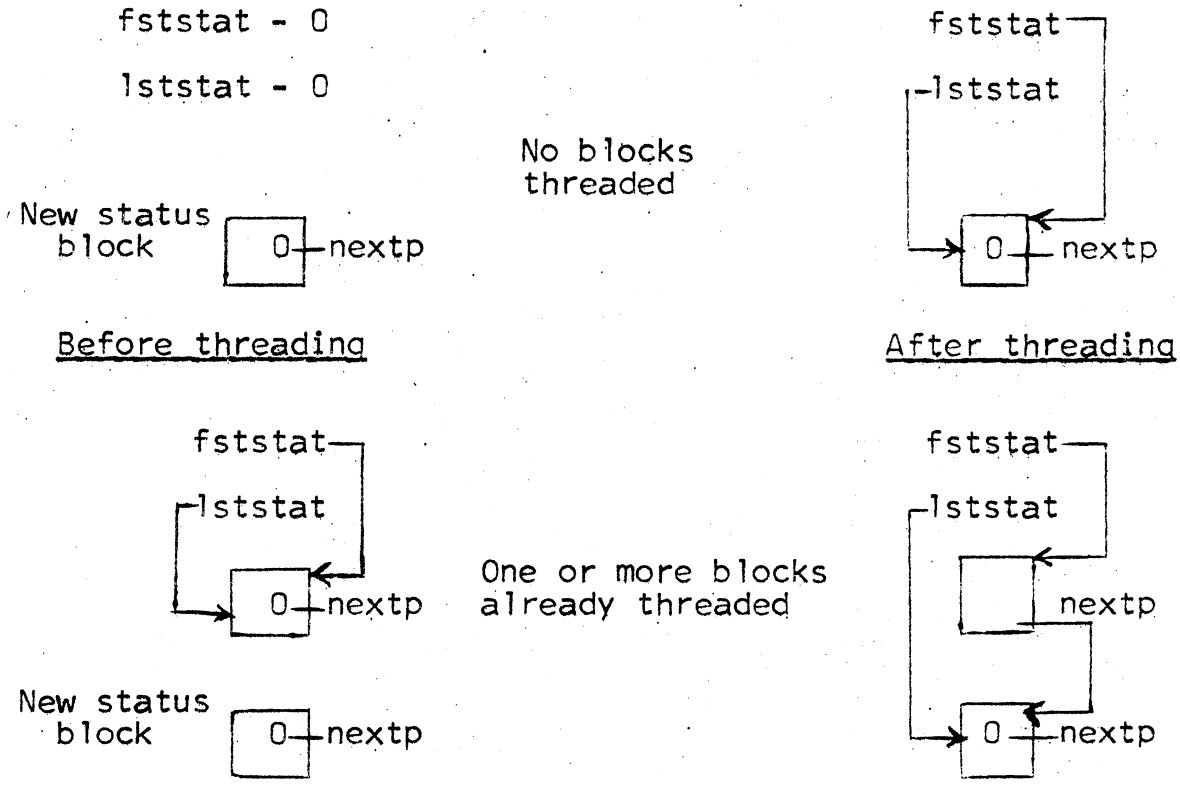
The status block structure is allocated in the LCT. It has the following declaration:

```
/* Declarations for chained status frame */

dcl 1 sfrm based(fststat),      /* chained status */
    2 nextp bit (18),           /* offset of next frame */
    2 time bit (54),            /* time at interrupt */
    2 statwd bit (72);          /* interrupt status word */
```

After allocating the structure, its forward pointer, "sfrm.nextp", is zeroed to indicate no further entries are currently threaded to this item. The hardware status and time are then copied from the appropriate entry in the CST. Note that the oldest entries for a particular channel are copied first.

If no entries are currently chained in the LCT, the forward thread, "lct.fststat", is set to point at the allocated status block. If one or more entries are already threaded in the LCT, the forward thread of the last allocated block (exclusive of the block under consideration), "sfrm.nextp", is set to point at the new status block. In either event, the end-chain pointer, "lct.laststat", is set to point at the newly allocated block.

Example



After threading the status block within the LCT, the matching entry within the CST must be unthreaded. The CST unthreading is accomplished by simply setting the thread starting pointer, "cst.xtab(n).fstx", to point to the same place as the current first block's forward pointer. In effect, this action threads around the first block. If the new forward pointer is zero, there are no more entries in the thread. For an empty thread, the end pointer, "cst.xtab(n).lstx", is zeroed and the entire chain of (now unused) status blocks is returned to the free storage chain by splicing the beginning of the free storage chain, "cst.xtab(0).fstx", to point to the first status block in the thread and setting the last status block in the thread to point to the old beginning of the free storage chain. After updating the chain, gioc_stat$move returns.

Status Word Translation - statfil

In order to relieve the DIM writer of much of the burden of deciphering raw hardware status words, the GIM offers the capability of pre-programmed translation of status words into meaningful symbolic units. In order to obtain translations, the request$status module makes the following call:

```
call statfil (lctp, lgch, sfp, swdt, actbit, srtn)
```

where the arguments are declared as:

```
lctp ptr                  /* pointer to Logical Channel
                             Table */
lgch fixed bin (12)       /* logical channel number */
sfp ptr                   /* pointer to user's status frame */
swdt bit (72)             /* hardware status word */
actbit bit (1)            /* channel activity bit */
srtn bit (36)             /* standard GIM error return word */
```

One can conceive of the translation mechanism as a reverse implementation of the normal Class Driving Table (CDT) manipulations used in list editing. That is, one receives the necessary indices, literals, etc. instead of giving them.

In order to remove device-dependent information from the GIM, three standards must be adhered to in constructing a Class Driving Table (CDT) used for status word translations:

1. Field 1 of the status translation part of the CDT must contain a description of a terminate condition form of hardware status word.

2. Field 2 of the status translation part of the CDT must contain a description of an adapter error hardware status word.

3. Field 3 of the status translation part of the CDT must contain a description of a GIOC-wide error hardware status word.

If the above three requirements are met, then the DIM writer can expect to find three straightforward status flags contained in the returned status frames. Moreover, as will be shortly demonstrated, the GIM also expects to use these flags. Assuming the above requirements are met, the following will be adhered to by the GIM:

1. Statf.stat(1) = 1 if and only if a terminate condition was detected in a relevant hardware status word.

2. Statf.stat(2) = 1 if and only if an adapter error was detected in the GIOC adapter containing the relevant physical channel.

3. Statf.stat(3) = 1 if and only if a GIOC-wide error has occurred.

To facilitate translations, the "compressed" status word is used throughout the statfil module. The compressed status word is a condensed raw status word which leaves only those items of interest to the user. It will be recalled that such items as DCW residue, LPW tally, etc. are all handled separately by other GIM programs. The compressed status word which is constructed has the following declaration:

```
dcl 1 cswd based (p),     /* declarations for compressed
                             status word */
    2 cause bit (4),      /* bits 0-3 of mod B GIOC status
                             word */
    2 intsig bit (2),     /* bits 4-5 of mod B GIOC status
                             word */
    2 status bit (12),    /* device status from GIOC
                             adapter */
    2 dcwsw bit (1),      /* low-order bit of device channel
                             number */
    2 given bit (4),      /* not used on mod B GIOC; on mod
                             A GIOC, the four bits relate
                             how much status is relevant */
```

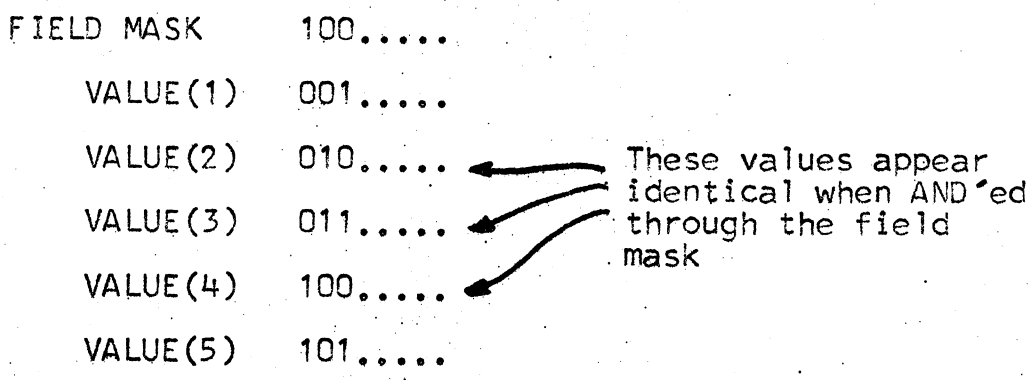
Following initial setup, statfil prepares for translation by determining the greatest number of fields that can be safely processed by taking the smaller of the number of CDT fields and the length of the status array, "statf.stat1". This number is then used to control the following loop iteration count. The following processing occurs for each field found in the CDT for type 1 (status request) entries.

For each field, the CDT is checked to insure that the field is defined. Undefined fields will cause the matching status frame entry to be set to zero; no other processing action for that frame takes place. For a defined field, a pointer to the appropriate CDT "field" sub-structure is generated. The action code for this field is extracted from the "field" sub-structure.

An action code of 1 indicates a mask-value substitution. In mask-value substitution, statfil searches the "value" array of the current field for an item which, when viewed through the current field mask, matches the compressed hardware status word. Upon finding a match, the index of the "value" array containing the match is placed in the status frame entry, "statf.stat(n)". If no match occurs, a zero is placed in the status frame entry.

Note that mask-value substitution must be carefully considered when constructing a CDT as it is apparent that the field mask selected for a particular field has a powerful effect on the choice of "value" entries. A moments consideration of the following entries will reveal that the example's field mask has the effect of nullifying three of the "value" entries.

EXAMPLE



An action code of 2 indicates literal substitution. In literal substitution, the compressed hardware status word is AND'ed through the current field mask, 24 bits are extracted from the result and placed in the proper status frame. The selection of which 24 bits to use is given by the "field_end" quantity for the current field. The literal substitution option might well be used to extract the raw 12-bit device status and return it to the caller for further processing.

Any other action code is in error and will cause the "illfld" error to be set. Since the action codes are derived from the CDT itself, the "illfld" error indicates a defective Class Driving Table. After translating the hardware status word, statfil must now tend to any channel activity and check for major system errors.

The current channel activity is matched against the detection of a terminate status in the current hardware status word.

Should the channel have ceased activity (as demonstrated by the current status word) a call to cread will insure that all data has been moved to the user's area. (see Copying Data Into a User's Area). Possible errors from cread include: bad GIOC number, "badcall", GIOC not available, "giocnf", system or machine error, "syserr", and LCT space exhausted, "tmlst". After copying any outstanding data, a call to lpw$safe will insure the channel is completely stopped. Possible errors include bad GIOC number, "badcall", and GIOC not available, "giocnf". After shutting down the channel, all DCW and data areas are released via a call to mkdcw$free.

The GIM now tests statf.stat(2) to see whether an adapter-wide error occurred. A typical adapter error is a delay-line sychronization error on the teletype adapter. Should an adapter error occur, the GIM proceeds to note an adapter error for all channels connected to the adapter containing the current channel. The adapter error bit, "adper", (found in the CST) is turned ON for all affected channels.

After testing for an adapter error, the GIM tests the statf.stat(3) word for a GIOC-wide error. Such errors reflect the ultimate in calamity as all Multics devices except the Fire Hose Drum will be adversely affected. Upon detecting a GIOC-wide error, all channels on the affected GIOC have their GIOC error bit, "giocer", (found in the CST) turned ON.

Having checked for major errors, statfil tests the adapter error bit, "adper", (within the CST) for the current channel. Should the bit be ON, the adapter error flag is set in the proper status frame, "statf.stat(2)", and the CST entry is reset. Similar actions occur for the GIOC error, "giocer", except that the GIOC error flag is set in "statf.stat(3)".

After completion of the above error checking, statfil is finished.