HONEYWELL INTEROFFICE CORRESPONDENCE

PHOENIX OPERATIONS – HONEYWELL INFORMATION SYSTEMS

DATE	771212	PHONE		MAIL ZONE		COPIES	
TO	Specification File						
FROM	C. Bjerke [illegible]						
COMPONENT	LADC						
SUBJECT	I/O Design Specification						

SF section:		14.4
doc #	:	0257A-0
page	:	1

This specification has incorporated changes to the original design specification suggested by the I/O Design Review Committee and as such represents the design to be implemented.

Design Specifications for a New I/O System

1. INTRODUCTION

The interim I/O System consists of two parts: IOQ and IOS. IOQ contains the caller and user interfaces; IOS, which was adapted from GCOS IOS, contains the I/O scheduling, interrupt, and error recovery routines.

In order to meet several requirements of the CP-6 I/O System, The interim IOS will be replaced with a new set of I/O processing routines ("the new IOS"). Although some internal changes to IOQ will be necessary, the interfaces to IOQ for routines outside of the I/O System will remain essentially unchanged.

1.1 REQUIREMENTS

The New I/O System for CP-6 will provide the following features:

- o It will be written mostly in PL6 for easy maintenance.
- o Each handler will be an independent set of routines. Common code will be primarily subroutines.
- o Table searches will be minimized by using queue structures where applicable.
- o It will be designed for a multi-processor environment.
- o It will be designed so that a wide variety of device handlers may be accomodated, including those specifically mentioned herein.
- o Disk I/O will be scheduled so as to minimize arm positioning delays and rotational latency.

1.2 Conventions

1.2.1 Module and Entry Point Names

All module and entry point names will have the following prefixes:

Prefix	Module
NIC$	Console Handler
NID$	Disk Handler
NIL$	Line Printer Handler
NIO$	IOQ
NIP$	Card Punch Handler
NIQ$	Packet Allocator
NIR$	Card Reader Handler
NIS$	Driver and Interrupt Distributor (BMAP)
NIT$	Magnetic Tape Handler
NIU$	Utility Subroutines (PL6)

1.2.2 External Symbol Names

All external symbol names defined in the above modules will have the prefix: NI_.

1.2.3 Calling Sequences

All calling sequences will be PL-6 calling sequences, which are described in AF sect. 6.3, #0137.

1.3 Scope

This document describes the new IOS. Section 2 discusses those aspects of the hardware which must be taken into consideration when designing the I/O System. Section 3 contains an overview. Section 4 describes the data structures. Section 5 describes the procedures. Section 6 describes the changes to IOQ and entry points of IOQ which are used by IOS.

This document does not describe the individual device handlers, other than indicating the general modus operandi thereof.

1.4 References

This design specification is based on information contained in the following documents:

58001190	EPS-1 4 Megaword System Controller
43A239854	EPS-1 6000B Input/Output Multiplexer (IOM) Centr
43A177880	EPS-1 6000 IOM Peripheral Subsystem Interface Ad
43A177879	EPS-1 MPC - PSI Link Adapter
58001108	EPS-1 Unit Record MPC Controller/Subsystem
58008518	EPS-1 MTP601 Tape Controller
43A232230	EPS-1 DSC181/DSC190 Controller

2. HARDWARE CONSIDERATIONS

The general configuration of the L66 I/O hardware is shown in Fig. 2.1. There may be up to four CPUs. By the use of mask registers, I/O interrupts can be selectively directed to different CPUs. However, in order to maintain the current concept utilized in the Job Scheduler, all I/O interrupts will be sent to one CPU. There may be more than one System Controller (SCU), but if all of the CPUs and IOMs are to have the same address space, the only purpose for more SCUs is to provide more than 4 megawords of memory. There may be up to four Input/Output Multiplexors (IOMs), but each IOM is controlled independently and identically. Multiple IOMs can be used to provide greater I/O bandwidth and/or more I/O channels.

A Micro-Programmed Controller (MPC) may have one or two Link Adapters, each of which is capable of performing one data transfer operation at a time. Thus, two Link Adapters allow two simultaneous data transfer operations on dual-access tape and disk subsystems. Each Link Adapter may be connected to one or two Peripheral Subsystem Interface Adapters (PSIAs). The only purpose for having two PSIAs attached to one Link Adapter is to provide redundant paths to the MPC. Typically, the two PSIAs would be connected to different IOMs. Each PSIA may control up to eight I/O channels. The latter are usually called logical channels, to distinguish them from physical channels, which connect PSIAs to Link Adapters. The use of multiple logical channels on a PSIA depends on the type of peripheral subsystem.

A unit record MPC can control up to eight peripherals such as card readers, line printers, and card punches. Each device is assigned the same logical channel number on all of the PSIAs connected to the MPC. Thus, there may be up to four paths to each device (2 Link Adapters X 2 PSIAs). However, since unit record data transfers are always buffered in the MPC and the data transfer time is much less than the device cycle time, more than one Link Adapter or PSIA only provide redundant paths to the MPC. Thus, a unit record peripheral would usually be accessed by the same channel. Any contention between the devices for logical channels would be resolved by the MPC and thus would be transparent to the I/O software in the CPU.

A magnetic tape MPC can control up to 16 tape drives via a 1 X N or a 2 X N switch. Thus, with two Link Adapters, two tape drives may be doing simultaneous data transfers. Keeping a Link Adapter busy requires only two logical channels on the PSIA: one for the current request being processed, and one to hold a request which can be started immediately when the current request is finished. If there are more than two logical channels, a request issued to the lowest priority channel may never be started because enough requests are issued on higher priority channels to keep the Link Adapter busy. Thus, I/O scheduling for a tape subsystem must resolve contention for the channels and, for maximum throughput, should keep both Link Adapters (if there are two) busy.

A disk subsystem may have one or two MPCs, each of which may have one or two Link Adapters, thus a maximum disk subsystem may perform four simultaneous data transfers. On a disk subsystem, the multiple logical channels of the PSIA are used to implement seek overlap. When a request is issued to a disk channel, the MPC initiates the seek phase of the request, then releases the logical channel so that other logical channels may be processed. In this manner, several disk drives may all be seeking simultaneously. When one of them arrives at the requested sector, the MPC restarts the corresponding logical channel, and performs the data transfer operation. If a disk arrives at the requested sector while a data transfer is in progress, the on-sector interrupt from the drive will be ignored and will occur again on the next revolution. However, if maximum performance is to be obtained from the disk subsystem, the disk I/O software must still schedule disk operations so as to minimize arm positioning delays and rotational latency. Thus, I/O scheduling for a disk subsystem must resolve contention for the channels, it should keep all of the Link Adapters busy, and it should schedule the requests for a drive so as to minimize seek delays.

3. OVERVIEW

The new IOS will consist of the Driver, the Interrupt Distributor, the Lost Interrupt Poller, and several device handlers, one for each type of supported device. The Driver will set up the connect and payload channel mailboxes and issue the Connect I/O Channel (CIOC) instruction to initiate the I/O operation. The Interrupt Distributor will be entered whenever an interrupt occurs. It will determine the channel which caused the interrupt and call the Poster of the device handler for that channel. The Lost Interrupt Poller is called periodically to check for lost interrupts. In addition, there will be some common subroutines, such as an interface to KEYIN.

Each handler will consist of a Scheduler and a Poster. The Scheduler is called by IOQ or the Poster to schedule the device for the I/O operation. The Poster is called by the Interrupt Distributor when an interrupt occurs to start the next request on the interrupting channel, analyze the completed request, perform any necessary error recovery, and return the completed request to IOQ for end-action processing.

3.1 Table and Queue Structure

The static table structure for the New I/O System is shown in Fig. 3.1. The dynamic table and queue structure is shown in Fig. 3.2.

3.1.1 Tables

All of the I/O tables will be constructed at startup or recovery time from configuration information stored on disk. Each table will have a pointer in the New I/O System's static storage; this pointer will be set up when the table is constructed.

o Device Control Table (N$DCT)

  The Device Control Table contains scheduling information for all local and remote devices.

o Channel Table (NI$CHT)

  The Channel Table points to the Driver Queue Header and Poster for each channel.

o Driver Queue Header (NI$DQH)

  The Driver Queue Header contains scheduling data for a channel.

o Subsystem Queue Header (NI$SQH)

  The Subsystem Queue Header contains scheduling data for a subsystem.

3.1.2 Queues

All FIFO queues will be linked circularly, where each entry points to its successor, and the last entry points to the first. The queue header, _i.e._, the pointer to the queue, then points to the last entry in the queue. This saves space, in that the queue header needs only one pointer, yet allows easy enqueuing at either end of the queue and easy dequeuing at the head of the queue.

o Scheduler Queue

  Each device has a queue of outstanding requests called the Scheduler Queue. For most devices, this will be a FIFO queue, but for disk, the requests will be ordered according to arm position, so as to minimize arm positioning delays. The header of each Scheduler Queue is in the Device Control Table entry. As requests are received, IOQ enqueues them on the requested device's Scheduler Queue. When a channel has been assigned to this device, the device Scheduler will move those requests which can be processed as a group to the Driver Queue.

- Assign Queue

  Each subsystem will have a Assign Queue, which is a FIFO queue of devices waiting for channels to become available. The Assign Queue header is in the Subsystem Queue Header. The device Scheduler will put an idle device on the Assign Queue when there are requests on that device's Scheduler Queue. When a channel is assigned to the device, the device will be removed from the Assign Queue.

- Driver Queue

  Each channel will have a Driver Queue, which is a queue of requests that are ready to be started. This queue will be FIFO for most devices, but for disk, it will be ordered according to sector position within a track, so as to minimize rotational latency. At any moment in time, a Driver Queue will contain requests for only one device. For a disk channel, the Driver Queue will contain requests for only a single cylinder of a disk pack. The device Scheduler enqueues requests on the Driver Queue when a channel has been assigned. The device Poster removes a request from the Driver Queue when it terminates.

  This two-level queuing structure (Scheduler Queue and Driver Queue) is designed to maximize disk throughput. Basically, the Scheduler Queue is a queue of cylinders to be accessed, and the Driver Queue is a queue of requests for the current cylinder.

3.2 I/O Request Processing

In the New I/O System, IOQ is called for both local and remote peripheral requests. On receiving a request, IOQ enqueues the request on the Scheduler Queue for the device. Then, if this request is for a remote device, IOQ calls the Front End Interface to process the request. If this request is for a local device, IOQ allocates an IOS Request Packet and links it to the IOQ Request Packet. From the information in the IOQ Request Packet, IOQ then sets up the PCW, LPWX, and DCW list in the IOS Request Packet and calls the device Scheduler for the requested device.

If the device is idle, the device Scheduler enqueues the device on the Assign Queue for that subsystem. It then processes the Assign Queue, assigning a channel to each device thereon, until there are no more channels available or the Assign Queue is empty.

When a channel is assigned to a device (see section 3.4), the device is removed from the Assign Queue, and one or more requests, depending on the device, are moved from the device's Scheduler Queue to the channel's Driver Queue. The Driver is then called to start the first request on the channel's Driver Queue.

The Driver sets up the connect and payload channel mailboxes and issues the CIOC instruction to start the I/O operation.

When the interrupt occurs, the Interrupt Distributor determines the channel number and calls the device Poster for that channel. If no errors occurred, the Poster calls the Driver to start the next request on this channel. If there were no errors or the errors were unrecoverable, the completed request is returned to IOQ for end-action processing. If the channel is now idle, the Poster calls the device Scheduler to restart the channel. If an error occurred that must be retried, the request is rescheduled to be processed next. If repositioning is required before the retry, reserved IOS Request Packets are used to effect the repositioning and are scheduled ahead of the request packet. This rescheduling may be done in either of two ways, depending on the device. For sequential devices, _i.e._, devices other than disk, it requeues the IOS packet on the Driver Queue and calls the Driver to process the request. For disk devices, it requeues the IOS packet on the Scheduler Queue and calls the device Scheduler to process the request. If an error occurred which requires operator intervention, a message is sent to the operator's console, and the request and the contents of the channel's Driver Queue are moved back to the beginning of the device's Scheduler Queue. The device Scheduler is then called to restart the channel (with requests for another device). The completion of the operator intervention may be signaled either by a special interrupt or an operator keyin. In the former case, the Interrupt

Distributor will post the special interrupt to the device Poster. In the latter case, IOS' interface to KEYIN will post the keyin to the Poster. When the operator intervention is complete, the device will be inserted at the beginning of the Assign Queue, and the device Scheduler will be called to restart the device.

3.3 Channel Assignment

The multiplicity of channels which may access a subsystem requires that channels be assigned dynamically to devices as they are needed. For unit record subsystems, this channel assignment amounts only to selecting an available one from those configured. For tape and disk subsystems, Link Adapters and channels are assigned in a round-robin fashion. When there is a device to be assigned to a channel, the next Link Adapter with an available channel is found, and the device is assigned to the next available channel on that Link Adapter.

3.4 Direct Channel I/O

An entry point to the Driver will be provided for issuing a connect to a direct channel. The Front End Handler will have an Poster of its own for handling interrupts from the direct channel.

3.5 Lost Interrupts

Occasionally, the hardware may fail in such a manner that no interrupts are returned from an I/O operation. For this reason, we need a Lost Interrupt Poller, i.e., a routine which is called periodically, say, every 5-10 seconds, to poll all of the active channels to see if a channel has been active for an unreasonably long time. If this is so, i.e., a lost interrupt has occurred, the device Poster is called with a unique interrupt level. The latter will then take the appropriate error recovery action.

3.6 Error Statistics and Error Logging

All I/O errors will be logged in the system error log. Presumably there will be a ghost job that will monitor the error log and warn the operator if the error rate on a device exceeds an acceptable level (thresholding). The error log entry will include a time stamp, the external device identification, the IOM and channel numbers, the hardware status words, and any extended status that may be available. In the case of a memory error during I/O, memory mapping information will also be included.

3.7 Test and Diagnostics

The New I/O System will provide options in the IOQ Request Packet for specifying T&D I/O. This will take the form of a unique logical function code, which will indicate that the DCW list is already set up (in the job's virtual memory), and that no error recovery or logging are to be performed on this request. The I/O System tables will have provision for reserving a device for T&D.

4. <u>DATA STRUCTURES</u>

The following packets and tables will be used by the New I/O System:

4.1 <u>IOQ Request Packet (N$REQ)</u>

The IOQ Request Packet is used for all I/O requests for both local and remote devices. It contains the information which must be provided by the caller and space for information to be returned to the caller on completion.

```
%MAC N$REQ(NAME=N$REQ,STCLASS=BASED);
DCL 1 NAME STCLASS ALIGNED,            /* I/O REQUEST PACKET */
    0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7
   ¦-----------------¦-----------------¦-----------------¦-----------------
  0¦FL$
   ¦-----------------¦-----------------¦-----------------¦-----------------
  1¦DLA
   ¦DCTX                         ¦DRELADDR
   ¦-----------------¦-----------------¦-----------------¦-----------------
  2¦BUFSIZE                                ¦*¦FC         ¦OPFLG
   ¦                                       ¦ ¦           ¦U¦A¦E¦E¦W¦B¦H¦R¦
   ¦-----------------¦-----------------¦-----------------¦-----------------
  3¦BUF$
   ¦-----------------¦-----------------¦-----------------¦-----------------
  4¦PTP                                ¦# # # # # # # # # # # # # # # # #
   ¦-----------------¦-----------------¦-----------------¦-----------------
  5¦DCB$
   ¦-----------------¦-----------------¦-----------------¦-----------------
  6¦EAENTRY
   ¦-----------------¦-----------------¦-----------------¦-----------------
  7¦EAINFO
   ¦-----------------¦-----------------¦-----------------¦-----------------
 10¦EVNTINFO
   ¦-----------------¦-----------------¦-----------------¦-----------------
 11¦ARSIZE                                 ¦*¦CC         ¦USER#
   ¦                                       ¦ ¦ERR  ¦ABN  ¦
   ¦                                       ¦ ¦I¦P¦I¦E¦E¦B¦
   ¦-----------------¦-----------------¦-----------------¦-----------------
 12¦ARCT
   ¦-----------------¦-----------------¦-----------------¦-----------------
 13¦RCT
   ¦-----------------¦-----------------¦-----------------¦-----------------

      2 FL$ PTR,                       /* LINK TO NEXT PACKET IN QUEUE */
      2 DLA ALIGNED,                   /* DEVICE LOGICAL ADDRESS */
        3 DCTX UBIN(15) UNAL,          /* DEVICE TABLE INDEX */
        3 DRELADDR UBIN(21) UNAL,      /* DEVICE-RELATIVE ADDRESS */
      2 SLA REDEF DLA,                 /* SET LOGICAL ADDRESS */
        3 SETX UBIN(9) UNAL,           /* PACK SET INDEX */
        3 SRELADDR UBIN(27) UNAL,      /* SET-RELATIVE ADDRESS */
      2 BUFSIZE UBIN(20) UNAL,           /* BUFFER SIZE (BYTES) */
      2 * BIT(1),
      2 FC UBIN(6) UNAL,                 /* LOGICAL FUNCTION CODE */
      2 OPFLG,                           /* OPERATION FLAGS */
        3 USER BIT(1),                   /* USER-ASSOCIATED I/O */
        3 ARS BIT(1),                    /* SET ARS OF DCB */
        3 EVNT BIT(1),                   /* REPORT COMPLETION EVENT */
        3 EA BIT(1),                     /* CALL END-ACTION ROUTINE */
        3 WAIT BIT(1),                   /* BLOCK ASSOCIATED USER AFTER QUEUE
        3 BPF BIT(1),                    /* BUF$ CONTAINS PTR -> BUFFER */
        3 HOLD BIT(1),                   /* DO NOT RELEASE PACKET ON COMPLETI
        3 REQ BIT(1),                    /* RE-QUEUE PACKET */
        3 SET BIT(1),                    /* 0=REQ TO DEV, 1=REQ TO SET */
      2 BUF$ PTR,                        /* -> BUFFER IF BPF='1'B */
      2 BUFADDR REDEF BUF$ UBIN(26),     /* BUFFER ADDRESS IF BPF='0'B */
      2 BUFVIRT REDEF BUF$,              /* VIRTUAL BUFFER ADDRESS IF PTP
        3 * BIT(5),
        3 BASE UBIN(18) UNAL,
        3 BYTE UBIN(3) UNAL,
        3 * BIT(10),
      2 BUFREAL REDEF BUF$,              /* REAL BUFFER ADDRESS IF PTP=0 */
        3 EXTA UBIN(6) UNAL,
        3 ADR UBIN(18) UNAL,
        3 BYTE UBIN(2) UNAL,
        3 * BIT(10),
      2 PTP UBIN(18) UNAL,               /* BPF=0: PAGE TABLE POINTER (0=REA
      2 DCB$ PTR,                        /* -> USER'S DCB */
      2 EAENTRY EPTR,                    /* END-ACTION PROCEDURE */
      2 EAINFO UBIN(36),                 /* END-ACTION PARAMETER */
      2 EVNTINFO UBIN(36),               /* EVENT INFO */
      2 ARSIZE UBIN(20) UNAL,            /* ACTUAL RECORD SIZE */
      2 * BIT(1),
      2 CC,                              /* LOGICAL COMPLETION CODE (0=OK) *
        3 ERR,
          4 IO BIT(1),                   /* I/O ERROR */
          4 PE BIT(1),                   /* PARITY ERROR */
          4 INVD BIT(1),                 /* INVALID OPERATION */
        3 ABN,
          4 EOF BIT(1),                  /* END-OF-FILE */
          4 EOT BIT(1),                  /* END-OF-TAPE */
          4 BOT BIT(1),                  /* BEGINNING-OF-TAPE */
      2 USER# UBIN(9) UNAL,              /* USER ID */
      2 ARCT SBIN WORD,                  /* ACTUAL RECORD COUNT */
      2 RCT SBIN WORD;                   /* RECORD COUNT */
%MEND;
```

4.2 IOS Request Packet (NI$REQ)

The IOS Request Packet is allocated by IOQ for a local device I/O request. It resides in real memory and contains information specific to a local device I/O operation.

```
%MAC NI$REQ(NAME=NI$REQ,STCLASS=BASED);
DCL 1 NAME STCLASS DALIGNED,               /* I/O REQUEST PACKET */
      0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
     ¦-----------------¦-----------------¦-----------------¦-----------------
    0¦FL$
     ¦-----------------¦-----------------¦-----------------¦-----------------
    1¦IOQ$
     ¦-----------------¦-----------------¦-----------------¦-----------------
    2¦DCT$
     ¦-----------------¦-----------------¦-----------------¦-----------------
    3¦ERRCT      ¦LPWCONT     ¦# # # # # # # # # # # # # # # # # # # # # # # #
     ¦-----------------¦-----------------¦-----------------¦-----------------
    4¦PCW
     ¦*                     ¦AEX         ¦IDCW ¦M¦*
     ¦-----------------¦-----------------¦-----------------¦-----------------
    5¦
     ¦CHANNEL          ¦PTP                                ¦FLAGS
     ¦-----------------¦-----------------¦-----------------¦-----------------
    6¦LPWX
     ¦BASE                               ¦SIZE
     ¦-----------------¦-----------------¦-----------------¦-----------------
    7¦DCW
     ¦FWA                                ¦ICP  ¦TYPE ¦TALLY
     ¦-----------------¦-----------------¦-----------------¦-----------------
   10¦DCW
     ¦FWA                                ¦ICP  ¦TYPE ¦TALLY
     ¦-----------------¦-----------------¦-----------------¦-----------------
   11¦DCW
     ¦FWA                                ¦ICP  ¦TYPE ¦TALLY
     ¦-----------------¦-----------------¦-----------------¦-----------------
   12¦DCW
     ¦FWA                                ¦ICP  ¦TYPE ¦TALLY
     ¦-----------------¦-----------------¦-----------------¦-----------------
   13¦DCW
     ¦FWA                                ¦ICP  ¦TYPE ¦TALLY
     ¦-----------------¦-----------------¦-----------------¦-----------------
   14¦SEEK
     ¦SCL                   ¦SSZ    ¦SECTOR
     ¦-----------------¦-----------------¦-----------------¦-----------------
   15¦CL$
     ¦-----------------¦-----------------¦-----------------¦-----------------
   16¦STATUS
     ¦P¦P¦MAJOR  ¦MINOR     ¦O¦M¦*  ¦I¦*¦CHAN ¦IOM  ¦*          ¦RCR
     ¦-----------------¦-----------------¦-----------------¦-----------------
   17¦
     ¦NWA                                ¦NCP  ¦R¦*  ¦TALLYR
     ¦-----------------¦-----------------¦-----------------¦-----------------
```

```
        2 FL$ PTR,                              /* LINK TO NEXT PACKET IN QUEUE */
        2 IOQ$ PTR,                             /* -> ASSOCIATED IOQ REQUEST PACKET
        2 DCT$ PTR,                             /* -> DEVICE */
        2 ERRCT UBIN(6) UNAL,                   /* ERROR RETRY COUNT */
        2 LPWCONT BIT(6),                       /* LPW CONTROL BITS */
        2 PCW DALIGNED,                         /* PERIPHERAL CONTROL WORD */
          3 * BIT(12),
          3 AEX UBIN(6) UNAL,                   /* ADDRESS EXTENSION (FOR REAL I/O)
          3 IDCW BIT(3),                        /* INIT('7'O), IDCW FLAG */
          3 M BIT(1),                           /* MASK BIT */
          3 * BIT(14),
          3 CHANNEL UBIN(9) UNAL,               /* CHANNEL NUMBER */
          3 PTP UBIN(18) UNAL,                  /* PAGE TABLE POINTER */
          3 FLAGS BIT(9),                       /* PTP, PGE, AUX */
        2 LPWX ALIGNED,                         /* LIST POINTER WORD EXTENSION */
          3 BASE UBIN(18) UNAL,                 /* LOWER BOUND (MOD 2 WORDS) */
          3 SIZE UBIN(18) UNAL,                 /* SIZE (WORDS) */
        2 DCW(0:4),                             /* DCW LIST */
          3 FWA UBIN(18) UNAL,                  /* FIRST WORD ADDRESS */
          3 ICP UBIN(3) UNAL,                   /* INITIAL CHARACTER POSITION */
          3 TYPE BIT(3),                        /* DCW TYPE */
          3 TALLY UBIN(12) UNAL,                /* WORD COUNT */
        2 SEEK ALIGNED,                         /* DISK SEEK ADDRESS WORD */
          3 SCL UBIN(12) UNAL,                  /* SECTOR COUNT LIMIT */
          3 SSZ UBIN(4) UNAL,                   /* SECTOR SIZE */
          3 SECTOR UBIN(20) UNAL,               /* SEEK ADDRESS */
        2 CL$ PTR,                              /* LINK TO DCW LIST PACKETS */
        2 STATUS DALIGNED,                      /* STATUS WORDS */
          3 PRESENCE BIT(1),                    /* ALWAYS ONE */
          3 POWEROFF BIT(1),                    /* POWER OFF STATUS */
          3 MAJOR BIT(4),                       /* DEVICE MAJOR STATUS */
          3 MINOR BIT(6),                       /* DEVICE MINOR STATUS */
          3 ODD BIT(1),                         /* ODD WORD COUNT */
          3 MARKER BIT(1),                      /* MARKER INTERRUPT */
          3 * BIT(2),
          3 ININT BIT(1),                       /* INITIATION INTERRUPT */
          3 * BIT(1),
          3 CHAN BIT(3),                        /* CHANNEL STATUS */
          3 IOM BIT(3),                         /* IOM STATUS */
          3 * BIT(6),
          3 RCR UBIN(6) UNAL,                   /* RECORD COUNT RESIDUE */
          3 NWA UBIN(18) UNAL,                  /* NEXT WORD ADDRESS */
          3 NCP UBIN(3) UNAL,                   /* NEXT CHAR POSITION */
          3 READFLG BIT(1),                     /* READ/WRITE FLAG */
          3 * BIT(2),
          3 TALLYR UBIN(12) UNAL;               /* TALLY RESIDUE */
%MEND;
```

4.3 Device Control Table (N$DCT)

The Device Control Table contains scheduling information for both local and remote devices. It is accessed by a parallel table of pointers, called N$DCT$; the index into this table of pointers is the Device Control Table Index (DĊTX). The pointer table will be pointed to by the pointer, N$DCT$$. Thus, a reference to a field in the Device Control Table would appear as:

```
     N$DCT$$->N$DCT$(DCTX)->N$DCT.field.
```

This pointer-table structure allows DCT entries to be of different sizes and allows dynamic allocation of packets for remote devices.

```
%MAC N$DCT(NAME=N$DCT,STCLASS=BASED);
DCL 1 NAME STCLASS ALIGNED,              /* DCT ENTRY */
     0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
    |-----------------|-----------------|-----------------|-----------------
   0|FL$
    |-----------------|-----------------|-----------------|-----------------
   1|DEVNM
    | - - - - - - - - | - - - - - - - - | - - - - - - - - | - - - - - - - -
   2|
    |-----------------|-----------------|-----------------|-----------------
   3|DCTX                               |DFLG       |TYPE       |DVN
    |                                   |I|O|*|O|L|O|           |
    |-----------------|-----------------|-----------------|-----------------
   4|SQ$
    |-----------------|-----------------|-----------------|-----------------
   5|SPEC
    |-----------------|-----------------|-----------------|-----------------
   6|SPECINFO
    |-----------------|-----------------|-----------------|-----------------
   7|SQH$
    |-----------------|-----------------|-----------------|-----------------
  10|DQH$
    |-----------------|-----------------|-----------------|-----------------
  11|STATE      |STA|# # # # # # # # # # # # # # # # # # # # # # # # # # # #
    |-----------------|-----------------|-----------------|-----------------
  12|ERROR
    |-----------------|-----------------|-----------------|-----------------
  13|DISK
    |SETL$
    |-----------------|-----------------|-----------------|-----------------
  14|
    |*
    | - - - - - - - - | - - - - - - - - |-----------------|-----------------
  15|                                   |# # # # # # # # # # # # # # # # # #
    |                                   |# # # # # # # # # # # # # # # # # #
    |-----------------|-----------------|-----------------|-----------------
```

```
      2 FL$ PTR,                            /* FORWARD LINK FOR ASSIGN QUEUE */
      2 DEVNM CHAR(8) ALIGNED,              /* DEVICE NAME E.G. LP02 */
      2 DCTX UBIN(18) UNAL,                 /* DCT INDEX */
      2 DFLG,                               /* DEVICE FLAGS */
        3 INPUT BIT(1),
        3 OUTPUT BIT(1),
        3 * BIT(1),
        3 OCK BIT(1),                       /* OPERATOR'S CONSOLE */
        3 LCL BIT(1),                       /* LOCAL DEVICE */
        3 OLDSC BIT(1),
      2 TYPE UBIN(6) UNAL,                  /* DEVICE TYPE */
      2 DVN UBIN(6) UNAL,                   /* DEVICE NUMBER */
      2 SQ$ PTR,                            /* HEAD OF SCHEDULER QUEUE */
      2 SPEC EPTR,                          /* SPECIAL INTRP ENTRY */
      2 SPECINFO UBIN(36),                  /* SPECIAL INTRP INFORMATION */
      2 SQH$ PTR,                           /* -> SUBSYSTEM QUEUE HEADER */
      2 DQH$ PTR,                           /* CURRENTLY ASSIGNED DRIVER QUEUE
      2 STATE UBIN(6) UNAL,                 /* SCHEDULING STATE */
      2 STATUS UBIN(2) UNAL,                /* DEVICE STATUS (UP,DOWN,T&D) */
      2 ERROR UBIN(36),                     /* ERROR STATISTICS */
      2 DISK,                               /* DISK-SPECIFIC DATA */
        3 SETL$ PTR,                        /* LINK TO NEXT PACK IN SET */
        3 * BIT(54),
      2 PRINTER REDEF DISK,                 /* PRINTER-SPECIFIC DATA */
        3 VFC$ PTR,                         /* -> VFC IMAGE */
        3 CHAIN$ PTR,                       /* -> CHAIN IMAGE */
        3 VFCL UBIN(9) UNAL,                /* VFC IMAGE LENGTH */
        3 CHAINL UBIN(9) UNAL;              /* CHAIN IMAGE LENGTH */
%MEND;
```

4.4 Device Table (NI$DVT)

The Device Table contains static information for each type of local device in the system. The index into this table is the device type, which is contained in each local entry of the Device Control Table. This table will be pointed to by the pointer, NI$DVT$.

```
%MAC NI$DVT(NAME=NI$DVT,STCLASS="(0:0) BASED(NI$DVT$)");
DCL 1 NAME STCLASS ALIGNED,
     0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7
    |-----------------|-----------------|-----------------|-----------------
   0|DOT$
    |-----------------|-----------------|-----------------|-----------------
   1|SCHED
    |-----------------|-----------------|-----------------|-----------------
   2|POST
    |-----------------|-----------------|-----------------|-----------------
   3|CYLINDERS             |SURFACES    |CYLSIZE          |SECTORS     |P|#
    |-----------------|-----------------|-----------------|-----------------
   4|MODEL
    | - - - - - - - - | - - - - - - - - | - - - - - - - - | - - - - - - - -
   5|
    |-----------------|-----------------|-----------------|-----------------

       2 DOT$ PTR,                          /* PRE-HANDLER ENTRY */
       2 SCHED EPTR,                        /* -> DEVICE SCHEDULER */
       2 POST EPTR,                         /* POST-HANDLER ENTRY */
       2 CYLINDERS UBIN(12) UNAL,           /* (DISK) NUMBER OF CYLINDERS */
       2 SURFACES UBIN(6) UNAL,             /* (DISK) NUMBER OF SURFACES */
       2 CYLSIZE UBIN(9) UNAL,              /* (DISK) NUMBER OF SECTORS/CYLINDE
       2 SECTORS UBIN(6) UNAL,              /* (DISK) NUMBER OF SECTORS/TRACK *
       2 PSIA BIT(1),                       /* DEVICE ATTACHED TO PSIA */
       2 MODEL CHAR(8) ALIGNED;             /* MODEL NUMBER (ASCII) */
%MEND;
```

4.5 Channel Table (NI$CHT)

The Channel Table is indexed by the IOM number and the channel number and contains an entry for each possible channel (overhead and payload) in the system. The entry for each configured channel contains a pointer to the Driver Queue Header and an entry pointer to the Poster for that channel. This table will be pointed to by the pointer, NI$CHT$.

```
%MAC NI$CHT(NAME=NI$CHT,STCLASS="(0:63) BASED(NI$CHT$)");
DCL 1 NAME STCLASS ALIGNED,
     0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
    |-----------------|-----------------|-----------------|-----------------
   0|DQH$
    |-----------------|-----------------|-----------------|-----------------
   1|POSTER
    |-----------------|-----------------|-----------------|-----------------


      2 DQH$ PTR,                          /* -> DRIVER QUEUE HEADER */
      2 POSTER EPTR;                       /* POSTER */
%MEND;
```

4.6 Driver Queue Header (NI$DQH)

The Driver Queue Header contains configuration and scheduling information for a particular channel. For multi-device subsystems, the Driver Queue Headers are grouped by Link Adapter; for unit record subsystems, the Driver Queue Headers are grouped by device.

```
%MAC NI$DQH(NAME=NI$DQH,STCLASS=BASED);
DCL 1 NAME STCLASS ALIGNED,
     0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7
    |-----------------|-----------------|-----------------|----------------
   0|FL$
    |-----------------|-----------------|-----------------|----------------
   1|GATE
    |-----------------|-----------------|-----------------|----------------
   2|IOCHAN                    |STATUS      |LA          |STATE       |LOSTINT
    |IOM         |CHANNEL      |            |            |            |
    |-----------------|-----------------|-----------------|----------------
   3|SQH$
    |-----------------|-----------------|-----------------|----------------
   4|DQ$
    |-----------------|-----------------|-----------------|----------------
   5|ERROR
    |-----------------|-----------------|-----------------|----------------

      2 FL$ PTR,                      /* LINK TO NEXT CHANNEL ON LINK ADA
      2 GATE SBIN ALIGNED,            /* GATE ON DRIVER QUEUE */
      2 IOCHAN,
        3 IOM UBIN(6) UNAL,           /* IOM NUMBER */
        3 CHANNEL UBIN(6) UNAL,       /* CHANNEL NUMBER */
      2 STATUS UBIN(6) UNAL,          /* CHANNEL STATUS (UP,DOWN,T&D) */
      2 LA UBIN(6) UNAL,              /* LINK ADAPTER NUMBER */
      2 STATE UBIN(6) UNAL,           /* SCHEDULING STATE */
      2 LOSTINT UBIN(6) UNAL,         /* LOST INTERRUPT FLAG */
      2 SQH$ PTR,                     /* -> SUBSYSTEM QUEUE HEADER */
      2 DQ$ PTR,                      /* HEAD OF DRIVER QUEUE */
      2 ERROR UBIN(36);               /* ERROR STATISTICS */
%MEND;
```

4.7 <u>Subsystem Queue Header (NI$SQH)</u>

For each configured peripheral subsystem, there is a Subsystem Queue Header. It contains configuration and scheduling information for that subsystem. Each Subsystem Table entry consists of two sections: the first is global data, and the second is a table of data for each Link Adapter on the subsystem.

```
%MAC NI$SQH(NAME=NI$SQH,STCLASS=BASED);
DCL 1 NAME STCLASS ALIGNED,
     0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7
    ¦-----------------¦-----------------¦-----------------¦----------------
   0¦GATE
    ¦-----------------¦-----------------¦-----------------¦----------------
   1¦DCT$
    ¦-----------------¦-----------------¦-----------------¦----------------
   2¦NDCT        ¦NLA  ¦LAX   ¦# # # # # # # # # # # # # # # # # # # # # # # #
    ¦-----------------¦-----------------¦-----------------¦----------------
   3¦AQ$
    ¦-----------------¦-----------------¦-----------------¦----------------
   4¦LA
    ¦NCHAN       ¦NACT        ¦STA¦# # # # # # # # # # # # # # # # # # # # # #
    ¦-----------------¦-----------------¦-----------------¦----------------
   5¦
    ¦DQH$
    ¦-----------------¦-----------------¦-----------------¦----------------
   6¦
    ¦ERROR
    ¦-----------------¦-----------------¦-----------------¦----------------
     LEVEL 2 ARRAY:    4 ('   4'O) ENTRIES TOTAL.
    ¦-----------------¦-----------------¦-----------------¦----------------

      2 GATE SBIN ALIGNED,              /* GATE ON SCHEDULING TABLES */
      2 DCT$ PTR,                       /* -> FIRST DCT ENTRY */
      2 NDCT UBIN(6) UNAL,              /* NUMBER OF DEVICES */
      2 NLA UBIN(3) UNAL,               /* NUMBER OF LINK ADAPTERS */
      2 LAX UBIN(3) UNAL,               /* NEXT LINK ADAPTER INDEX */
      2 AQ$ PTR,                        /* HEAD OF ASSIGN QUEUE */
      2 LA(0:3),                        /* LINK ADAPTER TABLE: */
        3 NCHAN UBIN(6) UNAL,           /* NUMBER OF CONFIGURED CHANNELS */
        3 NACT UBIN(6) UNAL,            /* NUMBER OF ACTIVE CHANNELS */
        3 STATUS UBIN(2) UNAL,          /* LINK ADAPTER STATUS (UP,DOWN,T&D
        3 DQH$ PTR,                     /* -> NEXT CHANNEL'S DRIVER QUEUE H
        3 ERROR UBIN(36);               /* ERROR STATISTICS */
%MEND;
```

4.8 IOM Table

The IOM table contains data needed to manage the overhead channels of an IOM. There is a table of pointers to the IOM table packets; this pointer table is indexed by the IOM number.

```
%MAC NI$IOM(NAME=NI$IOM,STCLASS=BASED);
DCL 1 NAME STCLASS ALIGNED,
      0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8 0 1 2 3 4 5 6 7 8
     |-----------------|-----------------|-----------------|-----------------
    0|GATE
     |-----------------|-----------------|-----------------|-----------------
    1|MBX$
     |-----------------|-----------------|-----------------|-----------------
    2|IOM#        |LASTCON     |NEXTF        |NEXTS        |MAXS        |*    |ADDRE
     |-----------------|-----------------|-----------------|-----------------
    3|FLTBUF                               |SPECBUF
     |-----------------|-----------------|-----------------|-----------------

      2 GATE SBIN ALIGNED,                /* GATE ON CONNECT CHANNEL */
      2 MBX$ PTR,                         /* -> MAILBOXES */
      2 IOM# UBIN(6) UNAL,                /* IOM NUMBER */
      2 LASTCON UBIN(6) UNAL,             /* LAST CONNECTED CHANNEL */
      2 NEXTF UBIN(6) UNAL,               /* INDEX TO NEXT WORD IN FAULT BUFFI
      2 NEXTS UBIN(6) UNAL,               /* INDEX TO NEXT WORD IN SPECIAL IN
      2 MAXS UBIN(6) UNAL,                /* WORST CASE NEXTS */
      2 * BIT(3),
      2 ADDRESS UBIN(3) UNAL,             /* IOM ADDRESS (FOR CIOC) */
      2 FLTBUF UBIN(18) UNAL,             /* LOC. OF FAULT BUFFER (IN REAL) *
      2 SPECBUF UBIN(18) UNAL;            /* LOC. OF SPECIAL INTERRUPT BUFFER
%MEND;
```

5. PROCEDURES

5.1 Driver (NIS$DRIVER)

The Driver starts the I/O operation for the first request on a given Driver Queue.

Calling Sequence

```
     CALL NIS$DRIVER(DQH$);
```

where: DQH$ points to a Driver Queue Header.

Entry Conditions

Interrupts must be disabled.
The Driver Queue must not be locked.
The channel must not be busy.

Exit Conditions

The I/O is started.
The Driver Queue is not locked.

Description

The PCW (Peripheral Control Word) is set up from the PCW and PTP fields of the request packet and the channel number of the Driver Queue Header. The payload channel mailbox is set up to point to the DCW list and status doubleword of the request packet. The CIOC is then issued with the connect channel mailbox pointing to the PCW. Since the Driver may be called by the device Scheduler or the device Poster, it must be able to run with interrupts enabled or disabled. The Driver does not remove the first request from the Driver Queue; it is removed by the device Poster.

5.2 Interrupt Distributor (NIS$INTDIS)

The Interrupt Distributor determines the interrupting channel and calls the associated Poster.

Calling Sequence

The Interrupt Distributor is called by the wired-in CLIMB instruction to the entry descriptor at location 30(octal). This entry descriptor must be type 11 and must specify the Monitor's Linkage Segment, the Monitor's privileged Instruction Segment, and WSR (Working Space Register) number 0.

Description

The Interrupt Distributor determines the interrupt level and the interrupting channel number from the Interrupt Cell number in the Safe-Store Frame and the corresponding IMW (Interrupt Mask Word). If the interrupt was from the system fault channel or the special status channel, the fault word or special status word, respectively, is examined to determine the associated payload channel number. The fault word or special status word is altered to look like a peripheral status word by inserting a major status code of 17(octal) for a fault interrupt or 16(octal) for a special interrupt. The fault word or special status word is passed to the device Poster along with the Driver Queue Header of the interrupting payload channel and the interrupt level. The device Poster for the payload channel is then called to process the interrupt. On return from the device Poster, the Interrupt Distributor loops back to process any other interrupts that were indicated in the current IMW. When all of them have been processed, the Interrupt Distributor returns to the interrupted routine.

5.3 Lost Interrupt Poller (NIS$LOSTINT)

The Lost Interrupt Poller is called periodically to check for lost interrupts.

Calling Sequence

```
     CALL NIS$LOSTINT;
```

Entry Conditions

Interrupts must be enabled.
No driver queues may be locked.

Exit Conditions

Interrupts will be enabled.
No driver queues will be locked.

Description

The Lost Interrupt Poller is called periodically, say, every 5-10 seconds. Each time it is called, it scans all of the channels' Driver Queue Headers. For each channel, if the Lost Interrupt Flag is not set and the channel is active, the Lost Interrupt Flag is set. If the Lost Interrupt Flag is already set, a lost interrupt condition has occurred, since the channel has been active since the previous polling cycle. (The Lost Interrupt Flag is cleared by the Driver when starting a request and by the Interrupt Distributor when a termination interrupt occurs.) The lost interrupt is reported to the device handler by calling the device Poster with a unique interrupt level. Interrupts are disabled while each channel is being processed.

5.4 Lock Gate Routine (NIS$LOCK)

The Lock Gate routine is a fast assembly-language routine for locking a gate.

Calling Sequence

```
     CALL NIS$LOCK(GATE);
```

where: GATE is the gate to be locked.

Entry Conditions

GATE must be a word-aligned 36-bit field.

Exit Conditions

The gate is locked.

Description

A gate is used to synchronize code sequences running in two or more CPUs. The gate is locked when its value is zero. When a gate has been locked by one CPU, it cannot be locked by any other CPU. This routine locks a gate by using a Load Accumulator and Clear (LDAC) instruction. If the gate is already locked, NIS$LOCK loops on the LDAC until it is unlocked. If the gate is locked for an excessive length of time (more than 16 ms.), the system will crash with a lockup fault. The gate may be unlocked merely by storing a non-zero value into it.

5.5 Queue Handling Macros

These PL-6 macros are provided for enqueuing and dequeuing packets from FIFO queues.

Calling Sequences

To enqueue a packet at the tail of a queue:

```
%ENQUEUE(P#=P$,Q#=Q$);
```

To enqueue a packet at the head of a queue:

```
%REQUEUE(P#=P$,Q#=Q$);
```

To dequeue a packet from the head of a queue:

```
%DEQUEUE(P#=P$,Q#=Q$);
```

where:

P$ points to the packet to be queued, or, on return, points to the packet dequeued.
Q$ points to the queue header.

Entry Conditions

None

Exit Conditions

ENQUEUE
REQUEUE

The specified packet has been linked into the queue.

DEQUEUE

The first packet on the specified queue has been removed from the queue and returned in P$. If the queue is empty, P$=ADDR(NIL) is returned.

Description

If the queue is empty, ENQUEUE and REQUEUE set the queue header to point to the new packet, and set the link field of the packet to point to itself. If the queue is not empty, ENQUEUE inserts the new packet after the packet pointed to by the queue header, then sets the queue header to point to the new packet. REQUEUE inserts the new packet after the packet pointed to by the queue header but does not change the queue header.

If the queue is empty, DEQUEUE returns P$=ADDR(NIL). If not, it unlinks the packet following the packet pointed to by the queue header and returns it to the caller. If this is the last packet on the queue, the queue header is set to ADDR(NIL).

5.6 Device Scheduler (NIx$SCHED)

The device Scheduler enqueues the device on the Assign Queue, assigns a channel to the device, and schedules a list of requests to be executed.

Calling Sequence

```
CALL NIx$SCHED(DCT$);
```

where:
DCT$ points to the Device Control Entry for the device.

Entry Conditions

Interrupts are enabled.
The Subsystem Tables and Driver Queue are not locked.

Exit Conditions

If a channel was available, I/O has been started.
Interrupts are enabled.
The Subsystem Tables and Driver Queue are not locked.

Description

On entry to the device Scheduler, if there are requests on the device's Scheduler Queue and the device is not busy, it is enqueued on the Assign Queue. Then, if there is an available channel, the first device on the Assign Queue is removed and assigned to the next available channel, and requests from the device's Scheduler Queue are moved to the channel's Driver Queue. For most devices, only one list of requests will be moved to the Driver Queue at a time. However, in order to minimize disk rotational latency, all of the disk requests for a cylinder will be moved to the Driver Queue as a group. The Driver is then called to start the first request. On return from the Driver, the device Scheduler loops back to process another channel. When there are no more available channels or no entries on the Assign Queue, the Device Scheduler is done.

5.7 Device Poster (NIx$POSTER)

The device Poster analyzes interrupts, calls the Driver to restart the channel, performs error recovery, and returns completed requests to IOQ for end-action processing.

Calling Sequence

```
CALL NIx$POSTER(DQH$,LEVEL,STATUS);
```

where:
DQH$ points to the Driver Queue Header of the interrupting payload channel.
LEVEL is the interrupt level:

0	overhead
1	termination
2	marker
3	special

STATUS is the fault word or special status word.

Entry Conditions

Interrupts are disabled.
The Driver Queue is not locked.

Exit Conditions

Interrupts are disabled.
The Driver Queue is not locked.

Description

The device Poster is called by the Interrupt Distributor to process the interrupt. If this is a termination interrupt, the current request is removed from the Driver Queue. If there were no errors, the Driver is called to start the next request on the Driver Queue. Marker interrupts are ignored; this interrupt level is included for the Front End (Direct Channel) Handler. If this is a special interrupt, i.e., an asynchronous interrupt, it may be used by error recovery routines; otherwise it will be passed to IOQ for a possible caller of special interrupt control. If this is a fault interrupt, the fault word is stored in the second status word of the current request.

If there were no errors or the errors were unrecoverable, the completed request is returned to IOQ for end-action processing. If a recoverable error occurred, the appropriate error recovery technique is applied. To retry the operation, the device Scheduler is called. If repositioning is required, reserved request packets are used to effect the repositioning.

6. IOQ INTERFACES FOR IOS

The following entry points to IOQ are used only by IOS.

6.1 I/O Completion Routine (NIO$COMP)

NIO$COMP performs all caller- and user-related actions related to I/O completion.

Calling Sequence

```
     CALL NIO$COMP(Q$);
```

where: Q$ points to the request.

Entry Conditions

Interrupts are enabled.
No gates are locked.

Exit Conditions

The request packet has been either returned to the caller, requeued for a new operation, or released.

Description

From the hardware status words, NIO$COMP sets the completion code, actual record size, and actual record count fields of the IOQ Request Packet and releases the IOS Request Packet. Then, depending on the setting of bits in the request's OPFLG field, NIO$COMP may do one or more of the following:

- Set completion information in the user's DCB.
- Report an "I/O Completion" event to the Job Scheduler.
- Call an end-action routine. (Note: an end-action routine may not request any resources, such as request packets.)
- Requeue the IOQ Request Packet for a New I/O operation.
- Release the packet.

6.2 Special Interrupt Completion Routine (NIO$SPCOMP)

If a caller has requested special interrupt control, NIO$SPCOMP calls his end-action routine.

Calling Sequence

```
     CALL NIO$SPCOMP(DCTX,STATUS);
```

where:
DCTX is the Device Table Index of the interrupting device.
STATUS is the special interrupt hardware status word.

Entry Conditions

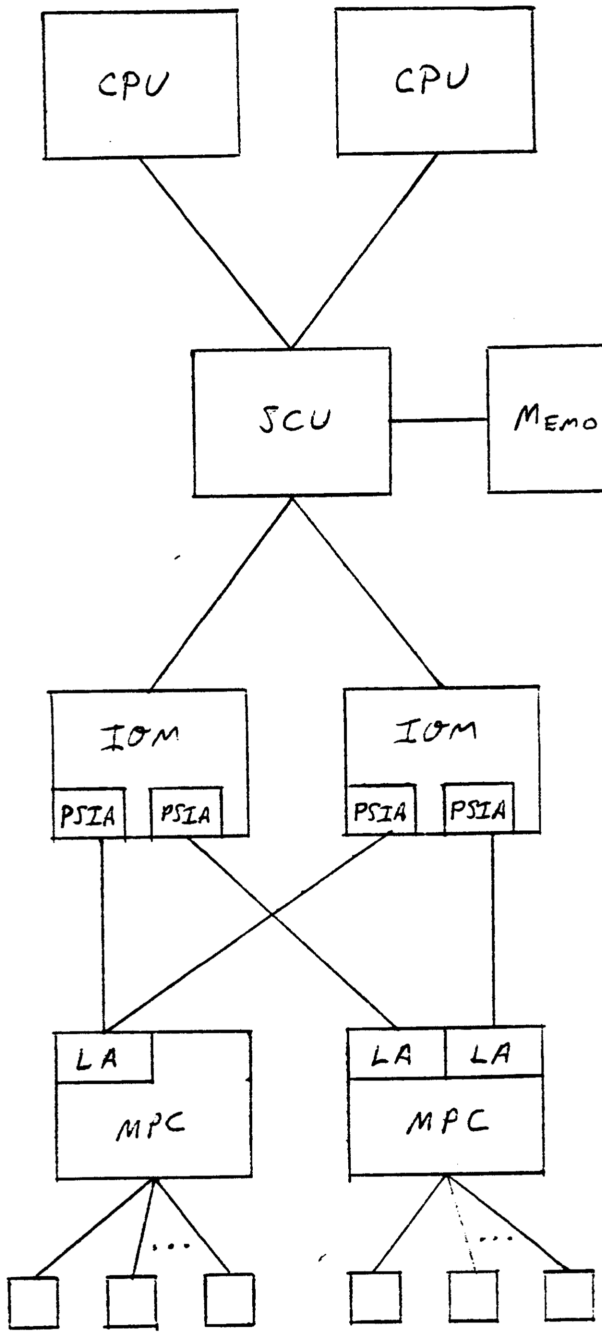
Interrupts are enabled.
No gates are locked.

Exit Conditions

If this special enterrupt has been requested, the caller's end-action routine has been called.

Description

NIO$SPCOMP is called for every special interrupt that is not used by the device Poster. It checks the Device Table to see if this special interrupt has been requested. If so, the requestor's end-action routine is called with the hardware status word. (Note: an end-action routine may not request any resources, such as request packets.)



Peripheral Devices

Fig. 2.1

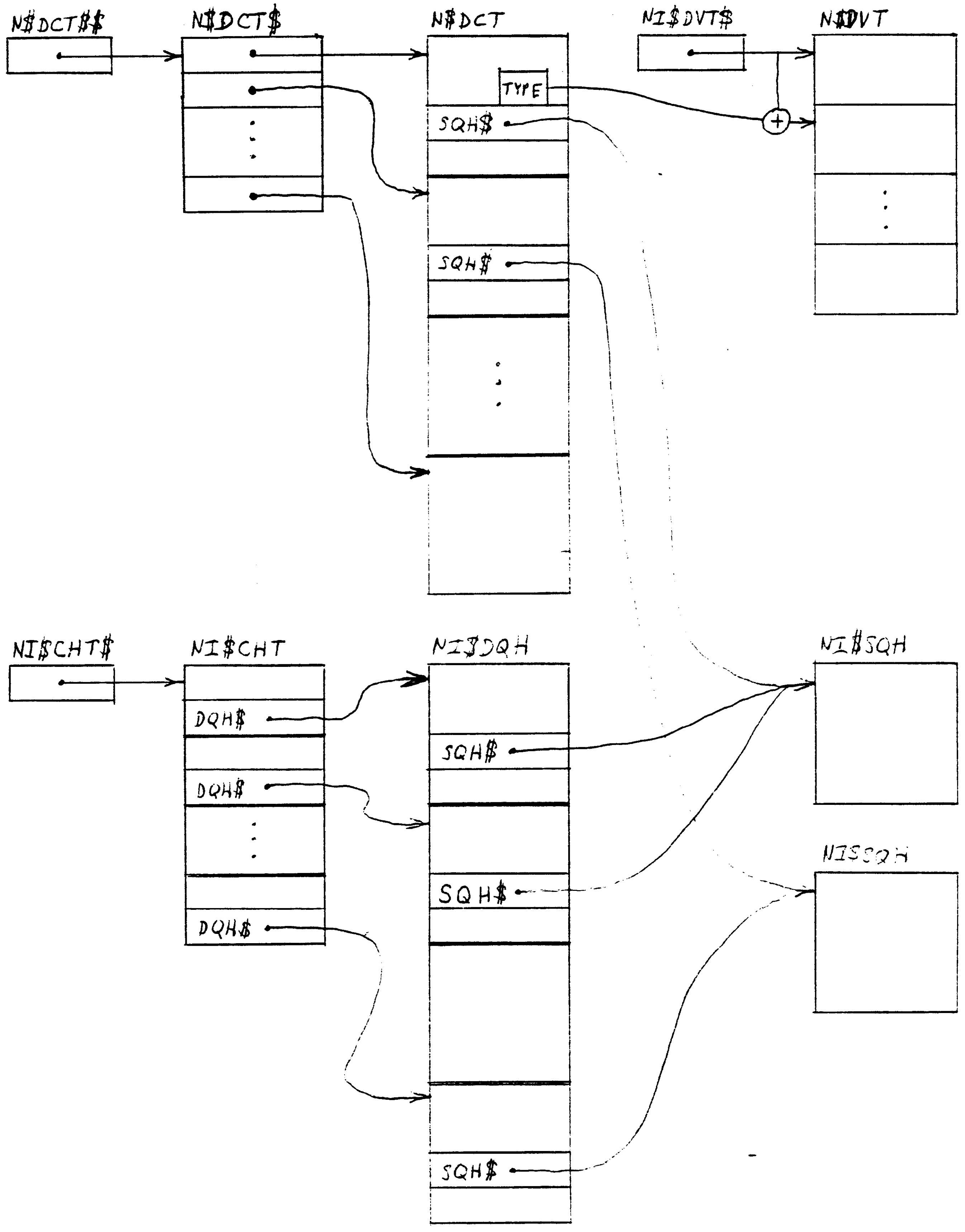

N$DCT$$
NDCT
N$DCT
TYPE
SQH$
SQH$
NI$DVT$
N$DVT
+
NI$CHT$
NI$CHT
DQH$
DQH$
DQH$
NI$DQH
SQH$
SQH$
SQH$
NI$SQH
NI$SQH

Fig. 3.1

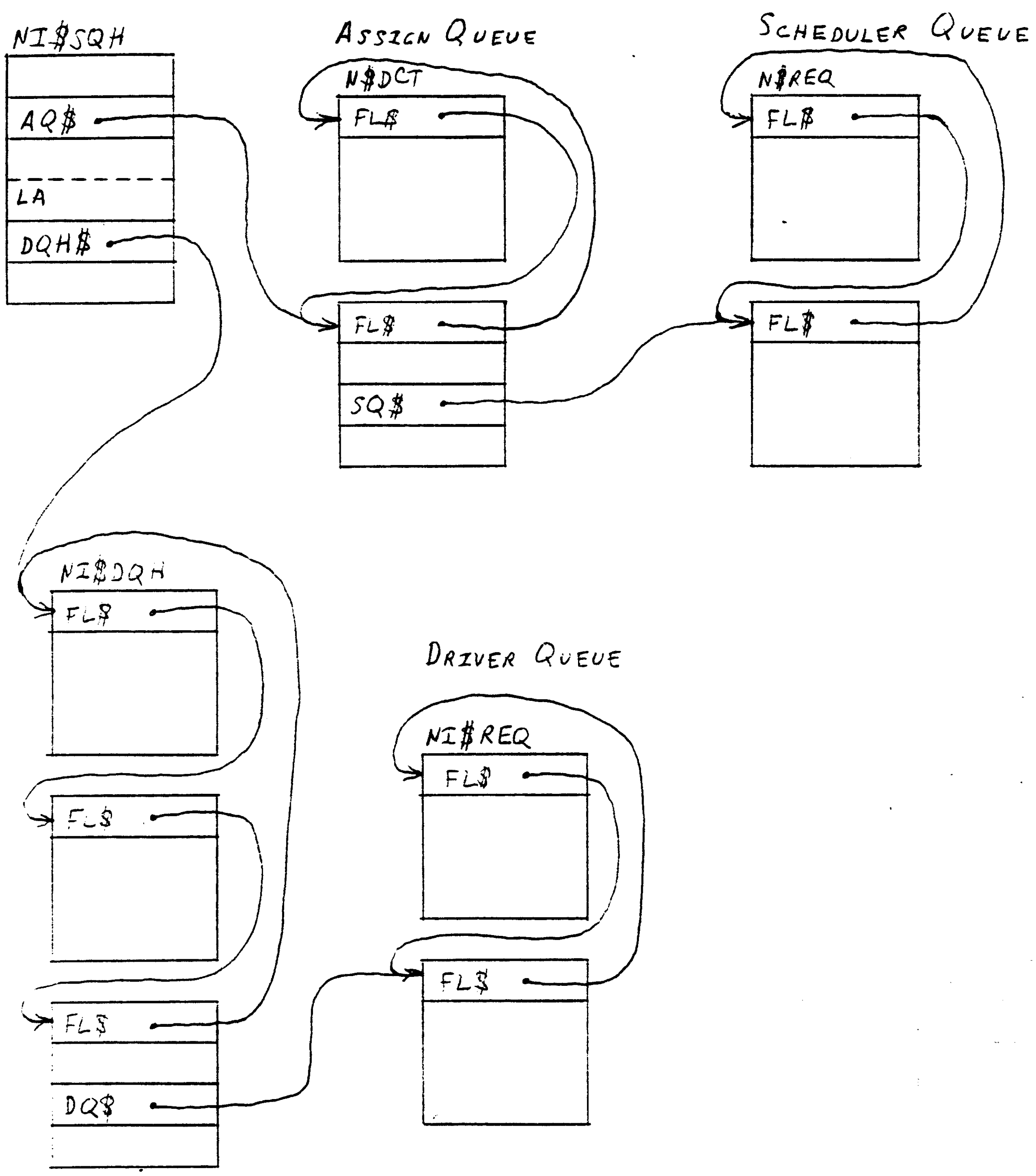

NI$SQH
AQ$
LA
DQH$
Assign Queue
N$DCT
FL$
FL$
SQ$
Scheduler Queue
N$REQ
FL$
FL$
NI$DQH
FL$
FL$
FL$
DQ$
Driver Queue
NI$REQ
FL$
FL$

Fig. 3.2