HONEYWELL

# DPS 6 GCOS 6 MOD 400 SYSTEM PROGRAMMER'S GUIDE – VOLUME I

SOFTWARE

# DPS 6
# GCOS 6 MOD 400 SYSTEM
# PROGRAMMER'S GUIDE — VOLUME I
# ADDENDUM A

SUBJECT

Additions and Changes to the Manual

SPECIAL INSTRUCTIONS

This is the first addendum to CZ05-02, dated March 1986. Insert the attached pages into the manual according to the collating instructions on the back of this sheet. Change bars in the margin indicate new or changed information; asterisks indicate deletions.

**Note:**

Insert this cover sheet behind the front cover to indicate the updating of the document with Addendum A.

SOFTWARE SUPPORTED

This manual supports Release 4.0 of the MOD 400 Executive.

ORDER NUMBER

CZ05-02A

September 1986

45987
0986
Printed in U.S.A.

Honeywell

## COLLATING INSTRUCTIONS

To update this manual, remove old pages and insert new pages as follows:

| Remove | Insert |
|---|---|
| iii through xxvi | iii, blank |
| | v through xxii |
| 1-3 through 1-8 | 1-3 through 1-8 |
| 1-11 through 1-14 | 1-11 through 1-14 |
| 2-1, 2-2 | 2-1, 2-2 |
| 3-1 through 3-4 | 3-1 through 3-4 |
| 3-7, 3-8 | 3-7, 3-8 |
| 3-17 through 3-20 | 3-17 through 3-20 |
| ~~4-1 through 4-4~~ | ~~4-1 through 4-4~~ |
| 4-11, 4-12 | 4-11, 4-12 |
| 5-5, 5-6 | 5-5, 5-6 |
| 6-7, 6-8 | 6-7, 6-8 |
| 7-1, 7-2 | 7-1, 7-2 |
| 7-5 through 7-8 | 7-5 through 7-8 |
| 8-1 through 8-4 | 8-1 through 8-4 |
| 8-9 through 8-12 | 8-9 through 8-12 |
| | 8-12.1, blank |
| 8-13, 8-14 | 8-13, 8-14 |
| | 8-14.1, blank |
| 8-17, 8-18 | 8-17, 8-18 |
| | 8-18.1, blank |
| 8-23, 8-24 | 8-23, 8-24 |
| 8-27, 8-28 | 8-27, 8-28 |
| | 8-28.1, blank |
| 8-55, 8-56 | 8-55, 8-56 |
| 8-67, 8-68 | 8-67, 8-68 |
| 8-71 through 8-84 | 8-71 through 8-84 |
| C-23, C-24 | C-23, C-24 |
| D-3, D-4 | D-3, D-4 |
| | h-1, blank |

USER COMMENTS FORMS are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

©Honeywell Information Systems Inc., 1986 File No.: 1R23 09/86 CZ05-02A

# DPS 6 GCOS 6 MOD 400 SYSTEM PROGRAMMER'S GUIDE - VOLUME I

SUBJECT

System Software, Including Executive Routines, Drivers, and Line Protocol Handlers, Accessible to Applications Written in Assembly Language

SPECIAL INSTRUCTIONS

This manual supersedes the *DPS 6 GCOS 6 MOD 400 System Programmer's Guide,* CZ05-01, dated July 1984. The manual has been extensively reorganized, change bars in the margin indicate technical changes; asterisks indicate deletions.

SOFTWARE SUPPORTED

This manual supports Release 4.0 of the MOD 400 Executive.

ORDER NUMBER

CZ05-02

March 1986

**Honeywell**

# *PREFACE*

This manual provides information useful to the Assembly language programmer for designing applications.

The manual describes system services available to the programmer for:

- System control
- Input/output to peripheral devices
- Input/output to communications devices.

The system services described include:

- Executive routines that can be invoked by monitor calls or macro calls
- Drivers servicing peripheral devices
- Line protocol handlers servicing communications devices.

Macro calls mentioned in this volume are described more fully in the System Programmer's Guide, Volume II. Assembly language is described in the Assembly Language Reference manual. Topics related to program preparation, execution, and checkout are described in the Application Developer's Guide.

USER COMMENTS FORMS are included at the back of this manual. These forms are to be used to record any corrections, changes, or additions that will make this manual more useful.

File No.: 1R13, 1S13

CZ05-02

The following symbols are used in this manual to define the format of command and directive lines:

Square brackets [ ] indicate an optional entry.

Braces { } enclose entries from which the user must make a choice.

Lowercase letters (e.g., id) indicate a symbolic variable whose exact value must be supplied by the user.

The character △ (delta) indicates one blank space.

Each section and appendix of this document is structured according to the heading hierarchy shown below. Each heading indicates the relative level of the text that follows it.

| Level | Heading Format |
|---|---|
| 1 (Highest) | ALL CAPITAL LETTERS, UNDERLINED |
| 2 | Initial Capital Letters, Underlined |
| 3 | ALL CAPITAL LETTERS, NOT UNDERLINED |
| 4 (Lowest) | Initial Capital Letters, Not Underlined |

## CONTENTS

## CONTENTS

Page

Page

Page

## CONTENTS

## ILLUSTRATIONS

# *ILLUSTRATIONS*

## *TABLES*

## *TABLES*

## TABLES

# Section 1
# INTRODUCTION

Volume I of the System Programmer's Guide provides general information useful to the Assembly language programmer for designing and executing applications. Sections 2 through 13 of the manual describe services provided by the system that can be invoked or controlled by Assembly language programs. The following subsections describe more specifically the content and organization of the manual.

## SYSTEM SERVICE MACRO CALLS

Sections 2 through 4 describe system services (functions) that can be invoked by macro calls or monitor calls. These are services for system control, file management, record management, and input/output to peripheral and communications devices. Table 1-1 lists alphabetically the macro calls by which system functions can be invoked. Throughout this manual; functions are referred to by their corresponding macro calls.

You can also invoke a function by a monitor call (MCL) instruction followed by the function's code. The function code assigned to each function/macro call is shown in column 3 of Table 1-1.

The manual provides an overview of functions belonging to the same group. In Section 2, for example, all the functions related to semaphores are listed together. Semaphores are defined as a mechanism for the sharing of a resource among members of the same task group. The part played in this mechanism by each of the listed functions is briefly indicated. Thus, the manual informs you of available macro calls and indicates their functional relationship.

Volume II of the System Programmer's Guide, by contrast, describes each macro call individually. The individual descriptions provide information (relating to macro call arguments and register contents) that enables the user to actually employ the call in the application.

## DEVICE DRIVERS AND LINE PROTOCOL HANDLERS

Section 6 describes the system software used for transmitting data between applications and peripheral (non-communications) devices. The section deals mainly with the data structures and codes by which the user instructs the device drivers and by which the drivers report the status of requested operations. (Macro calls related to input/output are discussed in earlier sections.)

Section 7 provides an overview of line protocol handlers, which are used for transmitting data between applications and communications devices. Sections 8 through 13 describe in detail the ATD, STD, PVE, BSC, TTY, BTF, and 3270 Terminal Facility line protocol handlers.

Programming considerations, such as trap handling and calling external procedures, are described in the appendices.

Table 1-1. System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $ABGRP | Abort group | 0D/0A | Task group control |
| $ABGRQ | Abort group request | 0D/07 | Task group control |
| $ACTID | Account identification | 14/02 | Identification and information |
| $ACTVG | Activate group | 0D/09 | Task group control |
| $BUAT | Bound unit, attach | 0C/09 | Task control |
| $BUDT | Bound unit, detach | 0C/0B | Task control |
| $BUID | Bound unit identification | 14/06 | Identification and information |
| $BULD | Bound unit, load | 0C/0A | Task control |
| $BUXFR | Bound unit transfer | 0C/07 | Task control |
| $CANRQ | Cancel request | 0C/01 | Task control |
| $CIN | Command in | 08/02 | Standard system file I/O |
| $CKPFL | Checkpoint file | 0D/11 | File management |
| $CKPT | Checkpoint | 0D/0F | Task group control |
| $CLFIL | Close file | 10/55-10/57 | File management |
| $CLPNT | Clean point | 0C/13 | File management |
| $CLRSW | Clear external switches | 0B/02 | External switch |
| $CMDLN | Command line process | 0C/08 | Task control |
| $CMSUP | Console message suppression | 09/02,09/03 | Operator interface |
| $CNCRQ | Cancel clock request | 05/01 | Clock |

*

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $CNSRQ | Cancel semaphore request | 06/01 | Semaphore handling |
| $CRB | Clock request block | - | Data structure generation |
| $CRBD | Clock request block offsets | - | Data structure generation |
| $CRDIR | Create directory | 10/A0 | File management |
| $CRFIL | Create file | 10/30 | File management |
| $CRGRP | Create group | 0D/03 | Task group control |
| * $CROAT | Create overlay area table | 07/0A | Overlay handling |
| $CRPSB | Create file parameter structure block offsets | - | Data structure generation |
| $CRRDB | Create file record descriptor block offsets | - | Data structure generation |
| $CRSEG | Create segment | 0C/0C | Task control |
| $CRTSK | Create task | 0C/02,0C/03 | Task control |
| $CWDIR | Change working directory | 10/B0 | File management |
| $DFCKP | Defer checkpoint | 0C/19 | Task control |
| $DFRHD | Defer request on head | 01/0D | Request and Return |
| $DFRTL | Defer request on tail | 01/0C | Request and Return |
| $DFSM | Define semaphore | 06/04 | Semaphore handling |

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $DIPSB | Device information parameter structure block offsets | - | Data structure generation |
| $DLDIR | Delete directory | 10/A5 | File management |
| $DLFIL | Delete file | 10/35 | File management |
| $DLGRP | Delete group | 0D/04 | Task group control |
| $DLOAT | Delete overlay area table | 07/0D | Overlay handling |
| $DLREC | Delete record | 11/30,11/31 | Data management |
| $DLSEG | Delete segment | 0C/0D | Task control |
| $DLSM | Delete semaphore | 06/07 | Semaphore handling |
| $DLTSK | Delete task | 0C/04 | Task control |
| $DQPST | Dequeue and post | 01/0B | Request and Return |
| $DSTRP | Disable user trap | 0A/02 | Trap handling |
| $ELEND | Error logging end | 02/09 | Physical I/O |
| $ELEX | Error logging information, exchange | 02/07 | Physical I/O |
| $ELGT | Error logging information, get | 02/08 | Physical I/O |
| $ELOG | Error logging table | - | Data structure generation |
| $ELST | Error logging, start | 02/05 | Physical I/O |
| $ENTID | Entry point identification | 14/07 | Identification and information |
| $ENTRP | Enable user trap | 0A/01 | Trap handling |

*

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $EROUT | Error out | 08/03 | Standard system file I/O |
| $EXTDT | External date/time, convert to | 05/04 | Date/time |
| $EXTET | External elapsed time, convert to | 05/0D | Date/time |
| $EXTIM | External time, convert to | 05/05 | Date/time |
| $FIB | File information block | - | Data structure generation |
| $FIBDM | File information block offsets (data management access) | - | Data structure generation |
| $FIBSM | File information block offsets (storage management access) | - | Data structure generation |
| $GAFIL | Get file access rights | 10/7C | File management |
| $GAPSB | Get file access rights parameter structure block offsets | - | Data structure generation |
| $GDTM | Get date/time | 05/06 | Date/time |
| $GIDEV | Get device information | 10/66 | File management |
| $GIFAB | Get file information, file attribute block offsets | - | Data structure generation |
| $GIFIL | Get file information | 10/60 | File management |

*

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $GIPSB | Get file information, parameter structure block offsets | - | Data structure generation |
| $GMEM | Get memory/get available memory | 04/02,04/03 | Memory allocation |
| $GNFIL | Get name | 10/3C | File management |
| $GNPSB | Get names parameter structure block offsets | - | Data structure generation |
| $GRFIL | Grow file | 10/38 | File management |
| $GRPID | Group identification | 14/08 | Identification and information |
| $GRPSB | Grow file parameter structure block offsets | - | Data structure generation |
| $GTACT | Get file accounting information | 10/42 | File management |
| $GTFIL | Get file | 10/20 | File management |
| $GTPSB | Get file parameter structure block offsets | - | Data structure generation |
| $GWDIR | Get working directory | 10/C0 | File management |
| $HDIR | Home directory | 14/0B | Identification and information |
| $INDTM | Internal date/time, convert to | 05/07 | Date/time |
| $INSID | Installation identification | 14/05 | Identification and information |
| $IORB | Input/output request block | - | Data structure generation |

*

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $IORBD | Input/output request block offsets | - | Data structure generation |
| $KILLT | Kill (abort) task | 0C/11 | Task control |
| $LKFIL | Link file | 10/3A | File management |
| $LKNME | Link Name | 10/47 | File management |
| $MACPT | Message group, accept | 15/01 | Intergroup message facility |
| $MCME | Message group, cancel enclosure | 15/06 | Intergroup message facility |
| $MCMG | Message group, count | 15/07 | Intergroup message facility |
| $MDFIL | Modify file | 10/41 | File management |
| $MDPSB | Modify file parameter structure block offsets | - | Data structure generation |
| $MGCRB | Message group, control request block | - | Data structure generation |
| $MGCRT | Message group control request block offsets | - | Data structure generation |
| $MGIRB | Message group, initialization request block | - | Data structure generation |
| $MGIRT | Message group initialization request block offsets | - | Data structure generation |
| $MGRRB | Message group, recovery request block | - | Data structure generation |
| $MGRRT | Message group recovery request block offsets | - | Data structure generation |

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $MINIT | Message group, initiate | 15/02 | Intergroup message facility |
| $MODID | Mode identification | 14/03 | Identification and information |
| $MRECV | Message group, receive | 15/03 | Intergroup message facility |
| $MSEND | Message group, send | 15/05 | Intergroup message facility |
| $MTMG | Message group, terminate | 15/04 | Intergroup message facility |
| $NCIN | New command in | 08/06 | Standard system file I/O |
| $NMLF | New message library | 08/08 | Standard system file I/O |
| $NPROC | New process | 0D/0B | Task group control |
| $NUIN | New user input | 08/04 | Standard system file I/O |
| $NUOUT | New user output | 08/05 | Standard system file I/O |
| $OPFIL | Open file | 10/50,10/51 | File management |
| $OPMSG | Operator information message | 09/00 | Operator interface |
| $OPRSP | Operator response message | 09/01 | Operator interface |
| $OVEXC | Overlay, execute | 07/00 | Overlay handling |
| $OVLD | Overlay, load | 07/01 | Overlay handling |
| $OVRCL | Overlay release, wait, and recall | 07/07 | Overlay handling |
| $OVRLD | Overlay area, reserve, and load overlay | 07/10 | Overlay handling |

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $OVRLS | Overlay area, release | 07/06 | Overlay handling |
| $OVRSV | Overlay area reserve, and execute overlay | 07/05 | Overlay handling |
| $OVST | Overlay status | 07/03 | Overlay handling |
| $OVUN | Overlay, unload | 07/0C | Overlay handling |
| $PERID | Person identification | 14/01 | Identification and information |
| $PPNTL | Postpone request on tail | 01/0E | Request and Return |
| $PRBLK | Parameter block | - | Data structure generation |
| $PRFAU | Profile record, accounting update | 24/42 | User registration |
| $PRFCR | Profile record, create | 24/20 | User registration |
| $PRFDL | Profile record, delete | 24/30 | User registration |
| $PRFGT | Profile record, get | 24/10 | User registration |
| $PRFIF | Profile record, get user information | 24/12 | User registration |
| $PRFUP | Profile record, update | 24/40 | User registration |
| $RBADD | Return request block address | 01/07 | Request and return |
| $RBD | Request block displacements | - | Data structure generation |
| $RBOOT | Reboot | 20/06 | Software reboot |
| $RBPRM | Modify reboot parameters | 20/05 | Software reboot |

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $RCLHD | Recall from head | 01/0F | Request and return |
| $RDBLK | Read block | 12/00-12/04 | Storage management |
| $RDREC | Read record | 11/10-11/16, 11/19 | Data management |
| $RDSW | Read external switches | 0B/00 | External switch |
| $RETRN | Return | - | Request and return |
| $RLDMP | Unlock dumpfile | 20/04 | Software reboot |
| $RLSM | Release semaphore | 06/03 | Semaphore handling |
| $RLTML | Release terminal | 17/04 | Terminal function |
| $RMEM | Return memory/return partial block of memory | 04/04,04/05 | Memory allocation |
| $RMFIL | Remove file | 10/25 | File management |
| $RNFIL | Rename file/rename directory | 10/40 | File management |
| $ROLBK | Roll back (recover) files | 0C/14 | File management |
| $RPDFC | Report message, display formatting and control | 0F/04 | Message reporter |
| $RPMSG | Report message | 0F/03 | Message reporter |
| $RQCL | Request clock | 05/00 | Clock |
| $RQGRP | Request group | 0D/00 | Task group control |
| $RQIO | Request I/O | 02/00 | Physical I/O |
| $RQSM | Request semaphore | 06/00 | Semaphore handling |
| $RQSPT | Request specific terminal | 17/02 | Terminal function |

*

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $RQTML | Request terminal | 17/03 | Terminal function |
| $RQTSK | Request task | 0C/00 | Task control |
| $RS | Restart | 0D/10 | Task control |
| $RSTID | Reset task identifier | 10/05 | File management |
| $RSVSM | Reserve semaphore | 06/02 | Semaphore handling |
| $RWREC | Rewrite record | 11/40,11/41 | Data management |
| $RVFPW | Reverify password | 24/01 | User registration |
| $SDL | Set dial | 1B/00 | Communications |
| $SETSW | Set external switches | 0B/01 | External switch |
| $SGRPA | Set group attributes | 0D/13 | Task group control |
| $SGTRP | Signal trap | 0A/03 | Trap handling |
| $SHCS | Shrink created segment | 0C/25 | Task control |
| $SHFIL | Shrink file | 10/37 | File management |
| $SHGWS | Shrink group work segment | 0D/16 | Task group control |
| $SHPSB | Shrink file parameter structure block offsets | - | Data structure generation |
| $SPGRP | Spawn group | 0D/05 | Task group control |
| $SPTSK | Spawn task | 0C/05,0C/06, 0C/15 | Task control |
| $SRB | Semaphore request block | - | Data structure generation |
| $SRBD | Semaphore request block offsets | - | Data structure generation |
| $STMP | Status memory pool | 04/06 | Memory allocation |

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $STFIL | Set terminal file characteristics | 10/46 | File management |
| $STTID | Set task identifier | 10/00 | File management |
| $SUSPG | Suspend group | 0D/08 | Task group control |
| $SUSPN | Suspend for interval; suspend until time | 05/02,05/03 | Clock |
| $SWFIL | Swap file | 10/5A | File management |
| $SYSAT | System attribute information, get | 14/11 | Identification and information |
| $SYSID | System identification | 14/04 | Identification and information |
| $TEST | Test completion status | 01/02 | Request and return |
| $TFIB | File information block offsets (data and storage management access) | - | Data structure generation |
| $TGIN | Task group input | 14/0C | Identification and information |
| $TIFIL | Test file for input | 10/62 | File management |
| $TOFIL | Test file for output | 10/63 | File management |
| $TRB | Task request block | - | Data structure generation |
| $TRBD | Task request block offsets | - | Data structure generation |
| $TRMRQ | Terminate request | 01/03,01/04 | Request and return |
| $TRPHD | Trap handler connect | 0A/00 | Trap handling |
| $ULFIL | Unlink file | 10/3B | File management |
| $ULNME | Unlink name | 10/48 | File management |

Table 1-1 (cont). System Service Macro Calls

| Macro Call Name (1) | Function Description (2) | Function Code (3) | Function Group (4) |
|---|---|---|---|
| $USIN | User input | 08/00 | Standard system file I/O |
| $USOUT | User output | 08/01 | Standard system file I/O |
| $USRID | User identification | 14/00 | Identification and information |
| $VLCKP | Validate checkpoint | 0D/12 | Task group control |
| $XPCS | Expand created segment | 0C/24 | Task control |
| $WAIT | Wait | 01/00 | Request and return |
| $WAITA | Wait any | 01/01 | Request and return |
| $WAITL | Wait on request list | 01/01 | Request and return |
| $WAITM | Wait on multiple requests | 01/01 | Request and return |
| $WIFIL | Wait file (input) | 10/64 | File management |
| $WLIST | Wait list generate | - | Data structure generation |
| $WLSTM | Wait list, generate multiple | - | Data structure generation |
| $WOFIL | Wait file (output) | 10/65 | File management |
| $WRBLK | Write block | 12/10,12/11 | Storage management |
| $WRREC | Write record | 11/20-11/26 | Data management |
| $WTBLK | Wait block | 12/20 | Storage management |
| $XFERU | Transfer user | 17/06 | Terminal function |
| $XPATH | Expand pathname | 10/D0 | File management |
| $XRETU | Transfer and return user | 17/07 | Terminal function |

# Section 2
# SYSTEM CONTROL FUNCTIONS

This section summarizes and briefly describes the system control macro calls that provide user access to system control functions. The macro calls are presented according to their functional groupings (see Table 1-1, column 4) as follows: *

| | |
|---|---|
| Clock | Physical I/O |
| Communications | Request and Return |
| Date/Time | Semaphore Handling |
| External Switch | Software Reboot |
| Identification and Information | Standard System File I/O |
| Intergroup Message Facility | Task Control |
| Memory Allocation | Task Group Control |
| Message Reporter | Terminal Control |
| Operator Interface | Trap Handling |
| Overlay Handling | User registration |

See Volume II of this manual for a detailed description of each macro routine/call.

*

*

## CLOCK FUNCTIONS

The macro calls for clock functions to allow user control of task execution according to an elapsed time period. These macro calls use the clock manager. The clock manager is a system component whose primary function is satisfying/completing task requests at a specified time or after a specified interval.

The clock manager services interrupts from the real-time clock. At each interrupt, the clock manager ascertains whether the time interval associated with a request to initiate execution of the task has been satisfied. Depending on information contained in the clock request block (see Appendix C), the system will do one of the following:

- Activate a task
- Schedule an indicated request block
- Release a semaphore.

The clock macro calls act to:

- Connect a clock request block to the timer queue
- Disconnect a clock request block from the timer queue
- Suspend the issuing task until an interval of time has passed
- Suspend the issuing task until a given date/time.

The clock function macro calls are:

| | |
|---|---|
| ● Cancel Clock Request | $CNCRQ |
| ● Request Clock | $RQCL |
| ● Suspend for Interval | $SUSPN |
| ● Suspend Until Time | $SUSPN |

Volume II describes the Clock Request Block ($CRB) macro call, which generates a clock request block.

## COMMUNICATIONS FUNCTIONS

The macro call for communications functions allows the user to set a telephone number to be used for automatic dialing. The macro routine/call is:

Set Dial $SDL

Section 4 discusses macro calls, other than Set Dial, applicable to communications processing.

## DATE/TIME FUNCTIONS

The macro calls for date/time functions allow the user to:

- Obtain the current internal date/time value
- Convert the internal date/time value to external date/time format
- Convert the internal date/time value to external time format
- Convert an external date/time value to internal format.

The date/time macro calls are:

| | |
|---|---|
| ● External Date/Time, Convert to | $EXTDT |
| ● External Time, Convert to | $EXTIM |
| ● External Elapsed Time, Convert to | $EXTET |
| ● Get Date/Time | $GDTM |
| ● Internal Date/Time, Convert to | $INDTM |

## EXTERNAL SWITCH FUNCTIONS

A task group can control its own execution by using external switch function macro calls to modify its external switches. An external switch operates much like a hardware switch on an operator's control panel. External switches can be set and cleared with the Modify Switches (MSW) command or with the $SETSW and $CLRSW macro calls.

An external switch word is associated with each task group. Each bit in the word corresponds to an external switch. Thus, each task group can manipulate 16 switches. A user program can contain instructions or statements to determine the settings of one or more of these switches. The program can then set or clear these settings to control its execution logic.

The macro calls allow the issuing task to:

- Set switches
- Clear switches
- Read the current values of the switches.

The macro calls are:

| | |
|---|---|
| ● Clear External Switches | $CLRSW |
| ● Read External Switches | $RDSW |
| ● Set External Switches | $SETSW |

## IDENTIFICATION AND INFORMATION FUNCTIONS

The macro calls for identification and information make available to the user the following information concerning the current task or task group:

| Information | Macro Call |
|---|---|
| Home directory pathname | $HDIR |
| Bound unit identification | $BUID |
| Installation identification | $INSID |
| System identification | $SYSID |
| Task group account identification | $ACTID |
| Task group input file name | $TGIN |
| Task group mode identification | $MODID |
| Task group person identification | $PERID |
| Task group user identification | $USRID |
| Entry point identification | $ENTID |
| Group identification | $GRPID |
| System attribute information | $SYSAT |

## INTERGROUP MESSAGE FACILITY FUNCTIONS

The message facility allows the task groups to exchange messages through a message queue called a mailbox. Before messages can be transmitted, the mailbox must have been created by means of the Create Mailbox command. Mailboxes are described in detail in the System User's Guide.

A message text consists of several nested units, or enclosures. The smallest unit is a record; the next largest unit, made up of records, is a quarantine unit; the largest, made up of quarantine units, is a message. A quarantine unit is the smallest amount of transmitted data that is available to the receiver. Because a message can comprise a group of records, it is called a "message group."

The transfer of messages is facilitated by three request blocks: message group control request block (MGCRB), message group initialization request block (MGIRB), and message group recovery request block (MGRRB). These data structures are tabulated in Appendix C and described in Volume II.

Message facility macro calls perform the following:

- Initialize communications between groups by setting values of the message group initialization request block (MGIRB)
- Validate the acceptor's access to an existing mailbox
- Ascertain the number of messages in a mailbox
- Identify the specific message to be accepted

- Request the receipt of a message, specifying values for the message group control request block (MGCRB)

- Delete the last record in an incomplete quarantine unit or delete the quarantine unit itself

- Send a message group

- Terminate a message group, normally or abnormally.

The message facility macro calls are:

| | |
|---|---|
| ● Message Group, Initiate | $MINIT |
| ● Message Group, Accept | $MACPT |
| ● Message Group, Count | $MCMG |
| ● Message Group, Receive | $MRECV |
| ● Message Group, Cancel Enclosure | $MCME |
| ● Message Group, Send | $MSEND |
| ● Message Group, Terminate | $MTMG |

## MEMORY ALLOCATION FUNCTIONS

The macro calls for memory allocation functions allow the user to dynamically obtain memory from the task group's memory pool, to return this memory when it is no longer needed, and to ascertain the amount of memory available in a specified pool.

The macro call that allocates a memory block has two forms: one form obtains a memory block of the specified size only; the other obtains the largest existing contiguous memory block if a block of the specified size cannot be found. The macro call that returns a memory block also has two forms: one form returns an entire memory block; the other returns a specified part of the block.

The macro calls are:

| | |
|---|---|
| ● Get Memory/Get Available Memory | $GMEM |
| ● Return Memory/Return Partial Block of Memory | $RMEM |
| ● Status Memory Pool | $STMP |

## MESSAGE REPORTER FUNCTIONS

The macro calls for message reporting allow an application to display error or help messages at the user's terminal.

The macro calls specify the code of a message that the Message Reporter retrieves from a message library.

The message reporting macro calls allow an application to:

- Display chained messages (i.e., after viewing the first message in the chain, the user can request further information)

- Substitute arguments for parameters in the message text (e.g., specify a device name in a "device disabled message")

- Return messages to an application buffer rather than to a terminal

- Display messages at terminals running in any of the following modes:

  - Command
  - Menu
  - Display formatting and control

The message reporting macro calls are:

- Report Message ($RPMSG)
- Report Message, Display Formatting and Control ($RPDFC).

## OPERATOR INTERFACE FUNCTIONS

The macro calls for operator interface functions enable tasks to communicate with the operator terminal by:

- Displaying a message on the operator terminal

- Sending a message to the operator terminal and receiving a response

- Activating or deactivating console suppression; i.e., suspending or restoring issuance of messages to the operator terminal for the issuing task group.

The macro calls are:

| | |
|---|---|
| ● Console Message Suppression | $CMSUP |
| ● Operator Information Message | $OPMSG |
| ● Operator Response Message | $OPRSP |

The $OPMSG and $OPRSP macro calls require input/output request blocks (IORBs), which can be generated by the $IORB macro call. (Section 5 describes request blocks in general, Appendix C describes the IORB in detail, and Volume II describes the $IORB macro call.)

## OVERLAY HANDLING FUNCTIONS

Overlay handling calls locate, load, execute, and unload fixed and floatable overlays. Fixed overlays are loaded into memory at a displacement from the base of the root segment fixed at link time. Floating overlays are loaded as follows: If a bound unit can be shared between task groups (i.e., is linked as globally sharable), its floating overlays are loaded into system memory; otherwise, floating overlays are loaded into any sufficient block of the issuing task's task group memory.

When bound units with fixed overlays are loaded, enough space is reserved in memory so that the linked, fixed overlay with the highest address can be loaded. Overlay handling calls similarly reserve overlay areas for floating overlays. Overlay areas are areas in memory of fixed size that accommodate the largest floating overlay associated with a bound unit. Overlay areas are managed by means of overlay area tables (OATs), which ensure that space in overlay areas is occupied only by overlays that are currently in use. Thus, overlay handling functions relieve the user of writing an overlay manager.

The overlay handling macro calls are:

| | |
|---|---|
| ● Overlay, Release, Wait, and Recall | $OVRCL |
| ● Overlay Area, Release | $OVRLS |
| ● Overlay Area, Reserve, and Execute Overlay | $OVRSV |
| ● Overlay Area, Reserve, and Load Overlay | $OVRLD |
| ● Create Overlay Table | $CROAT |
| ● Delete Overlay Table | $DLOAT |
| ● Overlay, Execute | $OVEXC |
| ● Overlay, Load | $OVLD |
| ● Overlay, Status | $OVST |
| ● Overlay, Unload | $OVUN |

## PHYSICAL I/O FUNCTIONS

The Request I/O ($RQIO) macro call, used in conjunction with the input/output request block (IORB), allows direct control by the user of device drivers or communication line protocol handlers. If direct access to devices is not a requirement, File System macro calls provide a more convenient means of handling input/output operations.

See Sections 6 and 7 for a complete description of physical I/O functions, including details on device drivers and line protocol handlers.

The macro routine/call for physical I/O is:

| | |
|---|---|
| Request I/O Transfer | $RQIO |

## REQUEST AND RETURN FUNCTIONS

The macro calls for request and return functions enable an issuing task to perform the following:

- Ascertain the address of the first request block in the queue of requests placed against it

- Ascertain the completion status of request blocks placed against it

- Defer the processing of a request placed against it

- Terminate the request that it is processing, marking it as completed
- Wait for the completion of its own request(s) before resuming execution
- Issue a common return sequence for called subroutines.

When a task defers the processing of a request placed against it, it dequeues the request and requeues it at a specified priority level on its request queue. (This priority level is not to be confused with the priority level, or interrupt level, at which the task is running.) The deferred request is requeued at either the head or tail of any other requests deferred at the specified priority level. The capability of deferring a request is typically used by device drivers in order to give precedence to one type of request over another type.

The macro calls for request and return functions are:

| | |
|---|---|
| • Dequeue and Post | $DQPST |
| • Defer Request on Tail | $DFRTL |
| • Defer Request on Head | $DFRHD |
| • Postpone Request on Tail | $PPNTL |
| • Recall from Head | $RCLHD |
| • Return Request Block Address | $RBADD |
| • Return | $RETRN |
| • Terminate Request | $TRMRQ |
| • Test Completion Status | $TEST |
| • Wait Any | $WAITA |
| • Wait for Operation to Complete | $WAIT |
| • Wait on Request List | $WAITL |
| • Wait on Multiple Request List | $WAITM |

Section 5 and Volume II describe the macro calls for generating request blocks. Appendix C shows request block formats.

## SEMAPHORE HANDLING FUNCTIONS

A semaphore is a mechanism for coordinating the use of resources within task groups. Once defined, semaphores control access to multiple resources and control multiple requests for the same resource.

A semaphore is defined for each resource to be controlled and is given a 2-character ASCII semaphore name, which is a system symbol recognized by the Monitor. Every requestor of a resource whose use must be coordinated issues appropriate Monitor calls to the named semaphore to request or release the resource. The task that defines the semaphore assigns the semaphore's initial value. The monitor increments or decrements this initial value when the resource is released or requested/reserved, respectively.

A requestor obtains use of a resource if the semaphore value is greater than zero at the time of the request. If the value is zero or negative, the requestor either waits until the resource becomes available or continues executing, depending upon the macro call issued to make the request. The initial value of the semaphore determines the number of users who can utilize a resource at a given time. An initial value of 2 allows two simultaneous users, an initial value of 4 allows four users, etc.

Semaphore function macro calls are used to:

- Define a semaphore and set its initial value
- Increment the current-value counter
- Decrement the current-value counter
- Queue a semaphore request block if the requested resource is not available
- Remove a semaphore request block from its queue
- Delete a semaphore.

The macro calls for semaphore handling are:

| | |
|---|---|
| ● Cancel Semaphore Request | $CNSRQ |
| ● Define Semaphore | $DFSM |
| ● Release Semaphore | $RLSM |
| ● Request Semaphore | $RQSM |
| ● Reserve Semaphore | $RSVSM |
| ● Delete Semaphore | $DLSM |

## SOFTWARE REBOOT

The Software Reboot Facility reinitializes the system without operator intervention. It is activated dynamically by exhaustion of trap save areas or indirect request blocks, and by Watchdog Timer timeouts. The user can direct that a dump be taken before reinitialization of the system.

The Software Reboot routines/calls are:

| | |
|---|---|
| ● Modify Reboot Parameters | $RBPRM |
| ● Reboot | $RBOOT |
| ● Unlock Dumpfile | $RLDMP |

## STANDARD SYSTEM FILE I/O FUNCTIONS

A task group can access standard system files (command-in, user-in, user-out, error-out, and message library) through standard system file I/O macro calls. Other macro calls shown below allow the task to redefine certain standard system files. Specifically, the macro routines enable a task to:

- Read the next record from the command-in file
- Write the next record to the error-out file
- Read the next record from the user-in file
- Write the next record to the user-out file
- Redefine the user-in file
- Redefine the user-out file
- Redefine the message library file.

The macro calls are:

| | |
|---|---|
| • Command In (read command-in file) | $CIN |
| • Error Output File | $EROUT |
| • New Command In | $NCIN |
| • New Message Library File | $NMLF |
| • New User Input File | $NUIN |
| • New User Output File | $NUOUT |
| • User Input File | $USIN |
| • User Output File | $USOUT |

## TASK CONTROL FUNCTIONS

The macro calls for task control allow the user to:

- Cancel a previously issued request

- Create, request, spawn, suspend, activate, delete, and abort a task

- Attach, load, transfer, and detach a bound unit to/from a task

- Create and delete a segment for a task's bound unit

- Expand and shrink a created segment

- Process command lines

- Roll back (recover) updated records in all files updated since the last execution of Clean Point

- Declare a "clean point" at which

  - Updates made to records are complete
  - Updated records are written to disk
  - The updated file is considered to be in a consistent state
  - Records previously locked by the issuing task are unlocked.

Macro calls for task control are:

| | |
|---|---|
| • Cancel Request | $CANRQ |
| • Clean Point | $CLPNT |
| • Command Line, Process | $CMDLN |
| • Create Segment | $CRSEG |

| | |
|---|---|
| • Delete Segment | $DLSEG |
| • Expand Created Segment | $XPCS |
| • Shrink Created Segment | $SHCS |
| • Create Task | $CRTSK |
| • Delete Task | $DLTSK |
| • Request Task | $RQTSK |
| • Spawn Task | $SPTSK |
| • Bound Unit, Attach | $BUAT |
| • Bound Unit, Load | $BULD |
| • Bound Unit, Detach | $BUDT |
| • Bound Unit, Transfer | $BUXFR |
| • Kill Task | $KILLT |
| • Roll Back | $ROLBK |

## TASK GROUP CONTROL FUNCTIONS

A task group is a named set of one or more tasks, memory space, files, peripheral devices, and priority levels. Any number of task groups may be defined. (Task groups and tasks are explained in detail in the System Concepts manual.)

The macro calls for task group control allow the user to:

- Create, spawn, request, or delete a task group
- Enable or disble certain functionalities (e.g., message chaining, ready prompt) for a task group
- Terminate a current task group and restart a task group request
- Abort a task group request
- Terminate a user session
- Declare a checkpoint from which processing can be restarted after premature termination of a group request
- Assign or disassign checkpoint files to a task group
- Abort a task group
- Terminate a user session.
- Reduce a task group's memory requirement by shrinking its group work segment (GWS).

A task executing under one group can initiate another group. First, a task group must be defined in order to create task group control structures and load the bound-unit root segment as the lead task. Then, a group request must be issued to activate the lead task for execution. Tasks can be executed concurrently in this task group with the use of control functions or commands.

The task group can be deleted; no more requests can be made against this group after it has been marked for deletion. When all tasks in the group terminate and become dormant, all memory associated with the group is returned to its memory pool, becoming available to other groups.

The several phases of task creation, activation, and deletion occur in sequence when a Spawn Task Group macro call is issued.

A task can suspend a task group's execution and then activate that task group.

A task can terminate the current group request and then restart the processing of the original task group request.

Aborting a task group deletes the group immediately, before all its tasks terminate and become dormant.

A task can terminate a user session, then either restart the group request, begin a new login sequence, or disconnect the user terminal.

A task can abort the current request for the activation of a specified group. In this case, the next request (if any) against that group will be processed.

Some macro calls listed below use a parameter block, which extends the argument list of the task request block. The macro call that generates parameter blocks ($PRBLK) is described in Volume II; block format is shown in Appendix C.

The macro calls for task group control are:

| | |
|---|---|
| ● Abort Group | $ABGRP |
| ● Abort Group Request | $ABGRQ |
| ● Activate Group | $ACTVG |
| ● Checkpoint | $CKPT |
| ● Checkpoint File | $CKPFL |
| ● Create Group | $CRGRP |
| ● Delete Group | $DLGRP |
| ● New Process | $NPROC |
| ● Request Group | $RQGRP |
| ● Set Group Attributes | $SGRPA |
| ● Spawn Group | $SPGRP |
| ● Suspend Group | $SUSPG |
| ● Shrink Group Work Segment | $SHGWS |

## TERMINAL CONTROL FUNCTIONS

Terminal control functions allow secondary logins and the transfer of primary or secondary users between task groups.

When someone logs into the system as a secondary user, the Listener component attaches a secondary user's terminal to an existing task group if the user, when logging in, specifies the task group and if that task group has requested a secondary terminal.

The macro calls for terminal control functions permit:

- The task group to request any secondary terminal
- The task group to request a specific secondary terminal
- The task group to transfer a user to Listener, along with a new login line that automatically associates the user with another task group
- The task group to transfer a user, along with a new login line, to Listener, which later returns the user to the task group
- The task group to release a secondary terminal.

The appropriate macro calls are:

| | |
|---|---|
| ● Request Specific Terminal | $RQSPT |
| ● Request Terminal | $RQTML |
| ● Release Terminal | $RLTML |
| ● Transfer and Return User | $XRETU |
| ● Transfer User | $XFERU |

## TRAP HANDLING FUNCTIONS

The macro calls for trap functions allow an application to designate the traps to be handled during its execution. Specifically, the macro calls allow the user to:

- Connect a user-written, generalized trap handling routine to a task
- Enable a specific trap or all traps
- Disable a specific trap or all traps.

Additionally, the user can transmit a software-generated trap condition to a specific task.

Appendix A describes traps and trap handling in detail.

The macro calls for trap handling are:

| | |
|---|---|
| ● Disable User Trap | $DSTRP |
| ● Enable User Trap | $ENTRP |
| ● Trap Handler Connect | $TRPHD |
| ● Signal Trap | $SGTRP |

## USER REGISTRATION FUNCTIONS

User registration functions enable a user to be registered in one or more subsystems, such as forms processing or networking. These functions create, retrieve, modify, and delete a subsystem record that establishes the user's access to a subsystem and contains various statistics.

Before a user's subsystem record can be created, the user must be registered in the system (as distinct from the subsystem) by the system administrator. To register a user in the system, the administrator creates a registration record by means of the Edit Profile utility. One user registration function, Profile Record, Get User Information ($PRFIF), retrieves limited information from the registration record. The subsystem and registration records belong to the profiles file, which is the system's user registration data base.

Using the Edit and List Profile utilities, the system administrator can maintain a user's subsystem record(s) as well as registration record. First, however, the system programmer must build a subsystem module as an interface between the utilities and subsystem records. Specifications for subsystem modules are given in Appendix F.

User registration macro calls allow the user to:

- Create a skeletal subsystem record that contains user id, time of creation, and subsystem id
- Read a subsystem record
- Read limited information from a registration record
- Update a subsystem record
- Request and verify a password from the user of a terminal that has experienced a phsyical disconnection.

User registration macro calls are:

| | |
|---|---|
| • Profile Record, Accounting Update | $PRFAU |
| • Profile Record, Create | $PRFCR |
| • Profile Record, Delete | $PRFDL |
| • Profile Record, Get | $PRFGT |
| • Profile Record, Get User Information | $PRFIF |
| • Profile Record, Update | $PRFUP |
| • Reverify Password | $RVFPW |

# Section 3
# FILE SYSTEM FUNCTIONS

File system macro calls enable applications to access data files, including device files. These functions fall into the following categories:

- File management
- Data management
- Storage management.

This section describes each category and its use of the File Information Block (FIB). All of the functions mentioned below are described in detail in Volume II of this manual.

## FILE MANAGEMENT FUNCTIONS

The macro calls for file management consist of the following functions: *

| | |
|---|---|
| Change Working Directory | $CWDIR |
| Close File | $CLFIL |
| Create Directory | $CRDIR |
| Create File | $CRFIL |
| Delete File | $DLFIL |
| Delete Directory | $DLDIR |
| Expand Pathname | $XPATH |
| Get Device Information | $GIDEV |

*

| | |
|---|---|
| Get File | $GTFIL |
| Get File Access Rights | $GAFIL |
| Get File Accounting Information | $GTACT |
| Get File Information | $GIFIL |
| Get Working Directory | $GWDIR |
| Grow File | $GRFIL |
| Link File | $LKFIL |
| Link Name | $LKNME |
| Unlink File | $ULFIL |
| Unlink Name | $ULNME |
| Open File | $OPFIL |
| Remove File | $RMFIL |
| Rename File/Directory | $RNFIL |
| Modify File | $MDFIL |
| Set Terminal File Characteristics | $STFIL |
| Set Task Identifier | $STTID |
| Reset Task Identifier | $RSTID |
| Test File For Input | $TIFIL |
| Test File For Output | $TOFIL |
| Shrink File | $SHFIL |
| Swap File | $SWFIL |
| Wait For File Input | $WIFIL |
| Wait For File Output | $WOFIL |
| Cleanpoint | $CLPNT |
| Rollback | $ROLBK. |

The macro calls listed above are preparatory to processing a file. Specifically, file management macro calls allow the user to perform the following:

- Create a file
- Delete a file
- Get a file (reserve a file for processing)
- Open a file
- Close a file
- Remove a file from processing
- Rename a file
- Modify a file's attributes

*

- Create a directory
- Delete a directory
- Rename a directory

- Change the working directory
- Get the name of the current working directory
- Expand disk space allocated to a file
- Contract disk space allocated to a file
- Expand pathname (develop a full pathname from a relative pathname)
- Get information about a file
- Test the status of an I/O activity (terminal)
- Wait for the completion of an asynchronous I/O activity (terminal)
- Set the file characteristics of a terminal
- Return (recover) a file to its last consistent state after a system or software failure
- Establish tasks, rather than groups, as independent users of file recovery and record locking services
- Re-establish groups as independent users of file recovery and record locking services
- Swap to the next section of a multivolume tape file or disk serial multivolume file.
- Link a file or directory to a new pathname

Although the following functions are available through macro calls, they are typically performed outside of program execution by means of execution control (ECL) commands:

- Get File
- Remove File
- Create File
- Delete File
- Grow File
- Shrink File
- Rename File
- Modify File
- Create Directory
- Delete Directory
- Change Working Directory
- Get Working Directory
- Set Terminal File Characteristics
- Associate File
- Dissociate File.

## DATA MANAGEMENT FUNCTIONS

The following macro calls are considered data management functions:

| | |
|---|---|
| Delete Record | $DLREC |
| Read Record | $RDREC |
| Rewrite Record | $RWREC |
| Write Record | $WRREC. |

The above macro calls provide for the transfer of logical records between the user's record storage area and external files. Before any data management calls can be executed, the file to be accessed must have been reserved (by means of the Get File or Create File functions) and opened (by means of the Open File function). Moreover, before a file can be opened, it must have been associated with a logical file number (LFN) by means of a Get File or Create File function. Thus, data management and file management macro calls are interdependent. Figure 3-1 partially illustrates this interdependence.

## STORAGE MANAGEMENT FUNCTIONS

The following macro calls perform storage management functions:

| | |
|---|---|
| Read block | $RDBLK |
| Wait block | $WTBLK |
| Write block | $WRBLK. |

These calls transfer physical blocks of data between the user's buffer and an external file. Storage management itself is used transparently by data management to perform input/output. An initial Read Record ($RDREC) call, for example, causes storage management to transfer a block of data from external storage to a buffer in memory. Data management then unblocks a record and transfers it to a second buffer within the application.

By means of storage management read and write functions, the user can transfer blocks of data directly to or from an application buffer, bypassing an intermediate buffer and the blocking/deblocking operations performed by data management. Although highly efficient, storage management places on the user responsibility for observing various file organizations and formats while blocking/deblocking. The user of storage management must also provide any necessary control information, such as control interval headers and logical record headers.

By creating two application buffers and by using the Wait Block macro call (described in Volume II) the user can perform asynchronous I/O (i.e., process one block of data while another is being transferred from device to memory).

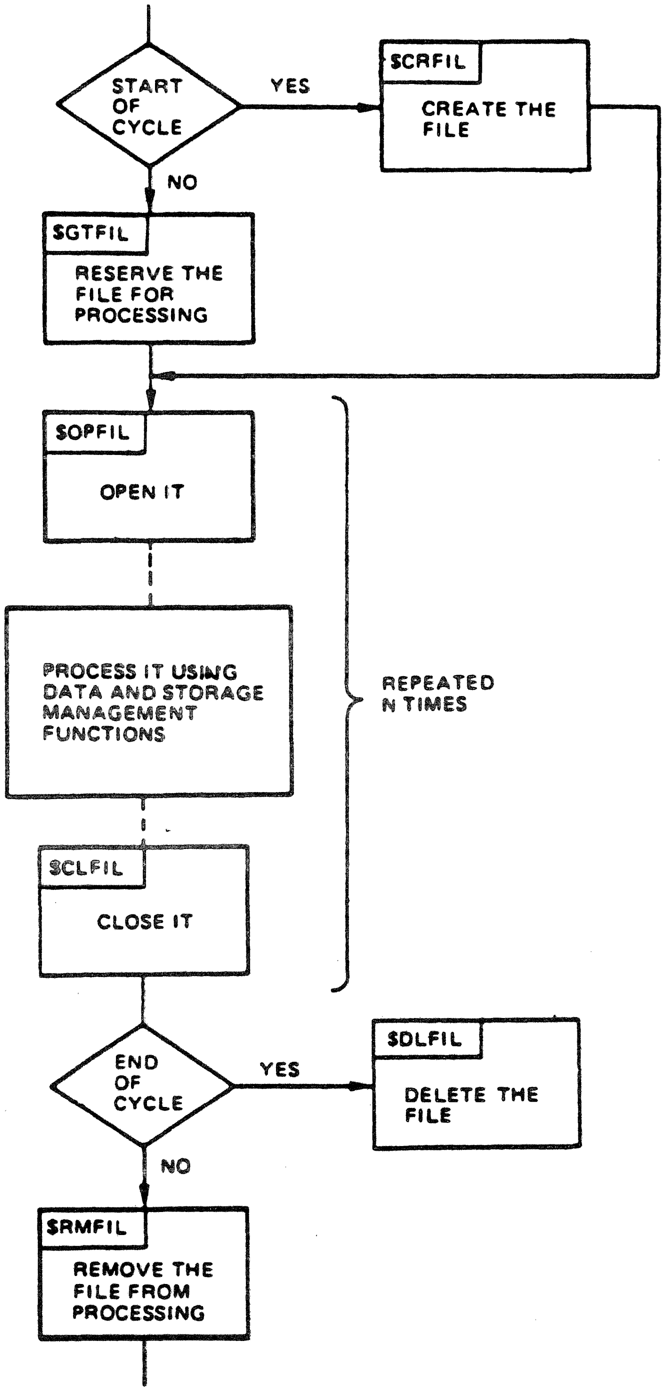


Figure 3-1. Life Cycle of a File

Like data management macro calls, storage management macro calls cannot be executed until the file to be accessed has been reserved, opened, and associated with an LFN.

## FILE INFORMATION BLOCK

Data management, storage management, and several file management functions must pass arguments to the file system by means of a data structure called the File Information Block (FIB). The arguments passed include the LFN of the file to be accessed, the address of the user's record area, the size of input and output records, and the type of key by which records are to be located.

The following macro calls must use an FIB:

| | |
|---|---|
| Open File | $OPFIL |
| Close File | $CLFIL |
| Swap File | $SWFIL |
| Test File | $TIFIL, $TOFIL |
| Read Record | $RDREC |
| Write Record | $WRREC |
| Rewrite Record | $RWREC |
| Delete Record | $DLREC |
| Read Block | $RDBLK |
| Write Block | $WRBLK |
| Wait Block | $WTBLK |

Some of the arguments required for one type of macro call (e.g., storage management) are not applicable to the other types. Thus, a FIB generated for data/file management functions differs in format from a FIB generated for storage management functions.

The user can generate a FIB and values for its entries by means of the $FIB macro call. Depending on the argument(s) supplied with it, $FIB does one of the following:

- Generates an FIB, with default values, for data/file management
- Generates an FIB for data/file or storage management, with values defined by the user
- Modifies values of an existing FIB.

Using $FIB, the user can set values for a new or existing FIB by means of keywords that specify a field and expressions that specify a value. The $FIB argument "IRL=90", for example, refers to the input record length field of a data/file managment FIB and sets a maximum input record length of 90 bytes. Other keywords are specific to storage management functions.

To modify the fields of an existing FIB, you can employ offset tags rather than $FIB keywords. (Offset tags are discussed later in this section and in Section 5). $FIBDM generates tags specific to data/file management functions; $FIBSM generates tags specific to storage/management functions. $TFIB generates two sets of tags applicable to both kinds of file system functions.

## File Information Block (FIB) for Data Management

Table 3-1 describes the entries of a FIB used with data/file management macro calls. The offset tags for these entries, generated by $FIBDM, are shown in Appendix C.

Table 3-1. File Information Block (FIB) for Data Management

| Entry | Size (bytes) | Description |
|---|---|---|
| Logical file number (LFN) | 2 | Specifies the logical file number (LFN) by which the file is refered to. The LFN is the common element linking the FIB and the external file; this connection is made via the $CRFIL or $GTFIL macro call (or equivalent command). |
| Program view | 2 | Describes user visibility to the file, and the file's functional capabilities. Bit 0 set to 0 indicates that this FIB is to be used for data management (record level) access. Table 3-2 describes this entry in detail and its bit settings for data management calls. |
| User record pointer | 4 | Identifies the start of the user-record area as follows:<br><br>$RDREC - Identifies the storage area into which records are delivered by the system.<br><br>$RWREC, $WRREC - Identifies the storage area from which records are taken by the system.<br><br>The storage area must be large enough to contain the longest record, excluding headers, to be written to or received from the file. |

Table 3-1 (cont). File Information Block (FIB) for Data Management

| Entry | Size (bytes) | Description |
|---|---|---|
| In record length | 2 | Specifies the maximum size (in bytes) of the user-record area for $RDREC operations. |
| Out record length | 2 | Specifies the actual size (in bytes) of the record to be written or read, as follows:<br><br>$RDREC - The system updates this entry to reflect the actual length (in bytes) of the last record delivered into the user-record area.<br><br>$RWREC, $WRREC - Specifies the actual length (in bytes) of the record, excluding the headers, to be written in the file. |
| In record status | 1 | On write operations, indicates the type of terminal control information in each record as follows:<br><br>0000 = unknown terminal control information<br><br>0001 = no terminal control information<br><br>0010 = standard GCOS 6 printer control characters |
| Out record status | 1 | On read-record operations bit 0 = 1 indicates that the record just read is a duplicate of a previous record (i.e., it contains the same key value as the previous record). On write-record or rewrite-record operations bit 0 = 1 indicates that the record just written is a duplicate (i.e., it contains the same key value as a record already in the file).<br><br>On read-record operations bit 1 = 1 indicates that there are more duplicates for this record still remaining in the file. |

Table 3-1 (cont). File Information Block (FIB) for Data Management

| Entry | Size (bytes) | Description |
|---|---|---|
| Out record status (cont) | | For example, if three records exist with the same key value, then reading the first one will return in this entry:<br><br>bit 0 = 0<br>bit 1 = 1;<br><br>reading the second record will return:<br><br>bit 0 = 1<br>bit 1 = 1;<br><br>reading the last record will return:<br><br>bit 0 = 1<br>bit 1 = 0 |
| In record type | 2 | $RDREC - Specifies the record type of the record to be read. 'FFFF' indicates that any record type is acceptable. |
| Out record type | 2 | $WRREC, $RWREC, $DLREC - Specifies the record type of the record to be updated.<br><br>$RDREC - Specifies the record type of the record delivered to the user. |
| In key pointer | 4 | Identifies the start of the user-key area in which the key value is stored for the following $RDREC macro call functions:<br><br>Read with key<br>Read position equal<br>Read position greater than<br>Read position greater than or equal<br>Read position forward<br>Read position backward<br><br>For the following $WRREC macro call functions:<br><br>Write with key<br>Write position equal<br>Write position greater than<br>Write position greater than or equal |

Table 3-1 (cont). File Information Block (FIB) for Data Management

| Entry | Size (bytes) | Description |
|---|---|---|
| In key pointer (cont) | | Write position forward<br>Write position backward<br><br>For the following $RWREC macro call function:<br><br>Rewrite with key<br><br>And for the following $DLREC macro call function:<br><br>Delete with key<br><br>For CALC, Primary, and Alternate keys, the keys to be used must be initialized within the user's record area and the field must point to that key.<br><br>The type of key is specified in the "in key format" entry below. |
| In key format | 1 | Identifies the type of key pointed to by the "in key pointer" entry above, as follows:<br><br>0 - None specified; the type of key is determined by the format of the file.<br><br>1 - Primary, Relative, or CALC (Random), as determined by the file format:<br><br>● Primary key for indexed files<br><br>● Relative key for relative files<br><br>● CALC key for random files<br><br>2 - Simple key<br><br>3 - Alternate key<br><br>-1 - Current key of reference<br><br>The entry is meaningful only for the macro calls specified in the "in key pointer" entry defined above. |

Table 3-1 (cont). File Information Block (FIB) for Data Management

| Entry | Size (bytes) | Description |
|---|---|---|
| In key length | 1 | Specifies the length (in bytes) of the user-key area identified in the "in key pointer" entry described above. Only meaningful for primary, alternate, and CALC keys; simple and relative keys are always assumed to be four bytes. |
| Out record address | 4 | This field is available for the system to place the media address of the last record transferred by the last data management macro call.<br><br>Normally, this address is a 32-bit simple key (i.e., it specifies the control interval and logical record number within the control interval). However, if the file is accessed via a relative key as specified in the "in key format" field, then this address is a 32-bit relative key (i.e., relative logical record number in the file).<br><br>This field is undefined if the operation is not performed as expected.<br><br>For card readers, printers, and terminal devices, this field contains a count of the records transferred; i.e., this field is incremented by 1 for each access to the device. |
| Reserved | 4 | This entry is reserved for future use; must be set to zeros. |

## Program View Entry in FIB for Data Management

Table 3-2 shows the contents of the 2-byte program view entry for data management (record level) access. The program view entry describes to the file system how the file is to be accessed, and, to some extent, what it looks like to the programmer. The file system uses the FIB's contents to ensure that the file is accessed only as intended. Keywords of the $FIB macro call and offset tags generated by $FIBDM both provide a means of refering to fields within the program view entry.

Bits 0 through 9 of the program view entry are processed only when the file is opened, and cannot be changed while the file is open.

Table 3-2. Program View Entry in FIB for Data Management

| Entry | Size (bits) | Description | Related Macro Calls |
|---|---|---|---|
| Access level (Bit 0) | 1 | Specifies that file is accessed via data management macro calls, as follows:<br><br>0 - Access via data management macro calls. | $OPFIL |
| Process rules (Bits 1-4) | 4 | Specifies how the file can be processed; that is, it specifies which types of data management macro calls are allowed as follows:<br><br>Binary / Permitted Macro Calls<br>1000 $RDREC<br>0100 $WRREC<br>0010 $RWREC<br>0001 $DLREC<br><br>nnnn Any combination of the settings to allow the desired data management macro calls listed above.<br><br>A macro call that is not permitted (as specified in this field) causes an access violation error. | |
| Key type (Bits 5-9) | 5 | Specifies the type of keys that can be used to access the file as follows:<br><br>Binary / Permitted Key Type<br>10000 Primary | $OPFIL |

Table 3-2 (cont). Program View Entry in FIB for Data Management

| Entry | Size (bits) | Description | Related Macro Calls |
|---|---|---|---|
| Key type (Bits 5-9) (cont) | | 01000 CALC (Random)<br><br>00100 Alternate<br><br>00010 Relative<br><br>00001 Simple<br><br>00101 Alternate and Simple<br><br>10101 Alternate and Simple plus Primary<br><br>01101 Alternate and Simple plus CALC<br><br>00111 Alternate and Simple plus Relative<br><br>If the key type specified in this field is not permitted by the type of file being processed, a bad program view error results. The following types of keys are allowed by the specified types of files:<br><br>File Organization — Key Type<br><br>UFAS Indexed, Alternate — Primary<br><br>UFAS Random — CALC<br><br>UFAS Disk Resident Files — Alternate<br><br>UFAS Relative, Fixed Relative — Relative<br><br>UFAS Disk Resident Files — Simple | |

Table 3-2 (cont). Program View Entry in FIB for Data Management

| Entry | Size (Bits) | Description | Related Macro Calls |
|---|---|---|---|
| Record class (bit 10) | 1 | Specifies type of logical records that can be present in the file as follows:<br><br>0 - Any type (i.e., fixed- or variable-length records allowed).<br><br>1 - Only fixed-length records allowed. | $RDREC<br>$WRREC<br>$RWREC |
| Record visibility (Bit 11) | 1 | Specifies whether or not deleted records are skipped during read next record ($RDREC) operations as follows:<br><br>0 - Deleted records not visible (i.e., skip them)<br><br>1 - Deleted records are visible (i.e., the system issues the record not found return code when a deleted record is accessed). | $RDREC |
| Key storage area alignment (Bit 12) | 1 | Specifies the boundary alignment of the user-key area (see "in key pointer" entry in Table 3-1) as follows:<br><br>0 - Key storage area begins at even-byte boundary (word-aligned).<br><br>1 - Key storage area begins at odd-byte boundary. | $RDREC<br>$WRREC<br>$RWREC<br>$DLREC |

Table 3-2 (cont). Program View Entry in FIB for Data Management

| Entry | Size (Bits) | Description | Related Macro Calls |
|---|---|---|---|
| Record storage area alignment (Bit 13) | 1 | Specifies the boundary alignment of the user-record area (see "User Record Pointer" entry in Table 3-1) as follows:<br><br>0 - Record storage area begins at even-byte boundary (word-aligned).<br><br>1 - Record storage area begins at odd-byte boundary. | |
| Transcription mode (Bit 14) | 1 | Specifies how data is to be transferred as follows:<br><br>0 - Data is transferred in device-specific native (ASCII) mode.<br><br>1 - Data is transferred in binary transcription mode. (See Note 2.) | $RDREC<br>$WRREC |
| Reserved (Bit 15) | 1 | Reserved; must be zero. | None |

NOTES

1. Bits 10 through 15 may be set after an $OPFIL macro call and before any data management macro call.

2. Binary transcription mode is meaningful only for card devices, seven-track tapes, and EBCDIC tapes. For card devices, this mode is equivalent to verbatim mode (see Section 6).

## File Information Block (FIB) for Storage Management Access

Table 3-3 describes the entries of a FIB used with storage management macro calls. The offset tags for these entries, generated by $FIBSM, are shown in Appendix C.

## Program View Entry in FIB for Storage Management

Table 3-4 shows the contents of the 2-byte program view entry for storage management (block level) access. The program view entry describes to the file system how the file is to be accessed, and to some extent, what it looks like to the programmer. The file system uses the FIB's contents to ensure that the file is accessed only as intended. Keywords of the $FIB macro call and offset tags generated by $FIBSM both provide a means of referring to fields within a program view entry.

Bits 0 through 9 of the program view entry are processed only when the file is opened, and cannot be changed while the file is open.

## Offsets Definitions

You can refer to specific locations in the file information block and other argument structures by using offsets definition macro calls. These calls, summarized in Section 5 and described in detail in Volume II of this manual, define offsets tags.

Table 3-5 shows the offsets definition macro calls and the structures for which they define tags.

Offsets definition macro calls can be specified only once per assembly procedure. They provide tags that are equated to specific offsets in argument structures and FIBs. For example, assuming that the address of an argument structure labeled FILE_A has been loaded into a base register as follows:

```
        LAB  $B4,FILE_A
```

and assuming that $CRPSB has been specified, the following address syllable can be used to refer to the argument structure entry that identifies the control interval size:

```
        $B4.R_CISZ
```

This entry effectively points to the displacement FILE_A+5 in the parameter structure.

Volume II of this manual describes each displacement definition macro routine/call and its tags, displacements, and entry names in detail.

Table 3-3. File Information Block (FIB) for Storage Management

| Entry | Size (bytes) | Description |
|---|---|---|
| Logical file number (LFN) | 2 | Specifies the logical file number with which the file is referenced. The LFN is the common element linking the FIB with the external file; this connection is made with the $CRFIL or $GTFIL macro call, or equivalent command. |
| Program view | 2 | Describes the user visibility to the file and the file's functional capabilities. Bit 0 set to 1 indicates that this FIB is to be used for storage management (block level) access. Table 3-4 describes this entry in detail, and its bit settings for storage management macro calls. |
| Buffer pointer | 4 | Identifies the start of the buffer area as follows:<br><br>$RDBLK - Identifies the buffer area into which blocks of data are delivered.<br><br>$WRBLK - Identifies the buffer area from which blocks of data are taken. |
| Transfer-size | 2 | Specifies the size (in bytes) of the data transfer (i.e., the size of the buffer). |
| Block size | 2 | Specifies the size of the block (in bytes). For disk files the size must be a multiple of physical sector size. |
| Block number | 4 | Specifies the starting block number for the I/O transfers; is relative to the start of the file and to the block size (described above). This entry is incremented by 1 after each I/O transfer; therefore, a user's dynamic changes to the block size also require changes to the contents of this entry. The first block in a file is block 0. |
| Reserved | 16 | Reserved for later use; must be set to zeros. |

Table 3-4. Program View Entry in FIB for Storage Management

| Entry | Size (bits) | Description | Related Macro Calls |
|---|---|---|---|
| Access level (Bit 0) | 1 | Specifies that file is accessed via storage management macro calls, as follows:<br><br>1 - Access via storage management macro calls. | $OPFIL |
| Process rules (Bits 1-4) | 4 | Specifies how the file can be processed; that is, it specifies which types of storage management macro calls are allowed as follows:<br><br>Binary — Permitted Macro Calls<br>1000 — $RDBLK<br>0100 — $WRBLK<br>1100 — $RDBLK, $WRBLK<br><br>A macro call that is not permitted in this field causes an access violation error. | |
| Reserved (Bits 5-12) | 8 | Reserved; must be set to zeros. | |
| Buffer Alignment (Bit 13) | 1 | Specifies the boundary alignment of the user buffer (see "Buffer Pointer" in Table 3-3) as follows:<br><br>0 - Buffer begins at even-byte boundary (word aligned).<br><br>1 - Buffer begins at odd-byte boundary. | $RDBLK<br>$WRBLK |

Table 3-4 (cont). Program View Entry in FIB for Storage Management

| Entry | Size (Bits) | Description | Related Macro Calls |
|---|---|---|---|
| Transcription mode (Bit 14) | 1 | Specifies how data is transferred as follows:<br><br>0 - Data is transferred in device-specific native (ASCII) mode.<br><br>1 - Data is transferred in binary transcription mode. (See Note 2.) | $RDBLK<br>$WRBLK |
| Synchronous/ asynchronous indicator (Bit 15) | 1 | Specifies whether or not $RDBLK or $WRBLK macro calls are executed synchronously or asynchronously as follows:<br><br>0 - $RDBLK or $WRBLK macro calls are to be executed synchronously. When synchronous $RDBLK or $WRBLK macro calls are issued, a $WTBLK macro call is not required to synchronize buffer use.<br><br>1 - $RDBLK or $WRBLK macro calls are to be executed asynchronously (i.e., a $WTBLK macro call is required to synchronize.) | $RDBLK<br>$WRBLK |

NOTES

1. Bits 10 through 15 may be set after an $OPFIL macro call and before any Storage Management macro call.

2. Binary transcription mode is meaningful only for card devices, seven-track tapes, and EBCDIC tapes. For card devices, this mode is equivalent to verbatim mode (see Section 6).

Table 3-5. Offsets Definition Macro Calls

| Macro Call | Affected Structure |
|---|---|
| $CRPSB | Argument structure for Create File macro call ($CRFIL) |
| $CRRDB | Record descriptor block pointed to by the $CRPSB argument structure |
| $DIPSB | Argument structure for Get Device Information macro call |
| $GTPSB | Argument structure for Get File macro call ($GTFIL) |
| $GAPSB | Argument structure for Get File Access Rights macro call ($GAFIL) |
| $GIPSB | Argument structure for Get File Information macro call ($GIFIL) |
| $GRPSB | Argument structure for Grow File macro call ($GRFIL) |
| $GIFAB | File attribute block pointed to by the $GIPSB argument structure |
| $GNPSB | Argument structure for Get Name macro call ($GNFIL) |
| $SHPSB | Argument structure for Shrink File macro call ($SHFIL) |
| $TFIB | File information block for the following macro calls:<br><br>Open File $OPFIL<br>Close File $CLFIL<br>Test File $TIFIL, $TOFIL<br>Read Record $RDREC<br>Write Record $WRREC<br>Rewrite Record $RWREC<br>Delete Record $DLREC<br>Read Block $RDBLK<br>Write Block $WRBLK<br>Wait Block $WTBLK |

Table 3-5 (cont). Offsets Definition Macro Calls

| Macro Call | Affected Structure |
|---|---|
| $FIBDM | File information block specific to data management (record level) access; used for the following macro calls:<br><br>Open File $OPFIL<br>Close File $CLFIL<br>Test File $TIFIL, $TOFIL<br>Read Record $RDREC<br>Write Record $WRREC<br>Rewrite Record $RWREC<br>Delete Record $DLREC |
| $FIBSM | File information block specific to storage management (block level) access; used for the following macro calls:<br><br>Open File $OPFIL<br>Close File $CLFIL<br>Read Block $RDBLK<br>Write Block $WRBLK<br>Wait Block $WTBLK |
| $MDPSB | Argument structure for Modify File macro call ($MDFIL) |

# Section 4
# COMMUNICATIONS PROCESSING FUNCTIONS

Communications processing refers, in this section, to the transfer of data between an application program and a remote device (i.e., terminal or printer). A remote device is one connected to a Multi-Line Controller (MLC); a local device is attached instead to a Multiple Device Controller (MDC). The control of local devices by means of device drivers is discussed in Section 6.

## OVERVIEW OF COMMUNICATIONS PROCESSING

The user can control the transfer of data between an application program and a remote device either by means of the file system or, more directly, by physical input/output.

Using the file system, the programmer employs many of the file system functions described in Section 3 (e.g., Open File, Read Record, Write Record). The parameters for these operations are passed between the application program and the file system by means of the file information block (FIB), which is also described in Section 3. The system translates the values of FIB entries into values for the entries of the input/output request block (IORB). Thus marked, the IORB provides instructions to a line protocol handler (LPH), which carries out the desired input/output operation.

Using physical I/O, the programmer directly constructs and issues the IORB instead of doing so indirectly by means of file system functions and the FIB. To write output to a terminal, for example, the programmer performs the following:

1. Generates an IORB by means of the $IORB macro call

2. Generates IORB offsets tags (by means of the $IORBD macro call), which enable the programmer to refer to and fill fields in the IORB

3. Sets, in the appropriate IORB fields, a write function code and parameters specializing the write operation

4. Issues a Request Input/Output ($RQIO) macro call, which causes the appropriate LPH to perform the operation indicated by the IORB.

The above example assumes that the device being written to has already been connected by means of previously issued $IORB and $RQIO macro calls (as explained in the final subsection).

## COMMUNICATIONS PROCESSING THROUGH THE FILE SYSTEM

The following subjects are discussed below:

- File system functions applicable to the communications processing

- Synchronous and asynchronous I/O

- Use of specific file system functions

- Sequences of file system functions useful for communications processing

- Use of the Set Terminal Characteristics function/command for changing terminal characteristics.

### File System Functions

The file system functions applicable to communications processing fall under the headings of File Management and Data Management.

#### FILE MANAGEMENT FUNCTIONS

By means of these functions, a terminal can be reserved for processing, opened, closed, and associated with a logical file number (LFN) that identifies the file to the system. The macro calls that perform these and other related functions are:

| | |
|---|---|
| Get File | $GTFIL |
| Open File | $OPFIL |

*

| | |
|---|---|
| Close File | $CLFIL |
| Test File | $TIFIL/$TOFIL |
| Wait File | $WIFIL/$WOFIL |

## DATA MANAGEMENT FUNCTIONS

Data management functions enable an application to read and write logical records either synchronously or asynchronously. (Synchronous and asynchronous I/O operations are explained later in this section.) Data management functions are:

| | |
|---|---|
| Read Record | $RDREC |
| Write Record | $WRREC |

### Synchronous Input/Output

A terminal can be configured for either synchronous or asynchronous I/O operations. In synchronous operations, the processing of data and the transfer of data (between application and terminal) occur sequentially rather than simultaneously. Thus, the application must wait until the transfer of data is complete before processing can resume. Synchronous I/O is best suited to situations in which data is transferred between an application and a single terminal and to such activity as connecting and disconnecting a terminal.

### Asynchronous Input/Output

If a terminal is configured for asynchronous I/O, data is transferred between the terminal and the application by way of a system buffer. Thus, asynchronous I/O allows the application to process records while the file system reads or writes records to or from the buffer.

Asynchronous I/O and the data/file management functions listed above allow an application to access multiple interactive terminals efficiently. For terminals operating asynchronously, the system automatically schedules an anticpatory read, which transfers input entered at the terminal to a buffer in system memory. If an application immediately issues a Read Record ($RDREC) call, the task must wait until the system buffer has received input from the terminal. While the task is waiting, data may be available from another terminal reserved by the application. Instead, the application can issue the Test Input File ($TIFIL) macro call to determine whether a read has completed at a specific terminal. Alternatively, Wait File for Input ($WIFIL) can be used to wait until a read has completed at any of the reserved terminals. A subsequent Read Record to the terminal would then return the data for processing by the application. The Test File function also enables an application to test the completion of a physical connection to a terminal before issuing an order to that terminal.

## Using File System Functions

This subsection provides specific information on the use of the following data and file management functions:

| | |
|---|---|
| Get File | $GTFIL |
| Open File | $OPFIL |
| Test File | $TIFIL/$TOFIL |
| Wait File | $WIFIL/$WOFIL |

### GET FILE ($GTFIL) MACRO CALL GUIDELINES

The Get File function reserves a file for processing and connects a file to a logical file number (LFN). The LFN is used in other file system calls (e.g., $OPFIL, $RDREC, $WRREC) to refer to the file in question. Normally, the Get File function is invoked by a Get File command outside program execution.

The arguments for the Get File ($GTFIL) macro call in an Assembly language communications program must have the values shown in Table 4-1.

### OPEN FILE ($OPFIL) MACRO CALL GUIDELINES

The Open File function allocates buffer space (if required) and physically connects the device or terminal.

The Open File macro call $OPFIL, when used in communications, must include the location of the file information block (FIB), which in turn must contain a valid program view item.

### TEST FILE ($TIFIL, $TOFIL) MACRO CALL GUIDELINES

Before the application issues a $RDREC macro call, it can issue the Test Input File ($TIFIL) macro call to check whether input is available.

Table 4-1. Arguments for Get File ($GTFIL) Macro Call

| Argument | Argument Value |
|---|---|
| Logical file number (LFN) | A value from 0 through 4095 |
| Pathname pointer | Must point to a pathname of a communications device (e.g., !TTY01) |
| Concurrency control | According to how the application uses the device (normally zero for exclusive use) |
| Remaining arguments | Zero |

Before the application issues a $WRREC macro call, it can issue the Test Output File ($TOFIL) macro call to check whether the preceding output operation was completed.

WAIT FILE ($WIFIL, $WOFIL) MACRO CALL GUIDELINES

The use of the Wait File macro call permits an application to wait for the completion of an outstanding read or write order. The Wait File macro call can be used with a set of terminals or devices. Test and Wait File macro calls differ in terms of when control is returned to the calling routine. A Test File call will return immediately with a busy or not busy status. An application would block the execution of lower level tasks with repeated test file calls to a busy file. This problem can be avoided by issuing a Wait File macro call in lieu of successive Test File macro calls.

$WIFIL is used to wait for input from any device/terminal; $WOFIL to wait for completion of output to any device/terminal.

## Macro Call Sequences

This subsection describes sequences of file system macro calls commonly used by applications that access communications devices. Each sequence of macro calls applies to a different type of communications processing.

The types of communications processing illustrated below are:

- Input only (TTY or STD data entry applications)
- Output only (receive-only printer (ROP) application)
- Bidirectional (the device is opened either for input or output, but not both (BSC 2780))
- Interactive (TTY, STD, BTF, or BSC 3780 applications).

MACRO CALL PROCEDURES FOR DATA ENTRY TERMINALS

Table 4-2 shows the procedure for using file system macro calls in a communications application involving data entry terminals.

MACRO CALL PROCEDURES FOR OUTPUT-ONLY TERMINALS

Table 4-3 shows the procedure for using macro calls in communications applications involving output-only terminals.

## Macro Calls for a Single Interactive Terminal

Table 4-4 describes the procedures for using macro calls in communications applications involving only one interactive terminal that has been configured for non-buffered synchronous input/output operation.

Table 4-2. Macro Call Procedures for Data Entry Terminals

| Procedure Step | Action by Application Program | System Actions |
|---|---|---|
| 1 | Issue $GTFIL macro call. | |
| 2 | Issue $OPFIL macro call with FIB program view bit 1 set to 1, bit 2 set to 0. | Issues asynchronous connect; returns a normal status to the program. |
| 3 | Issue $WIFIL macro call to wait until connect is complete and input is available. (With multiple devices, the $WIFIL macro call can be issued with a list of LFNs, effectively giving up control until input is available from one or more devices in the list.)<br><br>Otherwise, if application is to do other processing (not giving up control), issue $TIFIL macro call. | Returns when a read has been satisfied.<br><br>If connect is not complete, returns a busy status. If connect is complete, issues an asynchronous read and returns a busy status until read is complete. |
| 4 | If not-busy status is returned, issue $RDREC macro call. | With read operation complete, moves data from system buffer to application's buffer, issues another asynchronous read, and returns a normal status to the program. |
| 5 | If an error status is returned, exit from the procedure. | |
| 6 | When read is successful, return to step 3 to request more data from the device. | |
| 7 | When application processing is completed, issues $CLFIL macro call. | Issues a disconnect. |
| 8 | Issue a $RMFIL macro call. | |

Table 4-3. Macro Call Procedures for Output-Only Terminals

| Procedure Step | Action by Application Program | System Actions |
|---|---|---|
| 1 | Issue $GTFIL macro call. | |
| 2 | Issue $OPFIL macro call with FIB program view bit 1 set to 0, bit 2 set to 1. | Issues an asynchronous connect, returns a normal status to the program. |
| 3 | Issue $WOFIL macro call to wait until connect is complete and output can be transmitted. (With multiple devices, the $WOFIL macro call can be issued with a list of LFNs, effectively giving up control until output can be sent to one or more of the devices in the list.) | Will return when output can be transmitted. |
| | Otherwise, if the application is to do other processing (not give up control), issue a $TOFIL macro call. | If connect is not complete, returns a busy status. If connect is complete, returns a not busy status if output can be transmitted. |
| 4 | If not-busy status is returned, issue $WRREC macro call. | Moves data from application buffer to system buffer. Issues asynchronous write and returns a normal status to the application. |
| 5 | If error status is returned, exit from the procedure. | |
| 6 | When write is successful, return to step 3 to transmit more data to the device. | |
| 7 | When application processing is complete, issue $CLFIL macro call. | Issues disconnect according to device type. |
| 8 | Issue $RMFIL macro call. | |

Table 4-4. Macro Call Procedures for Single Interactive Terminal

| Procedure Step | Action by Application Program | System Actions |
|---|---|---|
| 1 | Issue $GTFIL macro call. | |
| 2 | Issue $OPFIL macro call with FIB program view bit 1 set to 1, program view bit 2 set to 1. | |
| To read from the terminal and then write to the terminal: | | |
| 3 | Issue $RDREC macro call. (This effectively gives up control until the read is satisfied.)<br><br>If error status returned, exit from the procedure. | Data is read directly into the application buffer. |
| 4 | Process the data just read. | |
| 5 | Issue $WRREC. (This effectively gives up control until the write is complete.) If an error status is returned, exit from the procedure. | Data is written directly from the application buffer. |
| 6 | If additional input is expected, refer to step 3. | |
| 7 | When application processing is complete, issue $CLFIL macro call. | Issues a disconnect. |
| 8 | Issue $RMFIL macro call. | |

## MACRO CALL PROCEDURES FOR MULTIPLE INTERACTIVE TERMINALS

Table 4-5 describes the procedures for using macro calls in communications applications involving multiple terminals configured for buffered, asynchronous operation.

Figure 4-1 illustrates the procedure's flow.

Table 4-5. Macro Call Procedures for Multiple Terminals

| Procedure Step | Action by Application Program | System Actions |
|---|---|---|
| 1 | Issue $GTFIL macro call to each terminal. | |
| 2 | Issue $OPFIL macro call to each terminal with FIB program view bit 1 set to 1, bit 2 set to 1. | Issues asynchronous connect; returns normal status to the program. |
| To read from a terminal and then write to a terminal: | | |
| 3 | Issue $WIFIL macro call with a list of LFNs. (This will effectively give up control until input is available from one or more terminals in the list.) | Returns when a read is complete and data is available. Returns the LFN of the first terminal in the list for which data is available. |
| 4 | Issue $RDREC macro call. | Moves data from system buffer to application's buffer, issues another asynchronous read, and returns a normal status to the program. |
| 5 | If an error status is returned, exit from the procedure. | |
| 6 | Process the data just read. | |
| 7 | Issue $WRREC macro call. (This will give up control until output can be sent to terminal.) | Waits until output can be sent, moves data from the application's buffer to system buffer, and issues an asynchronous write. |
| 8 | If additional input is expected from any terminal, see step 3. | |
| 9 | When application processing is complete, issue $CLFIL call. | Issues disconnect. |

Table 4-5 (cont). Macro Call Procedures for Multiple Terminals

| Procedure Step | Action by Application Program | System Actions |
|---|---|---|
| 10 | Issue $RMFIL macro call. | |

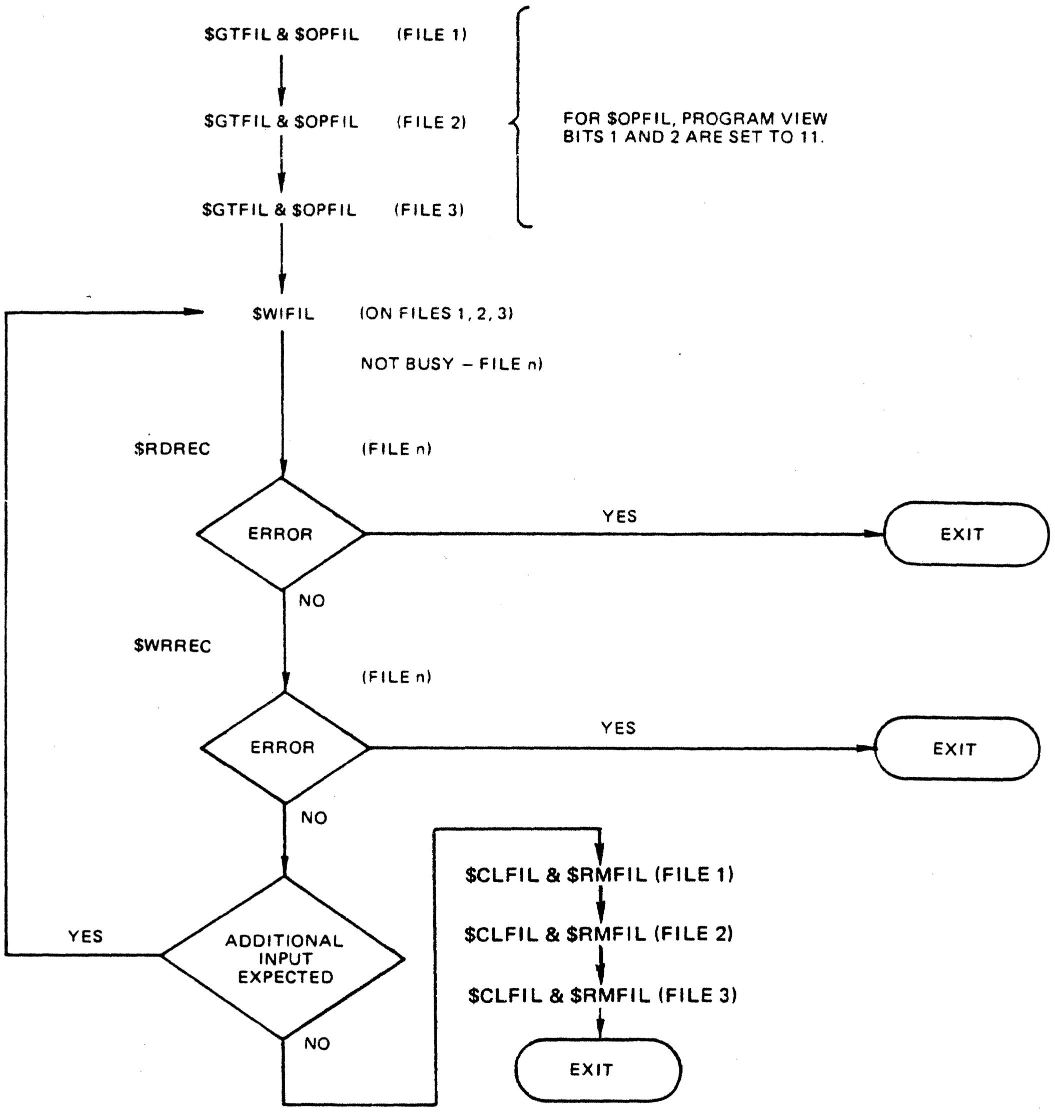


Figure 4-1. Simplified Program Logic for Multiple Interactive Terminals

## Changing A Terminal File's Characteristics

The file characteristics (e.g., line length or record size, detabbing, device type, operational mode) of a terminal are established at the time of system configuration. These characteristics can be changed by the file system user at execution time, before the file associated with the device is opened, through use of the Set Terminal Characteristics command (STTY) or macro call ($STFIL).

Of particular interest to the communications user are the STTY arguments that control the operational modes of a device. Examples of operational modes include echoplex, use of control bytes, and optional end-of-message processing. The user can specify operational modes by specifying a -MODES argument or by setting bits of a device specific word.

### SPECIFICATION BY -MODES ARGUMENT

The file system user can most conveniently specify operational modes by means of the -MODES arguments of the STTY command. For example, to specify the terminal's echoplex feature, the user enters -MODES ECHO. Conversely, the user enters -MODES ^ECHO to suppress the echoplex feature. To reset all operational modes to those designated at the time of configuration, the user invokes the control argument -RESET.

### SPECIFICATION BY DSW BIT SETTINGS

In some instances, the file system user may be required to specify the operational modes of a device by setting bits in the device-specific word (DSW) I_DVS in the IORB. This requirement occurs when the user wishes to alter an operational mode for which a -MODES argument has not been defined.

Specification by DSW bit settings is accomplished through the DSW1 and DSW2 arguments of the STTY command or $STFIL macro call. The DSW1 argument is used to change the I_DVS field in connect and disconnect IORBs that the file manager issues against a com- munications device; DSW2 is used to change the I_DVS field in the read and write IORBs that the file manager issues against the same device. A user, for example, can specify BSC 2780/3780 control byte processing by setting bit 4 in DSW1 to zero.

To change a terminal's operating characteristics through the bit settings of the DSW, proceed as follows.

1. Determine which line protocol handler is servicing the terminal to be modified. One source for this information is the system's Configuration Load Manager (CLM) file (usually >SID>CLM_USER). In this file, a DEVICE directive names each device supported by the file system; each DEVICE directive in the file is paired with a station-defining directive that specifies the LPH serving the device.

2. Ascertain the operational characteristics established for the device at the time of configuration. The operational characteristics of a device are determined by the device-specific words of an IORB. The bit values of the device-specific words are set by the system; these default values are shown in Table 4-6 below. The user should consult the appropriate sections in this manual for the significance of particular bits in device-specific words. The sections that should be referenced are as follows:

| Device_Unit (LPH) | Section |
|---|---|
| Asynchronous Terminal Driver (ATD) | 8 |
| Synchronous Terminal Driver (STD) | 9 |
| Polled VIP Emulator (PVE) | 10 |
| BSC Line Protocol Handler (BSC) | 11 |
| TTY Line Protocol Handler (TTY) | 12 |
| BSC3270 Terminal Facility (BTF) | 21 |

The system-defined default values for device-specific words can be changed at the time of configuration by means of the STTY directive.

3. To change temporarily a DSW value that is in effect, enter a new value by means of the STTY command or $STFIL function. The new value will remain in effect only during the current session. To permanently change the operating characteristics of a device, use the STTY directive (described in the System Building and Administration manual).

Table 4-6. System Defaults for DSW1 and DSW2

| Device_Unit | DSW1 | DSW2 |
|---|---|---|
| TTY | 0000 | 0030 |
| BSC | 0000 | 0000 |
| PVE | 0000 | 0000 |
| XBSC | 0040 | 0000 |
| ATD | 0000 | 0030 |
| STD | 0103 | 0010 |
| BTF | 0000 | 0000 |

## COMMUNICATIONS PROCESSING THROUGH PHYSICAL I/O

The physical input/output (I/O) interface permits direct control by the user over communications processing. Used only with Assembly language programs, the physical I/O interface enables communications applications to:

- Call appropriate line protocol handlers (LPHs) directly through the communications subsystem rather than through the file system.
- Control the data structure, specifically the input/output request block (IORB), that directly affects device operations and/or characteristics.

### Physical I/O

The following conventions apply to use of physical I/O:

- Before requesting I/O transfers, an application must reserve a line or device through a $GTFIL monitor call or a GET command. Otherwise, all physical I/O requests will be rejected with an error code of 085A
- The I/O request block (IORB) is the standard control structure used by an LPH.
- An application program requests an I/O transfer by issuing a Request I/O ($RQIO) macro call.
- At the time of the $RQIO macro call, the B4 register contains the address of the IORB supplied by the application program.
- When configured, all LPHs and associated devices are identified by a set of unique LRNs at the time of system building. A line protocol handler is invoked when its LRN is included in the IORB for a subsequent $RQIO macro call.
- Bit F of IORB field I_CT1 must be set to 1; this is required for any I/O request.
- Before giving up control, the LPH maps the hardware return status into the status word I_ST of the application's IORB.

Table 4-7 lists the status codes that are returned (in the left byte of I_CT1) to indicate the result of an I/O request.

Table 4-7. I/O Request Status Codes Returned in I_CT1

| Code Number (Hexadecimal) | Meaning |
|---|---|
| 0 | No error, operation complete |
| 1 | Request block already busy (T=1) |
| 2 | Invalid LRN |
| 3 | Illegal wait |
| 4 | Invalid field values in the IORB |
| 5 | Device not ready |
| 6 | Device timeout on other than connect |
| 7 | Hardware error |
| 8 | Device disabled |
| 9 | File mark encountered |
| A | Controller unavailable |
| B | Device unavailable |
| C | Inconsistent request |
| F | EOT received (for BSC3780 and ATD stream mode) |
| 10 | Device timeout on connect |
| 34 | Requested ATD mode not configured |
| 35 | Requested ATD mode not configured for this controller |

NOTES

1. The 08 (device disabled) status is returned on an I/O request when the application has disabled the logical resource. It is also returned if a connect or disconnect has been issued against a line or device that is currently being connected (by a prior connect order) or disconnected (by a prior disconnect order).

2. The 0B (device unavailable) status is returned with every read or write IORB that has been aborted by a disconnect request with queue abort. This status can also indicate the loss (drop) of a communication line.

Table 4-7 (cont). I/O Request Status Codes Returned in I_CT1

| Code Number (Hexadecimal) | Meaning |
|---|---|
| | 3. When the 07 (hardware error) status is found in I_CT1 or in $R1 on a resume after wait, look at the IORB field I_ST to identify the specific error. |
| | 4. The 0C (inconsistent request) status indicates illogical I/O requests: read or write before connect, duplicate connect or disconnect requests, write after disconnect. |

## Using Physical I/O

Two fields within the IORB specify the operation to be performed.

1. The function code (Table 4-10), indicated by bits C through F of I_CT2 in the IORB (Table 4-8), specifies the particular operation.

2. The I_DVS item in the IORB, used with the function code, specializes the input/output order.

To request execution of an I/O operation, the application, with the $RQIO macro call, must transfer control to the physical I/O interface. At the time of the request, the B4 register must contain the address of the IORB being requested. The $RQIO macro routine initiates the I/O operation, and returns control to the requesting application.

The IORB may specify either synchronous or asynchronous execution.

When the IORB specifies synchronous I/O (bit 9 of I_CT1=0), return to the calling application is delayed by the Executive until the I/O operation is complete. On return of control to the application, both the return status field in I_CT1 of the IORB and the R1 register will contain one of the status codes shown in Table 4-7.

When the IORB specifies asynchronous I/O (bit 9 of I_CT1=1), control returns immediately without waiting for I/O completion, and the instruction at the return point is executed as soon as the system initiates the requested I/O operation.

To obtain the completion status (in R1 register) when using asynchronous I/O, the application should issue a $WAIT or $TEST macro call. The $WAIT macro call blocks execution of the application until the requested I/O operation is marked as complete. At completion of the I/O operation, the application should first check the R1 register to see that the I/O request was successful. Any error will be defined there. Hardware errors will be indicated in the IORB software status word I_ST (see Table 4-9). The $TEST macro call returns the completion status of the IORB if the I/O transfer has completed, or returns status 0801 if I/O has not completed. The $TEST macro call allows the application to continue processing pending completion of an I/O transfer, whereas $WAIT does not.

Residual range, indicated in the IORB, shows how much of the requested data was transferred. The residual range value in I_RSR of the IORB is meaningful only when the A-bit in the I_ST item (Table 4-8) of the IORB has been set on.

## DATA STRUCTURES

Data structures control the interactions among an application program, its line protocol handlers, and the devices it uses. The input/output request block (IORB) is the interface between the application and line protocol handler. The IORB and its use are described below in general terms. Later sections describe the contents of specialized IORBs for each of the line protocol handlers.

### Input/Output Request Blocks

The IORB is the standard means for requesting a physical I/O service. As described in this section, the IORB is used with physical I/O communications interfaces. The physical I/O part (through 13+2*$AF in Figure 4-2) is directly usable at the physical I/O interface. The logical part (beginning with 14+2*$AF) is used by forms processing software, by the local mail facility (interprocess communication), and by the message group request blocks MGIRB, MGCRB, and MGRRB.

Generated by the Input/Output Request Block macro call ($IORB), the IORB contains all the information that an application requesting an I/O service must specify to define the operation to be performed. Specifically, the IORB includes the following:

- Logical resource number (LRN) that identifies the I/O device being addressed

- Location and size of the buffer to be used for physical I/O transfers

- Type of operation as specified by the function code and optional device-specific word

- Information, concerning results of the I/O request, returned by the line protocol handler to the application after I/O completion.

When the IORB is used with a $RQIO macro call, the device named in the IORB should have been previously reserved by a Get File ($GTFIL) macro call. The logical resource number (LRN) required by the IORB can be obtained by issuing a Get File Information ($GIFIL) macro call. For further details, see the description of the Request I/O ($RQIO) macro call in Volume II.

Figure 4-2 shows the format of the IORB. Table 4-8 defines the separate entries in the IORB. Later sections in the manual describe the significance of the device-specific word (I_DVS), software status word I_ST, and other IORB words for the various line protocol handlers.

NOTES

1. The labels used in the figure to identify IORB fields (e.g., I_CT1, I_ADR) can be generated by the $IORBD macro call, described in Volume II.

2. The offset symbol $AF signifies the number of words required to specify a memory address. In this system, $AF is equivalent to two words.

3. The asterisk (*) in the formulas in the "Word" column of Figure 4-2 and Table 4-8 is a multiplication sign.

4. The shaded fields in Figure 4-2 are for system use only. Fields not shaded must be initialized by the application requesting the I/O operation.

## IORB SOFTWARE STATUS WORD (I_ST)

The line protocol handler maps into the IORB software status word I_ST (Table 4-9) the return status of the hardware or line protocol handler.

The bit settings in the software status word I_ST indicate to the application the status of the hardware, as shown in Table 4-9.

The meanings of bit settings in the software status word I_ST for specific devices are shown in tables in later sections that describe the line protocol handlers for those devices.

| WORD | LABEL | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| –3 | I_LRX | RESERVED | | | | EXTENDED LRN | | | | | | | | | | | |
| –$AF<br>–1 | I_RRB<br>I_SEM | REQUEST BLOCK POINTER, OR SEMAPHORE NAME | | | | | | | | | | | | | | | |
| 0 | I_LNK | RESERVED FOR SYSTEM USE AS POINTER | | | | | | | | | | | | | | | |
| $AF | I_CT1 | RETURN STATUS | | | | | | | | T | W | U | S | P | R | D | 1 |
| 1+$AF | I_CT2 | LRN | | | | | | | | 0 | B | 0 | E | FUNCTION | | | |
| 2+$AF | I_ADR | BUFFER ADDRESS – 2-WORD POINTER | | | | | | | | | | | | | | | |
| 2+2*$AF | I_RNG | RANGE – NUMBER OF BYTES TO BE TRANSFERRED | | | | | | | | | | | | | | | |
| 3+2*$AF | I_DVS | DEVICE – SPECIFIC WORD | | | | | | | | | | | | | | | |
| 4+2*$AF | I_RSR | RESIDUAL RANGE – NUMBER OF BYTES NOT TRANSFERRED | | | | | | | | | | | | | | | |
| 5+2*$AF | I_ST | DEVICE STATUS WORD 1 | | | | | | | | | | | | | | | |
| 6+2*$AF | I_EXT | TOTAL IORB EXTENSION LENGTH (IN WORDS) | | | | | | | | PHYSICAL I/O EXTENSION LENGTH (IN WORDS) | | | | | | | |
| 7+2*$AF | I_DV2 | DEVICE – SPECIFIC WORD 2 | | | | | | | | | | | | | | | |
| 8+2*$AF | I_FCS | DEVICE PHYSICAL CONTROL WORD 1 | | | | | | | | | | | | | | | |
| | | FUNCTION CODE 1 | | | | | | | | FUNCTION CODE 2 | | | | | | | |
| 9+2*$AF | I_HDR | DEVICE PHYSICAL CONTROL WORD 2<br>(VALID IF B-BIT (E) IS 1) | | | | | | | | | | | | | | | |
| 10+2*$AF | I_ST2 | DEVICE PHYSICAL CONTROL WORD 3 | | | | | | | | | | | | | | | |
| | | SECOND STATUS WORD | | | | | | | | TIME–OUT VALUE | | | | | | | |
| 11+2*$AF | I_QDP | DEVICE PHYSICAL CONTROL WORD 4 | | | | | | | | | | | | | | | |
| 12+2*$AF | I_TAB | DEVICE DEPENDENT; ATTRIBUTE OR DESCRIPTOR | | | | | | | | | | | | | | | |
| 13+2*$AF | I_CON | PREORDER CONTROL | | | | | | | | | | | | | | | |
| 14+2*$AF | I_LOG | FIRST WORD OF LOGICAL PART OF IORB | | | | | | | | | | | | | | | |

Figure 4-2. Communications Input/Output Request Block (IORB)

Table 4-8. Communications Input/Output Request Block (IORB)

| Word | Label | Bits | Description |
|---|---|---|---|
| -3 | I_LRX | 0-3 | Reserved for system use. |
| | | 4-15 | Extended logical resource number (LRN). If byte 0 (bits 0 to 7) of I_CT2 contains the value 253 (x'FD'), this field indentifies the device to be used. |
| -$AF | I_RRB/ I_SEM | 0-31 | Depending on the S- or R-bits of I_CT1, this word contains a task request block pointer (R-bit on) or a semaphore name (S-bit on). Set by user; used by system at termination of request. |
| 0 | I_LNK | 0-31 | Reserved for system use; two-word pointer. |
| $AF | I_CT1 | 0-7 | Return status. (See Table 4-7). |
| | | 8 (T) | This bit is set (on) while the request using this IORB is executing; it is reset when the request terminates. The system controls this bit; user should not change it. |
| | | 9 (W) | Wait bit. Set by user when the requesting task is not to be suspended pending the completion of the request that uses this IORB. If W = 0, then the D, R, and S bits may not be set. |
| | | A (U) | User bit. User may or may not use this bit; system does not change it. |
| | | B (S) | Release semaphore indicator. Values: 0 = No semaphore in I_SEM. 1 = Release, on completion, semaphore item named in I_SEM. |
| | | C(P) | Must be set by user if IORB is to be referenced by a Wait Any ($WAITA) macro call. If set, IORB can be referenced only by $WAIT or $WAITA issued by the requesting task. |

Table 4-8 (cont). Communications Input/Output Request Block (IORB)

| Word | Label | Bits | Description |
|---|---|---|---|
| $AF (cont) | I_CT1 (cont) | D (R) | Return IORB indicator. Values: 0 = No request pointer in I_RRB. 1 = Dispatch task request block named in I_RRB; after completion of this request, the system executes $RQTSK, using I_RRB. |
|  | I_CT1 | E (D) | Delete IORB indicator, used usually with B (S) and D (R) bits. 0 = No delete. 1 = When task terminates, return memory to the pool where IORB is the first entry of its memory block. |
|  |  | F | I/O bit. Must be set to 1. |
| 1+$AF | I_CT2 | 0-7 | Logical resource number (LRN). If this field contains any value other than 253 (x'FD'), it indentifies the device to be used. If this field contains 253, I_LRX contains the LRN value. |
|  |  | 8 | Must be 0. |
|  |  | 9 (B) | Byte index. 0 = buffer begins in leftmost byte of word; 1 = buffer begins in rightmost byte. |
|  |  | A (P) | Reserved for system use. |
|  |  | B (E) | Extended IORB indicator. 0 = Standard (nonextended) IORB. 1 = IORB extended as specified by I_EXT. Set by user. (See I_EXT below.) |
|  |  | C-F | Function code. See Table 4-10. |
| 2+$AF | I_ADR | 0-31 | Buffer address; 2-word pointer. |
| 2+2*$AF | I_RNG | 0-15 | Range. Indicates number of bytes to be transferred. |
| 3+2*$AF | I_DVS | 0-15 | Device-specific information. Set by user. |
| 4+2*$AF | I_RSR | 0-15 | Residual range. Indicates the number of bytes not transferred. Filled in by the system on completion of the order. |

Table 4-8 (cont). Communications Input/Output Request Block (IORB)

| Word | Label | Bits | Description |
|---|---|---|---|
| 5+2*$AF | I_ST | 0-15 | Status word. Reflects the mapping of the hardware status into software status format. Set by system after I/O completes. Used also by the ATD and STD LPHs as a peripheral address field. |
| 6+2*$AF | I_EXT | 0-7 | Left byte: Number of words in the IORB extension, not including this I_EXT word. |
| | | 8-F | Right byte: Number of words in physical part of IORB extension, not including this I_EXT word; must be less than or equal to total extension length shown in the left byte. |
| | | | This word applies only when the B (E) bit in I_CT2 is 1. |
| 7+2*$AF | I_DV2 | 0-F | Device-specific word 2. Contains device-specific information. |
| 8+2*$AF | I_FCS | 0-F | Device physical control word 1. |
| 9+2*$AF | I_HDR | 0-F | Device physical control word 2. |
| 10+2*$AF | I_ST2 | 0-F | Device physical control word 3. |
| 11+2*$AF | I_QDP | 0-F | Device physical control word 4. |
| 12+2*$AF | I_TAB | 0-F | Device physical control word 5. |
| 13+2*$AF | I_CON | 0-F | Device physical control word 6. |
| 14+2*$AF | I_LOG | 0-F | First word of logical part of IORB. Used by forms processing software, in message control, and by local mail message group request blocks. |

## Communications Function Codes

All line protocol handlers perform similar functions for the devices and applications that they service. These functions are performed by the line protocol handler's request and interrupt processing codes.

Table 4-9. Software (I_ST) Status Codes

| Bit in IORB's I_ST | Meaning When Bit Set On |
|---|---|
| 0 | - |
| 1 | Read error (PVE, BTF) |
| 2 | Data service rate error |
| 3 | Lost line bid or RVI received (BSC, BTF) |
| 4 | Communication control block service error |
| 5 | No stop bit on character input (TTY); conversational reply received (BSC3780); IORB purged because of BREAK signal (ATD, TTY, BTF) |
| 6 | Long record (BSC, ATD, BTF) |
| 7 | ITB/ETB or ETX received (BSC); poll failure (PVE) |
| 8 | Framing error (ATD); NAK limit reached (PVE, BTF) |
| 9 | Checksum or parity error limit reached (PVE); parity error (ATD) |
| A | Nonzero residual range |
| B | Phone disconnect |
| C | End-of-transmission received (BSC); Invalid response received (BTF) |
| D | Transparent message received (BSC) |
| E | NAK limit reached (BSC); Busy received (BTF) |
| F | Nonexistent resource; bus parity error; fatal uncorrectable memory error |

An application can request specific functions by providing a function code in the IORB supplied when it requests I/O service. The application uses the last four bits of its IORB's I_CT2 entry (see Figure 4-2) to enter the function code for the functions summarized in Table 4-10.

Table 4-10. Communications LPH Function Codes

| Function Code in IORB | Communications Function |
|---|---|
| 1 | Write |
| 2 | Read |
| 5 | Define-form (used only by the ATD LPH) |
| 9 | Read break |
| A | Connect |
| B | Disconnect |

The connect and disconnect functions may be used with non-communications devices, in which case they are processed as "no-ops". Thus, no matter how connected to the system, all TTY devices and noninteractive (e.g., card reader and printer) devices can be controlled by the same application program. This provision is useful for program development and test purposes.

WRITE FUNCTION (CODE 1)

This function allows data to be written to a specific device. When a line protocol handler (LPH) receives a write request, it transfers the indicated data from the application's buffer to the device, according to the information supplied in the device-specific word of the application's IORB.

READ FUNCTION (CODE 2)

This function allows data to be read from a specific device. When the LPH receives a read request, it tranfers data from the device to the application's buffer, according to the information supplied in the device-specific word of the application's IORB.

DEFINE-FORM FUNCTION (CODE 5)

This function is used by the ATD LPH for forms processing to define fields, their subfields, and their attributes. A define-form order does not itself result in actual physical I/O. (Refer to Section 8 for more details.)

READ BREAK (CODE 9)

This function allows an application to be notified of an operator-generated break condition on synchronous or asynchronous terminals. The function also allows for the selective cancellation of outstanding read break orders. (Refer to Section 8 for more details.)

CONNECT FUNCTION (CODE A)

The connect function provides a logical and physical connection between an application program and a communications device.

As a logical function, the connect function is a request to use the specified communications device. If that resource is being used, an error return results. In that case, the application must determine whether that resource is sharable (as established by the installation's procedures) and proceed accordingly.

As a physical function, the connect function establishes a physical path to the communications device associated with the specified logical resource number (LRN). This implies, when the device is to be connected over a switched line, that the system software should complete call establishment on the line associated with that device. The request times out after 5 minutes.

If the connect function is not completed, the system will not process any requests for the communications device and will return an error status.

The connect function must be requested before any other function, since communications devices are configured into the system in a disconnected state.

DISCONNECT FUNCTION (CODE B)

The disconnect function provides both the logical (normal and abnormal) and physical disconnection between the application and a communications device.

As a logical function, the disconnect function indicates that the use of the designated device is to be terminated.

For a logical disconnect, issue a disconnect request (function code B) with the E-bit in I_DVS set off (dequeue remaining IORBs for device) and the F-bit in I_DVS set on (do not hang up phone). At this point, any pending read or write requests are returned to the application program with a B status (device unavailable). Continued use of the device requires that the application program issue a connect.

As a physical function, the disconnect function must specify, by setting the F-bit in I_DVS to 0, the physical disconnection of a line.

# *Section 5*
# *DATA STRUCTURE GENERATION*

This section summarizes the macro routines that generate and define system data structures. There are two kinds of data structures: those that apply to system control functions and those that apply to file system functions. The macro calls that generate both kinds of data structures are described in detail in Volume II of this manual. The formats of the generated data structures are tabulated in Appendix C.

## SYSTEM CONTROL DATA STRUCTURES

System control data structures that are visible to the user consist of the following:

- Request blocks
- Parameter block and wait lists.

### Request Blocks

When requesting certain operations, tasks generate request blocks in order to specify the parameters of the requested operation. The first five words of all request blocks are identical in format; these words pass parameters to the system. The W-bit, for example, in the third word of request blocks, specifies whether or not the requesting task is to be suspended until the requested operation is completed. Additional words convey to the system information specific to the request block type.

One type of request block, the task request block, passes parameters to the requested task as well as to the system. These additional parameters are arguments that control the execution of the task being requested. They are entered into a variable-length field of the task request block called an argument list.

Table 5-1 lists the request blocks and the macro calls that generate them.

The arguments supplied with each of the above macro calls sets values for fields of the corresponding request block. For example, the first argument of the Input/Output Request Block ($IORB) macro call specifies the logical resource number (LRN) of the device to perform the input/output operation. The number specified by this argument is placed in the request block generated by the $IORB macro call.

## Request Block Offsets Macro Calls

Each request block macro call is paired with a request block offsets macro call. Request block offsets macro calls generate tags for every entry in a corresponding request block, allowing symbolic references to request block fields by application code. These tags are not generated by request block macro calls. An application may use a request block macro call to construct a request block, and then issue a request block offsets call to facilitate modification of the existing block by executing code.

Unlike the arguments of request block macro calls, the tags generated by offset macro calls refer to all fields of the corresponding request block. Offset tags refer to fields in which values are returned by the system, whereas macro call arguments refer only to fields in which values are entered by the user.

Table 5-1. Request Blocks

| Request Block | Macro Call |
|---|---|
| Clock request block (CRB) | $CRB |
| Input/output request block (IORB) | $IORB |
| Message group request blocks | |
| Message group control (MGCRB) | $MGCRB |
| Message group initialization (MGIRB) | $MGIRB |
| Message group recovery (MGRRB) | $MGRRB |
| Semaphore request block (SRB) | $SRB |
| Task Request block (TRB) | $TRB |

As mentioned above, the first five words of all request blocks are identical. Each offset macro call, however, refers to these words by different tags. The fourth word of the semaphore request block, for example is S_CT1, whereas the fourth word of the task request block is labeled T_CT1. The programmer, therefore, can include several types of offset macro calls in an application without multiply defining symbols.

No arguments are specified with offsets macro calls. Only one offsets macro call of a particular type is allowed in an application.

Macro calls that generate offsets tags for request blocks are listed below:

| | |
|---|---|
| Clock Request Block Offsets | $CRBD |
| Input/Output Request Block Offsets | $IORBD |
| Message Group Control Request Block Offsets | $MGCRT |
| Message Group Initialization Request Block Offsets | $MGIRT |
| Message Group Recovery Request Block Offsets | $MGRRT |
| Semaphore Request Block Offsets | $SRBD |
| Task Request Block Offsets | $TRBD |

## Parameter Block and Wait Lists

The parameter block and wait lists are system control data structures that differ in format from request blocks.

A parameter block is equivalent to the task request block's argument list, mentioned above; it is generated by the Parameter Block ($PRBLK) macro call. Parameter blocks are a standard means of passing arguments between tasks. By specifying the number and length of arguments, as well as the arguments themselves, a parameter block allows the receiving task to locate each argument in the list (or block).

A wait list is a list of request blocks to be serviced before the task issuing the wait list macro call completes its own execution. A wait list consists of a count of the number of request blocks to be waited on, followed by the request blocks' addresses. The list is generated by the Wait List ($WLIST) macro call. Another macro call, Wait on Request List ($WAITL) causes the task manager to scan the wait list and activate the waiting task when any of the listed requests are marked as completed.

A multiple wait list contains the same information as does the wait list; in addition, it specifies the number of request blocks that must be completed before a waiting task is to be activated. A multiple wait list is generated by the Generate Multiple Wait List ($WLSTM) macro call.

## FILE SYSTEM DATA STRUCTURES

A file information block (FIB) is used by running applications to request input/output operations. Other data structures are used outside of program execution by functions that create and modify files, or return information about files already created. Both types of data structures are discussed below.

### File Information Block

The file information block is the means by which an application passes to the file system the parameters of a requested input/output operation. The fields of the FIB specify such items as a file's logical resource number (LFN), by which the system identifies the file; the record or block size; and the address of the user's buffer.

The following macro calls use an FIB:

| | |
|---|---|
| Open File | $OPFIL |
| Close File | $CLFIL |
| Test File | $TIFIL, $TOFIL |
| Read Record | $RDREC |
| Write Record | $WRREC |
| Rewrite Record | $RWREC |
| Delete Record | $DLREC |
| Read Block | $RDBLK |
| Write Block | $WRBLK |
| Wait Block | $WTBLK |

FILE INFORMATION BLOCK MACRO CALL

The file information block is generated by the File Information Block ($FIB) macro call. An $FIB macro call can do one of the following:

- Build a new FIB with default values determined by the system
- Build a new FIB, specifying its contents by means of arguments supplied with the call
- Generate instructions to alter the contents of an existing FIB.

The file system performs three functions: data management, file management, and storage management. An FIB pertinent to one type of function may not be pertinent to another type. Data management involves the transfer of logical records; storage management, the transfer of blocks of records. The fields of an FIB applicable to data management, would specify the size and location of logical records; the fields of an FIB applicable to storage management, the size and location of record blocks. The FIB macro call has two sets of arguments, pertaining to data/file management and storage management.

## FIB OFFSET MACRO CALLS

For the same reason that the $FIB has more than one set of arguments, there are several macro calls that generate FIB offset tags. (The use of offset tags is explained earlier in this section.) The FIB offsets macro calls are:

```
$FIBDM
$FIBSM
$TFIB
```

The $FIBDM and $FIBSM macro calls generate sets of tags that are specific to data/file management and storage management, respectively. A third offsets macro call, $TFIB, generates two sets of tags, applicable both to data/file and to storage management. The $TFIB macro call would be issued by an application requesting both data/file management and storage management services.

### Macro Call Argument Structures

Macro calls that create and modify files, or return information about existing files must specify many parameters, as a file can take many different forms. Typically, these macro calls have a single argument that points to a list of arguments, or an argument structure. Offsets macro calls are available to facilitate modifying or referring to the fields of an argument structure. Table 5-2 lists the file system macro calls that require argument structures and the offsets macro calls that supply tags for these structures.

### Size Tags

Data structures for file system macro calls can either be declared statically or built dynamically. In the latter case, memory for the structure is dynamically obtained by means of the Get Memory ($GMEM) macro call at the time of execution. The memory thus obtained should be cleared to zeros to ensure that fields of the structure reserved for future are zero-filled. Each offset macro call generates a size tag for specifying the size of the corresponding data structure. The size tag can be used to specify the amount of memory requested (when issuing the Get Memory macro call), or used to clear the structure to zeros.

Example:

$B4 points to a file information block (FIB). The structure is cleared with the instructions:

```
    LDV  $R1,F_SZ-1       R1=SIZE OF FIB MINUS 1
$A  CL   $B4.$R1          CLEAR ONE WORD
    BDEC $R1,>-$A         LOOP UNTIL ALL WORDS CLEARED
```

Table 5-2. Argument Structures and Offsets Tags

| Calls Requiring Argument Structures | Calls Generating Offset Tags |
|---|---|
| Create File ($CRFIL) | Create File Parameter Block Structure Offsets ($CRPSB) |
| | Create File Record Descriptor Block Offsets ($CRRDB) |
| Get Device Information ($GIDEV) | Get Device Information Parameter Structure Block Offsets ($DIPSB) |
| Get File Access Rights ($GAFIL) | Get File Access Rights Parameter Structure Block Offsets ($GAPSB) |
| Get File Information ($GIFIL) | Create File Record Descriptor Block Offsets ($CRRDB) |
| | Get File Information Parameter Structure Block Offsets ($GIPSB) |
| | Get File Information File Attribute Block Offsets ($GIFAB) |
| Get Name ($GNFIL) | Get Name Parameter Structure Block Offsets ($GNPSB) |
| Grow File ($GRFIL) | Grow File Parameter Structure Block Offsets ($GRPSB) |
| Modify File ($MDFIL) | Modify File Parameter Structure Block ($MDPSB) |
| Shrink File ($SHFIL) | Shrink File Parameter Structure Block Offsets ($SHFIL) |

# Section 6
# DEVICE DRIVERS

This section describes the internal system software known as device drivers and some related data structures, principally the input/output request block (IORB), by which the device driver is controlled. A device driver performs all data transfers between a non-communication peripheral device and an application program requesting input/output. Line protocol handlers analogously perform input/output between applications and communications devices, which are attached to a multi-line controller (MLC). The remainder of this section describes non-communication peripheral device drivers. Line protocol handlers are described in later sections.

## INPUT/OUTPUT DRIVERS

Applications can request and instruct drivers to do physical I/O directly by means of the Request Input/Output ($RQIO) and Input/Output Request Block ($IORB) macro calls. Most often, applications invoke drivers indirectly when issuing file system macro calls such as Read Record ($RDREC) and Write Record ($WRREC). When executing these calls, the file system generates IORBs to instruct the drivers. If an application requests a driver to do physical I/O ($RQIO), the application must have previously reserved the peripheral device ($GTFIL) via the file system.

Drivers are reentrant programs capable of supporting the concurrent operation of several devices at the same time. The priority level at which they run is selected by the user when the system is configured. Requests by applications for I/O activate the drivers, which in turn initiate data transfer that is simultaneous with the operation of the central processor. Drivers process an interrupt from the peripheral device to the central processor when the transfer of data is terminated.

## Device Driver Data Structures

Two data structures control the interaction between an application program, its device drivers, and the devices the program uses. These structures are the input/output request block (IORB) and the resource control table (RCT).

The IORB is the interface which does physical I/O directly between the application and its device driver. Through the IORB, the application defines the I/O service that it wishes to be performed. Also, the IORB contains information returned by the driver to the requesting task concerning the outcome of the I/O request. The resource control table (RCT) is the interface between the driver and its device(s), and is not normally accessible to users of Honeywell-supplied drivers described in this section.

*

## Device Driver Conventions

The following conventions apply to all input/output device drivers.

- The I/O request block (IORB) is the standard control structure used by a driver. It is described later in this section.
- The $RQIO macro call is used to request a driver.
- The B4 register contains the address of the IORB supplied by the caller; the IORB contains the LRN of the device to be used.
- The I/O-specific words of the IORB (I_CT2 through I_DVS) are not modified by the driver.
- If a device becomes inoperable, it can be disabled with an operator command and another device can be substituted.
- Drivers are reentrant and interrupt driven; one driver supports many devices of the same type.
- Synchronous and asynchronous I/O are supported.
- The hardware status is always mapped into the software status word in the task's IORB (I_ST) before the driver relinquishes control.

## Driver Functions and Function Codes

All drivers perform similar functions on behalf of the devices and application tasks they service. These functions are carried out by the driver's request processing and interrupt processing code. The application task requests specific functions by providing a function code in the IORB that it supplies when it requests I/O service. These specific function codes are summarized in Table 6-1 and discussed under the specific function heading in the following pages.

The application task uses the last four bits of the IORB entry I_CT2 to enter the function code for the functions summarized in Table 6-1.

### CONNECT FUNCTION (fc=A)

This function may be used with noninteractive devices for program compatibility. The driver of a noninteractive device treats this function as a NOP and immediately posts the IORB back to the requester with successful status (operation complete). *

### DISCONNECT FUNCTION (fc=B)

The disconnect function as a logical function indicates that use of the indicated device is terminated. Termination may be either normal or an abort of all queued read or write requests issued by this user program. *

### WAIT ONLINE FUNCTION (fc=0)

This function allows a caller to wait until a device becomes ready for use, or until a specific time interval has passed.

All non-communications devices generate interrupts when their availability changes. For example, when a printer runs out of paper, an interrupt is generated and the device is not ready for use; when the paper is installed and the device is again ready, another interrupt is generated. *

When a driver receives a service request from a task using the "wait online" function code in the IORB that it supplies (0000 in the last four bits of I_CT2), and the device is not ready, the driver sets a timer for 5 minutes and suspends. When the driver is reactivated, either by a ready interrupt from the device or by a timeout, it deactivates the timer, checks the device-ready bit in the hardware status word, and places a 0 or 6 value in the return status field of the IORB depending on the condition of that bit. See Table 6-2 and the return status codes for the $RQIO macro call (which is described in Volume II). The rightmost 2 digits of the 4-digit hexadecimal status code are placed in the return status field.

Table 6-1. Input/Output Function Code

| IORB Function Code | Device | | | | |
|---|---|---|---|---|---|
| | Card Reader | Card Reader/ Punch | Printer | Disk | Magnetic Tape |
| 0 | Wait online | Wait online | Wait online | Wait online | Wait online |
| 1 | NA | Write (punch) | Write | Write | Write |
| 2 | Read | Read | NA | Read | Read |
| 3 | NA | Write file mark (punch) | NA | NA | Write file mark |
| 4 | NA | NA | NA | NA | Position block* |
| 5 | NA | NA | NA | Format write | NA |
| 6 | NA | NA | NA | Format read | Position file** |
| 9 | NA | NA | NA | NA | NA |
| A | Connect | Connect | Connect | Connect | Connect |
| B | Disconnect | Disconnect | Disconnect | Disconnect | Disconnect |
| E | NA | NA | NA | Read disabled device | Read disabled device |

*Positive range of one is forward space to start of next block.
Negative range of one is backspace to beginning of previous block.

**Positive range of one is forward space to next tape mark.
Zero range is backspace to previous tape mark.
Negative range of -1 is rewind to BOT.
Negative range of -2 is rewind to BOT and unload.
Negative range of -3 is write at EOF gap.

Table 6-2. Return Status Codes (Last Two Digits)

| Code Number (Hexadecimal) | Meaning |
|---|---|
| 00 | No error, operation complete |
| 01 | Request block already busy (T=1) |
| 02 | Invalid LRN |
| 03 | Invalid wait |
| 04 | Invalid parameters |
| 05 | Device not ready |
| 06 | Device timeout on other than connect |
| 07 | Hardware error |
| 08 | Device disabled |
| 09 | File mark encountered |
| 0A | Controller unavailable |
| 0B | Device unavailable |
| 0C | Inconsistent request |
| 10 | Device timeout on connect |
| 11 | Write protect error |
| 17 | Memory access violation |
| 31 | Possible disk head failure |
| 32 | Possible disk media failure |

NOTES

1. When status 07 is returned, look in I_ST to identify the specific hardware error.

2. Status 0B is returned with every read or write IORB that has been aborted by a disconnect request with queue abort. The disks and tapes are disabled until the system's automatic volume recognition routine calls the enable device function.

3. Status 0C indicates illogical peripheral driver requests (e.g., read or write before connect; duplicate connect or disconnect requests; write after disconnect).

The wait online function should not be issued to a device that is currently ready for use unless you expect it to become unavailable for a limited time (e.g., the operator has been instructed to change a volume mounted on a disk device currently in use).

## WRITE FUNCTION (fc=1)

The write function is available for all devices except the card reader. This function allows the writing of data to a particular device. When a driver receives a write request, it transfers the indicated data from a user buffer to the device according to the specifications supplied in the task's IORB.

## READ FUNCTION (fc=2)

The read function is available for all devices except local and remote printers. This function allows reading data from a particular device. When a driver receives a read request, it transfers the data from the specified device to a user buffer according to the specifications supplied in the requesting task's IORB.

## READ DISABLED DEVICE FUNCTION (fc=E)

This function, available only to disk or magnetic tape devices, allows the driver to bypass the device-disabled test during validity checking.

This function is used by the system's automatic volume recognition (AVR) module, which recognizes the volume label of the volume on the disabled device, then enables the device so that attempts to read data from it can continue.

## WRITE TAPE MARK FUNCTION (fc=3)

The write tape mark function, which is available to magnetic tape devices, allows you to put a mark block on a referenced magnetic tape.

## POSITION BLOCK FUNCTION (fc=4)

The position block function, which is available to magnetic tape devices, allows you to position a referenced magnetic tape forward or backward one block.

## FORMAT WRITE (fc=5)

The format write function, available only to disk devices, allows you to format a disk device. The number of sectors per track depends upon the device type.

## FORMAT READ (fc=6)

The format read function, available only to disk devices, allows you to read all identifier and data fields on a track. The read begins at the first sector following the index mark and proceeds in the order in which the identifiers are recorded.

## POSITION TAPE MARK FUNCTION (fc=6)

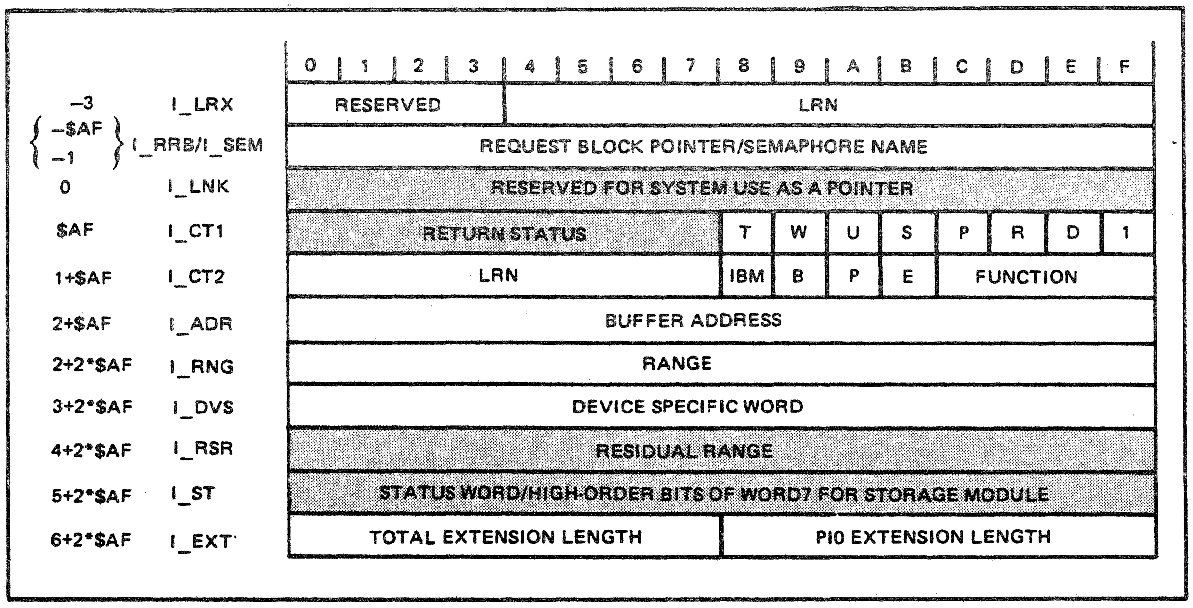
The position tape mark function, which is available to magnetic tape devices, allows the user to:

- Position forward a referenced magnetic tape beyond the next tape mark
- Position backward a referenced magnetic tape before the current tape mark
- Rewind to BOT
- Rewind to BOT and unload.

## INPUT/OUTPUT REQUEST BLOCK

The input/output request block (IORB) contains all information that a task requesting an I/O service can specify to define the operation to be performed. In addition, it contains information returned by the driver to the requesting task concerning the outcome of its I/O request.

Figure 6-1 shows the format of a nonextended IORB. Unshaded fields must be initialized by the task requesting the I/O operation. The shaded fields are set by the driver to return information about the I/O request to the caller, or are controlled by the Executive.



| | | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -3 | I_LRX | RESERVED | | | | LRN | | | | | | | | | | | |
| -$AF / -1 | I_RRB/I_SEM | REQUEST BLOCK POINTER/SEMAPHORE NAME | | | | | | | | | | | | | | | |
| 0 | I_LNK | RESERVED FOR SYSTEM USE AS A POINTER | | | | | | | | | | | | | | | |
| $AF | I_CT1 | RETURN STATUS | | | | | | | | T | W | U | S | P | R | D | 1 |
| 1+$AF | I_CT2 | LRN | | | | | | | | IBM | B | P | E | FUNCTION | | | |
| 2+$AF | I_ADR | BUFFER ADDRESS | | | | | | | | | | | | | | | |
| 2+2*$AF | I_RNG | RANGE | | | | | | | | | | | | | | | |
| 3+2*$AF | I_DVS | DEVICE SPECIFIC WORD | | | | | | | | | | | | | | | |
| 4+2*$AF | I_RSR | RESIDUAL RANGE | | | | | | | | | | | | | | | |
| 5+2*$AF | I_ST | STATUS WORD/HIGH-ORDER BITS OF WORD7 FOR STORAGE MODULE | | | | | | | | | | | | | | | |
| 6+2*$AF | I_EXT | TOTAL EXTENSION LENGTH | | | | | | | | PIO EXTENSION LENGTH | | | | | | | |

86-040

Figure 6-1. Format of I/O Request Block

Table 6-3 defines the specific IORB entries in a nonextended IORB. (See the "Communications Processing Functions" section for descriptions of IORB extensions.) Table 6-4 defines the software status word (I_ST) in the IORB. Device-specific IORB information is provided in the separate device driver descriptions later in this section.

NOTE

The offset labels used to refer to IORB fields (e.g., I_CT1, I_ADR) can be generated by the $IORBD macro call, which is described in Volume II.

## CALLER INTERFACE WITH DEVICE DRIVER

To request execution of an I/O operation, the caller must issue a $RQIO macro call with $B4 pointing to the IORB to be serviced. If the IORB specifies synchronous I/O (W-bit reset), the issuing task is suspended until the I/O operation is complete.

If the IORB specifies asynchronous I/O, the instruction at the return point is executed as soon as the system queues the IORB on the driver's level. The application may issue a $WAIT or $TEST macro call when appropriate for the asynchronous request.

Upon return from a synchronous request, the caller must check the R1 register to see if the request was successful. Upon return from an asynchronous request, the caller must check R1 to see if the request was accepted and successfully initiated. For either type of request, any invalid user argument is indicated in R1. Hardware errors are defined in IORB entry I_ST (see Table 6-4).

Residual range denotes how much of the requested data transfer was actually performed. If I_RSR equals zero, all data was transferred. For an asynchronous request, register R1 would be checked on return from the Request I/O macro call; R1, I_ST, and I_RSR should be checked after return from a $WAIT macro call.

Those fields not shaded in Figure 6-1 must be initialized by the task requesting the I/O operation. The remaining fields are set by the driver to return information about the I/O request to the caller or are controlled by the Executive. Table 6-3 describes the purpose of each field.

Other information needed to perform the I/O request is found in the IORB. The caller-supplied standard function code in I_CT2 is mapped by each driver into one or more device functions required to perform the actual request.

The LRN supplied by the caller in the IORB serves as a device identifier.

Table 6-3. Contents of I/O Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| -3 | I_LRX | 0-3 | Reserved for system use. |
| | | 4-15 | Extended logical resource number (LRN). If byte 0 (bits 0 to 7) of I_CT2 contains the value 253 (x'FD'), this field indentifies the device to be used. |
| -$AF<br>-1 | I_RRB/<br>I_SEM | 0-31<br>0-15 | Depending on the S- or R-bits of I_CTl, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). Set by user; used by system at termination of request. |
| 0 | I_LNK | 0-31 | Reserved for system use. 2-word pointer to indirect request block. |
| $AF | I_CTl | 0-7 | Return status |
| | | 8(T) | This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it. |
| | | 9(W) | Wait bit. Set by user if requesting task is not to be suspended pending completion of the request that uses this IORB. For a $OPMSG call, the setting of the W- bit in output IORB controls return to the caller. For a $OPRSP call, setting of W-bit in <u>input</u> IORB controls return to the caller; setting of W-bit in output IORB has no significance. For either call, return to caller is immediate if significant W-bit is on. If significant W-bit is off, return to caller occurs after the order is completed. |
| | | A(U) | User bit. User may or may not use this bit; the system does not change it. |
| | | B(S) | Release semaphore indicator.<br><br>0 = No release; 1 = Release, on completion, semaphore item named in I_SEM. |

Table 6-3 (cont). Contents of I/O Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| $AF (cont) | I_CT1 (cont) | C(P) | Must be set by user if IORB is to be referenced by a Wait Any ($WAITA) macro call. If set, IORB can be referenced only by $WAIT or $WAITA issued by the requesting task. |
| | | D(R) | Return IORB indicator. |
| | | | 0 = No dispatch; 1 = Dispatch task request block named in I_RRB after completion of this request. If 1, system executes $RQTSK, using I_RRB, when the task terminates. |
| | | E(D) | Delete IORB indicator. Used usually with the B(S) and D(R) bits. |
| | | | 0 = No delete; 1 = Delete and when task terminates, return memory to pool where IORB is first entry of its memory block. |
| | | F(1) | Implicit task start address. Must always be 1 for IORB. |
| 1+$AF | I_CT2 | 0-7 | Logical resource number (LRN). If this field contains any value other than 253 (x'FD'), it indentifies the device to be used. If this field contains 253, I_LRX contains the LRN value. |
| | | 8(IBM) | IBM-type request. Changes interpretation of I_DVS to task word, and of I_RSR and I_ST to configuration words A and B, respectively. |
| | | 9(B) | Byte index. 0 = buffer begins in leftmost byte of word; 1 = buffer begins in rightmost byte. Must be off if input/output buffer begins at left byte of word whose address is contained in word 3 (I_ADR) of IORB. Must be on if input/output buffer begins at the right byte. |
| | | A(P) | Private space; reserved for system use. |

Table 6-3 (cont). Contents of I/O Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 1+$AF (cont) | I_CT2 (cont) | B(E) | Extended IORB indicator. 0 = Standard (nonextended) IORB; 1 = IORB extended to at least 6+2*$AF items. Set by user. (See I_EXT below.) |
| | | C-F | Function code. Driver or LPH function, see Table 6-1. |
| 2+$AF | I_ADR | 0-31 | Buffer address. 2-word pointer. Word address of message buffer (which contains an output message or is to receive an input message). |
| 2+2*$AF | I_RNG | 0-15 | Range. Number of bytes to be transferred. Used as input field for cartridge disk or mass storage unit. Buffer size in bytes. This is the length of an output message or the maximum length allowed for an input message. |
| 3+2*$AF | I_DVS | 0-15 | Device-specific information. |
| 4+2*$AF | I_RSR | 0-15 | Residual range. Indicates the number of bytes <u>not</u> transferred. Filled in by the system on completion of the order. Used by the cartridge disk and mass storage unit drivers as a data offset value. |
| 5+2*$AF | I_ST | 0-15 | Modified device status. Shows mapping of hardware status into software status format. See Table 6-4. Set by user as input field high-order bits of sector number of mass storage unit. Set by system after I/O completion. |
| 6+2*$AF | I_EXT | 0-7 | Left byte. Number of words, in binary, in the IORB extension, not including this I_EXT word. |
| | | 8-15 | Right byte. Number of words, in binary, in physical I/O part of IORB extension, not including this I_EXT word. This count must be less than or equal to the total extension length specified in the left byte (0-7). This word is present only when the B(E) bit in I_CT2 is 1. (See Section 7 for a description of IORB extensions.) |

## Table 6-4. IORB Software Status Word (I_ST)

| Bit Position | Device | | | | | | |
|---|---|---|---|---|---|---|---|
| | Card Reader | Card Reader/ Punch | Printer | Diskette | Lark Disk | Cartridge Module Disk and Disk Storage Unit | Magnetic Tape |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | | | |
| 2 | Over/ underun | Data service rate error | 0 | Over/underrun | Over/underrun | Over/underun | Retryable error |
| 3 | Mark sense mode | Invalid ASCII code | End of form | Deleted field | Write protect error | Write protect error | Write pro- tect error |
| 4 | 40-column mode | Punch echo or read registration | 0 | Read error | Read error | Read error | Corrected media error |
| 5 | 51-column mode | Light/dark check | 0 | Device fault | Invalid seek | Invalid seek | Tape mark |
| 6 | External clock track | Card jam | 0 | Missed data synchron- ization | Missed data synchron- ization | Missed data synchronization | BOT |
| 7 | Read check | 0 | 0 | Unsuccessful search | Unsuccessful search | Unsuccessful search | EOT |
| 8 | ASCII code error | 0 | 0 | Two-sided | Missed clock pulse | Missed clock pulse | Long record |
| 9 | 0 | 0 | 0 | 0 | Missed sector pulse | Successful retry | Nonretryable error |
| A | 0 | 0 | 0 | Seek error | Seek error | | 0 |
| B | 0 | 0 | 0 | 0 | 0 | 0 | Operation check |
| C | 0 | 0 | 0 | 0 | 0 | 0 | High density |
| D | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| E | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| F | Fatal error | Fatal error | Fatal error | Fatal error | Fatal error | Fatal error | Fatal error |

NOTES

1. Nonexistent resource, bus parity, and uncorrected memory errors are combined into bit 15 of I_ST, but each occurrence is noted separately in the RCT.

2. The online drivers will flag, in the RCT, corrected memory errors and driver or hardware corrected errors.

## DEVICE DRIVERS

The remainder of this section discusses the device drivers in the following order:

- Card reader/Card reader-punch driver
- Printer driver
- Disk driver *
- Magnetic tape driver.

### Card Reader/Card Reader-Punch Driver

The card reader and card reader-punch devices are serviced by a single driver. The driver uses six function codes; i.e., read, write, write file mark (reader/punch only), connect, disconnect, and wait online. In addition, its IORB word I_DVS can be coded to define the character code of the input; namely, ASCII or verbatim. These values are specified in the IORB as defined in Table 6-6.

The translation/mapping of these codes from punched card format into memory on reading is described below.

In addition to the standard driver functionality discussed earlier, this driver also:

- Detects and discards unsolicited interrupts
- Detects an end-of-file condition and sets the appropriate return status (ASCII GS character in column 1 of any card=EOF)
- Detects "device not ready" condition and sets appropriate error condition.

#### ASCII MODE

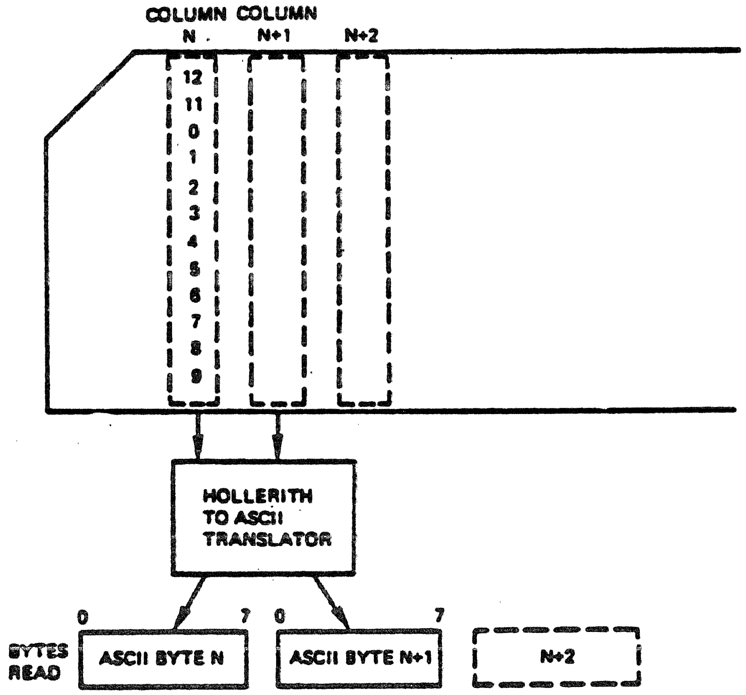
In this mode, punched cards are processed as shown in Figure 6-2. Each card column consisting of a 12-bit ASCII card code is converted into an 8-bit ASCII byte and stored in the main memory.

The ASCII card code table as specified in American National Standard X3.26 is given in Table 6-5. Note that no multiple punches in rows 1 through 7 are allowed and, thus, the 12-bit card code allows a maximum of 256 unique codes to be defined.

Translation is done by the card reader attachment that also provides a software-visible IORB status indicator that is set whenever an invalid ASCII card code is detected. This error condition is signaled by a 0107 in the R1 register if any card column read had a hole pattern that was not one of the legal hole patterns given in Table 6-5. The invalid card code causes an ASCII-EO (all 1s) code to be loaded in the main memory.

Table 6-5. Hollerith-ASCII Code Table

| | b8 b7 b6 b5 | 0 0 0 0 | 0 0 0 1 | 0 0 1 0 | 0 0 1 1 | 0 1 0 0 | 0 1 0 1 | 0 1 1 0 | 0 1 1 1 | 1 0 0 0 | 1 0 0 1 | 1 0 1 0 | 1 0 1 1 | 1 1 0 0 | 1 1 0 1 | 1 1 1 0 | 1 1 1 1 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| b4b3b2b1 | COL ROW | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | COL ROW |
| 0000 | 0 | NUL 12-0-9-8-1 | DLF 12-11-9-8-1 | SP NO PCH | 0 0 | @ 8-4 | P 11-7 | ` 8-1 | p 12-11-7 | 11-0-9-8-1 | 12-11-0-9-8-1 | 12-0-9-1 | 12-11-9-8 | 12-11-0-9-6 | 12-11-8-7 | 12-11-0-8 | 12-11-9-8-4 | 0 |
| 0001 | 1 | SCH 12-9-1 | DC1 11-9-1 | ! 12-8-7 | 1 1 | A 12-1 | Q 11-8 | a 12-0-1 | q 12-11-8 | 0-9-1 | 9-1 | 12-0-9-2 | 11-8-1 | 12-11-0-9-7 | 11-0-8-1 | 12-11-0-9 | 12-11-9-8-5 | 1 |
| 0010 | 2 | STX 12-9-2 | DC2 11-9-2 | " 8-7 | 2 2 | B 12-2 | R 11-9 | b 12-0-2 | r 12-11-9 | 0-9-2 | 11-9-8-2 | 12-0-9-3 | 11-0-9-2 | 12-11-0-9-8 | 11-0-8-2 | 12-11-0-8-2 | 12-11-9-8-6 | 2 |
| 0011 | 3 | ETX 12-9-3 | DC3 11-9-3 | # 8-3 | 3 3 | C 12-3 | S 0-2 | c 12-0-3 | s 11-0-2 | 0-9-3 | 9-3 | 12-0-9-4 | 11-0-9-3 | 12-0-8-1 | 11-0-8-3 | 12-11-0-8-3 | 12-11-9-8-7 | 3 |
| 0100 | 4 | ECT 9-7 | DC4 9-8-4 | $ 11-8-3 | 4 4 | D 12-4 | T 0-3 | d 12-0-4 | t 11-0-3 | 0-9-4 | 9-4 | 12-0-9-5 | 11-0-9-4 | 12-0-8-2 | 11-0-8-4 | 12-11-0-8-4 | 11-0-9-8-7 | 4 |
| 0101 | 5 | ENQ 0-9-8-5 | NAK 9-8-5 | % 0-8-4 | 5 5 | E 12-5 | U 0-4 | e 12-0-5 | u 11-0-4 | 11-9-5 | 9-5 | 12-0-9-6 | 11-0-9-5 | 12-0-8-3 | 11-0-8-5 | 12-11-0-8-5 | 11-0-9-8-3 | 5 |
| 0110 | 6 | ACK 0-9-8-6 | SYN 9-2 | & 12 | 6 6 | F 12-6 | V 0-5 | f 12-0-6 | v 11-0-5 | 12-9-6 | 9-6 | 12-0-9-7 | 11-0-9-6 | 12-0-8-4 | 11-0-8-6 | 12-11-0-8-6 | 11-0-9-8-4 | 6 |
| 0111 | 7 | BEL 0-9-8-7 | ETB 0-9-6 | ' 8-5 | 7 7 | G 12-7 | W 0-6 | g 12-0-7 | w 11-0-6 | 11-9-7 | 12-9-8 | 12-0-9-8 | 11-0-9-7 | 12-0-8-5 | 11-0-8-7 | 12-11-0-8-7 | 11-0-9-8-5 | 7 |
| 1000 | 8 | BS 11-9-6 | CAN 11-9-8 | ( 12-8-5 | 8 8 | H 12-8 | X 0-7 | h 12-0-8 | \ 11-0-7 | 0-9-8 | 9-8 | 12-8-1 | 11-0-9-8 | 12-0-8-6 | 12-11-0-8-1 | 12-0-9-8-2 | 11-0-9-8-6 | 8 |
| 1001 | 9 | HT 12-9-5 | EM 11-9-8-1 | ) 11-8-5 | 9 9 | I 12-9 | Y 0-8 | i 12-0-9 | y 11-0-8 | 0-9-8-1 | 9-8-1 | 12-11-9-1 | 0-8-1 | 12-0-8-7 | 12-11-0-1 | 12-0-9-8-3 | 11-0-9-8-7 | 9 |
| 1010 | 10 | IF 0-9-5 | SUB 9-8-7 | * 11-8-4 | : 8-2 | J 11-1 | Z 0-9 | j 12-11-1 | z 11-0-9 | 0-9-8-2 | 9-8-2 | 12-11-9-2 | 12-11-0 | 12-11-8-1 | 12-11-0-2 | 12-0-9-8-4 | 12-11-0-9-8-2 | 10 |
| 1011 | 11 | VT 12-9-8-3 | ESC 0-9-7 | + 12-8-6 | ; 11-8-6 | K 11-2 | [ 12-8-2 | k 12-11-2 | { 12-0 | 0-9-8-3 | 9-8-3 | 12-11-9-3 | 12-11-0-9-1 | 12-11-8-2 | 12-11-0-3 | 12-0-9-8-5 | 12-11-0-9-8-3 | 11 |
| 1100 | 12 | FF 12-9-8-4 | FS 11-9-8-4 | , 0-8-3 | < 12-8-4 | L 11-3 | \ 0-8-2 | l 12-11-3 | \| 12-11 | 0-9-8-4 | 12-9-4 | 12-11-9-4 | 12-11-0-9-2 | 12-11-8-3 | 12-11-0-4 | 12-0-9-8-6 | 12-11-0-9-8-4 | 12 |
| 1101 | 13 | CR 12-9-8-5 | GS 11-9-8-5 | - 11 | = 8-6 | M 11-4 | ] 11-8-2 | m 12-11-4 | } 11-0 | 12-9-8-1 | 11-9-4 | 12-11-9-5 | 12-11-0-9-3 | 12-11-8-4 | 12-11-0-5 | 12-0-9-8-7 | 12-11-0-9-8-5 | 13 |
| 1110 | 14 | SO 12-9-8-6 | RS 11-9-8-6 | . 12-8-3 | > 0-8-6 | N 11-5 | ^ 11-8-7 | n 12-11-5 | ~ 11-0-1 | 12-9-8-2 | 9-8-6 | 12-11-9-6 | 12-11-0-9-4 | 12-11-8-5 | 12-11-0-6 | 12-11-9-8-2 | 12-11-0-9-8-6 | 14 |
| 1111 | 15 | SI 12-9-8-7 | US 11-9-8-7 | / 0-1 | ? 0-8-7 | O 11-6 | _ 0-8-5 | o 12-11-6 | DEL 12-9-7 | 11-9-8-3 | 11-0-9-1 | 12-11-9-7 | 12-11-0-9-5 | 12-11-8-6 | 12-11-0-7 | 12-11-9-8-3 | EO 12-11-0-9-8-7 | 15 |



NOTES: 1. This translator will provide a status indicator which will be set whenever an illegal Hollerith code is read.

2. The translator shown above is in the card reader attachment.

Figure 6-2. ASCII Card-to-Memory Code Formatting

## VERBATIM MODE

In this mode, punched cards are processed as shown in Figure 6-3. The card column pattern is stored in bits 4 through 15 of the main memory word with bits 0 through 3 set to zero. All two-hole patterns are valid during a verbatim mode operation. The device-specific fields in the IORB are given below.

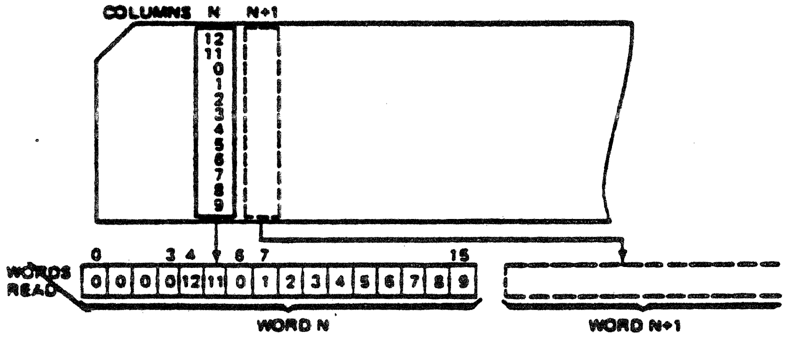


Figure 6-3. Verbatim Mode Formatting

## CARD READER/CARD READER-PUNCH DEVICE-SPECIFIC IORB FIELDS

Table 6-6 defines the device-specific fields in the IORB not previously defined. Refer to "Driver Functions and Function Codes" earlier in this section.

## CARD READER/CARD READER-PUNCH HARDWARE STATUS CODE MAPPING

The card reader/card reader-punch controller returns to the driver various codes, which are made visible to the application by way of the IORB as shown in Tables 6-7 and 6-8.

Table 6-6. Card Reader/Card Reader-Punch IORB Fields

| IORB Word | Field | Definition | Use |
|---|---|---|---|
| I_CT2 | Function code | 0 = Wait online | See "Wait Online Function" earlier in this section. |
| | | 1 = Write | Driver "writes" card for "range" number of bytes. |
| | | 2 = Read | Driver "reads" card for "range" number of bytes. |
| | | 3 = Write file mark | Driver "writes" end-of-file card. |
| | | A = Connect<br>B = Disconnect | See "Connect Function" and "Disconnect Function" earlier in this section. |
| I_RNG | Range | 0≤range≤32K-1 | If range is greater than card size, residual range reflects the difference. |
| I_DVS | Device specific | 0 12 13 14 15<br>\| 0 0 \| mode \|<br>mode: 0=ASCII<br>2=verbatim | Defines character set of data being read. |
| I_RSR | Residual range | 0≤ initial range | Detects device malfunction. |

Table 6-7. Card Reader IORB Hardware/Software Status Code Mapping

| Hardware Status | IORB Word I_ST | Meaning If Bit Set |
|---|---|---|
| 0 | - | Device ready |
| 1 | - | Attention |
| 2 | 2 | Data service rate error |
| 3 | 3 | Mark sense mode |
| 4 | 4 | 40-column card mode |
| 5 | 5 | 51-column card mode |
| 6 | 6 | External clock track |
| 7 | 7 | Read check error |
| 8 | 8 | ASCII code error |
| - | - | |
| - | - | |
| - | - | |
| 12 | - | Corrected memory error |
| 13 | 15 | Nonexistent resource/fatal error |
| 14 | 15 | Bus parity error/fatal error |
| 15 | 15 | Uncorrectable memory error/fatal error |

Table 6-8. Card Reader/Punch Hardware/Software Status Code Mapping

| Hardware Status | IORB Word I_ST | Meaning If Bit Set |
|---|---|---|
| 0 | - | Device ready |
| 1 | - | Attention |
| 2 | 2 | Data service rate error |
| 3 | 3 | Invalid ASCII code |
| 4 | 4 | Punch echo or read registration |
| 5 | 5 | Light/dark check |
| 6 | 6 | Card jam |
| 7 | - | |
| 8 | - | |
| - | - | |
| - | - | |
| - | - | |
| 12 | - | Corrected memory error |
| 13 | 15 | Nonexistent resource/fatal error |
| 14 | 15 | Bus parity error/fatal error |
| 15 | 15 | Uncorrectable memory error/fatal error |

## Printer Driver

The printer driver performs all data transfers to line and serial printers as well as terminal print devices. Format control of printing can be achieved by supplying a control byte as the first entry in a data buffer. The control byte is included in the range count of the IORB for the request. The presence of a control byte is indicated by bit 4 of the IORB's I_DVS word.

### PRINT CONTROL BYTE

The format of the control byte is:

| Bit: | 0 | 1 | 2 | 3 | 4 7 |
|---|---|---|---|---|---|
| Field: | Y | PP | | V | COUNT |

The control byte, if supplied, is interpreted differently by line/serial printer and terminal printer devices. The significance of the control byte for both device types is shown in Table 6-9 under "Action Caused".

Table 6-10 summarizes control byte settings as hexadecimal and ASCII values.

These conventions permit a control byte (e.g., 41) to be used with a printer driver (whose default I_DVS word is all zeros)
* without extra spacing or overprinting. This driver supports a terminal format convention that does not require a control byte. This convention treats the first byte of the range as data, with spacing as follows:

* Printer - Space one line or skip to head-of-form if at end-of-form, then print.

Bit 4 (F-bit) in I_DVS controls format selection.

### PRINTER DEVICE-SPECIFIC IORB FIELDS

Table 6-11 defines the IORB fields whose contents are specific to the printer driver.

### PRINTER HARDWARE/SOFTWARE STATUS CODE MAPPING

Table 6-12 indicates the hardware/software status code mapping for printers.

Table 6-9. Print Control Byte

| Field | Action Caused | |
|---|---|---|
| | Line/Serial Printer (Space Before Print) | Terminal Printer (Space After Print) |
| Y | Not used. | 0 = Use carriage return and/or line feed in I_DVS.<br><br>1 = Ignore carriage return and/or line feed in I_DVS. |
| PP | 00 Print; ignore V and count/fields; single space to end-of-form; then skip to head-of-form.<br><br>01 Do not print; perform actions defined in V and count fields.<br><br>10 Print; perform actions defined in V and count fields.<br><br>11 Reserved for system use. | Not used. |
| V | 0 Prespace according to count field.<br><br>1 If count = 0, skip to head-of-form. If count is between 1 and 11, and the VFU option is present, skip to the VFU channel defined by the count field.<br><br>If count is greater than 11, or there is no VFU option, do one prespace. | 0 = No prespace.<br><br>1 = Prespace three lines; count field must be 0. |

## Disk Driver

A single disk driver supports the following disk devices: diskette, Lark II disk, cartridge module disk, mass storage unit, and all fixed-disks (Winchester technology).

Table 6-10. Print Control Byte Summary

| Code | | Resulting Action |
|---|---|---|
| Hexadecimal | ASCII | |
| | | Line/Serial Printers |
| 00-1F | NUL-US | Single space, then print; skip to head-of-form at end-of-form. |
| 20-2F | - / | Space count lines; do not print. |
| 30-3F | 0 - ? | Skip to VFU channel number in count, do not print. |
| 40-4F | @ - O | Space count number of lines, print. |
| 50-5F | P - _ | Skip to VFU channel number in count, print; 50 = skip to head-of-form. |
| 60-6F | - o | Reserved for future use. |
| 70-7F | p - DEL | Reserved for future use. |
| | | Terminal Printers |
| 00-0F<br>20-2F<br>40-4F | NUL-SI<br>- /<br>@ - O | No prespace, print. |
| 10-1F<br>30-3F<br>50-5F | DLE-US<br>0 - ?<br>P - _ | Prespace three lines; print. |
| 60-6F | ' - o | Reserved for future use. |
| 70-7F | p - DEL | Reserved for future use. |

## DISK DRIVER CONVENTIONS FOR DISKETTE

The following driver conventions apply to diskette:

- The disk driver supports both 8- and 5 1/4-inch diskette devices. For the 8-inch diskette, both single- and double-sided diskettes may be used. Support of the 5 1/4 inch diskette consists of double-sided and double-density.

- The driver does not explicitly reference the volume ID of the diskette; therefore, the user must ensure that volumes addressed are on the proper drives.

Table 6-11. Printer IORB Fields

| IORB Word | Field | Definition | Use |
|---|---|---|---|
| I_CT2 | Function code | 0 = wait online<br>1 = write | See "Driver Function and Function Codes". Driver will "write" from I_ADR "range" number of bytes. |
| I_RNG | Range | 0≤range≤32K-1 | If range is greater than line size, residual range reflects the difference. |
| I_DVS | Device-specific | 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15<br>0 0 0 0 F 0 0 0 0 0 0 0 0 0 0 0<br><br>F: 0 = Assumes line printer format control (control byte)<br>1 = Assumes terminal format control (no control byte)<br><br>All other bits must be zero. | |
| I_RSR | Residual range | See Note | |
| I_ST | Software status word | Shown below | Mapped from RCT hardware status. |

NOTE

For cases where original range is less than or equal to line length, the value in the residual range has the following meanings:

0 - Completed space/print operation.

other - Residual spacing value is contained in the value I_ST value field.

- All sector addresses used in the IORB are relative to track 0/sector 0.
- The driver converts the volume relative sector number, defined in the IORB, into physical track and sector numbers, and to a "side" value for two-sided diskette, which it then sends to the device to define the operation.
- The driver can support more than one diskette device, as long as each device is configured at a different level.

Table 6-12. Printer Hardware/Software Status Code Mapping

| Hardware Status | IORB I_ST | Meaning If Bit Set |
|---|---|---|
| 0 | - | Device ready |
| 1 | - | Attention |
| 2 | 2 | Lost data |
| 3 | 3 | End-of-form |
| 4 | 4 | Lines per inch: 0 = 6; 1 = 8 |
| 5 | 5 | Protocol error |
| 6 | 6 | Power up |
| 7 | 7 | Eight bit mode |
| - | - | |
| - | - | |
| - | - | |
| - | - | |
| 12 | - | Corrected memory error |
| 13 | 15 | Nonexistent resource/fatal error |
| 14 | 15 | Bus parity error/fatal error |
| 15 | 15 | Uncorrectable memory error/fatal error |

- A diskette sector is 128 bytes long (8 inch) or 256 bytes long (5 1/4 inch). If range is less than sector length, a write command will zero fill the rest of the sector. If range is greater than sector length on either a read or a write, the driver will read/write multiple sectors including switching to the next adjacent track, if necessary.

- There are 16 sectors per track for 5 1/4 diskette; 26 sectors per track for 8 inch diskette.

- There are three models:

  <u>1 track per cylinder:</u>
  77 cylinders

  <u>2 tracks per cylinder:</u>
  77 cylinders
  80 cylinders

- If hardware errors occur, the operation (seek or read/write) will be retried up to eight times (five retries and three retries with recalibrate).

- If the device is not ready, a return status of "device not ready" (5) will be returned.

Tables 6-13 and 6-14 define IORB fields specific to
* diskette. Other IORB fields are described in Table 6-3.

Table 6-13. Diskette IORB Fields

| IORB Word | Field | Definition | Use |
|---|---|---|---|
| I_CT2 | Function code | 0 = wait-online<br>1 = write data<br>2 = read data<br>5 = format write<br>6 = format read<br>E = read disabled device | Specifies I/O operation. |
| I_DVS | Device-specific | Relative sector number | Driver converts this to physical track number and physical sector number on the track, and to a "side" value for two-sided diskette. |
| I_ST | Software status | Shown below | Hardware status word from diskette (following I/O). |
| I_RSR | Residual range | $0 \leq$ original range | Residual range will always be equal to zero (i.e., transfer completed) unless there is a hardware malfunction, or an invalid track number is supplied during a read or write operation. |

NOTE

To ensure compatibility of an application with other devices, clear to zero the IORB words I_RSR and I_ST before making an I/O request.

## DISK DRIVER CONVENTIONS FOR LARK DISK

The Lark device is a random access, rotating 8-inch disk with both removable and fixed platters.

The following conventions apply to Lark devices:

- Sector size is 256 bytes; there are 64 sectors per track.

Table 6-14. Diskette Hardware/Software Status Code Mapping

| Hardware Status | IORB I_ST | Meaning if Bit Set |
|---|---|---|
| 0 | - | Device ready |
| 1 | - | Attention |
| 2 | 2 | Data service rate error |
| 3 | 3 | Deleted field |
| 4 | 4 | Read error |
| 5 | 5 | Device fault |
| 6 | 6 | Missed data synchronization |
| 7 | 7 | Unsuccessful search |
| 8 | - | Two-sided diskette |
| - | - | |
| 10 | 10 | Seek error |
| 12 | - | Corrected memory error |
| 13 | 15 | Nonexistent resource/fatal error |
| 14 | 15 | Bus parity error/fatal error |
| 15 | 15 | Uncorrectable memory error/fatal error |

- The driver does not explicitly refer to the volume ID of the disk; the user must ensure that the volumes addressed are on the proper drives.

- All sector addresses used in the IORB are relative to cylinder 0, track 0, sector 0.

- There are two models:

  <u>2 tracks per cylinder:</u>
  204 cylinders
  622 cylinders

- The driver converts the volume relative sector number, defined in the IORB, into physical cylinder, track, and sector numbers, which it then sends to the device to define the operation.

- The Lark disk requires two LRNs, one for the fixed and one for the removable platter.

- Offset read capability is provided by specifying the desired displacement in the I_RSR field of the IORB.

- Offset write capabilities are not provided.

Tables 6-15 and 6-16 show IORB fields specific to the Lark device. Other IORB fields are described in Table 6-3.

Table 6-15. Lark Disk IORB Fields

| IORB Word | Field | Definition | Use |
|---|---|---|---|
| I_CT2 | Function Code | 0 = Wait online<br>1 = Write<br>2 = Read<br>5 = Format write<br>6 = Format read<br>E = Read disabled device | Specifies I/O operation. |
| I_DVS | Device specific | Relative sector number | Driver converts this to physical cylinder, track, and sector number to locate the data needed. |
| I_ST | Software status | See Table 6-16 | Hardware status from disk (following I/O). |
| I_RSR | Residual range | $0 \leq$ original range | Prior to a read, an offset value may be specified here so that reading can begin at a location other than the physical sector boundary; after I/O operation, the field contains the number of bytes not transferred in the operation. |

NOTE

To ensure compatibility of an application with other disk devices, clear to zero the IORB word I_ST before requesting I/O.

## DISK DRIVER CONVENTIONS FOR MASS STORAGE UNIT

The following driver conventions apply to mass storage units:

- Sector size is 256 bytes; there are 64 sectors per track.
- The driver does not explicitly refer to the volume ID of the disk pack, so the user must ensure that the volumes addressed are on the correct drives.

Table 6-16. Lark Disk Hardware/Software
Status Code Mapping

| Hardware Status | IORB I_ST | Meaning If Bit Set |
|---|---|---|
| 0 | - | |
| 1 | - | |
| 2 | 2 | Over or underrun |
| 3 | 3 | Write protect error |
| 4 | 4 | Read error |
| 5 | 5 | Invalid seek |
| 6 | 6 | Missed data synchronization |
| 7 | 7 | Unsuccessful search |
| 8 | 8 | Missed clock pulse |
| 9 | 9 | Successful recovery |
| 10 | 10 | Seek error |
| 11 | - | |
| 12 | - | |
| 13 | - | |
| 14 | - | |
| 15 | 15 | Fatal error |

- All sector addresses in the IORB are relative to cylinder 0, track 0, sector 0. There are four models:

  <u>5 tracks per cylinder:</u>
  411 cylinders
  823 cylinders

  <u>19 tracks per cylinder:</u>
  411 cylinders
  823 cylinders

- The driver converts the volume relative sector number, defined in the IORB, into physical cylinder, track, and sector numbers, which it then sends to the device to define the disk address.

- The volume relative sector numbers exceed the maximum number that may be stored in one I_DVS word. Place high order bits in I_ST; low order bits in I_DVS.

- The mass storage unit requires only one LRN.

- The driver combines seek and data transfer functions. When errors occur, eight attempts are made to correct the error: five seek/data transfers, and three seek/data transfers with recalibrate.

- Offset read capability is provided by specifying the required displacement in the I_RSR field of the IORB.

- Offset write capability is not provided.
- When the driver notes a change in the ready state, it disables the device (by a software switch) and notifies the file manager to execute the automatic volume recognition procedures.

Tables 6-17 and 6-18 show IORB fields specific to the mass storage unit. Other IORB fields are described by Table 6-3.

Table 6-17. Mass Storage Unit IORB Fields

| IORB Word | Field | Definition | Use |
|---|---|---|---|
| I_CT2 | Function Code | 0 = Wait online<br>1 = Write<br>2 = Read<br>5 = Format write<br>6 = Format read<br>E = Read disabled device | Specifies I/O operation. |
| I_DVS | Device specific | Relative sector number | Driver converts this to the physical cylinder, track, and sector number to locate the data needed. |
| I_RSR | Residual range | $0 \leq$ original range | Prior to a read, an offset value may be specified here so that reading can begin at a location other than the physical sector boundary; after I/O operation the field contains the number of bytes not transferred in the operation.<br><br>After an I/O operation, the field contains the number of bytes not transferred. |
| I_ST | Software status | See Table 6-18 | Prior to an order, this field contains the high-order bits of the relative sector number. After the operation, it contains the hardware status from device. |

Table 6-18. Mass Storage Unit Status Code Mapping

| Hardware Status | IORBV I_ST | Meaning If Bit Set |
|---|---|---|
| 0 | - | |
| 1 | - | |
| 2 | 2 | Over/underrun |
| 3 | 3 | Device fault |
| 4 | 4 | Read error |
| 5 | 5 | Invalid seek |
| 6 | 6 | Missed data synchronization |
| 7 | 7 | Unsuccessful search |
| 8 | 8 | Missed clock pulse |
| 9 | 9 | Successful recovery |
| 10 | 10 | Reserved |
| 11 | - | |
| 12 | - | |
| 13 | - | |
| 14 | - | |
| 15 | 15 | Fatal error |

## DISK DRIVER CONVENTIONS FOR CARTRIDGE MODULE DISK

The following driver conventions apply to the cartridge module disk:

- Sector size is 256 bytes; there are 64 sectors per track.
- The driver does not explicitly refer to the volume ID of the disk; the user must ensure that the volumes addressed are on the correct drives.
- All sector addresses in the IORB are relative to cylinder 0, track 0, sector 0. The models are:

  <u>1 track per cylinder:</u>

  411 cylinders (removable, 8-megabyte)
  411 cylinders (fixed, 8-megabyte)
  823 cylinders (removable, 16-megabyte)
  823 cylinders (fixed, 16-megabyte)

  <u>3 tracks per cylinder:</u>

  823 cylinders (removable, 16-megabyte)
  823 cylinders (fixed, 48-megabyte)

  <u>5 tracks per cylinder:</u>

  823 cylinders (removable, 16-megabyte)
  823 cylinders (fixed, 80-megabyte)

The driver converts the volume relative sector number, defined in the IORB, into physical cylinder, track, and sector numbers, which it then sends to the device to define the disk address.

- The volume relative sector numbers exceed the maximum number that may be stored in I_DVS word; place high order sector bits in I_ST, low order sector bits in I_DVS.
- The fixed and removable portions of the cartridge module disk each require a separate LRN.
- The driver combines seek and data transfer functions. When errors occur, eight retries are made (five seek/data transfers, three seek/data transfers with recalibrate).
- Offset reading (not writing) is provided by specifying the required displacement in the I_RSR field of the IORB.
- When the driver detects a change in the ready state, it disables the device, both fixed and removable (with a software switch), and notifies file management to execute the system's automatic volume recognition procedures.

Tables 6-19 and 6-20 show the IORB fields specific to the cartridge module disk. Other IORB fields are described by Table 6-3.

*

## Magnetic Tape Driver

The magnetic tape driver manages all standard data transfer requests to and from 9-track phase encoded (PE), and 9-track nonreturn to zero inverted (NRZI) tape drives on one or more magnetic tape controllers. The tape drive characteristics supported by this tape driver are shown in Table 6-21.

The driver provides the following callable functions:

- Wait online
- Write
- Read (forward)
- Position block (forward and backward)
- Position forward or backward by tape mark, rewind to beginning of tape (BOT), rewind to BOT and unload.

The driver operates in the following modes:

- Odd parity

Table 6-19. Cartridge Module Disk IORB Fields

| IORB Word | Field | Definition | Use |
|---|---|---|---|
| I_CT2 | Function Code | 0 = Wait online<br>1 = Write<br>2 = Read<br>5 = Format write<br>6 = Format read<br>E = Read disabled device | Specifies I/O operation. |
| I_DVS | Device specific | Relative sector number | Driver converts this to the physical cylinder, track, and sector number to locate the data needed. |
| I_RSR | Residual range | 0 $\leq$ original range | Prior to a read, an offset value may be specified here so that reading can begin at a location other than the physical sector boundary; after I/O operation the field contains the number of bytes not transferred in the operation. |
| I_ST | Software status | See Table 6-20 | Prior to an order, this field contains the high-order relative sector bits. After the I/O operation, the field contains the hardware status, from the device. |

- Minimum data block, MDB (American National Standard specifies 18 or more characters per block in write, 12 or more in read)

- MDB-inhibited (if fewer than the specified number of characters must be read or written, this mode is required).

If MDB mode is specified for a write and the range is less than 18 characters, a parameter error is reported. If MDB mode is specified for a read and the range is less than 12 characters, you receive the first portion (requested range) of the first valid block and an unequal length check. If a "short record" is detected, a corrected media error is reported in status word, I_ST. If a record of less than 18 characters is written or less than 12 characters is read, the inhibit block size check bit (bit 12 of the device specific word, I_DVS) must be set.

Table 6-20. Cartridge Module Disk Status Code Mapping

| Hardware Status | IORB I_ST | Meaning If Bit Set |
|---|---|---|
| 0 | - | |
| 1 | - | |
| 2 | 2 | Over/underrun |
| 3 | 3 | Device fault |
| 4 | 4 | Read error |
| 5 | 5 | Invalid seek |
| 6 | 6 | Missed data synchronization |
| 7 | 7 | Unsuccessful search |
| 8 | 8 | Missed clock pulse |
| 9 | 9 | Successful recovery |
| 10 | 10 | Reserved |
| 11 | - | |
| 12 | - | |
| 13 | - | |
| 14 | - | |
| 15 | 15 | Fatal error |

Table 6-21. Characteristics of Supported Tape Drives

| Tape Drive Type | Speed (ips) | | | Density (bpi) | | | | | Parity | |
|---|---|---|---|---|---|---|---|---|---|---|
| | 45 | 75 | 125 | 6250 | 1600 | 800 | 556 | 200 | Odd | Even |
| 9-track NRZI | X | X | - | - | - | X | X | - | X | - |
| 9-track PE | X | X | - | - | X | X | - | - | X | - |
| 9-track GCR GCR mode | - | X | X | X | - | - | - | - | X | - |
| 9-track GCR PE mode | - | X | X | - | X | - | - | - | X | - |

Beginning of tape (BOT), end of tape (EOT), and end of file (EOF) conditions are reported for appropriate user action. If an error occurs in a case when the operation can be retried, the driver backspaces and reissues the order up to 32 times before reporting a hardware error. If an error occurs and no retry is possible, the driver rewinds and forward spaces to the problem block and reissues the order once before reporting a hardware error. The driver does not check the tape volume identifier.

The EOT return status is not returned for read operations; only the EOT status word bit is set. It is assumed that appropriate application software conventions will prevent reads that would force the tape off the end of the reel.

The resident magnetic tape driver is interrupt driven and must execute with a resident Executive and with the central processor in the privileged state. It can support, on an adapter, one data transfer simultaneously with one or more rewind/rewind-unload orders.

Tables 6-22 and 6-23 show IORB fields specific to magnetic tape devices. Other IORB fields are described by Table 6-3.

Table 6-22. Magnetic Tape IORB Fields

| Word | Field | Definition |
|---|---|---|
| I_CT2 | Function Code | 0 = Wait online<br>1 = Write<br>2 = Read<br>3 = Write filemark<br>4 = Position by block (see range)<br>6 = Position file (see range) |
| I_DVS | Device specific | 0 3 12 13-15<br>\| 0 0 0 \| B \| 0 0 0 0 0 0 0 0 \| I \| mode \|<br><br>B: 0 = Enable buffering in GCR buffered tape; 1 = Disable buffering in GCR buffered tape. Note that B is recognized only when GCR buffered tape is positioned at BOT. Must be specified for read, write, and write tape-mark operations when positioned at BOT.<br><br>I: 0 = Normal American National Standard block sizes; 1 = Inhibit sensing for American National Standard block size.<br><br>mode: Must be zero |
| I_RNG | Range | Write: 1 through 7FFF<br><br>Read: 0=Backspace one block; 1 through 7FFF is valid for data transfer<br><br>Position by block: Negative is back space; 0 is invalid; positive is forward space<br><br>Position by file: -2 = Rewind and unload<br>-1 = Rewind to BOT<br>0 = Backspace to previous tapemark<br>1 = Forward space to tapemark |
| I_RSR | Residual range | Nonzero when physical block exceeds range. |

Table 6-23. Magnetic Tape Hardware/Software Status Code Mapping

| RCT R_STTS | IORB I_ST | Meaning If Bit Set |
|---|---|---|
| 0 | - | Device ready |
| 1 | - | Attention |
| - | 1 | Rewinding |
| 2 | 2 | Error - Operation can be retried |
| 3 | - | Must be zero |
| - | 3 | Write protected |
| 4 | 4 | Corrected media error |
| 5 | 5 | Tape mark |
| 6 | 6 | BOT |
| 7 | 7 | EOT |
| 8 | 8 | Unequal record length |
| 9 | 9 | Error - Operation cannot be retried |
| 10 | 10 | Must be zero |
| 11 | 11 | Operation check |
| 12 | - | Corrected memory error |
| - | 12 | High density |
| 13 | 15 | Nonexistent resource/fatal error |
| 14 | 15 | Bus parity error/fatal error |
| 15 | 15 | Memory error - correction impossible/fatal error |

# *Section 7*
# *LINE PROTOCOL HANDLERS*

This section provides an overview of line protocol handlers. Subsequent sections describe specific line protocol handlers in detail. A communications protocol is a set of conventions or rules for the transmission of data. Communications protocols are used in the transfer of information between a local CPU and remote terminal or host CPU.

A line protocol handler (LPH) is the implementation of a particular communications protocol. Accordingly, each LPH supports a specific class of communications device, such as synchronous VIP terminals, or a communications protocol, such as the BSC 2780/BSC 3780 binary synchronous communications protocol. The following LPHs can be configured at system building:

ATD

The asynchronous terminal driver (ATD) supports asynchronous terminals, serial printers, and certain asynchronous X-ON/X-OFF protocols. The ATD LPH has five operational modes: Teletype compatible (TTY), field, block, ASPI, and X-ON/X-OFF.

STD

The synchronous terminal driver (STD) LPH supports specific synchronous terminal devices. These devices are the polled visual information projection (VIP) terminals and associated ROPs.

PVE

This synchronous LPH supports communications between computers. It emulates the polled VIP protocol for use in communications with remote Honeywell hosts that support polled VIP terminals.

BSC 2780/BSC 3780

This synchronous LPH supports communications between computers. It supports a station (device or computer) that utilizes the BSC 2780 or BSC 3780 binary synchronous communication (BSC) protocol in communications with a remote host.

TTY

This asynchronous LPH supports specific asynchronous terminal devices. These devices are classified as teleprinter-compatible, and include certain automatic send/receive (ASR), keyboard send/receive (KSR), and VIP terminals.

BSC-TF

This synchronous LPH supports communications between certain IBM terminals and computers running MOD 400. The BSC 3270 Terminal Facility supports IBM 3270 type terminals, printers, and cluster controllers.

The user may write a line protocol handler if it conforms to the same internal interface requirements used by the Honeywell-supplied line protocol handlers.

## LINE PROTOCOL HANDLER FUNCTIONS

Line protocol handlers transfer data between a communications device and the application that uses it. These handlers consist of two parts -- one resident in main memory and the other (called the channel control program (CCP)) resident in the MLC. The main memory-resident portion of the LPH is concerned with the processing of transmitted/received data at the block, message, or field level. The MLC-resident component is concerned with the transmission/reception of the individual data characters that make up the block, message, or field level data aggregate.

### Main Memory-Resident LPH

The portion of the line protocol handlers resident in main memory performs the following:

- When the system is bootstrapped:
  - Validates communication device types by reading the device's identification number.

- Initializes the communication device and sets it to the priority level at which it is to operate.

- Validates the application's input/output request block (IORB) fields.
- Converts user-supplied functions into device-specific MLC orders.
- Sets a timer and a monitor for data set status changes.
- Initiates the MLC I/O operation.
- Detects and processes MLC I/O interrupts.
- Reads return status from the communication device to ascertain result of an I/O operation.
- Processes error recovery, when possible.
- Processes unsolicited timeouts and data set status changes.
- Forms composite status in the IORB, including residual range, from all of the processed MLC orders.
- Posts back the application's IORB with the appropriate hardware and software status information.

## MLC-Resident LPH (CCP)

A channel control program (CCP) is the MLC-resident portion of an LPH. Through the appropriate hardware device-pac attached to the MLC, the channel control program controls transmission of data over communication lines. It serves to:

- Store or fetch individual characters in or from the buffer supplied with the IORB
- Perform translation, substitution, and deletion operations on individual characters
- Insert/delete protocol or device-specific header or trailer information.

## MLC COMMUNICATIONS HANDLER

The MLC communications handler receives processor orders from the main memory-resident portion of the line protocol handler and activates the appropriate channel control program (see above and Figure 7-1) to process the orders. The handler also:

- Processes a line protocol handler's requests for control functions or for data transfer operations

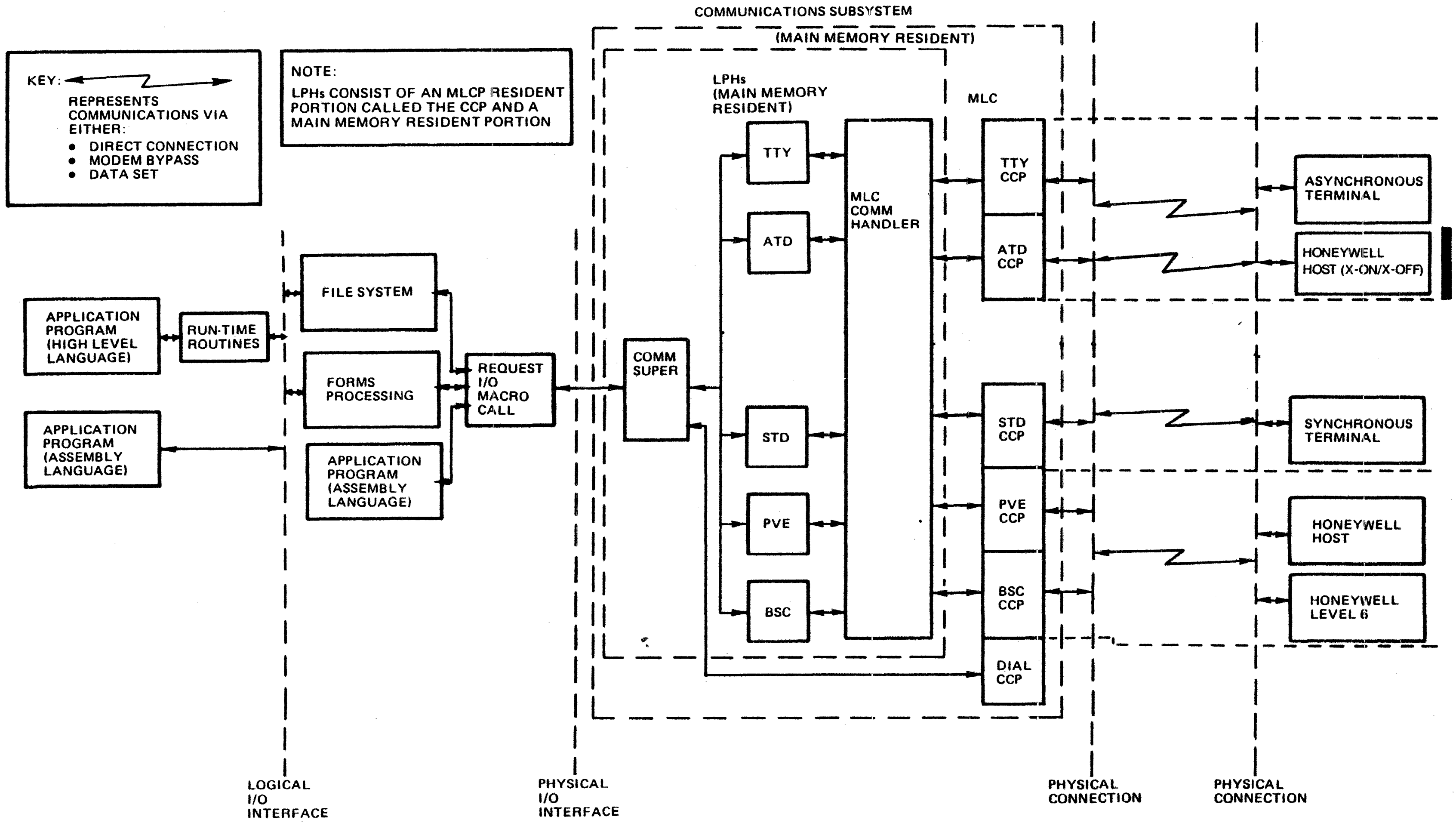


Figure 7-1. Communications Overview

- Services interrupts from the MLC and passes them to the appropriate line protocol handler.

## COMMUNICATIONS SUBSYSTEM OPERATION EXAMPLE

The following example and Figure 7-1 indicate the interaction of the communications subsystem's components in the processing of a connect, write, and disconnect request. The operations described apply to the physical I/O interface, without reference to a specific device or line protocol.

This example refers to the communications supervisor. The communications supervisor resides in main memory and provides the interface to communications applications programs at the physical I/O level. It queues application programs' requests for services, activates the appropriate line protocol handler, interacts with an application through system software when an I/O order is complete, and provides a set of common line protocol handler services (e.g., establishing/disestablishing data set communications, monitoring for time-outs and data set status changes).

Example:

1. The communications supervisor receives the application's connect request through the physical I/O interface, and passes it to the DIAL channel control program (CCP) within the multiline communications processor (MLC).

2. The DIAL CCP establishes a physical communication connection to the device.

3. The main memory-resident portion of the appropriate line protocol handler (LPH) processes the logical connection.

4. The communications supervisor passes the application's subsequent write request to the main memory-resident LPH, which translates the request into one or more MLC communications handler requests.

5. Each MLC communications handler request results in one or more orders to the MLC. (These orders not only describe the data to be transferred, but also cause the invocation and execution of the appropriate CCP.)

6. The appropriate CCP processes each of the write orders, which transmits the data to the device. During this time, the main memory-resident LPH terminates itself.

7. When the MLC senses completion of the data transfer, the CCP issues an interrupt, which is processed first by the communications supervisor and then by the MLC communications handler.

8. The MLC communications handler reactivates the main memory-resident portion of the LPH at the interrupt level, to minimally process the interrupt.

9. When processing is complete, control passes to the MLC communications handler, which causes processing at the interrupt level to be suspended.

10. If additional processing is necessary, the main memory-resident portion of the LPH can schedule itself to perform post-interrupt processing on a non-interrupt level.

11. The application's disconnect request is processed in the same manner as the connect request, but in the opposite order.

    a. The main memory-resident portion of the LPH performs the necessary logical disconnect processing.

    b. The physical connection is appropriately disconnected by the DIAL CCP.

The logic of the write operation in this example would apply to a read operation.

## EXTENDED LRN SUPPORT

The ATD, TTY, and BSC line protocol handlers support Logical Resource Numbers (LRNs) greater then 255. All line protocol handlers support the extended IORB format, even if an LPH does not support LRNs greater than 255.

## 8-BIT DATA SUPPORT

The ATD, STD, and PVE line protocol handlers support the extended ASCII character set (often called 8-bit data).

*

### 7-Bit Data Plus Parity and Truncation

In this mode, the data is transmitted and received with parity checking and generation. The high-order bit of the character is unconditionally truncated. This mode is supported by the ATD, STD, PVE, and TTY LPHs on the MLCP, MLC-16, and DPS 6/22 controllers.

### 7-Bit Data Plus Parity and Shift-In/Shift-Out

In this mode, the data is transmitted and received with parity checking and generation. The shift-in/shift-out technique is used to transfer the high-order bit of the character. This mode is supported by the ATD LPH on the MLC-16 controller.

### 8-Bit Data Without Parity

In this mode, the data is transmitted and received with no parity checking and generation. 8 bits of data is transfered per character. This mode is supported by the ATD, STD, and PVE LPHs on the MLCP, MLC-16, and DPS 6/22 controllers.

### 8-Bit Data Plus Parity

In this mode, the data is transmitted and received with parity checking and generation. 8 bits of data is transfered per character. This mode is supported by the ATD LPH on the MLC-16 controller.

## MODEM SUPPORT

For asynchronous devices, the communications subsystem provides the following modem support:

- Bell System Data Sets: Types 103A, 113F, 202, 212A
- Honeywell modem bypass
- Any modem type defined by the user at system building
- Honeywell-supplied direct-connect cables.

For medium speed synchronous communications, the communications subsystem provides the following modem support:

- Bell System Data Sets: Types 201A, 201B, 201C, 203, or 208A

- Honeywell modem bypass

- Honeywell-supplied direct connect cables

- Any modem type defined by the user at system building time.

For high-speed synchronous communications, the communications subsystem provides support for the Bell System Data Sets: Types 301B or 303.

## AUTO CALL UNIT

When configured into the system, the Auto Call Facility uses an Auto Call Unit (ACU) to initiate a line connection with a remote auto answer data set. The facility operates in the following manner:

1. The user associates the Auto Call Unit with a particular communications channel at system building time by using the ACU CLM directive.

2. The user enables the Auto Call Facility by setting bit 2 of the I_DVS word to one on a connect request. The facility is supported by all LPHs.

3. When the connect request is processed, the system attempts to dial a line, using a list of telephone numbers supplied at system building, the first entry of which is null. The first number to be dialed can then be specified with a Set Dial ($SDL) macro call or with the Set Autodial Telephone Number (SDL) command. If the first number on the list is not specified (by the macro call or command), the system skips to the next number on the list.

4. The facility dials each number on the list three times at 40-second intervals until the list is exhausted or a connection made, whichever occurs first.

5. The facility checks that a connection to the modem has been made.

6. When the connection has been made, control is passed to the LPH, which processes the logical portion of the connect request.

The Auto Call Unit supports Data Auxiliary Set Automatic Calling Units 801A and 801C. The ACU adapter and the adapter for its associated data line must be on the same controller.

Two data set options are required to use the Auto Call Unit:

- The option that terminates the call, through the data set, after the DSS (data set status change) goes on

- The option that stops the ACR timer when the DSS goes on.

## COMMUNICATIONS SUBSYSTEM ERROR AND CORRECTION PROCEDURES

The communications subsystem detects errors that may occur over communications lines by means of parity checking, block checking, and timeout checking.

## Parity Error Check

The system sends a parity (check) bit with each transmitted character. The parity bit, plus the number of character bits set to 1, will always be an odd or even-numbered total for every character, according to whether transmission is odd parity (total is an odd number) or even parity (total is an even number). The ATD and TTY line protocol handlers support parity error checking.

## Block Error Check

The communications subsystem uses two kinds of block error checking: the longitudinal redundancy check (LRC) and the cyclic redundancy check (CRC). The computed check characters are known as block check characters (BCC).

### LONGITUDINAL REDUNDANCY CHECK (LRC)

The LRC is a simple check that is applied to the entire message. The system appends an LRC character, which is an exclusive OR of all the characters in the message, to the end of every message. The STD and PVE line protocol handlers use the LRC method in 7-bit data mode and CRC16 method in 8-bit data mode.

### CYCLIC REDUNDANCY CHECK (CRC)

The CRC method is also block-oriented. The system computes the CRC block check character(s), using special algorithms applied to the data to be checked. The system then appends the BCC to the message. The BSC, STD, and PVE line protocol handlers use the CRC method of checking errors.

### BSC BLOCK CHECK CHARACTER (BCC)

In ASCII transmission, the 8-bit BCC is the result of an exclusive OR operation on all bits transmitted, beginning with the first character following the STX and ending with the ITB , ETB, or ETX control character. It is based on the polynomial:

$$X^8 + 1 .$$

In EBCDIC transmission the BCC is 16 bits, and is calculated by the system with the checking polynomial:

$$1 + X^2 + X^{15} + X^{16} .$$

## Timeout Check

After sending a message, the LPH waits for an acknowledgment from the receiving device. When there is no acknowledgment after a specific interval, the LPH retransmits the message. When there is no acknowledgment after a specified number of transmissions, the LPH takes whatever action is specified by the protocol.

# *Section 8*
# *ATD LINE PROTOCOL HANDLER*

The Asynchronous Terminal Driver (ATD) line protocol handler supports certain asynchronous terminals, serial printers, and certain types of asynchronous data streams.

The ATD LPH operates in five modes:

- TTY mode, which supports line-at-a-time transfer of data to or from any teletype compatible (TTY) terminal.
- Field mode, which supports field and forms processing on VIP7200, VIP7800, HDS 2, and VIP7300 class terminals.
- Block mode, which supports transfer of blocks of data to or from any VIP7800 class terminal.
- ASPI mode, which supports output to ASPI printers.
- X-ON/X-OFF mode, which supports transfer of data on any asynchronous line that utilizes an X-ON/X-OFF flow-control protocol.

The ATD LPH can be accessed at the Physical I/O, File System, or VDAM level. At the Physical I/O level, the LPH is accessed through the Request I/O ($RQIO) macro call and an associated input/output request block (IORB). This interface can be used with any mode of the LPH and provides for complete control of the selected mode.

The LPH is accessed indirectly through the File System. For example, to read input from a terminal, an application issues a Read Record macro call, supplying parameters for the call in an associated file information block (FIB). The File System translates the macro call and FIB parameters into a $RQIO macro call and associated read IORB. The File System interface is most useful in providing a sequential file interface to terminals (operating in TTY and block mode), serial printers (operating in ROP mode), and a variety of asynchronous devices (operating in X-ON/X-OFF mode). The File System interface does not support field mode.

The LPH is accessed indirectly through VDAM. For example, to read input from a terminal, an application issues a Read Form macro call, supplying parameters for the call in an associated VDAM Terminal Control Request Block (VTCRB). VDAM translates the macro call and VTCRB parameters into $RQIO monitor calls and associated IORBs. The VDAM interface is most useful in transfering data via forms from VIP7200, VIP7300, HDS 2, and VIP7800 classes of terminals operating in ATD field mode.

The remainder of this section provides:

- A summary of ATD operational modes
- A description of common functions
- A detailed description of each mode.

This section uses the term "HDS 2" terminal. The LPH software refers to this terminal as a VIP8300.

## ATD MODES

A particular mode is selected by means of a connect IORB and remains in effect until a disconnect IORB is received. The following subsections indicate the uses of each mode.

### TTY Mode

TTY mode is the default ATD operating mode. The user need not specify this mode in the connect IORB device specific word (DSW). This mode is used primarily by the File System, which treats a terminal (configured by means of the DEVICE directive) as a sequential file. In this mode, a terminal can be used as the input and output file of a task group (i.e., user-in, user-out, command-in, error-out).

TTY mode provides for line-at-a-time input and output. Character-cancel, line-delete, input-terminator, and escape key functionality is provided to aid the operator in data entry operations at the terminal. Support is also provided for a break key. (The terminal keys that represent these functionalities can be redefined by the terminal operator through the Set Terminal File Characteristic (STTY) command.) TTY mode supports a variety of asynchronous terminals including VIP7100, VIP7200, VIP7201, VIP7207, VIP7801, VIP7803, VIP7808, VIP7813, VIP7814, VIP7824, VIP7301, VIP7303, VIP7305, VIP7307, HDS 2; TWU1001, TWU1003, TWU1005; TN0300, TN1200, and other teletype (KSR, ASR) terminals.

Field Mode

Field mode allows forms-oriented processing to be performed (on certain terminals) by applications such as Display Formatting and Control (DFC), menu subsystem, and Data Entry Facility (DEF). A form consists of a series of fields. A field is a series of contiguous locations on the terminal screen into which only selected types of data can be entered. For example, a terminal operator can enter only "0" through "9" into a numeric field. The validation of data entered into a field is accomplished by ATD under application control.

Field mode allows the operator to modify entered fields easily. The break key is configurable by means of the STTY command. Break or supervisory messages are displayed in a communications region (line) on the terminal screen. Field mode processing is limited to the following terminals: VIP7200, VIP7201, VIP7207, VIP7801, VIP7808, VIP7301, VIP7303, VIP7305, VIP7307, HDS 2, VIP7813, VIP7814, and VIP7824.

Block Mode

Block mode is supported by the VIP7800 series of terminals. In block mode, the operator can locally edit terminal input without ATD involvement. Depression of the transmit key causes the LPH to receive data from the terminal in blocks of fully-edited input. Block mode can be used at either the Physical I/O or File System level.

Terminal input is locally edited by means of cursor control, character insertion/deletion, and line insertion/deletion keys. Termination of input is accomplished by depression of the transmit key. The break key is configurable by means of the STTY command. When the terminal is operating in no-roll mode, supervisory messages can be displayed in a communications region (line) on the terminal screen. Block mode processing is limited to the following terminals: VIP7801, VIP7803, VIP7808, VIP7813, VIP7814, and VIP7824.

ASPI Mode

ASPI mode supports selected serial and letter-quality receive-only printers (ROPs). This mode provides full control-byte processing; it also detects and analyzes, in some cases, printer offline conditions. ASPI mode is supported at the Physical I/O or File System level. ASPI mode is limited to the following serial printers: PRU1004, PRU7007, PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7210, and PRU7270.

### X-ON/X-OFF Mode

X-ON/X-OFF mode supports asynchronous devices that use the X-ON/X-OFF flow control protocol. This mode protects applications and devices from buffer overflow by suspending transmission when the receiver is not ready to accept data. X-ON/X-OFF mode supports a wide variety of devices, including terminals, serial printers, paper tape readers, and personal computers. X-ON/X-OFF mode is also used as a transport facility for file transfer applications. X-ON/X-OFF has four operational modes: TERMINAL, PRINTER, FILETRAN, and RAW. All operational modes require a full-duplex communications line. The maximum supported transfer rate is 9600 bits per second.

## I/O FUNCTIONS SUPPORTED BY ATD

The ATD line protocol handler supports five logical functions. Each is listed below with its associated function code (fc).

- Connect (fc = A)
- Disconnect (fc = B)
- Read (fc = 2)
- Write (fc = 1)
- Define form, field mode only (fc = 5)
- Break (fc = 9).

These functions are requested through the input/output request block (IORB). An application places in the right byte of IORB word I_CT2 the code of the desired function. A connect request establishes the mode in which subsequent functions (e.g., read, write) are performed.

## IORB PROCESSING

The ATD LPH is activated by an application-generated $RQIO macro call. Associated with this macro call is an input/output request block (IORB) that specifies the operation to be initiated. The IORB contains a function code, a buffer address, and range (in most cases), and parameters that specialize execution of the requested operation. Figure 8-1 shows a representative IORB, as required for field mode processing.

### IORB Size

The required size of an IORB depends on the mode selected by the application. Field mode requires that an extended-length IORB be used for all orders (including connect). If a standard length read or write IORB is received when the terminal is connected in field mode, that IORB is treated as a supervisory message.

The other ATD modes require standard-length IORBs. Extended IORBs can optionally be used when connecting a terminal in block mode to ascertain the terminal's type (which is returned in the extended portion of the IORB).

| WORD | LABEL | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| -3 | I_LRX | RFU | | | | EXTENDED LRN | | | | | | | | | | | |
| 0 | I_LNK | RESERVED FOR SYSTEM USE AS POINTER | | | | | | | | | | | | | | | |
| $AF | I_CT1 | RETURN STATUS | | | | | | | | T | W | U | S | 0 | R | 0 | 1 |
| 1+$AF | I_CT2 | LRN | | | | | | | | 0 | B | 0 | E | FUNCTION | | | |
| 2+$AF | I_ADR | BUFFER ADDRESS SAF 1-WORD POINTER LAF 2-WORD POINTER | | | | | | | | | | | | | | | |
| 2+2*$SAF | I_RNG | RANGE - NUMBER OF BYTES TO BE TRANSFERRED | | | | | | | | | | | | | | | |
| 3+2*$AF | I_DVS | DEVICE-SPECIFIC WORD | | | | | | | | | | | | | | | |
| 4+2*$AF | I_RSR | RESIDUAL RANGE - NUMBER OF BYTES NOT TRANSFERRED | | | | | | | | | | | | | | | |
| 5+2*$AF | I_ST | DEVICE STATUS WORD | | | | | | | | | | | | | | | |
| 6+2*$AF | I_EXT | TOTAL IORB EXTENSION LENGTH (IN WORDS) | | | | | | | | PHYSICAL EXTENSION LENGTH (IN WORDS) | | | | | | | |
| 7+2*$AF | I_DV2 | DEVICE-SPECIFIC WORD 2 | | | | | | | | | | | | | | | |
| 8+2*$AF | I_FCS | TOTAL KEYSTROKES | | | | | | | | | | | | | | | |
| 9+2*$AF | I_HDR | READ OFFSET | | | | | | | | | | | | | | | |
| 10+2*$AF | I_ST2 | FIELD MODIFICATION INDICATOR | | | | | | | | | | | | | | | |
| 11+2*$AF | I_QDP | DEVICE_ID; RELATIVE RESIDUAL RANGE | | | | | | | | | | | | | | | |
| 12+2*$AF | I_TAB | EDIT OFFSET (INPUT); TERMINATION CHARACTER (INPUT) | | | | | | | | | | | | | | | |
| 13+2*$AF | I_CON | ABSOLUTE ADDRESS INDICATOR; PRE-ORDER READ AND WRITE CODE; TERMINATION CHARACTERS; VFN VALUE | | | | | | | | | | | | | | | |
| 14+2*$AF | I_LOG | START OF FIELD ATTRIBUTE TABLE | | | | | | | | | | | | | | | |

Figure 8-1. ATD IORB

## Expanded LRN Support

LRNs 0 to 252 and 256 to 4095 are supported by ATD. If the LRN is in the range of 256 to 4095, the extended (negative direction) IORB format must be used.

## IORB Device-Specific Word

The device-specific word I_DVS is used in conjunction with each of the I/O functions. This word serves to modify the activity of a particular function. For example, the setting of bit 15 in I_DVS determines whether the communication line is disconnected on completion of a disconnect function.

## Processing Order of IORBs

An application can issue one I/O order against a terminal (or line) and wait for its completion, or issue several IORBs. Outstanding read and write orders and non-abortive disconnects are queued sequentially. In TTY, field, and block mode, write orders are processed before read orders if the read order is not in progress. Define form orders, read and write orders with the option to purge outstanding I/O requests, and abortive disconnects are executed immediately after being received by the LPH.

## Purging Queued IORBs

In the following cases, the LPH purges queued IORBs and posts the incomplete orders back to the requesting application:

1. The application issues a disconnect order with an abort request (purge IORB indicator in I_DVS word of IORB is set to 0). All read and write orders that are active or queued at the time of the disconnect order are purged and posted to the issuing task with a "device unavailable" (010B) return status.

2. A line disconnect (data set status change) occurs. All active or queued read and write orders are purged and posted with a "device unavailable" return status. Both the line and station are disconnected.

3. The application issues a purge-all order in field mode. All active or queued read and write orders are purged and posted to the issuing task with a "device unavailable" return status. Both the line and station remain connected.

4. A break signal is detected (BREAK key pressed) and the user has previously issued a read-break IORB (i.e., function code 9 in I_CT2, and bit 0 in I_DVS set to 0). See "Break Processing by ATD LPH" below.

5. The application issues a block write order with the purge option. Active or queued write orders are purged or posted with "device unavailable" return status. Both the line and station remain connected.

6. The application issues a block read order with the purge option. Active or queued read orders are purged or posted with "device unavailable" return status. Both the line and station remain connected.

## IORB Error Processing

All ATD modes report errors in the same manner. A 2-byte error code is placed in register R1. The left byte indicates the component detecting the error; the right byte indicates the error itself. The right byte is also placed in IORB field I_CT1. Table 8-1 lists the return codes as they appear in the left byte of I_CT1.

Table 8-1. ATD Return Codes

| Status Byte | Meaning |
|---|---|
| 0 | No error; operation complete |
| 1 | Request block is already busy |
| 2 | Invalid LRN |
| 3 | Illegal wait |
| 4 | Invalid argument(s):<br>● Improper set-up of IORB<br>● Improper buffer size<br>● Improper set-up of data in certain buffers |
| 5 | Device not ready. Reported when the following devices are in an off-line state: TWU1001, 1003, 1005; PRU7070, 7075; and serial printer attached to VIP7800 terminal |
| 6 | Timeout on order other than connect |
| 7 | Hardware error:<br>● Parity error (block mode, X-ON/X-OFF mode)<br>● Framing error<br>● Data lost-buffer overflow (X-ON/X-OFF mode)<br>● Data service error (receive overrun)<br>● Communications control block service error<br>● Fatal MLC error |
| 8 | Device disabled<br>● Connect or disconnect pending<br>● Device logically disabled by system |
| A | Controller unavailable |

Table 8-1 (cont). ATD Return Codes

| Status Byte | Meaning |
|---|---|
| B | Device unavailable<br>● Read/write IORBs purged by purge option<br>● Read/write IORBs purged by disconnect<br>● Read/write IORBS purged by disconnect with queue abort<br>● Attempt made to connect to a 7800 class terminal that is in local mode |
| C | Inconsistent or illogical request<br>● Connect order issued against a device that is currently connected<br>● Disconnect order issued against a device that is currently disconnected<br>● Read/write IORB issued; line not connected<br>● Connect order issued to VIP7800 attached printer when terminal has already been connected in field mode<br>● Field mode connect order issued to VIP7800 terminal when attached printer has already been connected<br>● Field mode read issued before define form request<br>● Read request outstanding when new define form request issued<br>● Block missed on block mode read |
| F | End of file detected (X-ON/X-OFF mode) |
| 10 | Timeout on connect |
| 34 | The mode of ATD is not configured |
| 35 | The mode of ATD is not configured for this controller |
| 38 | Sub-LRN not configured |
| 39 | Logical connect failed |
| 3F | Connect or disconnect in progress |

The status word (I_ST) of the IORB contains additional information that qualifies the major status code returned in I_CT1. The significance of certain bits of the status word is the same for all ATD modes. Table 8-2 shows the meaning of these bits.

For the significance of mode specific bit settings, refer to the descriptions of the individual ATD modes found later in this section.

Table 8-2. Status Word of IORB (I_ST)

| I_ST Bit | Meaning When Bit Set to 1 |
|---|---|
| 0 | Abort terminated IORB |
| 1 | Mode specific |
| 2 | Data service rate error (receive overrun) |
| 3 | Mode specific |
| 4 | Communication control block service error |
| 5 | IORB purged because of break signal |
| 6 | Mode specific |
| 7 | Mode specific |
| 8 | Framing error |
| 9 | Parity error |
| A | Nonzero residual range (read only) |
| B | Phone hang-up on disconnect |
| C | Mode specific |
| D | Mode specific |
| E | Mode specific |
| F | Fatal error<br>● Unrecoverable memory error<br>● Bus parity error<br>● Non-existent resource error |

## Return of Device ID

Table 8-3 shows the values returned in the right byte of IORB field I_QDP when an extended length connect IORB completes and is posted back to the application.

Table 8-3. Device IDs Returned in IORB

| Value in I_QDP | Marketing Identifer |
|---|---|
| 45 | VIP7100 |
| 46 | VIP7200 |
| 47 | VIP7207 (data entry) |
| 48 | VIP7808 (word processing mode) |
| 49 | VIP7801 |
| 4A | VIP7808 (general purpose mode) |
| 4B | VIP7803 (word processing) |
| 4C | TTY (KSR) |
| 4D | TermiNet 0300 |
| 4E | TermiNet 1200 |
| 50 | VIP7813/VIP7824/VIP7825 (MF) async |
| 51 | ROSY 24 |
| 52 | ROSY 26 |
| 54 | Spinwriter 5518 |
| 55 | Spinwriter 3508 |
| 56 | PRU7200 (Qume 96 characters) |
| 57 | Sara 22 |
| 58 | PRU7210 (Qume 130 characters) |
| 5B | PRU7070 |
| 5C | PRU7170 |
| 5D | PRU7270 |
| 5E | PRU7175 |
| 5F | PRU7075 |
| 61 | VIP7814 sync (rev. 2 firmware) |
| 62 | VIP7814/VIP7815 async |
| 63 | VIP7201 |
| 64 | VIP7301 |
| 65 | VIP7307 (data entry) |
| 66 | VIP7303 (word processing) |
| 67 | VIP7305 (multifunction) or microSystem 6/10 console (black and white) |
| 90 | NIP (ASPI non-impact printers) |
| A0 | HDS 2 (in native VIP8300 mode) |
| A1 | VIP7306 |

## SUPERVISORY MESSAGE PROCESSING

When a terminal is processing forms, the supervisory message line provides a communication region (typically the bottom line of the terminal) through which the operator can interact with the system independently of the forms processing application. ATD provides support for supervisory messages on the following VIP terminals when they are connected in either field or block mode: VIP7200, VIP7201, VIP7207, VIP7801, VIP7803, VIP7808, VIP7813, VIP7814, VIP7824, VIP7301, VIP7303, VIP7307, and HDS 2.

Supervisory message processing is specified by means of a non-extended read or write IORB with bit 9 in I_DVS set to 1. The use of this bit is optional in ATD field mode, because supervisory message orders are already distinguished from normal field mode orders by being non-extended. The location of the supervisory message line depends on the ATD mode, and the type and operational mode of the terminal. When, for example, a terminal is connected in field mode, it operates in no-roll mode. If the terminal is a VIP7200, then the supervisory message line is (typically) the 24th line. If, however, the terminal is a VIP7801, then the supervisory message line is always the 25th line of the terminal.

The following diagram shows supervisory message line location for supported VIP terminal classes and ATD LPH modes.

| | VIP Terminal Class | | | | |
|---|---|---|---|---|---|
| | 7100 | 7200 | 7300 | 7800 | HDS 2 |
| TTY mode | 1 | 1 | 1 | 1 | 1 |
| Field mode | N/A | 2 | 3 | 3 | 3 |
| Block mode | N/A | N/A | N/A | 4 | N/A |
| X-ON/X-OFF mode | N/A | 1 | 1 | 1 | 1 |

where:

1 = When supervisory messages are written to the terminal, output is at current cursor position. There is no way to acknowledge the write; all reads are treated as normal device reads.

2 = Read/write activity is directed to the designated supervisory message line, which is normally line 24.

3 = Read/write activity is directed to the line 25.

4 = Read/write operation is predicated on the roll bit (bit 9) of the connect I_DVS. If the terminal is in roll mode (bit 9 = 0), writes begin at the current cursor position and reads are treated as normal device reads. If the terminal is not in roll mode (bit 9 = 1), reads and writes are directed to the 25th line.

All writes to the supervisory message line are truncated to 80 characters, and the residual range indicates the amount of data not written. If supervisory message writes are specified, bit 8 of the read/write I_DVS becomes significant. If bit 8 = 0, supervisory messages must be acknowledged before the write is posted back to the application. If bit 8 = 1, supervisory messages need not be acknowledged by the operator.

In TTY and X-ON/X-OFF modes, supervisory writes (which are treated as standard data writes) are not acknowledged. In other modes, the operator acknowledges a supervisory message by pressing one of the following keys:

| | |
|---|---|
| Field mode: | function key 10, CLEAR key, or transmit key. |
| Block mode: | function key 10. |

## CONTROL BYTE PROCESSING

Control byte processing is a TTY, block, ASPI, and X-ON/X-OFF mode option that is specified by a bit setting in the I_DVS word of the write order. When selected, this option indicates that the first byte of the output buffer is to be used as a control byte. This byte must be included in the range (I_RNG) value of the write IORB.

The format of the control byte is:

| Bit: | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| Field: | Y | P | P | V | COUNT | | | |

The possible values for these fields and their significance to ATD are shown in Table 8-3.1.

The head-of-form sequence, specified by bit 3 of the control byte, is a form feed for the following devices: TWU1001, TWU1003, TWU1005; PRU7070, PRU7075, PRU7270, PRU7170, PRU7175. These are stand-alone devices not attached to a VAF7821 buffered printer adapter. For other devices, head-of-form consists of a carriage return and three line feeds.

Table 8-3.1  ATD Control Bytes

| Bit Value | Field Name | Description |
|---|---|---|
| 80 | (Y) | 1 - Do not do post-order C/R L/F |
| 60 | (PP) | Bits<br>5 6<br>0 0 - Print; ignore V and CCCC fields<br>0 1 - Do not print; do V and CCCC fields<br>1 0 - Print; do V and CCCC fields<br>1 1 - Reserved for future use |
| 10 | (V) | 1 - If CCCC=0, send form feed or three line feeds for terminals; if CCCC>0, send one line feed<br><br>0 - Pre-space as per CCCC field |
| 0F | (CCCC) | Count field (number of line feeds to send) |

## 8-BIT DATA SUPPORT

ATD supports 8-bit data mode. You can select either an 8-bit data path or the appropriate algorithms used when an application uses 8-bit data on a 7-bit communication channel. The number of data bits that can be transmitted on a communications channel is specified at system build time using the ATD CLM directive. If the communication channel does not support 8-bit data, you can specify truncation or shift-in/shift-out. This is accomplished by the compression algorithm argument. If truncation is selected, the transmitted data will have the high-order bit removed. If shift-in/shift-out is specified, a shift-in character (hex 0F) will preceed the first character to be transmitted that has a high-order bit equal to one. Before the next character in the data stream that has a high-order bit equal to zero is sent, a shift-out character (hex 0E) will be sent. This will continue with a shift-in or shift-out character preceeding the actual data character every time the high-order bit changes. In this way, 8-bit data is sent over a 7-bit communication channel. For example if the formula "1/4 + 1/2 = 3/4" were to be sent, it would be represented by the hex string:

```
SI 1/4 SO    +     SI 1/2 SO    =     SI 3/4

0F 3C 0E 20 2B 20 0F 3D 0E 20 3D 20 0F 3E
```

Shift-in/shift-out is supported only on MLC-16 controllers.

## CONNECT PROCESSING

After a connect IORB is processed by the communication supervisor, the IORB is passed to ATD. The following is a list of actions taken by ATD for all connects:

1. Make sure the channel control program (CCP) is loaded.

2. If a sub-LRN is used, determine if a connect for another component is in progress. If so, defer the connect until later.

3. If the device is a hard copy device or TTY terminal, and it is not a configured device type, go to Step 8.

4. Send an inquiry sequence to the terminal. If there is no response, retry for 4.5 minutes at regular intervals. If no error occurs, go to Step 6.

5. Post the current IORB with device unavailable.

6. If the device is not self configuring, go to Step 7; otherwise validate the inquiry response. If the response is a valid (supported) terminal, set the device type and go to Step 8. If not a valid supported terminal, set the default device type to VIP7200 and go to Step 8.

7. Set the terminal type as defined by CLM.

8. Configure the terminal characteristics.

9. Validate the mode required and enter the correct ATD module.

## SELF-CONFIGURING TERMINALS

The device type can be defined as self-configuring by using an asterisk (*) as the device type argument of the ATD directive at configuration time. In this way, the ATD line protocol handler will configure the device when the first connect is recieved for the device. Self-configuration is performed by sending an inquiry sequence to the device and setting the configuration based on the response. If an unknown response is received, the device is configured as a VIP7200. If the device doesn't respond, the sequence is retried at regular intervals for approximately 4.5 minutes. If after this time the device doesn't respond, the connect is returned to the user with the device unavailable error (010B).

The following restrictions apply to self-configuring terminals:

- The device must be either a VIP780x, VIP781x, VIP782x, VIP730x, HDS 2, VIP7201, VIP7200, or any ASPI printer.

- The device must be configured at a baud rate that is suitable for all devices connected to the channel.

- Applications that determine the device type by issuing a Get Device Information monitor call should be aware that prior to a connect being issued, the device type will be x'FF' and a valid device type will only be returned after a connection has been established.

- A self-configuring line will not re-establish a new device on that line unless a physical disconnect or a line drop is detected on a previous session. To maintain the terminal in the same configuration, only logical disconnects and connects can be requested in any session.

## BUFFERED PRINTER ADAPTER (BPA) SUPPORT

ATD supports the buffered printer adpater (BPA). The BPA allows the attachment of a serial printer (PRU1003, PRU1005, PRU7070, PRU7075, PRU7170, PRU7175, or PRU7270) to a VIP7801, VIP7803, VIP7808, VIP7813, VIP7814, VIP7824, VIP7816, or VIP7856 terminal. An application can use the serial printer when the attached terminal is connected and operating in either TTY or block mode. Use of the printer with a terminal connected in field mode is not allowed.

The BPA can be accessed at the physical I/O or File System level. It must be configured with the BPA directive. If accessed through the File System, the BPA directive must be paired with a DEVICE directive specifying a ROP device unit.

Before issuing write orders to the BPA, the application must first establish a connection to it. To use the BPA at the physical I/O level, the application issues I/O orders to the work station with a single LRN that refers to the terminal display/keyboard and the BPA. A sub-LRN specified in I_ST differentiates between orders directed to the terminal display/keyboard and to the BPA. A sub-LRN of 0 refers to the display /keyboard; a sub-LRN of 1 refers to the BPA. *

When the attached printer is servicing a write order, the terminal keyboard is locked. *

## BREAK PROCESSING BY ATD LPH

In TTY, field, X-ON/X-OFF, and block mode, break processing is initiated when the terminal's BREAK (BRK) key is pressed. Results differ, depending on whether the task issued a read-break I/O order request for that terminal.

### Break Processing with Read Break Request

A task issues a read break request when the IORB specifies a function code value of 9 in I_CT2 and a value of 1 in bit 0 of I_DVS. I_ADR of the IORB must have a null address.

The communications supervisor queues read break requests on a last-in, first-out basis.

When the terminal's break key is pressed, and a read break request has been issued, the terminal is now in "break mode" for subsequent I/O requests. Break processing proceeds as follows:

1. When a write order is active, and:

   a. Bit 7 in I_DVS of the write IORB is 1, the order completes normally; break processing then begins with step 2 below

   b. Bit 7 in I_DVS of the write IORB is 0, or when a read order is active, either order is terminated and posted to the issuing task with IORB settings shown in step 2.

2. All other queued read and/or write IORBs are posted back to their respective tasks with:

   a. I_RSR containing the range value specified in I_RNG

   b. Bits 5 and 10 of I_ST set to 1

   c. Left byte (status) in I_CT1 has value of 0.

3. The last (last-in, first-out) read break request is posted to the issuing task with:

   a. Bit 5 of I_ST set to 1

   b. Left byte (status) of I_CT1 has value of 0.

4. Read and write orders issued by the "broken task" (i.e., task in break mode) are posted back (without execution) with IORB values described in step 2 above.

5. Read and write orders from tasks not in break mode (i.e., that did not issue receive-break requests) are accepted and executed.

Break mode remains in effect until a task issues another read break request or a cancel break request (i.e., until provision has been made for processing the next break signal). A task issuing another read break request to a device which is in break mode is indicating that it wishes to be the task notified of the next break. A task issuing a cancel break request to a device which is in break mode is indicating that it does not wish to be the task notified of the next break; the task to be notified of the next break is the one that issued the most recent read break order.

A cancel break request is specified with an IORB having a function code of 9 in I_CT2 and bit 0 of I_DVS set to 1. A cancel break request causes one or all queued read break IORBs to be posted back to their issuing tasks. If bit 1 of I_DVS is 0, the request specifies the cancellation of only the most recently issued read break request. If bit 1 of I_DVS is 1, the request specifies the cancellation of all active and queued read break requests. The cancel break IORB and purged read break IORB(s) are posted back to their issuing tasks with:

Bit 5 in I_ST1 set to 0
Left byte (status) in I_CT1 set to 0.

<u>Break Processing with No Read Break Request</u>

When a break signal is received and no read break request has been issued, only the current active order is affected. The break signal is processed as follows:

1. If there is no active order, the break signal is ignored.

2. When a read order is active, the order is terminated and posted to the issuing task with:

   a. I_RSR containing the range value specified in I_RNG less the number of characters entered. In field mode, I_QDP is also updated.
   b. Bits 5 and 10 of I_ST set to 1
   c. Left byte of I_CT1 set to 0.

3. When a write order is active and bit 7 in I_DVS is 1, the break signal is ignored and the write order completes normally.

4. When a write order is active and bit 7 in I_DVS is 0, the order is posted to the issuing task with:

   a. I_RSR containing the range value specified in I_RNG
   b. Bits 5 and 10 of I_ST set to 1
   c. Left byte of I_CTl set to 0.

## PRINT SCREEN

ATD supports a print screen facility that is activated by invoking the PRTSCN bound unit. In order to invoke PRTSCN, an LDBU directive to load BOXQUE must be included in your CLM file. The default key to activate print screen is the shifted function 4 key, which can be changed to any two character escape sequence key with the STTY command. The print screen facility can be used on VIP7300, HDS 2, and VIP7800 series terminals as well as the VIP7201. The message "SCREEN COPY IN PROGRESS" is displayed at the start and "SCREEN COPY SUCESSFUL" on the 25th line at the termination of the facility. These messages are suppressed on the VIP7201 terminal, which does not support the 25th line. Print screen is restricted when a terminal is in the break state.

## TTY MODE

The TTY mode of ATD provides for line-at-a-time transfer of data to or from teletype-compatible asynchronous terminals.

TTY mode supports six functions:

- Connect
- Disconnect
- Read
- Write
- Break
- Wait-On-Line

These functions are requested through standard-length IORBs. An application can optionally use an extended IORB for a connect operation.

A connect order establishes the mode in which the connected terminal operates. Because TTY is the default mode of the ATD LPH, an application need not explicity specify the mode in the device-specific word (I_DVS) of the connect IORB.

### TTY Mode and Extended Character Set (8-Bit Data)

In TTY mode, ATD passes data through and does not alter the data in any way. The following is a list of actions that ATD TTY mode takes for the extended character set. Refer to Table D-1 for a description of the extended character set.

- C0 character set handling is as described in the Read/Write functions of TTY mode.
- The G0, G1, and C1 character sets are passed through.
- The Line Termination character must be within the C0 or G0 character sets. The ESC sequence must also be within the G0 character set.
- The Line Cancel, Character Delete, Break Character, and Print Key are handled the same as the Line Termination character above.

Connect Function (TTY Mode)

The following paragraphs describe the options that an application can specify with a connect order.

The Auto Call option, which is supported by all system-supplied LPHs, is described in Section 7. This option enables an application to establish a connection with an 801-A or 801-C ACU data set.

The default IORB setting for the BELL option allows the output of bells to a terminal. If the option is not specified, the output of bells to a terminal is suppressed, even under error conditons.

When the terminal is a VIP7801, VIP7803, VIP7808, VIP7813, VIP7814, VIP7824, or VIP7816/26, specification of character mode (which is the default) causes the terminal to be physically configured in character mode with the echoplex and roll options set.

When the buffered option is selected, a VIP7800 class terminal is configured in text mode with the no-echoplex and no-roll options set. This means that data entered at the terminal is not transmitted (to the LPH) until the transmit key is depressed. Prior to pressing the transmit key, the operator can edit information displayed on the terminal by means of the cursor control and erase keys. When ATD receives and processes the transmitted data, the LPH acts on <u>any</u> line cancel or character delete sequence encountered in the data stream. That is, the LPH does not accept as data the @, \, or CTL-X characters. This point bears emphasis; the operator of a buffered terminal who uses the cursor-back key to erase a character might well forget that pressing the @ key has the same effect. If the operator mistakenly enters the @ character as data, the LPH deletes the next character when data is ultimately transmitted from the terminal. Care must be exercised when entering teletype control sequences from a buffered terminal.

Connect IORB (TTY Mode)

This subsection summarizes the bit settings that govern the connect options already described.

Table 8-4 shows bits of the connect I_DVS word that are applicable to TTY mode. All other bits must be zero.

The bit settings in word I_ST are signficant when a serial printer is attached to the terminal by means of a buffered printer adapter. On connect orders, the field specifies whether the terminal or attached printer is being addressed. The permitted values are:

0 = Terminal
1 = Attached serial printer.

Table 8-4. I_DVS Word in Connect IORB (TTY Mode)

| Bit Number | Meaning for Connect Function |
|---|---|
| 2 | 0 = Do not use auto dial<br>1 = Use auto dial |
| 3 | 0 = Allow output of bells to the terminal<br>1 = Supress output of bells to the terminal |
| 13 | 0 = Character mode<br>1 = Buffered mode |

## Disconnect Function (TTY Mode)

An application uses the disconnect IORB to terminate TTY mode processing. The following paragraphs describe the options that an application can specify with a disconnect order.

If the abort queued orders option is specified, outstanding IORBs (active and queued) are terminated with a "device unavailable" status (010B). The disconnect order is immediately serviced. If the abort order is not specified, all outstanding IORBs are allowed to complete before the disconnect order is serviced.

If the hang-up option is selected, the terminal is physically disconnected when the disconnect order is serviced. If the hang-up option is not specified, the communications connection remains active after servicing of the disconnect order (i.e., the terminal is logically disconnected, but remains physically connected).

## Disconnect IORB (TTY Mode)

This subsection summarizes the IORB bit settings that govern the disconnect options just described.

Table 8-5 shows bits of the disconnect IORB that are applicable to the TTY mode of ATD. All other bits must be zero.

The bit settings in word I_ST are signficant when a serial printer is attached to the terminal by means of a buffered printer adapter. On disconnect orders, the field specifies whether the terminal or printer is being addressed. The permitted values are:

0 = Terminal
1 = Attached serial printer.

Table 8-5. I_DVS Word in Disconnect IORB (TTY Mode)

| Bit Number | Meaning for Disconnect Function |
|---|---|
| 14 | 0 = Abort outstanding requests<br>1 = Wait until outstanding requests complete before disconnecting terminal |
| 15 | 0 = Hang-up the phone<br>1 = Do not hang-up the phone |

## Read Function (TTY Mode)

The following TTY mode read functions support the entry of data by the terminal operator. They are activated by pressing terminal keys. In some cases, an application can designate the key that activates a particular function by means of the Set Terminal File Characteristics (STTY) command. These functions are not controllable through the IORB. The read IORB is used to pass data to the application once it has been entered and edited by the operator.

### OPERATOR FUNCTIONS

TTY mode functions that support data entry operations are the following:

| Function | Action |
|---|---|
| Character delete | Delete a previously entered character |
| Line cancel | Cancel the current line of input |
| Hide | Accept the next character as data (i.e., do do not interpret it as a control character) |
| Terminate read | Signal completion of the current read order |
| Break | Generate break signal to application controlling the terminal |

#### Operator Function Keys

The LPH performs one of the functions just listed when the operator keys the appropriate code sequence. Typically, the depression of a single terminal key will generate the proper code sequence. For example, on a VIP7301 terminal, depression of the cursor-left key causes the generation of the code sequence 1B44, which causes the LPH to delete the prior character.

The code sequence that initiates a function is determined by the device-type parameter of the ATD directive. That code sequence can later be altered by the STTY command. Table 8-6 shows the initial (default) codes associated with device-types that can be specified with the ATD directive.

Table 8-6. Default Values of Special Characters by Device Type

| Device Type | Character Delete | | Line Cancel | | Line Break | | Read Terminator | |
|---|---|---|---|---|---|---|---|---|
| | Hex | Key-Cap | Hex | Key-Cap | Hex | Key-Cap | Hex | Key-Cap |
| HDS 2 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7200 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7201 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7207 | 1B44 | <- | 1B60 | CLEAR | 00 | BREAK | 0D | RETURN |
| VIP7301 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7303 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7305 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7306 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7307 | 1B44 | <- | 1B60 | CLEAR | 00 | BREAK | 0D | ENTER |
| VIP7801 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7803 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7813 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7814 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7808 | 1B44 | <- | 1B4B | ERASE | 00 | BREAK | 0D | RETURN |
| VIP7100 | 40 | @ | 18 | CTL-X | 00 | BREAK | 0D | RETURN |
| TWU1001 | 40 | @ | 18 | CTL-X | 00 | BREAK | 0D | RETURN |
| TWU1003 | 40 | @ | 18 | CTL-X | 00 | BREAK | 0D | RETURN |
| TWU1005 | 40 | @ | 18 | CTL-X | 00 | BREAK | 0D | RETURN |
| TN 0300 | 40 | @ | 18 | CTL-X | 00 | BREAK | 0D | RETURN |
| TN 1200 | 40 | @ | 18 | CTL-X | 00 | BREAK | 0D | RETURN |
| TTY | 40 | @ | 18 | CTL-X | 00 | BREAK | 0D | RETURN |

## Character Delete and Line Cancel

As the preceding table indicates, the operator can delete characters and cancel lines in two different ways, depending upon the device type. On some devices, referred to in this context as hard copy terminals, deleting a character requires depressing the @ key. On other devices, the cursor back (<-) key is used; these devices are called video terminals.

On hard copy terminals, cancelling a line is accomplished by depressing and holding the CTL key and pressing X. On video terminals, the operator uses the ERASE or CLEAR key.

On hard copy and video terminals, editing is performed by different actions, and different information is displayed at the terminal during the editing operation. However, the modification of buffer contents and the information returned in the IORB is the same. The following paragraphs explain in detail the procedure and process of editing on each type of terminal.

<u>Character Deletion on Hard Copy Terminals</u>. Character deletion is performed on the current line (i.e., before the carriage return key is pressed). Pressing the @ key deletes the character immediately preceding the @ character, and, if echo was requested, displays the @ character. Each succeeding @ entry deletes another character, from right to left, up to the beginning of the line.

The I_RSR value in the issuing program's IORB indirectly reflects the number of characters accepted at the time the order was terminated. For example, if the operator enters AXC@@B followed by a carriage return, the I_RSR value shows that only two characters (A and B) were entered. Note that pressing the @ key does not actually delete a character, but moves back by one character position a pointer in the read buffer. In the example just given, X is overwritten by B, but C (though rejected by the operator and not reflected in the I_RSR value) is present in the buffer, following B.

<u>Line Cancellation on Hard Copy Terminals</u>. To cancel the current line (before carriage return is entered), the operator depresses and holds the CTL (control) key and presses X. This action deletes the current line, displays the *DEL* message on the next line. The LPH reissues the read order, using the original buffer and range. Line cancellation does not clear the buffer of characters entered into the buffer before the line cancellation action.

<u>Character Deletion on Video Terminals</u>. Pressing the cursor-left (<-) key erases from the screen the character last entered, and removes it from the associated read buffer. When the completed read IORB is posted to the issuing application, I_RSR indirectly reflects the number of characters accepted when the order was terminated. For example, if the operator enters ABC<-, I_RSR shows only that two characters (A and B) were entered. Again, as with character deletion on hard copy terminals, extraneous information may appear in the rest of the buffer.

<u>Line Cancellation on Video Terminals</u>. The key used is either ERASE or CLEAR, depending on the device type (see Table 8-6). The effect is to erase all characters on the current line and to reposition the cursor to the beginning of the erased line. The LPH reissues the read order, using the original buffer and range. Line cancellation does not clear the buffer of characters entered into the buffer before the line cancellation action.

## Read Termination

The operator can terminate a read order in one of three ways.

1. Press the transmit key.

2. Press the user-selectable read-termination key. The carriage return key is the default termination key on both hard-copy and video terminals. The operator can designate another key by means of the STTY command. The terminating character (generated by carriage return or a user-designated key) is not stored in the buffer; the LPH optionally echoes a carriage return and/or line feed to the terminal.

3. Generate a two- or three-character escape sequence. Any terminal function key or cursor control key generates a two- or three-character escape sequence. This sequence can be used to terminate a read operation, provided that it has not previously been designated for line cancel, character delete, or break operations. ATD stores the terminating sequence in the read buffer and optionally echoes a carriage return and/or line feed, as appropriate. The read IORB is posted back to the application.

## Break

The break key provides an interruption or attention signal to the system software. After detecting a break, the LPH may terminate write orders and read orders. For a detailed description of break functionality, see "Break Processing with Read Break Request" earlier in this section.

The break key can be changed by means of the STTY command.

## Hide Function

The hide function allows the operator to enter a character (such as @, carriage return, and cursor-left) that the LPH would otherwise interpret as a control character. The hide function key is a control P (x'10'). The operator keys a control P immediately before the character to be entered as data. The LPH interprets the control P as an escape character (i.e., does not place the control P in the buffer) and echoes a backslash, if echo was requested. The LPH then stores the next character in the buffer without interpretation, echoing it if echo was requested. If the hidden character (immediately following the control P) is not printable, it is still stored in the buffer, but a period (.) is echoed to the terminal.

The control P key is used for the hide function on hard-copy and video terminals. The hide function key cannot be changed by the STTY command.

## READ ORDER FUNCTIONALITY

The following options, unlike those just described, are not under direct control of the operator. Instead, they are specified by the application in an IORB.

### Echo

If the echo option is selected, any keyed input is echoed, or "reflected" back to the terminal. If echo is not selected, keyed input will not be echoed and the cursor will not move as the operator enters data at the terminal.

### Line Feed

If this post order option is selected by the application, a line feed is sent to the terminal upon completion of a read order. A line feed is not echoed if the read IORB specified the no echo or the no line feed option.

### Carriage Return

If this post order option is selected by the application, a carriage return is sent to the terminal upon completion of a read order. A carriage return is not echoed if the read IORB specifies the no echo or the no carriage return option.

## READ IORB (TTY MODE)

An application specifies the options just described by setting bits in the IORB word I_DVS. Table 8-7 gives the individual significance of these bits. All other bits must be zero.

Table 8-7. ATD Word I_DVS in TTY Mode Read IORB

| Bit Number | Meaning for Field Read Function |
|---|---|
| 10 | 0 = Do not echo input or move the cursor<br>1 = Echo input; move cursor |
| 11 | 0 = Do not send post-order line feed<br>1 = Send post-order line feed |
| 12 | 0 = Send post-order carriage return<br>1 = Do not send post-order carriage return |

Write Function (TTY Mode)

The following options are specified by an application in the write IORB.

OFF LINE

If the off-line option is specified, the LPH detects and reports a device-not-ready condition (0105) when a TWU1003 or 1005 is disconnected or non-operational. It is recommended that these devices be configured (through the use of the STTY function) for non-buffered file system output. This configuration ensures that offline conditions can be properly processed by the system. If the off-line option is not specified, ATD does not detect or report off-line conditions.

CONTROL BYTE PROCESSING

If specified, the control byte option indicates that the first byte in the output buffer is to be used for pre-order control. A control byte must be included in the range (I_RNG) of data to be transmitted. For a detailed description of this option, including control byte format, see "Control Byte Processing" earlier in this section.

QUIT ON BREAK

If this option is specified, a break signal can interrupt the execution of the write order. Otherwise, a break signal cannot be used to prematurely terminate an active write order.

CARRIAGE RETURN

If the carriage return option is specified, a carriage return is sent to the terminal after the completion of the write order.

LINE FEED

If this option is specified, a line feed is sent to the terminal after the completion of the write order.

Write IORB (TTY Mode)

This subsection summarizes the bit settings that govern TTY mode write options.

Table 8-8 gives the significance of the bits in the IORB word I_DVS that are applicable to TTY mode ATD. All other bits must be zero. Bit 13 is significant only for devices configured as PRU1005/TWU1005, TN300, TN1200, or through the BPA. If this bit is set, bit 11 should be set to zero.

Table 8-8. ATD Word I_DVS in TTY Mode Write IORB

| Bit Number | Meaning for TTY Write Function |
|---|---|
| 2 | 0 = Do not check for TWU1003, TWU1005 offline conditions<br>1 = Check for TWU1003, TWU1005 offline conditions |
| 4 | 0 = Include control byte<br>1 = Do not include control byte |
| 7 | 0 = Stop output on detection of a break<br>1 = Do not stop output on detection of a break |
| 11 | 0 = Do not send post-order line feed<br>1 = Send post-order line feed |
| 12 | 0 = Send post-order carriage return<br>1 = Do not send post-order carriage return |
| 13 | 0 = Control byte is defined for terminal printers<br>1 = Control byte is defined for ASPI printers |

Bit settings in word I_ST are signficant when a serial printer is attached to the terminal by means of a VIP7800 buffered printer adapter. On write orders, the field specifies whether the terminal or printer is being addressed. The permitted values are:

0 = Terminal
1 = Attached serial printer

## Device Configuration (TTY Mode)

Hardware switches on a device connected in TTY mode should be set in the following positions. (The device may not support all of the switches mentioned below). TTY (character) mode:

CHARACTER/BUFFER switch in CHARACTER position
DUPLEX HALF/FULL SWITCH in FULL positon
LOCAL COPY/ECHO switch set as required by user (normally set to echo)
Speed configured between 110 and 9600 bits per second
ROLL/NO ROLL switch set to ROLL

## Error Processing

When a parity error is detected in keystroke input, an audible alarm sounds and the typed character is ignored. When the read order is posted, the return status in I_ST indicates detection of parity error(s) (bit 9 = 1).

If a framing error or receive overrun conditon is detected, the read order terminates and a hardware error (0107) is returned; I_ST indicates the specific reason for abnormal termination.

## TTY Mode Timeout Processing

Timeouts may occur during the processing of read orders. A timeout occurs when the operator does not terminate the input operation within 5 minutes after entering the first character. There is no timeout if the operator does not enter any characters. The threshold of this timeout (5 minutes) can be changed via the read parameter of the CLM directive TIMEOUT.

## FIELD MODE

The field mode of ATD allows an application to process a set of fields, commonly called a form. In this mode, each field that an operator keys into the form is validated by the ATD LPH and is passed to the application, one field at a time. This mode should not be used if the terminal itself is performing (local) field validation and forms processing. The concepts of forms, fields, subfields, and field validation are defined below.

## Field Mode and Extended Character Set (8-Bit Data)

The following is a list of actions that ATD field mode takes for the extended character set. Refer to Table D-1 for a description of the extended character set.

- The C0 and G0 character set handling is as described in the Read, Write, and Define Form functions of field mode (described later in this subsection).
- Any character in the C1 character set terminates a read.
- Characters in the range X'A0' through X'BF' inclusive have no character validation.
- Characters in the range X'C0' through X'FF', excluding X'D7' and X'F7' have alphabetic validation.
- Characters X'D7' and X'F7' have no validation.

## Forms, Fields, and Subfields

A field is a series of contiguous locations into which meaningful data can be entered. A subfield is a portion of a field (less than or equal to the field size) that accepts data only in accordance with the definition of the subfield. There are no limits on the number of fields that a form may contain. Each field may contain one to nine subfields. A field may not be longer than 80 characters and may not extend over one line (row) of the terminal display area.

An example of the relationship between field and subfield is an 8-character alphanumberic employee ID consisting of a 5-character employee number and a 3-character department designator. The first subfield would be defined as 5-digit characters and the second subfield as 3-alphabetic characters.

## INPUT VALIDATION

The input to a subfield is validated by reference to a field attribute descriptor. A subfield descriptor must specify one of the following validation/edit attributes:

- Digit (0-9)
- Numeric (0-9, decimal point, minus sign, plus sign, comma)
- Alphabetic (A-Z, a-z, period, space, comma, hyphen, apostrophe)
- Alphanumeric (all numeric and alphabetic)
- No validation (95-character code set equivalent to the last 6 columns of the ASCII table, excepting DEL. Note that the hyphen and minus sign are the same ASCII character, as are the period and decimal point).

When an invalid character is entered into a subfield requiring validation, an audible alarm is sounded, the cursor remains in its current position, and the character is not accepted or echoed. The LPH continues to process the current order without notifying the application of the input error. When the field is completed and accepted by the LPH, further validation may be performed by the application. For reasons of security, an application may specify (in I_DVS) no echo for a field. When an invalid character is entered into such a field, no audible alarm is sounded.

## AUTO-INSERT CHARACTERS

An auto insert character is a predetermined character in a predetermined location within a field. It is defined as a subfield by the field attribute descriptor. Consider, as an example, the standard Social Security account number:

123-45-6789

This field occupies 11 positions. It can be defined as an 11-character numeric field, in which case the operator must key in the hyphen. It can also be defined as follows:

A digit subfield of 3 positions
An auto-insert character
A digit subfield of 2 positions
An auto-insert character
A digit subfield of 4 positions

In this case, the operator may not key in anything but digit characters. The hyphens are inserted automatically by the LPH.

Contiguous auto-insert subfields are not allowed; at least one other type of subfield must be defined between auto-insert subfields within a field. An auto-insert must not be the first or last subfield of a field.

## SEPARATE SIGN FIELD

The separate sign subfield allows the operator to enter a minus or plus sign as the first character of a field. If a character other than a minus or plus sign is entered, a plus is assumed and placed in the buffer associated with the field read order. The keyed character is then stored in the buffer. If echo is requested, the assumed plus sign, followed by the keyed character, is displayed on the screen.

If the operator moves the cursor to the left into a separate sign subfield, a new value (+ or - ) may be entered. However, if the operator enters another character or moves the cursor right into the separate sign subfield, the default sign (+) is stored in the buffer and displayed on the screen (assuming specification of echo).

The separate sign subfield must be the first subfield of the field. It may only be used in conjunction with a decimal-point and digit subfields.

## MUST RELEASE FIELD

Must release fields are the same as normal fields with one exception: the field is not considered complete at end-of range; the operator must key in a terminator character. Take, for example, a form containing two fields. One field is a zip code, defined as digit, length 5; the other field is the customer name, defined as alphabetic, length 20. In a data entry environment, the zip code would probabably not be defined as a must release field; after the operator keys in the 5 digits, the cursor automatically moves to the next field. The customer name field, however, would probably be defined as a must release field, forcing the operator to key in a terminator character regardless of the length of the customer name. (Valid termination characters are defined later in this section under "Termination of Field".)

If the operator fails to enter an appropriate termination character after filling a field (i.e., after entering 20 alphabetic characters, in the preceding example), an audible alarm sounds until a valid terminator character is entered.

## DECIMAL POINT AND DECIMAL POINT PROCESSING

If the decimal point subfield is used, the separate sign must also be specified. The separate sign subfield must be the first subfield of the field. The decimal point subfield must occur somewhere later in the field description and is used by the LPH as an aligment position. The decimal point subfield must not occupy the last position of the field and only one such subfield can be used within a field.

If the operator keys in a plus or minus sign as the first character of a field, the sign is stored in the read buffer and transmitted to the terminal (assuming that echo is specified in the IORB). If the operator keys in any other character except the decimal point as the first character, that character is stored as the second character of the field (following successful validation). It too is echoed to the terminal if echo is specified. If the operator keys in the decimal point character, or if the cursor occupies the position in the field designated for the decimal point, the decimal point character is stored in the buffer at the next available position.

The decimal point character is also transmitted to the screen, assuming specification of echo. The next character entered is treated as part of the next digit subfield following the decimal point subfield, and is validated according to the attributes of that subfield. The operator is not allowed to move the cursor left into an designated decimal point position. An audible alarm is sounded if this is attempted.

This attribute must be used in conjunction with the separate sign and digit subfields. Also, there can be only one occurrence of this subfield and it cannot occupy the last position of the field.

## FIELD DESCRIPTOR AND DEFINE FORM

Before a read order in field mode can be processed, the application must either issue a define form request or incorporate a field descriptor in the IORB itself. Bit 2 of I_DV2 indicates whether the IORB is carrying the integrated field descriptor along with the read request. If the bit is on, the field descriptor starts at offset I_LOG in the IORB. Alternatively, with bit 2 of I_DV2 set off, the application must issue a define form order that points to a set or table of field attribute descriptors that define the form.

Integrating a field descriptor in the IORB is the preferred approach, because an application can more efficiently alter an integrated descriptor than one that is part of a external table. After altering the attributes defined by a integrated descriptor, the application issues a single read order; after altering the attributes defined by a descriptor in a table, the application must issue a new define form order and a field read order. Two I/O orders are required rather than one.

## USING THE INTEGRATED FIELD ATTRIBUTE DESCRIPTOR

When using the integrated field attribute descriptor, the application must specify in words the total extension length of the IORB. The integrated descriptor begins at offset I_LOG, which is the first word of the logical part of the IORB. The value for the total size of the IORB extension must include both the size of the physical IORB extension (seven words) and the size of the integrated field attribute descriptor.

## USING DEFINE FORM

The following conventions apply to the use of the define form order and the associated table of field attribute descriptors.

1. The IORB that requests a define form order is physically extended.

2. The define form order must be issued before any read order that refers to the field attribute table pointed to by the define form order.

3. After a define order is issued referencing a field attribute table, subsequent define form orders may not be issued while read orders that reference the initial field attribute table are outstanding. The define form order remains active and the associated attribute table is used for all subsequent field reads until another define form or a disconnect order is issued, or a line disconnect is detected.

4. The table address is passed in I_BAD of the define form IORB. The range (I_RNG) must specify the length of the table in bytes. The logical portion of the IORB (I_FCN through I_CON) must be zero.

5. The attribute table must begin on a word boundary; consequently, the buffer bit (bit 8) of I_CT2 must be zero.

6. Once the field attribute descriptor table and its address have been established, any subsequent field read order must specify in I_TAB the word offset to the desired field attribute descriptor. Accordingly, all field attribute descriptors must start on a word boundary.

7. The application may organize the attribute table in any manner that is convenient (as long as the descriptors start on word boundaries). The descriptors may be interspersed with other information, if conservation of memory is not a prime consideration.

8. Conservation of memory can be acheived by the following measures:

   a. If the attributes of two or more fields are exactly alike, only one descriptor is needed. All read orders referring to the identical fields would reference the same descriptor.

   b. In some cases, it might be advantageous to apportion the descriptors describing a form into a set of attribute tables rather than into a single table. Only one table of the set would be in memory at a time; when another attribute table was needed, the application would issue another define form order.

## FORMAT OF THE FIELD ATTRIBUTE DESCRIPTOR

Field attribute descriptors have a single format, whether integrated into a field read IORB or belonging to an field attribute descriptor table. A field may contain one to nine subfields. The field attribute descriptor consists of the following:

- A one-byte entry defining the length of the field descriptor

- A one-byte entry defining the must-release attribute
- A two-byte entry defining the type and range of the subfield (there can be up to nine such subfield definitions)
- A two-byte field descriptor terminator.

The format of these field descriptor components is shown in the following diagram.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| L | M | R1 | A1 | R2 | A2 | ... | Rn* | An* |

where:

L = Length of field descriptor (in bytes), not including this byte; a hexadecimal value in the range 5 to 15.

M = Must release field. Bit 4, when set to 1, signifies that the entire field is designated a must release field. The other bits are reserved for future use and must be zero.

Entries L and M constitute a 2-byte descriptor header.

R = Range of a subfield, in decimal, or zero

A = Attribute of the subfield; a hexadecimal value

Rn* = The value of the last two R and A entries must be zero,
An* to indicate the end of the descriptor. These two entries constitute the terminator.

If the value of a range byte (R) is greater than 0 and less than or equal to 80, the value of the attribute byte (A) has the following significance:

| Value | Meaning |
|---|---|
| 00 | No validation |
| 10 | Digit (0-9) |
| 30 | Numeric (0-9, 1/4, 1/2, divide sign, multiply sign, decimal point, minus sign, plus sign, comma) |
| 40 | Alphabetic (A-Z, a-z, extended characters, period, space, comma, hyphen, apostrophe) |
| 70 | Alphanumeric (all numeric and alphabetic) |

If the value of a range byte (R) is zero, the value of the attribute byte (A) has the following significance:

| Value | Meaning |
|---|---|
| 00 | End of field |
| 20-7E | Auto-insert character |
| 80 | Separate sign |
| 81 | Decimal position |

The range of the total field, specified in I_RNG of the field read IORB, may not exceed 80 characters. The range value can normally be computed with the following formula:

range = sum of R1...Rn subranges + number of auto-insert characters + 1 (if separate sign specified) + 1 (if decimal point specified).

## Supervisory Message Processing

When a terminal is in field mode, the application may "escape" to a supervisory message line by issuing read/write orders with standard, non-extended IORBs. Escaping to the supervisory message line allows two-way communication between operator and application that does not disrupt the processing of a form displayed on the terminal. For example: An operator (who is using a terminal both for forms processing and as an operator console) receives a device unavailable message on the bottom line of the terminal. The form being processed is not altered by the supervisory message. The operator acknowledges the supervisory message and continues processing the form.

### IORB VALUES

Supervisory messages are designated by a common bit (bit 9) in the read/write device specific word. The use of this bit is optional in field mode, because supervisory message orders are already distinguished from field mode orders by being non-extended.

Bit 8 of I_DVS becomes significant when supervisory message writes are specified. If bit 8 = 0, supervisory messages must be acknowledged. If bit 8 = 1, acknowledgement by the operator is not required.

### LOCATION OF MESSAGE LINE

If the terminal is defined at system building time as a VIP7800, VIP7300, or HDS 2 class terminal, the supervisory message line is the 25th line of the CRT. If the terminal is defined as a VIP7200 or VIP7207, the application may designate (in I_FCS) any line from 1 through 24 as the supervisory line.

## PROCESSING ORDER

Supervisory message orders are processed by ATD in the order received, with write orders having priority over read orders. Assume, for example, that four supervisory messages are issued and queued in the order listed: write, read, write, read. The two writes will be completed before the reads are processed.

If supervisory message orders are intermixed with extended IORB field mode orders, the messages are processed in the order received, with write orders again having priority over read orders. Assume, for example, that three orders are issued and queued in the order listed: field mode read, supervisory write, supervisory read. The orders will be processed in this order: supervisory write, field read, supervisory read.

## SUPERVISORY MESSAGE CONVENTIONS

The following conventions apply to the processing of supervisory messages:

1. The receipt of a supervisory message by the LPH does not cause the premature termination of the current order, whether the current order is a supervisory message or normal field order.

2. Control byte and post-order control processing does not apply to supervisory messages.

3. If the type-ahead option was selected at connect time, a supervisory message results in a purge of the type-ahead character queue.

4. When writing a supervisory message, the application must not imbed in the message text control sequences that move the cursor (e.g., carriage return, line feed).

5. The range of a supervisory write order cannot exceed 80 characters. Data in excess of 80 characters is not sent to the terminal.

6. The operator must acknowledge the receipt of each supervisory message by depressing function key 10, the transmit key, or the CLEAR key.

7. The operator can edit a response to a supervisory message read through the use of TTY edit control characters.

8. The break function is not operational when a supervisory message read is being processed.

9. An operator keying in a response to a supervisory message read initiates transmission of the response by one of the following actions:

a. Depressing the carriage return key

b. Depressing the transmit key

c. Entering the number of characters specified in I_RNG of the IORB issued by the application to read the operator's response.

10. The range of a supervisory message read order cannot exceed 80 bytes. If a longer range is specified, a range of 80 is used, and the residual range set accordingly.

11. ATD field mode applications that specify supervisory message processing and use the VIP7808, VIP7803, or VIP7303 in word processing mode must set to 1 bit 7 of I_DV2 in the connect IORB. This action ensures that the LPH keeps the terminal in word processing mode when servicing supervisory message requests.

## CALCULATOR KEY PAD SUPPORT

The multifunction (MF) keyboard includes a calculator key pad with plus (+) and minus (-) keys. These keys generate three-character escape sequences. When a MF keyboard is attached to a terminal running in field mode, ATD, by default, translates the escape sequences into ASCII plus or minus characters (X'2B' or X'2D', respectively). ATD then stores the ASCII character in the application's buffer. The application can request ATD not to translate the escape sequences by setting to 1 bit 1 of word I DV2 in the field connect IORB. When this bit is set, the escape sequences terminate the read order; ATD stores the terminating sequence in the IORB as explained later in this section under "ATD Handling of Termination Codes".

### Application Responsibilities in Processing Fields

The application is responsible for:

1. Initializing the read buffer with blanks, underscores, or semiconstant values.

2. Initializing the terminal display, through a field write order, with the same initialization sequence set in the read buffer.

3. Justification (left, right) after the field read is complete.

4. Decimal point alignment after the field read is complete.

5. Space suppression.

6. Logical validation of field content (beyond what is provided by ATD).

## Field Mode Functions

Field mode supports six I/O request blocks:

| | | |
|---|---|---|
| Connect | Define Form | Write |
| Disconnect | Read | Break. |

All but the break function require an extended-length IORB. When using an extended length IORB, bit 11 in I_CT2 must be set on, the right byte of I_EXT must specify a physical extension of seven words, and the left byte of I_EXT must specify a minimum total size of at least seven words.

### CONNECT FUNCTION

An application selects field mode by using an extended-length connect IORB and setting bits 8, 9, 10, and 11 of I_DV2 to the field processing subfunction code of 2. (Bit 10 is set to one; the other three bits are zero.) In field mode, the connect IORB can specify the following options.

#### Auto Call

Specification of auto call in I_DVS enables an application to establish a connection with either a 801-A or 801-C ACU data set. The auto call feature is described in Section 7.

#### Bell

The default setting of I_DVS allows the output of bells to a terminal. If the option is specified, the LPH suppresses the output of bells to a terminal even under error conditions. This means, for example, that the operator receives no indication when the LPH rejects entry into a field, or when entry of a terminator is required (when processing a must release field).

#### Validation Field Notification (VFN)

Specifying the VFN option (in I_DV2) causes the ATD, instead of issuing a bell, to post back the current read order with a return status of zero whenever the operator attempts to enter an invalid character into an active field.

Having specified the VFN option, the application determines the reason for the termination of the read order. If the order was terminated by the attempt to enter an invalid character (e.g., keying an "A" into a numeric subfield), ATD places an error code in I_CON. Having found this code, the application issues a supervisory message write to inform the operator of the error. Once the operator acknowledges the message and the supervisory message is posted back to the application, the application can reissue the interrupted field read and continue processing from the last valid keystroke (by means of a read with offset, which is described later in this section).

Selectable Field Validation Sets

This option (specified in I_DV2) allows the application to select the set of ASCII characters constituting a field type.

There are three validation sets that can be selected:

Standard ATD set
VIP7700 set
VIP7800 set

User applications must select the default ATD set. The other validation sets are used by system-supplied software that supports emulation of VIP7700 and VIP7804 terminals.

Word Processing Mode (WPM) Indicator

This option is specified (in I_DV2) by system-supplied software when the word processing graphics mode (WPM) of a VIP7803, VIP7808, VIP7813, VIP7824, VIP7303, VIP7305, or HDS 2 is used. This option is necessary to provide proper processing of supervisory messages when the terminal is in WPM mode.

Cursor Out of Field

If specified (in I_DV2), this option allows the operator to "cursor out" of a field and thus terminate the read of that field. The reason for termination is reported by ATD in the extended portion of the read IORB (I_TAB). If the option is not selected, the operator cannot use the cursor left key (at the beginning of a field) or cursor right key (at the end of a field) to terminate an active field read.

Type Ahead

This option, when specified (in I_DV2) helps to prevent the loss of input characters when a read order is not active (i.e., when a write order is active and/or a read order has not been issued by the application.) If this option is chosen, ATD queues (in a 32-character key-ahead buffer) input characters that are keyed when a read order is not active. Later, when the read order becomes active, these characters are validated against the field attribute descriptor and echoed (if echo was requested). Detection of an invalid character causes an audible alarm to sound and the type-ahead character queue to be purged. Cursor right and left and end-of-field conditions are acted on by ATD when the read order becomes active. If this option is not selected, characters are accepted only when a read order is currently active. The keying of characters when a read order is not active causes an audible alarm to sound. The type-ahead queue is purged by any of the following events:

1. An input character in the queue is found to be invalid.

2. The application issues a supervisory message read or write order.

3. The operator presses the break key.

4. The terminal is disconnected.

5. The application issues a purge-all I/O order.

6. The application issues a read IORB with the the purge type-ahead queue bit set on.

7. The request issues a read order with terminal enquiry (ENQ) or with terminal read cursor address (RCA) specified as pre-order function in the IORB.

VIP7200, VIP7207 Supervisory Message Line

When issuing a connect to a VIP7200 or VIP7207, the application can specify (in the right byte of I_FCS) the line (row) to be used for supervisory messages. Possible values are 0 through 18 hexadecimal. If 0 is entered, line 24 is used. This field is ignored if the device is a VIP7800 or VIP7300 class terminal; in this case, line 25 is always used for supervisory messages.

Terminal Type (Device ID)

The application can check the device ID of the connected terminal by interrogating the right byte of I_QDP in the completed connect IORB.

Connect IORB (Field Mode)

This subsection summarizes the bit settings that govern the connect IORB options just described.

Bit Settings of I_DVS. Table 8-9 gives the signficance of bits in the connect IORB I_DVS word that are applicable to field mode ATD. All other bits must be zero.

Table 8-9. ATD Word I_DVS in Connect IORB

| Bit Number | Meaning for Connect Function |
|---|---|
| 2 | 0 = Do not use auto dial<br>1 = Use auto dial |
| 3 | 0 = Allow output of bells to the terminal<br>1 = Suppress output of bells to the terminal |

Bits Setting of I_DV2. Table 8-10 gives the significance of bits of the connect IORB word I_DV2 that are applicable to field mode ATD. All other bits must be zero.

Table 8-10. ATD Word I_DV2 in Connect IORB (Field Mode)

| Bit Number | Meaning for Connect Function |
|---|---|
| 1 | 0 = Translate codes generated by calculator pad + and - keys<br>1 = Terminate read when these keys are struck |
| 4 | 0 = No Validation Field Notification (VFN) support<br>1 = VFN support |
| 5,6 | 00 = Use standard field validation set (required setting)<br>01 = Use VIP7700 field validation set (reserved for system use)<br>10 = Use VIP7804 field validation set (reserved for system use) |
| 7 | 0 = Terminal is VIP7200, VIP7201, VIP7207, VIP7801, VIP7301, VIP7307; or VIP7803, VIP7808, VIP7813, VIP7814, VIP7824, VIP7303, VIP7305, HDS 2 and is not operating in word processing graphics mode (required setting)<br>1 = Terminal is VIP7803, VIP7808, VIP7813, VIP7824, VIP7303, VIP7305, HDS 2 and is operating in word processing graphics mode (reserved for system use) |
| 8 | Must be 0[a] |
| 9 | Must be 0[a] |
| 10 | Must be 1[a] |
| 11 | Must be 0[a] |
| 12 | 0 = Operator not allowed to cursor out of field<br>1 = Operator allowed to cursor out of field (terminating field read) |
| 13 | 0 = No type ahead queue<br>1 = Type ahead queue is supported |

[a] Bits 8 through 11 must be set as indicated to indicate a field mode connect.

Bit Settings of I_FCS and I_QDP. The right byte of I_FCS specifies the line (or row) number of VIP7200, VIP7201, or VIP7207 that is used as the supervisory message line. Possible values are 0 through 18, hexadecimal. Zero indicates use of the 24th line (VIP7200 class terminals) or the 25th line (VIP7300, HDS 2, or VIP7800 class terminals).

Values Returned on Completion of a Connect Order. On completion of the connect order, the right byte of I_QDP contains the device ID of the terminal (refer to Table 8-3).

DISCONNECT FUNCTION (FIELD MODE)

The disconnect IORB is used to terminate field mode processing. A disconnect IORB can specify the following two options.

Abort Queued Orders

If this option is selected, all outstanding IORBs, even if active, are terminated with a device unavailable (010B) status. The disconnect order is then immediately serviced. If this option is not selected, all outstanding IORBs are allowed to complete (in the order of their issuance) before the disconnect order is serviced.

Hang Up

If this option is selected, the communications line is physically disconnected when the disconnect order is serviced. If this option is not selected, the terminal/line remains physically connected after processing of the disconnect order (i.e., the terminal is logically disconnected, but remains physically connected). Table 8-11 gives the significance of bits of the disconnect IORB I_DVS word that are applicable to the disconnect options just described. All other bits must be zero.

Table 8-11. ATD Word I_DVS in Disconnect IORB

| Bit Number | Meaning for Disconnect Function |
|---|---|
| 14 | 0 = Abort outstanding requests<br>1 = Wait until outstanding requests complete before disconnecting the terminal |
| 15 | 0 = Hang up the phone<br>1 = Do not hang up phone |

## READ FUNCTION (FIELD MODE)

An extended-length field read IORB is used to obtain validated input that has been keyed into a field displayed at a terminal. The input to a field is validated by means of a field descriptor, which must be associated with the field read order. The descriptor may either be integrated into the read IORB or belong to a table of descriptors pointed to by a define form IORB. (For further detail, see "Field Descriptor and Define Form" earlier in this section).

### Pre-order Control

Pre-order control arguments are specified in I_DV2 and I_CON of the read order IORB. Pre-order control is used to perform the following actions prior to a field read:

- Positon cursor
- Issue bell
- Erase line (i.e., clear screen from cursor position to end of line)
- Issue enquiry command (ENQ) to VIP7801, VIP7803, VIP7808, VIP7301, VIP7307, HDS 2 and read terminal's response
- Issue a read cursor request command (RCA) to terminal and read current position of the cursor.

If the pre-order control request is an ENQ or RCA command, the read is not treated as a field read. After sending an ENQ or RCA control sequence to the terminal, the LPH places the terminal's response in the buffer associated with the read request. Before issuing an ENQ or RCA read order, the application must specify in the read IORB no echo of incoming characters and no post-order control. (For more information about the ENQ and RCA commands, see the hardware documentation of the terminal in question.)

### Termination of a Field Read

An operator intentionally terminates a field read by one of two actions:

1. The operator types a valid data character into the last position of a field that is not a must release field. This action sets I_RNG to and I_TAB to zero.

2. The operator types a control sequence that terminates the read order. The character(s) making up the control sequence must fall in certain ranges (defined below) of the ASCII character set; the significance of the sequence, however, is determined by the application. Keying a control sequence sets a non-zero residual range in I_RSR. The terminator sequence is stored in the I_TAB and I_CON fields of the IORB, and not the buffer; it is not included in the residual range calculation or echoed to the terminal.

The terminating sequence may be a one-character control character or an escape sequence from one to four characters long.

1. One-Character Terminating Codes. A one-character terminator must be one of the following ASCII codes: 00-1A, 1C-1F, 7F. Note that codes 10 and 11 are not treated as terminators by ATD if the terminal is a VIP7207 or VIP7307. The use of codes 10 and 11 is not recommended if compatibility with all terminal types is desired.

2. Multi-character Terminating Codes. Multi-character terminators are two-, three-, or four-character sequences beginning with the escape code 1B. The second character of the sequence must be in the range 20 - 7E.

   A two-character sequence must consist of the escape character (1B) followed by 20 to 57; 59 to 5A; 5C to 72; or 74 to 7E.

   The VIP7800, VIP7300, and HDS 2 terminal classes support three- and four-character escape sequences. The first two characters must be 1B followed by either 58, 5B, or 73.

Escape sequences longer than four characters are not supported; the fifth and any successive character(s) are treated as data. For further information on escape sequences, refer to documentation describing a specific terminal.

ATD Handling of Termination Codes

The terminating sequence keyed by an operator is placed by ATD in extended IORB fields I_TAB and I_CON. The following rules apply.

1. One-character codes. One-character codes are placed in the right byte of I_TAB and are in the range of 00 - 1A, 1C - 1F or 7F.

2. Two-character escape sequences. The escape character (1B) is not stored. The second character is stored in the right byte of I_TAB and is in the range of 20 - 7E (excluding 5B, 58, and 73).

3. Three- and four-character escape sequences. The escape character (1B) is not stored. The second character is stored in the right byte of I_TAB and is 5B, 58, or 73. The left byte of I_CON contains the third character; the right byte of I_CON contains the fourth character. The value X'00' in the right byte of I_CON signifies that the terminating code is a three-character escape sequence.

Entry of Invalid Characters

The effect of entering an invalid character into a field requiring validation depends on whether the validation failure notification (VFN) option was selected at connect time.

1. VFN Option Not Selected. An audible alarm sounds (if output of bells is supported), the cursor remains in its current position, and the invalid character is not echoed. The LPH continues to process the current order without notifying the application of the input error.

2. VFN Option Selected. Field read order is returned to application with a 0 status in I_CT1. Right byte of I TAB contains X'FF', indicating that I_CON contains one of the following error codes:

   1 = Illegal entry into a digit subfield
   2 = Illegal entry into a numeric subfield
   3 = Illegal entry into an alphabetic subfield
   4 = Illegal entry into an alphanumeric subfield

Residual Range and Relative Residual Range

When a field read order is terminated, the residual range, returned in I_RSR of the IORB, reflects the maximum cursor position reached while the read order was active. The relative residual range, returned in I_QDP of the IORB, reflects the position of the cursor when the order was terminated. The values of residual and relative residual range may differ if the cursor back (<-) key was entered during a read order. Suppose, for example, that the operator keys

```
AXC<-<-B<-
```

followed by a carriage return. The residual range shows that three characters (ABC) were entered; the relative residual range indicates that the cursor was in the second position when the order was terminated by a carriage return.

The residual and relative residual range are set equal to the original range if a read order is prematurely terminated by a communication line loss, a purge-all, or an abortive disconnect.

## Use of Cursor Keys

When the operator moves the cursor left (<-) or right (->) within a field, the LPH's buffer pointer is adjusted and the buffer contents remain unchanged. For example, after the operator keys

ABC->->

followed by a carriage return, the buffer contains ABCxx, with xx being the previous contents of the buffer. (The residual range indicates that five characters were entered.)

## Statistics

The total keystroke count for a field read is returned in I_FCS of the field read IORB when the order terminates. When the read order is active, the count is incremented once for each of the following:

- Data character (valid or invalid)
- Cursor right (->)
- Cursor left (<-)
- The terminating character sequence.

Statistics are not returned in the IORB if the read order is prematurely terminated (e.g., by a communication line loss).

## Read With Offset

An application can specify an offset when issuing a field read order so that the operator can start entering data in the middle of a field. When issuing a read with offset, an application does the following:

- Specifies in I ADR the starting address of a buffer that contains the data from the field previously read.

- Specifies in I_RNG the size of the buffer pointed to by I_BAD.

- Specifies in I_HDR the offset from the start of a field to a position within the field where the cursor is to be placed and where the read with offset is to begin. Permissible values are in the range 1 through 4F, hexadecimal.

- Optionally, specifies in I_CON the cursor position to the start of the field.

For example, a 20-character alphabetic field begins in row 2 column 1. The application previously issued a field read with no offset, but the field entered by the operator contained an invalid character in the tenth position of the field. The application recognizes the error and reissues the read with:

- A pre-order bell
- A pre-order positioning of the cursor at row 2, column 1
- An offset of 9 specified in I_HDR
- The address and range of the buffer containing the previously read data, specified in I_ADR and I_RNG, respectively.

During a read with offset, the operator is allowed to cursor left or right within the entire field. Cursoring out of a field follows the normal termination rules. The LPH calculates the residual range and the relative residual range for a read with offset order as if the operator had entered the characters preceding the specified offset. Read with offset may be used with or without the type-ahead option.

Type-Ahead

If the type-ahead option was selected at connect time, the application may select the "purge type-ahead queue" option in the field read IORB. This option causes the LPH to purge the type-ahead queue before processing the read order. The option is useful if the application detects an error in field read and wants to re-issue the read after purging the queue.

Cursor Out of Field

When issuing a field read order, an application can override the selection of the cursor-out-of-field option made at connect time. That is, by setting a bit in I_DV2 of the read IORB, the application can specify that the operator cannot cursor out of the field.

Support of VIP7207 and VIP7307 Terminals

Through support of the ALPHA key and implied numeric shift, ATD supports data entry operations on the VIP7207 and VIP7307 terminals. The purpose of this functionality is to allow the operator to enter alpha (i.e., lower case) characters from a data entry terminal while the terminal is shifted to uppercase as a result of numeric lock or implied numeric shift. It is perceived by the operator as a terminal function related to character entry, and is not tied into the field validation operation. Field validation checks are done after the character is translated; if the resultant character is invalid it is rejected at that time.

The data entry terminal transmits a code 10 when the ALPHA key is depressed and a code 11 when the key is released. The LPH interprets code 10 as a shift to the "alpha" set of characters, translating the data characters following the code 10 into equivalent alpha codes until a code 11 is received. ATD so interprets codes 10 and 11 whether or not type-ahead is in effect.

When the operator is responding to a supervisory message read, ATD treats codes 10 and 11 as data, placing them in the application's buffer; no translation is performed. After the supervisory message read is complete, the LPH reverts to the mode (ALPHA or implied numeric shift) that was in effect immediately before the supervisory read.

The purpose of the numeric shift option (specified in I_DV2) is to reduce the number of keystrokes required of the operator during the entry of numeric data by enabling the application to shift the state of the terminal instead of requiring the operator to depress the numeric shift key. The use of this option is not restricted to numeric type validation fields, and it can be used wherever it will save the operator keystrokes. Thus, for alphanumeric fields that typically consist mostly of digits, this implied shift would cause the terminal to echo digits and the ALPHA key could be used to enter the occasional letter.

When a field read with implied numeric shift is requested, the characters entered are translated before the field validation operation is performed. For example, the operator normally enters alpha data. The terminal is set for alpha: the numeric lock is not set and the ALPHA key is not used. The application issues an order to read a three-character alphabetic subfield and a three-character digit subfield with the implied numeric shift option. The operator, using the central keyboard (not the numeric keypad), enters ABCUIO. ABC123 is placed in the application buffer.

Table 8-12 lists the data codes produced by a key in its unshifted (alpha) state and shifted (numeric) state. The two characters shown in each line of the table are produced by a single key. The first character is produced when the keyboard is unshifted or when the alpha key functionality is in effect. The second character is generated when the key board is shifted or when the implied numeric shift option is in effect.

## Read IORB (Field Mode)

This subsection summarizes the bit settings that govern the field read IORB options just described.

Bit Settings of I_DVS. Table 8-13 gives the significance of bits of the field read IORB I_DVS word that are applicable to field mode ATD. All other bits must be zero.

Bit Settings of I_DV2. Table 8-14 shows bits in the field read IORB word I_DV2 that are significant to ATD. All other bits must be zero.

Table 8-12. Data Entry Keyboard Unshifted/Shifted Translations

| Unshifted (alpha) | Shifted(numeric) | Terminal |
|---|---|---|
| S (53) | > (3E) | |
| X (58) | ? (3F) | |
| T (54) | [ (5B) | VIP7207 |
| R (52) | [ (5B) | VIP7307 |
| H (48) | \ (5C) | VIP7207 |
| G (47) | ] (5D) | VIP7207 |
| T (54) | ] (5D) | VIP7307 |
| R (52) | ^ (5E) | VIP7207 |
| W (57) | _ (5F) | |
| { (7B) | \| (7C) | VIP7207 |
| { (7B) | } (7D) | VIP7307 |
| } (7D) | ~ (7E) | VIP7207 |
| ^ (5E) | ~ (7E) | VIP7307 |
| G (47) | none | VIP7307 |
| B (42) | ! (21) | |
| C (43) | " (22) | |
| @ (40) | # (23) | |
| * (2A) | $ (24) | |
| P (50) | & (26) | |
| N (4E) | ( (28) | |
| E (45) | ) (29) | |
| Q (51) | + (2B) | |
| % (25) | , (2C) | |
| < (3C) | . (2E) | |
| none | / (2F) | VIP7207 |
| H (48) | / (2F) | VIP7307 |
| / (2F) | 0 (30) | |
| U (55) | 1 (31) | |
| I (49) | 2 (32) | |
| O (4F) | 3 (33) | |
| J (4A) | 4 (34) | |
| K (4B) | 5 (35) | |
| L (4C) | 6 (36) | |
| M (4D) | 7 (37) | |
| , (2C) | 8 (38) | |
| . (2E) | 9 (39) | |
| D (44) | : (3A) | |
| F (46) | ; (3B) | |
| V (56) | = (3D) | |

NOTES

1. "None" means that no code is generated.
2. Unless specified, the code translations apply to both VIP7207 and VIP7307 terminals.
3. Keys not represented in the table generate the same code in unshifted or shifted state.

Table 8-13. ATD Word I_DVS in Field Mode Read IORB

| Bit Number | Meaning for Field Read Function |
|---|---|
| 10 | 0 = Do not echo input<br>1 = Echo input |

Table 8-14. ATD Word I_DV2 in Field Read IORB

| Bit Number | Meaning for Field Read Function |
|---|---|
| 0 | 0 = Do not purge type-ahead queue<br>1 = Purge type-ahead queue |
| 1 | 0 = No implied numeric shift<br>1 = Implied numeric shift |
| 2 | 0 = No integrated field descriptor (define-form order required)<br>1 = Integrated field descriptor (starting at I_LOG) |
| 7 | 0 = Do not override cursor-out-of field capability<br>1 = Override cursor-out-of field capability specified in connect I_DV2 (field read terminates when attempt is made to cursor-out of a field) |
| 12 | 0 = Do not send pre-order bell<br>1 = Send pre-order bell |
| 13 | 0 = Do not send pre-order erase-line escape sequence<br>1 = Send pre-order erase-line escape sequence |
| 14 | 0 = Right byte of I_CON contains pre-order control (see Table 8-15); left byte must be zero<br>1 = I_CON contains pre-order cursor positioning information (see Table 8-15) |
| 15 | 0 = I_CON is not meaningful (no pre-order control)<br>1 = I_CON contains pre-order control information |

Bit Settings in I_CON. This field can be used to specify pre-order control. If so used, bit 14 of I_DV2 must be set. I_CON can be used to specify two kinds of pre-order control:

1. Pre-order cursor positioning. The application must indicate this use of I_CON by setting bit 15 of I_DV2 to one.

2. Pre-order control other than cursor positioning. The application must indicate this use of I_CON by setting bit 15 of I_DV2 to zero.

Table 8-15 shows the values of I_CON when used for either kind of pre-order control.

Table 8-15. ATD Word I_CON in Field Read IORB.

| Bit Number | Hex Value | Meaning for Field Read Function |
|---|---|---|
| Pre-Order Control Cursor Positioning Information | | |
| 0-7 | 01-50 | Defines column coordinate (hexadecimal) |
| 8-15 | 01-18 | Defines row coordinate (hexadecimal) |
| Other Pre-Order Control Information | | |
| 0-7 | | Must be zero |
| 8-15 | 00 | Line feed and carriage return |
| | 01 | Line feed |
| | 02 | Carriage return |
| | 03 | Bell |
| | 04 | Reserved for future use |
| | 05 | Reserved for future use |
| | 06 | Restore device's default attributes (VIP7800, 7300 class terminals); high intensity (VIP7200 class terminals) |
| | 07 | Low intensity attribute (VIP7800, 7300, 7200 class terminals) |
| | 08 | Cursor up |
| | 09 | Cursor down |
| | 0A | Cursor forward |
| | 0B | Cursor back |
| | 0C | Cursor home |
| | 0D | Erase end of line |

Table 8-15 (cont). ATD Word I_CON in Field Read IORB.

| Bit Number | Hex Value | Meaning for Field Read Function |
|---|---|---|
| | | Other Pre-Order Control Information (cont.) |
| | 0E | Erase end of display |
| | 0F | Clear (VIP7800, VIP7300, HDS 2 class terminals); reset (VIP7200 class terminals) |
| | 10 | Read cursor request binary (VIP7800, VIP7300, HDS 2 class terminals); read cursor address (VIP7200 class terminals) |
| | 11 | Blink (VIP7800, VIP7300, HDS 2 class terminals) |
| | 12 | Hide (VIP7800, VIP7300, HDS 2 class terminals) |
| | 13 | Inverse video (VIP7800, VIP7300, HDS 2 class terminals) |
| | 14 | Underline (VIP7800, VIP7300, HDS 2 class terminals) |
| | 15 | Secondary character set (VIP7800 class terminals) |
| | 16 | Enquiry (VIP7800, VIP7300, HDS 2 class terminals) |

NOTES

1. If codes 11 through 16 are used for terminal classes other than VIP7800, VIP7300, and HDS 2, an invalid parameter error (0104) will be returned.

2. When specifying codes 10 or 16, the application must supply, in IORB fields I_ADR and I_RNG respectively, the address and size of a buffer to receive the terminal's response. A buffer size of 4 bytes is required if code 10 is specified; a buffer size of 9 is required if code 16 is specified.

Bit Settings in I HDR and I TAB. If an application issues a read with offset, the right byte of I_HDR must contain the byte offset, expressed as a hexadecimal value in the range 1 through 4F. If no offset is required, the right byte of I_HDR must be zero.

When an a field read order is issued in conjunction with a define form order, I_TAB contains a word offset to the proper field attribute descriptor.

Values Returned by a Field Read Order

The following paragraphs summarize the information returned by ATD in fields of a terminated field read IORB.

I_RSR shows the maximum cursor position (offset) upon termination of the field read order.

I_FCS shows the total number keystrokes entered by the operator during the field read.

I_ST2 indicates, by a value of one in bit 15, that validated data was entered into the field.

I_QDP shows the current cursor position (offset) upon termination of the field read order.

I_TAB indicates termination condition as shown below:

| | |
|---|---|
| 0 = | End of range. Valid data has been entered into the entire field. |
| -1 = | Invalid character entered into field. If the VFN option was selected at connect time, I_CON provides additional information (see "Entry of Invalid Characters into a Field" earlier in this section.) |
| >0 = | ASCII code for one of the following: |

Single character terminator entered by operator

Second character of a two-character escape sequence

Second character of a three- or four-character escape sequence; I_CON contains remaining character(s) of the sequence

For the permissible values of terminator characters and sequences see "Termination of Field Read" earlier in this section.

## WRITE FUNCTION (FIELD MODE)

An extended-length IORB is used for all write orders directed against a terminal connected in field mode.

Write orders are typically used to:

- Display on the terminal screen a set of field "templates" associated with a form
- Purge all outstanding field read and write orders.

## Purge All Subfunction

The purge all option is a special form of the field write order. It is exercised by specifying a subfunction code of three in I_DV2, bits 8 through 11. Bits 10 and 11 are one; the other bits are zero. When this subfunction is specified, all other bit settings in I_DV2 and I_DVS are ignored. The write order causes outstanding read and/or write orders (active and queued) to be posted with a device unavailable (010B) status. Further, if the type-ahead option was specified at connect time, the type-ahead queue is purged.

## Quit on Break Option

If this option is specified (in I_DVS), a break signal can prematurely terminate an active write order.

## Pre-order Control

Four bits in I_DV2 control pre-order activity. By setting the range I_RNG to zero, the application can issue a write order that requests only pre-order activity. Alternatively, the write order can request both pre-order activity and the output of data to the terminal. In either case, the subfunction code (bits 8 through 11) of I_DV2 must be zero.

By manipulating bits in I_DV2, an application can:

- Send a bell
- Erase end-of line
- Use I_CON for cursor positioning operations
- Use I_CON for pre-order control operations.

These options are also available with field read orders and have been described in earlier parts of this section that concern the field read function.

## Write IORB (Field Mode)

This subsection describes bit settings in the field write IORB that govern the options just described.

Bit Settings in I DVS. Table 8-16 gives the significance of bits of the IORB word I_DVS that are applicable to field mode write. All other bits must be zero.

Bit Settings in I DV2. Table 8-17 gives the significance of bits in IORB word I_DV2 that are applicable to field mode write.

Bit Settings in I CON. The bit settings in this field is the same as those previously described for the field read function, with this exception: In a field write IORB, I_CON does not support codes 10 (read cursor address) and 16 (Enquiry).

Table 8-16. ATD Word I_DVS in Field Mode Write IORB.

| Bit Number | Meaning for Field Write Function |
|---|---|
| 7 | 0 = Stop output on detecting break<br>1 = Do not stop output on detecting break |
| 8 | 0 = Acknowledgement of supervisory messages required<br>1 = No acknowledgement required |

Table 8-17. ATD Word I_DV2 in Field Write IORB.

| Bit Number | Meaning for Field Write Function |
|---|---|
| 8 | Must be zero |
| 9 | Must be zero |
| 10-11 | 00 = Normal write<br>11 = Purge all outstanding read and/or write orders |
| 12 | 0 = Do not send pre-order bell<br>1 = Send pre-order bell |
| 13 | 0 = Do not send pre-order erase-line<br>1 = Send pre-order erase-line |
| 14 | 0 = Right byte of I CON contains pre-order control (see Table 8-15); left byte must be zero<br>1 = I CON contains pre-order cursor positioning (see Table 8 -15) |
| 15 | 0 = I CON is not meaningful (no pre-order control)<br>1 = I CON contains pre-order control information |

## FIELD MODE DEVICE CONFIGURATION

Hardware switches on a device connected in field mode should be set in the following positions. (The device may not support all of the switches mentioned below).

CHARACTER/BUFFER switch in CHARACTER position
DUPLEX HALF/FULL switch in FULL position
LOCAL COPY/ECHO switch in ECHO position
ROLL/NO ROLL switch in NO ROLL position
Speed set between 1200 and 9600 bits per second

## FIELD MODE RETURN STATUS CODES

The following return status codes are returned in the R1 register. The status code returned in I_CT1 is the right byte of the status code returned in the R1 register when the I/O order is complete.

### Invalid Argument Status (0104)

This status is returned for the following reasons:

- In a field read IORB
  - I_RNG (buffer size) is zero
  - Invalid pre-order control option or cursor position coordinate in I_CON
  - The format or values of a field descriptor are invalid.
- In a field write IORB
  - Invalid pre-order control option or cursor position coordinates in I_CON
  - Improper bit settings in I_DV2 (bits 8 through 12 must be all zero; else bits 8 and 9 set to zero and bits 11 and 12 set to one).

### Inconsistent Request Status (010C)

This status is returned for the following reasons:

- In a field connect IORB
  - The IORB specifies a field mode connect to a terminal that is supporting (a connected) serial printer that is attached to the terminal by a buffered printer adapter.
- In a define form IORB
  - A read order is presently using an outstanding and active define form order; definition of a new define form is not allowed.
- In a field read IORB
  - A field attribute descriptor has not been specified.

## FIELD MODE ERROR PROCESSING

When a parity error is detected in keystroke input, an audible alarm sounds and the typed character is ignored. When the read order is posted, the return status in I_ST indicates detection of parity error(s) (bit 9 = 1).

If a framing error or receive overrun condition is detected, the read order terminates and a hardware error (0107) is returned; I_ST indicates the specific reason for abnormal termination.

FIELD MODE TIMEOUT PROCESSING

Timeouts may occur during the processing of read orders. A timeout occurs when the operator does not terminate the input within 5 minutes after entering the first character. The timeout value of 5 minutes can be changed via the read parameter of the CLM directive TIMEOUT. There is no timeout if the operator does not enter any characters. Write orders do not incur timeouts.

## BLOCK MODE

Block mode is applicable only to the VIP7800 class of terminals (VIP7801, VIP7803, and VIP7808) and is intended to support the terminal in its native text or forms mode.

Block mode supports five functions: Connect, Disconnect, Read, Write, and Break.

These functions are requested through standard-length IORBs. An application can optionally use an extended IORB for a connect operation.

### Block Mode and Extended Character Set (8-Bit Data)

The handling of the extended character set for block mode is the same as that described for TTY mode.

### Connect Function

A connect order establishes the mode in which the connected terminal operates. Block mode is selected by setting bit 0 of I_DVS to one. If an extended-length connect IORB is used, the terminal's device ID is returned in the IORB extension (right byte of field I_QDP).

An application specifies in I_RNG of the connect IORB the size of data blocks to be transmitted from the terminal. Permissible block sizes range from 22 to 270F bytes, hexadecimal. If an application fails to specify a valid block size, the connect order is rejected with an 0104 (invalid argument) error code.

Transmitted blocks terminate with either an end-of-block (ETB) or an end-of-text (ETX). When a block is transmitted from the terminal, the type of terminator (ETB or ETX) is passed to the application through the IORB and through an optional control word, which is described below. The following options can be specified when connecting in block mode.

## AUTO CALL

The Auto Call option, which is supported by all system-supplied LPHs, is described in Section 7. This option allows an application to establish a connection using an 801-A or an 801-C ACU data set.

## CONTROL WORD

At connect time, an application can specify control word processing for subsequent read and write orders. If this option is specified, ATD treats the first two bytes of the user's buffer as a control word. If control byte processing is also specified, the third byte of the user's buffer is considered the control byte. ATD uses the control word primarily to pass information to an application on completion of read orders. ATD places similar information in the IORB word I_ST when a read order completes. The first 4 bits of the control word contains information that is passed to the application upon completion of a read order. It has the following format:

| BIT | 0 | 1 | 2 | 3 | 4 — 15 |
|---|---|---|---|---|---|
|  | 0 | x | x | x | LRN 0 - 4095 |

Bit 1

0 = -
1 = Block missed; data lost

Bit 2

0 = -
1 = Long block; data lost

Bit 3

0 = ETX terminated block
1 = ETB terminated block

The last 12 bits of the control word specify the logical resource number of the referenced terminal. If specified, the control word must be included in the range (I_RNG) of the associated data buffer.

## SPACE SUPPRESSION

If this option is specified, ATD configures the terminal to suppress spaces, in certain instances, when transmitting data. One example of space suppression is the replacement of spaces between fields by a horizontal tab character or unit separator character; another example is the elimination of spaces at the end of lines that are terminated by a carriage return and line feed. For additional details, consult the documentation for the terminal in question.

NO ROLL

Selecting this option keeps the terminal from scrolling line 1 "off the screen" when text (including a carriage return) is entered into line 24. This option is especially useful to applications that process forms. If this option is not used, the screen scrolls as new text is entered in line 24. Roll mode is the customary operating mode chosen by an application that processes line-at-a-time input from the terminal.

## Connect IORB (Block Mode)

This subsection summarizes the bit settings that govern the connect options already described. Table 8-18 gives the significance of bits of the connect I_DVS word that are applicable to block mode.

A block size must be specified in this field if block mode is selected (bit 0 of I_DVS is one).

The I_ST word field is significant when a serial printer is attached to the terminal by means of a buffered printer adapter. On connect orders, the field specifies whether the terminal or attached printer is being addressed. The permitted values are:

0 = Terminal
1 = Attached serial printer

Upon completion of a connect order, ATD returns in the right byte of I_QDP the device ID of the terminal (refer to Table 8-3).

Table 8-18. I_DVS Word in Connect IORB (Block Mode)

| Bit Number | Meaning for Connect Function |
|---|---|
| 0 | 0 = Do not use block mode<br>1 = Use block mode |
| 2 | 0 = Do not use auto dial<br>1 = Use auto dial |
| 4 | 0 = Include control word<br>1 = Do not include control word |
| 8 | 0 = Do not use space suppression<br>1 = Use space suppression |
| 9 | 0 = Use roll<br>1 = Use no roll |

## Disconnect Function (Block Mode)

An application uses the disconnect IORB to terminate block mode processing. The following paragraphs describe the options that an application can specify with a disconnect order.

If the abort option is specified, outstanding IORBs (active and queued) are terminated with a "device unavailable" status (010B). The disconnect order is immediately serviced. If the abort option is not specified, all outstanding IORBs are allowed to complete before the disconnect order is serviced.

If the hang-up option is selected, the terminal is physically disconnected when the disconnect order is serviced. If the hang-up option is not specified, the communications connection remains active after servicing of the disconnect order (i.e., the terminal is logically disconnected but remains physically connected).

## Disconnect IORB (Block Mode)

This subsection summarizes the IORB bit settings that govern the disconnect options just described. Table 8-19 shows bits of the disconnect IORB that are applicable to the block mode of ATD. All other bits must be zero.

The bit settings in word I_ST are significant when a serial printer is attached to the terminal by means of a buffered printer adapter. On disconnect orders, the field specifies whether the terminal or attached printer is being addressed. The permitted values are:

0 = Terminal
1 = Attached serial printer

Table 8-19. I_DVS Word in Disconnect IORB (Block Mode)

| Bit Number | Meaning for Disconnect Function |
|---|---|
| 14 | 0 = Abort outstanding requests<br>1 = Wait until outstanding requests complete before disconnecting the terminal |
| 15 | 0 = Hang up the phone<br>1 = Do not hang up the phone |

## Read Function (Block Mode)

The read order is used to obtain blocks of data transmitted from the terminal. It is the application's responsibility to specify a buffer size large enough to hold a complete block of data. If a block exceeds the buffer capacity of the order, the IORB is posted with a "long record" status (bit 6 of I_ST is 1).

### OPERATOR FUNCTIONS

The operator edits information at the terminal by using the following keys:

- Cursor control
- Character insertion/deletion
- Line insertion/deletion
- Line/screen erase.

The operator signals termination of input by pressing the TRANSMIT key. A break key enables the operator to interrupt a read order or to (possibly) terminate a write order.

### APPLICATION FUNCTIONS

An application selects the following options by setting bits in the device-specific word (I_DVS) of the IORB.

#### Abort Read

If this option is specified, ATD posts to the application any active and queued read IORBs. The posted IORBs show a device unavailable status (010B) in I_CT1 and the abort indicator (bit 0) in I_ST set to one. The read order issued with this option causes no I/O activity; it is posted back to the application with a zero status.

#### Supervisory Messages

Specification of this option indicates that the read order is directed to the supevisory message line. This option is meaningful only if the terminal is operating in no-roll mode. In no-roll mode, the supervisory message line is line 25. In roll mode, supervisory message reads are treated as normal reads.

#### Line Feed and Carriage Return

Specifying the line feed and/or carriage return option causes, respectively, a line feed and/or carriage return to be sent to the terminal when the read order is completed.

### READ IORB (BLOCK MODE)

An application specifies the options just described by setting bits in the IORB word I_DVS. Table 8-20 gives the significance of these bits. All other bits must be zero.

Table 8-20. ATD Word I_DVS in Block Mode Read IORB

| Bit Number | Meaning for Block Read Function |
|---|---|
| 0 | 0 = Normal read<br>1 = Abort read |
| 9 | 0 = Normal read<br>1 = Supervisory message read |
| 11 | 0 = Do not send post-order line feed<br>1 = Send post-order line feed |
| 12 | 0 = Send post-order carriage return<br>1 = Do not send post-order carriage return |

### Write Function (Block Mode)

The write order is used to transmit data blocks to the terminal.

#### WRITE ORDER PROCESSING

Write orders have priority over read orders. If a read order has been issued but is not in progress, any issued write order executes immediately. Once all outstanding write orders have completed, the outstanding read order is reestablished. If a read order is in progress (i.e., entry of data from the terminal has begun), the write order waits for the read to complete.

#### KEYBOARD LOCK

**The keyboard lock command is sent to VIP780x terminals only. All VIP781x terminals receive keyboard transmit lock commands.** Before the write order is executed by ATD, the LPH locks the terminal's keyboard. This action prevents processing conflict between the LPH and terminal. After the write order is processed, the keyboard is unlocked if the completed write order specified an ETX terminator (indicating the end of the message transmission to the terminal). If, however, the contents of the write order contains an escape sequence that elicits a response from the terminal, the device will ignore the keyboard unlock command; the application must issue another write order to unlock the keyboard.

#### WRITE ORDER OPTIONS

An application can specify the following options in I_DVS of the write IORB.

Abort Write

If this option is specified, ATD posts to the application any active and queued write IORBs. The posted IORBs show a device unavailable status (010B) in I_CT1 and the abort indicator (bit 0) in I_ST set to one. A write order issued with this option causes no I/O activity; it is posted back to the application with a zero status.

Preemptive Data Write

This option is meaningful only when the terminal is actively transmitting data. The option allows a write order to be processed between the transmission (by the terminal) of two ETB blocks or one ETB block followed by an ETX block. Normally, once a read operation is started by the application (to receive terminal transmissions), it is allowed to proceed (often requiring the issuance of several read IORBs) until the last text block (terminated by ETX) is received.

Control Byte Processing

Specification of control byte processing indicates that the first byte in the application's output buffer is to be used for pre-order control. A control byte must be included in the range (I_RNG) of data to be written to the terminal. For a detailed description of this option, including control byte format, refer to "Control Byte Processing" earlier in this section.

ETX/ETB Option

As mentioned earlier, ATD locks the keyboard during processing of a block mode write order. If the write order specifies ETB, indicating that a another block of the message is to follow, the keyboard remains locked after completion of the write order. Alternatively, if the write order specifies ETX, indicating the end of the message, the keyboard unlocks after completion of the order.

Quit On Break

If this option is specified in I_DVS, a break signal can interrupt the execution of an active write order. Otherwise, a break signal cannot be used to terminate an active write order prematurely.

Supervisory Messages

Specification of this option indicates that the write order is directed to the supervisory message line. This option is meaningful only if the terminal is operating in no-roll mode. In no-roll mode, the supervisory message line is line 25. In roll mode, supervisory message writes are treated as normal writes.

Supervisory Message Acknowledgement

If this option is specified, it indicates that a supervisory message written to a terminal is to be acknowledged by the terminal operator. Again, supervisory messages are meaningful only if the terminal has been connected in no roll mode. In roll mode, supervisory messages are treated as normal writes and the acknowledgement option does not apply. For a full discussion of this topic, refer to "Supervisory Message Processing" earlier in this section.

Line Feed and Carriage Return

Specifying the line feed and/or carriage return option causes, respectively, a line feed and/or carriage return to be sent to the terminal when the write order is completed.

## Write IORB (Block Mode)

This subsection summarizes the bit settings that govern the write order options already described. Table 8-21 gives the significance of bits of the write I_DVS word that are applicable to block mode. All other bits must be zero.

The bit settings in word I_ST are significant when a serial printer is attached to the terminal by means of a buffered printer adapter. On write orders, the field specifies whether the terminal or printer is being addressed. The permitted values are:

0 = Terminal
1 = Attached serial printer

## Device Configuration (Block Mode)

In block mode, the speed of a terminal must be configured between 110 and 9600 bits per second.

## Return Status Codes (Block Mode)

ATD returns status codes in I_CT1 and I_ST. The status code returned in I_CT1 is the right byte of the status returned in the R1 register (when the I/O order is completed).

STATUS CODES IN I_CT1

The invalid argument status (0104) is returned when an invalid block size is specified in I_RNG of a connect IORB.

The device unavailable status (010B) is returned when a read or write order is purged as a result of a purge-all read request or purge-all write request, respectively.

Table 8-21. ATD Word I_DVS in Block Mode Write IORB

| Bit Number | Meaning for Block Write Function |
|---|---|
| 0 | 0 = Normal write<br>1 = Abort write |
| 3 | 0 = Normal write<br>1 = Preemptive write |
| 4 | 0 = Include control byte<br>1 = Do not include control byte |
| 6 | 0 = ETX (unlock keyboard after write order completes)<br>1 = ETB (keep keyboard locked after write order completes) |
| 7 | 0 = Stop output on detection of a break<br>1 = Do not stop output on detection of a break |
| 8 | 0 = Operator must acknowledge supervisory message<br>1 = Operator need not acknowledge supervisory message |
| 9 | 0 = Normal write<br>1 = Supervisory message write |
| 11 | 0 = Do not send post-order line feed<br>1 = Send post-order line feed |
| 12 | 0 = Send post-order carriage return<br>1 = Do not send post-order carriage return |

The inconsistent request status (010C) is returned for a read order that is issued subsequent to a data loss. This status indicates that one or more data blocks were missed prior to the issuance of the current read order.

STATUS CODES IN I_ST

Table 8-22 shows status information returned in I_ST upon completion of a block mode order.

### Error Processing (Block Mode)

When a parity error is detected on a data transmission from the terminal, an ASCII SUB character (1A) is placed in the application's buffer in lieu of the erroneous character. The read order is posted with a hardware error status (0107), and bit 9 of I_ST is set to one to indicate that one or more parity errors were detected during the read.

Table 8-22. IORB Word I_ST (Block Mode)

| Bit | Meaning when Bit Set to One |
|---|---|
| 0 | Read or write order aborted |
| 1 | ETB received; (ETX received if bit off) |
| 3 | Block missed; was received from terminal without a read order having been issued |
| 6 | Long record received; buffer insufficient to contain received data |

Detection of a framing error or receive overrun condition prematurely terminates the read order. The order is posted with a hardware error status (0107); I_ST indicates the reason for abnormal termination.

## Timeout Processing (Block Mode)

In block mode, there are no timeouts for read or write orders.

## ASPI MODE

The ASPI (asyncronous serial printer interface) mode of ATD services selected serial printers that use an ETX/ACK protocol. It supports five functions, using standard-length IORBs:

Connect
Disconnect
Write
Read
Wait-on-line.

### ETX/ACK Protocol

Use of this protocol avoids a buffer overflow condition, in which an application transmits data to a device faster than the device can print the data. Buffer overflow is most likely to occur while the device is executing commands, such as carriage return or form feed, that move the print head or carriage. Without an ETX/ACK protocol, the application or device driver must pad data transmissions with fill characters, which the device does not print. While the fill characters are being edited out, the device has time to perform carriage returns or line feeds.

The ETX/ACK protocol renders padding unnecessary. Using this protocol, the LPH sends data to the printer a block or frame at a time. (The size of the block or frame depends on the buffering capacity of the device.) The LPH terminates the block with the ETX character. The serial printer responds with an ACK control character when (if the unit is double-buffered) it can accept another block or when it has sucessfully printed the last block of data. Having received the ACK control character, the LPH starts transmitting the next data block. The ASPI LPH supports a basic and advanced type of ETX/ACK protocol.

BASIC ETX/ACK PROTOCOL

The basic ETX/ACK is used by printers (the PRU1004 and PRU7007). It supports:

- A basic transmission procedure
- Detection of offline serial printer conditions by means of an attention read order
- Report of the printer's marketing identifier by means of a status read order.

ADVANCED ETX/ACK PROTOCOL

The advanced ETX/ACK is used by the PRU7070, PRU7170, PRU7270, PRU7175, PRU7200, PRU7210, and PRU7075 serial printers. It supports:

- An advanced ETX/ACK transmission procedure called the asynchronous serial printer interface (ASPI)
- Detection and report of all off-line serial printer conditions
- Report of the printer's marketing identifier and device status by means of a status read order.

<u>Connect Function</u>

An application selects the ASPI mode of ATD by setting bit 10 of I_DVS to one when issuing the connect order. When the device connected is a PRU7070, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7075, the LPH issues an enquiry to the device for status. The serial printer's response to the request for status allows ATD to specialize its processing to the characteristics of the device. If the device fails to respond to the request for status, ATD posts back the connect order with a device unavailable (010B) status. When connecting in ASPI mode, the application can specify the auto call option. Specifying auto call in I_DVS enables an application to establish a connection using either an 801-A or an 801-C ACU data set.

## Connect IORB (ASPI Mode)

Table 8-23 gives the significance of the I_DVS bits that govern the connect option. All other bits must be zero.

## Disconnect Function

An application uses the disconnect IORB to terminate ASPI mode processing. The following paragraphs describe the options that an application can specify with a disconnect order.

ABORT QUEUED ORDERS

If the abort option is specified, outstanding IORBs (active and queued) are terminated with a device unavailable status (010B). The disconnect order is immediately serviced. If the abort option is not specified, all outstanding IORBs are allowed to complete before the disconnect order is serviced.

HANG UP

If the hang-up option is selected, the terminal is physically disconnected when the disconnect order is serviced. If the hang-up option is not specified, the communications connection remains active after servicing the disconnect order (i.e., the terminal is logically disconnected, but remains physically connected).

## Disconnect IORB (ASPI Mode)

Table 8-24 gives the significance of the I_DVS bits that govern the disconnect options. All other bits must be zero.

## Write Function (ASPI Mode)

The write order is used to transmit data to the serial printer. Once the LPH has verified the buffer range and address in the IORB, it performs control byte processing (if specified in I_DVS). ATD then services the write request. The data written can be of any length; using the ETX/ACK protocol, ATD sends the data a block-at-a-time to the printer.

Table 8-23. I_DVS Word in Connect IORB (ASPI Mode)

| Bit Number | Meaning for Connect Function |
|---|---|
| 2 | 0 = Do not use auto dial<br>1 = Use auto dial |
| 10 | 0 = Do not select ASPI mode<br>1 = Select ASPI mode |

Table 8-24. I_DVS Word in Disconnect IORB (ASPI Mode)

| Bit Number | Meaning for Disconnect Function |
|---|---|
| 14 | 0 = Abort outstanding requests<br>1 = Wait until outstanding requests complete before disconnecting terminal |
| 15 | 0 = Hang up the phone<br>1 = Do not hang up the phone |

## CONTROL SEQUENCES

An application can control the write operation by means of control sequences imbedded in transmitted data.

### DC4 Control Sequence

Write orders to a PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, or PRU7270 support a feature that is useful to the application designer. If DC4 is the final character of an output message, the write order transmitting this message is posted back to the application only when the device has printed the entire message. If DC4 is not supplied at the end of the message, the write order is posted back when the printer (by means of an ACK response) declares itself ready to receive the last block of message data. Conceivably, the device could fail to print the last block after receiving it. Thus, the DC4 sequence provides assurance that the entire message is actually printed. By setting to 1 bit 14 of word I_DVS in the write IORB, an application can instruct ATD to append DC4 to the buffer contents.

### Other Sequences

An application can place in the data buffer the customary serial printer control characters (e.g., carriage return, line feed, horizontal tab). Other serial printer command and control sequences are available to the application. These can be used to change such printing characteristics as type pitch (number of characters per inch) and number of lines per inch. The user should consult the appropriate device manual for a more detailed discussion of printer control sequences.

### Prohibited Sequences

An application cannot place in the data buffer the ETX (X'03'), STX (X'02'), DC4 (X'14'), DLE EOT (X'10',X'04'), RIS (X'16',X'63') or ENQ (X'05') control characters, which are used by the ETX/ACK protocol.

WRITE OPTIONS

An application selects the following options by setting bits in the device-specific word (I_DVS) of the write order.

Control Byte

Through the use of a control byte, an application can specify the customary pre-order control operations. If present, the control byte is the first byte in the output buffer. The application indicates its presence by setting a bit in I_DVS. The application must also include the byte in the range (I_RNG) of the data to be transmitted. For a detailed description of the control byte option, including control byte format, see "Control Byte Processing" earlier in this section.

Line Feed and Carriage Return

Specifying in I_DVS the line feed and/or carriage return option causes, respectively, a line feed and/or carriage return to be sent to the printer when the write order completes.

Write IORB (ASPI Mode)

Table 8-25 gives the significance of bits of the write I_DVS word that are applicable to a ASPI mode write order. All other bits must be zero.

Table 8-25. ATD Word I_DVS in ASPI Mode Write IORB

| Bit Number | Meaning for ASPI Write Function |
|---|---|
| 1 | 0 = Not a link command<br>1 = Link command |
| 4 | 0 = Include control byte<br>1 = Do not include control byte |
| 11 | 0 = Do not send post-order line feed<br>1 = Send post-order line feed |
| 12 | 0 = Send post-order carriage return<br>1 = Do not send post-order carriage return |
| 14 | 0 = Do not send DC4 at end of message<br>1 = Send DC4 at end of message |
| 15 | 0 = Not a retry of a write IORB<br>1 = Retry of a write IORB |

## Read Function (ASPI Mode)

The read order is used to obtain status information from the serial printer. Two types of read orders can be issued: normal status read and attention status read. An application indicates in I_DVS the type of read desired.

### NORMAL STATUS READ

When an application issues a normal status read order, the IORB field I_ADR must point to a 10-byte buffer. Upon completion of the read order, this buffer contains a device identifier and may additionally contain status information. PRU1004 and PRU7007 printers provide a device ID in the first byte of the status buffer; the remaining bytes are unused.

PRU7070, PRU7170, PRU7175, PRU7200, PRU7210, PRU7270, and PRU7075 printers provide device status information in addition to the device ID, which is supplied in the first status byte. Refer to the appropriate serial printer manual for additional information on device status. Table 8-26 summarizes the device IDs that are returned in response to a status read request.

### ATTENTION READ

This option informs the application when a device has gone off-line or has been reset by the operator. The status buffer is not updated to reflect the device ID of the printer. If this option is specified, the read order is returned to the issuing application only when:

- The printer runs out of ribbon
- The printer runs out of paper
- The printer's break switch is pressed.

Table 8-26. Device IDs for Serial Printers

| Printer | Device ID (hexadecimal) |
|---|---|
| PRU1004 | 21 |
| PRU7007 | 22 |
| PRU7070 | 31 |
| PRU7075 | 32 |
| PRU7170 | 33 |
| PRU7175 | 34 |
| PRU7200 | 43 |
| PRU7270 | 38 |

## Read IORB (ASPI Mode)

Table 8-27 shows the signficance of bits of the I_DVS word that are applicable to a ASPI mode read order. All other bits must be zero.

## Status Codes Returned in I CT1 (ASPI Mode)

ATD returns status codes in I_CT1 and I_ST. The status code returned in I_CT1 consists of the right byte of the status returned in the R1 register (when the I/O order completes). A status code often has more than one possible meaning. As explained later in "Status Information under I_ST", a user can determine a specific meaning by referring to word I_ST. For example, the status 0104, in itself, can mean zero buffer address or zero buffer range. If bit 13 of I_ST is set to 1, the status 0104 means zero buffer address. If, however, bit 14 of I_ST is set to 1, the status 0104 means zero buffer range.

SUCCESSFUL COMPLETION (0000)

A zero status (in I_CT1) indicates successful completion of the order. A write order IORB can additionally indicate (in I_ST) a device attention condition (initiated by the operator) on PRU7170, PRU7175, PRU7200, PRU7210, and PRU7270 printers. This condition in no way interferes with successful completion of this or subsequent orders placed against the printer. By initiating a device attention condition, the operator can directly interact with the application that is controlling the serial printer.

INVALID ARGUMENT STATUS (0104)

This status is returned for the following reasons:

- In write IORB
  - Zero buffer address.
  - Zero buffer range.
- In read IORB, read buffer less than 10 bytes long.

Table 8-27. ATD Word I_DVS in ASPI Mode Read IORB

| Bit Number | Meaning for ASPI Mode Read Function |
|---|---|
| 0 | 0 = Normal status read<br>1 = Attention status read (PRU7170, PRU1004, PRU7007, PRU7175, PRU7200, PRU7210, and PRU7270) |

DEVICE NOT READY STATUS (0105)

An order is posted back with this status when a PRU7070, PRU7075 PRU7170, PRU7175, PRU7200, PRU7210, or PRU7270 printer is in an offline state. The LPH issues this error status once; subsequent or outstanding write orders are serviced when the device is put in an online, operational state. The reported offline condition is usually caused by the OFF-LINE button being pressed or by the printer running out of ribbon or paper.

HARDWARE ERROR STATUS (0107)

This status is returned in a write order for the following reasons:

- Hardware printer fault (PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210)

- Failure of printer to respond to print or status commands (PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210).

Status Information in I_ST

The bit settings in I_ST qualify the status codes returned in I_CTl, as shown in Table 8-28. The first column of the table gives the bit in I_ST; the second column gives the status code returned in I_CTl; the third column shows the significance of the status code (column 2) when the I_ST bit (column 1) is set to one.

## Error Processing

An application should retry the write order after an 0105 error (device not ready) has been returned. An application should assume that the device will fail to complete any print operation when a 0107 (hardware error) status is reported. The application must disconnect if it receives a 0170, 010B, or 010C error in the IORB.

## Timeout Processing

There is a default 15-second timer for write orders. This timer can be changed in the CLM USER file.

## X-ON/X-OFF MODE

The X-ON/X-OFF mode of ATD, called AXD (Asynchronous X-ON/X-OFF Driver), uses the X-ON/X-OFF protocol to control asynchronous data flow on an asynchronous communication line. With AXD, applications and devices are protected from losing data on buffer-full conditions. AXD is used with asynchronous devices that support the X-ON/X-OFF protocol, including serial printers, terminals, personal computers, and paper tape readers. If a

Table 8-28. IORB Word I_ST (ASPI Mode)

| Bit | Return Status | Meaning When Bit Set to 1 |
|---|---|---|
| 2 | 0000 | Operator initiated attention (PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210) |
| | 0105 | Device offline (PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210) |
| | 0107 | Hardware printer fault (PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210) |
| 13 | 0104 | Zero buffer address |
| | 0105 | Paper out (PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210) |
| | 0107 | No response by device to print or status commands (PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210) |
| 14 | 0104 | Zero buffer range |
| | 0105 | Ribbon out (PRU7070, PRU7075, PRU7170, PRU7175, PRU7200, PRU7270, or PRU7210) |

device does not support X-ON/X-OFF, no inherent data loss protection is provided by the driver. AXD can still be used if the device is insensitive to receiving X-ON and X-OFF characters from AXD (most Honeywell terminals operate this way). The use of X-ON/X-OFF by AXD cannot be disabled.

AXD can be used as a facility for file transfers between systems. The degree of transparency on received data is selectable with AXD. AXD is especially useful with simple file transfer applications (such as the use of the Line Editor to receive a file from a personal computer) because the flow control is provided by the driver. Applications that have built-in flow control have an extra layer of protection when using AXD.

To support diverse types of operations, AXD provides different operational modes -- TERMINAL mode, PRINTER mode, FILETRAN mode, and RAW mode. The modes are mainly distinguished by the way they process received data. All modes except RAW provide X-ON/X-OFF protection in both directions. RAW mode has flow control only when receiving.

## X-ON/X-OFF Protocol

The X-ON/X-OFF protocol is a standard means of controlling data flow over an asynchronous communications line. Two special characters are used under the protocol to control data flow, DC3 (hex 13), and DC1 (hex 11). The DC3 means "suspend transmission" (X-OFF), the DC1 means "resume transmission" (X-ON). When incoming data cannot be processed (because there is no read order or the input buffer is almost full), an X-OFF is issued to the sending side. Upon receipt of the X-OFF, the sender suspends transmission. When the receiver has a favorable buffer condition allowing it to again accept data, it sends an X-ON to resume the flow. These actions occur invisibly to the application.

For an application to receive data, it must supply AXD with read orders. If data comes across the line while there is no read order, AXD saves the character(s) in its reserve buffer and sends an X-OFF to stop the sender. If the sender does not stop within seven characters, subsequent data is lost and the application is notified with an error status on the next read, or simply with a bell signal in terminal mode. Since DC1 and DC3 are special protocol characters, they must not exist in the data being received unless they are "hidden" using the AXD hide function or some other method.

## SUPPORTED IORB TYPES

AXD supports five functions, using standard-length IORBs: Connect, Disconnect, Read, Write, and Break.

### Connect Function

The following paragraphs describe the AXD options that an application can specify with a connect order. References to IORB offsets such as I_CT1 and I_DVS are depicted in Figure 8-1.

#### AUTO CALL

Specifying auto call in I_DVS enables an application to establish a connection using either an 801-A or an 801-C ACU data set. *

#### PROCESS READS ASYNCHRONOUSLY

This option prevents incoming data from being lost between read orders when the sending device does not support X-ON/X-OFF. With this option, an application can issue up to four consecutive asynchronous read orders (W-bit of I CT2 set to 1 in the read IORB). AXD immediatly sends each read to the communication controller to wait for incoming data. When a read completes, it is posted to the application with appropriate status and the next read on the controller is instantly ready to receive.

While the maximum of four simultaneous reads are on the controller, any additional reads issued by the application wait on AXD's queue until an earlier read completes. With the Asynchronous Read option, the device specific word of the first issued read order also serves as the device specific word for all subsequent reads until the end of the file is reached.

If this option is not selected, read orders are processed by AXD synchronously (one at a time), even if issued asynchronously by the application and each uses it's own device specific word. This option is not available in TERMINAL mode of AXD.

AXD MODE

The mode is chosen to suit the type of operation being performed during this connect. The modes are:

- TERMINAL mode, for interactive, line-at-a-time, processing on a teletype-compatible (TTY device)
- PRINTER mode, for sending to a receive-only device (such as a serial printer or a graphics plotter)
- FILETRAN mode, for either-direction file transfer with control character support on input
- RAW mode, for completely transparent input processing.

SOLICIT INITIAL TRANSFER

When this option is selected, AXD issues an X-ON to the sender, (to indicate initial readiness to receive), when the application issues its first read order.

REQUIRE INITIAL X-ON

When this option is selected, AXD holds execution of the application's first issued write order until an initial X-ON comes in from the receiver. Be aware that the write order is subject to timeout while waiting for the X-ON. This option must not be used with the RAW mode of AXD.

LOGICAL CONNECT

This option specifies that a logical re-connect of the line is to be done. This should only be done after a logical disconnect.

## Connect IORB (AXD Mode)

Table 8-29 gives the significance of the I_DVS bits that govern the connect options already described. All other bits must be zero.

Table 8-29. I_DVS Word in Connect IORB (AXD Mode)

| Bit Number | Meaning for Connect Function |
|---|---|
| 1 | 0 = Do not use AXD<br>1 = Use AXD |
| 2 | 0 = Do not use auto dial<br>1 = Use auto dial |
| 3 | 0 = Process read orders synchronously<br>1 = Process read orders asynchronously |
| 5-7 | 100 = Use PRINTER mode of AXD<br>011 = Use FILETRAN mode of AXD<br>010 = Reserved<br>001 = Use TERMINAL mode of AXD<br>000 = Use RAW mode of AXD |
| 8 | 0 = Do not solicit initial transfer<br>1 = Solicit initial transfer |
| 9 | 0 = Do not require initial X-ON<br>1 = Require initial X-ON |
| 15 | 0 = Do a physical connect<br>1 = Do a logical connect |

## Disconnect Function

An application uses a disconnect IORB to terminate AXD mode processing. The following paragraphs describe the options that an application can specify with a disconnect order.

### ABORT QUEUED ORDERS

If the abort option is specified, outstanding IORBs (active and queued) are terminated immediately and returned with a device unavailable status (010B). The line is then disconnected. If the abort option is not specified, all outstanding IORBs are allowed to complete before the disconnect order is serviced.

### HANG UP

If the hang-up option is selected, the device is physically disconnected when the disconnect order is serviced. If the hang-up order is not specified, the communications connection remains active after servicing of the disconnect order (i.e., the device is logicaly disconnected, but remains physically connected).

END-OF-FILE/DISCONNECT

If the end-of-file/disconnect option is selected, AXD sends an EOF character prior to processing the disconnect order. This option can be used, for example, when sending a file via the Copy utility.

## Disconnect IORB (AXD Mode)

Table 8-30 gives the significance of the I_DVS bits that govern the disconnect options. All other bits must be zero.

Table 8-30. I_DVS Word in Disconnect IORB (AXD Mode)

| Bit Number | Meaning for Disconnect Function |
|---|---|
| 13 | 0 = Do not send EOF character before disconnect<br>1 = Send EOF character before disconnecting |
| 14 | 0 = Abort outstanding requests<br>1 = Wait until outstanding requests complete before disconnecting |
| 15 | 0 = Hang up the phone<br>1 = Do not hang up the phone |

## Read Function

The following paragraphs describe the options an application can specify with a read order. Note that these read order attributes supplement those defined by the AXD mode chosen in the connect order.

ECHO

If this option is specified, received data is echoed or "reflected" back to the transmitter. In TERMINAL mode, only the data characters are echoed. The other AXD modes echo all data and control characters received for verification purposes. If the echo option is not specified, no echo is performed.

LINE FEED

If this option is selected, a line feed is sent to the terminal upon completion of a read order. If this option is not selected, no post order line feed is sent. This option is available in TERMINAL mode only.

CARRIAGE RETURN

If this option is selected, a carriage return is sent to the terminal upon completion of a read order. If this option is not selected, no post order carriage return is sent. This option is available in TERMINAL mode only.

Read IORB (AXD Mode)

An application specifies the read options by setting bits in the I_DVS word of the read IORB. Table 8-31 gives the individual significance of these bits. All other bits must be zero.

Write Function

All AXD modes process writes the same way, except that in RAW mode, writes cannot be suspended with an X-OFF. Write order attributes are determined by the write IORB. The following options are specified in the write IORB.

EDITED WRITE

If the edited option is specified, all control characters found in the output buffer are sent out preceeded by a DLE character. The following are considered control characters on an edited write: DC1 (hex 11), DC3 (hex 13), DLE (hex 10), ESC (hex 1B) PAD (value is configurable), END-OF-RECORD (value is configurable), END-OF-FILE (value is configurable), CHARACTER-DELETE (value is configurable), and LINE-CANCEL (value is configurable)

CONTROL BYTE PROCESSING

If specified, the control byte option indicates that the first byte in the output buffer is to be used for pre-order control. A control byte must be included in the range (I_RNG) of data to be transmitted. For a detailed description of this option including control byte format, see "Control Byte Processing" earlier in this section.

SUPPRESS TRAILING BLANKS

If this option is specified, space characters (hex 20) found between the last non-space and end of range, are not sent.

Table 8-31. I_DVS Word in Read IORB (AXD Mode)

| Bit Number | Meaning for Read Function |
|---|---|
| 10 | 0 = Do not echo input<br>1 = Echo input |
| 11 | 0 = Do not send post-order line feed<br>1 = Send post-order line feed (TERMINAL mode) |
| 12 | 0 = Send post-order carriage return (TERMINAL Mode)<br>1 = Do not send post-order carriage return |

UNBREAKABLE WRITE

If this option is specified, a break signal cannot interrupt the execution of the write order. If a break occurs during an "unbreakable write", the order runs to normal completion but the next write is automaticaly posted back with break status unless it too is an "unbreakable write." If this option is not selected, a break signal can be used to prematurely terminate an active write order.

LINE FEED

If this option is specified, a line feed is sent after the completion of the write order.

END OF RECORD

If this option is specified, an end of record character is sent after the completion of the write order.

END OF FILE

If this option is specified, an end of file character is sent after the completion of the write order.

## Write IORB (AXD Mode)

An application specifies the write options by setting bits in the I_DVS word of the write IORB. Table 8-32 gives the significance of these bits. All other bits must be zero.

## Control Characters

Control characters cause special actions to occur when received in the input data stream. Each mode of AXD supports a different subset of control characters. A control character not supported in a particular mode is taken as data. The AXD control characters are as follows:

| Control Character | Default Value | Supporting AXD Mode |
|---|---|---|
| X-OFF | DC3 (hex 13) | TERMINAL, FILETRAN, PRINTER |
| X-ON | DC1 (hex 11) | TERMINAL, FILETRAN, PRINTER |
| End of Record | CR (hex 0D) | TERMINAL, FILETRAN |
| End of File | FS (hex 1C) | PRINTER, FILETRAN |
| Pad | DEL (hex 7F) | TERMINAL, FILETRAN |
| Hide | DLE (hex 10) | TERMINAL, FILETRAN |
| Char-Delete | ESC,D (hex 1B44) | TERMINAL |
| Line-Cancel | ESC,K (hex 1B4B) | TERMINAL |

All control character values except X-ON, X-OFF, and Hide can be changed by using the STTY command.

Table 8-32. I_DVS Word in Write IORB (AXD Mode)

| Bit Number | Meaning for Write Function |
|---|---|
| 2 | 0 = Do not edit this write<br>1 = Edited write: Preceed contol chars with DLE |
| 3 | 0 = Do not send post-order EOF<br>1 = Send post order EOF |
| 4 | 0 = First byte is pre-order control byte<br>1 = First byte is data |
| 6 | 0 = Do not supress trailing blanks<br>1 = Supress trailing blanks |
| 7 | 0 = Allow write to be broken with break key<br>1 = Defer break to next write |
| 11 | 0 = Do not send post-order line feed<br>1 = Send post-order line feed |
| 12 | 0 = Send post-order end-of-record character<br>1 = Do not send post-order end-of-record character |

The following paragraphs describe each of the control character functions:

X-OFF

When AXD receives an X-OFF character, it suspends output until an X-ON character is received.

X-ON

When AXD receives an X-ON character, output is allowed to begin/resume.

End of Record

When AXD receives an end-of-record character, the current read IORB is terminated and posted back to the application with a normal return status of zero and a non-zero residual range status in I_ST1.

End of File

When AXD receives an end-of-file character, all outstanding read orders are terminated and posted back to the application with end of file status (OF), and non-zero residual range status in I_ST1.

Pad

When AXD receives a pad character, it is simply discarded (i.e., it is not stored in the application's input buffer). No other action is taken.

Hide

When AXD receives a hide character, it is recognized as a hide character and then discarded. The next character received is stored in the application's input buffer, regardless of that character's possible control meaning (i.e. it is treated as data). In TERMINAL mode, a backslash character (\) is echoed to the terminal to indicate the next key typed will be taken as data.

Character-Delete

Receiving the character delete sequence causes AXD to send a backspace/space/backspace sequence to erase from the screen the character last entered.

Line-Cancel

If the configured line-cancel code is not a two-character sequence beginning with hex 1B (i.e., if it is a single character), AXD displays on the terminal *DEL* and puts the cursor on a new line. Otherwise, AXD sends the exact 1Bxx sequence it received to erase the line and put the cursor back to the leftmost position. In either case, AXD reissues the read order, using the original buffer and range. AXD does not issue the *DEL* sequence on systems with MLCP communications controllers.

## Escape Sequence Processing

AXD supports escape sequences in TERMINAL mode only. An escape sequence is a string of one to seven characters starting with hex 1B. The only escape sequences AXD recognizes as its own are LINE-CANCEL and CHARACTER DELETE. Anything else is considered a user escape sequence and is processed as follows. The 1B and next immediate character are stored in the application's input buffer and the read is terminated. The remainder of the sequence (up to 7 characters), is saved in AXD's reserve buffer until the application can put up a read of sufficient length to capture the rest.

## Timeout Processing

When a read or write order times-out, that order and all other outstanding I/O orders are posted back to the application with timeout status (06).

READ TIMEOUTS

In FILETRAN mode,and RAW mode, the timeout timer is started when the read order is issued. A timeout occurs if the read does not complete in time. The default timeout value for read orders is five minutes but can be changed with the TIMEOUT CLM directive (see the System Building and Administration manual). There is no timeout on read orders in TERMINAL mode and PRINTER mode.

WRITE TIMEOUTS

In all AXD modes, a timeout timer begins when a write order is issued. The default timeout value for write orders is thirty seconds. The timeout can be changed using the TIMEOUT CLM directive.

## Error Processing

When a parity, frame, or receive-overrun error is detected, an ASCII SUB character (hex 1A) is stored in place of the received character that was in error. The read is allowed to run to a normal completion, but is posted with a hardware error (07) status; I_ST identifies which error occurred. Table 8-33 shows the meaning of the status bits in I_ST.

## Data Loss

Data loss can occur on receive if the sender ignores an X-OFF from AXD and continues to send beyond AXD's reserve buffer capacity (seven characters). The next read order will take on the saved characters and complete as normal, but will be posted with a hardware error (07) status, and I_ST will have bit 8 on, indicating that data loss has occurred. In TERMINAL mode, data loss is indicated with a Bell signal, not a hardware error.

## Break Processing

A break is initiated when a terminal's BREAK (BRK) key is pressed, or when certain devices are put off-line or powered off. The BREAK key can be changed by using the STTY command. AXD detects and reports break only when there is a read or write order running at the time of the break. AXD break processing is described under "Break Processing by ATD LPH", earlier in this section.

## Hardware Requirements

AXD requires a full-duplex communication line with an RS-232 or RS-422 interface. The line can be direct connect or through a modem. The maximum data transfer rate is 9600 bits per second.

Table 8-33. Status Word of AXD IORB (I_ST)

| I_ST Bit | Meaning When Bit Set to 1 |
|---|---|
| 0 | N/A |
| 1 | I/O order complete |
| 2 | Data service rate error (receive overrun) |
| 3 | Status complete |
| 4 | Communication control block service error |
| 5-7 | Cause of termination:<br><br>101 = Break<br>100 = End of file<br>011 = Line canceled<br>010 = Character deleted<br>001 = Timeout |
| 8 | Receive buffer overflow -- data lost |
| 9 | Parity error |
| A | Nonzero residual range (read only) |
| B | Data set status change |
| C | Corrected memory error |
| D | N/A |
| E | N/A |
| F | Fatal error<br><br>• Unrecoverable memory error<br>• Bus parity error<br>• Nonexistent resource error |

*

## AXD Operational Modes

AXD offers a choice of operational modes to accommodate different applications. Table 8-34 is a quick reference guide to the key features and control characters each mode supports. In the table, a "YES" means that the control character is supported in that mode. A "NO" means the control character is treated as data when received. While operating in AXD mode:

- All AXD modes support both reads and writes per connect (bi-directional I/O). For example, PRINTER mode can do reads and RAW mode can do writes.
- All AXD modes process write orders the same way. What differentiates the AXD modes is the way reads are processed.
- All AXD modes except RAW provide flow control in both directions. On incoming data, AXD always tries to stop the sender with an X-OFF when there is no read buffer available. When sending (except in RAW mode), AXD suspends transmission upon recipt of an X-OFF from the receiver.

Table 8-34. AXD Modes and Features

| Feature/ Control Character | Mode | | | |
|---|---|---|---|---|
| | TERMINAL | FILETRAN | RAW | PRINTER |
| File Transfer Functionality | TRANSMIT | RECEIVE/ TRANSMIT | RECEIVE | TRANSMIT |
| Character-Delete | YES | NO | NO | NO |
| Line-cancel | YES | NO | NO | NO |
| Hide | YES | YES | NO | NO |
| Pad | YES | YES | NO | NO |
| End-of-Record | YES | YES | NO | NO |
| End-of-File | NO | YES | NO | YES |
| Escape Sequences | YES | NO | NO | NO |
| Break | YES | YES | YES | YES |

The following paragraphs describe the uses of the AXD modes.

## TERMINAL Mode

TERMINAL mode is for use with interactive applications such as the Command Processor running on a TTY-compatible device. To aid in data entry operations on the terminal, this mode supports escape sequences and the operator function keys Character-Cancel, Line-Delete, Input-Terminator, Hide, and Break.

TERMINAL mode looks to the user much like the TTY mode of ATD. One difference, however, is the acceptance (without echo) of up to seven typed characters when there is no read for the terminal. After seven characters have been accepted, subsequent keystrokes receive the Bell signal. Saved characters are ultimately displayed when a read order is issued. Unlike ATD, the TERMINAL mode of AXD has no timeout on terminal read orders.

On most terminals, the user can stop and restart output to the screen by typing X-ON and X-OFF characters as follows:
X-ON = Control-Q, X-OFF = Control-S.

## FILETRAN Mode

FILETRAN mode is the recommended mode for receiving data non-transparently from a device or another system using AXD. In this mode, the ECHO feature can optionally be used to validate to the sending application that data was received correctly.

## PRINTER Mode

PRINTER mode is used to send files to a device which supports X-ON/X-OFF. AXD does not provide for the detection of device conditions such as off-line or paper-out. The application must provide reads for such status. Read orders in PRINTER mode terminate upon receiving an end-of-file character or at end of range. (The end-of-file character can be configured to be any character).

## RAW Mode

RAW mode is used to receive data transparently from a device or another system using AXD. Everything received is put into the application's input buffer. No control character checking is done. Read orders in this mode terminate at end of range only. RAW mode should not be used to write to a device that could send X-ON (DC1) and X-OFF (DC3), as they would be taken as data.

# Section 9
# SYNCHRONOUS TERMINAL DRIVER LINE PROTOCOL HANDLER

The Synchronous Terminal Driver (STD) line protocol handler (LPH) supports synchronous polled terminals, asynchronous receive-only printers (ROPs), and the PVE host link.

The basic VIP consists of a cathode ray tube (CRT) display screen and keyboard, with a synchronous communications interface. Its operating speeds are as follows:

| Device Type | Peripheral | Baud Rate |
|---|---|---|
| VIP7700 | ROP | 2000 to 4800 |
| VIP7700R/VIP7705R | ROP | 2000 to 9600 |
| VIP7804/VIP7805 | ROP | 2000 to 9600 |
| VIP7760 | ROP, DSK | 4800 to 9600 |
| VIP7740 | ROP, DSK | 9600 |
| VIP7710 | ROP | 9600 |
| VIP7814/VIP7815 | ROP/ASPI | 2400 to 9600 |
| VIP7816/VIP7817 | ROP/ASPI | 2400 to 9600 |
| VIP7824/VIP7825 | ROP/ASPI | 2400 to 9600 |
| VIP7826/VIP7827 | ROP/ASPI | 2400 to 9600 |
| TWU1901 | | 4800 to 9600 |

Receive-Only Printers

| | | |
|---|---|---|
| PRU1003 | PRU1901 | TN300 |
| PRU1005 | TN1200 | |

The PVE host link is operated at baud rates between 1200 and 19200.

## GENERAL STD LINE PROTOCOL HANDLER OPERATION

### Software Functional Support for the VIP

The following STD line protocol handler software functions support the basic VIP terminal:

- Poll and select communications procedures
- Poll line control
    - Poll list
    - Poll interval
    - Poll list stall interval
- Self configuration support for CRT devices
- Multipoint configuration support
- Switched and private line operation
- Auto-answer for switched network operation
- Modem, direct connect, and modem bypass interconnection modes
- 7-bit and 8-bit data support with appropriate algorithms for 8-bit data transmission on a 7-bit channel
- Message/block transfer to and from a CRT
- Master LRN processing
- Fully addressable CRT entry marker control
- Pre-editing (control byte) and post-editing (I_DVS)
- Transfer of hardware function code to and from the application
- Long Q frame
- Inactivity timeout
- Error recovery procedures
- Break processing (VIP7800 series only)
- Synchronous modem support for baud rates of less than 2000
- Half-duplex line function
- 2/4 wire line function.

The following functions support added terminal options:

- User-controlled CRT forms mode
- Message/block transfer to receive-only printer (ROP)
- User-controlled storage and retrieval of forms on the diskette (7740 and 7760 only).

## User-Supplied Software Functions for VIP Support

The application program must supply the following functions to support data exchange between the terminal and the application:

- User-specified device arguments (polling interval and, at system building, station addresses and device type).
- If the VIP is self-configuring, an asterisk (*) replaces the device type argument.

For messages to the VIP terminal, the application should provide:

- Optional; hardware function codes (1, 2 for all VIP except 7800 series, which only uses 1)
- Complete message text, including all required format control characters
- Optional; pre-editing and post-editing characters within message text
- Mandatory; complete forms definition message text for forms mode.

For messages received from the VIP terminal, the application must provide:

- Interpretation of hardware function codes (1, 2 for all except VIP7800 series, which only uses 1)
- Message processing (complete message or block, with possible use of master LRN with either)
- Interpretation of format codes (LF, CR, HT, VT) in the message text.

## STD Request Response Time

Table 9-1 shows how to calculate the request response times needed by the line protocol handler for the connect, read, and write functions for the listed devices.

Table 9-1. STD Line Protocol Handler Response Time

| Function | Response Time | Device |
|---|---|---|
| Connect | 5-minute timeout | Communications supervisor |
| Read or Write | The equation to calculate the time required to send/receive a message to/from the CRT is:<br><br>$M/C = T$<br><br>where:<br><br>M = Message size (range)<br>C = Number of characters per second (line speed). Possible values for C:<br>150 = 1200 baud<br>225 = 1800 baud<br>250 = 2000 baud<br>300 = 2400 baud<br>600 = 4800 baud<br>1200 = 9600 baud<br>2400 = 19.2 kilo baud<br>T = Timeout value in seconds<br><br>NOTE<br><br>If M is less than or equal to C, then T = 2. | All devices |
| | The equation to calculate the time required to send/receive a message to/from the ROP attached to a CRT is:<br><br>$T + (M/R) = V$<br><br>where:<br><br>T = CRT timeout value (from above)<br>M = Message size (range)<br>R = ROP transmit rate. Possible values for R:<br>10 = 100 baud<br>30 = 300 baud<br>120 = 1200 baud<br>V = Total timeout value in seconds | ROPs attached to VIPs |

## USING THE STD LINE PROTOCOL HANDLER

### STD-Specific IORB Values

The VIP-specific input/output request block (IORB) item I_CT2, device specific word I_DVS, and software status word I_ST are shown in Tables 9-2, 9-3, and 9-4, respectively. Bits not explicitly described in the tables must be 0. Section 4 describes the general form of the IORB.

Table 9-2. Function Codes in I_CT2 of the IORB

| Function Code | Definition | Use |
|---|---|---|
| 1 | Write | Used by the line protocol handler to complete the description of the requested I/O function. |
| 2 | Read | |
| A | Connect | |
| B | Disconnect | |

Table 9-3. STD Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| | For connect call only (function code A). |
| 0 | 0 = No meaning<br>1 = Terminal-generated block mode |
| 2 | 0 = Do not use Auto Call Unit<br>1 = Use Auto Call Unit |
| 3 | 0 = Set cursor to home position on page overflow (write request). (Not applicable to VIP7800 series.)<br>1 = Do not set cursor to home position on page overflow (write request). (Not applicable to VIP7800 series.) |
| 4 | 0 = Control word specified (read/write request).<br>1 = No control word specified (read/write request). |

Table 9-3 (cont). STD Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| 5, 6, and 7 | Logical poll interval (read request, polled lines only):<br>000 = Poll continuously<br>001 = 1-second poll interval<br>010 = 2-second poll interval<br>011 = 3-second poll interval<br>100 = 4-second poll interval<br>101 = 5-second poll interval<br>110 = 15-second poll interval<br>111 = 30-second poll interval |
| 8 | 0 = No space suppress (VIP7800 series only)<br>1 = Space suppress (VIP7800 series only) |
| 9 | 0 = Roll (VIP7800 series only)<br>1 = No roll (VIP7800 series only) |
| A | For ROP attached to VIP7700, VIP7700R, and VIP7760:<br>0 = 150/PRT ROP address.<br>1 = 150/NUL ROP address. |
| B | 0 = Hardware function codes are not specified (write request).<br>1 = Hardware function codes are specified (write request). (Not allowable for VIP7800 series.) |
| C | 0 = Do no timeout, use logical poll interval (read request).<br>1 = Timeout immediately (read request). |
| D | 0 = Return key equals transmit (VIP7800 series only).<br>1 = Return key equals normal (VIP7800 series only). |
| | For write call only (function code 1) |
| 0 | 0 = No meaning<br>1 = Abort write IORB subfunction |
| 3 | 0 = No preemptive write<br>1 = Preemptive write |
| 4 | 0 = Include control byte<br>1 = Do not include control byte |
| 5 | Reserved for system use (must be zero) |
| 6 | Reserved for system use |

Table 9-3 (cont). STD Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| 8 | 0 = No meaning<br>1 = If bit 9 = one, supervisory write with reset; otherwise, no meaning. |
| 9 | 0 = Normal message<br>1 = Supervisory message, if no roll on connect. |
| A | RFU |
| B | 0 = No line feed at end of message<br>1 = Line feed at end of message |
| C | 0 = Carriage return at end of message.<br>1 = No carriage return at end of message. |
| D, E, and F | Number of copies to be printed (VIP7800 series only):<br>000 = 1 copy<br>001 = 2 copies<br>010 = 3 copies<br>011 = 4 copies<br>100 = 5 copies<br>101 = 6 copies<br>110 = 7 copies<br>111 = 8 copies |
| For read call only (function code 2) | |
| 0 | 0 = No meaning<br>1 = Abort read IORB subfunction |
| 2 | 0 = No meaning<br>1 = ESC B Diskette request |
| 9 | 0 = Normal message<br>1 = Supervisory message, if no roll on connect. |
| For disconnect call only (function code B) | |
| 1 | 0 = No meaning<br>1 = Send DLE EOT (VIP7800 series only) |
| E | 0 = Purge outstanding requests and disconnect immediately.<br>1 = Wait until all requests are complete before disconnecting. |
| F | 0 = Hang up phone after disconnect.<br>1 = Maintain phone connection after disconnect. |

Table 9-4. STD Software Status Word I_ST in the IORB

| Bit | Contents of $R1 | Meaning When Bit is Set to 1 |
|---|---|---|
| 1 | 0 | ETB received |
| 2 | 0 | Data service error (transmit) |
| 6 | 0 | Long record received (receive) |
| 7 | 0 | Illegal character (transmit) |
| 8 | 0 | Sequence error (receive) |
| E | 104 | Range error |
| 7 | 106 | Read timeout |
| 8 | 107 | NAK limit reached |
| D | 107 | Page overflow |
| F | 107 | Busy |
| 0 | 10B | Abort |
| 2 | 10B | Data service error (receive) |
| 7 | 10B | Illegal character (receive) |
| 8 | 10B | Poll failure/sequence error |
| 9 | 10B | Excessive checksum/parity errors (receive) |
| B | 10B | Phone hang up |
| 3 | 10B | Read timeout |
| E | 10B | Not available |

## STD Self Configuration

The device type can be defined as self-configuring by using an asterisk (*) as the device type argument of the STD directive at configuration time. In this way, the STD line protocol handler will configure the device when the first connect is recieved for the device. Self-configuration is performed by sending an ENQ command to the device and setting the configuration based on the response. The response may be the device name in the case of VIP7800 series devices or a Q-frame for VIP7700 series devices. If an unknown resonse is made, the device is configured as a VIP7700. If the device doesn't respond, the ENQ command is tried 3 more times. If the device doesn't respond at the end of the fourth try, the connect is returned to the user with the device status being unavailable.

## STD Polling Options

Polling (the line protocol handler's request to the VIP terminal on a polled line for data) is subject to four kinds of control: two specified at system build, and two specified at connect time. The former consists of the poll lists and poll list stall, while the latter are the poll interval and poll duration.

The application, at connect time, is required to specify the arguments for the poll interval and poll duration, by setting the appropriate bits in the IORB's device-specific word I_DVS (Table 9-3).

STD POLL LIST

The poll list specifies the station addresses to be used in the polling sequence. Multiple occurrences of a particular address may be used to increase the polling frequency of that address. The list is defined at system build (see the System Building and Administration manual).

STD POLL LIST STALL

Poll list stall is the delay interval, in seconds, between poll list cycles. This delay is specified at system build (see the Building and Administration manual.

STD POLL INTERVAL

The poll interval specifies the minimum period of time between each successive request (poll) by the line protocol handler for data from a VIP terminal. The line protocol handler will poll the VIP once for each read request, and when the request is not satisfied, again after the specified poll period elapses.

For example, with a 1-second poll interval, the line protocol handler will issue the same read request every second. For a zero poll interval, the line protocol handler will poll the VIP terminal continuously.

The application specifies the poll interval according to the bit settings of bits 5, 6, and 7 in the device-specific word I_DVS of the IORB, as listed in Table 9-3.

STD POLL DURATION (TIMEOUT)

Poll duration, or the timeout interval, is the maximum time that the line protocol handler will wait for polled data from the VIP, before discontinuing the read attempt and read request. The possible timeout intervals are immediate (i.e., after only one poll) and indefinite (i.e., until requested data is received). The application specifies the poll duration or timeout interval with the bits 5, 6, 7, and C in the connect device-specific word I_DVS, according to the bit values shown in Table 9-3.

STD LINE PROTOCOL HANDLER POLL FUNCTIONS

Within the parameters specified in the poll argument values by the application, the line protocol handler provides all necessary polling functions (e.g., how terminals share a common line, or which terminal is processed next based on the poll list).

When the application bypasses these line protocol handler poll functions (i.e., by specifying immediate timeout after only one poll), the application must then provide for proper operation and coordination among all terminals on the line.

When the application is to issue to the terminal (VIP7800 series) writes containing TXA or TXD escape sequences, the user should first issue an asynchronous read. The use of immediate timeouts on reads in this case could cause the read to be issued and posted before the write is queued and issued, resulting in a loss of data from the TXA or TXD command.

Polling is defined as the actual read, not the reading of the poll list. Polling itself does not commence unless a read has been queued. Only those stations on the poll list which have reads queued will be polled.

## Control and Characterisitcs of STD Input (Keyboard/Screen)

### STD INPUT MESSAGE HEADER

The line protocol handler strips the message header from the input data, except for the hardware function codes, and does not include the header in the application's buffer.

### STD HARDWARE FUNCTION CODES

STD hardware function codes are listed in the appropriate hardware device manuals. These codes provide a special message labeling capability to be used by the application. This capability does not apply to the VIP7800 series.

The application can include two function codes in the message header of each text message to or from a terminal by setting at connect time the following in the IORB: (1) set to 1, bit B of the device-specific word I_DVS (see Table 9-3); and (2) set to 1, bit B (extension bit) of I_CT2 to specify that the IORB is extended (see Figure 4-2 and Table 4-11). The line protocol handler then inserts the two user-specified hardware function codes at read time into the IORB's I_FCS word.

The VIP7800 series has only one hardware function code that may be used by the application program. This function code appears as a two-character escape sequence in the data buffer. See the hard- ware manual.

### STD INPUT DATA

The line protocol handler places into the application's buffer all data, between the STX and ETX/ETB control characters, received from the VIP terminal or the PVE link. In 7-bit data mode, data is inserted into the buffer in 7-bit ASCII, with the most significant bit always zero. In 8-bit data mode, data is inserted into the buffer in 8-bit ASCII. In 7- bit mode, the LPH strips the ETX/ETB and LRC (longitudinal redundancy check character, see "Line Protocol Handler Functions," earlier) from the data and does not include them in the buffer. In 8- bit mode, the LPH strips the CRC16 (cycle redundancy check character) from the data and does not include it in the buffer.

## Control and Characteristics of STD Output

This subsection pertains to VIP output and is applicable to the keyboard, display screen, or receive-only printer (ROP) as indicated.

### STD OUTPUT MESSAGE HEADER

The STD line protocol handler supplies the output message header, but not the hardware function codes. Those for all but the VIP7800 series may be supplied by the application as described above under "STD Hardware Function Codes."

At write time, when the hardware codes are specified, they are placed in the I_FCS word of the IORB. To write function codes to the VIP7700 hardware, the application program must, at connect time, set bit B (extension bit) of the IORB's I_CT2 word to 1, to specify that the IORB is extended. When they are not specified (i.e., bit 8 of I_DVS set to 0 at connect time), the line protocol handler will insert two spaces, instead of function codes 1 and 2, into the I_FCS word (see Figure 4-2 and Table 4-11).

### CONTROL BYTE (SEND)

The control byte provides editing control (CR, LF, FF) for both ROPs and CRTs, as described later in this section.

### STD OUTPUT DATA

In 7-bit data mode, the application's output data must be 7-bit ASCII (the eighth bit is ignored). In 8-bit data mode, the application's output data must be 8-bit ASCII. Any ASCII control characters, if included in the application's data, are not transmitted.

### STD KEYBOARD/SCREEN OUTPUT EDITING CONTROL

The line protocol handler sends LF and CR editing characters for VIP keyboard/screen devices according to the values of the B- and C-bits of the device-specific word I_DVS (Table 9-3). The application specifies these bit values at write time to send the CR and LF characters, as follows:

| I_DVS Bits | | Editing Characters |
|---|---|---|
| B | C | Sent |
| 0 | 0 | CR |
| 0 | 1 | None |
| 1 | 0 | LF, CR |
| 1 | 1 | LF |

## STD RECEIVE-ONLY PRINTER EDITING SEQUENCE

The line protocol handler sends an output editing character sequence for the receive-only printer (ROP) according to the control byte supplied, the values of the B- and C-bits of the device-specific word I_DVS (Table 9-3), and the VIP type to which it is attached. The application specifies these bit values at write time to send the ROP output editing sequence, according to the ROP type and the VIP type to which it is attached, as shown in Table 9-5.

## STD RECEIVE-ONLY PRINTER CONTROL SEQUENCE

The STD line protocol handler sends an output control sequence according to the ROP type and the VIP type to which it is attached as shown in Table 9-6.

Table 9-5. STD Receive-Only Printer Editing Sequence

| CRT Type | Attached ROP Type | I_DVS Bits B | C | Output Editing Sequence |
|---|---|---|---|---|
| All | All | 0 | 0 | CR |
| All | All | 0 | 1 | None |
| VIP7700, 7700R, 7760; VTS7710, 7740 | TN1200, PRU1005 | 1 | 0 | LF, CR, 36 DELs |
| VIP7700, 7700R, 7760; VTS7710, 7740 | TN300, PRU1003 | 1 | 0 | LF, CR, 9 DELs |
| VIP7800 series | TN300, TN1200, PRU1003, PRU1005 | 1 | 0 | LF, CR |
| VIP7700, 7700R, 7760; VTS7740 7710 | TN1200, PRU1005 | 1 | 1 | LF, 36 DELs |
| VIP7700, 7700R, 7760; VTS7740, 7710 | TN300, PRU1003 | 1 | 1 | LF, 9 DELs |
| VIP7800 series | TN300, TN1200, PRU1003, PRU1005 | 1 | 1 | LF |
| TWU1901 | | 1 | 0 | LF, CR |

Table 9-6. STD Receive-Only Printer Control Sequence

| CRT Type | Attached ROP Type | Form Feed | ZZZZ | Output Editing Sequence |
|---|---|---|---|---|
| VIP7800 series | All | X | | FF |
| TWU1901 | | X | | FF |
| VIP7700, 7700R, 7760; VTS7740, 7710 | TN300, PRU1003 | X | | FF, 65 DELs |
| VIP7700, 7700R, 7760; VTS7740, 7710 | TN1200, PRU1005 | X | | FF, 250 DELs |
| VIP7800 series | None | X | | CLR |
| VIP7700, 7700R, 7760; VTS7740, 7710 | None | X | | FF, DEL |
| VIP7800 series | All | | X | LF, CR |
| VIP7700, 7700R, 7760; VTS7740, 7710 | TN300, PRU1003 | | X | LF, CR, 9 DELs |
| VIP7700, 7700R, 7760; VTS7740, 7710 | TN1200, PRU1005 | | X | LF, CR, 36 DELs |
| All | None | | X | LF, CR |

NOTES

1. Form feed (FF) is specified by bit 3 of the control byte.

2. ZZZZ represents the number of times an output editing sequence is performed and is specified by bits 4 through 7 of the control byte.

3. CLR clears all data attributes, moves the cursor to home position, and puts the terminal in text mode.

## PRINTER ESCAPE SEQUENCE FOR VIP7800 SERIES

For the VIP7800 series VIP, the STD LPH transmits the following printer escape sequence before the first data message:

```
        1B 5B 33 70 (PHLF)
```

and transmits the following printer escape sequence after the last data message:

```
        1B 5B 3C 70 (PEOM)
```

### Receive-Only Printer Support

Receive-only printer support by the STD LPH falls into three categories:

- VIP7800 series attached ROP support
- VIP7700, 7700R, and 7760 attached ROP support
- PRU 1901 support.

For the VIP7800 series attached ROPs, the STD inserts the start of message printer escape sequence (print host with local fill (PHLF)) before the first text message and appends the start print escape sequence (printer end of message (PEOM)) to the last text message. The application may supply the CR, LF, or FF characters minus the time fill characters in the text buffer, or may instruct the STD LPH to supply the CR, LF, or FF characters via the control byte or IORB device-specific word. Upon receipt of the CR, LF, or FF character, the VIP7800 series printer adapter supplies the required time fill characters. For HT or VT, the application must supply the HT or VT character and required time fill characters in the text buffer. In this mode an extended print buffer of 132 print positions is available, as well as the option to have all text transparent. Use of any of the options provided by the VIP7800 series printer adapter (e.g., copies option) requires the application to supply the appropriate escape sequence in the text buffer.

For the VIP7700, 7700R, and 7760 attached ROPs, the STD LPH supports the transparent (150 PRT) and nontransparent (150 NUL) print modes based on the setting of I_DVS bit A of the ROP connect IORB.

- Transparent mode: Allows the user to supply the CR, LF, or FF characters and timing fill characters in the text buffer, or instruct the STD to insert them. An extended print buffer of 132 print positions is also available in this mode.

- Nontransparent mode: The user need not include the CR or LF characters in the text buffer. The message received by the terminal is interpreted in the display format (80 print positions), and the necessary CR and LF characters are supplied by the terminal.

## VIP7800 Series Support

While certain operations of the VIP7800 series terminals are configurable by the application (e.g., roll, space suppress) via the connect IORB, the STD LPH imposes the following operational modes in order to ensure proper terminal operation:

- Block transmit auto: Successive blocks will be sent by the terminal, each time the terminal is polled, until the last block has been transmitted.
- Verify before process: The terminal normally operates in verify before process mode. In this mode, the terminal does not process the data unless the BCC indicates that no errors have occured. The transmitted data is restricted to 1024 characters, including all text plus the control characters CR, LF, FF, and DEL, supplied by both the application and the STD LPH.
- Process before verify: The user wishing to use blocks larger than 1024 characters must, at configuration time, specify PB after the 7800 series device type. In this mode, the terminal displays characters as it receives them, without protecting the integrity of the screen.

## VIP7826 Support

The VIP7826 terminal is a dual mode terminal that provides a bridge from the VIP7700 family to the VIP7800 series terminals. The two modes are switch as well as command selectable. STD will send a special escape sequence to set the device to operate in the mode specified at configuration time.

## Inactivity Time Support

If inactivity time is specified at configuration time (timeout value) for an STD channel, the timeout value is set for all stations connected to the channel. Inactivity time is the time when no read, writes, connects, or disconnects are being processed for a particular station. After reaching the timeout value specified at configuration time, any previous action is purged, an error message is displayed, and the station is disconnected.

## TWU1901 Support

The TWU1901 is a synchronous, polled, hard-copy device with a keyboard. It should be configured as a CRT; the LPH will handle the addressing (150 PRT; 150 NUL).

## Master LRN Processing

Master LRN processing enables one receive buffer to service up to a maximum of 32 terminals on a multi-dropped line. This technique drastically reduces the number of receive buffers required to support a multi-dropped environment. It is applicable only to read requests, and is supported through the user-supplied control word (described below).

This feature is used most effectively when the application issues two or more asynchronous read requests, each specifying different buffers. The issuance of multiple read requests allows the application to process received data while the STD LPH polls another terminal for data. However, use of the master LRN feature does not guarantee that all terminals associated with the master LRN are accessed sequentially, since STD does not poll for data unless a read request has been queued. When data has been received in this mode, STD returns the LRN, for which data was received, in the right half-byte (RHB) of the user-supplied control word. The application can then determine which terminal requires a response.

## Sub-LRN Support

The ROP and diskette can be accessed only by sub-LRN. Access to ROPs is handled by the File System, assuming that the appropriate ROP and STDLN CLM directives are entered at configuration time. To access the ROP at the physical I/O level, the application must set the sub-LRN in field I_ST to 1. If an error is returned and the same IORB re-issued, the sub-LRN must be reset, because STD might have returned a status in I_ST when posting the IORB, overwriting the original sub-LRN.

## Block Mode Processing

Block mode processing is the transmission or reception of small data blocks, which are components of a large message. It conserves buffer space, conserves total message transmission time in the presence of errors, and reduces line errors. This mode, applicable to the VIP7760, CTS7600, and VIP7804, is supported through the user-supplied control word.

In block mode transmit processing (ETB), a large message is transmitted in small blocks of data. The application is responsible for issuing an individual write request for each of these blocks. In block mode receive processing (ETB), the terminal, when polled, sends blocks of data until the last block is transmitted. The application, in this instance, is responsible for issuing the read request needed to initiate the transmission.

When I_DVS for the connect request specifies "terminal-generated block mode", the application must set the IORB range (RB_RA) to the size of the block expected. For the VIP7804, the RB_RA values are 20-2704 (i.e., 32-9999 decimal). For the VIP7700, 7700R, and 7760, the RB_RA value is FF (256 decimal).

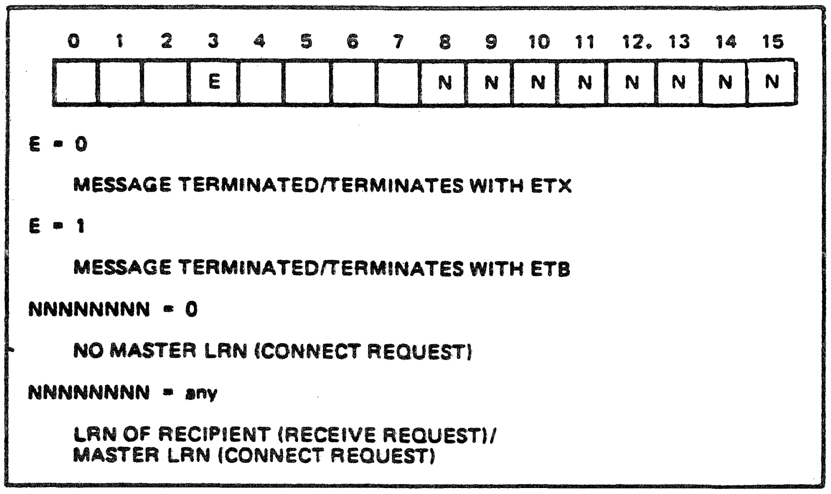
## Control Word

Master LRN and block mode processing require an additional word at the beginning of the data buffer. The connect request specifies control word utilization, and the IORB buffer address (RB_ADR) contains the address of the control word. If master LRN processing is desired, the right half-byte (RHB) of the control word must be set to the LRN which was designated as the master. If master LRN processing is not desired, the application must set the RHB of the control word to zero (0) (see Figure 9-1).

Block mode processing requires that the application include the control word in the IORB range (RB_RA) of the read/write request. The IORB buffer address (RB_ADR) must contain the address of the control word. On write requests, if the data to be sent is a block (ETB), the application must set bit 3 of the control word. If the data to be sent is an entire message or the last block of a message, the application must set bit 3 of the control word to zero. On read requests, if the received message was terminated with ETB, the STD LPH sets bit 3 of the control word to 1. If the received message terminated with ETX, the STD sets bit 3 of the control word to zero. The use of the control word does not preclude the use of the control byte for printer editing. If the control word is used with the control byte, the control word precedes the control byte in the buffer.

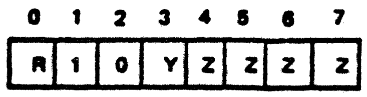
## Control Byte

The control byte provides editing control (CR, LF, FF) for both ROPs and CRTs. This control is effected by setting bit 4 of the write IORB, which causes STD to treat the first character (third if control word is specified) of the data as the control byte. The control type is examined by the STD and, according to the bit settings, the STD transmits the appropriate characters before the data. The control byte format is shown in Figure 9-2.



0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
E N N N N N N N N
E = 0
MESSAGE TERMINATED/TERMINATES WITH ETX
E = 1
MESSAGE TERMINATED/TERMINATES WITH ETB
NNNNNNNN = 0
NO MASTER LRN (CONNECT REQUEST)
NNNNNNNN = any
LRN OF RECIPIENT (RECEIVE REQUEST)/
MASTER LRN (CONNECT REQUEST)

Figure 9-1. Control Word



R
RESERVED (NOT EXAMINED)
Y=0
DO NOT ISSUE FORM FEED SEQUENCE
Y=1
ISSUE FORM FEED SEQUENCE
ZZZZ
NUMBER OF LINES TO SKIP BEFORE PRINTING (BINARY)
(E.G., IF ZZZZ=0100, STD LPH WILL PERFORM
4 FF SEQUENCES)

Figure 9-2. Control Byte

## Output Data and Invalid Characters

The data must be 7-bit ASCII (the eighth bit is ignored). The ASCII control characters SOH (01), STX (02), and ETX (03), are not transmitted if included in the data. Instead, an SYN (16) is transmitted by the STD LPH in place of each occurrence.

## VIP7800 Series Message Range Requirements (Verify Before Process Mode)

The maximum number of characters that the VIP7800 series terminal (CRT/ROP) may have written to it is 1024 (verify before process). If the application specifies editing control for the CRT/ROP, the STD LPH inserts/appends the appropriate control characters (CR, LF, FF, PHLF, PEOM) to the data. While these characters do not appear in the data buffer and are not included in the IORB range, they do occupy terminal buffer space. Therefore, the application must account for these characters when issuing the data write request. Specifically, the data plus STD-generated transmitted characters must be less than or equal to 1024.

## VIP7800 Series Terminal Transmission Modes and Cursor Positioning

The VIP7800 series terminal supports two methods for transmitting data to the host and for positioning the cursor.

1. Return=Normal Mode: Data is transmitted from the terminal to the host by pressing the TRANSMIT key. The cursor is positioned to the next cursor position following the data to be transmitted. The RETURN key may be used to move the cursor to column 1 of the next line.

2. Return=Transmit Mode: Data is transmitted from the terminal to the host by pressing the RETURN key. Pressing the AUTO LINE FEED (AUTO LF) key changes the cursor's position after the data is transmitted, as follows:

   AUTO LINE FEED depressed.
     Cursor is positioned to column 1 of the next line.

   AUTO LINE FEED not depressed.
     Cursor is positioned to column 1 of the current line.

   The received range residue is modified to not reflect the reception of the CR/LF or CR.

## VIP7800 Series Break Processing

Break processing on the VIP7800 series is performed by a shifted or unshifted function key rather than by the BREAK key used on other terminals. Shifted or unshifted function keys F1 to F9, F11, and F12 are the only keys that can be defined as break keys. The default key is shifted F12. The break function is configurable by means of the STTY command (STTY -BREAK).

When the terminal is operating in no-roll mode, pressing the shifted break function key causes the ** BREAK ** message to be displayed on the 25th line of the terminal. To respond to this message, the operator should:

1. Acknowledge the break message by pressing function key 10

2. Respond to the break condition by:

   a. Entering the UW, SR, or PI command, as appropriate
   b. Pressing the transmit key after entering the command

## Supervisory Messages

Supervisory message handling is applicable only to the VIP7800 series. To read or write supervisory messages, an application must first connect the terminal with bit 9 of I_DVS set to 1 (no roll).

### SUPERVISORY MESSAGE READS

To read a supervisory message, the application must set bit 9 of I_DVS to one in the read IORB. Servicing of the supervisory read order places the cursor on the 25th line. To return the cursor from the supervisory message line to the data region of the screen, the operator must:

1. Press the return or transmit key (depending on the terminal's operating mode) in order to terminate the read.

2. Press function code 10.

SUPERVISORY MESSAGE WRITES

To write a supervisory messge, the application must set bit 9 of I_DVS to one in the write IORB. Servicing of the supervisory write order places the cursor and message on the 25th line.

If bit 8 of I_DVS in the write IORB is set to one, STD repositions the cursor to the location it occuppied before the supervisory write. If this bit is set to zero, the cursor remains on the 25th line until the operator presses function code 10.

## Diskette Handling for the CTS7760 and VTS7740

The following conventions apply:

- The diskette cannot be accessed through the file system. The application must use physical I/O, setting I_ST to 2. The application must reset I_ST when re-using an IORB to issue an I/O order.
- Device specific words in connect, read, and write IORBs must indicate no control byte.
- The first two bytes of the application buffer must be one of the following escape sequences:

| Escape Sequence | Meaning |
|---|---|
| 1B 57 | Write |
| 1B 42 | Read, display on terminal |
| 1B 56 | Read, send to host |
| 1B 51 | Erase |

- To read or write buffers over 256 characters, an application must use ETB processing. Alternatively, CTS7760 and VTS7740 hardware allows the block size to be set at 128 characters.

## Two- and Four-Wire Line Function

Two types of wire connections are supported:

1. Two-wire: Two physical wires (one pair) make up the electrical circuit onto which a data set may be connected. There is a 250 millisecond data set turnaround time.
2. Four-wire: Four physical wires (two pairs) make up the electrical circuit onto which a data set may be connected. Four-wire does not infer full-duplex operation.

## Long Q Frame Line Function

ALL VIP terminal types supported by the STD LPH must be set to long Q frame (i.e., the Q-frame response by the terminal is SYN SYN SYN SYN SOH EOT).

## ERROR PROCESSING BY STD LINE PROTOCOL HANDLER

Table 9-7 lists the errors reported by the STD line protocol handler for any VIP configuration. It also lists corresponding return status error codes (see Table 4-10), corresponding bits in the STD software status word I_ST (see Table 9-4), and possible recovery actions.

Table 9-7. Errors Reported by STD Line Protocol Handler

| Error Condition | Posted Error Return Status | I_ST Bit | Possible Recovery | Comments |
|---|---|---|---|---|
| Error during open | B | As reported | | |
| "Not available" message received | 7 | E | None | |
| Page overflow not corrected | 7 | D | None, or retry once | |
| Invalid range in IORB | 4 | E | None | |
| Read timeout | 6 | 7 | Immediate return | |
| NAK limit reached | 7 | 8 | Retry four times | |
| Busy received | 7 | F | | |
| Purged due to immediate close or read/write abort | B | None | | |
| Station disabled | B | None | | |
| Data service rate error | 0 (transmit)<br>7 (receive) | 2<br>2, 8 | Not applicable<br>Retry four times | Not fatal |
| Long record | 0 | 6 | None (ACK sent to VIP) | Data lost |
| Illegal character | 0 (transmit) | 7 | Replace illegal character with SYN characters | Bad character in application's buffer |
| Sequence error | B (receive) | 8 | | |
| Phone hang up | B | B | None | |
| Excessive checksum or parity error | B | 9 | Retry four times | |
| Poll failure | B | 8 | Retry four times | |

# *Section 10*
# *POLLED VIP EMULATOR LINE PROTOCOL HANDLER*

The Polled VIP Emulator (PVE) Line Protocol Handler (LPH) allows a DPS 6 system to be connected to a communications link that operates according to the polled VIP protocol. The line can be half or full duplex, dedicated, or switched, and operates at up to 9600 baud for the MLC controller and up to 19200 baud for the MLC-16 controller. The PVE LPH also provides functionality to recognize and respond to the VIP7760 controller poll.

The computer that controls the communications link is known as the control station (CS), which can be any Honeywell host system that supports the VIP protocol.

## GENERAL PVE LINE PROTOCOL HANDLER OPERATION

A PVE LPH, which is configured in a tributary processor, supports up to 32 tributary stations per line. Each tributary station appears to the control station as a VIP terminal. To the control station, each PVE tributary station is known by a poll address, and to the tributary processor, by a logical resource number (LRN). There is a one-to-one relationship between the poll address and the LRN.

An application running in a tributary processor issues read and write requests against an LRN associated with a tributary station. Similarly, the control station communicates with a tributary station by issuing poll and selection orders with the appropriate poll or selection address. Figure 10-1 illustrates a typical PVE configuration.

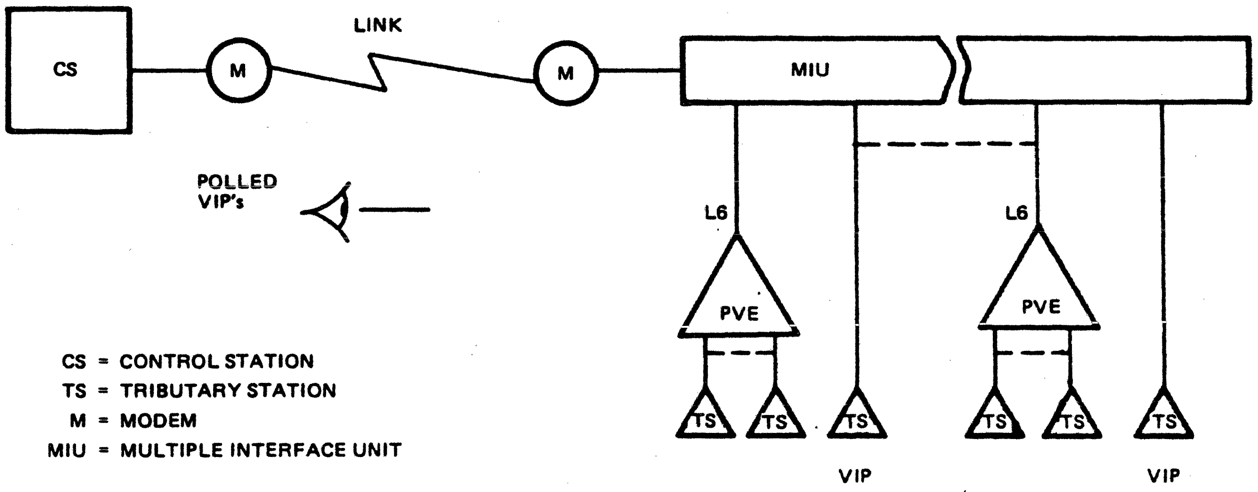


Figure 10-1. Typical PVE Configuration

When the PVE receives a select request with the LRN-associated poll address, it forwards the message to the tributary station to satisfy the application's read request. When the PVE receives a poll request for the LRN-associated poll address, it forwards the message to the control station to satisfy the application's write request. Thus, the application provides the equivalent of the screen and keyboard, with read and write requests, respectively. The PVE LPH supports only the screen and keyboard features of the VIP.

The PVE LPH also supports controller poll processing. This processing option, specified at system build, permits the PVE line protocol to support controller poll orders. Such orders are issued by the control station in support of a VIP7760 (CTS7600) controller configuration. A typical controller configuration is shown in Figure 10-2. As many as eight controllers can be associated with a single communications link; up to 32 uniquely-identifiable stations can be associated with the controllers, grouped in any number under each controller. Each station so grouped, however, must have a unique poll address.

The advantage of controller poll processing is that the control station can issue a single controller poll message to a set of stations that are attached to the controller, instead of issuing individual sequential poll messages to the same set of stations. When the PVE receives a controller poll message, it individually checks all PVE stations that are associated with that controller. If any station has a write request pending, PVE forwards the message to the control station in response to the controller poll request.

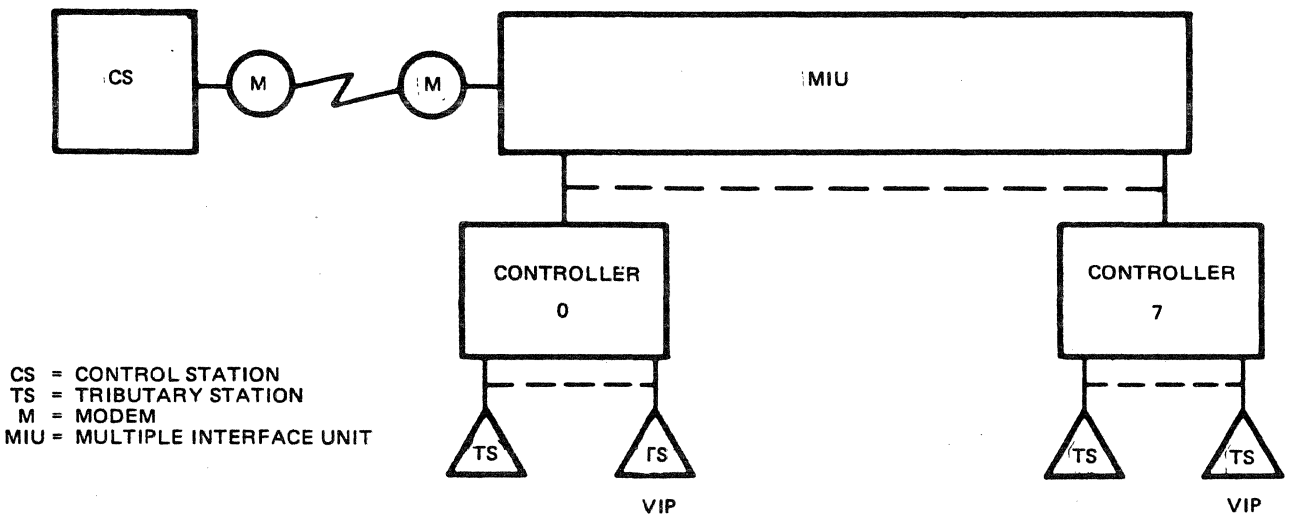


Figure 10-2. Typical Controller Poll Configuration

The PVE LPH also supports user-specified delayed status in response to error-free received messages. When used, the application informs PVE that all received error-free messages are not to be immediately ACK'ed, but PVE is to delay the response until the application specifies the correct response (ACK, BUSY, NA, or PGOF). This procedure is achieved by the connect IORB. Upon receiving an error-free message, PVE will POST the current receive IORB. The application analyzes the received message and isssues a CONTROL WRITE to PVE. The CONTROL WRITE is the means for specifying the correct status response to the message. Figure 10-3 illustrates the typical delay response procedure.

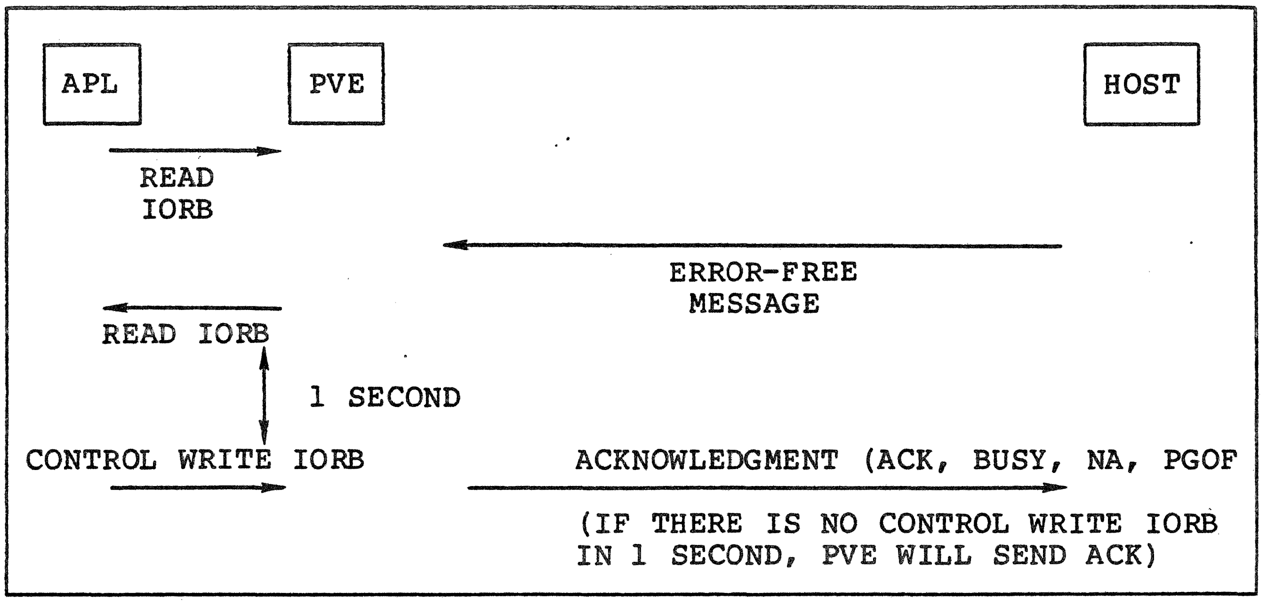

APL
PVE
HOST
READ IORB
ERROR-FREE MESSAGE
READ IORB
1 SECOND
CONTROL WRITE IORB
ACKNOWLEDGMENT (ACK, BUSY, NA, PGOF
(IF THERE IS NO CONTROL WRITE IORB IN 1 SECOND, PVE WILL SEND ACK)

Figure 10-3. Typical Delay Response Procedure

## PVE-Specific IORB Values

The PVE-specific IORB item I_CT2, device-specific word I_DVS, and software status word I_ST are shown in Tables 10-1, 10-2, and 10-3, respectively. Bits not explicitly described in the tables must be 0. Section 4 describes the general form of the IORB.

Table 10-1. Function Codes in I_CT2 in the IORB

| Function Code | Definition | Use |
|---|---|---|
| 0 | Wait online | Used by the line protocol handler to complete the description of the requested I/O function |
| 1 | Write | |
| 2 | Read | |
| A | Connect | |
| B | Disconnect | |

Table 10-2. PVE Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| | For connect call only (function code A) |
| 0 | 1 = Delayed Response |
| 2 | 1 = Auto Call |
| 3 | 1 = User Supplying ACK |
| 4 | 1 = Non Polled Line |
| 8 | 0 = PVE Specified FCS<br>1 = User Specified FCS |
| 9 | 1 = User Supplied Full VIP Header |
| 10 | 0 = Accept Received DEL Character<br>1 = Strip Received DEL Character |
| 11 | 0 = SOH EOT Q-Frame<br>1 = EOT Only Q-Frame |

Table 10-2 (cont). PVE Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| 12 and 13 | LPH response to application when LPH receives data but no read IORB is available:<br>00 = Send NAK<br>01 = Send ACK<br>10 = Send BSY status<br>11 = Send NAK (same as 00) |
| 15 | 1 = Logical Connect |
| | For disconnect call only (function code B) |
| 14 | 0 = Abort (dequeue) all IORBs on request queue<br>1 = Process all outstanding requests on request queue |
| 15 | 0 = Disconnent phone after disconnect<br>1 = Maintain phone connection after disconnect |
| | For write call only (function code 1) |
| 4 | 1 = Set Device Busy |
| 5 | 1 = Set Device Unbusy |
| 6 | 1 = Abort Write |
| 7 | 1 = Send ETB |
| 9 | 1 = Control Write |
| 12 | 1 = Disk Control Write |
| 13 | 1 = ROP Control Write |

## VIP Protocol Message Structure for PVE

Figure 10-4 shows two VIP protocol message structures for PVE.

## Control and Characteristics of PVE Input

### PVE INPUT MESSAGE HEADER

The PVE LPH strips the message header, between the SOH and STX control characters, and does not include it in the application's buffer.

Table 10-3. PVE Software Status Word I_ST in the IORB

| Bit | Meaning When Bit Set to 1 |
|---|---|
| 0 | N/A |
| 1 | N/A |
| 2 | Data service rate error |
| 3 | N/A |
| 4 | Communications control block (CCB) service error |
| 5 | N/A |
| 6 | Long record |
| 7 | 0 = ETX character received<br>1 = ETB character received |
| 8 | NAK limit reached |
| 9 | Excessive checksum/parity errors |
| A | Nonzero residual range |
| B | Phone hang-up |
| C | N/A |
| D | N/A |
| E | N/A |
| F | Fatal error: bus parity or memory error |

## PVE HARDWARE FUNCTION CODES

PVE hardware function codes are listed in the appropriate hardware device manuals. These codes provide a special message-labeling capability to be used by the application.

The application can include two function codes in the message header of each text message by setting at connect time the following in the IORB: (1) set to 1, bit 8 of the device-specific word I_DVS (see Table 10-2); and (2) set to 1, bit B (extension bit) of I_CT2 to specifiy that the IORB is extended (see Figure 4-2 and Table 4-11). The LPH then inserts the two user-specified hardware function codes at read time into the IORB's I_FCS item.

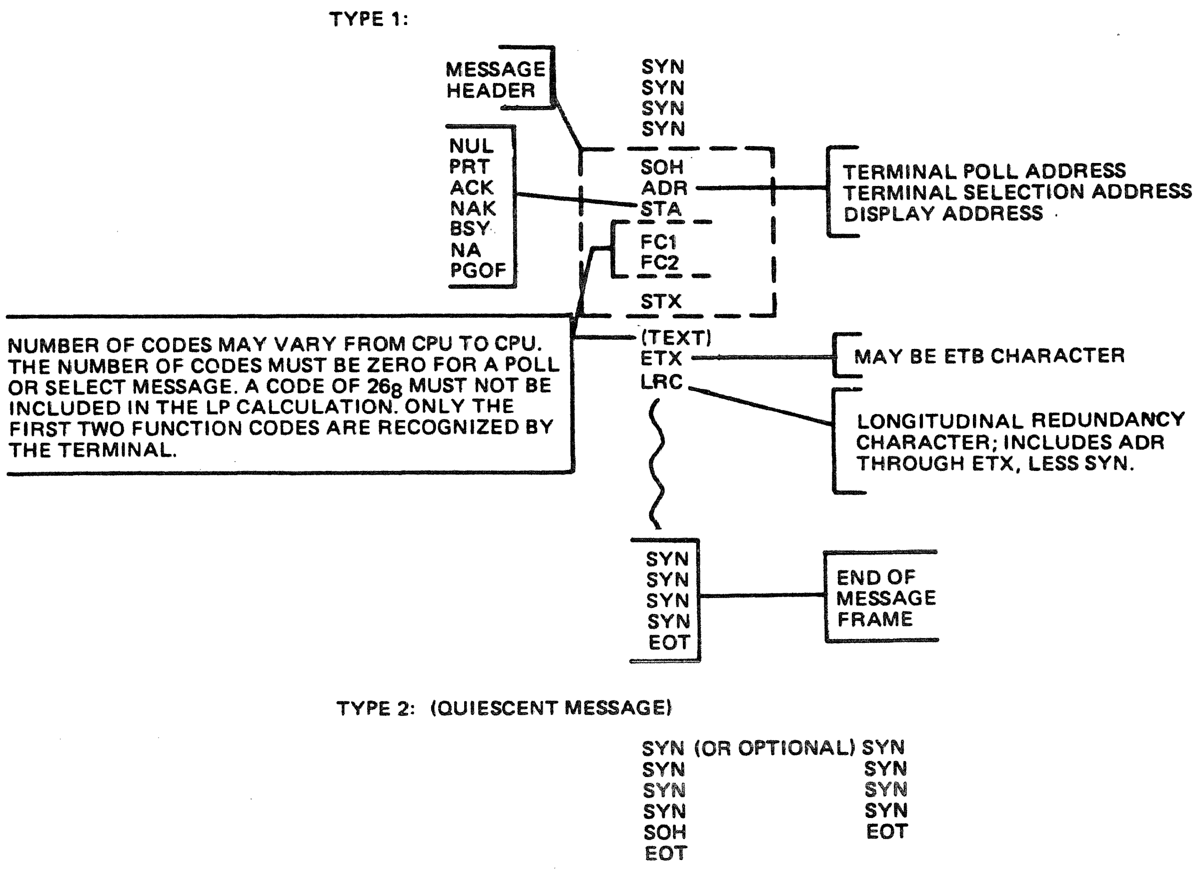

TYPE 1:
MESSAGE HEADER
SYN
SYN
SYN
SYN
NUL
PRT
ACK
NAK
BSY
NA
PGOF
SOH
ADR
STA
FC1
FC2
STX
TERMINAL POLL ADDRESS
TERMINAL SELECTION ADDRESS
DISPLAY ADDRESS
NUMBER OF CODES MAY VARY FROM CPU TO CPU. THE NUMBER OF CODES MUST BE ZERO FOR A POLL OR SELECT MESSAGE. A CODE OF 26₈ MUST NOT BE INCLUDED IN THE LP CALCULATION. ONLY THE FIRST TWO FUNCTION CODES ARE RECOGNIZED BY THE TERMINAL.
(TEXT)
ETX
LRC
MAY BE ETB CHARACTER
LONGITUDINAL REDUNDANCY CHARACTER; INCLUDES ADR THROUGH ETX, LESS SYN.
SYN
SYN
SYN
SYN
EOT
END OF MESSAGE FRAME
TYPE 2: (QUIESCENT MESSAGE)
SYN (OR OPTIONAL) SYN
SYN SYN
SYN SYN
SYN SYN
SOH EOT
EOT

Figure 10-4. VIP Protocol Message Structure for PVE

## PVE INPUT DATA

The LPH places all data between the STX and ETX control characters into the application's buffer. The data is inserted into the buffer in 7- or 8-bit ASCII. The LPH strips the ETX and LRC (longitudinal redundancy check character, see Section 7, "Communications Subsystem Error and Correction Procedures") from the data and does not include them in the buffer.

It also strips DEL characters when the application, at connect time, sets to 1 the A-bit of the device-specific word I_DVS (Table 10-2).

By setting the C- and D-bits of I_DVS as shown in Table 10-2, the application can control the response that the LPH sends when it receives data, but no read IORB is available.

## Control and Characteristics of PVE Output

### PVE OUTPUT MESSAGE HEADER

The PVE LPH normally supplies the output header between the SOH and STX control characters. The application can specify hardware function codes (1, 2) as described above under "PVE Hardware Function Codes." To write function codes, the application must, at connect time, set bit B (extension bit) of the IORB's I_CT2 item to 1, to specify that the IORB is extended. At write time, when specified, the codes are extracted from the I_FCS item of the IORB. When the codes are not specified (bit 8 of I_DVS set to 0 at connect time), the LPH will supply two spaces, instead of the codes, into I_FCS. (See Figure 4-2 and Table 4-11.)

### PVE TERMINAL ADDRESS (ADR) AND MESSAGE STATUS (STA)

The PVE LPH supplies an ADR (terminal address) of X'60' (keyboard/screen) and an STA (message status) of NUL to the application.

### PVE OUTPUT DATA

The application's output data may be 7-or 8-bit ASCII. In case of 7-bit ASCII, the most significant bit is used by the LPH during transmission of odd parity. Output data must not include the ASCII control characters SOH, STX, ETB, ETX, EOT, or SYN.

The LPH supplies output ETX control characters and longitudinal redundancy check characters (LRCs) (described in Section 7, "Communications Subsystem Error and Correction Procedures").

### PVE LINE PROTOCOL HANDLER TIMEOUT INTERVALS

Table 10-4 lists the timeout intervals used by the LPH for the connect, read, and write functions. The LPH will attempt or reattempt the functions until the indicated timeout period has elapsed. In addition to the interval in the table, there is also a gross timeout of one minute, which expires when the control station ceases to poll or select any tributary station.

Table 10-4. PVE Timeout Intervals

| Function | Timeout Interval |
|---|---|
| Connect | 200 seconds |
| Read | Indefinite |
| Write | Indefinite |

## ERROR REPORTING BY PVE LINE PROTOCOL HANDLER

Table 10-5 lists the errors reported by the PVE LPH. It also lists corresponding return status error codes (see Table 4-10) and corresponding bits in the software status word I_ST (see Table 10-3).

Table 10-5. Errors Reported by PVE Line Protocol Handler

| Error Condition | Posted Error Return Status | I_ST Bit | Comments |
|---|---|---|---|
| No interrupt from MLC | 6 | 7 | Poll failure or CCP/MLC failure |
| NAK limit reached | 7 | 8 | Write failure |
| Purged due to immediate close | B | None | |
| Station disabled | B | None | |
| Fatal error interrupt level | B | None | |
| Data service rate error | 0 (send)<br>7 (receive) | 2<br>2, 8 | Not fatal |
| Communication control block service rate error | 7 | 4, 8 | |
| Long record | 0 | 6 | Not fatal |
| Phone hang-up | B | B | |
| Nonexistent resource, or Bus parity error, or Unrecoverable memory error | B | None | |

# *Section 11*
# *BSC2780/3780 LINE PROTOCOL HANDLER*

The Binary Synchronous Communication (BSC) BSC2780/BSC3780 Line Protocol Handler (LPH) supports BSC2780 and BSC3780 point-to-point, nontransparent or transparent EBCDIC, or nontransparent ASCII transmission between a DPS 6 system and another host system (subject to certain restrictions).

The BSC3780 protocol is similar to the standard BSC2780 protocol and unless specifically stated otherwise, the rest of this section and the term BSC pertain to both.

## GENERAL BSC LINE PROTOCOL HANDLER OPERATION

When a station (device or computer) at either end of a communication line has a message to send, it requests use of the line by sending an ENQ bit message. (See Appendix G for definition of ENQ and other control characters.) The receiving station must respond with an ACK/0 sequence before the sending station can transmit a data message.

### BSC Transmit and Receive Operations

A station that has control of the line, i.e., the right to transmit, is known as the master (primary) station. The station that relinquishes control, i.e., will receive, is the slave (secondary) station. Primary and secondary are arguments of the BSC CLM directive used during system build.

When the first data message from the master station is successfully received, the slave station responds with an ACK/1 sequence. Acknowledgments for subsequent remaining messages alternate between ACK/0 and ACK/1. The master/slave status for each respective station remains in effect until the master station gives up control by sending an end-of-transmission (EOT) character (which is not acknowledged by the slave station).

When a bidding station does not receive an ACK/0 response within a specified interval (timeout period), it sends another ENQ message. At the same time, or at nearly the same time, the other station may be sending an ENQ message, bidding for the line. Thus both stations may be bidding with neither receiving an ACK response. This is known as line contention. Line contention can be avoided by designating one station as the primary and the other as secondary during system build. Then when the designated primary station receives an ENQ response to its bid message, it retransmits the ENQ message to the secondary station, which in turn ignores its own bid request and responds to the primary station with an ACK or NAK.

The BSC line protocol handler allows a receiving station to reply to a data message with an reverse interrupt (RVI) message if it has an urgent requirement to transmit data.

Figure 11-1 illustrates bids and other interactions between a master and slave station.

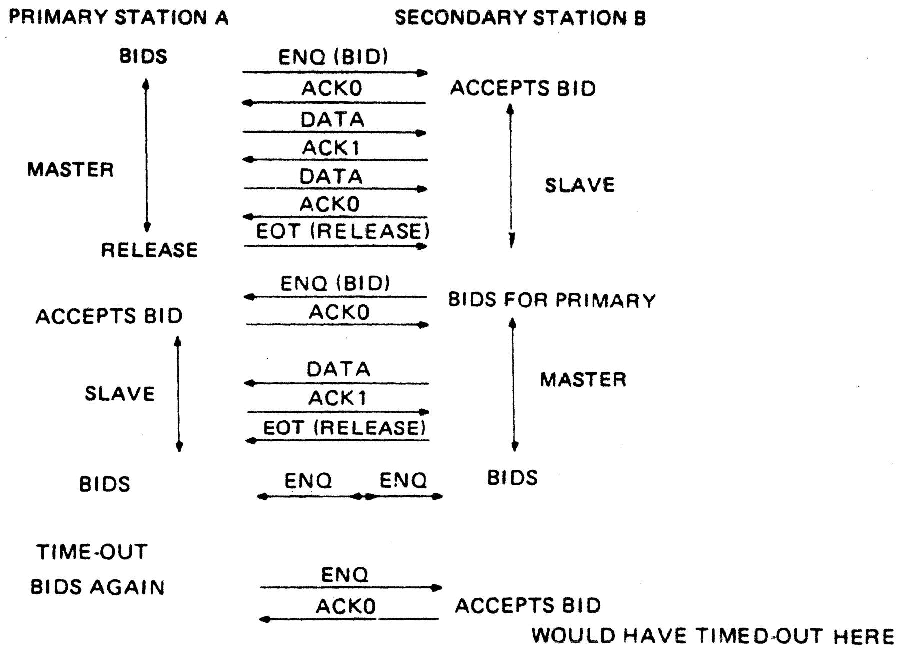


Figure 11-1. Example of BSC Communication

## BSC Data Transmission Modes

BSC operates in either basic data transmission mode or in advanced data transmission mode, according to whether a control byte is included in the data being transmitted. (See "BSC Control Byte (Receive)" and "BSC Control Byte (Send)" later in this section.)

In basic data transmission mode, there is no control byte included in the data being transmitted along the communications line.

In advanced data transmission mode, the application includes a control byte that occupies the first byte of the output buffer but is not transmitted across the line. The control byte indirectly controls the operation of the line protocol handler (e.g., sending an ETB or ETX), or conveys information about a data transfer (e.g., whether transparent text was received).

## BSC2780 and BSC3780 Differences

The BSC3780 protocol differs from the BSC2780 protocol in that the BSC3780 protocol allows an application to:

- Receive a conversational reply

- Receive two records and to transmit a single record, when the double-block option is selected at connect time (whereas the BSC2780 protocol allows both transmission and reception of two records)

- Receive multi-block records and to transmit a single record, when the multi-block option is selected at connect time (whereas the BSC2780 protocol allows both transmission and reception of multi-block records)

- Receive and transmit selected BSC control characters in nontransparent mode.

## BSC Record Types

The BSC LPH supports three forms of record transmission:

1. Single-record transmission
2. Two-buffer transmission
3. Multiple intermediate text block (ITB) sequence.

To identify the record constructs in a more meaningful and uniform manner, the following terms are used:

- Single-block (in place of single-record or single-buffer)
- Double-block (in place of two-buffer)
- Multi-block (in place of multiple ITB sequences).

## BSC2780/BSC3780 Features

The following discussions in this subsection include references to BSC-specific fields in the input/output request block IORB (see Table 4-8) and to control bytes. See Tables 11-4 and 11-5 later in this section for descriptions of the device-specific word I_DVS and software status word I_ST, respectively. Control bytes are described under "Control Byte (Receive)" and "Control Byte (Transmit)".

### BSC DOUBLE-BLOCK FEATURE

With the double-block feature, the use of the second buffer reduces line turnaround time, i.e., two records can be transmitted with only one acknowledgment. However, there are these disadvantages:

- When a line (parity) error occurs, both records must be retransmitted.
- One transmission requires that two writes be issued, which are not posted until an acknowledgment is received.
- Four buffers are necessary to operate the line efficiently.

Figure 11-2 shows record transmissions with and without the double-block feature.

Before selecting the double-block feature, compare the advantage of better line utilization against the disadvantages of a more complex program and increased buffer usage, and consider the following:

1. In BSC2780 with the double-block option, two records can be received or transmitted (using an ITB (intermediate text block) sequence).

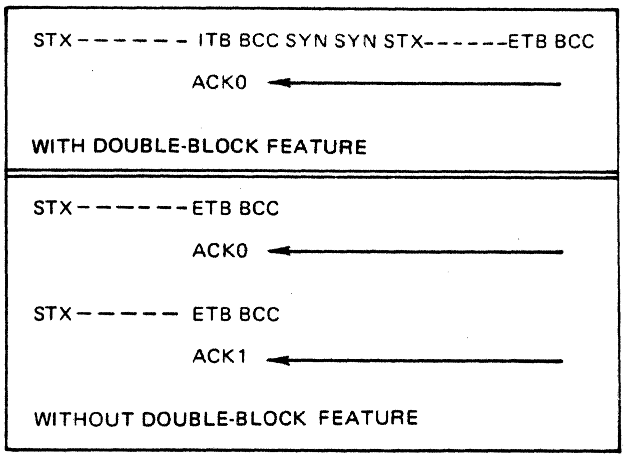

STX------ ITB BCC SYN SYN STX------ETB BCC
ACK0
WITH DOUBLE-BLOCK FEATURE
STX------ETB BCC
ACK0
STX----- ETB BCC
ACK1
WITHOUT DOUBLE-BLOCK FEATURE

Figure 11-2. BSC Double-block Feature in Record Transmission

2. In BSC3780, with the double-block option, two records can be received, using an ITB sequence, and single records can be transmitted. This implies that an application using BSC3780 must be able to receive up to two records at any one time, but can only initiate single-record transmission.

3. The double-block feature cannot be used with synchronous reads, because the intermediate files being received may be terminated by an ETX record. If the ETX record is the first of the two records being read, the second read (synchronous) would not be posted to the system.

   For example:

```
┌►READ (asynchronous)  )
│ .                    |
│ . process            |
│ .                    }    Assumes always two records
│ READ (synchronous)   |    per transmission.
│ .                    |
└─.  process           |
  .                    )
```

   The following sequence is better:

```
  READ (asynchronous)
┌►READ (asynchronous)
│ WAIT (1)
│ .
│ .  process
│ .
│ READ (asynchronous)
│ WAIT (2)
│ .
└─.  process
  .
```

## BSC MULTI-BLOCK FEATURE

The multi-block feature allows an application to send or receive from 1 to 7 records in a single transmission. Use of multi-block reduces line turnaround time in that only one write order, one user buffer, and one acknowledgment are required for the transmission of multiple records.

This feature is optionally selected at connect time. When using BSC2780, an application selects the multi-block feature to both send and receive multiple-record transmissions for the duration of the connect; single- and double-block transmissions are precluded. When using BSC3780, an application selects the multi-block feature only to receive multi-block transmissions.

For this feature to be selected at connect time, its use must have been provided for at system build. This is accomplished by an argument in the BSC CLM directive. Indicating the possible use of the multi-block feature during system build does not require that it be selected for use at connect time, since selection is optional. However, selection of this feature at connect time is prohibited if the possibility of its use was not provided for during system build.

When BSC3780 and multi-block are specified at connect time, the receive and transmit buffers must be organized as shown in Figure 11-3. The buffer shown in Figure 11-3 is divided into two sections, a header section and a data section. The data section contains the user's records, referred to as data blocks. Only the data blocks of a data buffer are transmitted, with the appropriate protocols inserted. The header section is interpreted by and controls the processing of the BSC LPH. Table 11-1 defines the contents of the buffer's header section. Figure 11-4 illustrates the transmission of the data blocks shown in Figure 11-3.

Table 11-1. Multi-block Header Section Field Descriptions

| Transmit (BSC2780 Only) | | Receive (BSC2780/BSC3780) |
|---|---|---|
| Control Field | Byte 0: Contains optional control byte. | Same as for transmit. |
| | Byte 1: Must be zero. | |
| Block Count | Number of blocks to be transmitted. | Maximum number of blocks which can be received. |
| | Posted back with actual number transmitted. | Posted back with actual number received. |
| Block Offset | Word offset from base of buffer to beginning of data block. | Same as for transmit. |
| | Posted back with contents unchanged. | |
| Block Size | Number of characters (bytes) in data block to be transmitted. | Maximum number of characters (bytes) which can be received. |
| | Posted back with residual range, if any. | Posted back with actual number received. |

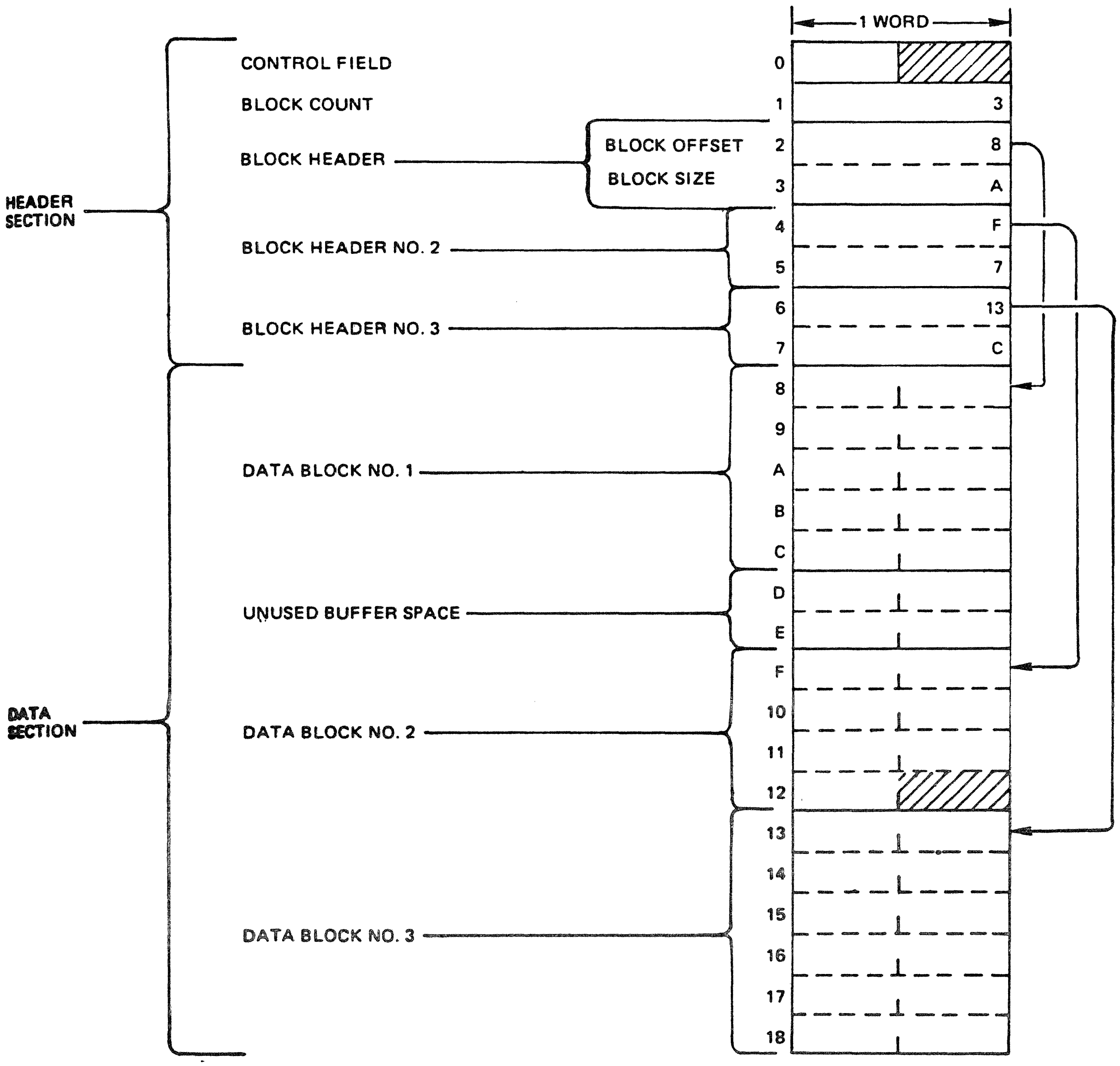


Figure 11-3. Multi-block Buffer Organization

```
STX - - - (DATA BLOCK NO. 1) - - - ITB BCC SYN SYN STX
- - - (DATA BLOCK NO. 2) - - - ITB BCC SYN SYN
STX - - - (DATA BLOCK NO. 3) - - - ETB BCC

                              ACK0 ←
```

Figure 11-4. BSC Multi-Block Transmission of Buffer Shown in Figure 11-3.

The following rules apply to the construction of a multi-block buffer:

- Data blocks cannot overlap.
- Each data block begins on a word boundary.
- The range value in the IORB need not be specified; this value is calculated from the buffer header section by system software. The range value returned in the posted read IORB is the header size plus total block lengths plus any gap(s) between blocks. Range value returned in the posted write IORB is equal to the header size.
- Buffer space may exist between data blocks but must not be used because it may be overwritten by system software.
- Block headers and the corresponding data blocks they define must be in the same sequence. The first header block must define the first·data block, the second header block must define the second data block, etc. These header blocks precede the data blocks.
- During a single connect, the number of data blocks can vary for each transmission, provided they do not exceed the maximum allowed. This maximum number, which can vary from 1 to 7, is specified by the user at system build.
- If, during a read operation, fewer data blocks are received than were specified in the block-count field of the buffer's header section, the block-count field is set equal to the number of data blocks received and the IORB is posted back with bit A of the software status word I_ST set to 1, indicating nonzero residual range.

## BSC TEMPORARY TEXT DELAY (TTD) FEATURE

The following describes the sequence of the temporary text delay (TTD) feature:

1. When a master station receives an ACK, and no output request block (IORBs) are queued, that station waits 2 seconds for one IORB (or two IORBs when there are two buffers) to be queued.
2. The master station then sends the temporary text delay (TTD) control character sequence (STX, ENQ) to the slave station.
3. When the slave station responds with a NAK, the master station checks whether the application has queued the appropriate write requests. If the write requests are not queued, the master station continues the TTD sequence until the application issues the necessary requests.

4. If the EOT or ETX bit (A-bit or D-bit) in the I_DVS word of the IORB is set (Table 11-4), one write request will effect transmission.

Figure 11-5 is an example of the temporary text delay sequence.

## BSC WAIT BEFORE ACKNOWLEDGE (WACK) FEATURE

A BSC slave station will send ACK/0 and ACK/1 responses to messages satisfactorily received, provided there is at least one outstanding read request (two with the double-block feature), in addition to the request being processed.

1. When no read is queued, the slave station posts the current read, waits 2 seconds for read requests to be queued, then sends a WACK response, indicating to the master station that the last message was received, but the slave station cannot accept more data.

2. The master station waits (timeout), then sends an ENQ message.

3. If a read request was queued during the timeout, the slave station responds with an ACK, and the master station can send its next data message.

4. If no read request was queued during the timeout, the slave station waits another 2 seconds, and when necessary sends another WACK sequence.

The ASCII and EBCDIC WACK sequences are DLE ; and DLE , respectively. Figure 11-6 is an example of the wait before acknowledge (WACK) sequence.

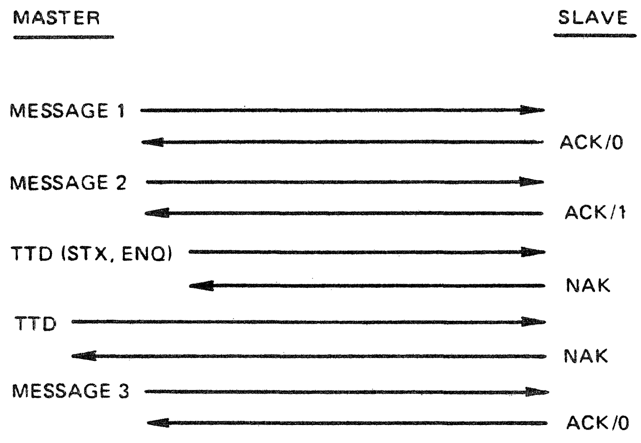


Figure 11-5. BSC Temporary Text Delay (TTD) Sequence Example

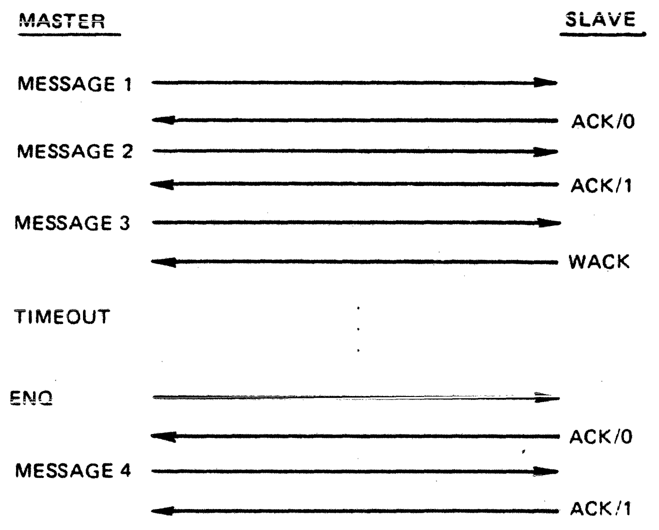


Figure 11-6. BSC Wait Before Acknowledge (WACK) Sequence Example

## BSC REVERSE INTERRUPT (RVI) FEATURE

When a slave station is processing read requests and must unexpectedly transmit an urgent message, that station must issue a reverse interrupt (RVI) message, which informs the master station that the slave station is requesting control of the line.

On receiving an RVI character, the master station should empty its buffers and give up control of the line. However, the master station does not have to acknowledge the RVI by giving up control.

The application can request the BSC line protocol handler to send an RVI character, by either of the following methods:

1. Use of the control byte. The application issuing read requests issues a transmit request with bit 5 of the control byte set to 1 (see Figure 11-12) and with the urgent message in the application's buffer.

2. Use of the device-specific word I_DVS of the IORB. The application issuing read requests issues a transmit request with the B-bit of I_DVS set to 1 and with the urgent message in the application's buffer.

The application issuing write requests can detect an RVI character by either of these methods:

1. Test bit 3 of the control byte after a successful write request is posted. A bit setting of 1 indicates that the RVI for that IORB was received.

2. Test bit 3 of the IORB's software status word I_ST. A bit setting of 1 indicates an RVI was received.

Figure 11-7 is an example of a reverse interrupt (RVI) sequence.

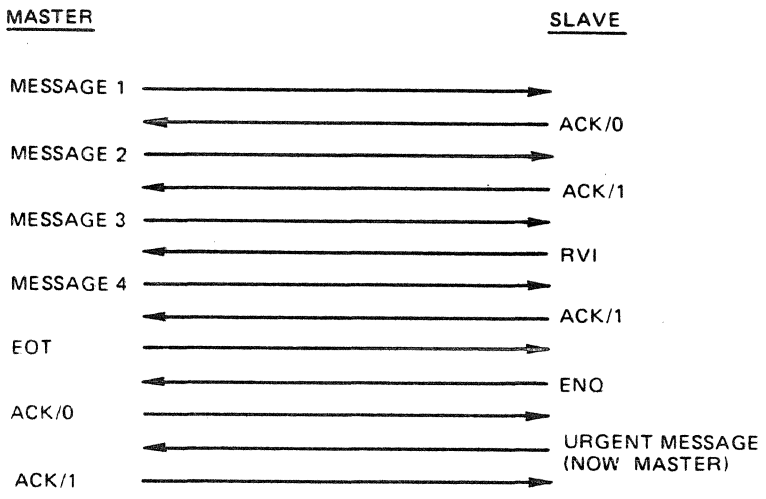


Figure 11-7. BSC Reverse Interrupt (RVI) Sequence Example

## BSC END OF TRANSMISSION (EOT) FEATURE

The application program, by any one of the following methods, can cause the BSC line protocol handler to send an end-of-transmission (EOT) message:

- At connect time, specify use of the control byte by setting to 0 bit 4 of the IORB's device-specific word I_DVS. When bit 4 of the first byte of the application's buffer (control byte, specified at write time) is set to 1, the BSC line protocol handler will send an EOT control character after the data in the application's buffer is successfully transmitted.

- When the control byte is not specified at connect time, set to 1 A-bit of the IORB's device-specific word I_DVS at write time. The BSC line protocol handler will send an EOT control character after the data in the application's buffer is successfully transmitted.

- After successful completion of a write request, issue a disconnect with or without a queue abort, and no physical disconnect. The master station will send an EOT character and give up its master status. However, when another IORB is queued for write, that station will again request its master status.

The application can detect receipt of an EOT control character in either of the following ways:

- If the control byte was specified at connect time, bit 4 of the control byte, of the read request on which the EOT was received, will be set to 1.
- If the control byte was not specified at connect time, bit 12 of the software status word I_ST, of the request on which the EOT character was received, will be set to 1.

With either method, the line protocol handler does not post any read requests queued before the EOT character was detected. To remove read requests from the queue, the application must issue a disconnect with a queue abort. The line protocol handler always posts the IORB with a device unavailable (B) return status (Table 4-10). The BSC line may or may not be available for further use, depending on whether or not an EOT character was sent abnormally.

## BSC SWITCHED LINE DISCONNECT (DLE EOT) FEATURE

A DLE EOT sequence is used to indicate the imminent intent of the transmitting station to do a physical disconnect. Use of this feature is selected at connect time by setting bit 5 of the IORB's I_DVS word to 1. If bit 5 is instead set to 0, the line protocol handler will transmit EOT instead of a DLE EOT. This transmission of the EOT will occur as described in the preceding description of EOT.

Reception and notification of a DLE EOT sequence, indicating pending line hang-up by the transmitting station, is performed in two ways:

- If the control byte was not specified at connect time, bits 9 and C are both set to 1 in the IORB's software status word for the read request on which the DLE EOT was received.
- If the control byte was specified at connect time, bits 3 and 4 of the control byte are both set to 1 for the read request on which the DLE EOT was received.

Transmission of a DLE EOT sequence is initiated for a disconnect request when the following two conditions are both true:

- Bit F of the disconnect IROB's device-status word is set to 0 (this is a request for a physical disconnect).
- Bit 5 of the connect IORB's device-specific word was set to 1.

Table 11-2 defines the conditions under which EOT or DLE EOT is selected for transmission or reported as having been received.

Table 11-2. Transmission and Reception Conditions for EOT and DLE EOT

| Receiving Station | | |
|---|---|---|
| Connect IORB Selected | Character Transmitted | Reported to Application |
| EOT | DLE EOT | EOT |
| DLE EOT | DLE EOT | DLE EOT |
| DLE EOT | EOT | EOT |
| **Transmitting Station** | | |
| Connect IORB Selected | Disconnect IORB Specified | Character Transmitted |
| EOT | Logical Hangup | EOT |
| EOT | Physical Hangup | EOT |
| DLE EOT | Logical Hangup | EOT |
| DLE EOT | Physical Hangup | DLE EOT |

## BSC Line Protocol Handler Timeout Interval

When a line is idle (no station controls the line), the timeout interval in waiting for a line-request bid is 10 minutes.

Once a station has successfully bid for a line, the timeout interval for subsequent reads (from the slave station) or writes (from the master station) is 12 seconds. These timeout intervals can be altered by using the TIMEOUT CLM directive.

## BSC Features Specific to BSC3780

The following subsections discuss the BSC3780 conversational reply feature, double-block feature, and the transmission /reception of BSC control characters.

### BSC3780 CONVERSATIONAL REPLY FEATURE

The conversational reply feature permits a BSC3780 application, after transmission of an entire message (whose last record is denoted by an ETX rather than an ETB), to selectively receive a message from a host computer without a preliminary line bid sequence.

The conversational reply sequence serves as the affirmative reply to the last message transmission block, and as a break or interrupt to later transmissions. The line protocol handler indicates to the application receipt of a conversational reply sequence in bit 5 of the IORB software status word I_ST, and/or in bit 2 of the control byte of the ETX write order.

In the following example, a BSC3780 application attempts to transmit three 2-record messages to a remote host computer. The transmission sequence is interrupted by the receipt of a conversational reply, which occurs after transmission of the second message. After the complete conversational reply (containing one or more records) is received, transmission of the third message can resume, following completion of a successful line bid sequence. Figure 11-8 illustrates the example sequence.

The application's use of the conversational reply feature requires that the application issue the requisite number of read orders (dependent on single or double-block mode) before the transmission of a text block that terminates with an ETX sequence. If the application does not issue the required read(s), the last text block is not transmitted, and the line protocol handler will initiate a temporary text delay (TTD) sequence until the necessary read orders are issued. If the application does not transmit an ETX sequence, it need not issue supporting read order(s).

BSC3780 DOUBLE-BLOCK FEATURE

The discussion under "BSC Double-block Feature" earlier in this subsection applies also to BSC3780 operation.

BSC3780 TRANSMISSION/RECEPTION OF BSC CONTROL CHARACTERS

In BSC2780 nontransparent mode, detection of any BSC control characters within a message would abort the transmission or reception of that message.

In BSC3780 nontransparent mode, selected, noncritical BSC con- trol characters (i.e., STX, SOH, DLE, NAK, and EOT) can be suc- cessfully transmitted and received.

## USING THE BSC2780/BSC3780 LINE PROTOCOL HANDLER

### BSC-Specific IORB Values

The BSC-specific IORB item I_CT2, device-specific word I_DVS, and software status word I_ST, are shown and defined in Tables 11-3 through 11-5, respectively. User-specified bits not specifically described in the tables must be 0. Section 4 has a general description of the IORB.

## Specifying Use of BSC2780/BSC3780 to the System

The inclusion of BSC2780/BSC3780 in the system is done during system build. The application can select and use either BSC2780 or BSC3780 according to the setting of bit 9 in the device-specific word I_DVS in the IORB (see Table 11-4).

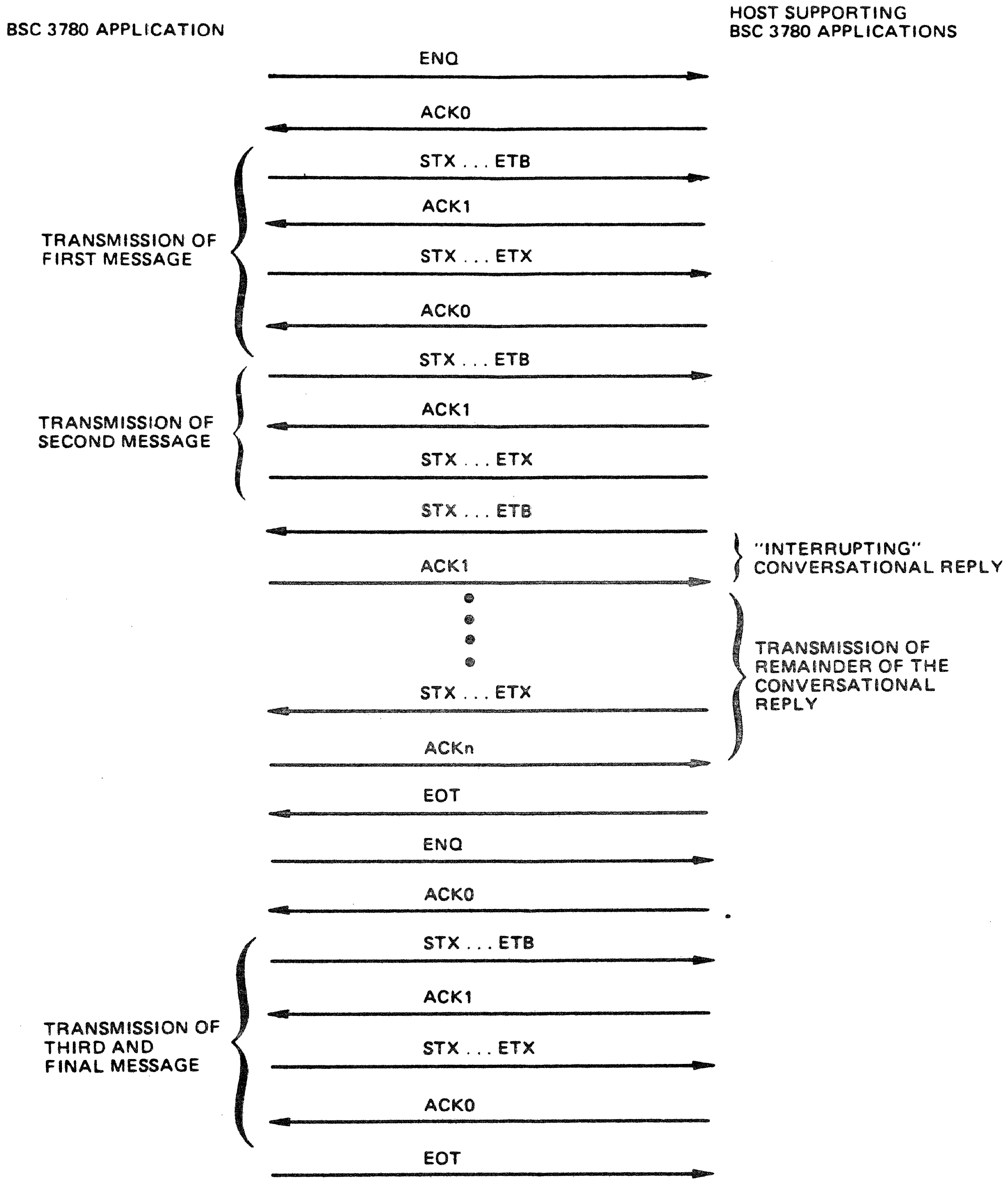

BSC 3780 APPLICATION
HOST SUPPORTING BSC 3780 APPLICATIONS
ENQ
ACK0
STX . . . ETB
ACK1
STX . . . ETX
ACK0
TRANSMISSION OF FIRST MESSAGE
STX . . . ETB
ACK1
STX . . . ETX
TRANSMISSION OF SECOND MESSAGE
STX . . . ETB
ACK1
"INTERRUPTING" CONVERSATIONAL REPLY
STX . . . ETX
ACKn
TRANSMISSION OF REMAINDER OF THE CONVERSATIONAL REPLY
EOT
ENQ
ACK0
STX . . . ETB
ACK1
STX . . . ETX
ACK0
EOT
TRANSMISSION OF THIRD AND FINAL MESSAGE

Figure 11-8. Example of Conversational Reply in BSC3780 Transmission Sequence

Table 11-3. Function Codes in I_CT2 Field in the IORB

| Function Code | Definition | Use |
|---|---|---|
| 0 | Wait online | Used by the line protocol handler to complete the description of the requested I/O function. |
| 1 | Write | |
| 2 | Read | |
| A | Connect | |
| B | Disconnect | |

Table 11-4. BSC Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| 0 | Must be zero |
| 1 | Must be zero |
| | For connect call only (function code A) |
| 2 | 0 = Do not use Auto Call Unit<br>1 = Use Auto Call Unit |
| 3 | Must be zero |
| 4 | 0 = Use control byte<br>1 = Do not use control byte |
| 5 | 0 = Do not support DLE EOT seqeuence<br>1 = Support DLE EOT sequence |
| 6 | Must be zero |
| 7 | 0 = Use single-block or double-block feature. Bit 7 must be zero for bit 8 to be meaningful.<br>1 = Use multi-block feature. (Bit 8 must be zero.)<br>For BSC2780: Both send and receive.<br>For BSC3780: For receive only. |
| 8 | 0 = Use single-block per transfer<br>1 = For BSC2780: Use double-block for send/receive<br>For BSC3780: Use double-block for receive |

Table 11-4 (cont). BSC Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| 9 | 0 = Use BSC2780 protocol<br>1= Use BSC3780 protocol |
| For write call only (function code 1) | |
| A | 0= Do not send EOT after this transmission<br>1 = Send EOT after this transmission |
| B | 0 = Do not send RVI if station is in slave status<br>1 = Send RVI if station is in slave status |
| C | 0 = Send data in nontransparent mode<br>1 = Send data in EBCDIC transparent Mode |
| D | 0 = Send ITB or ETB charcters following the data<br>1 = Send ETX characters following the data |
| For disconnect call only (function code B) | |
| E | 0 = Abort (dequeue) all IORBs on request queue<br>1 = Process outstanding requests on request queue |
| F | 0 = Disconnect line on completion. If bit 5 was set to 1 on connect, then send DLE EOT sequence.<br>1 = Do not disconnect line on completion |

Formats and Characteristics of BSC Input Data

The formats and characteristics of BSC input data for both ASCII and EBCDIC are described and illustrated below.

Figure 11-9 shows the format and contents of BSC input data received from another computer.

BSC CONTROL BYTE (RECEIVE)

When bit 4 of the IORB's device-specific word I_DVS is set to 0 at connect time (see Table 11-4), the BSC line protocol handler uses the first byte of the application's buffer as the control byte. Figure 11-10 shows the control byte's format and content.

Table 11-5. BSC Software Status Word I_ST in the IORB

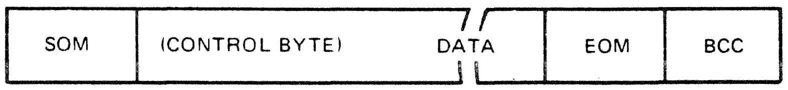
| Bit | Meaning When Bit Set to 1 |
|---|---|
| 0 | N/A |
| 1 | N/A |
| 2 | Data service rate error |
| 3 | Lost line bid; RVI received |
| 4 | Communications control block service error |
| 5 | Conversational reply received (BSC3780 only) |
| 6 | Long record |
| 7 | 0 = ITB and/or ETB characters received<br>1 = ETX character received |
| 8 | N/A |
| 9 | 0 = not meaningful<br>1 = DLE EOT received, if bit C is also 1 |
| A | Nonzero residual range |
| B | Phone hang-up |
| C | EOT character received |
| D | Transparent message received |
| E | NAK limit reached |
| F | Fatal error: bus parity or memory error |

## ASCII INPUT FOR BSC

ASCII input characteristics and format (Figure 11-9) are as follows:

1. SOM (start-of-message) consists of the STX control character only.

2. The control byte (if specified at connect time) is stored in the first byte of the application's buffer, and indicates the end-of-message (EOM) sequence. When bit 7 is 0, it indicates detection of an ITB or ETB control character; when 1, it indicates detection of an ETX character. Note that bit 7 of both the control byte and of I_ST are specified.



SOM (START OF MESSAGE)
A ONE- OR TWO-CHARACTER SEQUENCE THAT IS STRIPPED BY THE BSC LPH.

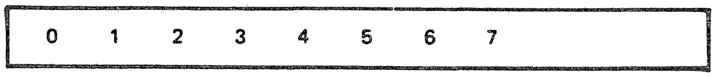
CONTROL BYTE
THE CONTROL BYTE, IF SPECIFIED, IS THE FIRST BYTE OF THE APPLICATION'S DATA.

DATA
INFORMATION STORED IN THE APPLICATION'S BUFFER AND SPECIFIED AT READ TIME.

EOM (END OF MESSAGE)
A ONE- OR TWO-CHARACTER SEQUENCE THAT IS STRIPPED BY THE BSC LPH.

BCC
AN LRC CHARACTER OR CRC CHARACTER THAT IS INSERTED BY THE BSC LPH.

Figure 11-9. BSC Input Data Format and Contents



BITS 0 THROUGH 2
NOT APPLICABLE; NOT EXAMINED

BIT 3 = 0
NOT MEANINGFUL

BIT 3 = 1
DLE EOT RECEIVED IF BIT 4 IS ALSO 1

BIT 4 = 0
DATA STORED IN APPLICATION'S BUFFER

BIT 4 = 1
EOT RECEIVED; NO DATA STORED IN APPLICATION'S BUFFER

BIT 5
NOT APPLICABLE; NOT EXAMINED

BIT 6 = 0
DATA RECEIVED IN NONTRANSPARENT MODE

BIT 6 = 1
DATA RECEIVED IN TRANSPARENT MODE

BIT 7 = 0
ITB OR ETB RECEIVED

BIT 7 = 1
ETX RECEIVED

Figure 11-10. Control Byte (Receive) for BSC Line Protocol Handler

3. Data must be 7-bit ASCII with odd parity. The BSC line protocol handler strips the parity bit and resets it to zero when it stores it in the application's buffer.

4. The EOM sequence, one of the three control characters ITB, ETB, or ETX, is indicated by bit 7 of the IORB software status word I_ST after a successful read is posted. See Table 11-5 for bit 7 indicators.

5. The BCC (block check character) is described in Section 7, "Line Protocol Handler Functions."

## EBCDIC INPUT FOR BSC

EBCDIC input format and characteristics are as follows:

1. SOM (start-of-message) consists of the STX control character only.

2. The control byte (if specified at connect time) is stored in the first byte of the application's buffer, and indicates the end-of-message (EOM) sequence, as follows:

| | |
|---|---|
| Bit 4 = 1 | End of transmission (EOT) detected. |
| Bit 7 = 0 | ITB or ETB character detected. |
| Bit 7 = 1 | ETX character detected. |

3. Data must be 8-bit EBCDIC; it will not have any BSC control characters.

4. The EOM sequence, one of the control characters ITB, ETB, or ETX, is indicated by bit 7 of the IORB software status word I_ST after a successful read is posted. See Table 11-5 for bit 7 indicators.

5. The BCC (block check character) is described in Section 7, "Line Protocol Handler Functions."

## TRANSPARENT EBCDIC INPUT FOR BSC

Transparent EBCDIC input format and characterisitcs are as follows:

1. SOM (start-of-message) consists of the two-character sequence DLE, STX.

2. The control byte, if specified at connect time, is stored in the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence according to the bit 7 setting (Figure 11-10).

3. Data may be any EBCDIC character, including BSC control characters.

4. EOM (end-of-message) sequence may be one of the following, indicated by bit settings of the IORB software status word I_ST, after a successful read has been posted:

I_ST Bits

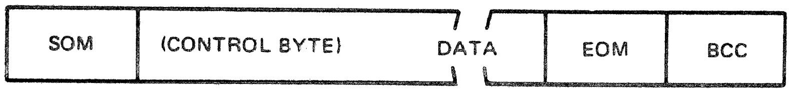
| D | 7 | Resulting EOM Sequence |
|---|---|---|
| 1 | 0 | DLE, ITB |
| 1 | 0 | DLE, ETB |
| 1 | 1 | DLE, ETX |

5. The block check character (BCC) is described in Section 7, "Line Protocol Handler Functions."

## Formats and Characteristics of BSC Output Data

Formats and characteristics of BSC output data (both ASCII and EBCDIC) are described and illustrated below.

Figure 11-11 shows the format and content of BSC data transmitted to another computer.



SOM
: A ONE- OR TWO-CHARACTER SEQUENCE THAT IS INSERTED IN FRONT OF THE DATA BY THE BSC LPH.

CONTROL BYTE
: THE CONTROL BYTE, IF SPECIFIED, IS STORED IN THE FIRST BYTE OF THE APPLICATION'S BUFFER.

EOM
: A ONE- OR TWO-CHARACTER SEQUENCE THAT IS INSERTED BY THE BSC LPH.

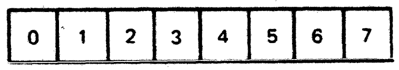
BCC
: AN LRC CHARACTER OR CRC CHARACTER THAT IS INSERTED BY THE BSC LPH.

DATA
: INFORMATION THAT IS TRANSMITTED FROM THE APPLICATION'S BUFFER BY THE BSC LPH.

Figure 11-11. Format and Content of BSC Output

### BSC CONTROL BYTE (SEND)

When bit 4 of the IORB's device-specific word I_DVS is set to 0 at connect time (see Table 11-14), the BSC line control handler uses the first byte of the application's buffer as the control byte. Figure 11-12 shows the format and content of the BSC line protocol handler's control byte for sending data.



BITS 0, 1
NOT APPLICABLE, NOT USED
BIT 2=1
CONVERSATIONAL REPLY RECEIVED
BIT 3=1
RVI RECEIVED (RETURN STATUS ONLY)
BIT 4=1
SEND THE DATA THAT IS IN YOUR BUFFER AND, AFTER IT HAS BEEN ACKNOWLEDGED, SEND EOT
BIT 5=1
SEND AN RVI RESPONSE ON THE NEXT ACKNOWLEDGMENT OF A READ
BIT 6=0
SEND NONTRANSPARENT EBCDIC
BIT 6=1
SEND TRANSPARENT EBCDIC OR ASCII
BIT 7=0
SEND ITB OR ETB
BIT 7=1
SEND ETX

Figure 11-12. Control Byte (Send) for BSC Line Protocol Handler

## BSC ASCII OUTPUT

ASCII output characteristics and format are as follows:

1. SOM (start-of-message) consists of only the STX character.

2. The control byte, when specified, is assumed to be the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence, which is either ITB, ETB, or ETX, designated as follows:

   a. Bit 6 must be 0.

   b. Bit 7 = 0. Send ITB or ETB. ITB is sent when the record is odd numbered (1, 3, 5, etc.) and the double-block feature is used.

      Bit 7 = 1. Send ETX.

   If the control byte is not specified, the EOM sequence is defined by I_DVS as described in 4 below.

3. Data must be 7-bit ASCII; it cannot have any BSC control characters.

4. EOM, which is either ITB, ETB, or ETX, can be indicated by the control byte (see 2 above) or by the C- and D-bits of the IORB device-specific word I_DVS (Table 11-4 as follows):

   a. C-bit must be zero.

   b. D-bit = 0. Send ITB or ETB. ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the double-block feature is used.

      D-bit = 1. Send ETX.

5. BCC (block check character) is described in Section 7, "Line Protocol Handler Functions."

## BSC EBCDIC OUTPUT

EBCDIC output characteristics and format are as follows:

1. SOM (start-of-message) consists of only the STX character.

2. The control byte, when specified, is assumed to be the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence, which is either ITB, ETB, or ETX, designated as follows:

   a. Bit 6 must be 0.

   b. Bit 7 = 0. Send ITB or ETB. ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the double-block feature is used.

      Bit 7 = 1. Send ETX.

      If the control byte is not specified, the EOM sequence is defined by I_DVS as described in 4 below.

3. Data may be 8-bit EBCDIC; it cannot have any BSC control characters.

4. EOM (end-of-message), which is either ITB, ETB, or ETX, can be indicated by the control byte (see 2 above) or by the C- and D-bits of the IORB device-specific word I_DVS (Table 11-11) as follows:

   a. C-bit must be zero.

   b. D-bit = 0. Send ITB or ETB. ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the double-block feature is used.

      D-bit = 1. Send ETX.

5. BCC (block check character) is described under "Line Protocol Handler Functions", earlier.

## BSC TRANSPARENT EBCDIC OUTPUT

Transparent EBCDIC output characteristics and format are as follows:

1. SOM (start-of-message) consists of the two-character sequence DLE, STX.

2. The control byte, when specified, is assumed to be the first byte of the application's buffer, and indicates the EOM (end-of-message) sequence, which is either DLE ITB, DLE ETB, or DLE ETX, designated as follows:

   a. Bit 6 must be 0.

   b. Bit 7 = 0. Send DLE ITB or DLE ETB. DLE ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the double-block feature is used.

      Bit 7 = 1. Send DLE ETX.

   If the control byte is not specified, the EOM sequence is defined by I_DVS as described in 4 below.

3. Data may be any EBCDIC character, including any BSC control characters.

4. EOM, which can be either DLE ITB, DLE ETB, or DLE ETX, can be indicated by the control byte (see 2 above) or by bit 4 and bit D of the IORB device-specific word I_DVS (Table 11-4) as follows:

   a. Bit 4 must be 1.

   b. D-bit = 0. Send DLE ITB or DLE ETB. DLE ITB is sent when the record is odd-numbered (1, 3, 5, etc.) and the double-block feature is used.

      D-bit = 1. Send DLE ETX.

5. BCC (block check character) is described in Section 7, "Line Protocol Handler Functions".

# *Section 12*
# *TTY LINE PROTOCOL HANDLER*

The TTY line protocol handler supports asynchronous terminal devices, generically classified as teleprinter-compatible (TTY), that include certain ASR, KSR, and visual information projection (VIP) terminals.

A basic TTY terminal consists of either a printer and keyboard or a VIP7100/VIP7200/VIP7800 display and keyboard. (Paper tape is not supported.) Each type of TTY terminal has an asynchronous communications interface that permits operation at up to 9600 baud.

## GENERAL TTY LINE PROTOCOL HANDLER OPERATION

### TTY Message Formats

Figure 12-1 illustrates TTY message formats. On input, the application receives only the text portion of the message. On output messages, the application can control print format with a control byte that is specified as the first character of the output buffer (in the IORB device-specific word I_DVS, described later). At connect, read, or write, the application can, with the I_DVS word, dynamically specify which message format is to be used.

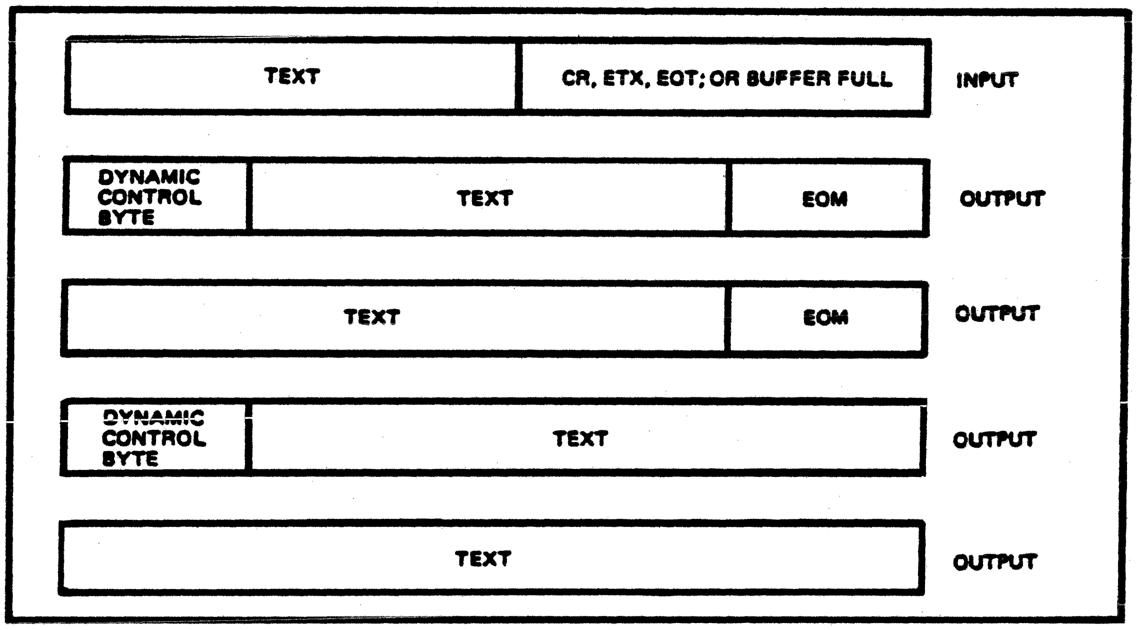


Figure 12-1. TTY Message Formats

## TTY Character Mode and Buffered Mode Transmission

### TTY CHARACTER MODE

Transmission for all TTY terminals is usually in character mode (one character at a time), a characteristic of the hardware that provides that:

- The TTY line protocol handler does all editing of data before any transmission.
- Multiple input lines are not allowed at the same time.

### TTY BUFFERED MODE (VIP7200 AND VIP7800)

For VIP7200 and VIP7800 only, the buffered mode, available as a hardware option, permits:

- The TTY line protocol handler to process multiple lines of input at the same time
- The operator to do local editing of data before transmission
- The application to instruct the TTY line protocol handler not to edit input data.

Buffered mode permits the TTY line protocol handler to process a write order while a read order is pending. A "quasi full duplex" operation gives the line protocol handler the ability to have the application sent to the terminal sequences that cause the terminal to send information back to the application's buffer.

Buffered quasi full duplex operates as follows:

1. When the channel control program (CCP) of the multiline controller (MLC) is currently processing a write order to the terminal, a subsequent read or write operation is not given to the CCP until the current write order completes.

2. When the CCP is processing a read order and the next order is a write order, that write order is processed while the read order is active.

3. When the write order (step 2) completes and the read order has not yet completed, a subsequent read or write order will not be processed until the read is completed. When the read order is completed before the write order, actions in 1 above take effect.

4. When the read order is completed, the line protocol handler returns to its original state, i.e., no orders pending. The line protocol handler can initiate read or write orders to the CCP.

## VIP7200 AND VIP7800 HARDWARE SWITCH OPTIONS WITH CHARACTER OR BUFFERED MODE

The TTY line protocol handler supports the following VIP7200/VIP7800 hardware switch options for character mode or buffered mode operation as follows:

| Character Mode | Buffered Mode |
|---|---|
| CHARACTER/BUFFER switch in CHARACTER position | CHARACTER/BUFFER switch in BUFFER position |
| Internal Even/Odd Parity switch set to select EVEN | Internal Even/Odd Parity switch set to select EVEN |
| HALF/FULL DUPLEX switch in FULL position | HALF/FULL DUPLEX switch in FULL position |
| | LINE/PAGE switch as required by user |
| | Internal end-of-message switch set to select ETX or EOT only |

VIP7200 AND VIP7800 FUNCTION AND CONTROL KEYS

Function and control keys on the VIP7200 and VIP7800 are supported only in buffered mode.

When issuing a write request that will cause an automatic response by the terminal, the application must first issue an asynchronous read request, then issue a write request that contains a control message to the terminal.

## TTY Line Protocol Handler Timeout Intervals

Table 12-1 lists the TTY line protocol handler's timeout intervals for the LPH functions. These timeout intervals can be altered by using the TIMEOUT CLM directive.

Table 12-1. TTY Line Protocol Handler Timeout Intervals

| Line Protocol Handler Function | Timeout Interval |
|---|---|
| Connect | Five minutes |
| Read | Character mode: Five minutes after receipt of the first character of the message<br>Buffered mode: Five minutes after the line protocol handler receives the request |
| Write | Thirty seconds |

## USING THE TTY LINE PROTOCOL HANDLER

### TTY-Specific IORB Values

The TTY-specific IORB item I_CT2, device-specific word I_DVS, and software status word I_ST are shown and defined in Tables 12-2, 12-3, and 12-4, respectively. User-specified bits not specifically described in these tables must be 0. Section 4 describes the general form of the IORB.

### Control and Characteristics of TTY Input Data

This subsection describes user control over the characteristics of TTY input data, and applies to character-mode processing unless otherwise noted.

Table 12-2. Function Codes in I_CT2 of the IORB

| Function Code | Definition | Use |
|---|---|---|
| 0<br>1<br>2<br>A<br>B | Wait online<br>Write<br>Read<br>Connect<br>Disconnect | Used by the line protocol handler to complete the description of the requested I/O function |

Table 12-3. TTY Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| 0 | Must be zero |
| 1 | Must be zero |
| For connect call only (function code A) | |
| 2 | 0 = Do not use Auto Call Unit<br>1 = Use Auto Call Unit |
| 3 | Must be zero |
| 4 | 0 = First byte in buffer on output is a control byte<br>1 = First byte in buffer on output is a data byte |
| For read call only (function code 2) | |
| 5 | 0 = Input data is in nontransparent mode<br>1 = Input data is in transparent mode |
| 6 | Must be zero |
| For write call only (function code 1) | |
| 7 | 0 = Stop output immediately on detecting a BRK received from the terminal<br>1 = Continue output when BRK detected |
| 8 | Must be zero |
| 9 | Must be zero |

Table 12-3 (cont). TTY Device-Specific Word I_DVS in the IORB

| Bit Number | Meaning of Bit Setting |
|---|---|
| For read call only (function code 2) | |
| A | 0 = Do not echo keyboard input<br>1 = Echo keyboard input |
| For read and write calls (function codes 2, 1) | |
| B | 0 = No LF (line feed) at end of message<br>1 = LF (line feed) at end of message |
| C | 0 = CR (carriage return) at end of message<br>1 = No CR (carriage return) at end of message |
| For connect call only (function code A) | |
| D | 0 = Data transfer is in character mode<br>1 = Data transfer is in buffered (block) mode |
| For disconnect call (function code A) | |
| E | 0 = Abort (dequeue) all IORBs on the request queue<br>1 = Process outstanding requests on the request queue |
| F | 0 = Hang up phone after disconnect<br>1 = Do not hang up phone after disconnect |

## TTY CONTROL BYTE (INPUT)

The description of the TTY control byte for output (see "TTY Control Byte (Send)" below) applies also to the TTY line protocol handler's control byte for input.

## TTY NONTRANSPARENT INPUT

TTY input is nontransparent when the application sets bit 5 of the IORB's device-specific word I_DVS (Table 12-3) to 0. Input is accepted until either the end-of-range or a CR (carriage return), ETX (end of text), or EOT (end of transmission) control character is reached. The CR, ETX, or EOT control character is not transmitted as part of the message.

## TTY TRANSPARENT INPUT

TTY input text is transparent when the application sets to 1 bit 5 of the device-specific word I_DVS at read time (Table 12-3). All input data, including any control characters, is stored in the buffer until end-of-range is reached.

Table 12-4. TTY Software Status Word I_ST in the IORB

| Bit | Meaning When Bit Set to 1 |
|---|---|
| 0 | N/A |
| 1 | N/A |
| 2 | Data service rate error |
| 3 | N/A |
| 4 | Communications control block (CCB) service error |
| 5 | No stop bit in character input |
| 6 | Long record |
| 7 | N/A |
| 8 | N/A |
| 9 | N/A |
| A | Nonzero residual range |
| B | Phone hang-up |
| C | N/A |
| D | N/A |
| E | N/A |
| F | Fatal error: bus parity or memory error |

## TTY LINE FEED (LF) AND CARRIAGE RETURN (CR) INPUT SEQUENCE

The application can specify at read time a sequence of LF and CR characters, with the B- and C-bits of the IORB's device-specific word I_DVS, as indicated in Table 12-3. When the message is received successfully, the specified character combinations are retransmitted back to the terminal.

## KEYBOARD INPUT CHARACTER AND LINE CONTROL

When an input character with a parity error is received, the line protocol handler sends a BEL character back to the terminal. The user must then retype that input character if it is to be included in the text being sent to the application.

The user can correct or delete erroneous characters or lines and can declare control characters to be data characters, as described below.

To correct one or more characters in the current line, i.e., before the CR is pressed, press the @ key. This deletes the character that immediately preceded the @ character, and displays the @ symbol. Each succeeding @ entry deletes another character, moving from right to left to the beginning of the line.

To delete the current line, i.e., before the CR is entered, press and hold the CTRL (control) key and press X. This deletes the current line, displays the message *DEL* on the next line, and results in a carriage return. The user can then enter a correct line.

To cause a control character (e.g., @, CTRL X, CR, and \) to be accepted as a data character (transparent mode) press the backslash (\) key before entering that control character. The system interprets the backslash as an escape character. In transparent mode, all input characters are data characters and have no editing functions.

TTY DISPLAY OF INPUT CHARACTERS

The user can cause an input character to be echoed to the terminal (displayed on the screen or typed on the console) by setting to 1 the A-bit of the device-specific word I_DVS (Table 12-3). For full duplex printers, the application need specify that characters be returned only when they are to be echoed by the system software.

TTY INPUT IN BUFFERED MODE (VIP7200 AND VIP7800 ONLY)

When the application at connect time sets to 1 the D-bit of the device-specific word I_DVS, input is accepted until an ETX or EOT control character or end-of-range is encountered.

When the application sets bit 5 of I_DVS to 1 at read time, TTY input in buffered mode is transparent, i.e., there is no editing. When the bit 5 is set to 0, TTY input in buffered mode is nontransparent; i.e., control characters are edited.

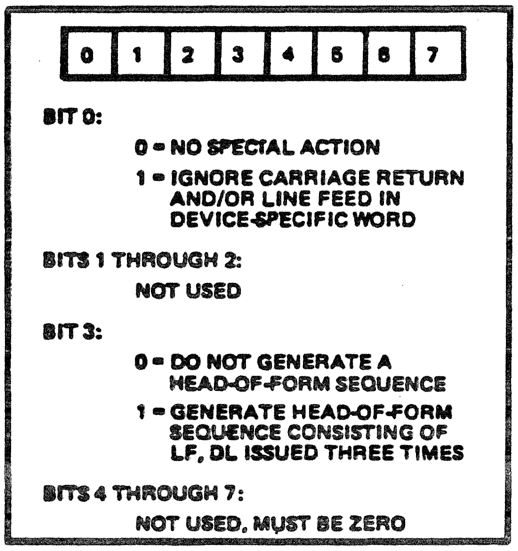
As in character mode, the application can specify an LF and CR sequence, as described above under "Line Feed (LF) and Carriage Return (CR) Input Sequence."

## Control and Characteristics of TTY Output Data

This subsection describes user control of the characteristics of TTY output data and is applicable to character-mode processing unless otherwise stated.

## TTY CONTROL BYTE (SEND)

The TTY line protocol handler's control byte, included as the first character of the application's buffer, controls the message's head-of-form sequence. At connect time, the application specifies the control byte by setting to 0 bit 4 of the IORB's device-specific word I_DVS (Table 12-3). Figure 12-2 shows the format and content of the TTY control byte.



BIT 0:

0 = NO SPECIAL ACTION

1 = IGNORE CARRIAGE RETURN AND/OR LINE FEED IN DEVICE-SPECIFIC WORD

BITS 1 THROUGH 2:

NOT USED

BIT 3:

0 = DO NOT GENERATE A HEAD-OF-FORM SEQUENCE

1 = GENERATE HEAD-OF-FORM SEQUENCE CONSISTING OF LF, DL ISSUED THREE TIMES

BITS 4 THROUGH 7:

NOT USED, MUST BE ZERO

Figure 12-2. Control Byte for TTY Line Protocol Handler

## END-OF-MESSAGE (EOM) SEQUENCE ON TTY OUTPUT

The EOM sequence is controlled by the B- and C-bits of the IORB's device-specific word I_DVS (Table 12-3), as specified by the application at write time. The TTY line protocol handler sends an EOM sequence according to the following B- and C-bit values:

| I DVS Bits | | |
|---|---|---|
| B | C | EOM Sequence |
| 0 | 0 | CR, DEL, DEL |
| 0 | 1 | DEL, DEL |
| 1 | 0 | LF, CR, DEL, DEL |
| 1 | 1 | LF, DEL, DEL |

At read time, the application can specify the same B- and C-bit values in order to send an EOM sequence back to the terminal when the message is successfully received.

## TTY DETECTION OF BRK CHARACTERS

When the application sets to 0 bit 7 of the device-specific word I_DVS at write time, the line protocol handler will immediately stop all output when it detects a BRK key character in the input stream from the terminal. The line protocol handler ignores the BRK character when bit 7 is set to 1, until the write order is completed.

## TTY OUTPUT IN BUFFERED MODE

Control and characteristics for TTY output in buffered mode are the same as described above for character mode. However, in processing in buffered mode (VIP7200/VIP7800 only), the line protocol handler processes all physical I/O requests in the same sequence as they are received. If there is already an outstanding read request, only a subsequent write request can be initiated before the read request is satisfied or the timeout for that read request is elapsed.

# *Section 13*
# *3270 TERMINAL FACILITY LINE PROTOCOL HANDLER*

The BSC 3270 Terminal Facility (BTF) line protocol handler supports ASCII and EBCDIC transmission between the 3270 type IBM controllers and a DPS 6 system.

The following IBM controllers are supported:

3271 models 1 and 2
3274 models 1C and 51C
3276 models 2, 3 and 4.

The following attached devices are supported:

3277 display model 2
3278 display models 2, 3, 4 and 5
3279 display models 2 and 3
3287 printer models 1 and 2.

The IBM controllers and devices listed above are described fully in the IBM literature. It is assumed that the reader is familiar with the operation of IBM 3270 type devices and BSC protocol.

This section includes the following topics:

- Software functions supported
- Software restrictions

- Modes of operation
  - TTY Mode
  - ROP Mode
  - Block Mode
  - Logical Terminal Mode
- IORB processing
  - device specific word
  - processing order
  - purging queued IORBs
- Break processing
- AID keys in TTY processing
- Read commands supported
- DARTS probe points table.

SOFTWARE FUNCTIONS SUPPORTED

The BTF supports the following functions of the IBM 3270 controllers:

- Multi-point configuration
- Modem, direct connect, and modem bypass interconnection
- Auto answer for switched network operation
- 2/4 wire lines
- Four selectable presentation control modes
  - TTY (terminal characteristics) mode
  - Block mode
  - ROP (read-only printer) mode
  - Logical Terminal mode
- Selectable data translation on a per order basis
- Selectable line poll frequency
- Support of 3270 device alternate buffer size through a CLM configurable input buffer size (Block mode only)
- Master LRN processing
- Supervisory message processing (Block mode only)
- Break processing (TTY mode only)

- Limited 7800 attribute interpretation (TTY mode only)
- Pre-editing (control byte) and post-editing selected by device specific words (DSW) bits
- Return of 3270 device status/sense inforamtion with each order
- Entry of 24 lines of input in the transmission (multiple line input) from the terminal.

The BTF is accessed indirectly through the File System. To read input from a terminal an application issues a Read Record request, either by issuing a Read Record macrocall and supplying parameters in an associated file information block (FIB), or by a User-in macrocall or high-level language statement. The File System translates the macrocall and FIB parameters into a $RQIO macrocall and associated read Input/Output Request Block (IORB).

The File System interface provides a sequential file interface to terminals. It can be used with any mode of the LPH, but since the application cannot access the File System's IORBs, those features which utilize the extended IORB format (i.e., Master LRN, 3270 Status/Sense Reporting and, in TTY mode, AID byte reporting) cannot be used.

## SOFTWARE FUNCTIONS RESTRICTED

The BTF restricts the following functions of the IBM 3270 controllers:

- Transparent mode is NOT supported; this implies that extended highlighting, color, programmed symbols and 14 bit addressing are not supported.
- Screen size is restricted to 1920 characters in TTY mode
- ASCII code terminals are not supported.

## MODES OF OPERATION

The BTF operates in four modes:

1. TTY mode, which supports line-at-a-time transfer of data to or from any of the above display models (CRTs).
2. ROP (receive-only printer) mode, which supports the transfer of data to the printer via the File System.
3. Block mode, which supports the transfer of 3270 datastream data to or from any of the supported devices.
4. Logical Terminal mode, which supports the transfer of data to and from another computer.

The mode is selected by means of a connect IORB and remains in effect until a disconnect IORB is received. The following subsections describe the uses of each mode.

## TTY Mode

TTY mode is the default operating mode for CRT terminals; the user need not specify this mode in the connect IORB Device Specific Word (DSW). This mode is used primarily by the File System, which treats a terminal (configured by means of the device directive) as a sequential file. In this mode a terminal can be used as the input and output file of a task group (i.e., user-in, user-out, command-in, and error-out).

### LINE AT A TIME INPUT AND OUTPUT

TTY mode provides for line-at-a-time input and output. The field or line following the last input or ouput operation is unprotected to allow input by the operator. All other fields on the screen are protected. The operator enters data into the unprotected field.and can perform editing with the terminal's own editing capabilities. When satisfied, the operator transmits it to the DPS6 by pressing ENTER.

### MULTIPLE LINE INPUT FEATURE

For situations in which interaction between the application and the operator is unnecessary, the BTF supports a multiple line input feature for inputting a large volume of data. The operator requests this mode by pressing the PA1 key. The operator can then enter and edit data on any of the 24 lines. When satisfied with the input, the operator transmits it to the DPS 6 by pressing ENTER. In order to be transmitted, a field must have been modified by the operator. The operator can return to the line-at-a-time method of operation pressing the PA1 key again.

### BREAK MODE PROCESSING

Break mode processing is supported in TTY mode through the PA2 key. If the terminal is in multiple line input mode when break processing is entered, it returns to normal line-at-a-time mode. When break mode is entered, any input at terminal not transmitted to the host is lost.

## ROP Mode

ROP mode is the default operating mode for printer terminals. The user need not specify this mode in the connect IORB DSW. Printout is formatted at the printer by form-feed (FF), carriage return (CR) and line feed (LF) characters. Note that the ASCII LF character is converted into the EBCDIC new line (NL) character. An end mark (EM) character is appended to the printer ouput. Post-order editing is not supported as an implied CR. LF is done at the end of a write operation to a printer.

A description of supported GCOS control bytes is listed in Table 13-1. The BTF retains write IORBs until one of the following occurs:

- The write IORB is posted with zero status and notification has been sent that the printer is ready for another write operator
- The write IORB is posted with an error status and notification has been sent that an error has occurred during printing
- The write IORB is posted with an error status indicating that a timeout has occurred and notification has not been sent.

Two types of read IORBs can be issued to printers:

1. Normal status reads - returns the last status/sense bytes returned by the printer.
2. Attention read - returned after a device status change.

## Block Mode

Block mode is intended for applications which make full use of the 3270 datastream facilities. Both CRTs and printers may be connected in Block mode.

Table 13-1. Supported GCOS Control Bytes.

| Hex | Meaning |
|---|---|
| 00-1F | Single line feed; print |
| 20 | No Print |
| 21-2F | Multiple (N) line feed |
| 30 | Form feed; no print |
| 31-3F | Single line feed; no print |
| 40 | Single line feed; print |
| 41-4F | Multiple (N) line feed; print |
| 50 | Form feed; print |
| 51-60 | Single Line Feed; Print |
| 61-6F | Multiple (N) Line Feed; Print |
| 70 | Single Line Feed; Print |
| 71-7F | Multiple (N) Line Feed; Print |

NOTE

(N) N = number of lines fed (range is from 1 to 15)

In Block mode writes and reads, the application has an option of specifying whether the data is in ASCII or EBCDIC. The operational differences between these two options are discussed below. The BTF directives allow you to specify if data is always ASCII (all translation done in the CCP), always EBCDIC (no translation required), or ASCII/EBCDIC (where translation is done in the LPH causing higher CPU usage). The ASCII and EBCDIC character sets and their equivalents are illustrated in Appendix D. Figure 13-1 shows the 3270 datastream.

## READ COMMANDS

Typically, an application should receive inbound data by issuing a read IORB to receive transferred information. In Block mode, the 3270 read commands, Read Buffer and Read Modified, are also supported. Read Buffer causes the entire buffer contents of the addressed device to be transferred back to the host and is used primarily for diagnostic purposes.

A major feature of Read Modified is null suppression. To execute a host initiated read command, the application must first issue a read IORB large enough to accommodate the inbound data stream. Then the read command is passed to the BTF in a write IORB. The completed read IORB buffer contains the received data stream. The Read Modified command should be used selectively because the general poll initiates a control unit generated Read Modified operation if an Attention Identification is generated with no status pending.

## ASCII CODE

ASCII is the default IORB option. The entire buffer is translated to or from ASCII with the exception of the following exempt bytes:

- The ESC, Command, WCC/CCC characters at the start of a write buffer

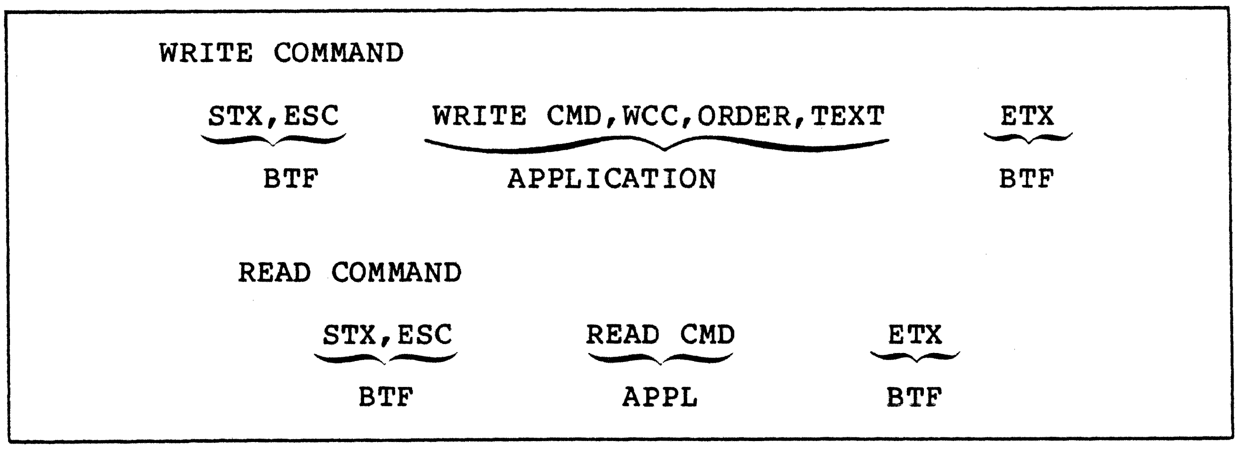


Figure 13-1. 3270 Data Stream

- The AID, Cursor 1, Cursor 2 characters at the start of a received message
- The buffer address bytes following the SBA, EUA and RA orders in the write buffer
- The attribute byte following the SF order in the write buffer
- The buffer addresss bytes following the SBA order in a received message.

In order to receive ASCII data in Block Mode the line should have been configured at CLM time as ASCII or ASCII/EBCDIC.

EBCDIC CODE

The BTF treats writes and reads specifying EBCDIC as messages to be passed directly between the application and the terminal. No inspection of message is done by the LPH. In order to receive EBCDIC data n Block Mode the line should have been configured at CLM time as EBCDIC or ASCII/EBCDIC.

## Logical Terminal Mode

Logical terminal mode allows the transfer of ASCII data to or from another computer.

On write orders the BTF adds a three byte header consisting of ESC.Erase_write.WCC before transmitting the message. On read orders all but the first five bytes of the message, a two byte address followed by three-bytes consisting of AID.cursor1,cursor2, is passed to the application.

Logical terminal mode is a connect time option rather than a CLM directive option to accommodate computers which require the ability to run full 3270 terminal emulation at one point, and a file transfer operation at another point.

## MASTER LRN PROCESSING

In order to use memory more efficiently, the BTF maintains a master LRN for each line. Master LRN processing requires an extended IORB.

The application issues a connect IORB using the extended IORB format, selecting master LRN in the left-hand byte of offset I_DV2. The application can then issue read IORBs to the master LRN. If there is input from a terminal connected in the master LRN mode and if there is no read IORB issued against the terminal's own LRN, then the input is transferred to the buffer of the first master LRN IORB. The LRN of the responding terminal is put in the left byte of offset I_DV2 of the IORB.

The application can select on a station basis whether to process any input from the terminal or operate in a two-way alternate (TWA) mode. For TWA mode, Operator Entry Time (OET) must be defined in the write IORB. A terminal is set in OET after a write to the terminal and reset after input is read from the terminal. Terminal input is only passed to the master LRN if the terminal is in OET.

## IORB PROCESSING

The BTF supports five logical functions:

- Connect
- Disconnect
- Read
- Write
- Break.

The device-specific word I_DVS is used in conjunction with each of the I/O functions. This word serves to specialize the activity of a particular function. For example, the setting of bit 15 in I_DVS determines whether the communication line is disconnected on completion of a disconnect function.

An application can issue one I/O order against a temrinal and wait for its completion or issue several IORBs. Outstanding read and wirte orders and non-abortive disconnects are queued sequentially. In all modes write IORBs have priority over read orders. A write order issued to a terminal whose operator is preparing input can cause the terminal to lose the input.

When a disconnect with queue abort is issued for an open terminal, the BTF purges queued IORBs and posts the incomplete orders back to the requesting applications. All queued read and write orders for that terminal are posted with a 010B status.

The IORB device-specific word I_DVS for connect, disconnect, read, and write calls are shown in Tables 13-2 through 12-5. The software status word I_ST and software return codes in I_CT1 are shown in Tables 13-6 and 13-7, respectively. Bits not explicitly described in the tables must be 0. Section 4 describes the general form of the IORB.

## AID KEYS

The read data stream begins with a heading consisting of the Attention Identification (AID) character and a two-character cursor address. The AID key default values are as follows:

| | | |
|---|---|---|
| ENTER (7D) | = | transmit screen information back from terminal |
| CLEAR (6D) | = | clear terminal screen |
| PA1 (6C) | = | enter into multiple line input in TTY mode |
| PA2 (6E) | = | exit from multiple line input in TTY mode |
| PA3 (6B) | = | break processing requested. |

Table 13-2. BTF Device-Specific Word I_DVS for connect calls

| Bit Number | Meaning of Bit Setting |
|---|---|
| 0 and 1 | 00 = Default mode (for CRT - TTY, for printers - ROP)<br>01 = Logical terminal mode<br>10 = Block mode<br>11 = Reserved for future use |
| 2 | 0 = Do not use autodial<br>1 = Use autodial |
| 3 and 4 | Must be zero |
| 5, 6, and 7 | Line poll interval:<br>000 = 1-second poll interval<br>001 = 1-second poll interval<br>010 = 2-second poll interval<br>011 = 3-second poll interval<br>100 = 4-second poll interval<br>101 = 5-second poll interval<br>110 = 15-second poll interval<br>111 = 30-second poll interval |
| 8 and 9 | Must be zero |
| A and B | Master LRN:<br>00 = Do not allow master LRN processing at this time.<br>01 = Allow master LRN processing.<br>10 = Reserved for future use.<br>11 = Allow master LRN processing, enforce two-way-alternate between line and application |

NOTE

The line poll interval is the interval the BTF pauses between polling all the control units configured on that line. The line poll interval is obtained from the first connect IORB for that line. Subsequent connect calls with different poll intervals are ignored.

Example: TPS 6 uses block mode and master LRN with two-way alternate processing. If the requested poll interval is one second, the DSW for a connect call is 8130.

Table 13-3. BTF Device-Specific Word I_DVS for disconnect

| Bit Number | Meaning of Bit Setting |
|---|---|
| C | 0 = -<br>1 = If terminal is in master LRN processing, purge a master LRN read |
| D | Must be zero |
| E | 0 = Purge all queued IORBs for this terminal<br>1 = Process queued IORBs before processing this disconnect |
| F | 0 = Hang up phone upon processing disconnect<br>1 = Leave phone connected |

Table 13-4. BTF Device-Specific Word I_DVS for read

| Bit Number | Meaning of Bit Setting |
|---|---|
| 0 | 0 = Normal status read<br>1 = Attention read |
| 1 and 2 | Must be zero |
| 3 | 0 = ASCII data wanted - translate it<br>1 = EBCDIC data wanted - no translation required |
| 9 | 0 = -<br>1 = Supervisory message, IORB |
| A | Must be zero |
| B | 0 = No line feed at end of message<br>1 = Line feed at end of message |
| C | 0 = Carraige return at end of message.<br>1 = No carriage return at end of message |
| D, E and F | Must be zero |

Table 13-5. BTF Device-Specific Word I_DVS for write

| Bit Number | Meaning of Bit Setting |
|---|---|
| 3 | 0 = ASCII data wanted - translate it<br>1 = EBCDIC data wanted - no translation required |
| 4 | 0 = Control byte present<br>1 = No control byte present |
| 5 | 0 = Process next operator entry<br>1 = Ignore next operator entry |
| 6 | 0 = -<br>1 = Dummy write - place terminal in OET |
| 7 and 8 | Must be zero |
| 9 | 0 = -<br>1 = Supervisory message, IORB |
| A | Must be zero |
| B | 0 = No line feed at end of message<br>1 = Line feed at end of message |
| C | 0 = Carraige return at end of message<br>1 = No carriage return at end of message |
| D, E and F | Must be zero |

Table 13-6. BTF Software Status Word I_ST in the IORB

| Bit | Contents of $R1 | Meaning When Bit is Set to 1 |
|---|---|---|
| 0 | 10B | No response from a read |
| 2 | 0 | Data service error (receive overrun) |
| 3 | 0 | RVI received |
| 4 | 107 | CCB service error |
| 5 | 0 | Break processing |
| 6 | 0 | Long record received |
| 8 | 107 | Excessive entries |
| A | 0 | Non-zero residual range |
| B | 10B | Phone hang up |
| C | 0 | Invalid response received |
| E | 0 | Busy received |
| F | 107 | FATAL error |

Table 13-7. BTF Return Codes in I_CTl in IORB

| Status byte | Meaning |
|---|---|
| 00 | No error; operation complete |
| 04 | Invalid argument(s) - improper set-p of IORB |
| 05 | Device not ready |
| 06 | Time-out on non-connect order |
| 07 | Hardware error |
| 0A | Controller unavailable |
| 0B | Device unavailable<br>- read/write purged by data status change<br>- read/write IORB purged by disconnect with queue abort |
| 0C | Inconsistent request<br>- connect order issued against a device that is currently connected<br>- read/write order issued against a device that is currently connected |
| 35 | CCP not configured for this communication controller |
| 36 | Telephone number required for autocall has not been supplied |
| 37 | Telephone numbers supplied for autocall not answered or busy |
| 38 | Sub_LRN not configured or enabled |
| 3F | Connect or disconnect in progress |

The BTF also allows you to alter the meaning of AID keys in TTY mode. Table 13-8 contains all recognized AID keys that are available to accommodate such need.

Table 13-8. Recognized AID keys in TTY mode

| AID | Hex (EBCDIC) | Read Modified Command Operation |
|---|---|---|
| ENTER | 7D | Rd Mod |
| PF1 | F1 | Rd Mod |
| PF2 | F2 | Rd Mod |
| PF3 | F3 | Rd Mod |
| PF4 | F4 | Rd Mod |
| PF5 | F5 | Rd Mod |
| PF6 | F6 | Rd Mod |
| PF7 | F7 | Rd Mod |
| PF8 | F8 | Rd Mod |
| PF9 | F9 | Rd Mod |
| PF10 | 7A | Rd Mod |
| PF11 | 7B | Rd Mod |
| PF12 | 7C | Rd Mod |
| PF13 | C1 | Rd Mod |
| PF14 | C2 | Rd Mod |
| PF15 | C3 | Rd Mod |
| PF16 | C4 | Rd Mod |
| PF17 | C5 | Rd Mod |
| PF18 | C6 | Rd Mod |
| PF19 | C7 | Rd Mod |
| PF20 | C8 | Rd Mod |
| PF21 | C9 | Rd Mod |
| PF22 | 4A | Rd Mod |
| PF23 | 4B | Rd Mod |
| PF24 | 4C | Rd Mod |
| PA1 | 6C | Short Rd |
| PA2 | 6E | Short Rd |
| PA3 | 6B | Short Rd |
| CLEAR | 6D | Short Rd |

Table 13-9 lists the offsets provided in patch ZQP327, which you can use to change the meaning of the existing AID keys. The right half byte of each word represents an AID key in hex, the left half byte represents the function of the AID key as follows:

00 = transmit or enter
01 = enter/exit into multiple-line-input
02 = reserved for future use
03 = break processing
04 = clear terminal screen
05 = no meaning.

Table 13-9. AID key Programming Offsets

| OFFSET | DEFAULT | MEANING |
|---|---|---|
| 005F | 7D00 | ENTER |
| 0060 | 6D04 | CLEAR |
| 0061 | 6C01 | PA 1 |
| 0062 | 6E02 | PA 2 |
| 0063 | 6B03 | PA 3 |
| 0064 | 7A05 | PF 10 |
| 0065 | 7B05 | PF 11 |
| 0066 | 7C05 | PF 12 |
| 0067 | 4A05 | PF 22 |
| 0068 | 4B05 | PF 23 |
| 0069 | 4C05 | PF 24 |
| 006A | C105 | PF 13 |
| 006B | C205 | PF 14 |
| 006C | C305 | PF 15 |
| 006D | C405 | PF 16 |
| 006E | C505 | PF 17 |
| 006F | C605 | PF 18 |
| 0070 | C705 | PF 19 |
| 0071 | C805 | PF 20 |
| 0072 | C905 | PF 21 |
| 0073 | C905 | PF 21 DUMMY |
| 0074 | C905 | PF 21 DUMMY |
| 0075 | F005 | TEST (or SYS REQ) |
| 0076 | F105 | PF 1 |
| 0077 | F205 | PF 2 |
| 0078 | F305 | PF 3 |
| 0079 | F405 | PF 4 |
| 007A | F505 | PF 5 |
| 007B | F605 | PF 6 |
| 007C | F705 | PF 7 |
| 007D | F805 | PF 8 |
| 007E | F905 | PF 9 |
| 007F | F905 | PF 9 DUMMY |

## DARTS PROBE POINTS

Strategic TIPS are incorporated into the BTF code so that desired data collection is possible when DARTS is invoked. Use these steps to collect information.

1. Add the following CLM directive to your CLM_USER file:

   ```
   DRIVER ZQDART,LRN,LEVEL,X'FFC0'
   ```

2. Create the data base file >>SID>BULLSEYE

```
TOE 35
BFR 1000 10 5
LOG >>UDD>DARTS_LOG
```

3. Create the log file >>UDD>DARTS_LOG

4. Invoke DARTS

```
$S DARTS
```

5. Specify the TICS file

```
0 XEC >>SID>ZQP327.TICS
```

6. Activate desired probe points for data collection

```
0 ACT C5XX
```

   where XX is either a two-digit hex number, or ** to indicate all TIPS.

7. Terminate DARTS activity

```
0 XIT
```

8. Collected data can be printed from group $H:

```
$H DARTS FMT >>UDD>DARTS_LOG -OUT !LPT00
```

Table 13-10 contains a list of available DARTS probe points. For further information about DARTS refer to the <u>GCOS 6 Data Base Augmented Real-Time Tracing System User's Guide.</u>

Table 13-10. DARTS Probe Points

| TIPS | Meaning |
|---|---|
| C500 | LPH Entry |
| C501 | Read IORB |
| C502 | Multiple Line Input Operation |
| C503 | Connect IORB |
| C504 | Write IORB |
| C505 | Disconnect IORB |
| C506 | CQB Terminate |
| C507 | Issue CQB(s) |
| C508 | Issue CQB(s) |
| C509 | Fatal Error |
| C50A | Setting up CQB |
| C50B | Event monitoring; task level processing |
| C50C | Post read/write IORB |

# *Appendix A*
# *TRAP HANDLING*

A trap is a special software or hardware related condition that may occur during execution of a task. Traps include such conditions as a program error, memory defect, arithmetic overflow, or the issuance of an instruction that calls for hardware/software not configured into the system. Table A-1 lists the traps to which the system's hardware/firmware responds.

The design of any application program should provide that when a trap occurs, the hardware/software response will include calling a dedicated software routine (a trap handler) to react to the trap. When trap handlers are provided, the task that caused the trap may now handle the trap in a systematic and orderly way.

## TRAP SAVE AREAS

Trap handling routines make use of trap save areas (TSAs). A trap save area is a 104-word data structure that contains the following:

- The contents of several registers. These registers are available for use by the trap handling routine because their contents can be restored upon the routine's completion.
- The instruction associated with the trap.
- The address contained in the program counter when the trap occurred. This is the address to which a return is made when the trap handler routine terminates.

## Table A-1. Contents of Selected Words of Trap Save Area When Trap Occurs

| Trap Number and Condition | Specific Event | Saved Instruction | Saved Z-Word* | Saved A-Word | Saved Program Counter |
|---|---|---|---|---|---|
| 0 Cleanup trap | One of the following: ● Trap condition for which no other trap is enabled ● Abort Task Group command | | | | |
| 1 Monitor call; implicitly handled by the operating system | MCL instruction | 0001 | 80x1 | Unspecified | Next location |
| 1 Software trap | PI Command | Unspecified | Unspecified | Unspecified | Unspecified |
| 2 Breakpoint instruction | BRK instruction | 0002 | 90x1 | Unspecified | Next location |
| 3 Scientific floating point operation when SIP hardware not in system | Scientific instruction whose address expression generates a reference to a register | Instruction that caused trap | 80x1 | Unspecified | Unspecified |
| | Scientific instruction whose address expression generates a reference to memory | First word on instruction that caused trap | 00xy | Effective address generated by address expression | Next instruction |
| 4 Unrecognized op code | Instruction not recognized by CPU or SIP | Same as for trap 5 | Same as for trap 5 | Effective address of trap operand | Next instruction |
| 5 Scientific Branch instruction when SIP hardware not in system, or any other operation not supported | Undefined instruction whose address expression generates a reference to a register | Instruction that caused trap | 80x1 | Unspecified | Next instruction |
| | Undefined instruction whose address expression generates a reference to memory | First word of instruction that caused trap | 00xy | Effective address generated by address expression | Next instruction |
| 6 Integer arithmetic overflow (with appropriate overflow trap enable bit of M1 register set to 1) | Overflow of target R-register during execution of instruction whose address expression generates a reference to a register | Instruction that caused trap | 80x1 | Unspecified | Next instruction |
| | Overflow of target R-register during execution of instruction whose address expression generates a reference to memory | First word of instruction that caused trap | 00xy | Effective address generated by address | Next instruction |
| 7 Scientific divide by zero | A scientific divide (SDV) instruction has a divisor of zero | Unspecified | Unspecified | Pointer to scientific instruction that caused trap | Next CPU instruction |
| 8 Exponential overflow | A scientific operation produces an exponent greater than +63 | Unspecified | Unspecified | Pointer to scientific instruction that caused trap | Next CPU instruction |

Table A-1 (cont). Contents of Selected Words of Trap Save Area When Trap Occurs

| Trap Number and Condition | Specific Event | Saved Instruction | Saved Z-Word | Saved A-Word | Saved Program Counter |
|---|---|---|---|---|---|
| 13 Unprivileged use of privileged operation | HLT, RTCN, RTCF, WDTN, or WDTF[c] instruction | Instruction that caused trap | 8001 | Unspecified | Next location |
| | LEV instruction whose address expression generates reference to a register | Instruction that caused trap | 8001 | Unspecified | Next instruction |
| | LEV instruction whose address expression generates a reference to memory | First word of instruction that caused trap | 000y | Effective address generated by address expression | Next instruction |
| | Input/output instruction whose first-word address expression generates a reference to a register | First word of instruction that caused trap | 8002 | Unspecified | First word of instruction that caused trap, plus 2 |
| | Input/output instruction whose first-word address expression generates a reference to memory | First word of instruction that caused trap | 000y | Effective address generated by address expression | First word of instruction that caused trap, plus y |
| 14 Unauthorized reference to protected memory | Instruction that refers to a memory segment with a ring privilege higher than the executing task's | First word of instruction that caused trap | 00xy | Effective address generated by address expression | Next instruction |
| 15 Reference to unavailable resource | Instruction whose address expression generates a reference to (1) a memory address higher than the highest memory address available but less than 64K or (2) through indexing, a "wraparound" memory address higher than 64K or less than 0 | First word of instruction that caused trap | 00xy | Effective address generated by address expression | Next instruction |
| | Input/output instruction that specifies an improper channel number; address expression generates a reference to a register | First word of instruction that caused trap | 808y | Unspecified | First word of instruction that caused trap, plus y |
| | Input/output instruction that specifies an improper channel number; address expression generates a reference to memory | First word of instruction that caused trap | 008y | Effective address generated by address expression | First word of instruction that caused trap, plus y |
| | WDTN or WDTF instruction and watchdog timer not installed | 0006 (WDTN); 0007 (WDTF) | 80x1 | Unspecified | Next location |

Table A-1 (cont). Contents of Selected Words of Trap Save Area When Trap Occurs

| Trap Number and Condition | Specific Event | Saved Instruction | Saved Z-Word* | Saved A-Word | Saved Program Counter |
|---|---|---|---|---|---|
| 16 Program logic error | RTT instruction while TSAP contains a null pointer | Instruction that caused trap | 80x1 | Unspecified | Next instruction |
| | Instruction whose address expression illegally generates reference to a register (i.e., this instruction is not permitted to use a register address syllable) | Instruction that caused trap | 80x1 | Unspecified | Next instruction |
| 17 Bus parity or memory error | Bus parity error or unrecoverable memory data error | Unspecified | Unspecified | Unspecified | Unspecified |
| 19 Scientific underflow | An operation produces an exponent value of less than -64 while the associated enable bit in register M5 is set | Unspecified | Unspecified | Pointer to scientific instruction that caused trap | Next CPU instruction |
| 20 Program error (SIP) | A program error is detected by the SIP | Unspecified | Unspecified | Pointer to scientific instruction that caused trap | Next CPU instruction |
| 21 Scientific significance error | An integer is truncated during floating-point-to-integer conversion while the associated enable bit in register M5 is set | Unspecified | Unspecified | Pointer to scientific instruction that caused trap | Next CPU instruction |
| 22 Scientific precision | The nonzero portion of a fraction is truncated while the associated enable bit of register M5 is set | Unspecified | Unspecified | Pointer to scientific instruction that caused trap | Next CPU instruction |
| 23 Nonexistent resource error | The SIP or Commercial CP attempts a write or read request bus cycle and receives a NAK | Unspecified | Unspecified | Pointer to scientific instruction that caused trap | Next CPU instruction |
| 24 Noncorrectable memory error or Megabus error | A read error occurs which the EDAC cannot correct, or the SIP or CIP detects a parity error | Unspecified | Unspecified | Unspecified | Unspecified |
| 25 Commercial CIP divide by zero | The divisor of a decimal divide instruction (DDV) is equal to zero | Unspecified | Unspecified | Pointer to Commercial CP instruction that caused trap | Next CPU instruction |

Table A-1 (cont). Contents of Selected Words of Trap Save Area When Trap Occurs

| Trap Number and Condition | Specific Event | Saved Instruction | Saved Z-Word[a] | Saved A-Word | Saved Program Counter |
|---|---|---|---|---|---|
| 26[d] Commercial CP illegal specification | Any of the following: <br>• Undefined CIP op code detected. <br>• One or both descriptors of an alphanumeric instruction is packed decimal. <br>• Decimal operand has a zero length. <br>• Operand in an Edit, VRF, or SRH instruction has a zero length. <br>• A separate signed decimal operand consists of only a sign (i.e., no digits). <br>• In a Move and Edit instruction, the length of the receiving field was not exhausted, but either there are no micro-ops or the sending field length is exhausted. <br>• A second data descriptor specifies an IMO, except for DCM and ACM instructions. <br>• The first data descriptor in a DSH specifies an IMO. <br>• A third data descriptor specifies an IMO. <br>• In an SRH instruction, the search length list is less than the search argument list, or operand length less than operand element length. <br>• In a VRF instruction, verify list length is less than verify argument length, or operand length is less than operand element length. | Unspecified | Unspecified | Pointer to Commercial CP instruction that caused trap | Next CPU instruction |

Table A-1 (cont). Contents of Selected Words of Trap Save Area When Trap Occurs

| Trap Number and Condition | Specific Event | Saved Instruction | Saved Z-Word[a] | Saved A-Word | Saved Program Counter |
|---|---|---|---|---|---|
| 27[d] Commercial CP illegal character | Any of the following: <br>● Illegal decimal digit detected (low-order four bits are not 0 through 9). <br>● Illegal sign digit detected (not a recognized sign value). <br>● Illegal overpunch digit detected. | Unspecified | Unspecified | Pointer to Commercial CP instruction that caused trap | Next CPU instruction |
| 28[d] Commercial CP truncation error | Receiving field of an alphanumeric instruction cannot contain all characters of the result. Whether or not a trap occurs, the receiving field is altered to contain the leftmost part of the result and the CI (TR) is set. | Unspecified | Unspecified | Pointer to CIP instruction that caused the trap | Next CPU instruction |
| 29 Commercial CP overflow | Any of the following: <br>● Receiving field of a decimal instruction cannot contain all significant digits of the result. <br>● During a Shift Lift instruction, a nonzero digit is shifted out. | Unspecified | Unspecified | Pointer to CIP instruction that caused the trap | Next CPU instruction |
| 48 Software trap | BREAK command | Unspecified | Unspecified | Unspecified | Unspecified |
| 49 Software trap | UW command | Unspecified | Unspecified | Unspecified | Unspecified |
| 53 Software trap | Resumption of power by automatic power resumption faciltiy | Unspecified | Unspecified | Unspecified | Next location |

[a]The Z-word format is described later in this section.

[b]This is the address of the high-order (leftmost) end of a two-word operand.

[c]If the watchdog timer is not present, this instruction causes a trap to vector 15 regardless of the privilege mode of the central processor.

[d]The Assembly Language Reference manual describes Commercial Central Processor trap handling in detail.

- Two words of additional information related to the trap.
- Trap handler work space (92 words).

The number of TSAs built by the system is determined by the value that the user gives the TSA argument of the SYS directive when configuring the system (refer to the System Building and Administration manual).

## TRAP HANDLING DURING TASK EXECUTION

There are several kinds of traps, as follows:

1. Traps handled by the system exclusively; Monitor Call is currently the only trap of this type.
2. Traps handled first by the system, then possibly by the user. These include Trace/Break if Debug is used, or SIP when the simulator is present.
3. Traps, if enabled, handled by the user program; otherwise, by the system's default trap handler.
4. Software generated traps, described below.
5. Cleanup (trap 0); not really a trap since there is no TSA (trap save area), which is indicated when $B3 is set to null and $R3 is set to zero. The condition causing trap 0 is the occurrence of a task abort. The task's trap 0 trap handler will also be entered from traps for which no user trap handler is enabled after the standard system default trap handler has processed the trap. In this case, $R1 contains the number of the unenabled, causative trap; other registers are unchanged. Return from Trap (RTT) execution is not possible.

In cases 2 and 3 above, which go to the user program, $R3 contains the trap number; $B3 contains a pointer to the TSA.

### Software Generated Traps

Software generated traps comprise the following:

- Program Interrupt (trap 1) - Caused by the PI command or signal trap ($SGTRP) macro call.
- Unwind (trap 49) - Caused by the Unwind command.
- Suspend (trap 48) - Caused by the system's break handler, BREAK function, or by the signal trap ($SGTRP) macro call. The system suspends the task when no handler is provided.

- Power resumption notification (trap 53) - Caused by automatic resumption of power after power failure on systems configured with the power resumption facility.

To receive the PI, Suspend, or power resumption notification trap, the user program must enable it with the $TRPHD and $ENTRP macro calls.

## Program Use of Traps

The average program requires that the trap handler address be set (with the $TRPHD macro call), and that the "cleanup trap" (trap 0) be enabled with the $ENTRP macro call. In more complex situations, requiring more than one cleanup action and, consequently, more than one trap handling routine, the trap handler address can be altered by means of the $TRPHD macro call.

To respond to Program Interrupt (PI), trap 1 must be enabled with the $ENTRP macro call. The trap handler distinguishes between Program Interrupt and cleanup (trap 0), by comparing $B3 with null (see above). In simple programs, for Program Interrupt to resume execution at some other location, the saved P-counter in the trap save area (TSA) must be set, and a Return from Trap (RTT) instruction executed. For more complex programs, the user program should set a flag, then execute a Return from Trap (RTT) instruction. The user program must then examine the flag at appropriate places to avoid interrupts at inappropriate times (e.g., in the middle of a write function).

Alternatively, trap 1 is not enabled; cleanup checks $R1 for X'0301' (the error message signifying that no trap handler exists for a Program Interrupt condition), then branches to the desired location. When cleanup occurs, cleanup (trap 0) is automatically disabled; it may be reenabled when required.

## CONTENTS OF TRAP-RELATED MEMORY AREAS

In examining a dump to determine the nature of a trap condition, check particularly the contents of the TSA. The TSA and related memory areas are illustrated in Figure A-1; their contents are described below.

- Trap Save Area Link - When the trap save area is in use, TSAL contains a null pointer (if this is the only or last trap save area connected) or it points to the next trap save area connected. The next TSA connected would be used for handling a trap condition encountered by the trap handling routine (i.e., a nested trap).

- I-Register - The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler. The high-order byte contains the quantity ($40_{16}$ - trap number).

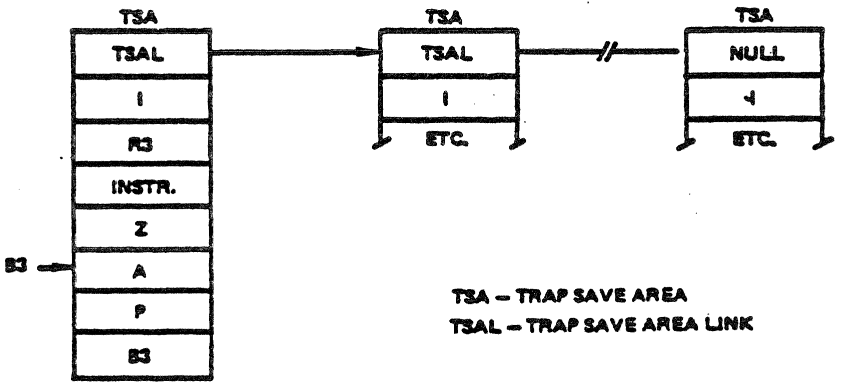


Figure A-1. Trap Handling Mechanism

- R3 Register - The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler.
- Instruction - The hardware/firmware stores the instruction associated with the trap. If a multiword instruction is involved, the first word is saved.
- Z-Word - This word contains miscellaneous information relative to the trap. The format of this word is shown below:

| BIT: 0 | 1 3 | 4 7 | 8 9 11 | 12 15 |
|---|---|---|---|---|
| R | 0 0 0 | BI | P R 0 0 | IS |

R - If R=0, the saved contents of the A-word are meaningful relative to this trap condition; if R=1, the saved contents of the A-word are <u>not</u> meaningful.

BI - 4-bit field that is meaningful only when an indexed bit or byte instruction is associated with the trap. If an indexed <u>bit</u> instruction is involved, BI indicates the four low-order bits of the associated index register; bit 7 of BI stores the least significant bit. If an indexed <u>byte</u> instruction is involved, bit 4 of BI indicates the least significant bit of the associated index register, and bits 5 through 7 are zeros.

PR - The privilege state of the task that was running when the trap occurred. 00 or 01 = nonprivileged state; 11 or 10 = privileged state. The value is taken from the P-bit of the S-register.

IS - The length (in words) of the instruction associated with the trap. If a multiword instruction is involved and the trap occurs before the entire instruction has been fetched, IS indicates the number of words that were fetched before the trap.

- A-Word - In many cases, this location contains an address associated with the trap. (This location is not meaningful if bit 0 of the Z-word contains a 1.) The nature of the saved address is governed by the specific trap condition and the specific instruction associated with the trap. Details relative to each trap condition are in Table A-1.

- Program Counter - The contents of the program counter are saved by the hardware/firmware when a trap occurs. This is the address to which a return is made when the trap handler completes. In most cases, the program counter will point to the instruction or location following the instruction associated with the trap. However, when an input/output instruction is involved, the program counter may point to an address <u>within</u> the instruction; in this case, the trap handler must modify this word before issuing a return to "normal" task processing.

- B3 Register - The contents of this register are saved by hardware/firmware when a trap occurs. This register is then available for use by the trap handler; as the trap handler is entered, the B3 register points to the A-word in the trap save area.

## SYSTEM SUPPLIED TRAP HANDLERS

The following software components provide trap handling facilities:

- Debug program
- Scientific Simulator
- Defective memory trap handler
- Default trap handler.

Traps handled by these system components can be passed onto user-written trap handlers, as explained later in this appendix.

### Trap Handling by the Debug Program

The Debug program operates as a task within the user group (Multi-User Debugger) or as a task group identified by $D ($D DEBUG). For a detailed description of the Multi-User Debugger and $D DEBUG, refer to the <u>Application Developer's Guide</u>. In this appendix, both debuggers are referred to collectively as the Debug program.

Once the Debug program is loaded, you may set, clear, or list breakpoints in the task code by use of Debug directives. When the application program is executed, Debug is activated by trap number 2, which occurs each time a breakpoint is encountered. The action specified by the Debug directive for that breakpoint will then be executed. For example, designated memory locations can be printed out and execution of the application program continued without operator intervention. Information can be printed on a console or a line printer.

## Trap Handling by Scientific Simulator

When a system's configuration does not include a Scientific Instruction Processor (SIP), this hardware component can be simulated by the Scientific Branch Simulator and the Floating Point Simulator, which, together, make up the Scientific Simulator.

### FLOATING-POINT SIMULATOR

The Floating-Point Simulator reacts to trap number 3 (scientific operation not in hardware), which occurs whenever the central processor encounters a nonbranch scientific instruction during task processing.

While processing scientific instructions, the simulator provides automatic alignment of the operand's hexadecimal mantissas. It achieves maximum available precision by requiring that mantissas have no leading zeros (i.e., all mantissas must be normalized).

Note the following programming consideration for the simulator:

- During its processing, the simulator may encounter an error condition related to a scientific instruction; the following can then occur:
  - The simulator consults trap vector 5 if it encounters a nonscientific instruction or other unrecognized instruction.
  - The simulator consults trap vector 7 if an SDV (Scientific Divide) instruction has a divisor of 0. The instruction will not be executed.
  - The simulator consults trap vector 8 if execution of a scientific instruction produces exponential overflow. The instruction will have been executed.
- To use a software routine to react to any of these trap conditions, you must provide a user-written trap handler. The simulator will be invoked to handle traps caused by execution of scientific instructions only if the trap numbers have been enabled for the task executing those instructions.

- No "overflow trap enable" bit of the M1 register should be set to 1 as the simulator begins operation.

## SCIENTIFIC BRANCH SIMULATOR

The Scientific Branch Simulator reacts to trap 5. It provides FORTRAN and Assembly language programs with the means to simulate the use of the scientific branch instructions.

The choice of the single-precision version (SSIP), or the double-precision version (DSIP) of the simulator is indicated in an argument of the system building SYS directive. Note the following programming considerations relative to the simulator:

For SSIP only:

- The simulator uses registers R4, R5, and R7 as scientific accumulator (S1) for comparisons; it uses R1, R2, and R3 as work registers.

- The simulator uses the G, L, and U bits of the I register to determine if the branch condition is true or false. When a normal return is made to the user program, the branch will be executed if the branch condition is true; otherwise, the next sequential instruction following the one that was trapped will be executed.

For both SSIP and DSIP:

- All other operation codes not handled by the Floating-Point Simulator or the Scientific Branch Simulator are passed to the next trap handler in trap 5.

### Defective Memory Trap Handler

The defective memory trap handler performs the following:

- Identifies to the user the physical and virtual address of defective memory.

- Informs the user whether or not the system remains operable after the detection of defective memory.

- Ensures that the area of defective memory will not be reallocated after its detection.

The user loads the defective memory trap handler at system configuration time by entering the LDBU directive and specifying the simple pathname to the bound unit ZXDEFM (refer to the System Building and Administration manual). The defective memory trap handler is automatically loaded by CLM as long as the bound unit ZXDEFM is in the system initialization directory (>>SID). If ZXDEFM is not in SID, defective memory traps will be handled by the default trap handler.

The defective memory trap handler responds to detection of defective memory by the following components:

- Central Processor Unit
- Scientific Simulator
- Input/Output controller.

If defective memory is detected by any of these three components, and the system is able to continue, the following message is sent to the operator's console, specifying the physical and virtual address of the defective memory:

```
PROBABLE MEMORY FAILURE, PHYSICAL ADDR=  ,VIRTUAL ADDR=
```

If the defective memory is CPU-detected (trap 17) and no user-written trap was enabled for trap 17, an X'0311' error message is also issued and the trapped program terminates.

If a user-written trap handler <u>is</u> enabled for trap 17, the defective memory trap handler ensures that the 32-word area containing the defective memory will not be reallocated to another task, and control is passed to the user-written trap handler, which normally returns task resources and terminates the task request.

If the defective memory is detected by the Scientific Simulator (trap 24), and, if no user-written trap handler is enabled for trap 24, the X'0318' error message is issued (see the <u>System Messages</u> manual) and the trapped program terminates.

A defective memory trap resulting from a file system I/O order produces the probable memory failure message followed by an X'0107' error message (see the <u>System Messages</u> manual).

If defective memory is detected, and the system is <u>unable</u> to continue, register contents are as follows:

```
$R1 - X'DEFA' (defective memory address)
$B1 - physical address of defective memory
$B2 - virtual address of defective memory
```

Knowledge of the address of defective memory permits the user to map the defect onto a specific memory board, which can then be replaced.

Whenever memory is found defective, it is returned to the memory manager and marked as unavailable for reallocation. Before memory can be returned to the memory manager, it must be relinquished by all of its users. For that reason, if memory found defective is within a shared area, such as a sharable bound unit or group control block, each task sharing that memory is liable to be trapped and terminated.

When defective memory is marked unavailable for reallocation, at least 32 words are so marked. Trap 17 and 24 identify the exact location of memory detected as defective. I/O controller detection is less precise since it knows that only some location within the buffer is defective. In this case the memory manager makes unavailable all pages containing any part of the suspect buffer. The address cited in the probable memory failure error message is the beginning of the suspect buffer.

### System Default Trap Handling

When a trap condition occurs in task code that has not enabled this particular trap or trap 0, an error message is written to the error-out file; the delete bit in the task control block is reset, the task is terminated, but the task's resources (memory and peripherals) are not released. Thus, a memory dump can be taken so that the error condition can be examined.

## USER-WRITTEN TRAP HANDLERS

User-written trap handlers are either task-specific or system-wide. Both types are described below.

### Task-Specific Trap Handlers

This type of trap handler is included in a task's bound unit; it resides in a task group's memory pool. A task-specific trap handler receives a trap only if the task, in whose bound unit the handler is included, has done the following:

- Specified the trap number, by means of the Enable User Trap ($ENTRP) macro call.

- Connected the trap handler to the trap's vector by means of the Trap Handler Connect ($TRPHD) macro call.

The task-specific handler receives the TSA contents exactly as if it was directly connected to the trap vector; but, in fact, the monitor has intercepted the trap and simulated the TSA in user-accessible memory.

### System-Wide Trap Handlers

A system-wide trap handler is loaded into system memory at the time of configuration. It is directly attached to a specific trap vector by user code. When any executing task in the system signals that trap, the trap handler directly responds, bypassing the Monitor (which, for a task-specific trap handler, would intercept and analyze the trap). Thus, system overhead is reduced; however, the same trap handling routine services all tasks that incur a given trap condition.

## PASSING TRAPS

It is assumed that all vendor-supplied and possibly some user-written trap handlers attached to the vector may encounter situations which should be passed to the system default trap handler. Also, several handlers can process the same trap. To pass a trap from one handler attached to a trap vector to the next handler:

1. Load the trap handler by means of an LDBU directive, thus placing the handler in system memory. The system, at the time of configuration, implicitly loads the Scientific Simulator's trap handler into system memory if the SSIP or DSIP argument was specified in a SYS directive.

2. Write the handler to include initialization subroutine table (IST) code that will execute when the LDBU load operation occurs and save the current address contents of the trap vector(s) to be simulated, inserting its own pointer(s) instead.

3. Code the user-written simulator to save the contents of all registers upon entry so that if the trap should be passed to the next trap handler, this handler can:

   a. Restore all saved registers.

   b. Execute a jump-indirect through the location containing the pointer of the next handler saved in step 2 above. The J-bit in the M1 register must be off when the jump-indirect is executed.

The rule is that each trap handler must get exactly the same information in registers and TSA that it would have received if it was the first trap handler accessed.

## PROGRAMMING CONSIDERATIONS FOR USER-WRITTEN TRAP HANDLERS

Note the following programming considerations relative to user-written trap handlers:

- A trap handler operates at the same priority level and in the same privilege ring as the task whose execution caused the trap.

- When a trap occurs, the hardware/firmware saves the task related contents of the I-register, the R3 register, and the B3 register in the trap save area. The trap handler is free to use these registers.

- See Table A-1 for a description of the contents of selected words in the trap save area when various traps occur.

- Upon entry to the user trap handler, the J-bit in the M1 register is arbitrarily turned off. Other bits in the M1 register remain as they were when the trap occurred. Register B3 contains a pointer to the A-word in the TSA. Register R3 contains the vector number of the trap.

- Traps that occur within the user trap handler abort the task if they are the same type as the trap currently being processed. This abort action prevents all TSAs from being tied up by recursive traps, and prevents traps within the MCL interface from going to the user trap handler.

- Every trap handler should be reentrant; i.e., it should not use an internal work area to store interim information, since this information could be lost if an interrupt occurs and, later, the same trap handler is called upon to execute at a different priority level.

- If you choose to define instructions of your own and have them interpreted by a trap handler connected to trap vector 5, you should limit the instructions to the user-reserved subset of the generic instructions. The following diagram illustrates the memory format of generic instructions.

```
BIT:  0               7 8               15
     +-----------------+-----------------+
     | 0 0 0 0 0 0 0 0 |        f        |
     +-----------------+-----------------+
```

  f - Function; the user-reserved range of values for f is $128 \leq f < 256$ (decimal).

- When a trap handler has finished its work, it must issue an RTT (Return From Trap) instruction. The M1 register is not restored. This instruction uses the current trap save area to restore the task-related contents of the I-register, the R3 register, the program counter, and the B3 register. Consequently, when the RTT instruction is executed, these elements of the trap save area should be "correct" (i.e., as saved when the trap occurred).

  Note that in some cases, particularly when a trap condition is related to an input/output instruction, the saved value of the program counter (in the trap save area) will point to a memory location _within_ the instruction itself. This is _not_ a legitimate point of return to "normal" task processing. In this case, the trap handler must modify the saved value of the program counter before issuing an RTT instruction.

  After the trap save area has been used to restore the registers indicated above, it is returned to the pool of available trap save areas pointed to by a memory location 0010.

- When a trap occurs, the contents of registers M1 through M7 are not saved in the TSA. Particular attention is drawn to the R1 through R7 overflow trap enable bits and the J-bit of register M1, which can be set by a privileged user. If the trap handler does not temporarily clear these bits during its execution, another user trap handler could be invoked erroneously on data register overflow or branches. Such bits must be restored upon exit from the handler.

# *Appendix B* *PROGRAMMING CONVENTIONS*

The following programming conventions are provided for designing application programs to interface smoothly with system software.

## MODULE AND FILE NAME CONVENTIONS

Program names and load module names that begin with Z are reserved for Honeywell use and should not be used for an application program. System module names are six characters in length; the second character defines the system component. Table B-1 lists the first two characters of each system module name and the system component that it relates to.

The names of files that are processed by program development software (compiler, assembler, and so on), are given a suffix by the particular component doing the processing. Table B-2 lists these suffixes.

Table B-1. System Module Name-Prefixes

| Name Prefix | System Component |
|---|---|
| ZA | Assembler |
| ZC | COBOL Compiler |
| ZE | Editor |
| ZF | FORTRAN Compiler |
| ZG | Configuration Load Manager |
| ZH | Trap Handler |
| ZI | Input/Output Drivers |
| ZL | Linker |
| ZM | Memory Management |
| ZO | Loader |
| ZP | Macro-Assembly Program |
| ZQ | Communications |
| ZR | RPG Compiler |
| ZS | Sort/Merge |
| ZT | TCLF Compiler and Processor |
| ZU | Utility Routines and Conversion Aids |
| ZX | Executive |
| ZY | File, Data and Storage Management |
| ZZ | Program units internal to File, Data and Storage Management |
| Z1 | Advanced FORTRAN Compiler |

Table B-2. System Program File Name Suffixes

| Suffix | File Type |
|---|---|
| .A | Assembly language source unit |
| .AO | Default user-out if user-in is disk |
| .AS | ADA language source unit |
| .B | BASIC source program unit |
| .C | COBOL language or C language source unit |
| .DB | Multi-user Debugger work file |
| .EC | Execution command (EC) |
| .F | FORTRAN language source unit |
| .L | List unit |
| .M | Link maps |
| .O | Object unit |
| .P | Macro-Assembly Program source program unit |
| .PS | PASCAL source unit |
| .Q | RPG Compiler generated linker directive file |
| .QK | Multi-user Debugger quick file |
| .R | RPG language source unit |
| .T | TCLF source program unit |
| .U | Auto report source unit |

## CALLING SEQUENCE FOR EXTERNAL PROCEDURES

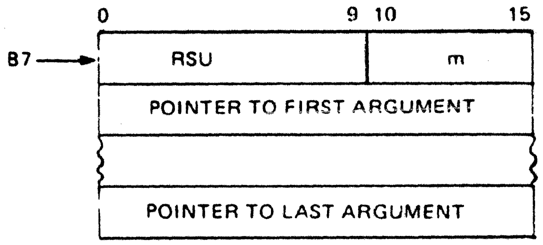
External procedures are those that are assembled or compiled separately from the calling procedure. These procedures may be either functions, that is, procedures returning a single value to the caller, or subroutines, namely, procedures that alter data contained in an area common to both the procedure and its caller. For example, the FORTRAN mathematical routines (sine, cosine, etc.) are external procedures. When it is necessary to write an Assembly language external procedure, use the calling sequence described below for compatibility with code generated by the language processors.

The external procedure calling sequence generated by the CALL statement in Assembly language, COBOL, BASIC, FORTRAN and RPG is of the form:

```
LAB $B7, list
LNJ, $B5.<entry
```

| | |
|---|---|
| list | Label assigned to the argument list |
| entry | External label of subroutine's entry point |

The external procedure should assume that register B5 contains the address of the caller's return point and register B7 points to an argument list having the format shown in Figure B-1.



RSU: Reserved for system use (must not be modified by called procedure)

m: Length of argument list given by 2n+1 where n is the number of arguments

Figure B-1. Argument List

## REGISTER CONVENTIONS

The system services use the following registers <u>without</u> preserving their contents: R1, R2, R6, R7, B2, and B4. If the information in these registers is of value to the application program, it should save the register contents before making a system control service request. Unless otherwise specified, the following registers will not be altered by the system services: S, I, R3, R4, R5, B1, B3, B5, B6, B7, T, RDBR, CI, SI, S1, S2, S3, and the M registers.

# *Appendix C*
# *DATA STRUCTURE FORMATS*

This appendix describes the following data structures:

- Clock request block (CRB)
- File information block (FIB)
- Input/output request block (IORB)
- Task request block (TRB)
- Parameter block
- Wait list
- Semaphore request block (SRB)
- Message group request blocks (MGCRB, MGIRB, MGRRB).

Any of the structures can be hand coded or generated by macro calls. All structures but the parameter block and wait list can be defined by macro call templates.

The first four items of the request blocks have an identical format (but slightly different contents, depending on the block type) as shown in Figure C-1. Later diagrams show the format of each block type; tables show the contents of the block entries.

The offset symbol $AF signifies that number of words required to specify a memory address. In this system, $AF is equivalent to two words.

The first field (-$AF or -1) of a request block need be present only when the request block pointer/semaphore name is needed.

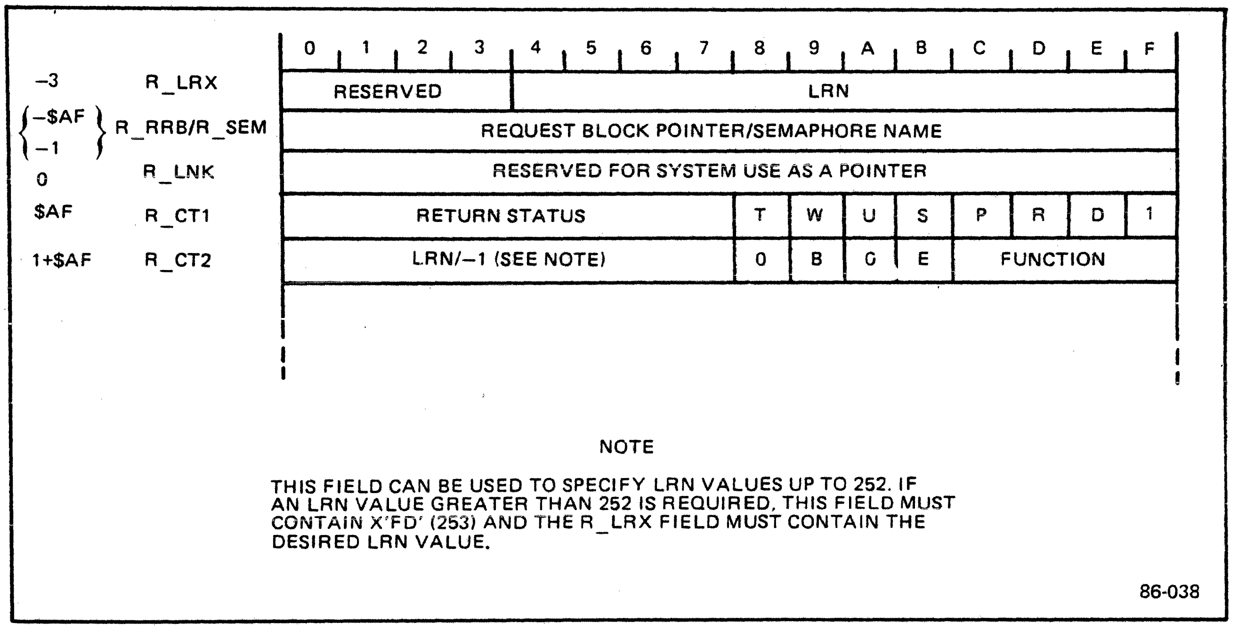

0 1 2 3 4 5 6 7 8 9 A B C D E F
-3 R_LRX
RESERVED
LRN
{-$AF, -1} R_RRB/R_SEM
REQUEST BLOCK POINTER/SEMAPHORE NAME
0 R_LNK
RESERVED FOR SYSTEM USE AS A POINTER
$AF R_CT1
RETURN STATUS
T W U S P R D 1
1+$AF R_CT2
LRN/-1 (SEE NOTE)
0 B G E
FUNCTION
NOTE
THIS FIELD CAN BE USED TO SPECIFY LRN VALUES UP TO 252. IF AN LRN VALUE GREATER THAN 252 IS REQUIRED, THIS FIELD MUST CONTAIN X'FD' (253) AND THE R_LRX FIELD MUST CONTAIN THE DESIRED LRN VALUE.
86-038

Figure C-1. First Four Items of Request Blocks

## CLOCK REQUEST BLOCK FORMAT

Figure C-2 shows the format of the clock request block; Table C-1 shows its contents.

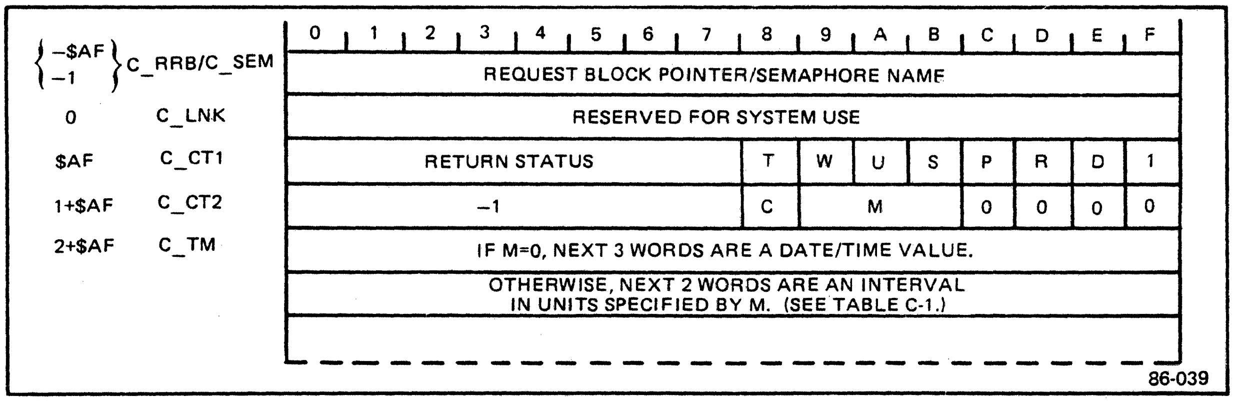

0 1 2 3 4 5 6 7 8 9 A B C D E F
{-$AF, -1} C_RRB/C_SEM
REQUEST BLOCK POINTER/SEMAPHORE NAME
0 C_LNK
RESERVED FOR SYSTEM USE
$AF C_CT1
RETURN STATUS
T W U S P R D 1
1+$AF C_CT2
-1
C M 0 0 0 0
2+$AF C_TM
IF M=0, NEXT 3 WORDS ARE A DATE/TIME VALUE.
OTHERWISE, NEXT 2 WORDS ARE AN INTERVAL IN UNITS SPECIFIED BY M. (SEE TABLE C-1.)
86-039

Figure C-2. Format of Clock Request Block

Table C-1. Contents of Clock Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| -$AF<br>-1 | C_RRB/<br>C_SEM | 0-31<br>0-15 | Depending on the S- or R-bits of C_CTl, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). |
| 0 | C_LNK | 0-15 | Reserved for system use. |
| $AF | C_CTl | 0-7 | Return status. |
| | | 8(T) | This bit is set on while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it. |
| | | 9(W) | Wait bit. Set if the requesting task is not to be suspended pending the completion of the request that uses this block. |
| | | A(U) | User bit. User may or may not use this bit; the system does not change it. In a user-built CRB, must be 0 initially. |
| | | B(S) | Release semaphore indicator.<br><br>0 = No release; 1 = Release, on completion of this request, semaphore named in C_SEM. |
| | | C(P) | Must be set by user if CRB is to be referenced by a Wait Any ($WAITA) macro call. If set, CRB can be referenced only by $WAIT or $WAITA issued by requesting task. |
| | | D(R) | Return clock RB indicator.<br><br>0 = No dispatch; 1 = Dispatch task request block named in C_RRB after completion of this request. |
| | | E(D) | Delete clock RB indicator, used usually with the B(S) and D(R) bits.<br><br>0 = No delete; 1 = Delete and, when task terminates, return memory to pool where CRB is first entry of its memory block. |
| | | F | Implicit task start address. Must always be 1 for CRB. |

Table C-1 (cont). Contents of Clock Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 1+$AF | C_CT2 | 0-7 | Value is -1. |
| | | 8(C) | When set, indicates this block is associated with a cyclic clock function. |
| | | 9-B(M) | When set, last two words contain an interval in units specified by M. Each interval value is as follows: 001 - in milliseconds; 010 - in tenths of a second; 011 - in seconds; 100 - in minutes; 101 - in units of clock resolution.<br><br>When reset (off), the last <u>three</u> words contain a date/time interval. |
| 2+$AF | C_TM | | Contents depend on M bit of C_CT2. |

## FILE INFORMATION BLOCK (FIB) FORMAT AND CONTENTS

Tables C-2 and C-3 show the format, and Tables C-4 and C-5 show the contents, of the file information block (FIB) for data management (record level) access, and for storage management (block level) access, respectively.

Table C-2. Format of FIB for Data Management

| Word | Label(s) | 0 1 2 3 4 5 6 7 | 8 9 A B C D E F |
|---|---|---|---|
| 0 | F_LFN | Logical file number (LFN) | |
| 1 | F_PROV | Program view | |
| 2<br>3 | F_URP | User record area pointer | |
| 4 | F_IRL | Input record length | |
| 5 | F_ORL | Output record length | |
| 6 | F_IRS/F_ORS | Input record status | Output record status |
| 7 | F_IRT | Input record type | |
| 8 | F_ORT | Output record type | |

Table C-2 (cont). Format of FIB for Data Management

| Word | Label(s) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 9<br>10 | F_IKP | Input key pointer | | | | | | | | | | | | | | | |
| 11 | F_IKF/F_IKL | Input key format | | | | | | | | Input key length | | | | | | | |
| 12<br>13 | F_ORA | Output record address | | | | | | | | | | | | | | | |
| 14<br>15 | F_RFU2 | Reserved | | | | | | | | | | | | | | | |

Table C-3. Format of FIB for Storage Management

| Word | Label(s) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | F_LFN | Logical file number (LFN) | | | | | | | | | | | | | | | |
| 1 | F_PROV | Program view | | | | | | | | | | | | | | | |
| 2<br>3 | F_UBP | User buffer pointer | | | | | | | | | | | | | | | |
| 4 | F_BFSZ | Buffer size | | | | | | | | | | | | | | | |
| 5 | F_BKSZ | Block size | | | | | | | | | | | | | | | |
| 6<br>7 | F_BKN1<br>F_BKN2 | Block number | | | | | | | | | | | | | | | |
| 8<br>9<br>10<br>11<br>12<br>13<br>14<br>15 | F_RFU3 | Reserved | | | | | | | | | | | | | | | |

Table C-4. Contents of FIB for Data Management

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 0 | F_LFN | 0-15 | Logical file number (LFN) |
| 1 | F_PROV | 0 | Access level. Set off for data management. |
| | | 1-4 | Process rules. Bit 1 for $RDREC, bit 2 for $WRREC, bit 3 for $RWREC, bit 4 for $DLREC. |
| | | 5-9 | Key type. Bit 5 for primary keys, bit 8 for relative keys, bit 9 for simple keys (bits 6 and 7 must be 00). |
| | | 10 | Record class. Set on for fixed-length records only; off for fixed- and variable-length records. |
| | | 11 | Record visibility. Set on if deleted records are to be visible; off if invisible. |
| | | 12 | Key storage alignment. Set on if storage area begins at odd-byte boundary; off if even-byte boundary. |
| | | 13 | Record storage area. Set on if record storage area begins on odd-byte boundary; off if even-byte boundary. |
| | | 14 | Transcription mode. Set on if data transferred in binary transcription mode; off if ASCII mode. |
| | | 15 | Must be 0. |
| 2,3 | F_URP | 0-31 | Start address of user record area. |
| 4 | F_IRL | 0-15 | Input record length (in bytes). |
| 5 | F_ORL | 0-15 | Output record length (in bytes). |
| 6 | F_IRS | 0-3 | 0000 - Unknown terminal control information; 0001 - Records contain no terminal control information; 0010 - Records contain standard GCOS 6 printer control characters. |
| | | 4-7 | Must be zero. |

Table C-4 (cont). Contents of FIB for Data Management

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| | F_ORS | 8 | Read operations. Set on if the key of the record just read duplicates the key of the record previously read.<br><br>Write/rewrite operations. Set on if key of the record just written is a duplicate. |
| | | 9 | Read operations. Set on if key of the record just read duplicates a record that is yet to be read. |
| | | 10-15 | Must be zero. |
| 7 | F_IRT | 0-15 | Must be set to X'FFFF' (all bits set on). |
| 8 | F_ORT | 0-15 | Must be set to X'0000' (all bits set off). |
| 9,10 | F_IKP | 0-31 | Start address of user key area. |
| 11 | F_IKF | 0-7 | Input key format. 0 for none specified; 1 for primary key; 2 for simple key. |
| | F_IKL | 8-15 | Input key length (in bytes). |
| 12,13 | F_ORA | 0-31 | Output record address. |
| 14,15 | F_RFU2 | 0-31 | Reserved for later use; must be X'00000000'. |

Table C-5. Contents of FIB for Storage Management

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 0 | F_LFN | 0-15 | Logical file number (LFN). |
| 1 | F_PROV | 0 | Access level. Set on for storage management. |
| | | 1-2 | Process rules. Bit 1 for $RDBLK; bit 2 for $WRBLK. |
| | | 4-12 | Must be X'00000000'. |
| | | 13 | Buffer alignment. Set on when buffer begins on odd-byte boundary; off when even-byte boundary. |

Table C-5 (cont). Contents of FIB for Storage Management

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 1 (cont) | F_PROV (cont) | 14 | Transcription mode. Set on when data transferred in binary transcription mode; off when transfer is in ASCII mode. |
| | | 15 | Synchronous/asynchronous indicator. Set on when $RDBLK and $WRBLK calls executed asynchronously; off when synchronously. |
| 2,3 | F_UBP | 0-31 | Start address of user buffer area. |
| 4 | F_BFSZ | 0-15 | Buffer transfer size (in bytes). |
| 5 | F_BKSZ | 0-15 | Block size (in bytes). |
| 6,7 | F_BKNO | 0-31 | Block number. |
| 8-15 | F_RFU3 | All | Reserved for later use; must be all zeros. |

## INPUT/OUTPUT REQUEST BLOCK (IORB) FORMAT

Figure C-3 shows the format of a nonextended input/output request block (IORB) (see Section 4 for a description of IORB extensions). Table C-6 defines the specific fields for a non-extended IORB. Table C-7 summarizes the IORB fields for operator interface functions.

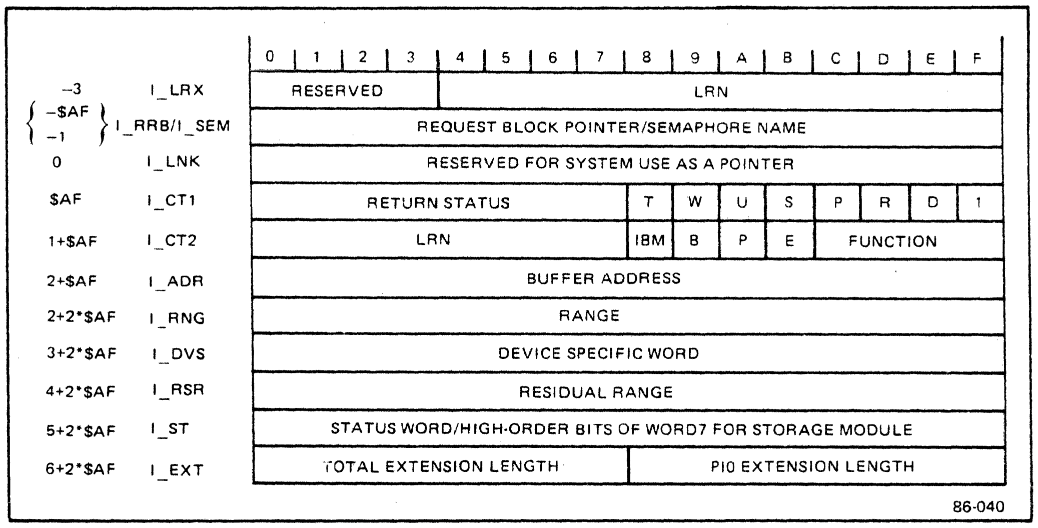

0 1 2 3 4 5 6 7 8 9 A B C D E F
-3 I_LRX RESERVED LRN
-$AF -1 I_RRB/I_SEM REQUEST BLOCK POINTER/SEMAPHORE NAME
0 I_LNK RESERVED FOR SYSTEM USE AS A POINTER
$AF I_CT1 RETURN STATUS T W U S P R D 1
1+$AF I_CT2 LRN IBM B P E FUNCTION
2+$AF I_ADR BUFFER ADDRESS
2+2*$AF I_RNG RANGE
3+2*$AF I_DVS DEVICE SPECIFIC WORD
4+2*$AF I_RSR RESIDUAL RANGE
5+2*$AF I_ST STATUS WORD/HIGH-ORDER BITS OF WORD7 FOR STORAGE MODULE
6+2*$AF I_EXT TOTAL EXTENSION LENGTH PIO EXTENSION LENGTH
86-040

Figure C-3. Format of I/O Request Block

Table C-6. Contents of I/O Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| -3 | I_LRX | 0-3 | Reserved for system use. |
| | | 4-15 | Extended logical resource number (LRN). If byte 0 (bits 0 to 7) of I_CT2 contains the value 253 (x'FD'), this field indentifies the device to be used. |
| -$AF<br>-1 | I_RRB/<br>I_SEM | 0-31<br>0-15 | Depending on the S- or R-bits of I_CT1, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). Set by user; used by system at termination of request. |
| 0 | I_LNK | 0-31 | Reserved for system use. 2-word pointer to indirect request block. |
| $AF | I_CT1 | 0-7 | Return status |
| | | 8(T) | This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it. |
| | | 9(W) | Wait bit. Set by user if requesting task is not to be suspended pending completion of the request that uses this IORB. For a $OPMSG call, the setting of the W- bit in output IORB controls return to the caller. For a $OPRSP call, setting of W-bit in <u>input</u> IORB controls return to the caller; setting of W-bit in output IORB has no significance. For either call, return to caller is immediate if significant W-bit is on. If significant W-bit is off, return to caller occurs after the order is completed. |
| | | A(U) | User bit. User may or may not use this bit; the system does not change it. |
| | | B(S) | Release semaphore indicator.<br><br>0 = No release; 1 = Release, on completion, semaphore item named in I_SEM. |

Table C-6 (cont). Contents of I/O Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| $AF (cont) | I_CT1 (cont) | C(P) | Must be set by user if IORB is to be referenced by a Wait Any ($WAITA) macro call. If set, IORB can be referenced only by $WAIT or $WAITA issued by the requesting task. |
| | | D(R) | Return IORB indicator.<br><br>0 = No dispatch; 1 = Dispatch task request block named in I_RRB after completion of this request. If 1, system executes $RQTSK, using I_RRB, when the task terminates. |
| | | E(D) | Delete IORB indicator. Used usually with the B(S) and D(R) bits.<br><br>0 = No delete; 1 = Delete and when task terminates, return memory to pool where IORB is first entry of its memory block. |
| | | F(1) | Implicit task start address. Must always be 1 for IORB. |
| 1+$AF | I_CT2 | 0-7 | Logical resource number (LRN). If this field contains any value other than 253 (x'FD'), it indentifies the device to be used. If this field contains 253, I_LRX contains the LRN value. |
| | | 8(IBM) | IBM-type request. Changes interpretation of I_DVS to task word, and of I_RSR and I_ST to configuration words A and B, respectively. |
| | | 9(B) | Byte index. 0 = buffer begins in leftmost byte of word; 1 = buffer begins in rightmost byte. Must be off if input/output buffer begins at left byte of word whose address is contained in word 3 (I_ADR) of IORB. Must be on if input/output buffer begins at the right byte. |
| | | A(P) | Private space; reserved for system use. |
| | | B(E) | Extended IORB indicator. 0 = Standard (nonextended) IORB; 1 = IORB extended to at least 6+2*$AF items. Set by user. (See I_EXT below.) |

Table C-6 (cont). Contents of I/O Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 1+$AF (cont) | I_CT2 (cont) | C-F | Function code. Driver or LPH function, see Table 6-1. |
| 2+$AF | I_ADR | 0-31 | Buffer address. 2-word pointer. Word address of message buffer (which contains an output message or is to receive an input message). |
| 2+2*$AF | I_RNG | 0-15 | Range. Number of bytes to be transferred. Used as input field for cartridge disk or mass storage unit. Buffer size in bytes. This is the length of an output message or the maximum length allowed for an input message. |
| 3+2*$AF | I_DVS | 0-15 | Device-specific information. |
| 4+2*$AF | I_RSR | 0-15 | Residual range. Indicates the number of bytes <u>not</u> transferred. Filled in by the system on completion of the order. Used by the cartridge disk and mass storage unit drivers as a data offset value. |
| 5+2*$AF | I_ST | 0-15 | Modified device status. Shows mapping of hardware status into software status format. See Table 6-4. Set by user as input field high-order bits of sector number of mass storage unit. Set by system after I/O completion. |
| 6+2*$AF | I_EXT | 0-7 | Left byte. Number of words, in binary, in the IORB extension, not including this I_EXT word. |
| | | 8-15 | Right byte. Number of words, in binary, in physical I/O part of IORB extension, not including this I_EXT word. This count must be less than or equal to the total extension length specified in the left byte (0-7). This word is present only when the B(E) bit in I_CT2 is 1. (See Section 7 for a description of IORB extensions.) |

Table C-7. Summary of IORB Fields for Operator Interface

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| $AF | I_CT1 | 9(W) | For a $OPMSG call, the setting of W-bit in the output IORB controls return to the caller. For a $OPRSP call, the setting of W-bit in the input IORB controls return to the caller; the setting of W-bit in the output IORB has no significance. For either call, return to caller is immediate if significant W-bit is on. If significant W-bit is off, return to caller occurs after the order is completed. |
| 1+$AF | I_CT2 | 0-7 | LRN = 0. |
|  |  | 9(B) | Must be off if input/output buffer begins at the left byte of the word whose address is contained in word 3 (I_ADR) of this IORB. Must be on if the input/output buffer begins at the right byte. |
| 2+$AF | I_ADR | 0-15 | The word address of the message buffer (which contains an output message or is to receive an input message). |
| 2+2*$AF | I_RNG | 0-15 | The buffer size in bytes. This is the length of an output message or the maximum length allowed for an input message. |

SEMAPHORE REQUEST BLOCK FORMAT

Figure C-4 shows the format of the semaphore request block; Table C-8 shows its content.

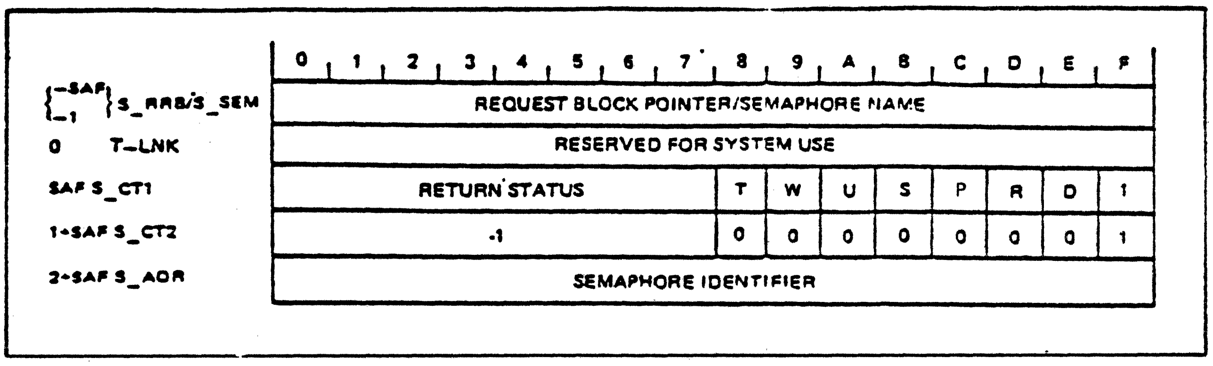


Figure C-4. Format of Semaphore Request Block

Table C-8. Contents of Semaphore Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| -$AF<br>-1 | S_RRB/<br>S_SEM | 0-31<br>0-15 | Depending on the S- or R-bits of S_CT1, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). Set by user; used by system when request terminates. |
| 0 | S_LNK | 0-15 | Reserved for system use. |
| $AF | S_CT1 | 0-7 | Return status. |
| | | 8(T) | This bit is set (on) while the request using the block is executing; it is reset when the request terminates. The system controls this bit; user should not change it. |
| | | 9(W) | Wait bit. Set if requesting task is <u>not</u> to be suspended pending the completion of the request that uses this block. |
| | | A(U) | User bit. User may or may not use this bit; the system does not change it. |
| $AF | S_CT1 | B(S) | Release semaphore indicator. 0 = No release; 1 = Release, on completion, semaphore item named in S_SEM. |
| | | C(P) | Must be set by user if SRB is to be referenced by a Wait Any ($WAITA) macro call. If set, SRB can be referenced only by $WAIT or $WAITA issued by the requesting task. |
| | | D(R) | Return semaphore RB indicator. 0 = No dispatch; 1 = Dispatch task request block named in S_RRB after completion of this request. |
| | | E(D) | Delete SRB indicator. Used usually with the B(S) and D(R) bits. 0 = No delete; 1 = Delete and, when task terminates, return memory to pool where SRB is first entry of its memory block. |
| | | F(1) | Implicit task start address. Must always be 1 for SRB. |

Table C-8 (cont). Contents of Semaphore Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 1+$AF | S_CT2 | 0-7<br>8-14<br>15 | Value is -1.<br>Must be zero.<br>Must be one. |
| 2+$AF | S_ADR | 0-15 | Semaphore identifier - two ASCII characters. |

## TASK REQUEST BLOCK FORMAT

Figure C-5 shows the format of the task request block; Table C-9 shows its contents.

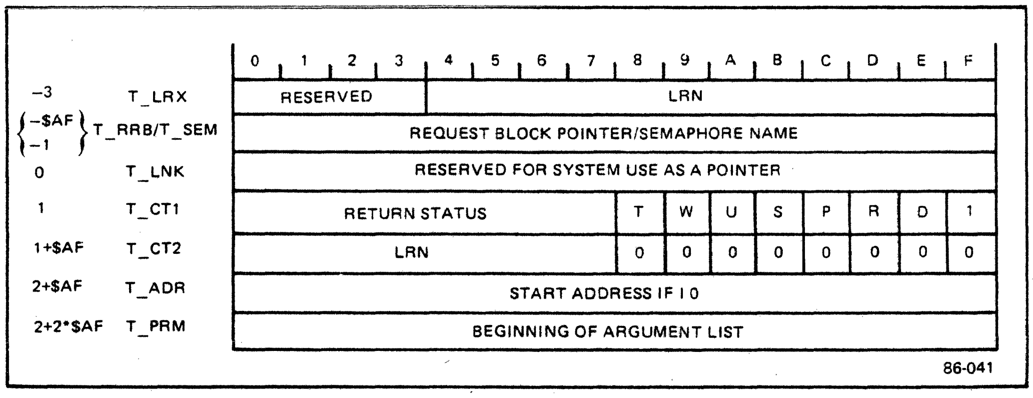


Figure C-5. Format of Task Request Block

Table C-9. Contents of Task Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| -3 | T_LRX | 0-3 | Reserved for system use. |
| | | 4-15 | Extended logical resource number (LRN). If byte 0 (bits 0 to 7) of I_CT2 contains the value 253 (x'FD'), this field indentifies the device to be used. |
| -$AF<br>-1 | T_RRB/<br>T_SEM | 0-31<br>0-15 | Depending on the S- or R-bits of T_CT1, this field contains a 2-word task request block pointer (R-bit on), or a 1-word semaphore name (S-bit on). Set by user, used by system when request terminates. |

Table C-9 (cont). Contents of Task Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 0 | T_LNK | 0-31 | Reserved for system use. |
| $AF | T_CT1 | 0-7 | Return status. |
| | | 8(T) | This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; the user should not change it. |
| | | 9(W) | Wait bit. Set by user if requesting task is <u>not</u> to be suspended pending completion of the request that uses this block. |
| | | A(U) | User bit. User may or may not use this bit; the system does not change it. |
| | | B(S) | Release semaphore indicator. 0 = No release; 1 = Release, on comple- tion, semaphore item named in T_SEM. |
| | | C(P) | Must be set by user if TRB is to be referenced by a Wait Any ($WAITA) macro call. If set, TRB can be referenced only by $WAIT or $WAITA issued by the requesting task. |
| | | D(R) | Return task RB indicator. 0 = No dis- patch; 1 = Dispatch task request block named in T_RRB after com- pletion of this request. |
| | | E(D) | Delete TRB indicator. Used usually with the B(S) and D(R) bits. 0 = No delete; 1 = Delete and when task terminates, return memory to pool where TRB is first entry of its memory block. |
| | | F(1) | Implicit task start address. Must always be 1 for TRB. |
| 1+$AF | T_CT2 | 0-7 | Logical resource number (LRN). If this field contains any value other than 253 (x'FD'), it indentifies the device to be used. If this field contains 253, I_LRX contains the LRN value. |
| | | 8-15 | Must be zero. |

Table C-9 (cont). Contents of Task Request Block

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 2+$AF | T_ADR | 0-15 | Start address if the I-bit of T_CT1 is reset (zero). |
| 2+2*$AF | T_PRM | | Beginning of argument list. |

## PARAMETER BLOCK FORMAT

Figure C-6 shows the format of the parameter block.

NOTE

The parameter value strings need not be contiguous with the address portion of the parameter block; if the block is system-generated, each parameter will have a trailing blank that is not included in the byte count.

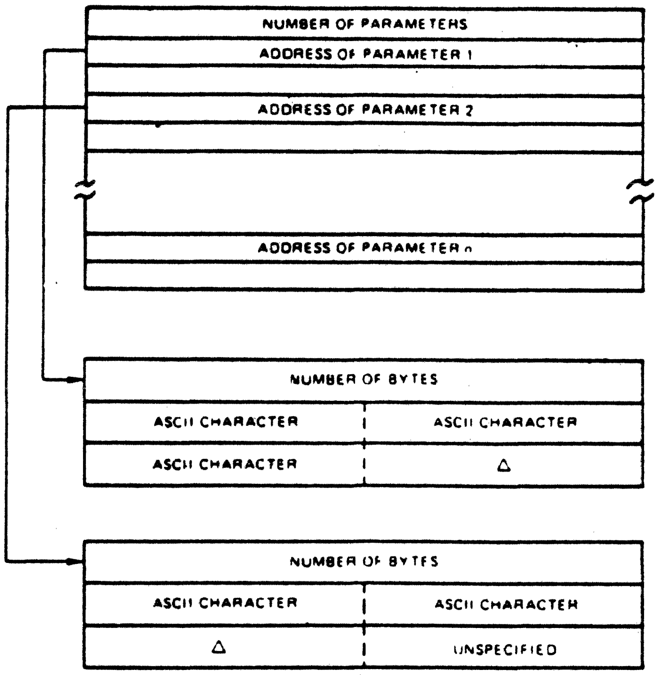


Figure C-6. Format of Parameter Block

## WAIT LIST FORMAT

Figure C-7 shows the format of the wait list.

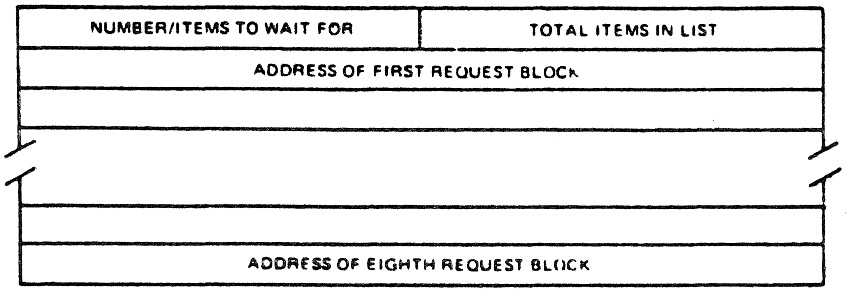

NUMBER/ITEMS TO WAIT FOR
TOTAL ITEMS IN LIST
ADDRESS OF FIRST REQUEST BLOCK
ADDRESS OF EIGHTH REQUEST BLOCK

Figure C-7. Format of Wait List

## MESSAGE GROUP REQUEST BLOCKS

Tables C-10, C-11, and C-12, respectively, show the content of the following message group request blocks:

- Message group control request block (MGCRB)
- Message group initialization request block (MGIRB)
- Message group recovery request block (MGRRB).

Templates for these request blocks are generated by the $MGCRT, $MGIRT, and $MGRRT macro calls, respectively.

The request blocks can be generated by the $MGCRB, $MGIRB, and $MGRRB macro calls, respectively.

Message group request blocks are used by the message facility for sending requests between task groups or tasks.

Table C-10. Message Group Control Request Block (MGCRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 0 | MC_OS | 0-31 | Pointer; reserved for system use. |
| $AF | MC_MAJ | | Major status. |
| | | 0-7 | Reserved for system use. |
| | | 8(T) | This bit is set (on) while the request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it. |
| | | 9(W) | Wait bit. Set if requesting task is not to be suspended pending the completion of the request that uses this block. |

Table C-10 (cont). Message Group Control Request Block (MGCRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| $AF (cont) | MC_MAJ (cont) | A(U) | User bit. User may use this bit; the system does not change it. Display processing uses this bit during a write. |
| | | B(S) | Release semaphore indicator. Values: 0 = No release; 1 = Release (on closeout) of semaphore, which must be in MC_OS -1. |
| | | C(P) | Must be set by user if MGCRB is to be referenced by a Wait Any ($WAITA) macro call. If set, MGCRB can be referenced only by $WAIT or $WAITA issued by the requesting task. |
| | | D(R) | Return request block indicator. Values: 0 = No dispatch; 1 = Dispatch request block whose address must be contained in MC_OS -$AF, after closeout of request. |
| | | E(D) | Delete request block. Values: 0 = No delete; 1 = Delete, and return memory to the pool where MGCRB is the first entry of its memory block. |
| | | F(1) | I/O bit. Must be set. |
| 1+$AF | MC_OPT | | General options: |
| | | 0-7 | Reserved for system use. |
| | | 8 | Must be 0. |
| | | 9 | Byte index. 0 = Buffer begins in leftmost byte of the word; 1 = Buffer begins in rightmost byte. |
| | | A | Must be 0. |
| | | B | Must be 1 (extended MGCRB). |
| | | C-F | Must be 0. |
| 2+$AF | MC_BUF | 0-31 | Buffer pointer. |
| 2+2*$AF | MC_BSZ | 0-F | Buffer range (in bytes). |
| 3+2*$AF | MC_DVS | | Record-type code. |
| | MC_REC | 0-F | On send, insert record-type code; on receive, return assigned record-type code. |

Table C-10 (cont). Message Group Control Request Block (MGCRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 4+2*$AF | MC_RSR | 0-F | Residual range (in bytes). |
| 5+2*$AF | MC_MRU | 0-7 | End message recovery unit (MRU). Reserved for system use. |
|  | MC_WTI | 8-F | Wait test indicator. Values: 00 = Return null to application; 01 = Wait. |
| 6+2*$AF | MC_EXT |  | Extension mechanism. |
|  |  | 0-7 | Binary value of 13+2*$AF, i.e., number of words in MGCRB following the extension word. |
|  |  | 8-F | Must be hexadecimal '7'. |
| 7+2*$AF | Next 7 words |  | Reserved for system physical I/O use. |
| 14+2*$AF | MC_FNC | 0-7 | Function. Reserved for system use. |
|  | MC_REV | 8-F | Revision. Must be hexadecimal '2'. |
| 15+2*$AF | MC_MGI | 0-F | Message group id. Returned in the $MINIT and $MACPT macro calls. |
| 16+2*$AF | MC_LVL |  | Enclosure level. |
|  | MC_LVR | 0-7 | Enclosure level requested. |
|  | MC_LVD | 8-F | Enclosure level detected according to following ASCII values: 0 = Not end of record; 1 = End of record; 2 = End of quarantine unit; 5 = End of message. |
| 17+2*$AF | MC_PCI | 0-F | Must be 0. |
| 18+2*$AF | MC_VDP | 0-31 | Must be zero. |
| 18+3*$AF | MC_TGI | 0-F | Reserved for system use. |
| 19+3*$AF | MC_TSK | 0-31 | Pointer. Reserved for system use. |
| 19+4*$AF | MC_NPI | 0-F | Must be 0. |
| 22+3*$AF | MC_LEN | 0-F | Length of text received. |

Table C-11. Message Group Initialization Request Block (MGIRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 0 | MI_OS | 0-31 | Pointer. Reserved for system use. |
| $AF | MI_MAJ | | Major status. |
| | | 0-7 | Reserved for system use. |
| | | 8(T) | This bit is set (on) while the request using this block is executing; it is reset when request terminates. The system controls this bit; user should not change it. |
| | | 9(W) | Wait bit. Set if the requesting task is not suspended pending the completion of the request that uses this block. |
| | | A(U) | User bit. User may or may not use this bit; the system does not change it. |
| | | B(S) | Release semaphore indicator. Values: 0 = No release; 1 = Release, on termination of this request, semaphore whose name must be in MI_OS -1. |
| | | C(P) | Must be set by user if MGIRB is to be referenced by a Wait Any ($WAITA) macro call. If set, MGIRB can be referenced only by $WAIT or $WAITA issued by the requesting task. |
| | | D(R) | Return request block indicator. Values: 0 = No dispatch. 1 = Dispatch, after end of this request, request block whose address must be contained in MI_OS -$AF. |
| | | E(D) | Delete I/O request block. Values: 0 = No delete; 1 = Delete, and return memory to the pool where this MGIRB is the first entry of its memory block. |
| | | F(1) | I/O bit. Must be set. |
| 1+$AF | MI_OPT | | General options: |
| | | 0-7 | Reserved for system use. |
| | | 8-A | Must be 0. |
| | | B | Must be 1 (extended MGIRB). |
| | | C-F | Must be 0. |
| 2+$AF | MI_BUF | 0-31 | Must be zero. |

Table C-11 (cont). Message Group Initialization Request Block (MGIRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 2+2*$AF | MI_BSZ | 0-F | Buffer range in bytes. Must be 0. |
| 3+2*$AF | MI_MPD | 0-F | Message path description identifier. Must be ASCII 01. |
| 4+2*$AF | MI_RSR | 0-F | Residual range in bytes. |
| 5+2*$AF | MI_MDE | 0-7 | Must be 0. |
|  | MI_IOP | 8-F | Must be 0. |
| 6+2*$AF | MI_EXT |  | Extension mechanism: |
|  |  | 0-7 | Binary value of 31+2*$AF, i.e., number of words in MGIRB following the extension word. |
|  |  | 8-F | Must be hexadecimal 7. |
| 7+2*$AF | MI_DV2 (three words) | 0-F<br>0-F<br>0-F | Maturity date/time in standard internal date/time format (see $INDTM). |
| 14+2*$AF | MI_FNC | 0-7 | Function. Reserved for system. |
|  | MI_REV | 8-F | Revision. Must be hexadecimal 2. |
| 15+2*$AF | MI_MGI | 0-F | Message group id. Returned in the $MINIT and $MACPT macro calls. |
| 16+2*$AF | MI_PCM (Two words) | 0-F<br>0-F | Must be 0.<br>Must be 0. |
| 18+2*$AF | MI_ADT |  | Address type: |
|  |  | 0-7 | Address type (initiator); must be ASCII 1. |
|  |  | 8-F | Address type (acceptor); must be ASCII 1. |
| 19+2*$AF | MI_NWI | 0-F | Must be 0. |
| 20+2*$AF | MI_NDI | 0-F | Must be 0. |
| 21+2*$AF | MI_MBI (Six words) | 0-F<br>0-F<br>0-F<br>0-F<br>0-F<br>0-F | Initiator mailbox name. Must be from 1 to 12 ASCII characters, blank-filled, left-justified as specified when the mailbox was created, indicating only messages with this identifier will be accepted; or all zeros, indicating messages with any identifier will be accepted. |

Table C-11 (cont). Message Group Initialization Request Block (MGIRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 27+2*$AF | MI_NWA | 0-F | Must be 0. |
| 28+2*$AF | MI_NDA | 0-F | Must be 0. |
| 29+2*$AF | MI_MBA (Six words) | 0-F 0-F 0-F 0-F 0-F 0-F | Acceptor mailbox name. Must be from 1 to 12 ASCII characters specifying the acceptor mailbox id, blank-filled, left-justified. |
| 36+2*$AF | MI_CNT | 0-F | Count of number of active messages in the mailbox. Returned with $MCMG macro call. |
| 37+2*$AF | MI_TGI | 0-F | Reserved for system. |
| 38+2*$AF | MI_TSK | 0-31 | Pointer. Reserved for system use. |
| 38+3*$AF | MI_SIP | 0-31 | Security information pointer. Points to the security information block (SIB) that points to the logical submittor block containing the user id (SI_PER), the account id (SI_ACC), and the mode (SI_MOD). |

Table C-12. Message Group Recovery Request Block (MGRRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 0 | MR_OS | 0-31 | Pointer. Reserved for system. |
| $AF | MR_MAJ | | Major status. |
| | | 0-7 | Reserved for system. |
| | | 8(T) | This bit is set (on) while request using this block is executing; it is reset when the request terminates. The system controls this bit; user should not change it. |
| | | 9(W) | Wait bit. Set if the requesting task is not to be suspended pending the completion of the request that uses this block. |

Table C-12 (cont). Message Group Recovery Request Block (MGRRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| $AF (cont) | MR_MAJ (cont) | A(U) | User bit. User may or may not use this bit; the system does not change it. |
| | | B(S) | Release semaphore indicator. Values: 0 = No release; 1 = Release, on closeout, of semaphore which must be in MC_OS -1. |
| | | C(P) | Must be set by user if MGRRB is to be referenced by a Wait Any ($WAITA) macro call. If set, MGRRB can be referenced only by $WAIT or $WAITA issued by the requesting task. |
| | | D(R) | Return request block indicator. Values: 0 = No dispatch; 1 = Dispatch request block, whose address must be in MC_OS -$AF, after closeout of this request. |
| | | E(D) | Delete I/O request block. Values: 0 = No delete; 1 = Delete, and return memory to the pool where MGRRB is the first entry of its memory block. |
| | | F(1) | I/O bit. Must be set. |
| 1+$AF | MR_OPT | | General options: |
| | | 0-7 | Reserved for system use. |
| | | 8-A | Must be 0. |
| | | B | Must be 1 (extended MGRRB). |
| | | C-F | Must be 0. |
| 2+$AF | MR_BUF | 0-31 | Pointer. Must be 0. |
| 2+2*$AF | MR_BSZ | 0-F | Buffer range. Must be 0. |
| 3+2*$AF | MR_ITP | 0-F | Must be 0. |
| 4+2*$AF | MR_RES | 0-F | Residual range. Reserved for system. |

Table C-12 (cont). Message Group Recovery Request Block (MGRRB)

| Word | Label | Bit(s) | Contents |
|---|---|---|---|
| 5+2*$AF | MR_RSN | 0-7 | Reason-for-terminate code. 0 = Normal message group termination; 22-26 = User-defined abnormal termination of message group. |
| | | 8-F | Reserved for system. |
| 6+2*4AF | MR_EXT | 0-7 | Binary value of 24+2*$AF, i.e., number of words in MGRRB following the extension word. |
| | | 8-F | Must be hexadecimal '7'. |
| 14+2*$AF | MR_FNC | 0-7 | Function. Reserved for system. |
| | MR_REV | 8-F | Revision. Must be hexadecimal 02. |
| 15+2*$AF | MR_MGI | 0-F | Message group id. Returned in the $MINIT and $MACPT macro calls. |
| 17+2*$AF | MR_CNC | 0-F | Reserved for system use. |
| 16+2*$AF | MR_FMT | 0-31 | Pointer. Must be 0. |
| 18+3*$AF | MR_MRU (Two words) | 0-F<br>0-F | Reserved for system use.<br>Reserved for system use. |
| 19+3*$AF | MR_AMU (Two words) | 0-F<br>0-F | Reserved for system use.<br>Reserved for system use. |

# *Appendix D*
# *ASCII AND EBCDIC CHARACTER SETS*

Tables D-1 and D-2 illustrate the ASCII and EBCDIC character sets, respectively. In addition to the characters, both tables show the binary and hexadecimal equivalents of the character set.

The following is a list of the control characters that appear in the two tables:

| | | | |
|---|---|---|---|
| ACK | Acknowledge | GE | Graphic Escape |
| BEL | Bell | GS | Group Separator |
| BS | Backspace | HT | Horizontal Tab |
| BYP | Bypass | IFS | Interchange File Separator |
| CAN | Cancel | IGS | Interchange Group Separator |
| CC | Cursor Control | IL | Idle |
| CR | Carriage Return | IRS | Interchange Record Separator |
| CU1 | Customer Use 1 | IUS | Interchange Unit Separator |
| CU2 | Customer Use 2 | LC | Lowercase |
| CU3 | Customer Use 3 | LF | Line Feed |
| DC1 | Device Control 1 | NAK | Negative Acknowledgement |
| DC2 | Device Control 2 | NL | New Line |
| DC3 | Device Control 3 | NUL | Null |
| DC4 | Device Control 4 | PF | Punch Off |
| DEL | Delete | PN | Punch On |
| DLE | Data Link Escape | RES | Restore |
| DS | Digit Select | RLF | Reverse Line Feed |
| EM | End of Medium | RS | Reader Stop |
| ENQ | Enquiry | SI | Shift In |
| EO | Eight Ones | SM | Set Mode |
| EOT | End of Transmission | SMM | Start of Manual Message |
| ESC | Escape | SO | Shift Out |
| ETB | End of Transmission Block | SOH | Start of Heading |
| ETX | End of Text | SOS | Start of Significance |
| FF | Form Feed | SP | Space |
| FS | Field Separator | STX | Start of Text |
| SUB | Substitute | UC | Uppercase |
| SYN | Synchronous Idle | US | Unit Separator |
| TM | Tape Mark | VT | Vertical Tab |

The graphic characters in the 8-bit ASCII character set are defined as follows:

| | |
|---|---|
| SP | Space |
| ! | Exclamation Mark |
| " | Quotation Mark |
| # | Number Sign |
| $ | Dollar Sign |
| % | Percent Sign |
| & | Ampersand |
| ' | Apostrophe |
| ( | Left Parenthesis |
| ) | Right Parenthesis |
| * | Asterisk |
| + | Plus Sign |
| , | Comma |
| - | Minus Sign, Hyphen |
| . | Period, Decimal Point |
| / | Solidus, Slash |
| : | Colon |
| ; | Semicolon |
| < | Less-than Sign |
| = | Equals Sign |
| > | Greater-than Sign |
| ? | Question Mark |
| @ | Commercial At Sign |
| [ | Left Square Bracket |
| \ | Reverse Solidus |
| ] | Right Square Bracket |
| ^ | Circumflex Accent |
| _ | Underline |
| ` | Grave Accent |
| { | Left Curly Bracket |
| \| | Vertical Line |
| } | Right Curly Bracket |
| ~ | Tilde |
| NBSP | No-Break Space |
| ¡ | Inverted Exclamation Mark |
| ¢ | Cent Sign |
| £ | Pound Sign |
| ¤ | Currency Sign |
| ¥ | Yen Sign |
| ¦ | Broken Bar |
| § | Paragraph Sign, Section Sign |
| ¨ | Diaeresis, Umlaut |
| © | Copyright Sign |
| ª | Feminine Ordinal Indicator |
| « | Left Angle Quotation Mark |
| ¬ | Not Sign |
| SHY | Soft Hyphen |
| ® | Registered Trade Mark Sign |
| ¯ | Macron, Overline, Overbar |
| ° | Degree Sign |
| ± | Plus-Minus Sign |
| $^2$ | Superscript Two |
| $^3$ | Superscript Three |
| ´ | Acute Accent |
| µ | Small Greek Letter Mu, Micro Sign |
| ¶ | Pilcrow (Paragraph Symbol) |
| · | Middle Dot |
| ¸ | Cedilla |
| $^1$ | Superscript One |
| º | Masculine Ordinal Indicator |
| » | Right Angle Quotation Mark |
| ¼ | Vulgar Fraction One Quarter |
| ½ | Vulgar Fraction One Half |
| ¾ | Vulgar Fraction Three Quarters |
| ¿ | Inverted Question Mark |
| À | Capital A With Grave Accent |
| Á | Capital A With Acute Accent |
| Â | Capital A With Circumflex Accent |
| Ã | Capital A With Tilde |
| Ä | Capital A With Diaeresis |
| Å | Capital A With Ring Above |
| Æ | Capital Dipthong A with E |
| Ç | Capital C With Cedilla |
| È | Capital E With Grave Accent |
| É | Capital E With Acute Accent |
| Ê | Capital E With Circumflex Accent |
| Ë | Capital E With Diaeresis |
| Ì | Capital I With Grave Accent |
| Í | Capital I With Acute Accent |
| Î | Capital I With Circumflex Accent |
| Ï | Capital I With Diaeresis |
| Ð | Capital Icelandic Eth |
| Ñ | Capital N With Tilde |

86-060

| | |
|---|---|
| Ò | Capital O With Grave Accent |
| Ó | Capital O With Acute Accent |
| Ô | Capital O With Circumflex Accent |
| Õ | Capital O With Tilde |
| Ö | Capital O With Diaeresis |
| x | Multiplication Sign |
| Ø | Capital O With Oblique Stroke |
| Ù | Capital U With Grave Accent |
| Ú | Capital U With Acute Accent |
| Û | Capital U With Circumflex Accent |
| Ü | Capital U With Diaeresis |
| Ý | Capital Y With Acute Accent |
| Þ | Capital Icelandic Thorn |
| ß | Small German Sharp s |
| à | Small a With Grave Accent |
| á | Small a With Acute Accent |
| â | Small a With Circumflex Accent |
| ã | Small a With Tilde |
| ä | Small a With Diaeresis |
| å | Small a With Ring Above |
| æ | Small Dipthong a With e |
| ç | Small c With Cedilla |
| è | Small e With Grave Accent |
| é | Small e With Acute Accent |
| ê | Small e With Circumflex Accent |
| ë | Small e With Diaeresis |
| ì | Small i With Grave Accent |
| í | Small i With Acute Accent |
| î | Small i With Circumflex Accent |
| ï | Small i With Diaeresis |
| ð | Small Icelandic Eth |
| ñ | Small n With Tilde |
| ò | Small o With Grave Accent |
| ó | Small o With Acute Accent |
| ô | Small o With Circumflex Accent |
| õ | Small o With Tilde |
| ö | Small o With Diaeresis |
| ÷ | Division Sign |
| ø | Small o With Oblique Stroke |
| ù | Small u With Grave Accent |
| ú | Small u With Acute Accent |
| û | Small u With Circumflex Accent |
| ü | Small u With Diaeresis |
| ý | Small y With Acute Accent |
| þ | Small Icelandic Thorn |
| ÿ | Small y With Diaeresis |

86-061

Table D-1. Extended ASCII Character Set

| | | | | | Set C0 | | Set G0 | | | | | | Set C1 | | Set G1 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | $b_4$ | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | | | | $b_3$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | | | | $b_2$ | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | | $b_1$ | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| $b_4$ | $b_3$ | $b_2$ | $b_1$ | $H_2$ \ $H_1$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0 | 0 | 0 | 0 | 0 | NUL | DLE | SP | 0 | @ | P | ` | p | | | NBSP | ° | À | Đ | à | ð |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | ! | 1 | A | Q | a | q | | | ¡ | ± | Á | Ñ | á | ñ |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | " | 2 | B | R | b | r | | | ¢ | ² | Â | Ò | â | ò |
| 0 | 0 | 1 | 1 | 3 | ETX | DC3 | # | 3 | C | S | c | s | | | £ | ³ | Ã | Ó | ã | ó |
| 0 | 1 | 0 | 0 | 4 | EOT | DC4 | $ | 4 | D | T | d | t | | | ¤ | ´ | Ä | Ô | ä | ô |
| 0 | 1 | 0 | 1 | 5 | ENQ | NAK | % | 5 | E | U | e | u | | | ¥ | µ | Å | Õ | å | õ |
| 0 | 1 | 1 | 0 | 6 | ACK | SYN | & | 6 | F | V | f | v | | | ¦ | ¶ | Æ | Ö | æ | ö |
| 0 | 1 | 1 | 1 | 7 | BEL | ETB | ' | 7 | G | W | g | w | | | § | • | Ç | × | ç | ÷ |
| 1 | 0 | 0 | 0 | 8 | BS | CAN | ( | 8 | H | X | h | x | | | ¨ | ¸ | È | Ø | è | ø |
| 1 | 0 | 0 | 1 | 9 | HT | EM | ) | 9 | I | Y | i | y | | | © | ¹ | É | Ù | é | ù |
| 1 | 0 | 1 | 0 | A | LF | SUB | * | : | J | Z | j | z | | | ª | º | Ê | Ú | ê | ú |
| 1 | 0 | 1 | 1 | B | VT | ESC | + | ; | K | [ | k | { | | | « | » | Ë | Û | ë | û |
| 1 | 1 | 0 | 0 | C | FF | FS | , | < | L | \ | l | \| | | | ¬ | ¼ | Ì | Ü | ì | ü |
| 1 | 1 | 0 | 1 | D | CR | GS | – | = | M | ] | m | } | | | SHY | ½ | Í | Ý | í | ý |
| 1 | 1 | 1 | 0 | E | SO | RS | . | > | N | ^ | n | ~ | | | ® | ¾ | Î | Þ | î | þ |
| 1 | 1 | 1 | 1 | F | S1 | US | / | ? | O | _ | o | DEL | | | ¯ | ¿ | Ï | ß | ï | ÿ |

86-1011

Table D-2. EBCDIC Character Set

| | | | | b4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | b3 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| | | | | b2 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 |
| | | | | b1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| b4 | b3 | b2 | b1 | H2 \ H1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | A | B | C | D | E | F |
| 0 | 0 | 0 | 0 | 0 | NUL | DEL | DS | | SP | & | – | ø | Ø | ° | µ | ¢ | { | } | \ | 0 |
| 0 | 0 | 0 | 1 | 1 | SOH | DC1 | SOS | | NBSP | é | / | É | a | j | ~ | £ | A | J | NSP | 1 |
| 0 | 0 | 1 | 0 | 2 | STX | DC2 | FS | SYN | â | ê | Â | Ê | b | k | s | Y | B | K | S | 2 |
| 0 | 0 | 1 | 1 | 3 | ETX | TM | | | ä | ë | Ä | Ë | c | l | t | • | C | L | T | 3 |
| 0 | 1 | 0 | 0 | 4 | PF | RES | BYP | PN | à | è | À | È | d | m | u | ƒ | D | M | U | 4 |
| 0 | 1 | 0 | 1 | 5 | HT | NL | LF | RS | á | í | Á | Í | e | n | v | § | E | N | V | 5 |
| 0 | 1 | 1 | 0 | 6 | LC | BS | ETB | UC | ã | î | Ã | Î | f | o | w | ¶ | F | O | W | 6 |
| 0 | 1 | 1 | 1 | 7 | DEL | IL | ESC | EOT | å | ï | Å | Ï | g | p | x | ¼ | G | P | X | 7 |
| 1 | 0 | 0 | 0 | 8 | | CAN | | | ç | ì | Ç | Ì | h | q | y | ½ | H | Q | Y | 8 |
| 1 | 0 | 0 | 1 | 9 | RLF | EM | | | ñ | β | Ñ | ` | i | r | z | ¾ | I | R | Z | 9 |
| 1 | 0 | 1 | 0 | A | SMM | CC | SM | | [ | ] | ¦ | : | « | ª | ¡ | ¬ | SHY | ı | $^2$ | $^3$ |
| 1 | 0 | 1 | 1 | B | VT | CU1 | CU2 | CU3 | • | $ | , | # | » | º | ¿ | \| | ô | û | Ô | Û |
| 1 | 1 | 0 | 0 | C | FF | IFS | | DC4 | < | * | % | @ | ð | æ | Đ | — | ö | ü | Ö | Ü |
| 1 | 1 | 0 | 1 | D | CR | IGS | ENQ | NAK | ( | ) | _ | ' | ý | ¸ | Ý | ¨ | ò | ù | Ò | Ù |
| 1 | 1 | 1 | 0 | E | SO | IRS | ACK | | + | ; | > | = | þ | Æ | Þ | ´ | ó | ú | Ó | Ú |
| 1 | 1 | 1 | 1 | F | SI | IUS | BEL | SUB | ! | ^ | ? | ” | ± | ¤ | ® | ‗ | õ | ÿ | Õ | |

**NOTE**

$H_1$ signifies the high bits (leftmost), and $H_2$ signifies the low bits (rightmost). For example, the letter a is 81 (Hex) or 1000 0001 (binary).

86-005

# *Appendix E* *DEVICE-SPECIFIC CONTROL CHARACTERS*

Tables E-1 and E-2 list the TTY and VIP nonalphanumeric control characters for devices supported by the communications subsystem.

NOTE

In this appendix, a slash between two characters indicates that both keys are pressed simultaneously, e.g., CTRL/H indicates that the CTRL key and H key are pressed at the same time.

Table E-1. TTY Nonalphanumeric Control Characters

| Character | Hex Value | Function | Key Strokes |
|---|---|---|---|
| ENQ | 05 | Answer back | CTRL/E |
| BEL | 07 | Ring Bell | CTRL/G |
| BS | 08 | Backspace (nondestructive cursor backward) | CTRL/H |
| LF | 0A | Line feed | CTRL/J |
| FF | 0C | Form feed (clear screen) | CTRL/L |

Table E-1 (cont). TTY Nonalphanumeric Control Characters

| Character | Hex Value | Function | Key Strokes |
|---|---|---|---|
| CR | 0D | Carriage return | CTRL/M |
| SP | 20 | Space | CTRL/P or space bar |

NOTE

In a terminal with lowercase capability, uppercase characters require the use of the shift.

Table E-2. VIP Nonalphanumeric Control Characters

| Character | Hex Value | Function | Key Strokes |
|---|---|---|---|
| BS | 08 | Backspace. | CTRL/H |
| HT | 09 | Horizontal tab. | CTRL/I |
| LF | 0A | Line feed. | CTRL/J or LINE FEED |
| FF | 0C | Form feed. | CTRL/L |
| CR | 0D | Carriage return. | CTRL/M or RETURN |
| ESC | 1B | First character of a 2-, 3-, or 4-character escape sequence used for VIP terminal control. | ESC |
| SP | 20 | Space. | CTRL/P or space bar |

# *Appendix F*
# *SUBSYSTEM MODULES*

This appendix describes subsystem modules: their purpose, structure, and interface with the Edit Profile and List Profile utilities. The intent of this appendix is to help developers create their subsystem modules according to the requirements of these two utilities. For the sake of clarity, references to the profiles file will be more technical than in the user-oriented manuals. Specifically, a 'section' of a user profile is a record in the profiles file; the 'attributes' of a section are the fields of the record; a 'section id' is a two-character record-type identifier located in the record header.

## SUBSYSTEM RECORDS

A subsystem record is 188 bytes long and consists of a system-defined portion and a subsystem-defined portion. The system-defined portion extends from bytes 0 through 59. This area of the record cannot be accessed by the subsystem or subsystem module.

The remainder of the record, defined by the subsystem, is subdivided into two regions. Subsystem region 1, also known as the access level region, extends from bytes 60 through 97. It can be read by the subsystem proper but can be written to only by Edit Profile (via a subsystem module). Region 2 extends from bytes 98 through 188. It can be read or written to both by the subsystem proper and Edit/List Profile (via a subsystem module). The subsystem proper uses profiles file macro calls, documented in Volume II of this manual, to read and write the subsystem records.

Subsystem-defined fields processed by Edit/List Profile (via a subsystem module) must begin on an even byte (word boundary).

## EDIT PROFILE (EP) SUBSYSTEM MODULES

When the System Administrator uses Edit Profile's ADD, MOD, or STATS functions on a subsystem record, EP calls the subsystem module of that record type. The module provides EP with code and data needed to perform the functions.

The module is a separate bound unit, linked non-sharable, and without overlays. The naming convention is EP_id, where id is the two-character record type identifier (section_id). The module resides in a directory under the loader's search rules.

An EP module contains up to seven fundamental elements listed below. The location within the module of any of these elements is not important except for the pointer array, which must begin at word one. Word zero (start address) must be the instruction jmp $B5 (8385). This prevents the module from being executed as an ECL command, which would cause a trap.

Elements of an EP module are:

1. Pointer array.
2. MOD function message number.
3. MODIFY routine.
4. Subsystem default values.
5. ADD routine.
6. STAT-names message number.
7. STATS descriptor table.

Elements 6 and 7 need only exist if the subsystem record contains statistics fields for display by the STATS function.

### Pointer Array

The pointer array starts at word 1 of the subsystem module and contains six IMA pointers to the elements (2 through 7) listed above (in the same order). If the subsystem record contains no statistics, then the pointers to elements 6 and 7 are zero.

The PTRAY assembler control statement is useful for creating the pointer array.

### MOD Function Message Number

This is a five digit number defined as a hex string constant (i.e., DC Z'nnnnn'), identifying a message in the message library. The message is actually a table of names that is displayed in list form by Edit Profile under the MOD function.

The entries in the table indicate which fields can be modified in the record. You may choose to specify individual field names or, instead, group fields into categories, in which case the table would contain the category names. The latter may be more helpful but would entail more work because Edit Profile processes only the initial table entries; your MOD routine would have to display the elements of the chosen category. In either case, the MOD routine must be coordinated with the make-up of this table.

The format of the table is as follows:

```
name 1/name 2/name 3...name n!
```

Note that the entries are lower case, the slash character is used as a separator, and the exclamation point is the end-of-table marker. All other characters, including space, are legal for a field name. The maximum length of an individual entry is 22 characters. The maximum total length is 240 characters (this is a message library limitation).

The following example shows the first pointer in the array pointing at the message number. Also shown is the message as it would appear in the message library.

```
ptrary DC <msgnum
       .
       .
       .
       .

msgnum DC Z'26301'
```

Message library entry:

```
263010100000 login_id/login line defaults/current
terminal/language key/login traits/password status!
```

Creation of the message and insertion into the message library must follow the rules of the Message Reporter, which are explained in the Application Developer's Guide.

Under the MOD function, Edit Profile retrieves the table from the message library and displays it as a list with each entry assigned an incremental number. Edit Profile also displays a 'NONE OF THE ABOVE' option following the last table entry, and prompts the user's selection. Figure F-1 shows the format of this list.

```
             XX Section Menu

        (1)    name 1
        (2)    name 2
        (3)    name 3
                 .
                 .
                 .
        (n)    name n
        (n+1)  NONE OF THE ABOVE

        Selection:
```

Figure F-1. MOD Function List Format

## MODIFY Routine

The subsystem module must contain a routine to change in memory the field that the user has selected. When the MOD function is executed, Edit Profile does the following:

1. Loads ($BULD) the appropriate subsystem module into memory.
2. Reads the subsystem record from the profiles file into Edit Profile's memory buffer.
3. Displays in menu-like form the subsystem-supplied table of names, each name assigned a number from 1 through n. The form also prompts the user's selection.
4. Validates the selection to be within the range 1 through n.
5. Converts the selected number to hexidecimal and loads it into $R1 (e.g., if 10 is selected, $R1 = 000A).
6. Points $B1 at the access level region (byte 60) of the record in memory.
7. Does a link and jump $B5 to the subsystem modify routine.

The subsystem MOD routine uses the value in $R1 to determine which entry was selected from the list. If a category was selected (assuming one was offered), the routine displays a similar type of list containing the entries under the chosen category, and then prompts for the user's selection.

When the user's selection identifies a specific field, the MOD routine prompts the user to supply the field's new contents/ value. The routine replaces (in memory) the old field contents with the new, puts zero in $Rl to signify a clean return, and jumps back to Edit Profile at the address originally in $B5. The MOD routine can alter only bytes 60 through 188 of the subsystem record (counting from physical byte 1).

The subsystem MOD routine:

- Uses the error-out and user-in paths for all dialogue
- Does input verification on user responses
- Does its own error reporting
- Accepts YES, Y, NO and N as responses to yes/no questions
- Recognizes '?' as the help key and responds with a help message if possible, or displays 'No help available'
- Recognizes '<' as a 'back-up' key and returns to the previous prompt/question. If '<' is received on the first prompt, puts -1 in $Rl and returns to Edit Profile.

When the subsystem MOD routine returns, Edit Profile checks $Rl and takes one of the following actions:

- $Rl = 0 (normal return). EP picks up at step 3 redisplaying the list but now offering two action keys (A) Accept and (N) Negate. If 'A' is selected, the record in memory (containing the changes) is written to the profiles file and the MOD function is exited.
- If 'N' is selected, the original record is read from the profiles file into memory, overwriting the changes made by the subsystem MOD routine. EP then announces that the changes have been negated and picks up again at step 3.
- $Rl = -1 (back-up key used). No change was made; EP redisplays the list (step 3). The action keys are not offered.
- $Rl = 0 or -1 (abnormal return). EP displays an error message and exits the MOD function.

<u>Subsystem Default Values</u>

The subsystem default values are a total of 128 bytes long and represent the initial contents of the subsystem-defined portion of the record (bytes 60 through 188).

When an ADD function is entered, EP creates a skeleton record by copying the values into a memory buffer before activating the subsystem ADD routine.

## ADD Routine

The subsystem module must contain a routine to build a new subsystem record in memory field-by-field using information gained from an interactive dialog with the system administrator.

When an ADD function is entered, Edit Profile builds a skeleton record in memory containing the subsystem default values. Edit Profile then sets $B1 pointing at the subsystem defined portion (byte 60) of the skeleton and does a link and jump $B5 to the subsystem ADD code.

The subsystem code issues directives and questions through the error-out file requesting the administrator to respond with field contents or answers to subsystem-specific questions. The responses are accepted through user-in and validated. If the response is invalid, the code issues a message to error-out indicating the problem and then re-issues the directive or question. Valid responses are put into the skeleton record at the appropriate offsets.

As in the MOD routine, the ADD routine must recognize the '?' (help) and '<' (back-up) keys as defined by Edit Profile.

Return to EP is at the orginal $B5 address. Upon return, EP checks the contents of $R1 as it does upon return from the MOD routine.

## STAT-Names Message Number

This is a 5-digit number defined as a hex string constant identifying a message in the message library. The message is a table of the names of the statistics fields that the STATS function is to display.

The format of the table is exactly the same as the one displayed by the MOD function and already described under "MOD Function Message Number."

## STATS Descriptor Table

The STATS descriptor table contains a 3-word entry for each statistic field in the record. The three words define the stat-type, offset, and size of the field.

The STAT-type word is a number indicating one of the STAT-types shown in Table F-1. The offset word indicates the offset of the field in words. (Keep in mind that the offset of the sixtieth word, for example, is 59.) The size word indicates the length of the field in words.

Table F-1. Edit Profile Statistic Field Types

| Type | Number |
|---|---|
| Decimal count | 1 |
| Hex count | 2 |
| Elapsed internal time | 3 |
| Internal date/time | 4 |

The following example shows a typical STATS descriptor table.

```
stats  DC 3, 49, 3 (elapsed time; offset: 49; size: 3 words)
          1, 52, 1 (decimal count; offset: 52; size: 1 word)
          2, 53, 2 (hex count; offset: 53; size: 2 words)
          4, 55, 3 (internal date/time; offset: 55; size: 3 words)
```

## LIST PROFILE (LP) SUBSYSTEM MODULES

When List Profile processes a subsystem record, it calls the LP subsystem module of that record type. The module is a separate bound unit, linked non-sharable and without overlays. The naming convention is LP_id, where id is the two character record type identifier (section_id). The module resides in a directory under the loader's search rules.

An LP subsystem module contains up to four fundamental elements listed below. The location within the module of any element is not important except for the pointer array, which must begin at word one. Word zero (start address) must be the instruction jmp $B5 (8385). This prevents the module from being executed as an ECL command, which would cause a trap.

Elements of an LP subsystem module are:

1. Pointer array.
2. Message number.
3. Descriptor table.
4. Special-field routine.

Element 4 need exist only if the subsystem record contains any special fields (see "Special Field Routine" later in this appendix).

## Pointer Array

The pointer array starts at word one of the module and contains three IMA pointers to the elements 2, 3, and 4 listed above (in the same order).

If the module does not contain a special-field routine, then the associated pointer is null (zeros).

## Message Number

This is a five digit number defined as a hex string constant (i.e., DC Z'nnnnn'), identifying a message in the message library. The message is actually a table of the field names in the subsystem record.

The field names are displayed in a column by List Profile when the record is listed. In format, the table is exactly the same as Edit Profile's table of modifiable field names, described earlier in this appendix under "MOD Function Message Number."

Creation of the message and insertion into the message library must follow the rules of the Message Reporter, which are explained in the Application Developer's Guide.

## Descriptor Table

The descriptor table contains a three-word entry for every field in the record. The three words define: field type, offset, and size. The field type word is a number indicating one or more field types shown in Table F-2. The offset word indicates the offset of the field in words. The size word indicates the length of the field in words. The table of field names and the descriptor table should contain entries only for subsystem-defined regions (i.e., bytes 60 through 188). The table should not contain entries for fields in the system-defined region of the record. No entry need exist for any unused area(s) in bytes 60 through 188.

## Special-Field Routine

Any field in the record whose contents require interpretation by the subsystem, such as an indicator word, cannot be processed by List Profile alone. Such fields require the existence of a subsystem module routine to interpret and display the field contents. When List Profile encounters a special field, it builds an argument structure comprising the following elements:

- Address of List Profile's memory buffer containing the subsystem record
- Address of List Profile's output buffer (points at the control word)

Table F-2. List Profile Field Types

| Type | Number |
|---|---|
| Decimal count | 1 |
| Hexadecimal count | 2 |
| Elapsed internal time | 3 |
| Internal date/time | 4 |
| RFU | 5 |
| Special field | 6 |
| ASCII | 7 |
| Radix 40 | 8 |
| Bit string | 9 |

- Offset of the current field (in words)
- Length of the current field (in words)
- Byte offset into List Profile's output buffer for placement of field contents.

The addresses are two words in length; the last three arguments are one word each.

List Profile places the ASCII field name in its output buffer, points $B4 at the argument structure, and does a link and jump $B5 to the subsystem routine. The routine uses the information in the argument block to do the following:

- Determine which special field is being processed (if there is more than one in the record)
- Interpret the contents of the field
- Place the translated ASCII meaning in List Profile's output buffer at the supplied offset
- Display the output buffer to user-out. (List Profile's output buffer is 80 bytes long including the control word. The maximum length of a field's contents is 60 ASCII characters.)

The routine maintains control for as long as it needs. In the case of an indicator word, for example, each bit may require a separate display, in which case the routine would have to clean the buffer after a $USOUT, interpret and print the next bit, and so on.

When done, the routine uses $R1 as a status register: a zero in $R1 signifies normal return; a non-zero value in $R1 signifies an error. The subsystem returns to List Profile at the address originally supplied in $B5.

## ASCII-ONLY SUBSYSTEM RECORDS

The Edit Profile and List Profile utilities can add, modify, and list a subsystem record without the use of a subsystem module if both of the following conditions are met:

- The subsystem-defined portion of the record contains only ASCII data
- The subsystem was declared (by the DEC function) to operate in this mode of record maintenance.

Edit Profile and List Profile view an ASCII-only record as having two data regions: region one in bytes 60 through 97; region two in bytes 98 through 188.

Under the ADD and MOD functions, Edit Profile prompts the System Administrator as follows:

```
Enter data for region 1:

Enter data for region 2:
```

Any unused portion of either region is blank filled. List Profile displays all of region 1 (38-characters) and 50 of the 90-characters in region 2.

## *MANUAL DIRECTORY*

The following manuals support the MOD 400 operating system.

| Base Publication Number | Manual Title |
|---|---|
| HE01 | ONE PLUS Guide to Software Documentation |
| CZ02 | GCOS 6 MOD 400 System Building and Administration |
| CZ03 | GCOS 6 MOD 400 System Concepts |
| CZ04 | GCOS 6 MOD 400 System User's Guide |
| CZ05 | GCOS 6 MOD 400 System Programmer's Guide - Volume I |
| CZ06 | GCOS 6 MOD 400 System Programmer's Guide - Volume II |
| CZ07 | GCOS 6 MOD 400 Programmer's Pocket Guide |
| CZ09 | GCOS 6 MOD 400 System Maintenance Facility Administrator's Guide |
| CZ10 | GCOS 6 MOD 400 Menu System User's Guide |
| CZ11 | GCOS 6 MOD 400 Software Installation Guide |
| CZ15 | GCOS 6 MOD 400 Application Developer's Guide |
| CZ16 | GCOS 6 MOD 400 System Messages |
| CZ17 | GCOS 6 MOD 400 Commands |
| CZ18 | GCOS 6 Sort/Merge |
| CZ19 | GCOS 6 Data File Organizations and Formats |
| CZ20 | GCOS 6 MOD 400 Transaction Control Language Facility |
| CZ21 | GCOS 6 MOD 400 Display Formatting and Control |
| CZ22 | GCOS 6 VISION Reference Manual |
| GZ13 | GCOS 6 MOD 400 R3.1 to R4.0 Migration Guide |
| HC01 | GCOS 6 MOD 400 Application Development Overview |

# INDEX

INDEX

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

| | | | |
|---|---|---|---|
| TITLE | DPS 6<br>GCOS 6 MOD 400<br>SYSTEM PROGRAMMER'S GUIDE – VOLUME I<br>ADDENDUM A | ORDER NO. | CZ05-02A |
| | | DATED | SEPTEMBER 1986 |

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE __________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

______________________

PLEASE FOLD AND TAPE–
NOTE: U.S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA 02154

POSTAGE WILL BE PAID BY ADDRESSEE

**HONEYWELL INFORMATION SYSTEMS**
**200 SMITH STREET**
**WALTHAM, MA 02154**

**ATTN: PUBLICATIONS, MS486**

**Honeywell**

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

| | | | |
|---|---|---|---|
| TITLE | DPS 6<br>GCOS 6 MOD 400<br>SYSTEM PROGRAMMER'S<br>GUIDE — VOLUME I | ORDER NO. | CZ05-02 |
| | | DATED | MARCH 1986 |

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

______________________

PLEASE FOLD AND TAPE—
NOTE: U. S. Postal Service will not deliver stapled forms

---

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
**200 SMITH STREET**
**WALTHAM, MA 02154**

**ATTN: PUBLICATIONS, MS486**

---

**Honeywell**

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

| | | | |
|---|---|---|---|
| TITLE | DPS 6<br>GCOS 6 MOD 400<br>SYSTEM PROGRAMMER'S<br>GUIDE — VOLUME I | ORDER NO. | CZ05-02 |
| | | DATED | MARCH 1986 |

ERRORS IN PUBLICATION

SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

______________________

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**
FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

**Honeywell**

HONEYWELL INFORMATION SYSTEMS
Technical Publications Remarks Form

| | | | |
|---|---|---|---|
| TITLE | DPS 6<br>GCOS 6 MOD 400<br>SYSTEM PROGRAMMER'S<br>GUIDE — VOLUME I | ORDER NO. | CZ05-02 |
| | | DATED | MARCH 1986 |

**ERRORS IN PUBLICATION**

**SUGGESTIONS FOR IMPROVEMENT TO PUBLICATION**

Your comments will be investigated by appropriate technical personnel and action will be taken as required. Receipt of all forms will be acknowledged; however, if you require a detailed reply, check here. ☐

FROM: NAME ______________________ DATE ____________

TITLE ______________________

COMPANY ______________________

ADDRESS ______________________

______________________

PLEASE FOLD AND TAPE–
NOTE: U. S. Postal Service will not deliver stapled forms

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

**BUSINESS REPLY MAIL**

FIRST CLASS PERMIT NO. 39531 WALTHAM, MA02154

POSTAGE WILL BE PAID BY ADDRESSEE

HONEYWELL INFORMATION SYSTEMS
200 SMITH STREET
WALTHAM, MA 02154

ATTN: PUBLICATIONS, MS486

**Honeywell**

45027, 0486, Printed in U.S.A.

CZ05-02