HP 93695V Synchronous Mux

Z80 Software Specification

Revised <851019.2258>

Table of Contents

Financial Network Services Pty. Ltd.
544 Botany Road
Alexandria
Sydney, Australia. 2016

Tel. (02) 693-1244

Introduction — SECTION 1

1.1 Scope of Software Specification

The software specication gives a detailed description of the Z80 and RTE.A driver software. It assumes a good knowledge of both the Z80 and HP A-Series architechture as well as a reasonable understanding of datacommunications in general, and RS232 and some protocol standards in particular.

1.2 Overview

The 93695V sync mux is a special product designed as an enhancement to the standard 12040B async mux. Like the 12040B it supports 8 channels and all the standard async features, however it has the following extra features:-

* 64Kb of RAM memory (16Kb on 12040B)
* Sync clock signals (TXC, RXC)
* Modem control signals (RTS, CTS, DCD, DTR)
* Zilog SCC controllers to replace SIO/2
* SCC gives extra features:-
    1. Baud rate generator per port
    2. NRZI, FM0, FM1 data encoding modes
    3. SDLC loop mode
    4. Auto echo and local loopback
    5. Digital Phase Locked Loop for clock recovery

* 4MHz Z80A on prototype (3.68MHz on 12040B)

* Fast EPROM (200ns) no EPROM wait states (As on 12040B)

The other features of the SCC are similar to the SIO/2, however because the clock signals and the modem control signals are now brought out to the RS232C panel the 93695V is considerably more functional than the 12040B.

The 93695V mux is only availible for A-Series and L-Series computers and does not have a version for the MXM and MXE models of the HP 1000 Family.

The 93695V supports these features:-

RS232C Signals

* TXD	2	-	Transmit Data
* RXD	3	-	Receive Data
* RTS	4	-	Request To Send
* CTS	5	-	Clear To Send
* DCD	8	-	Data Carrier Detect
* DTR	20	-	Data Terminal Ready
* TXC	15	-	Transmit Clock
* RXC	17	-	Receive Clock
* TTC	24	-	Transmit Timing Clock

Asynchronous features

* 50 - 9600 Baud on 8 ports, up to 19200 on 4 ports

* 5 - 8 Bits per character

* 1, 1.5, 2 Stop bits per character

* Parity gerneration and checking

* Break generation and detection

* Overrun detection

* Programmable clock factor (X16, X32, X64)

Synchronous Byte Level features

* 50 - 9600 Baud on 8 ports, 19200 on 4, 38400 on 2
* Bisync or Monosync
* 5 - 8 Bits per character
* Parity generation and detection
* Automatic sync insertion and deletion
* CRC-16 or CRC-CCITT generation and checking

Synchronous Bit Level Features

* 50 - 9600 Baud on 8 ports, 19200 on 4, 38400 on 2
* Automatic zero insertion and deletion
* CRC generation and checking
* Address search mode
* Abort generation and dectection
* SDLC loop mode
* DPPL on NRZI, FM0 and FM1 modes

For further information of the SCC communication features see the Zilog SCC Z8530 technical manual (Zilog Part No. 00-2057-02). Other manuals that are refered to are:-

Z80A CPU Technical Manual	Zilog Part	03-0029
Z80A CTC Technical Manual	Zilog Part	03-0036
Z80A DMA Technical Manual	Zilog Part	00-2013
Z80A SIO Technical Manual	Zilog Part	03-3033
HP 93695V 8-channel Sync Mux	HP Part	93695-90003
HP PSI Development Manual	HP Part	24602-90001

1.3 Configuration

The 93695V sync mux card is composed of the following major subsystems:-

* A-series I/O backplane interface

* 93695V mux backplane
* Z-80A component family
* Memory
* Modem control
* Communication interface

The detailed hardware description can be found in the HP hardware manual for the 93695V mux (HP P/N 93695-90003)

Z80 Software Modules — SECTION 2

2.1 General

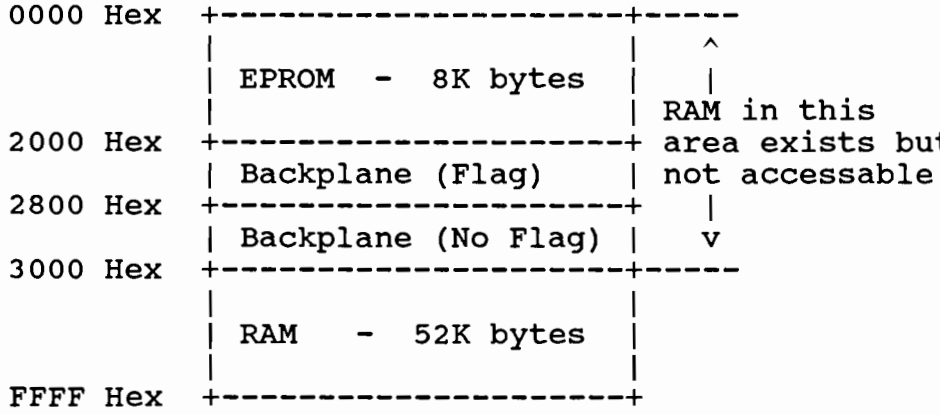
The Z80 subsystem is an independent entity to the A-series host processor, executing its own software asynchronously to the RTE.A operating system. This chapter describes the structure of the software that controls the Z80.

Although independant, the Z80 software is implemented to make it a slave processor to the A-series host. The driver in the A-series host issues commands that control and synchronize the execution of the Z80 so that it can correctly transfer data to and from the datacom subsystem.

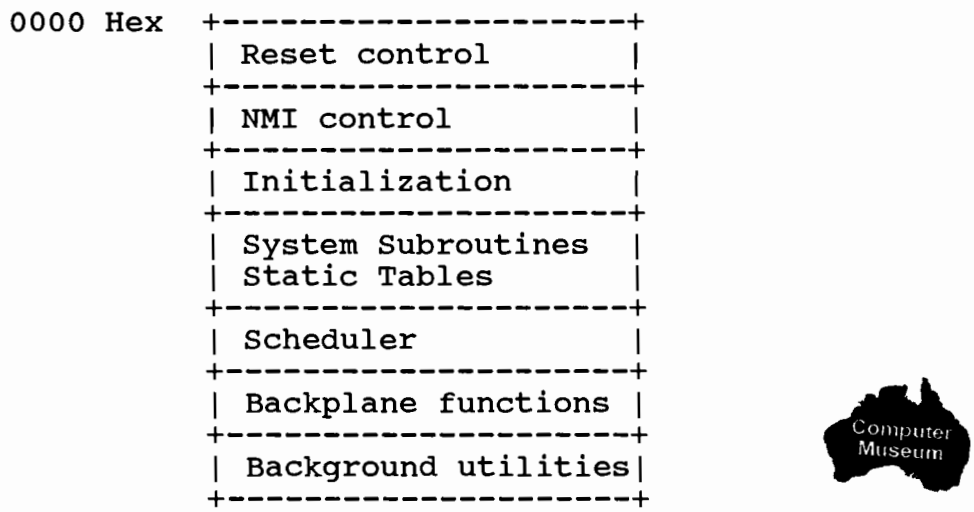
2.1.1 Memory Organization

The memory of the Z80 subsystem has three distinct areas, EPROM, Memory Mapped Backplane Space, and RAM. All the Z80 program and data areas exist within the EPROM and RAM. The EPROM contains the base executive control, some subroutines, and read only system tables. The RAM contains code that has been downloaded from the host A-series, dynamic system tables and data buffers.

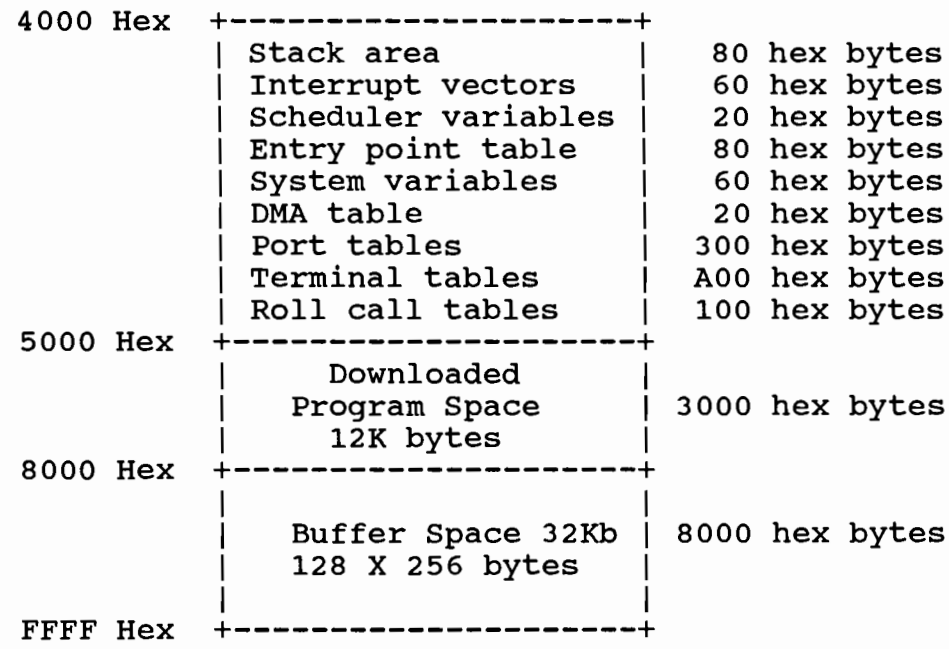
Memory Layout:-



Software in EPROM



Software In RAM



2.2 Interrupt Handling

There are 3 levels of software execution within the structure of the

Revised <851019.2258>

datacom subsystem.

* NMI interrupt level Highest priority
* Maskable interrupt level
* Background task level Lowest priority

The NMI interrupts are generated by the A-series interface driver. The maskable interrupts are generated by the SCC chips, the DMA chip, and the CTC timing channels. These interrupts can set event flags that the the scheduler recognises and uses to initiate the appropriate background tasks.

The time taken to service an interrupt is critical to the performance of the datacom subsystem. The target performance is to be able to run 8 ports at 9600 baud full duplex. This will give an aggregate throughput of 19,200 characters per second, or 19,200 interrupts per second.

In order to be able to achieve this performance the average interrupt handling time must be less than 50 microseconds (uS). The primary goal of the design of the datacom subsystem must be to meet this requirement. The SCC ports are buffered for both transmit and receive, and can tolerate at least a 1 character time delay in servicing an interrupt. This allows such events as NMIs and end of buffer servicing to take extra time (up to 100uS), provided they do not significantly increase the average time.

The Z80 DMA (connected to the backplane) degrades the Z80 CPU performance when it is operating. However one of the CTC channels (2) is used as a DMA pacer limiting its effect on the CPU to an acceptable level (??%).

The interrupt priority of the Z80 devices is SSC ports 6,7,4,5,2,3,0,1 then DMA and then CTC. The internal interrupt handling of these devices is covered in the appropriate technical manual.

2.3 Scheduler

The Scheduler is used to control the priority and order of execution of the background tasks. It is implemented as a sequential byte table of events in which the bits in any particular byte may have significance to the corresponding event.

Scheduler Event Table .....

Highest Priority								
NMI event	S	-	-	-	-	-	-	-
DMA completion ..	S	-	-	-	-	-	-	-
Backplane busy ..	S	-	-	-	-	-	-	-
IRQ pending	S	-	-	-	-	-	-	-
Timer (10mS)	S	-	-	-	-	-	-	-
RX I/O completion	7	6	5	4	3	2	1	0
TX I/O completion	7	6	5	4	3	2	1	0
New I/O	S	-	-	-	-	-	-	-
Poll completion .	7	6	5	4	3	2	1	0
Lowest Priority								

When the scheduler is it entered always starts at the top and scans down the list until it finds a byte that is not zero. It then schedules the corresponding task. If no events are found then the scheduler resumes scaning at the top again. Thus, when the system is idle it will spend all its time in the schedlue loop waiting for some event to happen.

2.4 Backplane Processing

The Backplane processing manages the interface between the A-series host and the Z80 datacom subsystem.

2.4.1 Backplane Transactions

In order to provide the funcionality required a set of backplane transactions are defined to control the flow of commands and data across the backplane. These transactions are divided into two groups, immediate and deffered. The immediate commands are executed at the NMI interrupt level and are generally fast to execute (20-30us) although the RESET and DOWNLOAD commands are execeptions. The deferred commands set a flag from the NMI interrupt level and are processed by a background task. These will typically take 1 to 10 ms to process.

Possible backplane transactions:-

Symbol	Opcode	Type	Description .....
$NOP	0	I	No Operation (Interrupt possible)
$RSET	1	I	Software reset the I/O card
$PORT	2	D	Setup Port parameters

Revised <851019.2258>

$TERM	3	D	Setup Terminal parameters
$	4		.
$	5		.
$RUN	6	I	Run/Halt Z80 program
$	7		.
$DWNL	8	I	Download/Unload Data
$	9		.
$STDT	A	D	Transfer Status table (Z80 to HP)
$RXDT	B	D	Transfer Read data (Z80 to HP)
$TXDT	C	D	Transfer Write data (HP to Z80)
$EIRQ	D	I	Enable IRQs
$DIRQ	E	I	Disable IRQs
$FIRQ	F	D	Fetch Next IRQ

D - Defered, I - Immediate

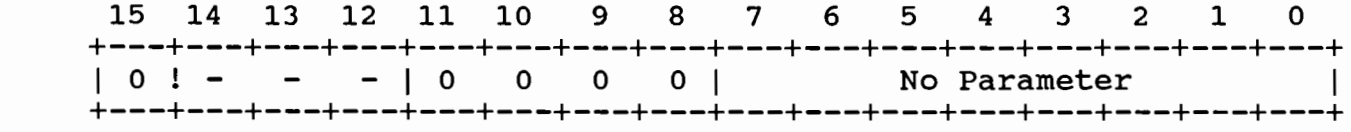
2.5 Backplane Transaction Formats

The backplane commands follow a specific format. Byte 1 is used to indicate the command type in bits 11 to 8. Bit 15 indicates that this command is a deferred NMI command to be fully executed by the background task. The NMI level processing required to set the deferred NMI flag takes about 20 microseconds to process. Bits 14 to 12 are only used by the deferred NMIs to indicate the number of words (2 bytes) following the command word.

2.5.1 $NOP Transaction

This command has no effect except to solicit a response from the card indicating that it has recognised the command. This can be used to generate an interrupt entry to the interface driver by issuing the command and then not waiting for the response.

Request Format:-



Reply Format:-

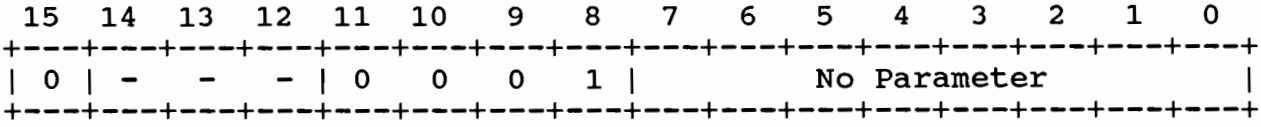
All zeros (After aprox 25uS at 4Mhz Z80 Clock)

2.5.2 $RSET Transaction

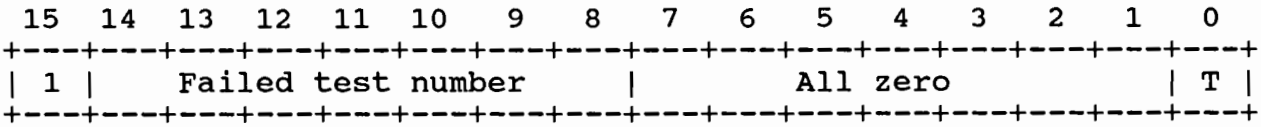
This command causes a software reset and initialization to be done by the card. Bit15 is set in the reply word to indicate a reset. This is the only response with Bit15 set and is used as an indicator to the interface driver that a reset has occured. This can occurr unexpectedly after a power fail or after a software crash on the card.

The initialization of the card sets the system tables and devices to a default state. It initializes the interrupt vector table to point to interrupt routines that passively handle any unexpected interrupts (however the devices are initialized so as not to cause any interrupts).

Request Format:-



Reply Format:-



T - Self test indicator

The initialization of the card performs some selftest functions that check the basic operation of the card. When the system is powered up or when the %T test command is executed from the A-series VCP the initialization code of the 93659V is entered. Bit0 of the reply is used to indicate if this card has passed its internal tests. If successful both Bit0 and the test number are zero, otherwise Bit0=1 and the test causing the failure is in the test number.

Test	-	Description of test
1	-	Z80 CPU (and Part of EPROM)
2	-	EPROM checksum
3	-	RAM test (Inverts memory and inverts back)
4	-	Z80 CPU instructions
5	-	CTC test

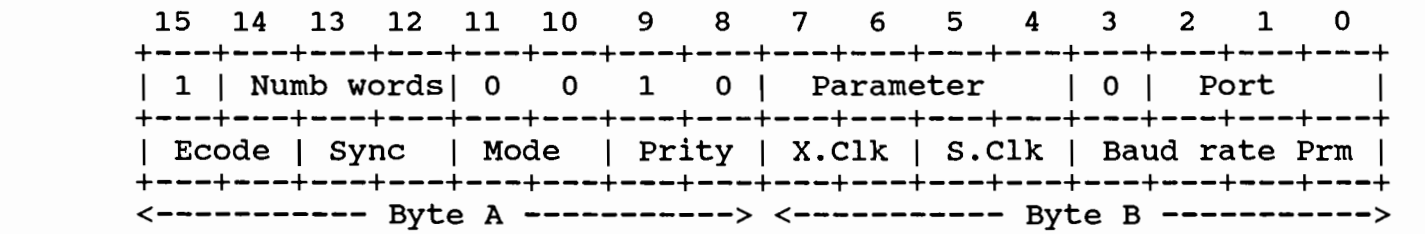
Revised <851019.2258>

```
 6    -    DMA test
 7    -    SCC  0 port A
 8    -    SCC  0 port B
 9    -    SCC  1 port A
10    -    SCC  1 port B
11    -    SCC  2 port A
12    -    SCC  2 port B
13    -    SCC  3 port A
14    -    SCC  3 port B
```

2.5.3 $PORT Transaction

This command is used to set the configuration for each of the SCC ports on the card.

Request Format:-

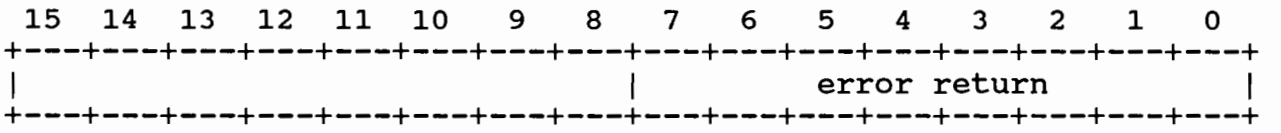



```
Param   -  Parameters initialized

  0     -  Set both parameters
  1     -  Set configuration mode (Byte A)
  2     -  Set Baud rate (Byte B)

Ecode   -  Encoding modes

 0 0    -  NRZ
 0 1    -  NRZI
 1 0    -  FM1 (transition = 1)
 1 1    -  FM0 (transition = 0)

Sync    -  Sync mode select

 0 0    -  8 Bit sync (monosync)
 0 1    -  16 Bit sync (bisync)
 1 0    -  SDLC Mode
 1 1    -  External sync

Mode    -  Operating mode

 0 0    -  Sync mode
```

```
 0 1   -   Async 1 stop
 1 0   -   Async 1.5 stop bits
 1 1   -   Async 2 stop bits

Prity  -   Parity select

 0 0   -   No Parity
 0 1   -   Odd parity
 1 0   -   No Parity
 1 1   -   Even parity

X.Clk  -   Clock multiplier

 0 0   -   X 1
 0 1   -   X 16
 1 0   -   X 32
 1 1   -   X 64

S.Clk  -   Clock source

 0 0   -   External clock
 0 1   -   Internal clock from BRG
 1 0   -   Not used
 1 1   -   DPPL output (must use BRG as source)
```

Baud	-	Rate	Baud	-	Rate
0000	-	150	1000	-	7200
0001	-	300	1001	-	9600
0010	-	600	1010	-	14400
0011	-	1200	1011	-	19200
0100	-	1800	1100	-	38400
0101	-	2400	1101	-	48000
0110	-	3600	1110	-	57600
0111	-	4800	1111	-	76800

Reply Format:-




```
      Error code  = 0    request processed
                  = 1    port number out of range
                  = 2    bad parameter specified
                  = 3    some terminals in this port not disabled
                  = 4    bad baud rate
                  = 5    illegal clock source
```

Revised <851019.2258>

2.5.4 $TERM Transaction

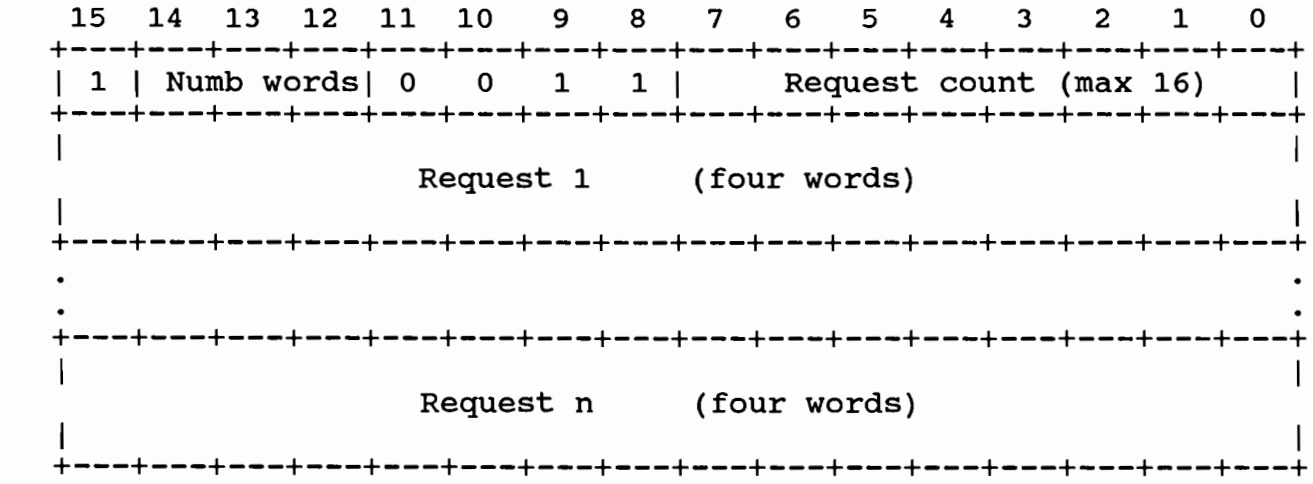
This command is used to set parameters and send commnads to the terminals. It is a deffered command with a variable number of fixed length requests per transaction. Each request consists of a fixed length of four words, even though not all request types use all the fields. The request count field gives the number of rquests in a transaction and is limited to 16 maximum.

The response to this command is also variable length and includes one status byte per request in the original transaction.

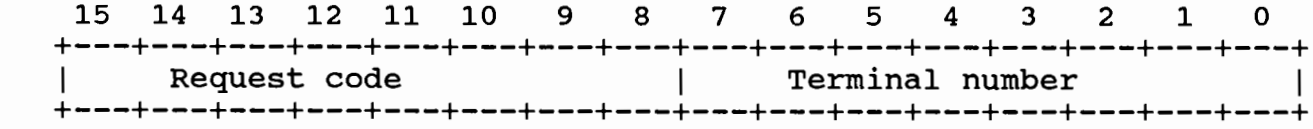
Request codes:-

6	-	Set terminal parameters
7	-	Enable terminal
8	-	Disable terminal
9	-	Activate polling
10	-	Deactivate polling

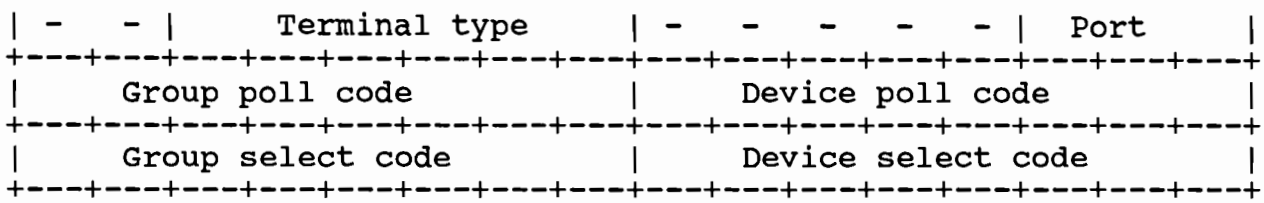
Transaction Format:-



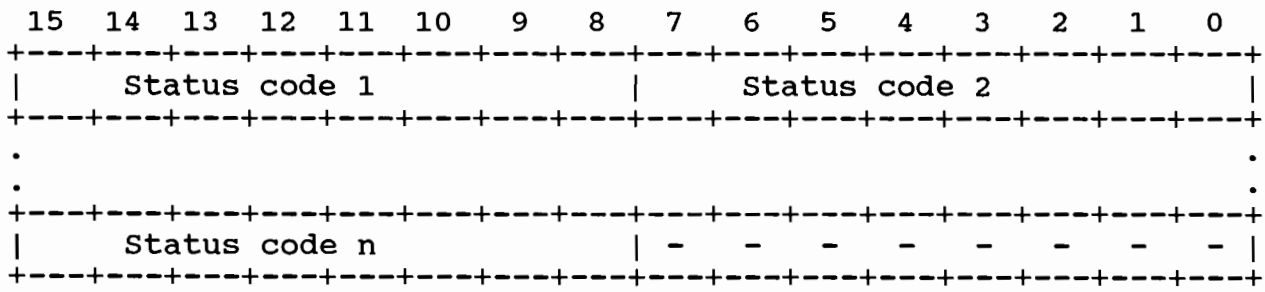
Request Format:-



Revised <851019.2258>



Reply Format:-



Status codes:-

0 - Request processed
1 - Terminal must be disabled
2 - Roll call table full
3 - Invalid terminal number
4 - Invalid request code
5 - Terminal not assigned to port
6 - Illegal terminal type (protocol not loaded)
7 - Invalid port number
8 - Group not disabled (group poll only)
9 - New group must be disabled (group poll only)

The set terminal paramaters request is used to configure a terminal to a port or to re-confiure a terminal to a different port. The terminal must be disabled before this request can be processed. The terminal type is checked by the firmware to ensure that the appropriate protocol is loaded.

The terminal enable, terminal disable, and activate and deactivate polling are not actioned immediately by the firmware. Each of these requests is queued as a new I/O event by setting the appropriate bit in TMNFLG and adding the terminal to the new I/O queue if this byte was previously non-zero.

In the case that the opposite request is pending (e.g. the TMFENB bit is set and the current request is to disable the terminal), the firmware will reset the opposite request pending bit (TMFENB in the example) and set the new request bit. Note that in the example it

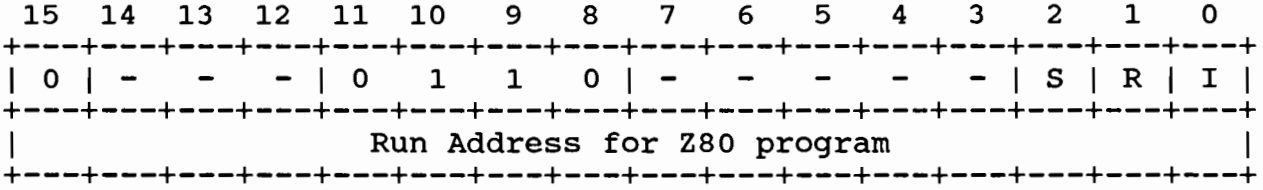
would not be necessary to queue the terminal on the new I/O queue, since it would already be queued.

2.5.5 $RUN Transaction

This command is used to RUN or HALT the program in the Z80 subsystem. It is normally used after code has been downloaded to the RAM to start the new program. This is an immediate command executing from the NMI interrupt level.

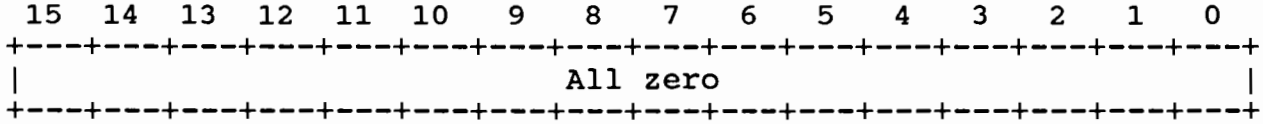
The HALT option disables all peripheral chips and halts with interrupts disabled.

Request Format:-



S - S=0 run scheduler, S=1 run at run address(RUN only)
R - R=0 Halt, R=1 Run
I - I=0 Interrupts disabled, I=1 Interrupts enabled (RUN onl?

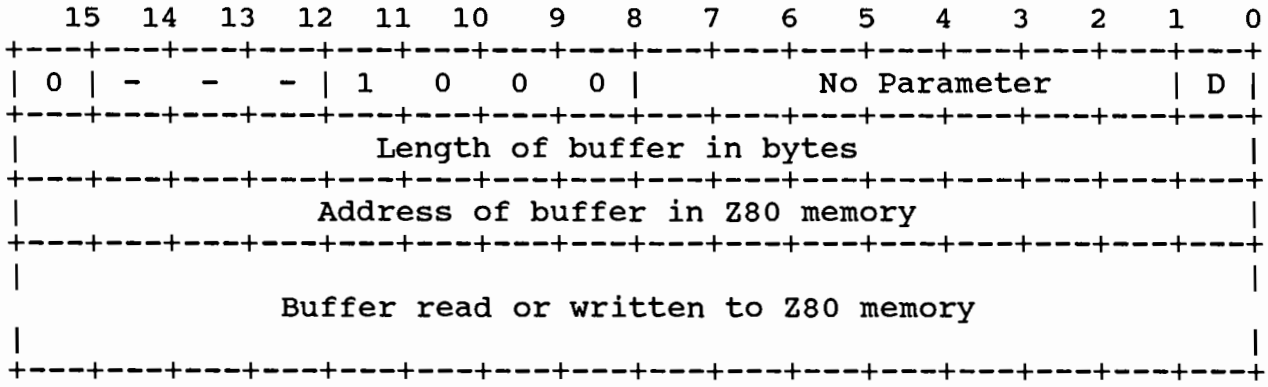
Reply Format:-



2.5.6 $DWNL Transaction

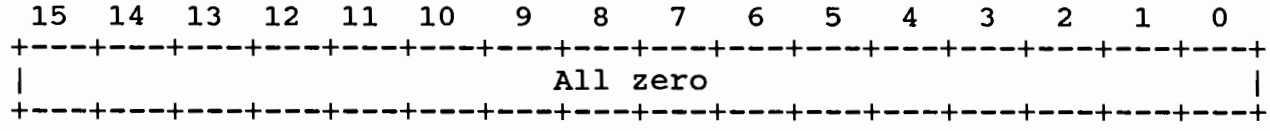
This command is used to download or upload data buffer to the Z80 memory. It is executed as an immediate command at NMI interrupt level. However because of the length of time required (up to 5ms) this command should not be used by the RTE.A interface driver. It is intended to be directly addressed by the download program before the Z80 card is activated.

Request Format:-



D - D=0 Write to card, D=1 Read from card

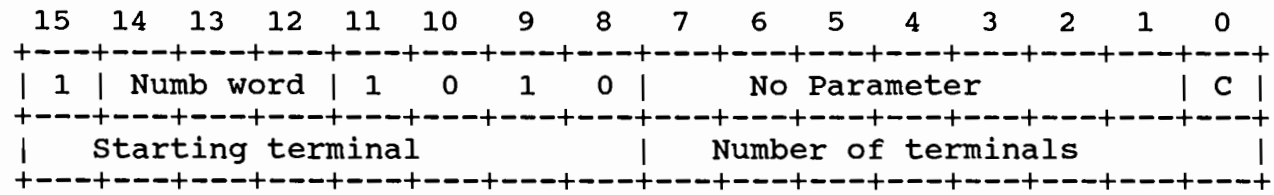
Reply Format:-



2.5.7 $STDT Transaction

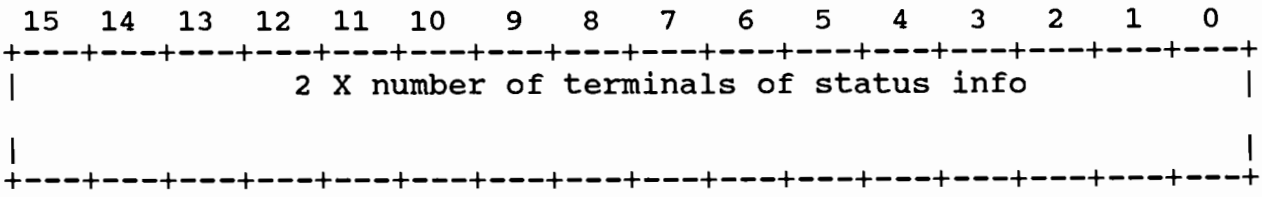
This command is used to read the terminal status information from the Z80 card back to the main system. It transfers the first four bytes from each terminal table. It has the option of clearing the error counter after reading the each table.

Request Format:-



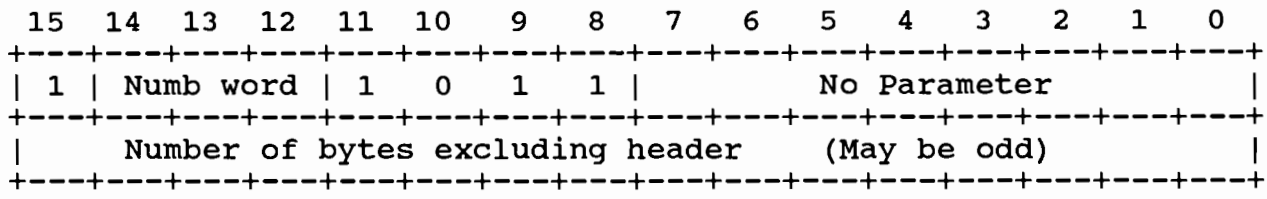
C - C=1 clear the error counter after read

Reply Format:-

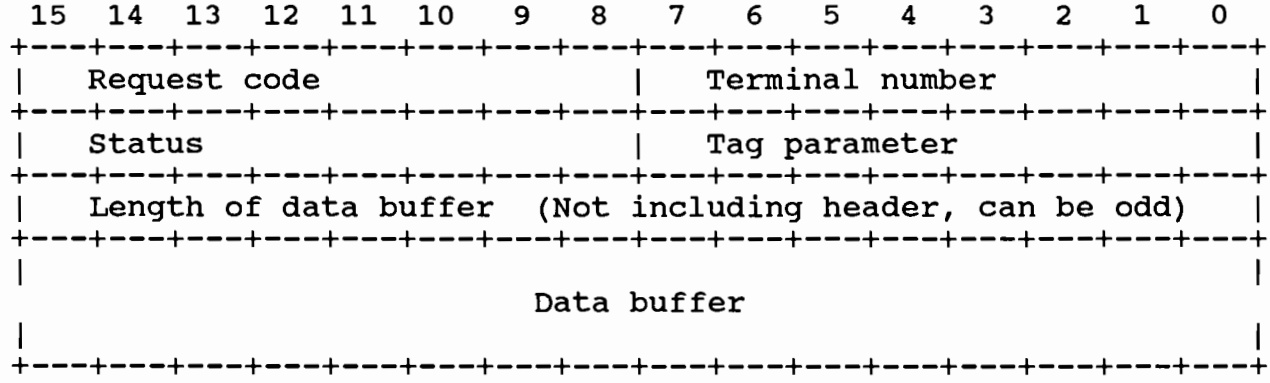


2.5.8 $RXDT Transaction

Request Format:-



Reply Format:-

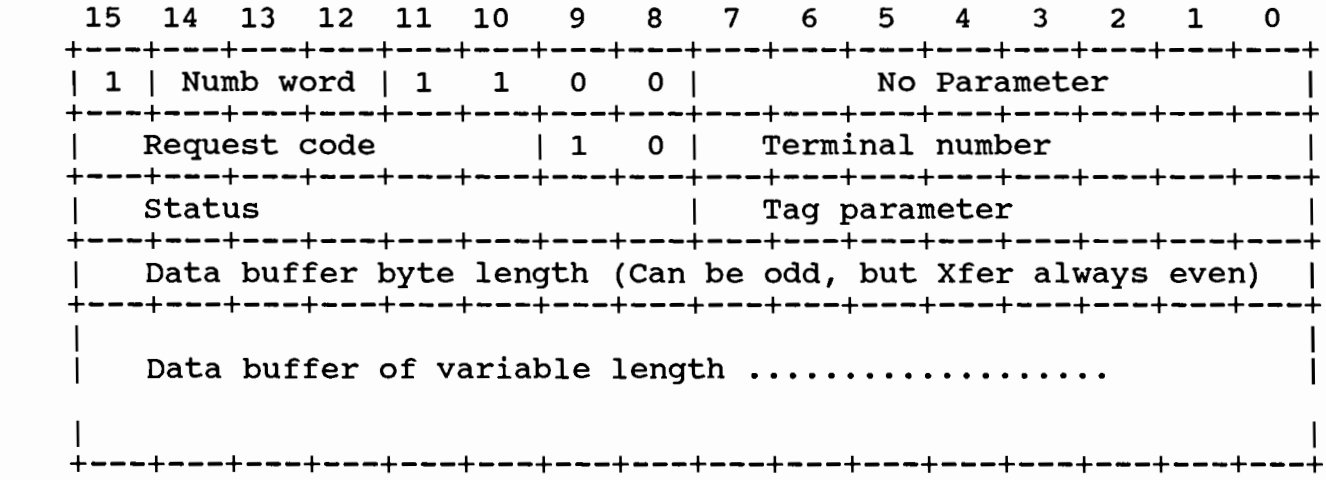


The length of the transfer will always be an even number of bytes and will include an extra 3 words for the message header.

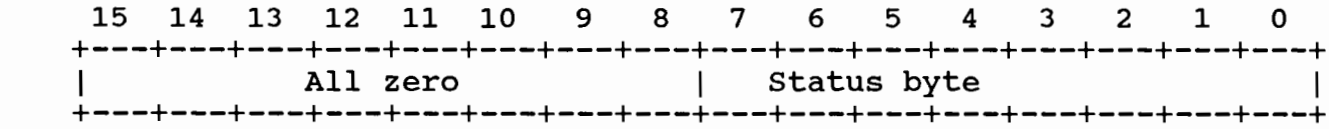
2.5.9 $TXDT Transaction

This command is used to transfer data from the HP to the Z80 subsystem.

Request Format:-



Reply Format:-

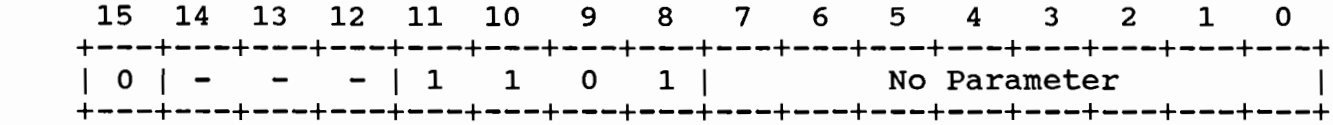


Status byte - =0, Transfer OK
- =1, No buffer space
=2, Buffer size illegal
=3, Terminal number illegal

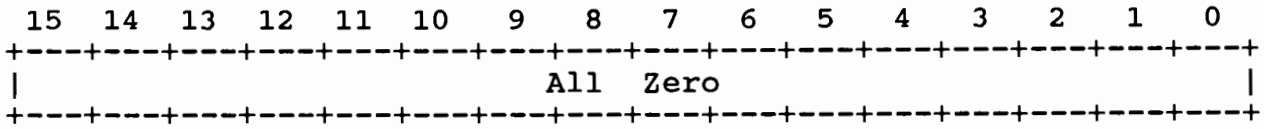
2.5.10 $EIRQ Transaction

The RTE.A interface driver uses this command to enable unsolicited interrupts (IRQs) from the Z80 subsystem after it has completed a sequence of backplane transactions with IRQs off. This is an immediate command executed in about 25 microseconds at the NMI interrupt level. It clears the Backplane Busy (BPB) flag which allows any pending IRQs to proceed.

Request Format:-



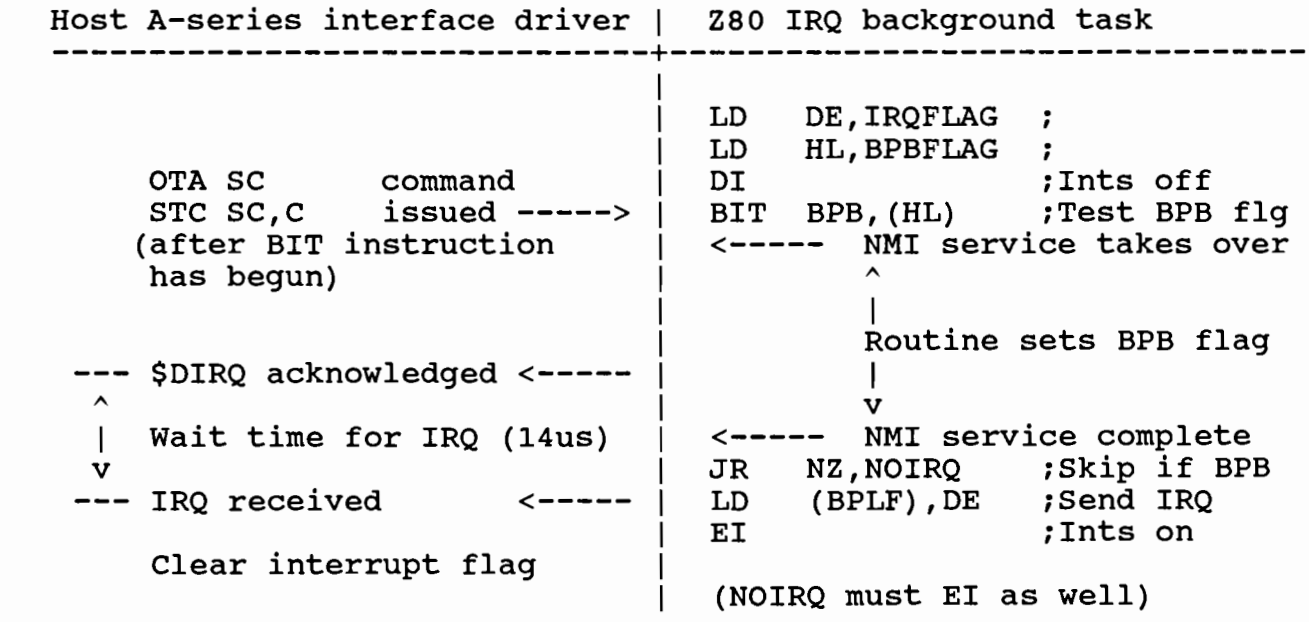
Reply Format:-



2.5.11 $DIRQ Transaction

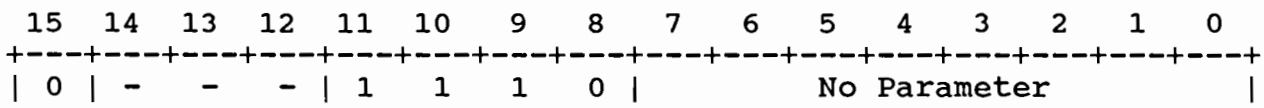
The RTE.A interface driver uses this command to disable unsolicited interrupts from the Z80 subsytem. This is an immediate command executing in about 25 microseconds at the NMI interrupt level. This command works in conjunction with the IRQ background task to ensure that a collision between a $DIRQ from the driver and an IRQ from the card does not upset the sequencing of the backplane transaction flow.

The $DIRQ sets the Backplane Busy (BPB) flag from the NMI level, while the IRQ process can only check this flag from the background task level with the maskable interrupts off. This leaves a small window of about 3 microseconds where it is possible for the IRQ process to commit to sending the IRQ after the NMI has been issued by the host but not initiated by the the Z80. This results in a time of approximately 14 microseconds where there is a small probability of receiving an unexpected flag or interrupt. To ensure this does cause a problem the interface driver must wait this time and then issue a flag clear before proceeding with other backplane commands.

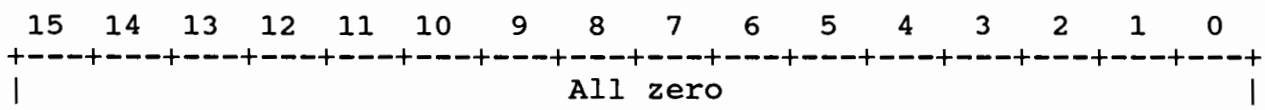


Revised <851019.2258>

Request Format:-



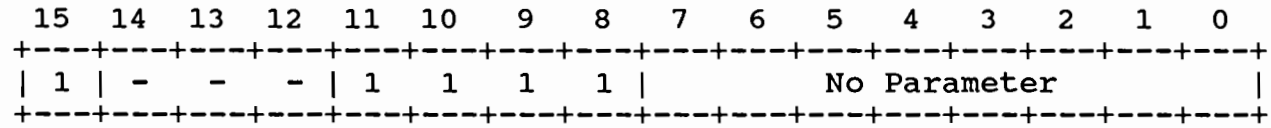
Reply Format:-



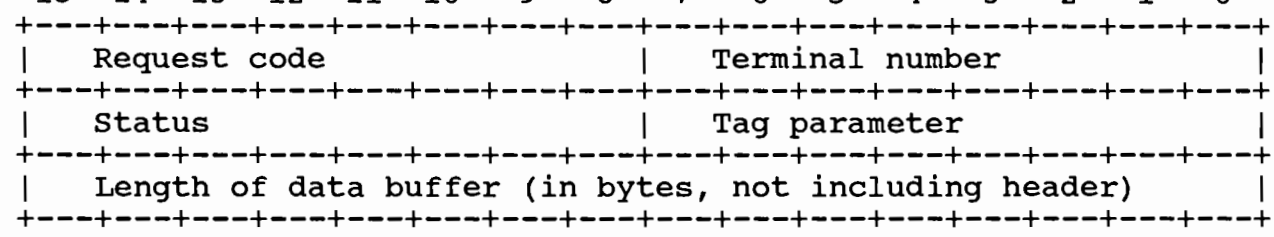
2.5.12 $FIRQ Transaction

This command is used to identify the next IRQ at the top of the IRQ queue within the Z80 subsystem. This is a deffered NMI command executed by the NMIIRQ routine within the backgound NMI tasks.

Request Format:-



Reply Format:-



Request codes:-

0 - nothing to send
1 - read completion
2 - write completion
3 -
4 - status update request

Revised <851019.2258>

5 - terminal error report

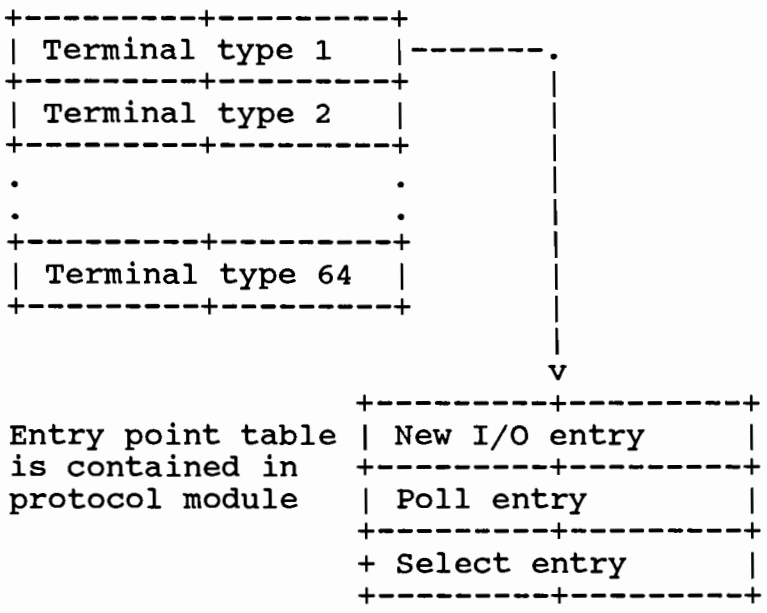
If the IRQ queue is empty when a $FIRQ is executed the 6 bytes are still returned however the request code is set to zero (so are the other bytes).

2.6 Z80 System Tables

2.6.1 Protocol Vector Entry Table

The protocol entry table is used by the scheduler to call the protocol modules. There is one entry per terminal type. This entry contains the address of the 6 byte entry table within each protocol module. There are a maximum of 64 entry point vectors, allowing for 64 different terminal types to be defined. This value may be expanded to 255 if required in the future.

The vector table is initialized to point to an error module, however when the protocol modules are downloaded this value is altered to point to their local entry point table.



The protocol module is entered with the registers set as follows:-

IX	-	Terminal table address
IY	-	Port table address
B	-	Terminal number
C	-	SCC data port (0-7)
HL	-	Entry point address
DE	-	Entry point offset 0,2,4
A	-	Request code only if new I/O (see $TERM transaction)

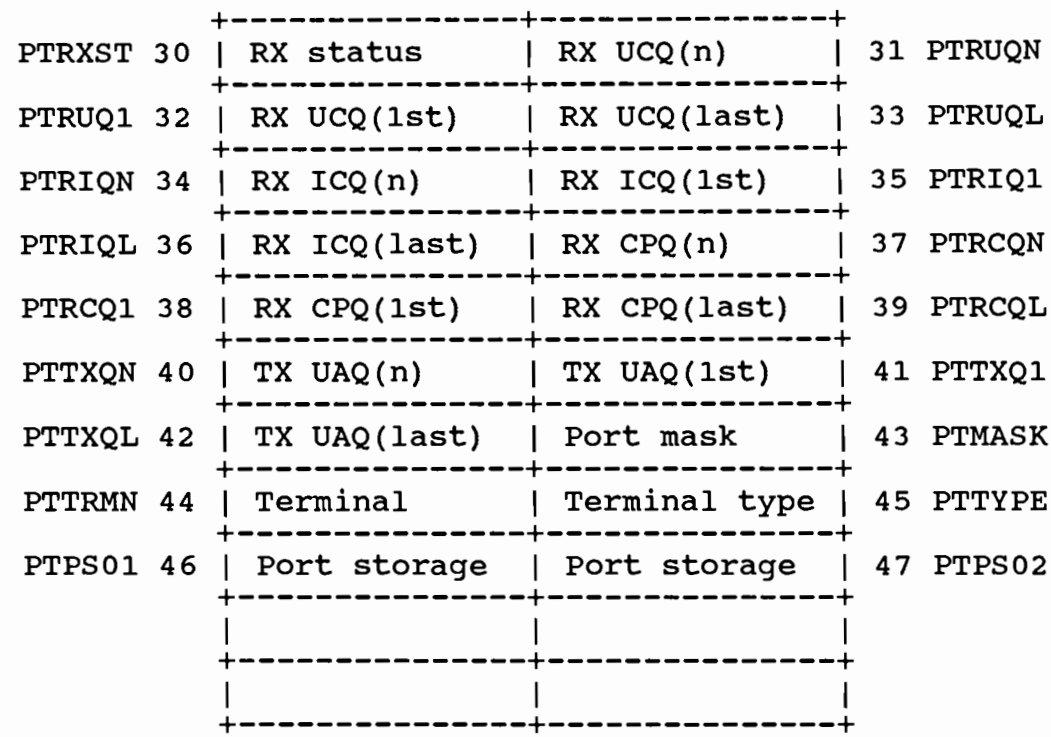
2.6.2 Port Tables

The port table contains all the information releated to an SCC port. It is designed for full duplex operation and has variables dedicated to transmit and receive functions.

Port table layout with symbol offsets:-

PTTM1R	0	Timer 1 Resl.	Timer 1 Clock	1	PTTM1C
PTTM1S	2	Timer 1 timeout schedule addrs		3	
PTTM2R	4	Timer 2 Resl.	Timer 2 Clock	5	PTTM2C
PTTM2S	6	Timer 2 timeout schedule addrs		7	
PTISV1	8	Interrupt schedule vector 1		9	
PTISV2	10	Interrupt schedule vector 2		11	
PTISV3	12	Interrupt schedule vector 3		13	
PTISV4	14	Interrupt schedule vector 4		15	
PTRXCV	16	RX Completion schedule vector		17	
PTTXCV	18	TX Completion schedule vector		19	
PTTXPT	20	TX port (C)	TX length (B)	21	PTTXBL
PTTXBF	22	TX buffer pointer (HL)		23	
PTTXST	24	TX status		25	
PTRXPT	26	RX port (C)	RX length (B)	27	PTRXBL
PTRXBF	28	RX buffer pointer (HL)		29	

Revised <851019.2258>



PTTM1R/C/S and PTTM2R/C/S - Port timeout clocks

There are two timeout clocks per port, usually one for Receive and the second for Transmit. The first byte in each is the resolution code. The resolution is the order of magnitude of the clock ticks. 1 is 10ms, 2 is 100ms, 3 is 1 second, 4 is 10 seconds. The clock ticks are counted by the second byte. When this reaches zero a timeout occurs and the routine at the timeout schedul address is entered with the maskable interrupts disabled. When the timeout routine has completed, CONTROL MUST BE RETURNED TO THE TIMER BACKGROUND TASK (using a RET 0) which will reenable the interrupts and continue processing the other timeout clocks. If a clock value is already zero then the clock is ignored during the timer cycles.

PTISV1/2/3/4 - Interrupt schedule vectors

These vectors are not used directly by the hardware but are used to point to port specific code to port common code. Normally the transmit and receive interrupt routines for a particular protocol will be coded separately for each port. However at end of buffer or other more complex tasks which are not executed frequently it is more practical to use port common code. The port table interrupt vectors are used to transfer control to these routines.

PTRXCV and PTTXCV - Completion schedule vectors

When a receive or transmit interrupt task wants to transfer control

to the background task it sets the appropriate port bit in the completion scheduler byte. The RX and TX completion background tasks will then enter the protocol module at the address specified by the RX or TX completion vector.

PTTXBF, PTTXPT, PTTXBL, PTTXST - Transmit buffer interrupt variables

These 4 bytes are for the transmitter interrupt handler. PTTXBF should be loaded into HL and PTTXPT into BC. The B register then contains the byte counter and C the data port address. Because the buffers are 256 bytes long and always start on a byte boundary the H of HL will not change. So only B an L need be stored back after the character has been sent. PTTXST is for transmission status such as timeout and underrun etc.

```
        .....                          ;Point of interrupt (POI)
        Interrupt acknowledge          ;
TXINT   EX   AF,AF'                    ;Save registers
        EXX                            ;
        LD   HL,(PTTXBF+PTTABn)        ;Buffer pointer
        LD   BC,(PTTXPT+PPTABn)        ;Port and length
        OUTI                           ;
        LD   H,L                       ;Move L -> H
        LD   L,B                       ;Move B -> L
        LD   (PTTXBL+PTTABn),HL        ;Save B and L
        JR   Z,TXENDB                  ;End of buffer
        LD   A,38H                     ;Reset IUS condition
        OUT  (SCCC n),A                ;on SCC channel
        EXX                            ;Restore registers
        EX   AF,AF'                    ;
        EI                             ;Interrupts back on
        RETI                           ;Return to POI
```

This routine illustrates a typical interrupt handler for buffer transmission coded for a specific port n. The execution time including the interrupt acknowledge is approximately 36 us at 4MHz.

PTRXBF, PTRXPT, PTRXBL, PTRXST - Receive buffer interrupt variables

These variables are used in a similar way to the TX variables.

PTRUQN/1/L - Uncompleted receive buffer queue

This queue is filled by the CPLPCS task before entering the protocol module at the poll entry point. It contains empty buffers to be used by the receive interrupt task. If there are not enough buffers to be able to satisfy the longest read the poll entry is skipped.

PTRIQN/1/L - Intermediate receive buffer queue

This queue is used by the protocol modules to store uncompleted receive buffers. The buffers are moved to the completed receive queue as soon as the buffer is verified as good. (ie no SDLC abort or CRC error etc.)

PTRCQN/1/L - Completed receive buffer queue

This queue is used to pass the completed receive buffers to the background protocol task. The background task will pass information buffers (rather that control buffers) on to the IRQ queue for sending to the host computer.

PTTXQN/1/L - Unacknowlegded transmit buffer queue

This queue is used by protocols such as SDLC for storing unacknowledged transmit buffers after they have been sent. If the buffers are not acknowledged before the timeout they are moved back to the front of the transmit queue in the terminal table for resending.

PTTRMN, PTTYPE - Terminal number and terminal type

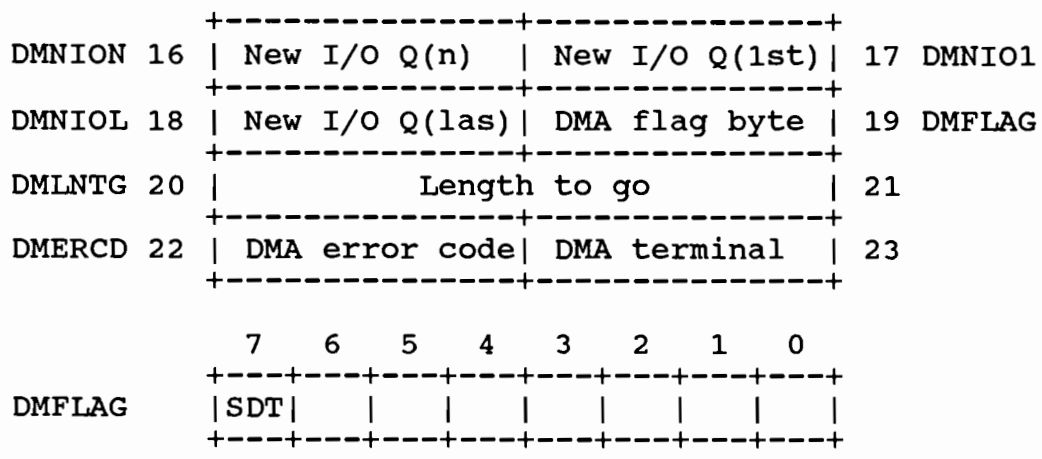
These parameters are copied from the terminal table of the terminal currently active on the port. This is done by CPLPCS before the protocol module is entered.

2.6.3 DMA Table

The DMA tables contain information about the current backplane DMA transfer and queuing information about the IRQ (Interrupt Request Queue) packets waiting to be transferred to the host processor.

DMCSCA	0	Completion schedule address		1	
DMNXBF	2	Next buffer	Buffers to go	3	DMBFTG
DMCBQN	4	Current Q(n)	Current Q(1st)	5	DMCBQ1
DMCBQL	6	Current Q(las)	IRQ Q(n)	7	DMIRQN
DMIRQ1	8	IRQ Q(1st)	IRQ Q(last)	9	DMIRQL
DMRXQN	10	Free RX Q(n)	Free RX Q(1st)	11	DMRXQ1
DMRXQL	12	Free RX Q(las)	Free TX Q(n)	13	DMTXQN
DMTXQ1	14	Free TX Q(1st)	Free TX Q(las)	15	DMTXQL

Revised <851019.2258>



DMSTDT (7) - Transfer terminal status table

DMCSCA - DMA completion schedule address

The DMA completion address is used to return control back to the NMI task that setup the backplane transfer. The NMI task wraps up the transfer and then acknowledges the NMI as completed to the A-series host.

DMNXBF and DMBFTG - Multi buffer pointer and counter

During a multi buffer DMA transfer the next buffer address and the number of buffers are used to control the scheduling of the DMA packets. At the start of each packet the buffer high order byte is stored in DMNXBF. The low order byte and the length are fetched form the buffer header and the DMA started. When it completes the value of DMBFTG is decremented and tested for zero. When zero the DMA routine sets the DMA completion flag causing the background task to be restarted at DMSCSA.

DMCBQN/1/L - DMA current buffer queue

This queue contains the buffers required for the current backplane transfer. It is setup by the $RXDT or $TXDT commands. During the $RXDT command the message buffers are copied from the IRQ queue and then transferred, while the $TXDT calculates the number of buffers from the length and fetches them from the free TX queue (DMTXQN/1/L). It also initializes the buffer header parameters as the buffers are added to the queue.

DMIRQN/1/L - Interrupt Request Queue

The Interrupt Request Queue (IRQ) contains all the buffers (Read completion, Transmit completion, and Status change) that need to be passed to the host. This queue is filled by the receive and transmit

protocol tasks and is emptied by the $FIRQ (fetch IRQ) and $RXDT commands from the main processor.

DMRXQN/1/L and DMTXQN/1/L - Free buffer queues

These queues contain the free buffers for the transmit and receive tasks. The receive queue is filled by the $RXDT process when receive data has been transferred to the main processor, or by $FIRQ after a status change buffer has been read by $FIRQ. The transmit queue is filled by $FIRQ after the transmission completion status has been transferred.

DMNION/1/L - New I/O queue

New I/O requests are generated by $TERM and $TXDT commands. The queue consists of a chain of terminal tables linked by the TMNWIO bytes. Note that this queue is different from the normal buffer queue with the linkage bytes containing the terminal table number. The new I/O requests are set in the TMNWIO byte. (See terminal table)

2.6.4 Terminal Tables

The terminal tables are used to keep the terminal specific information such as the poll and select codes etc.

Terminal table layout with symbol offsets:-

TMSTAT	0	Status	Status 2	1	TMSTT2
TMERRS	2	Error retry counter		3	
TMDWNC	4	Down counter	Port number	5	TMPORT
TMGLNK	6	Group linkage	Terminal type	7	TMTYPE
TMGPLL	8	Group Poll	Device Poll	9	TMDPLL
TMGSLC	10	Group Select	Device Select	11	TMDSLC
TMNSNB	12	N(s) counter	N(r) counter	13	TMNRNB
TMFLGS	14	Flags	TX Q(n)	15	TMTXQN
TMTXQ1	16	TX Q(1st)	TX Q(last)	17	TMTXQL
TMNWIO	18	New I/O link	New I/O flags	19	TMNFLG

Revised <851019.2258>

```
          +--------------+--------------+
```

TMSTAT - Terminal status byte

This byte indicates the status of the terminal. It is used by the CPLPCS processes (completed poll) to determine what to do during the next poll cycle. These bits are covered in detail in the CPLPCS description.

Status byte bits:-

```
             7   6   5   4   3   2   1   0
           +---+---+---+---+---+---+---+---+
TMSTAT     |DSB|SPL|INA|   |DWN|DNC|GPA|GPM|
           +---+---+---+---+---+---+---+---+
```

TMBDSB	(7)	-	Terminal disable
TMBSPL	(6)	-	Terminal slow poll
TMBINA	(5)	-	Terminal inactive (no poll)
TMBDWN	(3)	-	Terminal down
TMBDNC	(2)	-	Terminal down counter expired
TMBGPA	(1)	-	Group poll active
TMBGPM	(0)	-	Group poll master

TMSTT2 - Secondary Terminal Status

The secondary status is for the use of the protocol module driving the terminal. It is not used by any of the system functions.

TMERRS - Error retry counter

This error retry counter is only used for statistical purposes. It is incremented by the routine TMERINC when ever a protocol module detects an error event during a data transfer operation. It is transferred to the host processor during the $STDT (Status table transfer) backplane command along with TMSTAT and TMSTT2.

TMDWNC - Terminal Downcounter

The down counter is decremented by the system every 10 seconds while ever it is non zero. It is normally used by the select routines in the protocol modules to timeout a select that cannot complete due to a malfunction in the terminal or terminal software.

TMGLNK - Group Poll Linkage

The group poll linkage is only used when the protocol module allows group polling of the terminals. Terminals with a common port number and group poll address (GPA) are defined as being part of the same group. Scanning down the terminal table (0-127) the first terminal with a given port and GPA is the Group Master. This terminal will

have the Group Poll Master flag set in TMSTAT bit TMBGPM. If this bit isn't set then the terminal belongs to the group whose terminal number is in TMGPLK. When the $TERM command initializes a terminal table entry it will set the TMBGPM bit. It is the responsibility of the protocol module to set TMBGPA (group poll active) if it is going to do group polling. TMGPLK in the group master terminal table points to the currently active subterminal during a select cycle.

TMTYPE - Terminal Type

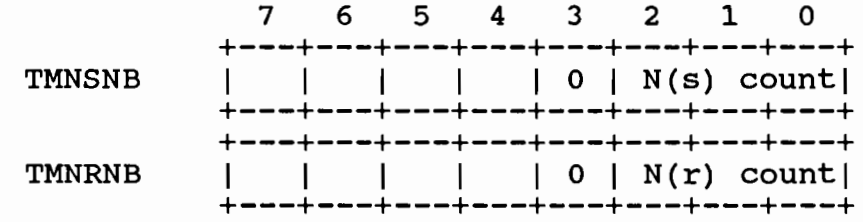
The terminal type determines which protocol module to use when scheduling a Poll, Select, or New I/O task. Sceduling is done via the Entry Point Vector Table.

TMGPLL, TMDPLL, TMGSLG, TMDSLC - Terminal group and device addresses

These bytes contain the terminal addresses. Not all protocols use all these bytes. TMGPLL is the only byte which is significant outside the specific protocol module and is used as part of the Group poll mechanism. The other bytes can be used by the protocol module for general addressing bytes. However it should be noted that the $TERM command will initialize all the address bytes.

TMNSNB, TMNRNB - SDLC sequence counters.

These bytes are used specifically by bit level protocols for the send and receive sequence counters. They are general purpose bytes for protocols that do not need N(s) and N(r) counters.



TMFLGS - Terminal table flags

General purpose flags for protocol module to define.

TMTXQN/1/L - Transmit queue

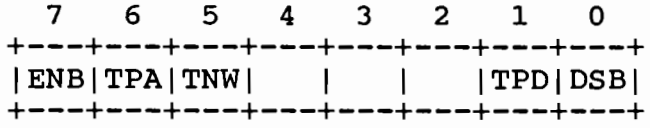
The transmit queue if filled by the $TXDT backplane transaction with new transmit buffers. When the buffers have been successfully sent or have expired their retry limits they are returned to the IRQ queue for notification to the host processor. Once the notified the host uses the $FIRQ command to read the status of the buffers and then the $FIRQ returns them to the free transmit queue (DMTXQN) in the DMA port table.

Revised <851019.2258>

TMNWIO, TMNFLG - New I/O linkage and flags

The new I/O linkage is used to queue the new I/O requests to the New I/O queue in the DMA port table. Each terminal entry can only be queued once in this queue. If more than one New I/O event is outstanding then they are identified by the bit settings in the New I/O flag byte. When the terminal table comes to the top of the DMA New I/O queue the flag bits are examined. If no bits are set then the terminal table is removed from the queue. Otherwise the protocol module is entered at the New I/O entry point.

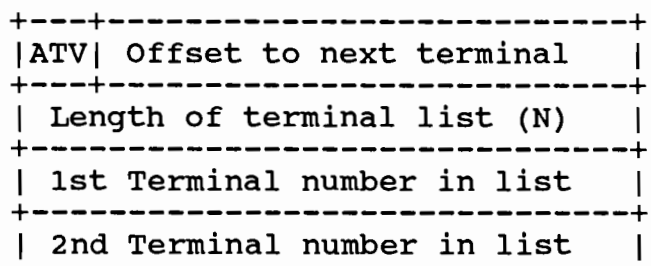
The sequence of execution of the event flags is up to the protocol module but it is recommended that they are executed from left to right. As each event is processed the corresponding flag bit MUST be cleared. If it is not cleared then the protocol will continue to be entered until it is cleared. The protocol module need not execute all the events at once, however it will remain on top of the New I/O queue until all the bits are cleared. Once all the bits are cleared the protocol module can remove the terminal table from the New I/O queue, if it doesn't NIOPCS will remove it in the next schedule loop.

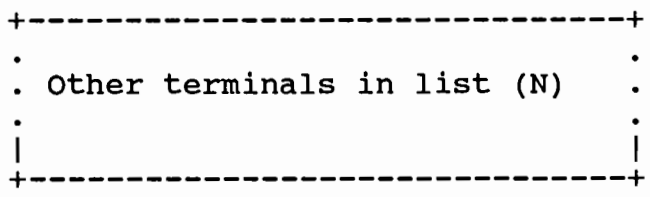


TMFENB	(7)	-	Terminal enable
TMFTPA	(6)	-	Terminal poll activate
TMFTNW	(5)	-	Terminal new write
TMFTPD	(1)	-	Terminal poll deactivate
TMFDSB	(0)	-	Terminal disable

2.6.5 Roll Call Table

The Roll call table is an ordered polling list. The terminals are called in the order on the list to be polled. If the terminal has a pending transmission on the transmit queue this takes precedence and will be scheduled instead of the poll. If the Active bit is not set then roll call table is removed from CPLPCS and polling on that port ceases.





ATV - Roll call tabl active.

2.7 Buffer Management and Queueing

This section deals with the data buffers. These buffers are an important resource to the datacom susbsytem and must be carefully organised to ensure that the system behaves efficiently and smoothly.

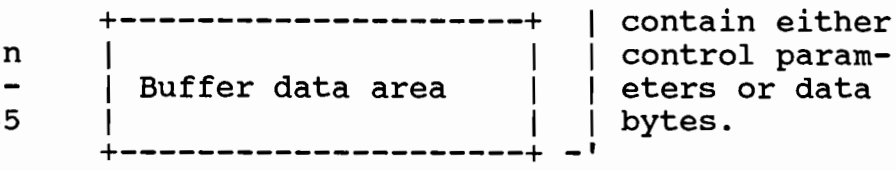
2.7.1 Buffer Structure

The buffers in the system are 256 bytes long of which 252 bytes are data area and the other 4 are control information. The buffers are allocated on a port basis with a fixed number given to the transmit and receive channels within a port. The buffer area begins on a 256 byte boundary so that the low order byte of the buffer address is always zero. This fact is assumed by many of the buffer handling routines and considerably simplifies the programming involved.

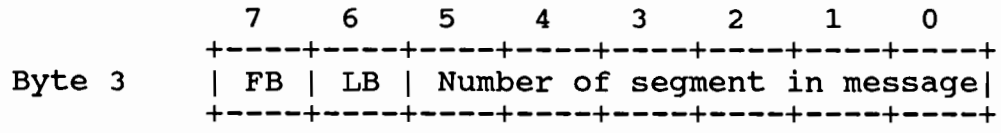
The 93695V card has 32K bytes available for buffers giving 128 buffers in total, 16 to each port and 8 to each channel within a port. The 12040B and 12042B only have 8K bytes for buffer area giving 64 buffers in total. This fact should be considered when designing software that must be compatable across the 3 cards.

Data Buffer Format:-

Byte 0	Buffer linkage (H)	H of HL to Next buffer
1	Data length in buf.	0-252 Bytes long
2	Buffer start offset	Offset.GE.4
3	Segment parameter	See 1. below
4	Parameters 0-252	This area can



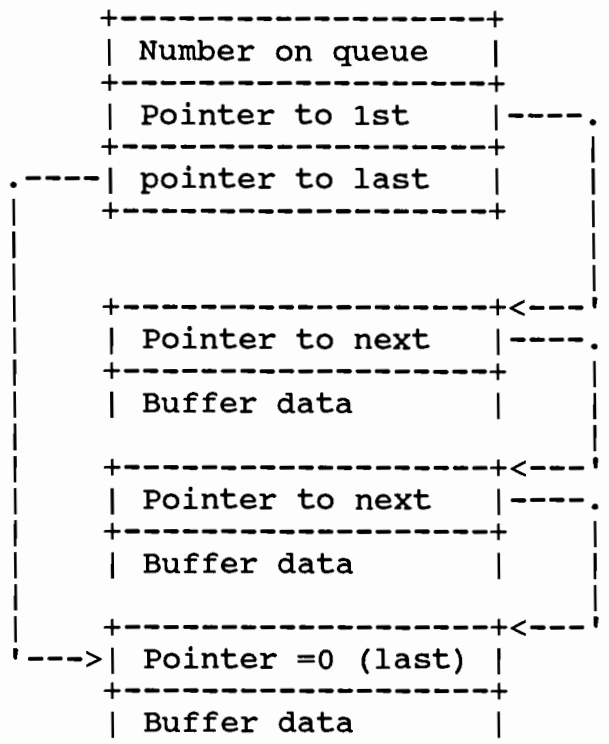
1. Segment parameter



FB - First buffer flag =1 if 1st in message
LB - Last buffer flag =1 if last in message

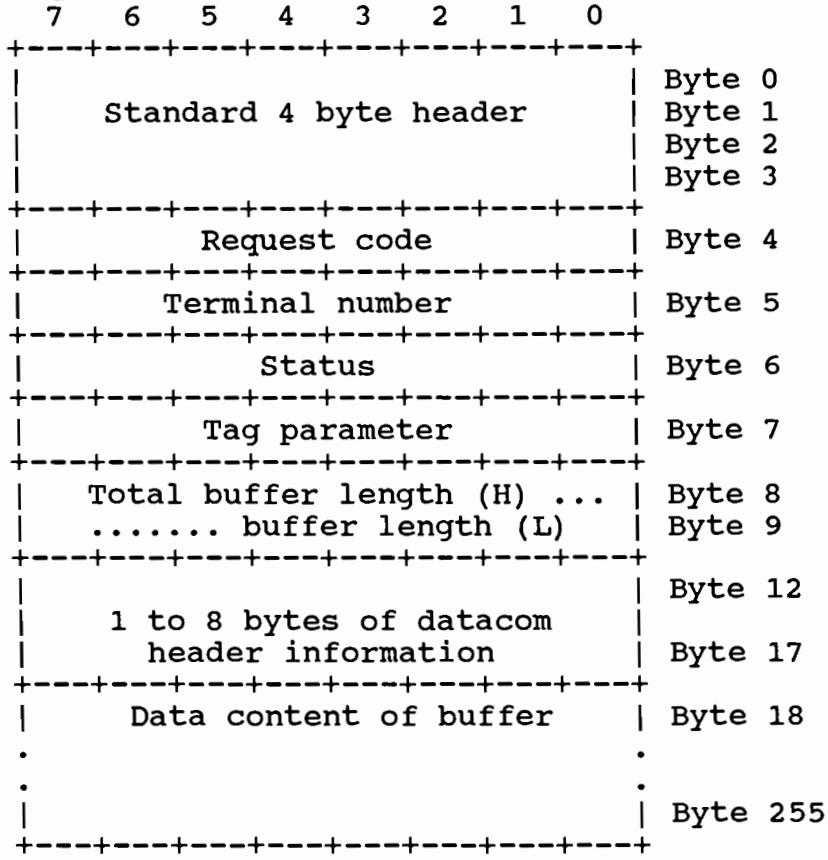
2.7.2 Buffer Queueing

The buffer are queued using a linked list. Each list has a queue header which contains the number on the queue, the pointer to the 1st, and the pointer to the last.



2.7.3 Receive and Transmit buffers

In addition to the 4 byte header that every buffer has there is an additional 14 bytes used in the first buffer of each transmit and receive buffer chain. These contain the terminal and status information required by some of the protocol and backplane modules.



The datacom header is up to 8 bytes long. The $TXDT backplane routine reserves this space for the protocol modules to use. (In fact is is setup by the interface driver in the host.) When receiving, the protocol modules should align the start on the datacom header so that the data portion of the buffer always starts on byte 18. If there are more than 8 bytes of datacom header then the protocol module must be coded specially to work around this restriction.

2.8 System Subroutine Modules

This section describes the subroutines in the system area. These modules are for general use by any of the processes within the datacom subsystem. The descriptions specify the purpose of the subroutine, the calling parameters the return parameters, and any registers that are altered.

2.8.1 QPUT - Add buffer to queue

QPUT is used to add a buffer to the end of the current buffer chain.

Execution Time = 17.5 us at 4MHz (queue previously empty)
20.0 us at 4MHz (queue not empty)

Calling Sequence

```
LD   HL,Address of buffer queue header block
LD   D,High byte of buffer address to be linked
CALL QPUT
```

Return Parameters

IX	-	Unchanged
IY	-	Unchanged
HL	-	Not defined
DE	-	Not defined
BC	-	Unchanged
A	-	Not defined
Flags	-	Not defined

2.8.2 QGET - Get buffer from queue

QGET is used to get the next buffer from the start of the buffer chain defined by the pointer HL. The P/V flag is set (PE) if there are no items on the queue, otherwise if the get is successful P/V is cleared (PO).

Execution Time = 17 us at 4MHz (If successful)
6 us at 4MHz (If queue empty)

Calling Sequence

```
LD   HL,Address of buffer chain header block
CALL QGET
JP   PE,Empty Queue
DE=Buffer address
```

Return Parameters

```
IX      -  Unchanged
IY      -  Unchanged
HL      -  HL+1
DE      -  Address of buffer fetched
BC      -  Unchanged
A       -  Not defined
Flags   -  P/V=1 (PE) if queue empty, else P/V=0 (PO)
```

2.8.3 QGETPUT - Get and put buffer

QGETPUT fetches the buffer from the top of the queue specified by HL and puts it on to the end of the queue specified by DE. If there are no buffers on the HL queue then P/V is set (PE) and the routine returns without attempting to put anything on the DE queue.

Execution Time = 6.0 us at 4MHz if get queue empty
36.75us at 4MHz if put queue previously empty
39.25us at 4MHz if put queue not empty

Calling Sequence

```
LD   HL,Address of get buffer queue header
LD   DE,Address of put buffer queue header
CALL QGETPUT
JP   PE,queue(HL) empty
operation OK
```

Return Parameters

```
IX         Unchanged
IY      -  Unchanged
HL      -  Not defined
DE      -  Not defined
BC      -  Unchanged
A       -  Not defined
Flags   -  PE if get fails, PO if OK
```

2.8.4 QJOIN - Join two queues together

QJOIN is used to join two queues together, with the source queue contents being appended to the destination queue. The source queue is set to empty.

Execution time:
```
 6.0 us if source queue empty
35.5 us if destination queue previously empty
39.25us if destination queue not previously empty
```

Calling sequence:
```
LD   HL,address of source queue header
LD   DL,address of destination queue header
CALL QJOIN
JP   PE,queueempty
```

Return parameters:
```
IX      - unchanged
IY      - unchanged
HL      - undefined
DE      - undefined
BC      - unchanged
A       - undefined
flags   - P/V=1 if source queue empty
          P/V=0 if source not queue empty
```

2.8.5 QMSGMOVE - Move 1st message to new queue

QMSGMOVE is used to move all buffers comprising the first message from the top of the source queue and append to the destination queue.

Execution time:
```
 6.0 us if source queue empty
(58.0+12n)us if destination queue previously empty
(61.5+12n)us if destination queue not previously empty
   (where n is number of buffers in first source queue messa??
```

Calling sequence:
```
LD   HL,address of source queue header
LD   DE,address of destination queue header
CALL QMSGMOVE
```

Return parameters:
```
Registers preserved: IX, IY, BC
Registers destroyed: HL, DE, A, flags
```

Revised <851019.2258>

2.8.6 QNIOPUT - Add terminal to new I/O queue

QNIOPUT isused to queue a terminal on the queue used for new I/O events. The queued items are the terminal tables themselves, not buffers. The linkage is via the TMNWIO field of the terminal table.

Execution time:
33.0 us if new I/O previously empty
66.0 to 70.0 us if new I/O queue not previously empty

Calling sequence:

```
LD   A,terminal number to added to queue
CALL QNIOPUT
```

Registers preserved: IX, IY, BC, A, flags
Registers destroyed: HL, DE

2.8.7 CYCLE - Select next scheduled port

The cycle subroutine is called upon entry to any background process where several ports may be scheduled, as indicated by corresponding bits in the schedule variable. The subroutine avoids any priority between ports, maps the bit in the schedule variable to the corresponding port number, and resets the port bit in the schedule variable.

On entry: HL = address of schedule variable (one bit per port)
This byte MUST NOT be zero for successful return
DE = address of process cycle byte (per process)

On exit: C = A = SCC data port number (0-7)
B register destroyed
other registers unchanged

Execution times:

If next port number returned:	27.5 us
If same port number returned:	76.5 us
Average:	52.0 us

2.8.8 PTBADR - Get Port table in IY

PTBADR is used to get the port table address in IY from the port number specified in the A register (port 0-7).

Execution Time = 19.25 microseconds

Calling Sequence

```
LD   A,port number
CALL PTBADR
IY=Port table address
```

Return Parameters

IX	-	Unchanged
IY	-	Port table address
HL	-	Not defined
DE	-	Not defined
BC	-	Unchanged
A	-	Not defined
Flags	-	Not defined

2.8.9 TTBADR - Get Terminal table in IX

TTBADR sets IX to point to the terminal table specified by the terminal number in A.

Execution Time = 27.0 microseconds if terminal number even
31.0 microseconds if terminal number odd

Calling Sequence

```
LD   A,terminal number
CALL TTBADR
IX=terminal table address
```

Return Parameters

IX	-	Terminal table address
IY	-	Unchanged
HL	-	Not defined
DE	-	Not defined
BC	-	Unchanged
A	-	Unchanged
Flags	-	Not defined

Revised <851019.2258>

2.8.10 RTBADR - Get Roll call table in HL

RTBADR is used to setup the roll call table address in HL from the port number specified in the A register

Execution Time = 15.5 microseconds

Calling Sequence

```
LD   A,High byte of buffer address
CALL RTBADR
HL=roll call table address
```

Return Parameters

IX	-	Unchanged
IY	-	Unchanged
HL	-	Roll call table address
DE	-	Not defined
BC	-	Unchanged
A	-	Unchanged
Flags	-	Not defined

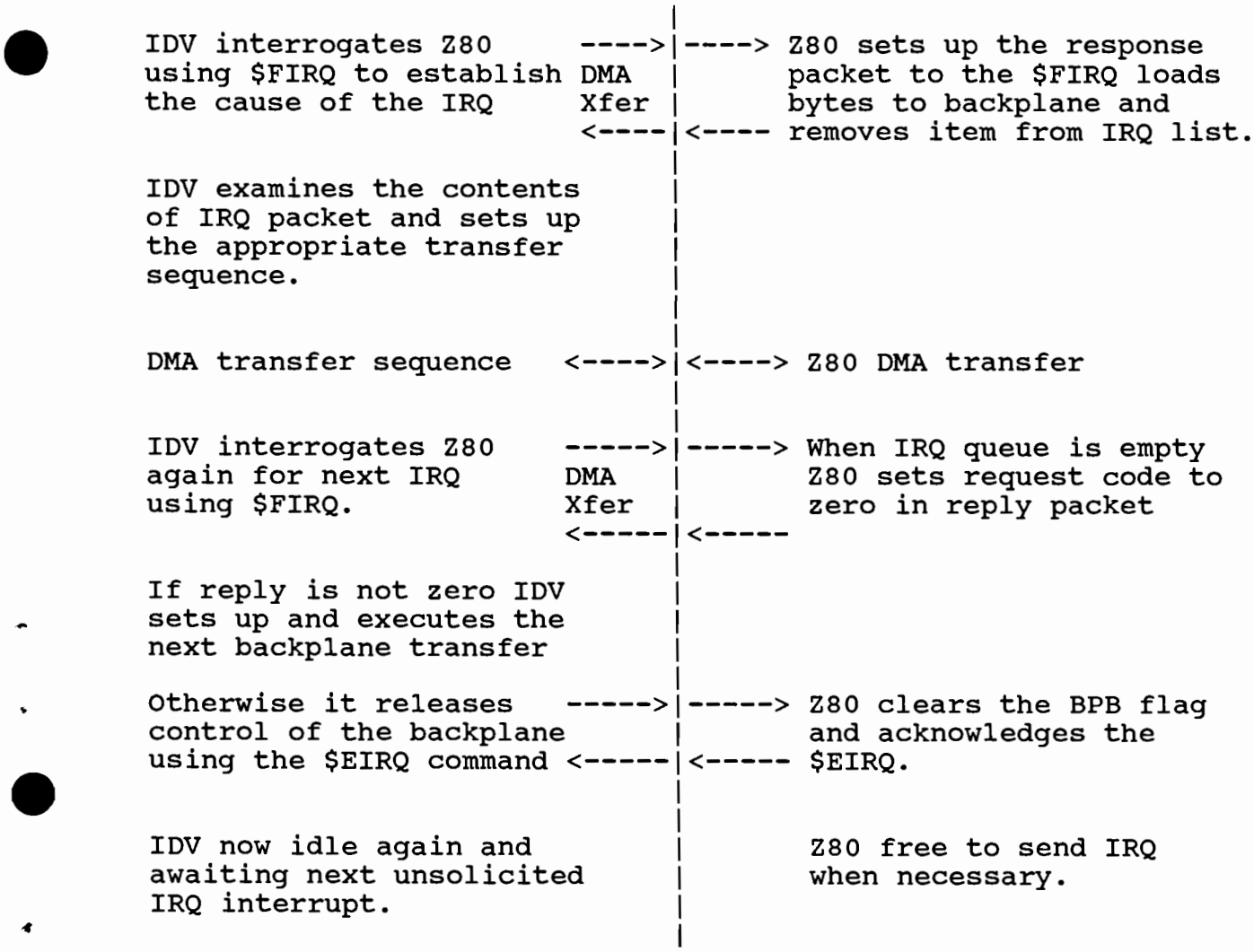
2.9 Backplane Transaction Flow

This section describes the detailed flow of events for backplane data transfer operations. These involve several of the backplane transaction commands for each type of transfer.

The backplane processing is controlled by a scheduler in the interface driver that uses an event list for dispatching the backplane tasks. The following diagrams illustrate some typical backplane activity.

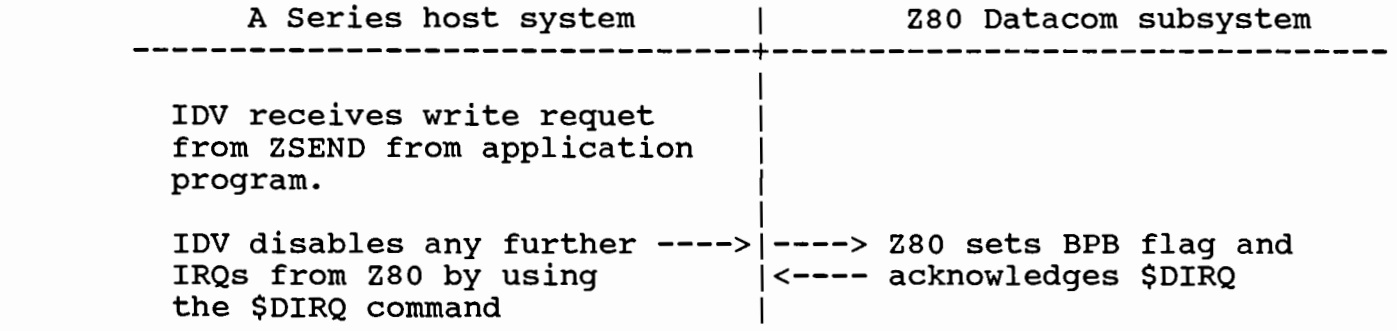
```
      A Series host system      |       Z80 Datacom subsystem
--------------------------------+--------------------------------
  Interface driver (IDV)        |
  idle at this time             |
                                |<---- Interrupt request (IRQ)
  IDV receives unsolicited <----|
  interrupt from Z80            |
                                |
  IDV disables any further ---->|----> Z80 sets BPB flag and
  IRQs from Z80 by using        |<---- acknowledges $DIRQ
  the $DIRQ command             |
```

Revised <851019.2258>

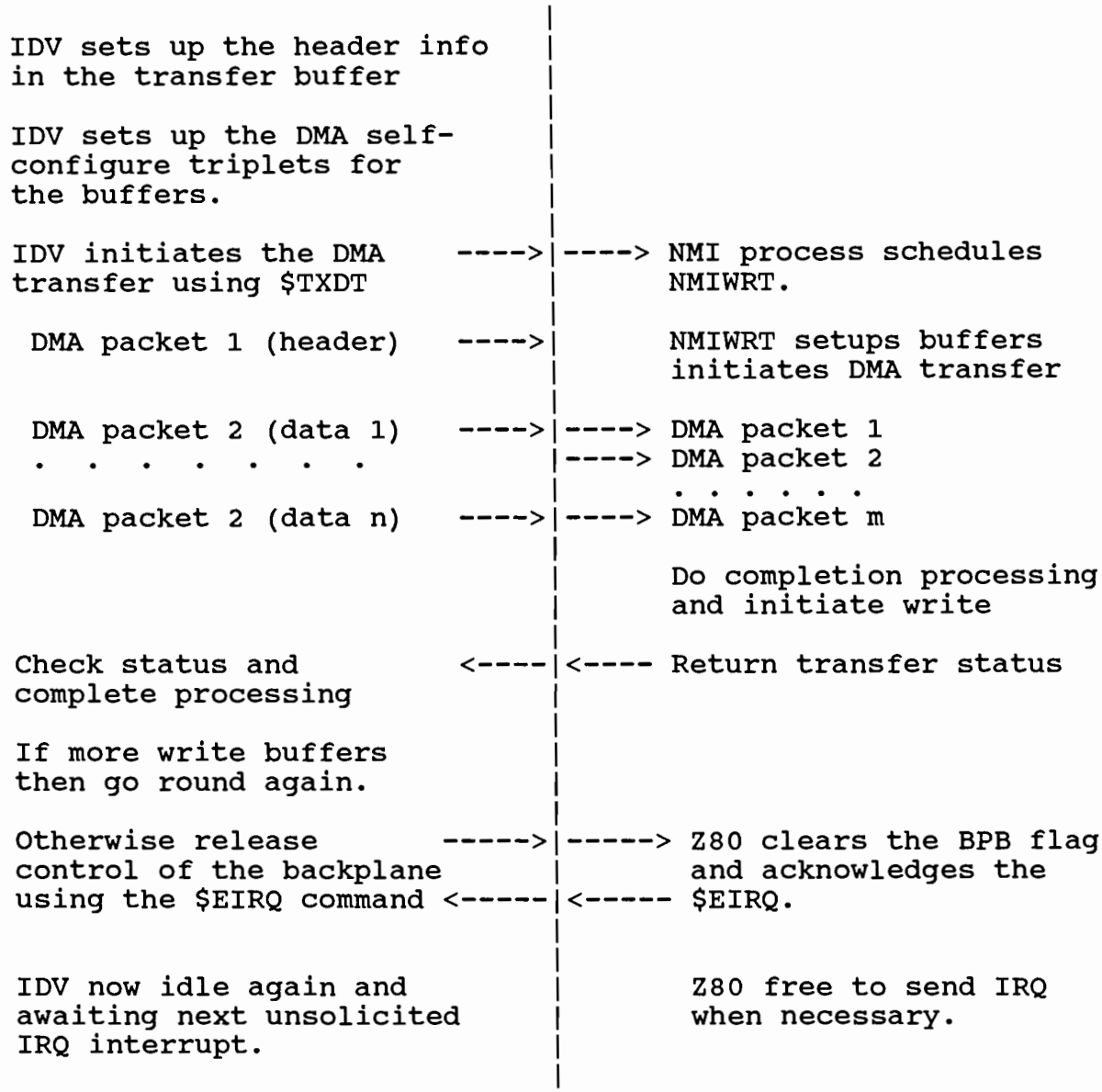


2.9.1 Write request to terminal

This diagram shows the events that take place during the $TXDT backplane transfer sequence.



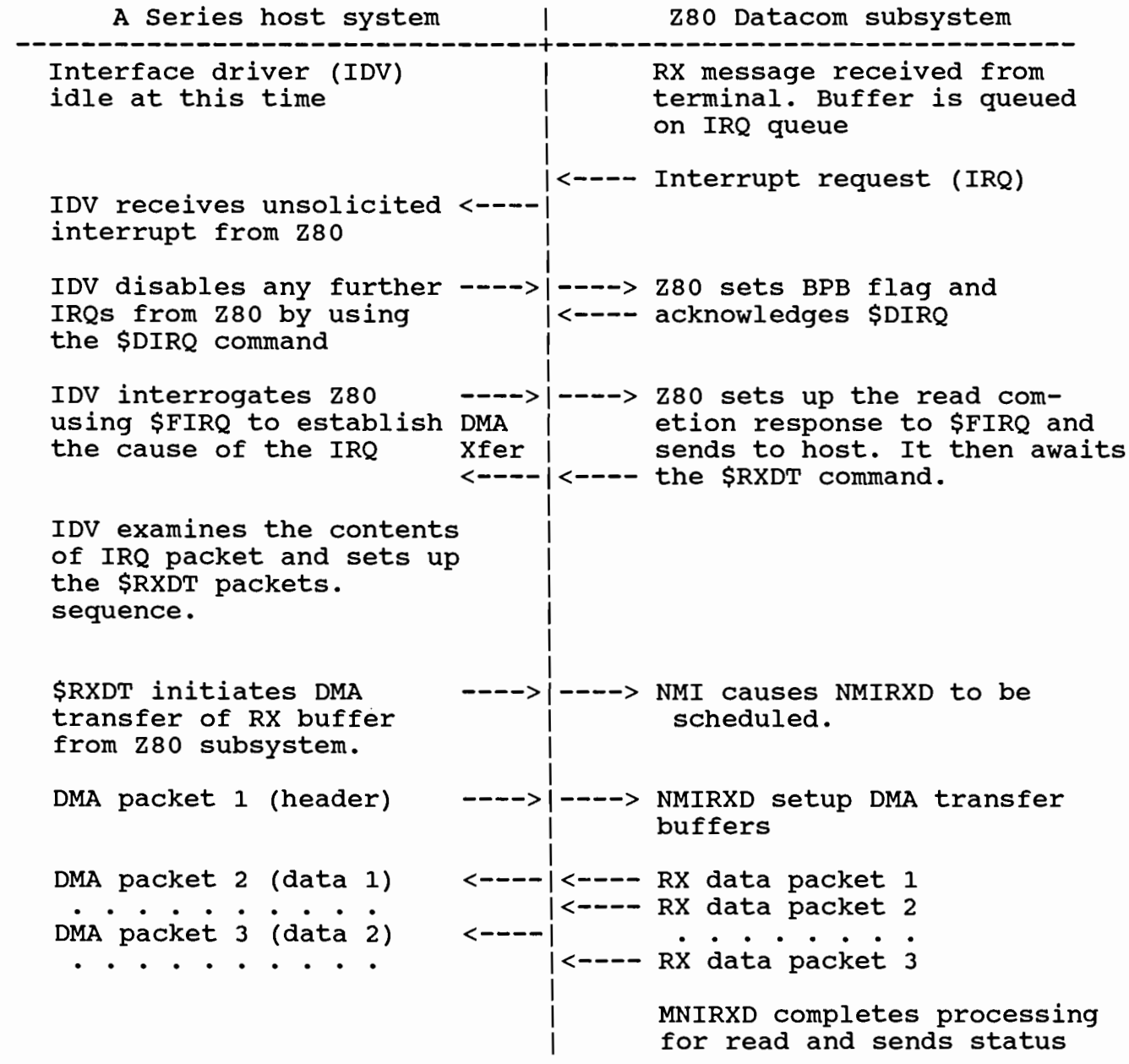
Revised <851019.2258>

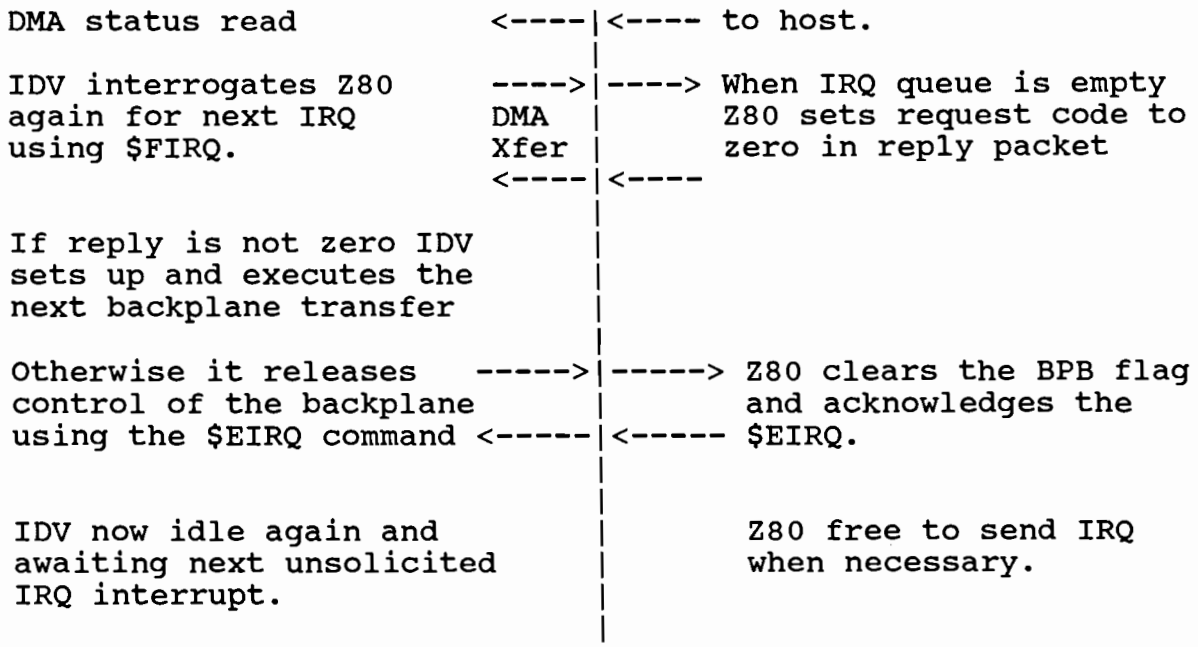


If there are not enough buffers in the Z80 subsystem to satisfy the transfer, a dummy DMA transfer is setup to write to the EPROM address space to force the A series host system to complete its DMA transfer. The status is set to indicate that the transfer has failed and the driver sets a flag to indicate that the mux is out of buffers. The transfer will be retried when the next write completion IRQ is fetched from the Z80 subsystem.

2.9.2 Read completion from terminal

For a terminal to receive a message it must first be enabled and then have polling activated, it is then ready to accept a message. When the messge arrives the buffer is queued on the IRQ for transfer to the host A series.





2.10 Background Tasks

2.10.1 TMRPCS - Timer Functions

The Timer Process is driven by CTC channel 3. The CTC interrupts every 10ms setting the flag in scheduler event table. On entry TMRPCS clears the event flag and scans the port table list looking for non zero timeout clocks with the resolution code set to 1. When it finds one it decrements the counter by 1 and checks for zero. If not zero then it continues to the next timeout clock. Otherwise it enters the timeout routine at the schedule address given in PTTMnS. The maskable interrupts are off and the registers set as follows:-

IX	-	Not defined
IY	-	Port table address
A	-	Not defined
B	-	Port number
C	-	SCC I/O port control address
D	-	Bit mask for OR with completion byte
E	-	Not defined
HL	-	Address of completion byte

If the routine is entered for port timer 1 then HL points to the RX completion byte. For port timer 2 it points to TX completion. None

of the registers need be saved by the timeout routine EXCEPT IY (this will not usually need to be altered).

The timeout routine normally will disable the receive or transmit interrupts, or both, set the timeout bit in the status, and set the I/O completion bit in the RX or TX completion scheduler event byte.

After completing the 10ms timeout servicing TMRPCS increments the 10ms counter TM10MS and tests for 10 or greater. If less than 10 it exits to the scheduler otherwise it repeats the timeout servicing for the 100ms timeouts. Every 1000ms it does the 1 second tiemouts and every 10,000ms the 10 second timeouts.

Also at the 10ms level the IRQ scheduler event flag is set if the IRQ queue length is non zero. This prevents $DIRQ from stalling the unsolicited interrupts to the host processor if there is a collision beteween $DIRQ and an IRQ interrupt. (See the $DIRQ backplane transaction).

At the 1 second timing level 1/10 of the terminals (13) have the slow poll bit TMBSPL cleared in the terminal table. This has the effect of giving any down terminals a slow poll cycle every 10 seconds.

Also at this 1 second timing level, in the same sequence as the slow poll processing, the terminal down counter is checked and decremented if it is non-zero. If it reaches zero after being decremented, the TMBDNC bit is set in the TMSTAT byte of the terminal table, thus indicating to the protocol module that the down counter has expired.

The terminal sequence is:-

```
cycle 1.  13, 26, 39, 52, 65, 78, 91,104,117,  2, 15, 28, 41,
      2.  54, 67, 80, 93,106,119,  4, 17, 30, 43, 56, 69, 82,
      3.  95,108,121,  6, 19, 32, 45, 58, 71, 84, 97,110,123,
      4.   8, 21, 34, 47, 60, 73, 86, 99,112,125, 10, 23, 36,
      5.  49, 62, 75, 88,101,114,127, 12, 25, 38, 51, 64, 77,
      6.  90,103,116,  1, 14, 27, 40, 53, 66, 79, 92,105,118,
      7.   3, 16, 29, 42, 55, 68, 81, 94,107,120,  5, 18, 31,
      8.  44, 57, 70, 83, 96,109,122,  7, 20, 33, 46, 59, 72,
      9.  85, 98,111,124,  9, 22, 35, 48, 61, 74, 87,100,113,
     10. 126, 11, 24, 37, 50, 63, 76, 89,102,115,  0,
```

The final cycle (10) only has 11 terminals in the in the sequence.

The terminal status request bit DMSTDT is set in the DMA port table every 10 seconds. This is the lowest priority $FIRQ event and is only fetched when the IRQ queue is empty. This will cause the host processor to execute a $STDT sequence to transfer the status infomation and to clear the DMSTDT bit.

Revised <851019.2258>

2.10.2 RXCPCS - RX I/O completion

This process will be scheduled whenever the RX completion byte in the scheduler event table is not zero. RXCPCS uses the CYCLE routine to determine the starting point for scanning the scheduler event byte. When a port is found RXCPCS enters the protocol module at the address pointed to by PTRXCV. The registers are set as follows:-

IX	-	Terminal table address
IY	-	Port table address
A	-	Not defined
B	-	Terminal number
C	-	SCC data port number
DE	-	Not defined
HL	-	Not defined

2.10.3 TXCPCS - TX I/O completion

The TX completion processing is identical to the receive processing except the entry point to the protocol module is PTTXCV.

2.10.4 NIOPCS - New I/O process

This process is intitiated by the $TERM and $TXDT backplane transactions. The event flag in the scheduler table is set whenever there are items on the NIO queue in the DMA port table. NIOPCS will enter the protocol module for the terminal at the top of the NIO queue with the registers set as follows:-

IX	-	Terminal table address
IY	-	Port table address
A	-	Not defined
B	-	Terminal number
C	-	SCC data port number
DE	-	Not defined
HL	-	Not defined

If TMNFLG is zero for the terminal on the top of the queue (ie no New I/O requests) the terminal is removed from the queue. If there are no terminals on the NIO queue the scheduler event byte is cleared and control is passed back to the scheduler. (See TMNLNK and TMNFLG

descriptions in the terminal table).

Note that NIOPCS will be entered whenever a $TERM or $TXDT transaction is executed. This may happen while a terminal is active (ie it is being polled or selected). It is the responsibility of the protocol module to determine what should be done (if anything). Normally for a polling protocol this entry can be ignored if the Active bit is set in the Roll Call table for the port that the terminal is attached to.

2.10.5 IRQPCS - Interrupt Request process

The interrupt request process interrupts the host processor whenever there are events on the IRQ queue. When the process is scheduled it tests the Backplane Busy flag. If this is set it returns to to the scheduler. Otherwise it clears the IRQ byte in the scheduler event table and sends an IRQ interrupt to the host processor. The IRQ interrupt is a 16 bit word set to hex 7FFF. There is a recovery mechanism for IRQs lost due to collision with $DIRQs from the main processor. (See TMRPCS and $DIRQ backplane transaction).

2.10.6 CPLPCS - Poll completion

CPLPCS is the polling mechanism. It is activated by the first terminal enable for each port. After enabled it uses the Roll Call Table to determine the sequence of polling. It is scheduled by a port entry in the CPLPCS scheduler event byte. It uses CYCLE to determine which port to start scanning for the next active port. The sequence of events in CPLPCS is:-

1. CPLPCS is entered by the scheduler.
2. Using CYCLE and the CPLPCS schedule byte it finds next active port that needs servicing.
3. Clear port bit in CPLPCS scheduler byte.
4. Using roll call table find next terminal to poll
5. Using procedure below determine if terminal needs polling or selecting.
6. If it does, enter protocol module at poll or select entry points. Otherwise continue at 8.

7. When protocol module completes poll cycle, return to 8.

8. Set port bit in CPLPCS and exit to scheduler.

The terminal status byte (TMSTAT) in the terminal table is used to determine if the terminal protocol module should be called. It uses the following procedure:-

If TMBDSB is set the terminal is disabled, no polling is done and any transmit buffers on TMTXQN are flushed with the disable flag set in the completion status. If bit TMBSPL is set then CPLPCS increments the Roll Call Table pointer and exits. Otherwise it checks if there are any items on the transmit queue. If there are CPLPCS calls the select routine of the protocol module. If there are no transmit buffers it checks if the poll inactive bit (TMBINA) is set. If not set it calls the protocol module poll routine. Otherwise it increments the roll call table pointer and exits back to the scheduler.

The protocol module is entered at either the poll or select entry points with the registers set as follows:-

IX	-	Terminal table address
IY	-	Port table address
A	-	Not defined
B	-	Terminal number
C	-	SCC data port number
DE	-	Not defined
HL	-	Not defined

The terminal number and terminal type are put into the port table at PTTRMN and PTTYPE.

RTE.A Driver	SECTION 3

3.1 General

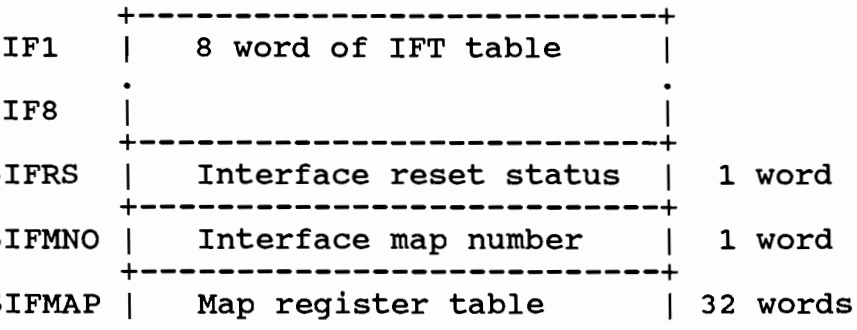
This chapter discusses the interface driver and the device drivers for the 93695V mux card. The interface driver is designed to support the 12040B mux and 12042B PSI cards as well as 93695V mux card. The differences are accommodated by the Z80 software on the cards. This topic is dealt with specifically in the chapter 'PSI and MUX Emulation'.

The interface driver for this card is DDI76. This has two possible device drivers, DDV76 and DDV77. DDV76 connects directly to the Telepac message system, while DDV77 allows access by normal RTE.A EXEC calls. This allows the specialized protocols available on the 93695V mux card to be accessed by general purpose RTE.A programs.

3.2 DVT and IFT Tables

There is one IFT table per interface card. This has an 88 word extent. The first two words contain the interface reset status and the interface map number respectively. The next 32 are the map register values for the 'Data 1 map'. The rest of it contains 18x3 word self configuration packets for DMA transfers.

IFT and IFT extent defintion



```
          +--------------------------+
$ISCFG    |   18x3 word self config  | 54 words
          +--------------------------+
```

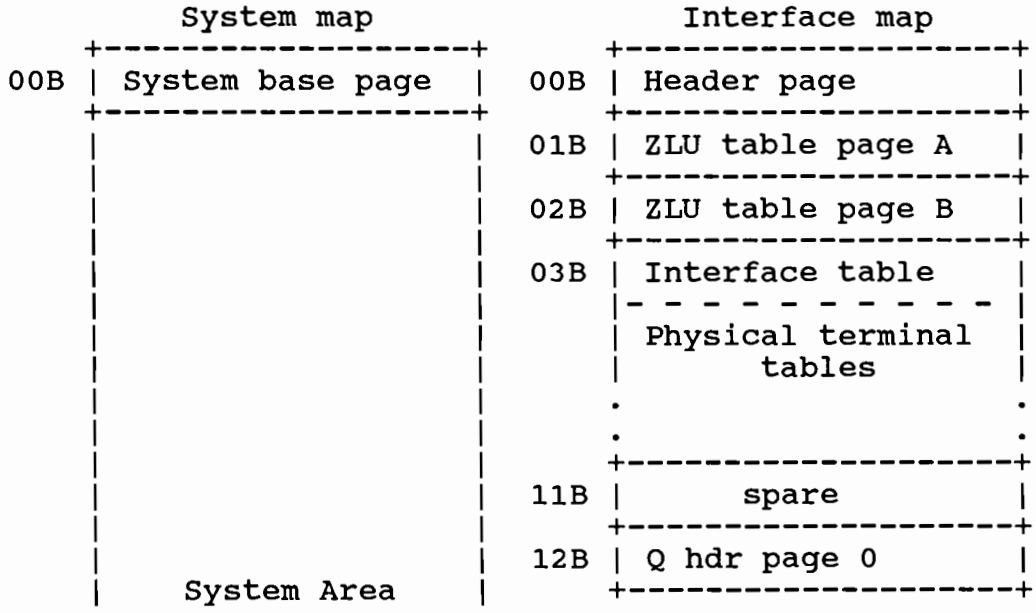
$IFRS - Interface reset status

The reset status indicates the current state of the interface and driver tables.

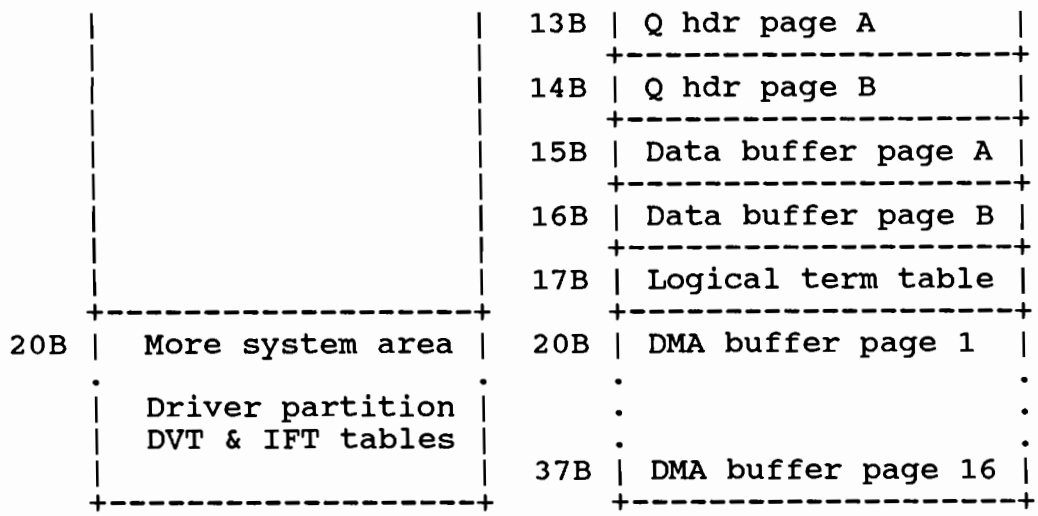
Symbol	Value (Octal)	Meaning .....
IFRSDR	000000B	Driver reset
IFRSDI	000001B	Driver initialized
IFRSIR	000002B	Interface being reset
IFRSIS	000003B	Interface being restarted
IFRSAC	100000B	Interface and driver active

3.3 Mapping Considerations

The interface driver operates with two maps. It executes in the driver partition in the system map and acesses the queues and data area using the interface map in 'Data 1 map'. Within the interface map it sets up the following structure.



Revised <851019.2258>



3.4 Terminal Tables

The terminal tables are originally setup by TTGEN in the memory image file and are subsequently maintained by TTMON. The driver only uses the interface and physical terminal tables. These are described in detail in the section on the Telepac message system.

3.5 Requests to the driver

When using the Telepac message system DDI76 is accessed via device driver DDV76. This is a null device driver, merely passing the commands onto the interface driver. All read and write requests are rejected by the interface driver, and only control requests in the range 30B to 35B are allowed.

Control	-	Description .....
30B	-	Wakeup command from ZCOM module
31B	-	Set driver page number (See ZMON)
32B	-	Reset driver and halt interface card
33B	-	Reset interface card
34B	-	Restart interface card
35B	-	Activate driver

Revised <851019.2258>

All other requests are made directly to the driver internal queues. The requests fall into three groups, port control, terminal control, and terminal data.

3.5.1 Start Up Requests Sequence

When the system is first brought up (cold start), it resets both the driver and the interface card, reloads the firmware, restarts the interface and enables the driver. The following is the sequence of start up requests to the driver:-

Reset driver (32B)
- brings driver to an initial known state
- halts interface card
- returns 1 if driver is not active
- returns 3 if interface does not respond to halt

Set page (31B)
- sets up interface table page number
- returns 8 system generation error detected
- returns 7 ZCOM configuration error detected
- returns 2 if driver has not been reset

Reset interface (33B)
- resets interface card
- returns 3 if no response from card
- returns selftest status if test fails

[Downloads firmware to card, directly by ZMON]

Restart interface (34B)
- starts running MUX firmware
- configures ports
- sets up terminal details
- returns 6 if error occurs during terminal activation

Enable driver (35B)
- enables and activates all terminals (according to TTGEN)
- requeues all write requests on high priority queue
- enables IRQ
- enables driver
- returns 6 if error occurs during terminal activation

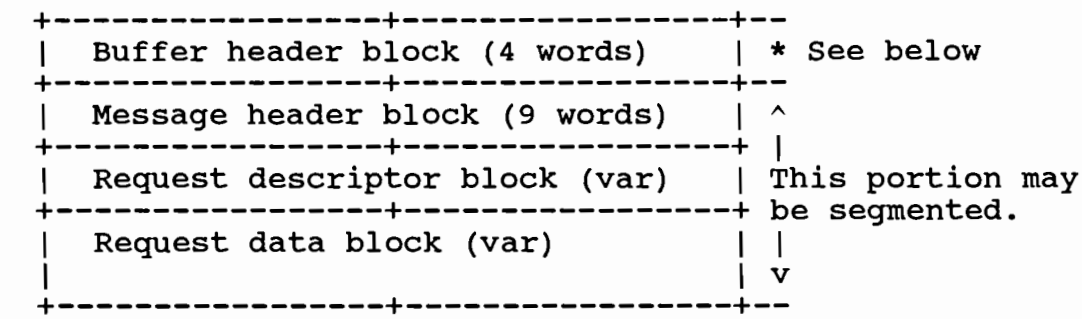
The control commands are to be executed immediately by the driver. At the end of each command, the result is returned in B register. As

usual, a zero return indicates successful completion.

3.5.2 Request buffer overview

The messages are either queued to the interface queue or one of the terminal transmit queues. The messages all have the 18 byte message header (see message header section in Telepac Message System chapter) followed by a request descriptor block, followed by data (if any).

Buffer overview:



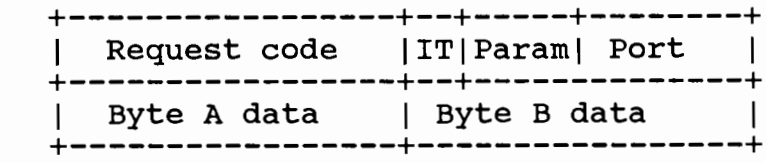
* Note that the buffer header block is repeated for each segment.

The request descriptor blocks for the set of driver requests are shown in detail in the following diagrams.

3.5.3 Port control requests

The port control requests are used to configure the mode of operation of the port. These requests are generated by the ZPORT application program call. They are queued on the interface table queue (INTFCQ). The call can either select the parameters set in the port as the data, or can append new values as part of the request. In the second case the new data values update the port table configuration parameter (IPORTn) in the interface table before the $PORT request is made to the interface card.

Request format:

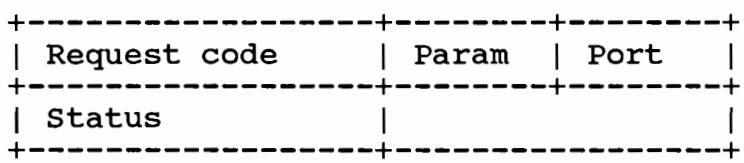


Request code =11 for port control request.

Param, Port, Byte A and B data see $PORT backplane transaction.

The IT bit, when set is used to indicate that the byte A and byte B values should be taken from the interface table.

Reply format:

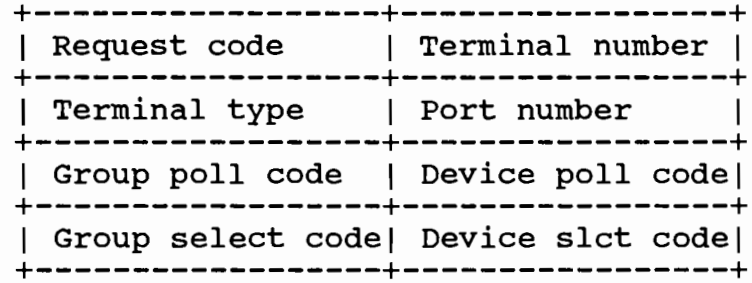


For more information on the type of data in the request and reply see the description of the $PORT backplane transaction in the chapter on the Z80 Software Modules.

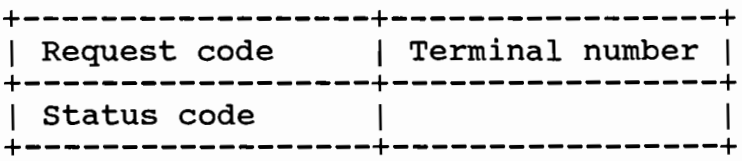
3.5.4 Terminal control requests

The terminal control requests are used control the terminal's mode of operation. They are generated by the ZCNTL application program subroutine call. The requests are queued to the high priority transmit queue (PTTXQA) in the physical terminal table.

Request format:



Reply format:

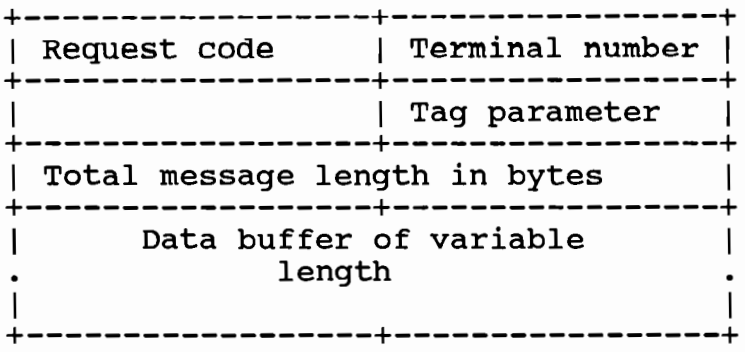


For more information on the requests and request data see the $TERM backplane transaction description in the chapter on the Z80 Software Modules.

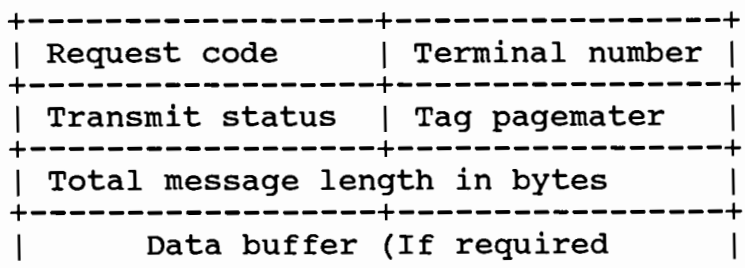
3.5.5 Terminal transmit data messages

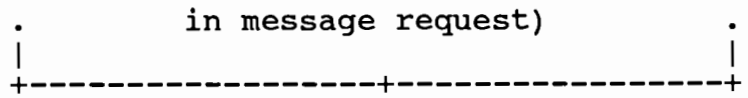
The transmit data messages are generated by the ZSEND application subroutine call. The messages are either queued on the high priority transmit queue (PTTXQA) or the low priority one (PTTXQB) depending on the LPR bit of the response code in the message header.

Request format



Reply format

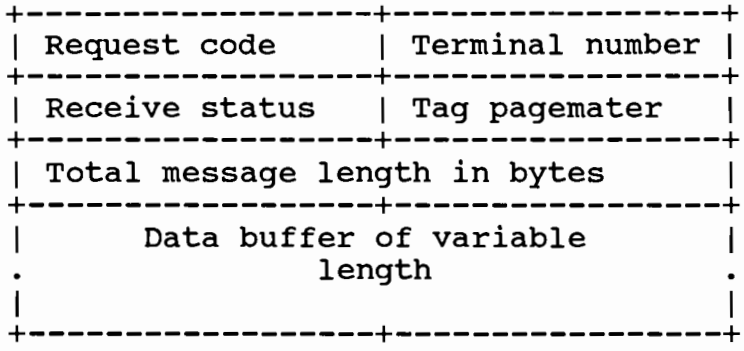




3.5.6 Terminal receive data messages

The receive message only have a reply format that is queued back to the nominated receiving program.

Reply format



3.6 Queueing and program suspension

The messages to the driver are queued directly to the driver tables. If a program must wait for a response or response message before continuing it must be suspended until the process completes. This is done by the ZCOM module initiating the request after the setup is complete. It puts the program into wait state 10B and writes ZLU number of the queue that the program is waiting on into ID segment $TMP1 and $TMP2. The ID segment address of the program is written into bits 14-0 of the parameter word (word 4) of the queue header linked to the ZLU.

When the request completes the driver reschedules waiting program provided it was in wait state 10B. If not then an error message is returned to ZMON and no action is taken.

Revised <851019.2258>