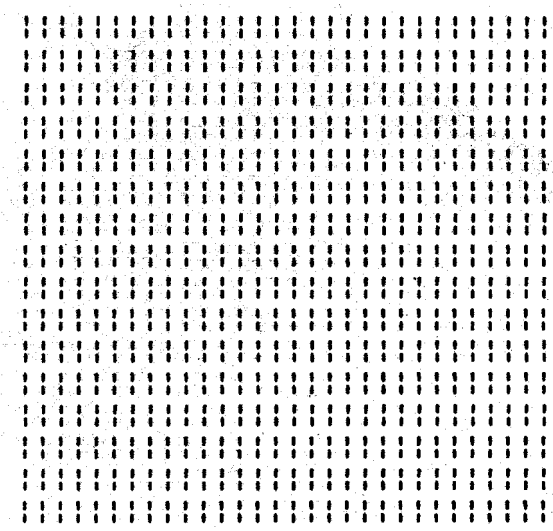
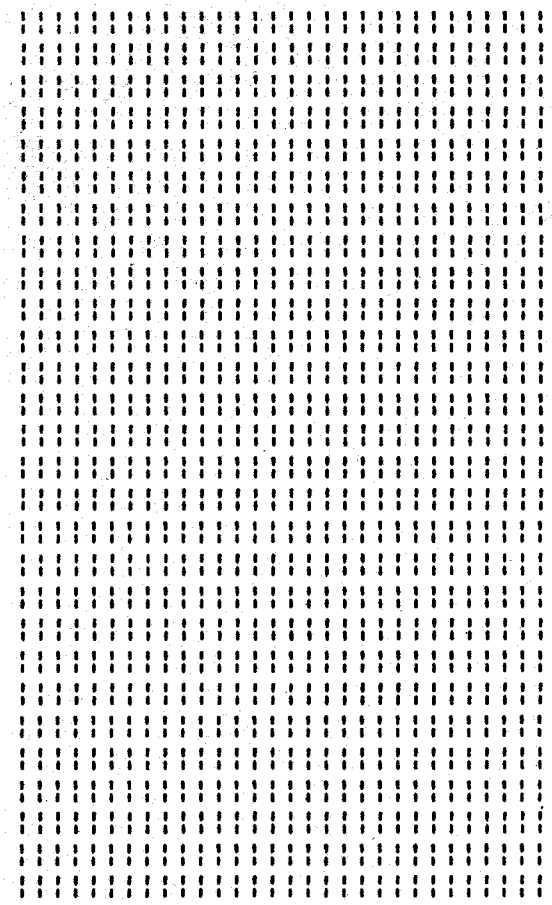


ZCPR3 and IOPs

A Tutorial on Input/Output Packages for the ZCPR3 User

Written by
Richard Conn



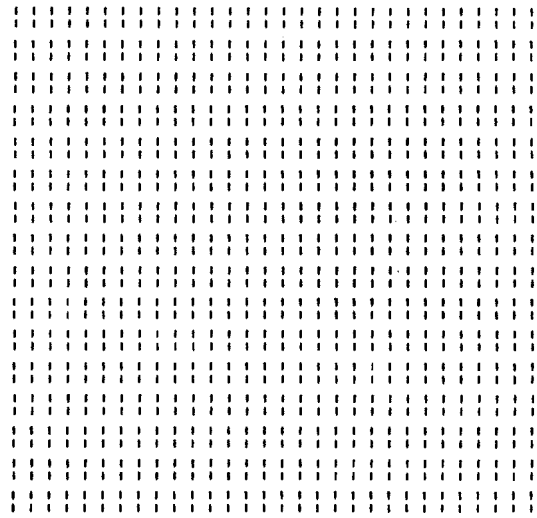
P R E F A C E

It has been over a year since the release of ZCPR3, and I have been using IOPs extensively in my work with and on the system since then. In many discussions with ZCPR3 users, it has been evident that the IOP concept and the uses of the IOP is not clear. While ZCPR3: The Manual (the reference book on ZCPR3) covers IOPs in some detail, we felt even more explanation was needed. Consequently, the purpose of this document is to discuss the use and implementation of IOPs in more detail. This document also provides a useful supplement to the documentation on the TERM III Communications System (contact Echelon, Inc.).

This document is written for an experienced ZCPR3 user with assembly language programming experience. Knowledge of the use and internal operation of ZCPR3 is necessary. The references section provides pointers to sources of more information on ZCPR3, ZRDOS, and the Z-System.

Richard Conn
July 31, 1985

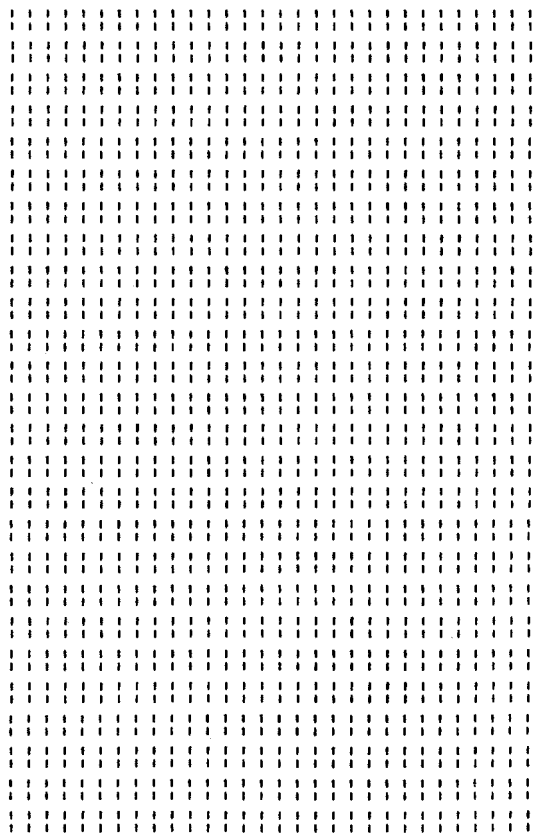
(PAGE INTENTIONALLY BLANK)



ZCPR3 and IOPs

A Tutorial on Input/Output Packages for the ZCPR3 User

Written by
Richard Conn



Distributed by:
Echelon, Inc.
101 First Street
Los Altos, CA 94022
415/948-3820

Z-Node Central:
415/489-9005

(PAGE INTENTIONALLY BLANK)

T A B L E O F C O N T E N T S

(PAGE INTENTIONALLY BLANK)

1. The Concept of an IOP

Input/Output Packages (IOPs) are segments of the Z-System (ZCPR3 with, optionally, the ZRDOS replacement for the CP/M BDOS) which contain groups of Input/Output device drivers for the Console, Reader, Punch, and List logical devices. A Z-System can contain several System Segments:

System Segment	Brief Description
Environment Descriptor	Provides information about the Z-System
Terminal Capabilities (TCAP or Z3T) Data	Provides information on how to control the user's terminal
Named Directory Buffer (NDR)	Contains data relating directory names to disks and user areas
Resident Command Package (RCP)	Contains a group of commands that stays resident in memory
Flow Command Package (FCP)	Contains IF, ELSE, FI (End IF), and XIF (Exit all IFs) commands
Input/Output Package (IOP)	Contains device drivers for the logical devices

FIGURE: Z-System Segments

All System Segments associated with a Z-System are optional, but the installation of the Environment Descriptor and Terminal Capabilities segments is highly recommended, and all System Segments should be installed in order to realize the full potential of the Z-System.

1.1. The BIOS

The BIOS is the system-dependent part of a CP/M or Z-System. A CP/M BIOS is functionally identical to a Z-System BIOS. The BIOS is divided into two parts: a jump table at the beginning of the BIOS and the routines which implement the BIOS functions within the body of the BIOS. The jump table provides a transportable method of accessing the routines within the BIOS. The base address of the BIOS jump table is found by subtracting 3 bytes from the address at memory locations 1 and 2. Knowing that all jumps in the table are three bytes long and in a specific order, access to any of the routines in the BIOS can be acquired by determining the offset of the desired BIOS routine from the base address of the BIOS jump table and adding this offset to the base address. The following figure outlines the structure of the BIOS jump table:

Class	Offset Hex	Offset Dec	Mnemonic	Comment
Bootstrap	00	0	BOOT	"Cold" (first-time) boot
Routines	03	3	WBOOT	"Warm" ("'C) boot
	06	6	CONST	Console **input status**
Character	09	9	CONIN	Console input character
Input/Output	OC	12	CONOUT	Console output.character
Routines	OF	15	LIST	List output **character**
	12	18	PUNCH	Punch output character
	15	21	READER	Reader input character
	18	24	HOME	Home disk **heads**
Disk	1B	27	SELDSK	Select **logical disk**
Access/Control	1E	30	SETTRK	Set track number
Routines	21	33	SETSEC	Set sector number
	24	36	SETDMA	Set DMA address
	27	39	READ	Read sector (128 bytes)
	2A	42	WRITE	Write sector (128 bytes)
List Status	2D	45	LISTST	List output **status**
Sector Xlate	30	48	SECTRAN	Sector translation

FIGURE: The BIOS Jump Table

Each of the BIOS routines is precisely defined. The functions of the routines, the registers used to pass input data to the routines, the registers used to pass output data from the **routines, and the side effects caused by the execution** of the routines are known from documentation available through a variety of sources **(see the references in Appendix A).**

From the BIOS Jump Table figure, it is apparent that there are three groups of routines:

1. the initialization routines (COLD and WARM)
2. the character input/output routines (including LISTST)
3. the disk access/control routines (including SECTRAN)

A Z-System supporting an IOP places **the second** group of routines, **the character** input/output **routines** (including LISTST), into the body of **the IOP rather than the body** of the BIOS. The BIOS jump **table** is identical in format to that of a standard BIOS, but the addresses of **the character input/output** routines are addresses of entry points in the IOP rather than **addresses of routines in** the body of the BIOS. The IOP, like the BIOS, consists of a jump **table followed** by the **routines of the** IOP. The jump **table entries** in the BIOS. for the character input/output routines address the jump table entries in the IOP, which in turn address **the routines in** the IOP.

1.2. Devices and Device Drivers

Consider the following: DO NOT THINK OF A DEVICE AS A PIECE OF HARDWARE, LIKE A CRT OR PRINTER. Instead, think in terms of device drivers, where the device is a logical entity consisting of zero or more pieces of hardware. The device is defined by a piece of software in a BIOS or IOP, and this piece of software is called a device driver. Device drivers are software, and devices are physical or logical hardware entities.

The BIOS supports this concept directly. The CONOUT (console output) routine outputs to a device called a "console". This "console" is assumed by many users to be a CRT, but it does not have to be. It could be:

1. a CRT
2. a modem
3. a null routine (bit-bucket, or simple return)
4. a CRT and modem in parallel (output goes to both a CRT and a modem before the CONOUT routine returns to its caller)
5. a CRT and printer in parallel (output goes to both a CRT and a printer)
6. a CRT, modem, and printer in parallel (output goes to all three)

There are many other possibilities, and in applications like a Remote Access System (RAS) where the console is a modem and various users access the computer over a telephone line, the number of possibilities grows quickly. Some RAS console devices could be:

1. a modem which also monitors a timer and reboots the system when a time period elapses
2. a CRT/modem combination where input comes from either the CRT or modem and output goes to both in parallel
3. a CRT/modem combination where input comes from the CRT and output goes to the CRT and modem in parallel
4. a modem which does not allow certain characters (like "C) to be accepted as input
5. a modem which also monitors a carrier detect line and reboots the microcomputer if this carrier signal is lost (the user hangs up)

A typical microcomputer running the Z-System supports a CRT, a modem, and a printer (at least), and the number of console devices which are possible is significantly greater than four. The capacity of the standard I/O byte (which supports at most four consoles) has been exceeded.

1.3. Advantages of an IOP

The IOP provides flexibility not found under conventional implementations of device I/O in the BIOS (with or without the I/O Byte). While designing ZCPR3, I found the need for the IOP to be pronounced. Relying on the BIOS for all the device I/O support was too restrictive:

1. the BIOS had to fit on the system tracks, so it was tight on space
2. the BIOS had no hooks (via the standard I/O Byte) to provide for more than four console, four reader, four punch, and four list devices

The implementation of the IOP eliminates both of these restrictions. All device drivers which were previously in the BIOS could now be placed within the IOP. The burden of initializing the IOP on cold boot was added to the BIOS, but this new overhead was far less than the overhead of the character I/O code which was replaced.

The IOP Buffer, being external to the BIOS, can be sized to meet the user's needs. Usually 1K or 1.5K bytes is adequate for most needs (Echelon has standardized on a length of 1.5K bytes). A large number of devices can be supported in this amount of space, and the LDR command allows an indefinite number of IOPs to be loaded as needed into the IOP buffer.

Finally, the IOP is name-oriented. All devices in an IOP can be referred to by name through the DEV and DEVICE commands of ZCPR3. DEV and DEVICE simplify the user's access to and selection of devices in an IOP, allowing him to review all possible device selections and select the desired devices by specifying a meaningful name. Each IOP contains its own set of device names and explanatory comments, so remembering which devices are available in a particular IOP is unnecessary.

2. IOP Initialization by the BIOS

The functions of the Z-System BIOS which uses an IOP are:

1. to initialize the IOP and other System Segments of the Z-System in the cold boot routine
2. to address into the IOP for all of its character input/output routines from the BIOS jump table
3. to support all disk routines
4. to load the Multiple Command Line buffer with a command sequence which performs further initialization (and loading) of the ZCPR3 system (including the System Segments)

The BIOS must store a simple IOP into the IOP buffer in memory, and this IOP will be replaced with a more complete IOP when the default command line is executed after the BIOS cold boot procedure completes. This default command line usually contains a ZCPR3 alias which includes an invocation of the LDR program to load a SYS.IOP segment. The following figures show some sample code which performs an IOP initialization. The BIOS jump table which addresses into an IOP is also shown.

```
iop       equ  Of600h          ;Address of IOP Buffer
          lxi  h,iodrivers     ;Default IOP Jump Table in BIOS
          lxi  d,iop           ;Location of IOP Buffer
          mvi  b,11*3          ;Size of jump table (11 3-byte
                                ; jumps)
copyiop:
          mov  a,m             ;Copy IOP jump table from BIOS
          stax d               ; into IOP Buffer
          inx  h
          inx  d
          dcr  b
          jnz  copyiop
          ...                  ;other code in cold boot routine
          ret
```

FIGURE: Sample IOP Buffer Initialization in the BIOS Cold Boot Routine

Primitive I/O Drivers which are loaded at Cold Boot time.

```
iodrivers:
iopstat:   xra  a              ;no IOP Status Routine
           ret
           db   0              ;fill out 3 bytes for jump table
iopsel:    xra  a              ;no IOP Select Routine
           ret
           db   0              ;fill out 3 bytes for jump table
iopname:   xra  a              ;no IOP Namer Routine
           ret
           db   0              ;fill out 3 bytes for jump table
iopinit:   ret                 ;Initialize Terminal
           db   0,0            ;Fill 3 bytes
```

The following routines are jumped into from the BIOS jump table

```
iconst:    xra  a              ;Console Input Status
           ret                 ;indicate that no char is pending
           db   0              ;Fill 3 bytes
iconin:    xra  a              ;Console Input Character
           ret                 ;return a binary 0
           db   0
iconout:   ret                 ;Console Output Character
           db   0,0            ;Fill 3 bytes
ilist:     ret                 ;List Output Character
           db   0,0            ;Fill 3 bytes
ipunch:    ret                 ;Punch Output Character
           db   0,0            ;Fill 3 bytes
ireader:   xra  a              ;Reader Input Character
           ret                 ;return a binary 0
           db   0
iliststst: on   Offh           ;List Status
           ret                 ;this always returns with A=OFFH
```

FIGURE (con't): Sample IOP Buffer Initialization in the BIOS Cold Boot Routine

```
iop       equ   Of600h          ;Address of IOP Buffer

          org   bios            ;BIOS starting address

          jmp   cboot           ;Cold boot entry point (in BIOS)
          jmp   wboot           ;Warm boot entry point (in BIOS)

          jmp   iop+12          ;Console status routine (in ION
          jmp   iop+15          ;Console input (in ION
          jmp   iop+18          ;Console output (in ION
          jmp   iop+21          ;List device output (in ION
          jmp   iop+24          ;Punch device output (in ION
          jmp   iop+27          ;Reader device input (in ION

          jmp   home            ;Home drive (in BIOS)
          jmp   setdrv          ;Select disk (in BIOS)
          jmp   settrk          ;Set track (in BIOS)
          jmp   setsec          ;Set sector (in BIOS)
          jmp   setdma          ;Set DMA address (in BIOS)
          jmp   read            ;Read the disk (in BIOS)
          jmp   write           ;Write the disk (in BIOS)

          jmp   iop+30          ;List device status (in ION

          jmp   sectran         ;Sector translation (in BIOS)
```

FIGURE: Sample BIOS Jump Table in Support of an IOP

With the code from both of these figures in the BIOS, the IOP is properly initialized and the BIOS is configured to use the IOP for all of its character input/output functions. The IOP presented in these figures, however, is extremely simple and only acts as a place holder to keep the system from crashing until the LDR program runs to load a useful IOP.

The LDR program must execute automatically on cold boot, and this is ensured under the Z-System by placing a command which runs LDR on an IOP in the Multiple Command Line Buffer of ZCPR3. Code which performs the proper initialization of the Multiple Command Line Buffer (MCL) could look something like this:

```
cmdline   equ   Of200h          ;address of MCL buffer

          lxi   h,defcmd        ;address of default command line
          lxi   d,cmdline       ;address of MCL buffer
          mvi   b,40            ;arbitrary 40 bytes
copymcl:
          mov   a,m             ;Copy default MCL from BIOS
          stax  d               ; into MCL Buffer
          inx   h
          inx   d
          dcr   b
          jnz   copymcl
          ...                   ;other code in BIOS
          ret

defcmd:   dw    cmdline+4       ;address of first character to run
          dw    0               ;filler
          db    'LDR SYS.IOP',0     ;startup command line
```

FIGURE: Sample MCL Buffer Initialization in the BIOS Cold Boot Routine

The default command line could be an alias (like STARTUP), where this alias contains the command "LDR SYS.IOP". Use of the alias is the preferred technique since it adds flexibility and ease of reconfiguration as the user's needs change. Once a Z-System has cold booted, the STARTUP alias can easily be changed to add or delete cold boot commands.

3. Design and implementation of an IOP

An IOP is laid out in a manner similar to a BIOS. All IOPs are divided into two parts:

1. a jump table at the front
2. a set of supporting routines after the jump table

All routines in an IOP are defined in terms of their function, input parameters, output parameters, and side effects. This chapter serves to document all routines within an IOP.

3.1. IOP Jump Table

The jump table of an IOP is organized as follows:

Section	Offset Hex	Offset Dec	Mnemonic	Descrit•,tion of Routine
IOP	00	0	STATUS	IOP Status Reporting
Status	03	3	SELECT	IOP Device Selection.
and	06	6	NAMER	IOP Device Name Reporting
Control	09	9	INIT	IOP Initialization
	OC	12	CONST	Console Input Status
	OF	15	CONIN	Console Input Character
	12	18	CONOUT	Console Output Character
BIOS	15	21	LIST	List Output Character
Interface	18	24	PUNCH	Punch Output Character
	1B	27	READER	Reader Input Character
	lE	30	LISTST	List Output Status
IOP Patch	21	33	PATCH	Patch Console
	24	36	COPEN	Open Console Recorder
IOP	27	39	CCLOSE	Close Console Recorder
Recorder	2A	42	LOPEN	Open List Recorder
	2D	45	LCLOSE	Close List Recorder
IOP ID	30	48	ID	IOP ID (5 bytes = Z3IOP)

FIGURE: Jump Table and ID of an IOP

The IOP Status and Control routines are used to initialize the IOP, determine the type of the IOP, obtain names of and comments on devices contained within the IOP, and select devices within the IOP. The ZCPR3 DEV and DEVICE utility commands perform all of their functions through these four routines.

The BIOS Interface routines in the IOP are the same as their counterparts in the BIOS. The BIOS simply indexes into the IOP at these routines (see the last chapter).

The IOP Patch routine is used to make temporary changes to a particular console selection. It allows the caller to provide his own drivers for a particular console and have the IOP call these when this console is selected.

The IOP Recorder provides for the facility of sending all output intended for a console or list device to a disk file and remote computer as well. The RECORD utility command of ZCPR3 controls this function.

The IOP ID is used to identify the IOP System Segment to the LDR command of ZCPR3. LDR will refuse to load an IOP into the IOP Buffer if this ID is not present. The ID consists of the five bytes 'Z310P'.

In all cases, the programmer must assume that (1) no registers are preserved by calling routines in an IOP, (2) the input and output parameter conventions presented in this document are adhered to, and (3) any register not specified in an output parameter list may be changed from its value before the call to the IOP routine.

3.2. IOP Status and Control Routines

This section describes the Status and Control Routines of the IOP. These routines are:

Section	Offset Hex	Offset Dec	Mnemonic	Description of Routine
IOP	00	0	STATUS	IOP Status Reporting
Status	03	3	SELECT	IOP Device Selection
and	06	6	NAMER	IOP Device Name Reporting
Control	09	9	INIT	IOP Initialization

3.2.1. STATUS

Brief Description:

STATUS returns information on the four logical devices (CON, RDR, PUN, and LST) supported by the IOP and the status of the IO Recorder function of the IOP. An identifying number of the IOP is also returned.

Discussion and Notes:

The IOP concept supports four logical devices, as a normal, unmodified BIOS does. Associated with each logical device is a byte pair which indicates how many device drivers are available for the logical device and which device driver is currently selected. This information is stored in a table as follows:

```
IOPTABLE:
     CON:       db   count,assignment
     RDR:       db   count,assignment
     PUN:       db   count,assignment
     LST:       db   count,assignment
```

In each case, "count" is the number of device drivers (0 to 255) available for the indicated logical device, and "assignment" is the number (0 to count-1) of the device driver which is currently assigned.

STATUS returns the base address (IOPTABLE) of this table in the HL register pair.

STATUS also returns with register A=0 and the Zero Flag Set if there is no I/O device support in the IOP. If there is I/O device support, the Zero Flag is cleared (NZ) and A is an indicator as follows:

MSB of A = 0	means	IO Recorder not available
MSB of A = 1	means	IO Recorder available
7 LSBs of A	mean	IOP Number (1..127)

If the IO Recorder routines are active, the most significant bit (MSB) of register A is set. Each IOP must have a number (selected by the implementer), and this is stored in the 7 least significant bits (LSBs) of register A. This number is not used by DEV, DEVICE, or RECORDER, and is of interest only to the implementer to track his various IOPs.

Input Parameters: None
Output Parameters:
HL = Address of IOP Status Table
A = Flag
A=0 and Zero Flag Set (Z) if no I/O device support
A<>0 if I/O device support --
MSB of register A indicates if IO Recorder available
other bits of register A indicate IOP number

3.2.2. SELECT

Brief Description:

SELECT is used to select a particular IOP device driver to be used for a given logical device.

Discussion and Notes:

Input parameters are passed in the BC register pair.

Register B is the logical device number, where CON = 0, RDR = 1, PUN = 2, and LST = 3. Any value greater than 3 results in no device selection and an error code.

Register C is the number of an IOP device driver to select. Register C may take on any value from 0 to count-1, where 'count' is the number of device drivers available for the given logical device. The STATUS routine can be used to determine the value of 'count'.

On exit, register A is an error code.

Input Parameters:
B = number of logical device
CON = 0 RDR = 1
PUN = 2 LST = 3
C = number of device driver (0 to count-1, where count is determined from the STATUS routine)

Output Parameters:
A=0 and Zero Flag Set (Z) if device selection error
(no device selected)
A=OFFH and NZ if selection made

3.2.3. NAMER

Brief Description:
NAMER returns the address of a string (sequence of bytes terminated by a null, or binary 0) which describes a device driver. This string may take one of two forms:

```
   db    'NAME ',0                      ;device name only, all caps
or db    'NAME textual description',0 ;name with description
```

NAME must be followed by at least one space.

Discussion and Notes:
NAMER returns a string containing the name and optional description of a device driverL The driver is referenced by a logical device number passed in register B and a device driver number passed in register C (same convention as for the SELECT routine).

The string MUST consist of at least a name followed by a space. Any text following this space is taken as a comment by the DEV and DEVICE commands.

Register B is the logical device number, where CON = 0; RDR = 1, PUN = 2, and LST = 3. Any value greater than 3 results in no device selection and an error code.

Register C is the number of an IOP device driver to provide the name for. Register C may take on any value from 0 to count-1, where 'count' is the number of device drivers available for the given logical device. The STATUS routine can be used to determine the value of 'count'.

Input Parameters:
B = number of logical device
CON = 0 RDR = 1
PUN = 2 LST = 3
C = number of device driver (0 to count-1, where count is determined from the STATUS routine)

Output Parameters:
HL = address of first character of a null-terminated string which is structured in one of two ways:

```
            db    'NAME ',0
or          db    'NAME textual description',0
```

A=Error code
A=0 and Zero Flag Set (Z) if device selection error
(B > 3 or C > count-1)
A=OFFH and NZ if valid device name string returned

3.2.4. INIT

Brief Description:
INIT initializes the devices controlled by the IOP.

Discussion and Notes:
INIT may perform any initializations required. These may include:

1. DART/USART communications attributes (number of bits to transmit, parity, etc)
2. baud rates
3. IO Recorder state (recommended to be set to OFF for both console and list recording)
4. the initial assignment of the device drivers for each logical device

Input Parameters: None
Output Parameters: None

3.3. BIOS Interface Routines

This section describes the BIOS Interface routines:

Section	Offset Hex	Offset Dec	Mnemonic	Descrij-ption of Routine
BIOS Interface	OC	12	CONST	Console Input Status
	OF	15	CONIN	Console Input Character
	12	18	CONOUT	Console Output Character
	15	21	LIST	List Output Character
	18	24	PUNCH	Punch Output Character
	1B	27	READER	Reader Input Character
	lE	30	LISTST	List Output Status

All of these routines support the same input and output parameters as the BIOS.

3.3.1. CONST

Brief Description:
CONST returns the input character status of the logical console ICON) device.

Discussion and Notes:
Register A=OFFH if a character is available from the console device. Register A=0 if no character is available.
Do not assume that the zero flag is set or reset. The value of the A register is the only output parameter.

Input Parameters: None
Output Parameters
A=OFFH if character pending, A=O if no character pending

3.3.2. CONIN

Brief Description:
CONIN returns the next byte from the console. If none is available at the time CONIN is called, CONIN will wait until a byte becomes available.

Discussion and Notes:
Depending on the IOP device driver selection, CONIN may or may not clear (set to zero) the MSB of the character returned.

Input Parameters: None
Output Parameters:
A = next byte available from the console

3.3.3. CONOUT

Brief Description:
CONOUT outputs the byte in register C to the console.

Discussion and Notes:
Depending on the selected device driver, the MSB of the byte may or may not be cleared.

Input Parameters:
C = byte to output to the console
Output Parameters: None

3.3.4. LIST

Brief Description:
LIST outputs the byte in register C to the list device.

Discussion and Notes:
Depending on the selected device driver, the MSB of the byte may or may not be cleared.

Input Parameters:
C = byte to output to the list device
Output Parameters: None

3.3.5. PUNCH

Brief Description:
PUNCH outputs the byte in register C to the punch device.

Discussion and Notes:
Depending on the selected device driver, the MSB of the byte may or may not be cleared (set to zero).

Input Parameters:
C = byte to output to the punch device
Output Parameters: None

3.3.6. READER

Brief Description:
READER returns the next byte from the reader device.

Discussion and Notes:
Depending on the IOP device driver selection, READER may or may not clear (set to zero) the MSB of the character returned.

Input Parameters: None
Output Parameters:
A = next byte available from the reader

3.3.7. LISTST

Brief Description:
LISTST returns the output status of the list device. Register A=OFFH means that the list device is ready to output another byte. Register A=0 means that it is not ready to output another byte.

Discussion and Notes:
Like the CONST routine, LISTST does not necessarily affect the Zero Flag. Only the value in the A register is affected.

Input Parameters: None
Output Parameters:
A = OFFH if the list device is ready to output another byte
A = 0 if the list device is not ready

3.4. IOP Patch

The IOP Patch routine is used to change the address of the device driver for a particular console device. There is only one IOP Patch routine:

Section	Offset Hex	Offset Dec	Mnemonic	Description of Routine
IOP Patch	21	33	PATCH	Patch Console

Brief Description:
PATCH allows the programmer to temporarily test an I/O device driver by forcing the IOP to index into a trio of CONST, CONIN, and CONOUT device drivers anywhere in memory.

Discussion and Notes:
This function is not required in an IOP and may be implemented only if desired.
On input, the HL register pair contains the address of a jump table structured as follows:

```
JMP ISTAT   ;Input status (0=no byte, 0ffh=byte)
JMP INPUT   ;Input character
JMP OUTPUT  ;Output character
```

PATCH replaces an implementer-selected console device driver with the indicated routines. Note that only one of the possible console device drivers is replaced. Hence, when this console is

selected (by NAME - see the NAMER routine documentation), the three indicated routines are called when the CONST, CONIN, and CONOUT routines of the IOP are called.

Since only one console device driver is affected by this, other console device drivers in the IOP are still available for selection.

This feature of the IOP was implemented solely for the purpose of testing a console device driver. Other applications, particularly in the area of Remote Access Systems (RASs), are possible.

Input Parameters:

HL = address of three-entry jump table

Output Parameters: None

3.5. IOP Recorder Routines

This section describes the IOP Recorder routines;

Section	Offset Hex	Offset Dec	Mnemonic	DescriVeltion of Routine
	24	36	COPEN	Open Console Recorder
IOP	27	39	CCLOSE	Close Console Recorder
Recorder	2A	42	LOPEN	Open List Recorder
	2D	45	LCLOSE	Close List Recorder

3.5.1. COPEN

Brief Description:

COPEN enables an IO Recorder for the console.

Discussion and Notes:

The nature of the recorder is up to the implementer of the IOP. The name of a file may be passed in register pair DE, where the file is named in an FCB which is structured as follows:

```
        db   disk      ;disk A = 1, current disk = 0
        db   'FILENAME'
        db   'TYP'
        db   0
        db   user      ;user 0 = 0
```

If this information is to be used by the IO Recorder, it should be copied into an FCB within the IOP for safe keeping.

Under ZRDOS-Plus, redirection of I/O to a disk file is possible since ZRDOS-Plus supports one-level reentrancy.

Two applications of the console IO Recorder are:

1. redirection of characters going to the console into a disk file
2. redirection of characters going to the console into a physical device, such as a remote computer

The STATUS routine of the IOP returns with the MSB of register A set if the IOP supports IO Recording.

The COPEN function must be associated with all console device drivers.

Input Parameters
HL = address of an FCB indicating the file (optional)
Output Parameters: None

3.5.2. CCLOSE

Brief Description:
CCLOSE terminates the console IO Recorder operation.

Discussion and Notes:
If COPEN recorded to a disk file, CCLOSE would close that disk file.
If COPEN sent output to a device, such as remote computer, CCLOSE may send a special control code to the device to instruct it to terminate recording.
The CCLOSE routine must be associated with all console device drivers.

Input Parameters: None
Output Parameters: None

3.5.3. LOPEN

Brief Description:
LOPEN opens the IO Recorder for the list device.

Discussion and Notes: See COPEN.
Input Parameters:
HL = address of FCB (optional)
Output Parameters: None

3.5.4. LCLOSE

Brief Description:
LCLOSE closes the IO Recorder for the list device.

Discussion and Notes: See COPEN and CCLOSE.
Input Parameters: None
Output Parameters: None

(PAGE INTENTIONALLY BLANK)

4. Analysis of a Sample IOP

This chapter is a running commentary on the Sample IOP whose source code is presented in Appendix B. All commentary is referenced by the line numbers given in this source code. Section 1 of Appendix B should be examined while reading through this chapter.

4.1. Analysis of the Sample IOP Source

Lines 1 to 38 comprise the front of the IOP. Note the base address of the IOP on line 6, the jump table on lines 16 to 34, and the IOP ID on line 38. When LDR loads an IOP, it checks to see that the proper number of jumps and that the IOP ID are present.

4.1.1. STATUS, SELECT, and NAMER Routines

The STATUS, SELECT, and NAMER routines are in lines 49 to 130. It is through these routines that the external environment determines the attributes of the IOP and issues device selection commands to the IOP.

STATUS returns the address of the IOP Status Table (IOPTABLE), which is shown in lines 43 to 47. This table is used to determine how many device drivers for each logical device are available and which device driver is currently selected. STATUS also returns a value in A. The MSB of A is set if the IO Recorder function of the IOP is supported. The rest of A contains a number from 1 to 127 which is used by the IOP implementer to identify the IOP. If the value of A is 0, then the IOP is considered to be non-operational by the ZCPR3 tools which address the IOP. The IOP number is not used by any ZCPR3 tools except to insure that this value is not zero.

SELECT is used to assign a device driver for a given logical device. The logical device is identified by the B register, and the desired driver is identified by the C register. B must have a value from 0 to 3 or an error condition is returned (see lines 68 to 70). B=0 selects the CON device, 1 the RDR, 2 the PUN, and 3 selects the LST device.

Lines 68 to 78 are used to locate the byte pair associated with a given logical device. Since IOPTABLE consists of two-byte entries, the value of B (which is 0 to 3) is doubled (to 0, 2, 4, 6) and used as an offset from the base address of IOPTABLE. This locates the desired byte pair. Lines 75 to 78 then compare the maximum number of devices to the requested device, insuring that the requested device number is within range. If not within range, the error routine (SELERR) is branched to. If within range, the pointer is advanced to the 2nd byte of the byte pair (line 79), and the new device is selected by storing the contents of C into the current selection byte (line 80).

NAMER is used to return a string which names a device driver and optionally provides a description. Like the SELECT input, NAMER expects a logical device number in B (0 to 3) and a device driver number in C. The indexing and error checking in NAMER (see lines 95 to 106) are similar to those in SELECT. Once certain that B and C are within range, NAMER then indexes through two address tables to locate the string.

The first table, IOPDNAMES, is addressed by the code in lines 107 to 112. The table IOPDNAMES is in lines 136 to 140. At line 107, DE contains the offset (0, 2, 4, 6) into the IOPDNAMES table for the CON, RDR, PUN, and LST devices, respectively. After line 112 is executed, HL contains the address of a table of addresses for the strings associated with a particular logical device. In this case, HL contains the value of one of these symbols:

CONNAMES RDRNAMES PUNNAMES LSTNAMES

See lines 132 to 173 for a review of these tables.

Now that the address of the desired address table is known, the code indexes into this table based on the device driver identified in the C register. Lines 113 to 121 do this indexing. Like the code in lines 96 to 112, the technique of doubling the index value (in the C register at line 113) and then adding this to HL, which contains the base address of the string address table for a particular logical device, is applied. After line 117 is executed, HL contains the address of the address of the desired string. Lines 118 to 121 simply extract the string's address and return it in HL.

Note that, for the sake of debugging, the NAMERROR routine not only returns the error code (A=0 and Zero Flag Set), but it also returns the address of an Error Message in HL.

The strings in lines 161 to 172 provide names to the various devices. Note that when no descriptions are provided (lines 161, 162, 167, 169, 171, and 172), the names are terminated by a space followed by an ending 0 (the string terminator). Also note that the names are capitalized.

4.1.2. Initialization and Device Drivers

The INIT routine in lines 174 to 181 is simple in this example. It turns off the IO Recorder flags. In a different IOP implementation, within the INIT routine would be code to configure the DARTS (7 or 8 bits, parity or none, baud rate, etc).

Lines 192 to 231 show the four basic routines for providing I/O to the CRT hardware. Note that these examples show the CRT's DART as being memory mapped (LDA and STA instructions are used instead of IN and OUT), and the data and status values are inverted (note the CMA instructions in lines 203, 211, 221, and 229). The values returned and the register conventions used are compatible with those required for the BIOS routines like CONIN and CONOUT (ie, passing output character in the C register).

Lines 234 to 275 show the four basic routines for providing I/O to the modem hardware. These examples show the modem's DART as being I/O mapped (IN and OUT are used). Similarly, lines 278

to 312 show the four basic routines for providing I/O to the printer hardware.

4.1.3. BIOS Interface Routines

Lines 314 to 375 contain the routines entered from the jump table which are indexed into from the BIOS. Namely, these routines (lines 317 to 345) are:

CONST CONIN CONOUT LIST PUNCH READER
LISTST

In all cases, the input and status routines (CONST, CONIN, READER, LISTST) return their values in the A register and require no input values, and the output routines (CONOUT, LIST, PUNCH) obtain the values to output from the C register. Consequently, since C carries the only input value, the same code (DRVRUN) can be used to process all of the BIOS entry routines if DRVRUN does not have an effect on the C register.

The routines are table-driven. DRVRUN accepts as input the address of the table for the logical device in HL and the number of the logical device in B (as for the SELECT and NAMER routines, B contains a value from 0 to 3). DRVRUN uses the value in B to obtain the number of the currently-selected device driver from the IOPTABLE (see lines 356 to 363). After line 363 is executed, B contains the number of the desired device driver. Lines 364 to 367 then double this number (in order to use it as an offset), and place it into DE. Line 368 obtains the address of the device driver address table.

After line 368 is executed, HL contains the address of one of the following tables:

TCONST TCONIN TCONOUT TLIST TREADER TPUNCH
TLISTST

DE contains the offset into the table pointed to by HL which, when added to HL (line 369), provides the address of the address of the device driver. Lines 370 to 373 obtain the address of the device driver in HL, and line 374 (PCHL) transfers control to the device driver.

Note the device driver tables in lines 376 to 416.

4.1.4. IO Recorder

The IO Recorder function is addressed in the BIOS Interface Routines. Note the call to CRECORD in line 326 and LRECORD in line 331. The code of CRECORD and LRECORD is in lines 443 to 454. Note that, in this particular implementation, CRECORD and LRECORD simply send the character to be output to the Modem if the CREC and LREC flags, respectively, are set. Remember the initialization routine, INIT, in lines 177 to 181? All INIT did was clear these flags so the IOP would not come up with recording on.

In this implementation, the IO Recorder serves to send output to the Modem as well as to the selected CONOUT or LIST device. In operation, the user is expected to have run a program

on the computer at the other end of the Modem connection which receives characters, sends a -S when its buffer is full (note that MODOUT in lines 263 to 275 pays attention to "S), writes its buffer to disk, and then sends some other character (°Q) to resume transmission through the MODOUT driver. The ZCPR3 command line "RECORD ON" calls the COPEN routine (lines 460 to 463), which simply sets the CREC flag to true. Likewise, "RECORD ON PRINTER" calls the LOPEN routine (lines 470 to 473). "RECORD OFF" calls the CCLOSE routine (lines 464 to 469) which clears the CREC flag and sends a -Z to the modem (which tells the program running there to close its file and exit). Likewise for LCLOSE (lines 474 to 479).

In looking back, I realize that MODOUT should have also checked for the output of -Z and not allowed it so the recorder on the computer tied to the modem would not accidentally terminate operation. Slight oversight.

4.1.5. Hardware Combinations

The simple device drivers in lines 192 to 313 can be easily combined into "hybrid" devices. The routines in lines 417 to 439 show such devices. CRTMODIST returns the input status from the CRT and Modem in parallel. It indicates if a character is pending on either device. CRTMODIN inputs a character from a CRT and Modem combination, where the character input comes from the CRT or the Modem, whichever receives a character first. CRTMOUOUT outputs to the CRT and Modem in parallel (note that the C register contains the character to output, and CRTOUT and MODOUT do not affect the C register in lines 225-231 and 263-275). CRTPRTOUT is similar to CRTMODOUT.

These combinations of devices are declared to the IOP through the device driver tables (lines 376 to 415). Note that the third console (selected driver is 2) is identified by CRTMODIST (line 382) for input status, CRTMODIN (line 390) for input, and CRTMODOUT (line 398) for output. This is the CRT and Modem in parallel for both input and output. In contrast, the fourth console (selected driver is 3) is identified by CRTISTAT (line 383) for input status, CRTIN (line 391) for input, and CRTPRTOUT (line 399) for output. This is the CRT input with CRT and printer output.

4.1.6. IOP Patching

The PATCH routine is the last to be discussed. It is in lines 484 to 500. Note that its sole purpose is to change the addresses for the fifth console (selected driver is 4). The addresses for PATISTAT, PATIN, and PATOUT are in lines 384, 392, and 400, respectively.

PATCH is extremely useful in debugging candidate IOP routines. The console can select the TEST device (via the SELECT routine), the test can be done, and then the console can select some other console device to restore order.

4.1.7. Adding Device Drivers

The sample IOP can be easily modified to add and remove device drivers. The code for the device driver itself must be added, and the following changes must be made:

1. modify the number of devices in IOPTABLE (lines 43-47)
2. modify the string address tables (lines 145-157)
3. modify the strings (lines 161-172)
4. modify the device driver tables (lines 376-415):
   - Console - Change TCONST, TCONIN, TCONOUT
   - Reader - Change TREADER
   - Punch - Change TPUNCH
   - List - Change MIST, TLISTST

4.2. Terminal Session

The terminal session in section 2 of Appendix B shows how the IOP which was analyzed above is assembled and prepared for use on a ZCPR3 system. The commands are discussed in order in the following paragraphs.

The command "lasm samiop.bbz" assembles SAMIOP.ASM and generates SAMIOP.HEX on drive B. The command "mload samiop" then creates SAMIOP.COM from SAMIOP.HEX (the assembler output).

SAMIOP.COM is not a true COM file. It is ORGed at some value other than 100H (see lines 6-11 in the Sample IOP listing). The command "ren sample. iop=samiop.com" creates the desired IOP file, with the file type of IOP.

SAMPLE.IOP is then loaded into the IOP buffer (and checked for validity before the load) by the command "ldr sample.iop". The IOP is now active (the INIT routine was called by LDR).

The command "dev d a" displays all devices. The device names and any descriptive comments are clearly displayed.

The command "dev c test" selects the device named TEST as the CONsole device. In SAMPLE.IOP, TEST is the device which can be patched by PATCH, and it defaults to the CRT (which is why the system is still running). The command which follows, "dev d c", displays the console device names and current selection.

Finally, "dev c crt" reassigns the device named CRT to the CONsole.

(PAGE INTENTIONALLY BLANK)

5. Extensions to the Original IOP Concept

Some proposals have been made concerning changes to the IOP standard which promise to (1) enhance the capabilities of the IOP and (2) make inter-system portability possible without the need to reassemble the IOP. This section discusses these proposals.

5.1. Internally Naming an IOP

A name may be placed within an IOP after the Z3IOP text without impacting the function of the IOP or affecting the operation of tools which interact with the IOP. This name is a string of ASCII characters terminated by a null (binary 0). Tools may read this name to determine if a specific IOP which they are designed to interact with has been loaded. The old and new IOP structures are compared:

```
Old IOP          New IOP          Comments
JMP xxx          JMP xxx            ; standard
...              ..               <-; jump
JMP xxx          JMP xxx            ; table
DB 'Z3IOP'       DB 'Z3IOP'       <-- IOP identifier
code             DB 'NAME',0      <-- CHANGE: IOP Name
```

Discussion: This extension does not impact the operation of any IOP tool which manipulates an IOP in the standard fashion (that is, only through the JMP table). No impact on any of the standard ZCPR3 tools has been identified. This extension is advantageous:

1. Special-purpose tools can be created to work with special-purpose IOPs.
2. The functionality of special-purpose IOPs can be greatly extended with the ability to adopt standards for special buffer areas which follow the name. For example, an IOP with the name of TIME may be designed where the bytes following the "DB 'TIME',0" contain the current time in some format.

Conclusion: This extension is a good idea and is adopted. No impact on any existing software is made.

Acknowledgment: Thanks to Joe Wright for this proposal.

5.2. Using Device Drivers in the BIOS

The basic philosophy of an IOP is that it removes all device drivers for the CON:, RDR:, PUN:, and LST: logical devices from within the BIOS. Situations exist, however, in which it is desirable to leave the device drivers in the BIOS and have the IOP simply act as a front-end to them. The BIOS jump table entries branch into the IOP, the IOP performs some preprocessing function (such as I/O redirection into a disk file), and then the

IOP branches back into the BIOS in order to perform the original intended function (such as console output). The device drivers for the logical devices are all or partially in the BIOS, and the IOP's purpose is to (1) intercept the BIOS calls before the device drivers in the BIOS process them and (2) perform some front-end function. Pictorially:

BIOS Call originated by code external to BIOS

V

BIOS Jump Table Vectors into IOP

V

IOP Performs Front-End Function

V

IOP Vectors Back into BIOS for Device Driver Function

A standard, transportable method of accessing the device drivers within the BIOS is required. That is, an IOP must know how to find the required device drivers once it is loaded. The following proposal is made:

1. An internal jump table must be placed within the BIOS. The addresses in these JMPs are the addresses of the device drivers residing in the BIOS. Note that the order of the JMPs is the same as the order of the JMPs following the Warm Boot JMP in the BIOS jump table. In this example, the base address of the BIOS is at the label BIOS, and the base address of the IOP buffer is at the label IOP. The address of the label IOPRET is somewhere within the BIOS after the BIOS jump table. The following figure compares the internal jump table with the BIOS jump table.

Internal Jam2 Table

```
IOPRET:
     JMP  CONST
     JMP  CONIN
     JMP  CONOUT
     JMP  LIST
     JMP  PUNCH
     JMP  READER
     JMP  LISTST
```

BIOS Jump Table

```
BIOS:
     JMP  COLD$BOOT
     JMP  WARM$BOOT
     JMP  IOP+12     ;CONST
     imp  IOP+15     ;CONIN
     imp  IOP+18     ;CONOUT
     JMP  IOP+21     ;LIST
     JMP  IOP+24     ;PUNCH
     JMP  IOP+27     ;READER
     JMP  IOP+30     ;LISTST
< more JMPs for Disk I/O >
```

2. Since the device drivers are already in the BIOS, the cold boot routine will initialize the IOP area to use these drivers. The following jump table and code is copied by the BIOS cold boot routine into the IOP buffer at the base address of the IOP buffer. A copy operation like this would be done to initialize the IOP jump table for any IOP implementation.

Initial lop Jump Table (Installed at Cold Boot)

Offset From lop	Code		Comments
	lop:		
0	XRA	A	; RETURN "NOT IMPLEMENTED" VALUE
	RET		; OCCUPIES 3 BYTES IN PLACE OF A JMP
	NOP		
3	XRA	A	; THERE ARE FOUR BEGINNING JUMPS
	RET		
	NOP		
6	XRA	A	; JMP 3
	RET		
	NOP		
9	XRA	A	; JMP 4
	RET		
	NOP		
12	JMP	IOPRET	;CONST ROUTINE WITHIN THE BIOS
15	JMP	IOPRET+3	;CONIN
18	JMP	IOPRET+6	;CONOUT
21	JMP	IOPRET+9	;LIST
24	JMP	IOPRET+12	;PUNCH
27	JMP	IOPRET+15	;READER
30	JMP	IOPRET+18	;LISTST
33	DB	0,0,0	;3 NOPS TO REPLACE REMAINING JMPS
36	DB	0,0,0	
39	DB	0,0,0	
42	DB	0,0,0	
45	XRA	A	; NOT IMPLEMENTED
	RET		
	NOP		
48	DB	'Z3IOP'	

The JMPs at offsets 12-30 branch back into the BIOS. No other lop functions are implemented.

3. During the execution of the Cold Boot routine, the address of the JMP at BIOS will be set to the value of IOPRET. Two instructions are required to do this:

```
LXI  H,IOPRET ;GET ADDRESS OF IOPRET
SHLD BIOS+1    ;STORE IT IN BIOS JMP TABLE
```

The only time that most systems transfer control to the JMP at BIOS is at power-on or reset. This change in the BIOS jump table (the JMP to COLD$BOOT now jumps to IOPRET, which in turn jumps to CONST) should have no effect on most systems. The benefit is that an lop can now look at the address at BIOS+1 and determine where the IOPRET jump table is. With this knowledge, the lop can access the routines in the IOPRET jump table for the support it requires.

lops can be moved from system to system without the need to be reassembled. An lop generation program can obtain the address of the lop buffer from the ZCPR3 Environment Descriptor and generate an lop which will execute correctly when placed in that

buffer. SPR (System Page Relocatable) formats or other relocation techniques can be used. Once the IOP begins executing, it can determine the address of the BIOS and index into the jump table at IOPRET as required.

Discussion: The only possible problem which has been identified is that the original cold boot address may be required for some purpose unique to a particular system. This proposal is reasonable and permits the transportability of IOPs at the binary level. Several IOPs (such as PKEY) have been implemented using this technique since this proposal was made, and no problems have been detected to my knowledge.

Care must be taken to use IOPs designed to obtain device driver support from the address at the BIOS cold boot routine ONLY on systems which have been installed as indicated above. If this is not done, the I/O support for a system will fail. It is suggested that the IOP generator be written to contain a test to insure that the BIOS cold boot address points to an IOPRET jump table. This test may be as simple as checking to see that there are seven JMP instructions starting at IOPRET.

Conclusion: This proposal does not impact the basic philosophy of the IOP design in any way. The original functionality of the IOP concept is retained and extended in an upward-compatible fashion. The proper safeguards should be taken in the design of the IOP generators.

This extension is approved and adopted.

Acknowledgment: Thanks to Joe Wright for this proposal.

A. References

A.1. ZCPR3 and Z-System

Echelon, Inc.
101 First Street
Los Altos, CA 94022
415/948-3820

-- **Echelon is the commercial** agent **for** ZCPR3 **and provides many services for ZCPR3 and Z-System users. ZCPR3, the Z-System, Z-System** tools, ZCPR3: The Manual, ZCPR3: The Libraries (not yet released), and many **other ZCPR3 and Z-System products** can be **purchased from** Echelon. Echelon publishes a newsletter to the ZCPR3 user community every two weeks. It monitors and supports ZCPR3 users through Remote Access Systems (Electronic Bulletin Boards) such as **Z-Node Central and other (over forty)** Z-Nodes around the world.

Z-Node Central
415/489-9005

-- Z-Node Central is the main Remote Access System used by Echelon. It supports electronic mail facilities and file transfer. **Questions regarding ZCPR3** and the **Z-System** can be submitted via electronic mail. Data on Echelon (including lists of services and prices for products) is available. A list of **people willing to spend time in helping** others in bringing up a ZCPR3 **and a Z-System is maintained here. Much of the information on Z-Node Central is distributed to all the other Z-Nodes around** the world.

BOOKS and PAMPEff.ETS

Conn, Richard. ZCPR3 **and IOPs, copyright 1985, published by Echelon,** Inc., 50 pages (free **from Z-Nodes).**

Conn, Richard. ZCPR3: The Libraries, copyright 1985, not yet published, 400+ pages (contact Echelon for price).

Conn, Richard. ZCPR3: The Manual, copyright 1985, published by Zoetrope, Inc., 351 pages, typeset, bound (contact Echelon for price).

Gaude', Frank. Z-News (Echelon Newsletter), published every two weeks by **Echelon, Inc., 4 to 12 pages (free** from Z-Nodes).

McCord, David. Z3&BYE, copyright 1985, published by Echelon, Inc., 10 pages (free from Z-Nodes).

McCord, David. ZNODE.CFG, copyright 1985, published by Echelon, Inc., 10 pages (free from Z-Nodes).

Wright, Dennis. ZRDOS Programmer's **Guide, copyright 1985, published by Echelon, Inc., 35 pages (contact Echelon for price).**

A.2. CP/M

Hogan, Thom. Osborne CP/M User Guide, copyright 1981, published by Osborne/McGraw-Hill, 200+ pages (contact Osborne/McGraw-Hill).

Johnson-Laird, Andy. The Programmer's CP/M Handbook, copyright 1983, published by Osborne/McGraw-Hi11, 400+ pages (contact Osborne/McGraw-Hill).

A.3. Other

Booch, Grady. Software Engineering with Ada, copyright 1983, published by Benjamin/Cummings, 450+ pages (contact Benjamin/Cummings).

Kernighan, Brian and Plauger, P.J. The Elements of Programming Style, 2nd Edition, copyright 1978, published by McGraw-Hill, 150+ pages (contact McGraw-Hill).

Kernighan, Brian and Plauger, P.J. Software Tools, copyright 1976, published by Addison-Wesley, 300+ pages (contact Addison-Wesley).

Leventhal, Lance. Z80 Assembly Language Programming, copyright 1979, published by Osborne/McGraw-Hill, 400+ pages (contact Osborne/McGraw-Hill).

Osborne, Adam. An Introduction to Microcomputers, Volume 1: Basic Concepts, 2nd Edition, copyright 1980, published by Osborne/McGraw-Hi11, 400+ pages (contact Osborne/McGraw-Hill).

A.4. Addresses of Some Publishers

Benjamin/Cummings Publishing Company
2727 Sand Hill Road
Menlo Park, CA 94025

Osborne/McGraw-Hill
2600 Tenth Street
Berkeley, CA 94710

B. Source Code for a Sample IOP

This Appendix presents the source code of a sample IOP which has been tested and used. The lines are numbered for reference purposes. Also, the assembly, linking, and loading (by LDR) and use of the IOP is illustrated by a terminal session.

B.1. Sample IOP Source

```
 1: ;
 2: ; SAMPLE IOP for study
 3: ; by Richard Conn
 4: ; 7/14/85
 5: ;
 6: iop     equ     OECOOH  ;base address of IOP
 7: ;
 8: ctrls   equ     'S'-'@' ;"S
 9: etrlz   equ     'Z'-'@' ;"Z
10: ;
11:         org     iop
12:
13: ;
14: ;  The IOP jump table
15: ;
16:         imp     status
17:         imp     select
18:         imp     namer
19:         imp     init
20: ;
21:         imp     const
22:         imp     conin
23:         imp     conout
24:         imp     list
25:         imp     punch
26:         imp     reader
27:         imp     listst
28: ;
29:         imp     patch
30: ;
31:         imp     copen
32:         imp     cclose
33:         imp     lopen
34:         imp     lclose
35: ;
36: ;  IOP ID (required for LDR)
37: ;
38:         db      'Z3IOP'
```

```
39:
40: ;
41: ; The following is the IOP Status Table
42: ;
43: ioptable:
44: con:    db      5,0     ;5 consoles, select console 0
45: rdr:    db      1,0     ;1 reader, select reader 0
46: pun:    db      1,0     ;1 punch, select punch 0
47: lst:    db      2,0     ;2 lists, select list 0
48:
49: ;
50: ; The status routine
51: ;       Return the address of the IOP Status Table in HL
52: ;       Return the IOP number in A
53: ;       This IOP supports recording, so set MSB of A
54: ;
55: status:
56:         lxi     h,ioptable      ;pointer to table
57:         mvi     a,82h           ;IO Recorder supported, IOP 2
58:         ora     a               ;set NZ flag
59:         ret
60:
61: ;
62: ; The select routine
63: ;       On input, B=logical device and C is driver
64: ;       On output, A=0 and zero flag set if error
65: ;
66: select:
67:         lxi     h,ioptable      ;pt to IOP table
68:         mov     a,b             ;double B so offset is 0,2,4,6
69:         cpi     4               ;make sure in range 0-3
70:         jnc     selerr
71:         add     b
72:         mov     e,a             ;DE = offset
73:         mvi     d,0
74:         dad     d               ;HL now points to device in IOP
75:         mov     a,m             ;get max number of devices
76:         cmp     c               ;check for driver error
77:         jZ      selerr          ;error if C = count
78:         jc      selerr          ;error if C > count
79:         inx     h               ;point to selected device byte
80:         mov     m,c             ;select the device
81:         mvi     a,0ffh          ;set OK return code
82:         ora     a
83:         ret
84: selerr:
85:         xra     a               ;set error return code
86:         ret
87:
```

```
 88: ;
 89: ; The Namer Routine
 90: ;       On input, B = logical device and C = driver
 91: ;       On output, HL = address of name string
 92: ;       On output, A=0 and Zero Flag Set if error
 93: ;
 94: namer:
 95:         lxi     h,ioptable      ;check to see that C is
 96:         mov     a,b             ; in range ... begin by
 97:         cpi     4               ; doubling B to 0,2,4,6
 98:         jnc     namerror        ; after making sure in
 99:         add     b               ; range 0-3
100:         mov     e,a             ;add offset to HL
101:         mvi     d,0
102:         dad     d               ;HL now points to IOP Table
103:         mov     a,m             ;get max device count
104:         cmp     c
105:         jz      namerror        ;error if C = count
106:         jc      namerror        ;error if C > count
107:         lxi     h,iopdnames     ;get address of logical
108:         dad     d               ; name table
109:         mov     em
110:         inx     h
111:         mov     d,m
112:         xchg                    ;HL now points to logical
113:         mov     a,c             ; name table - double C
114:         add     c               ; to get device driver name
115:         mov     e,a
116:         mvi     d,0             ;DE = offset
117:         dad     d               ;HL now points to driver name
118:         mov     em              ; address - get string address
119:         inx     h               ; in DE
120:         mov     d,m
121:         xchg                    ;HL now has string name address
122:         mvi     a,0ffh          ;set no error
123:         ora     a
124:         ret
125: namerror:
126:         lxi     h,errmsg        ;pt to some message
127:         xra     a               ;set error code
128:         ret
129: errmsg:
130:         db      'Name Error',0
131:
```

```
132: ;
133: ; This table gives the addresses of the address
134: ; tables for each of the logical devices
135: ;
136: iopdnames:
137:          dw      connames
138:          dw      rdrnames
139:          dw      punnames
140:          dw      lstnames
141: ;
142: ; These tables give the addresses of each of the
143: ; logical device name strings
144: ;
145: connames:
146:          dw      connl           ;there are 5 consoles
147:          dw      conn2           ; (see IOPTABLE above)
148:          dw      conn3
149:          dw      conn4
150:          dw      conn5
151: rdrnames:
152:          dw      rdrnl           ;there is 1 reader
153: punnames:
154:          dw      punnl           ;there is 1 punch
155: lstnames:
156:          dw      listnl          ;there are 2 lists
157:          dw      listn2
158: ;
159: ; These are the actual text strings returned by NAMER
160: ;
161: connl:   db      'CRT ',0
162: conn2:   db      'MODEM ',0
163: conn3:   db      'CRTMOD CRT and Modem in Parallel',0
164: conn4:   db      'CRTPRT CRT in and CRT/Printer out',0
165: conn5:   db      'TEST CRT by default',0
166: ;
167: rdrnl:   db      'MODEM ',0
168: ;
169: punnl:   db      'MODEM ',0
170: ;
171: listnl: db      'PRINTER ',0
172: listn2: db      'MODEM ',0
173:
```

```
174:
175:   This routine initializes the devices in the IOP
176:
177: init:
178:         mvi     a,O      ;set no IO Recording active
179:         sta     crec     ;console off
180:         sta     lrec     ;list off
181:         ret
182:
183:
184:    This system has three pieces of hardware connected:
185:         1. a CRT
186:         2. a modem
187:         3. a printer
188:    All devices are hypothetical
189:    The following are the simple device drivers for them
190:
191:
192:
193:    1. CRT
194:
195: crtdata equ     OF800H+3F8H      ;CRT data port
196: crtstat equ     OF800H+3F9H      ;CRT status port
197: crtrda  equ     4        ;RDA bit
198: crttbe  equ     8        ;TBE bit
199:
200:    Return input status in A (A=0 means no char available)
201: crtistat:
202:         lda     crtstat  ;check input status
203:         cma              ;status is inverted
204:         ani     crtrda   ;mask for RDA
205:         rz               ;0 if no char pending
206:         mvi     a, Offh  ;return OFFH if char pending
207:         ret
208:    Return output status in A (A=0 means not ready for output)
209: crtostat:
210:         lda     crtstat  ;check output status
211:         cma              ;status is inverted
212:         ani     crttbe   ;mask for TBE
213:         rz               ;0 if not ready
214:         mvi     a, Offh   OFFH if ready
215:         ret
216:    Return input byte in A (A=byte)
217: crtin:
218:         call    crtistat         ;wait for input
219:         jz      crtin
220:         Ida     crtdata ;get byte
221:         cma              ;data is inverted
222:         ani     7fh      ;mask
223:         ret
```

```
224:    Output byte in C to device
225: crtout:
226:         call    crtostat        ;wait for ready
227:         jz      crtout
228:         mov     a,c     ;get char from C
229:         cma             ;invert data
230:         sta     crtdata ;put byte
231:         ret
232:
233:
234:
235:    2. Modem
236:
237: moddata equ     80H     ;Modem data port
238: modstat equ     81H     ;Modem status port
239: modrda  equ     2       ;RDA bit
240: modtbe equ     1       ;TBE bit
241:
242:    Return input status in A (A=0 means no char available)
243: modistat:
244:         in      modstat ;check input status
245:         ani     modrda  ;mask for RDA
246:         rz              ;0 if no char pending
247:         mvi     a, 0ffh ;return OFFH if char pending
248:         ret
249:    Return output status in A (A=0 means not ready for output)
250: modostat:
251:         in      modstat ;check output status
252:         ani     modtbe  ;mask for TBE
253:         rz               0 if not ready
254:         mvi     a, 0ffh ;OFFH if ready
255:         ret
256:    Return input byte in A (A=byte)
257: modin:
258:         call    modistat        ;wait for input
259:         jz      modin
260:         in      moddata ;get byte
261:         ret
262:    Output byte in C to device with simple XON/XOFF Processing
263: modout:
264:         call    modistat        ;see if char pending
265:         jz      modoutl         ;continue if not
266:         call    modin           ;get char
267:         cpi     ctrls           ;see if -S
268:         jnz     modoutl         ;continue if not
269:         call    modin           ;wait for any next char
270: modoutl:
271:         call    modostat        ;wait for ready
272:         jz      modout
273:         mov     a,c     ;get char from C
274:         out     moddata ;put byte
275:         ret
```

```
276:
277:
278: ;
279: ; 3. Printer
280: ;
281: prtdata equ     20H     ;Printer data port
282: prtstat equ     25H     ;Printer status port
283: prtrda equ      1       ;RDA bit
284: prttbe equ      20H     ;TBE bit
285:
286: ; Return input status in A (A=0 means no char available)
287: prtistat:
288:          in      prtstat ;check input status
289:          ani     prtrda ;mask for RDA
290:          rz              ;0 if no char pending
291:          mvi     a,Offh ;return OFFH if char pending
292:          ret
293: ; Return output status in A (A=0 means not ready for output)
294: prtostat:
295:          in      prtstat ;check output status
296:          ani     prttbe ;mask for TBE
297:          rz              ;0 if not ready
298:          mvi     a,Offh ;OFFH if ready
299:          ret
300: ; Return input byte in A (A=byte)
301: prtin:
302:          call    prtistat        ;wait for input
303:          jz      prtin
304:          in      prtdata ;get byte
305:          ret
306: ; Output byte in C to device
307: prtout:
308:          call    prtostat        ;wait for ready
309:          jz      prtout
310:          mov     a,c     ;get char from C
311:          out     prtdata ;put byte
312:          ret
313:
```

```
314: ;
315: ; The following are the device selection routines
316: ;
317: const:
318:          lxi     h,tconst        ;point to driver table
319:          mvi     b,0             ;CON device
320:          imp     drvrun          ;run driver
321: conin:
322:          lxi     h,tconin
323:          mvi     b,0
324:          imp     drvrun
325: conout:
326:          call    crecord         ;send char to recorder if on
327:          lxi     h,tconout
328:          mvi     b,0
329:          imp     drvrun
330: list:
331:          call    lrecord         ;send char to recorder if on
332:          lxi     h,tlist
333:          mvi     b,3             ;LST device
334:          imp     drvrun
335: punch:
336:          lxi     h,tpunch
337:          mvi     b,2             ;PUN device
338:          imp     drvrun
339: reader:
340:          lxi     h,treader
341:          mvi     b,l             ;RDR device
342:          imp     drvrun
343: listst:
344:          lxi     h,tliststt
345:          mvi     b,3             ;LST device
```

```
346: ;
347: ;  The following routine selects the desired driver
348: ;       On input, B=logical device number
349: ;       IOPTABLE is used to find the current driver
350: ;       On input, HL=address of driver table
351: ;       Driver table contains address of all drivers
352: ;               which can be selected
353: ;
354: drvrun:
355:         push    h               ;save ptr to driver table
356:         lxi     h,ioptable      ;get selected driver number
357:         mov     a,b             ;double B for offset
358:         add     b
359:         mov     e,a
360:         mvi     d,0
361:         dad     d               ;HL pts to IOPTABLE entry
362:         inx     h               ;HL pts to selected driver
363:         mov     b,m             ;get selected driver in B
364:         mov     a,b             ; (C not used because it can
365:         add     b               ; contain a character if output
366:         mov     e,a             ; driver is being called)
367:         mvi     d,0
368:         pop     h               ;HL pts to driver table
369:         dad     d               ;HL pts to desired driver address
370:         mov     e,m
371:         inx     h
372:         mov     d,m
373:         xchg                    ;HL pts to driver
374:         pchl                    ;run the driver
375:
```

```
376: ;
377: ; These are the device driver tables
378: ;
379: tconst:
380:          dw      crtistat        ;selected driver 0 is CRT
381:          dw      modistat        ;selected driver 1 is Modem
382:          dw      crtmodist       ;selected driver 2 is CRT/Modem
383:          dw      crtistat        ;selected driver 3 'is CRT in,
383:                                  ; Printer out
384: patistat:                        ;patch point for PATCH routine
385:          dw      crtistat        ;selected driver 4 is CRT
386: ;
387: tconin:
388:          dw      crtin
389:          dw      modin
390:          dw      crtmodin
391:          dw      crtin
392: patin:                           ;patch point for PATCH routine
393:          dw      crtin
394: ;
395: tconout:
396:          dw      rrtout
397:          dw      modout
398:          dw      crtmodout
399:          dw      crtprtout
400: patout:                          ;patch point for PATCH routine
401:          dw      crtout
402: ;
403: tlist:
404:          dw      patout          ;selected driver 0 is Printer
405:          dw      modout          ;selected driver 1 is Modem
406: ;
407: treader:
408:          dw      modin           ;selected driver 0 is Modem
409: ;
410: tpunch:
411:          dw      modout          ;selected driver 0 is Modem
412: ;
413: tliststst:
414:          dw      prtostat        ;selected driver 0 is Printer
415:          dw      modostat        ;selected driver 1 is Modem
416:
```

```
417: ;
418: ; This is the driver set for the combination CRT/Modem Device
419: ; and the combination CRT/Printer Output Device
420: ;
421: crtmodist:
422:          call    crtistat        ;see if char available on CRT
423:          rnz                     ;return if so
424:          call    modistat        ;see if char available on Modem
425:          ret
426: crtmodin:
427:          call    crtistat        ;look for CRT char
428:          jnz     crtin           ;get char from CRT
429:          call    modistat        ;look for Modem char
430:          jnz     modin           ;get char from Modem
431:          imp     crtmodin        ;continue until CRT or Modem
431:                                  ; gives char
432: crtmodout:
433:          call    crtout          ;send to CRT
434:          call    modout          ;send to Modem
435:          ret
436: crtprtout:
437:          call    crtout          ;send to CRT
438:          call    prtout          ;send to Printer
439:          ret
440: ;
441: ; These are the drivers for the recorder output device
442: ;
443: crecord:
444:          lda     crec            ;check flag
445:          ora     a               ;0 means not recording
446:          rz
447:          call    modout          ;send char to modem to record
448:          ret
449: lrecord:
450:          lda     lrec            ;check flag
451:          ora     a               ;0 means not recording
452:          rz
453:          call    modout          ;send char to modem to record
454:          ret
```

```
456: ; These are the routines which enable device recording
457: ; For this IOP, Console and Printer recording amounts to
458: ;       sending characters to the modem
459: ;
460: copen:
461:          mvi     a,Offh          ;set flag
462:          sta     crec
463:          ret
464: cclose:
465:          mvi     a,0             ;clear flag
466:          sta     crec
467:          mvi     c,ctrlz         ;send "Z to modem
468:          call    modout
469:          ret
470: lopen:
471:          mvi     a,Offh          ;set flag
472:          sta     lrec
473:          ret
474: lclose:
475:          mvi     a,0             ;clear flag
476:          sta     lrec
477:          mvi     c,ctrlz         ;send "Z to modem
478:          call    modout
479:          ret
480: ;
481: crec:    ds      1               ;flag buffer
482: lrec:    ds      1               ;flag buffer
483:
484: ;
485: ; This is the patch routine
486: ; It sets the 5th device driver (driver select 4) to
487: ;       the drivers whose jump table is pointed to by
488: ;       HL; HL points to a table like the following:
489: ;               imp     ISTAT
490: ;               imp     INPUT
491: ;               imp     OUTPUT
492: ;
493: patch:
494:          shld    patistat        ;set address of input status
495:          lxi     d,3             ;offset of 3
496:          dad     d
497:          shld    patin           ;set address of input char
498:          dad     d
499:          shld    patout          ;set address of output char
500:          ret
501:
502:          end
```

B.2. Terminal Session

```
B1:ASM>lasm samiop.bbz
LINKASM AS OF 7/06/81

SAMIOP
SAMIOP
EEAD
006H use factor
502 input lines read
End of assembly

B1:ASM>mload samiop
MLOAD ver. 2.4   Copyright (C) 1983, 1984, 1985
by NightOwl Software, Inc.
Loaded 683 bytes (02ABH) to file B1-.SAMIOP.COM
Start address: ECOOH Ending address: EEACH Bias: OOOOH
Saved image size: 768 bytes (0300H, - 6 records)

++ Warning: program origin NOT at 100H ++

B1:ASM>ren sample.iop=samiop.com

B1:ASM>ldr sample.iop
ZCPR3 LDR, Version 1.3
 Loading SAMPLE.IOP

B1:ASM>dev d a

CON: Devices --
    TEST      - CRT by default
    CRTPRT    - CRT in and CRT/Printer out
    CRTMOD    - CRT and Modem in Parallel
    MODEM     -
    CRT       -
 Assignment is CRT
RDR: Devices --
    MODEM     -
 Assignment is MODEM

Strike Any Key PUN: Devices --
    MODEM     -
 Assignment is MODEM
LST: Devices --
    MODEM     -
    PRINTER   -
 Assignment is PRINTER

B1:ASM>dev c test
 CON: Assignment is TEST
```

```
B1:ASM>dev d c

CON: Devices --
    TEST     - CRT by default
    CRTPRT   - CRT in and CRT/Printer out
    CRTMOD   - CRT and Modem in Parallel
    MODEM    -
    CRT      -
 Assignment is TEST

Bl:ASM>dev c crt
 CON: Assignment is CRT
```

I N D E X