MCM

World's first
portable full-service
APL computer.
World wide
distribution/service.

MICRO COMPUTER MACHINES, INC

World's first
ortable full-service
PL computer.
World wide
distribution/service.

MICRO COMPUTER MACHINES, INC.
133 Dalton Street, Kingston, Ontario, Canada K7L-4W2
(613) 544-9860 ▪ TWX: 610-523-1155

SCI - 1200
COMMUNICATIONS SUB-SYSTEM

INSTALLATION INSTRUCTIONS
and
REFERENCE MANUAL

SCI - 1200

SERIAL COMMUNICATIONS SUB-SYSTEM

INSTALLATION INSTRUCTIONS

MODEM INTERFACE SIGNALS

MCM/70 SERIAL INTERFACE (EIA) - SCI-1200

INSTALLATION INSTRUCTIONS

1. Plug the unit into a 110V 60Hz. A.C. power source. Observe the red indicator near the power switch. If the indicator is off, push the power switch button. The indicator should now be lit, indicating that power is being supplied to the unit. If the indicator does not light, check the fuse. A blown fuse usually requires the unit to be returned for service.

2. Plug the MCM/700 into either the same power source or the service outlet on the SCI-1200 (This is an unswitched, unfused outlet). Plug the Omniport cable into the MCM/700 Omniport connector and one of the SCI-1200 Omniport connectors. (Note - this cable is keyed. Do not attempt to force it in the wrong way.)

3. Observe the setting of the address select switches behind the holes marked "ADD SEL". These are normally factory set at "02). The setting can be changed with a small screwdriver.

4. Start the MCM/700. Select the SCI-1200 for output using the address determined in Step 3:

   ⎕*OU* 2

   The response returned should be 2 193 8 or 2 225 8.

   This indicates that the interface whose address is 2 has been selected, that it is an SCI-1200, and that it is ready but not yet connected to any device.

5. From this point on, the procedure is different for each device to which the SCI-1200 is to be connected.

PRINTERS AND TERMINALS

A1. Place the "PROMPT" switch in the "ON" position, and on the MCM/700 type:

   ⎕*OU* 10

   This should respond 2 225 8
   (If it responds 0 0 0, repeat Step 4).

A2. Ensure that nothing is connected to the "MODEM" connector on the SCI-1200, and connect the terminal (or printer) to the small adapter box supplied with the SCI-1200.

   Place the "DTR" (Data Terminal Ready) switch in the "X" position (In this position, DTR is supplied by the terminal). Plug the adapter box into the "TERMINAL" connector on the SCI-1200.

Turn on the terminal. Obtain the device status with:

⎕*OU* 2 2

This should respond with 2 225 18. If it responds 2 225 8, move the "DTR" switch to the "1" position (This forces DTR to be true, since it has been determined that the terminal does not supply the signal).

A3. If the terminal is a 300 Baud ASCII device with APL typewriter pairing character codes, it should now be ready to accept output. To check this, type:

⎕←⍳9

The terminal should then display the numbers 1 through 9. If it does not, special support is required. Contact your local MCM representative for details.

A4. If the terminal has a keyboard, it can also be used for input to the MCM/700. To do this, type:

⎕*IN* 2

(For the 2, substitute whatever device address you are using. See step 3) The response should be the same as that obtained in step A2. Then type:

⍞ '*HELLO*'

The terminal should print "HELLO", and ring its bell, indicating that it is awaiting input. If you then type:

12345 $\boxed{\text{Return}}$

on the terminal keyboard, the terminal should echo the "12345", do a Return and Linefeed, and the MCM/700 should display

*HELLO*12345

MODEMS AND ACOUSTIC COUPLERS

B1. Place the "PROMPT" switch in the "OFF" position, and on the MCM/700 type:

⎕*OU* ⍳0

This should respond 2 193 18 (If it responds 0 0 0, repeat step 4).

B2. Ensure that nothing is connected to the "TERMINAL" connector on the SCI-1200, and plug the modem (or acoustic coupler) into the "MODEM" connector. If the modem has its onw line cord, plug it into its power source and turn it on. On the MCM/700, type:

⎕*OU* ⍳0

The response should be 2 193 24, indicating that the modem is ready but has not detected a line carrier.

B3. Ensure that if the modem has "ECHO", "COPY", or "ANSWER" switches, they are all off. Make the telephone or hard-wire connection. This should result in an audible tone from an

MCM-supplied modem. Type

```
⎕OU ⍳0∘⎕IN ⍳0
```

The response should now be 2 193 18, indicating that the modem has a carrier and is ready to transfer data.
(If the last number in the response is odd, the system to which you have just connected has tried to send data to the SCI-1200, which the MCM/700 has ignored).

B4. If the system to which you are connected uses a 300 Baud ASCII/APL typewriter-pairing overlay character code, you are now ready to talk to it. Type:

```
⍞''∘⎕←'-- (something the system expects)-- '
```

This should transmit the quoted character string to the external system, and display the first line of its response on the MCM/700. If it does not, special support is required. Contact your local MCM representative for details. (Note - if the external system does not respond, the MCM/700 will wait indefinitely for it to do so. Type CONTROL/"←" or CONTROL/SHIFT/"→" to escape from this situation.)

SCI-1200

MODEM INTERFACE SIGNALS

(25 PIN MALE CONNECTOR)

PIN NO.	SIGNAL	FUNCTION
1	Protective Ground (AA)	Chassis Ground
*2	Transmitted Data (BA)	Digital Data Output to modem from SCI-1200
*3	Received Data (BB)	Digital Input from modem to SCI-1200
*4	Request to Send (CA)	Control output from SCI-1200 to modem; on when SCI-1200 is ready to transmit data
*5	Clear to Send (CB)	Control input to SCI-1200 signifying modem is ready to transmit data to remote modem
*6	Data Set Ready (CC)	Control input to SCI-1200 which indicates that data set is ready
7	Signal Ground (AB)	Common signal and power ground return
8	Data Carrier Detected (CF)	Control input to SCI-1200 which indicates receipt of valid carrier by modem
9	Positive Voltage	+11 volts @ 60 mA
10	Negative Voltage	-11 volts @ 60 mA
20	Data Terminal Ready (CD)	Control output indicates SCI-1200 is ready to operate

SIGNAL LEVELS

* These are EIA RS-232 C signals with the following characteristics:

Inputs:	Mark -3 to -25 volts
	Space +3 to +25 volts
Outputs:	Mark -9 with 2K load
	Space +9 with 2K load

SCI-1200

TERMINAL INTERFACE SIGNALS

(25 PIN FEMALE CONNECTOR)

PIN NO.	SIGNAL	FUNCTION
1	Protective Ground (AA)	Chassis Ground
*2	Transmitted Data (BA)	Digital Input to SCI-1200
*3	Received Data (BB)	Digital Output From SCI-1200
*5	Clear to Send (CB)	Control Outputs from SCI-1200 Which indicated it is ready to send and receive data
*6	Data Set Ready (CC)	
*8	Data Carrier Detect (CF)	
7	Signal Ground (AB)	Common Signal and power ground return
**12	Aux Data Terminal Ready	Auxiliary Control input to SCI-1200 to indicate terminal is ready
14	Received Data (TTY)	Digital input data from TTY (contact closure)
16	Transmitted Data (TTY)	Digital output data to TTY (20 mA Current Loop)
18	Received Data Return (TTY)	Common return for received data
**19	TTY Enable	Control input to SCI-1200 which enables TTY
*20	Data Terminal Ready (CD)	Control input to SCI-1200 which indicates terminal is ready
*21	Monitor Data	Digital output data from SCI-1200 which monitors both transmitted and received data
25	Transmitted Data Return (TTY)	Common Return for transmitted data (20 mA Current Loop)

SIGNAL LEVELS

* These are EIA RS-232C signals with the following characteristics:

Inputs:	Mark	-3 to -25 volts
	Space	+3 to +25 volts
Outputs:	Mark	-9 with 2K load
	Space	+9 with 2K load

** These lines are enabled when grounded (connected to pin 7)

EIA INTERFACE

EIA INTERFACE

This interface, consisting of a device driver in the communications subsystem and an omniport EIA board, allows the user to communicate with almost any asynchronous device which is compatible with the EIA RS-232-C specification. A teletype current loop is also provided.

The standard tables provided for the system allows communication with a 300 baud ASCII half-duplex terminal using the APL/ASCII typewriter pairing overlay character set. In order to divert output to such a device, the user need simply connect the device to the system and type:

```
⎕OU ⎕YA 65 95
```

Or for input:

```
⎕OU ⎕YA 129 159
```

The following documentation describes handling of EIA devices in a non-standard fashion:

DIRECT ACCESS:

Before any communication can occur, the omniport EIA board must be informed of the protocol to be used for data transfer. This is achieved with the command byte. The least significant 6 bits of the command byte are control data. The most significant two bits of the command byte indicate the usage of the control data.

0(00)	- General control
1(01)	- Serial word control
2(10)	- Least significant data rate bits
3(11)	- Most significant data rate bits

The device status byte is defined as follows:

Bit 7	- Read Overrun
Bit 6	- Read Parity Error
Bit 5	- Read Framing Error
Bit 4	- Device Powered and Ready
Bit 3	- Receive Carrier Off
Bit 2	- Break Received
Bit 1	- Transmit Buffer Empty
Bit 0	- Receive Data Available

Bits 3 and 4 are static conditions. All other bits are set and reset by events.

Bit 0 is set when a complete serial data word has been received, and reset when the resulting data byte is read.

Bit 1 is set when the transmit buffer is ready to accept a byte for transmission, and reset when a data byte is output to it. If the interface is connected to a modem, this bit is forced low wherever the CLEAR TO SEND line from the modem is false.

Bit 2 is set when the RECEIVE DATA line stays in the SPACE condition for a time determined by a setting on the board. This time is factory set to 150 msec., and may be set anywhere in the range 15 to 200 msec.

Bit 3 is true when the EIA interface is connected to a modem, and the modem does not detect a receive carrier. When the interface is connected to a terminal, this bit is tied to bit 4.

Bit 4 is true whenever the interface is powered and is receiving either a DATA SET READY (from a modem) or DATA TERMINAL READY (from a terminal).

Bit 5 is set whenever the line is in a SPACE condition at the STOP position of a received serial data word.

Bit 6 is set whenever the parity of the received serial data word is incorrect.

Bit 7 is set whenever a serial data word is received with Bit 0 true. If this happens, the data in the buffer is replaced with the new data.

Command 0 - General Control:

Only the least significant 3 data bits are defined. They are used as follows:

Bit 0 - Master Reset. Resets all latches.

Bit 1 - Event reset. Resets status bits 0, 1, 2, 5, 6, and 7.

Bit 2 - Transmit Break. This sets the TRANSMIT DATA line in a SPACE condition for a time determined by a setting on the board. This time is factory set to 200 msec., and may be varied from 15 to 200 msec.

Command 1 - Serial Word Control:

The least significant 6 data bits are used as follows:

Bits 1 & 0	:	The number of data bits in a serial word, minus 5. Thus a 6-bit data word is indicated by 01.
Bit 2	:	If 1, indicates even parity.
Bit 3	:	If 1, indicates no parity bit (in this case, bit 2 is ignored)
Bit 4	:	If 1, indicates two stop bits (one is normal).
Bit 5	:	If 1, indicates that the REQUEST TO SEND line to the device is to be set true.

Commands 2 & 3 - Data Rate:

The data rate is determined from the Baud Rate B as follows:

```
R ← ¯1+⌊+.5+25000÷B
```

This is then represented as an 8 bit binary word. The least significant 4 bits are output with command 2, and the most significant 4 bits are output with command 3:

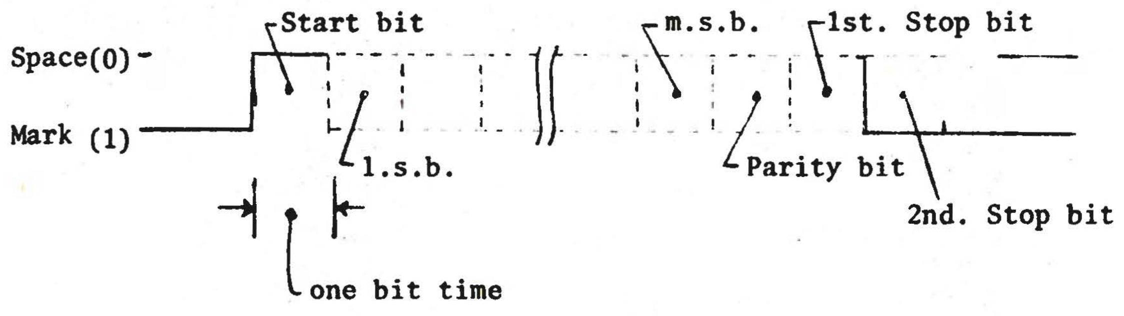
```
X ← 16 16 ⊤ R

⎕OU ADDR, 8 64 ⊥ 2,¯1↑X

⎕OU ADDR, 8 64 ⊥ 3,1↑X
```

DATA TRANSFER:

The most recently received data byte may be read with ⎕BI⍳0 (or ⎕BO⍳0).

A data byte *B* may be transmitted with ⎕BO *B* (or ⎕BI *B*).

Below is a time diagram of a serial data word. The least significant data bit is the first bit in the string. The parity bit is last.



<u>DEVICE TABLES</u>:

Direct access to an EIA device is rather clumsy and usually unnecessary. Considerable flexibility may be obtained through modification of the device tables.

Devices with an APL Character Set whose codes differ from the standard set can be supported simply by specifying the appropriate codes in the INPUT and OUTPUT translate segments.

Devices with non-APL Character Sets (such as standard ASCII) can be supported by representing characters which the device does not have by mnemonics. Even a Teletype 33 or EIA compatible card reader can be supported in this fashion.

Protocol differences are handled via the CONTROL segment. The function of each row of the control table is described below in detail.

<u>Row 0 - General Information</u>:

This row has four elements, the first of which was described in the general documentation under Device Control (page 1.6). The elements are:

XX,*MISC*,*WORD*,*RATE*

<u>*RATE*</u>: The data rate for the device is set by RATE. The value of this element is identical to *R* as discussed under Data Rate in the discussion of Direct Access (page 3.3). The default is 82 (300 baud).

<u>*WORD*</u>: The least significant 5 bits of the binary word represented by WORD are identical to the corresponding bits discussed above under Serial Word Control. If bit 6 is 1, parity errors are ignored on input. Otherwise, they are translated into a Bad Character. The default value is 70 (Ignore parity, even parity, 7 data bits).

<u>*MISC*</u>: The 8 bits of the binary word represented by MISC are used as follows:

Bit 7 - If 1, echo received data. (used for full duplex terminals) <u>Default = 0</u>.

Bit 6 - If 1, this device is a shifting device (see Shift Control under rows 7 and 8) <u>Default = 0</u>.

Bit 5 - If 1, Break is enabled on output. <u>Default = 1</u>.

Bit 4 - If 1, and Bit 5 = 1, Break on input is treated as an EOT (see EOT under row 9). <u>Default = 1</u>.

Bits 3 through 0 - After an EOT (see row 10) is transmitted to the device, the first *N* characters received are ignored, where *N* is the binary word represented by these 4 bits. This is used to dump acknowledgment of the transmitted EOT which may precede input text. Default = 0.

Row 1 - Continuation:

This was discussed under Device Control in the general documentation (page 1.6). Its form is:

WIDTH, *CHAR*, *INDENT*

The default value is 72, 130 (overstrike), 6.

Row 2 - Newline:

This row has four elements as follows:

TIME_OUT, *IDLES*, *CODE*, *CODE*

At the end of each physical line output, the two CODES are transmitted. If a code value is 128, its transmission is suppressed. Following the outputs, the number of IDLE characters (see row 4) determined by the following algorithm are transmitted:

IDLE_COUNT←1+⌊*CARRIAGE_POSITION*÷*IDLES*

If IDLES = 0, no Idles are transmitted. The default row is 0, 0, 13 (carriage return), 10 (linefeed). A newline received during input terminates the input operation. The default input code is 13 (carriage return). If TIME_OUT = 0, and an interval longer than TIME_OUT seconds elapses between the receipt of any two characters during an input operation, the receipt of a newline is simulated.

Row 3 - Backspace:

This row has only one element, the value of which is the code transmitted for backspace. If the code value is 128, transmission is suppressed. (Default = 8)

Row 4 - Idle:

Similar to Backspace, above, but used when idle characters are required. (Default = 0). Idle Characters received on input are ignored.

Row 5 - Carriage Return:

(Usage not yet defined)

Row 6 - Form Feed:

This row has four elements, the first two of which were discussed under Device Control in the general documentation (page 1.6). The form is:

LIMIT, *COUNT*, *CODE*, *CODE*

When the page limit is reached, the two CODEs are transmitted. If a code value is 128, its transmission is suppressed. The transmission is repeated COUNT times. The default row is 0 (paging off), 0, 10 (linefeed), 0 (idle). Form Feed is meaningless on input.

Row 7 & 8 - Shift Down & Up:

These rows have one element each. Devices with fewer than 7 data bits usually expand their character set with shift control. In effect, bit 6 of the data byte is set by Shift Up, and reset by Shift Down. On output, if row 0 indicates that this is a shifting device, a data byte which has bit 6 different from the current setting causes a shift code to be transmitted. If the code given is 128, transmission is suppressed (Default = 128).

Row 9 - End of Transmission:

Prior to each input operation, after the prompt (if any) has been output, an EOT is issued to the device. This "turns the line around", informing the device that input is requested. If the device precedes its input data with an EOT acknowledgment, row 0 specifies the number of characters in the acknowledgment, which are ignored.
This control row consists of two elements:

CODE, *CODE*

When an EOT is issued, the two CODES are transmitted. If a code is 128, its transmission is suppressed.

The default row is 0, 7 (idle, bell).

An EOT received during an input operation initiates an editing operation. If the device cannot accept prompts, all previous input for the current physical line is discarded. If the device can accept prompts, previous input at and to the rignt of the current position is discarded.

The remainder of the current physical line is re-transmitted to the device, preceded by a BOT and a newline, and followed by an EOT. The input operation is then resumed.

The default input code is 4 (EOT).

Row 10 - Beginning of Transmission:

At the end of each input operation, a BOT is issued to the device. This again "turns the line around", informing the device that output follows.

The form of this row is identical to that for row 9.

The default row is 0, (idle).

BOT on input is meaningless.

INPUT INTERRUPTION:

As with all operations, input may be interrupted with a Hard Interrupt (Control '→' on the main keyboard). The operation may also be terminated, however, with a Soft Interrupt or Attention (Control '←'). This causes a newline input to be simulated, and processing proceeds as it would if an actual newline were input from the device.

2. SYSTEM FUNCTIONS

SYSTEM FUNCTIONS

The Communications SubSystem contains the following system functions:

CHARACTER I/O:

⎕ ← *X*	Output
AV ← $\{SI\}$ ⍞ *AV*	Input

OMNIPORT ACCESS:

NV ← ⎕*IN NV*	Set SOURCE
NV ← ⎕*OU NV*	Set SINK
NS ← ⎕*BI* $\left[\begin{matrix}\iota 0\\ NS\end{matrix}\right]$	Direct SOURCE $\left[\begin{matrix}\text{input}\\ \text{output}\end{matrix}\right]$
NS ← ⎕*BO* $\left[\begin{matrix}\iota 0\\ NS\end{matrix}\right]$	Direct SINK $\left[\begin{matrix}\text{input}\\ \text{output}\end{matrix}\right]$
NS ← ⎕*YA NV*	Locate device ADDRESS

TABLE ACCESS:

YD ← $\{YD\}$ ⎕*YI*[*SI*]*YR*	Access SOURCE table segment
YD ← $\{YD\}$ ⎕*YO*[*SI*]*YR*	Access SINK table segment
NS ← *TS* ⎕*YR NAME*	READ Complete Table
NS ← *TS* ⎕*YW NAME*	WRITE Complete Table
TS ⎕*YX* ⍳0	EXPUNGE Complete Table

MISCELLANEOUS:

X ← ⎕*Y X*	Convert Character/Numeric
NS ← ⎕*DL NS*	Delay Seconds
NV ← ⎕*PC NV*	Read and Set Print Counters

Where:

X may be of any shape or type

NS is a numeric scalar

NV is a numeric vector

AV is a character vector

SI is a scalar index

YR is a table segment reference array

YD is a table segment data array

TS is '*I*' or '*O*'

NAME is a character vector which represents the name of APL variable.

$\begin{bmatrix} A \\ B \end{bmatrix}$ indicates two alternatives (either *A* or *B*).

{...} indicates that the enclosed item may or may not be present.

CHARACTER OUTPUT:

Syntax	:	{*R*←} ⎕ ← *B*
Domain	:	*B* may be character or numeric
Conformability	:	None
Result	:	If a result is requested, *B* is returned as *R*.
Operation	:	*B* is formatted and output to the current SINK device according to the Output Formatting rules (page 1.2).

CHARACTER INPUT:

Syntax	:	*R* ← *A* ⍞ *B*
Domain	:	*B* must be character *A* must be an index ≤ ⎕*IO*+132
Conformability	:	0 = ⍴⍴*A* 1 ≥ ⍴⍴*B* (×/⍴*B*) ≤ 132
Result	:	*R* is a character vector.
Operation	:	If the current SOURCE device is to be prompted, *B* is output to the device according to the Output Formatting rules (page 1.2). If *A* is present, the cursor (or carriage) is left positioned at the logical position of the character *B*[*A*] (note: origin dependence). If (×/⍴*B*)<1+*A*-⎕*IO*, *B* is extended on the right with spaces. If *A* is absent, its value is assumed to be (×/⍴*B*)+⎕*IO* -1. If the current SOURCE device is not interactive, no prompt is output. A line of input is then obtained from the device, and any prompt output is treated as if it were part of the input. The input is processed according to the Input Formatting Rules, and the resulting character vector is returned as *R*.

DIRECT DEVICE ACCESS:

Basically, six operations are available:

1) Address device
2) Read device code
3) Send command to device
4) Read device status
5) Send data to device
6) Read device data

The function of these operations is achieved with four APL system functions - ⎕*OU*, ⎕*IN*, ⎕*BO*, ⎕*BI*. All operations cause the device in question to be addressed, and its Answer-Back Code to be obtained. The Answer-Back Code is hard wired 8-bit response which returns information about the class of the device. The Answer-Back Code of a device is made up as follows: Let *ABC* be the code received, and suppose we set:

```
C ← 2 2 2 32 ⊤ ABC
```

Then the elements of *C* are interpreted as follows:

ELEMENT	*ABC* BITS	MEANING
C[1]	7	1 = Input Device
C[2]	6	1 = Output Device
C[3]	5	1 = Prompt Valid for Input
C[4]	4 - 0	Device Type

The following device types have so far been assigned:

TYPES	DEVICE
0	Built-in keyboard/display
1	MCM/EIA Interface
2	MCM/Hytype or QUME Printer
3	MCM/Card reader or punch
4	MCM Cassette Drive
5	MCM Diskette Drive

In communicating with the Omniport, the 8 bits of the Omniport are represented in the system by their base 2 value. Bit 7 on the Omniport is the most significant bit.

⎕*OU*, ⎕*IN* :-

These functions are identical in operation except that ⎕*OU* (short for ⎕*OUT*) sets the system SINK, while ⎕*IN* sets the system SOURCE.

Syntax	:	$R \leftarrow \begin{bmatrix} \square IN \\ \square OU \end{bmatrix} B$
Conformability	:	1 = ρρ*B* 2 ≥ ρ*B*
Domain	:	The elements of *B* must be integers in the range 0 to 255.
Result	:	1 ≡ ρρ*R* 3 = ρ*R*
Operation	:	The current SOURCE/SINK is set to the device whose interface address is *B*[1]. If *B* is empty, the SOURCE/SINK device is left unchanged. The device is selected, and the answer-back code from it is returned in *R*[2]. *R*[1] contains the device address. *R*[3] contains the current device status. If *B*[2] is present, it is output as a command to the device.

If no device is present, the answer-back code, status, and data will all be zero.

The command and status codes are peculiar to the device being accessed. Refer to the documentation for the device in question.

The sequence of events is:

1. Device Addressed
2. Answer-back code obtained
3. Command (if any) issued
4. Status obtained
5. Device de-addressed

⎕*BO*, ⎕*BI* :

These functions are used to output and input data via the omniport.

Syntax : $\{R \leftarrow\} \left\{\begin{matrix} ⎕BI \\ ⎕BO \end{matrix}\right\} B$

Conformability : $0 = \rho\rho B$
or $0 \equiv \rho B$

Domain : The elements of *B* must be integers in the range 0 to 255.

Result : $0 = \rho\rho R$ if *B* is empty
otherwise $(\rho R) \equiv \rho B$

Operation : If *B* is empty, the operation is a request for input, and *R* is input data from the current $\left\{\begin{matrix} \text{SOURCE} \\ \text{SINK} \end{matrix}\right\}$ device.

Otherwise, *B* must be a scalar, representing data which is output to the current $\left\{\begin{matrix} \text{SOURCE} \\ \text{SINK} \end{matrix}\right\}$ device. In this case, if a result is requested, *B* is returned as *R*.

The sequence of events in both cases is:

1. Device Addressed
2. Answer-back obtained
3. Data transferred
4. Device de-addressed

DEVICE ADDRESS LOCATION :

Syntax	:	*R* ← ⎕*YA B*
Domain	:	*B* must be numeric, in the range 0 to 255.
Conformability	:	1 = ρρ*B*
		(0≤ρ*B*)∧(3≥ρ*B*)
Result	:	0 = ρρ*R*
		R is the address of the device located, or 0 if no device was found.
Operation	:	0 = ρ*B* : The address returned is the lowest address which represents an attached device.
		1 ≤ ρ*B* : Only devices whose answer-back code is ⎕*IO* are examined.
		2 ≤ ρ*B* : The answer-back code (*ABC*) is masked as follows before being compared to ⎕*IO*.
		ABC ← 2⊥(8ρ2⊤*B*[2])∧(8ρ2)⊤*ABC*
		3 = ρ*B* : Only devices with addresses above 2+⎕*IO* are examined.

DEVICE TABLE SEGMENT REFERENCE :

Syntax : $R \leftarrow \begin{bmatrix} \square YI \\ \square YO \end{bmatrix}[I\ ;B$

Operation : The SOURCE device table is accessed with ⎕*YI*. The SINK device table is accessed with ⎕*YO*.

The segment requested is indicated by the origin - dependent index *I* as follows:

$$I = \square IO + \begin{cases} 0 & : \text{CONTROL segment} \\ 1 & : \text{INPUT segment} \\ 2 & : \text{OUTPUT segment} \end{cases}$$

CONTROL SEGMENT

Domain : Elements of *B* must be integers in the range 0 to 10.

Conformability : 1 ≥ ρρ*B*

Result : (ρ*R*) ≡ (ρ*B*), ⌈/*N* [*B*+⎕*IO*]
Where *N*[*J*] (in origin 0) is the length of Row *J* of the Control Segment, that is, since the rows of the Control Table are not all the same length, it is necessary to pick a result row length which will accomodate all requested control segment rows. *R* is integer.

Operation : Each row of *R* is the row of the control table indicated by the corresponding element of *B*. Short rows are padded on the right with zeroes.

NOTE: *B* is <u>not</u> origin dependent.

INPUT TRANSLATE SEGMENT

Domain	:	The elements of *B* must be integers in the range ¯10 to 127.
Conformability	:	1≥ρρ*B*
Result	:	(ρ*R*)≡ρ*B* *R* is integer.
Operation	:	Each element of *R* is the system code value to which the device code represented by the corresponding element of *B* translates on output. Note that for control characters, the system code value is minus the corresponding row number of the control segment. For non-control characters, the code value is that given by the conversion function ⎕*Y*.

OUTPUT TRANSLATE SEGMENT

Domain	:	*B* must be character
Conformability	:	1 ≥ ρρ*B*
Result	:	(ρ*R*) ≡ ρρ*B* *R* is integer
Operation	:	Each element of *R* is the device code value to which the corresponding character in *B* translates on output. If the character is to be represented as a mnemonic, the translate code is ¯1. The translate code for overstrike representation is ¯2.

DEVICE TABLE SEGMENT MODIFICATION :

Syntax : $R \leftarrow A \left\{\begin{matrix}\square YI\\ \square YO\end{matrix}\right\}[I]B$

Result : If a result is requested, *A* is returned as *R*.
Refer to the description of the monadic forms of these functions.

CONTROL SEGMENT

Domain : The elements of *A* must be integers in the range 0 to 255.

Conformability : (ρ*A*) ≡ (ρ*B*), ⌈/*N*[*B*+⎕*IO*]
where - *N* is as for the monadic form.

Operation : For each element *J* of *B*, row *J* of the control table is replaced with the first *N*[*J*+⎕*IO*] elements of the corresponding row of *A*. The remaining elements in the row are ignored.

If *B* contains duplicates, the operation is not defined.

INPUT SEGMENT

Domain : The elements of *A* must be integers in the range ¯10 to 127. Values other than system character code values or control code values are assumed to be the system canonical bad character code (108).

OUTPUT SEGMENT

Domain : The elements of *A* must be integers in the range ¯2 to 127.

INPUT and OUTPUT SEGMENTS

Conformability : (ρ*A*) ≡ ρ*B*
or (ρρ*A*) = 0

Operation : The translate value indicated by each element of *B* is set to the corresponding element of *A*. If *A* is a scalar, it is extended. If *B* contains duplicates, the operation is not defined.

DEVICE TABLE READ :

Syntax : *R* ← *A* ⎕*YR B*

Domain : *B* must be a character vector representing a valid APL name.
A must be the character scalar '*I*' or '*O*'.

Result : *R* is an integer scalar.

Operation : The variable named in *B* is replaced with the complete contents of the device table indicated by *A*,('*I*' for the SOURCE table, '*O*' for the SINK table).

If *B* does not name a variable or an undefined object, a RANGE ERROR is issued.

NOTE: The type of the resulting data specified to the variable is such that no function other than NULL (∘) will accept it as valid data. Any attempt to do so will result in a DOMAIN ERROR.

The result is the device answer-back code associated with the table at the time it was created. This code is the device answer-back code with either the OUTPUT or the INPUT and PROMPT bits masked off. The OUTPUT bit is masked off for the SOURCE table, and the INPUT and PROMPT bits are masked off for the SINK table.

DEVICE TABLE WRITE :

Syntax : *R* ← *A* ⎕*YW B*

Domain : *B* must be a character vector representing a valid APL name.
A must be the character scalar '*I*' or '*O*'.

Result : *R* is an integer scalar.

Operation : As for ⎕*YR*, except that a device table is created with the data in the variable named by *B*.

If *B* was not created by ⎕*YR*, or if the device table referred to by *A* already exists, a RANGE ERROR is issued. The result is the same as for ⎕*YR*.

DEVICE TABLE EXPUNGE :

Syntax : *A* ⎕*YX B*

Domain : *B* ≡ ⍳0

A must be the character scalar '*I*' or '*O*'.

Operation : The table indicated by *A* is unconditionally destroyed.

CHARACTER/NUMERIC CONVERSION :

Syntax	:	*R* ← ⎕*Y B*
Domain	:	*B* may be character or numeric. If *B* is numeric it must be integer data in the range 0 to 255.
Conformability	:	None
Result	:	(ρ*R*) ≡ ρ*B*
Operation	:	If *B* is character, *R* is numeric. Each element of *R* is the value of the system code for the corresponding character in *B*.

If *B* is numeric, *R* is character. Each element of *R* is the character for which the corresponding element of *R* is the system code value. Codes which do not correspond to valid system characters are mapped into this system canonical bad character. (Code 108). Control characters are not valid system characters.

See Appendix for system code values.

DELAY :

Syntax	:	*R* ← ⎕*DL B*
Domain	:	*B* must be numeric, in the range 0 to 25.5.
Conformability	:	0 =ρρ *B*
Result	:	*R* = *B*
Operation	:	A delay of at least *B* seconds occurs before the function returns its result. In most cases, the delay will be approximately equal to *B*. However, the delay timing is suspended while the control key on the main keyboard is held down. Timing resolution is 0.1 sec.

PRINT COUNTERS :

Syntax	:	*R* ← ⎕*PC B*
Domain	:	*B* must be integer in the range 0 to 255.
Conformability	:	1 = ρρ*B* 2≥ρ*B*
Result	:	2 ≡ ρ*R*
Operation	:	The print counters form a two-element vector. The first element is a page counter, and the second is a line counter. Each time a physical line is output to the current SINK DEVICE (if output is diverted), or to the current SOURCE DEVICE, if its address is the same as that of the current SINK, the line counter is incremented. Whenever the end of a page is reached (or whenever the line counter reaches 256, if paging is off), the line counter is reset to zero, and the page counter is incremented. The page counter is reset when it reaches 256. If *B* is two elements, they are used to set the current page and line counter. If *B* is one element, the page counter only is set. If *B* is empty, the counters are unaffected. The result is the page and line counter after setting according to *B*.

ON WRITING DEVICE SUPPORT

ON WRITING DEVICE SUPPORT

I - The following procedure should be used in setting up the device table for a completely new device. The examples are for a Teletype Model 33.

1. Determine the set of APL characters which correspond directly to single characters on the device. Let us assume that we have this set in a vector called A1.

```
          A1←'0123456789'
          A1←A1,'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
          A1←A1,' $''()*+,-./'
          A1
    01234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ $'()*+,-./
```

2. Determine the set of device codes which correspond to the above set of APL characters (ignore parity bits). Let us place this in a vector D1.

```
          D1←48+(⍳10)-⎕IO
          D1←D1,65+(⍳26)-⎕IO
          D1←D1,32+0 4,7+(⍳9)-⎕IO
```

3. If the device is to be used for output, load these values into the SINK table output translate segment. (Note: before accessing the device tables in any way, make sure that SOURCE and SINK are set to the device in question. For example, if the address of the device is 2, execute: ⎕IN 2∘⎕OU 2).

```
          D1 ⎕YO[2+⎕IO]A1
```

4. If the device is to be used for prompted input (this requires the PROMPT bit in the answer-back code to be on), load these values into the Source table output translate segment.

```
          D1 ⎕YI[2+⎕IO]A1
```

If this results in a TYPE ERROR, input may <u>not</u> be prompted.

5. If the device is to be used for input, A1 must be converted to numeric, and the result loaded into the Source table input translate segment.

```
          (⎕Y A1)⎕YI[1+⎕IO]D1
```

Note that only the Source table has an input translate segment.

6. Next determine the set of characters which are to be represented as overstrikes on the device. Let us call this set *A2*. Set these locations in both output translate segments to 2. (In this example, *A2* is empty).

```
          ¯2 ⎕YO[2+⎕IO]A2
          ¯2 ⎕YI[2+⎕IO]A2
```

7. Determine the set of characters which are to be represented as mnemonics on the device. Let us call this set *A3*. Set these locations in both output translate segments to ¯1.

```
          ALF←⎕Y (⍳109)-⎕IO
          A3←(~ALF∊A1)/ALF
          ¯1 ⎕YO[2+⎕IO]A3
          ¯1 ⎕YI[2+⎕IO]A3
```

8. Determine the set of device codes which correspond to control characters expected in input. Let us call this set DC. Load these locations in the input translate segment with the appropriate control codes (called AC).

```
          DC←13          (carriage return)
          AC←¯2          (newline)
          AC ⎕YI[1+⎕IO]DC
```

9. Fill all remaining unspecified slots in the input translate segment with Idle or Bad Character codes, as desired.

```
          X←(⍳32)-⎕IO
          DI←((~X∊CD)/X),⍳127
          ¯4 ⎕YI[1+⎕IO]DI
          X←32+(⍳95)-⎕IO
          DB←(~X∊A1)/X
          108 ⎕YI[1+⎕IO]DB
```

This can also be done by block filling the segment with Idles and/or Bad Characters prior to step 5.

10. At this point, the input and output translate segments have been completely set up. If desired, the contents of these segments should now be saved:

```
        YOO←⎕YO[2+⎕IO]ALF      (see step 7.)
        YIO←⎕YI[2+⎕IO]ALF
        YII←⎕YI[1+⎕IO]ALF
```

Note that since *YOO* and *YIO* are usually the same, some duplication can be avoided by simply setting up the SINK table output translate segment (which we have saved as *YOO*) and copying it into the source table output segment if prompts are to be output to the device:

```
        YOO ⎕YI[2+⎕IO]ALF
```

11. It is now necessary to set up the control segments. Since the form of the control table is different for each type of device, only the most common type, EIA, will be discussed here.

    First the row containing general information about the device (row 0) is set up.

    a) The first element of this row is made up of the sum of any of four possible values which represent flags controlling the device as follows:

       1 - Mnemonics valid on input.
       2 - Overstrikes valid on input.
       4 - Trailing spaces on each logical line truncated on output.
       32 - Prompt valid on input.

       In our case (*TY*33) we shall set the first element to:

       ```
       R00←32+4+1
       ```

       Since the device has no backspace, overstrikes are impossible (This is also the usual setting for a CRT, for which backspace is destructive).

    b) The second element is the sum of four possible flags and a count, defined as follows:

       128 - Echo input back to the device (FULL DUPLEX)
       64 - The device has a two-level code set.
       32 - A Break from the device interrupts output.
       16 - A Break from the device simulates an EOT on input.
       0-15 - Ignore this many characters at the start of each input

       In our case, we shall set the first element to:

       ```
       R01←128+32+16+0
       ```

c) The third element is the sum of four possible flags and a count as follows:

64	- Ignore parity errors on input
16	- Serial word has two stop bits
8	- Serial word has no parity bit.
4	- Serial word has even parity
3-0	- Number of data bits in serial word, minus 5.

In our case, we shall set the third element to:

```
R02←64+16+4+(7-5)
```

d) The fourth element is a code for the data rate, determined by the following formula:

```
R←⌊¯0.5+25000÷B
```

where *B* is the Baud rate for the device. In our case:

```
R03←⌊¯0.5+25000÷110
```

Having determined all four elements, they are then loaded into row 0 of the control segments:

```
R0←R00,R01,R02,R03
R0 ⎕YO[⎕IO]0
R0 ⎕YI[⎕IO]0
```

12. Row 1 of the control table is determined next. It consists of these elements - the display width to be used with the device, the code for the continuation character (device code, or 129 for mnemonic representation, 130 for overstrike representation, or 128 for no character), and the size of the indent to be placed at the beginning of continuation lines.

In our case, we shall use a width of 72, mnemonic representation of the continuation character, and a 6-space indent:

```
R1←72 129 6
R1 ⎕YO[⎕IO]1
R1 ⎕YI[⎕IO]1
```

13. Row 2 of the control segment gives the newline output sequence for the device. The first element is a time-out for input. If the element is non-zero, and that many seconds elapse between two successive characters during an input operation, a newline is simulated, terminating the input. In our case, we shall set:

```
        R20←0
```

The second element represents an idle count for the operation. If this element is non-zero, the physical cursor (or carriage) position at the end of the line is divided by this number, and the result is the number of idles transmitted after the newline. In our case:

```
        R21←0
```

The third and fourth elements are a pair of codes forming the newline operation. (These must be either device codes, or 128 for no character). In our case, the newline operation consists of a carriage return (13) followed by a line feed (10).

The resulting values are then loaded into the row:

```
        R2←R20,R21,10 13
        R2 ⎕YO[⎕IO]2
        R2 ⎕YI[⎕IO]2
```

14. Rows 3 and 4 of the control segment give the backspace and idle output codes for the device. In our case, the device has no backspace, and the idle code is 0:

```
        X34←2 1⍴128 0
        X34 ⎕YO[⎕IO]3 4
        X34 ⎕YI[⎕IO]3 4
```

Row 5 is not used.

15. Row 6 gives the form feed output sequence for the device. The first element is the number of lines to be printed on each page. The second element is a count of the number of times the pair of codes represented by the third and fourth elements are to be transmitted to make up the form feed action.

In our case, although a teletype usually has continuous paper, requiring the first element to be zero, we will assume we have loaded it with 66 line paper, of which 60 lines are to be used for printing:

```
R6←60 6 10 128
R6 ⎕YO[⎕IO]6
R6 ⎕YI[⎕IO]6
```

16. Rows 7 and 8 give the shift codes for a device with a two-level code set. Since the teletype is not such a device, we have:

```
X78←2 1⍴128
X78 ⎕YO[⎕IO]7 8
X78 ⎕YI[⎕IO]7 8
```

17. Rows 9 and 10 give the output sequences for EOT and BOT respectively. The code pair represented by the second and third element are output, followed by the number of idles given in the first element.

    In our case, END-of-Transmission and Beginning-of-Transmission are not very meaningful, since the teletype is a full-duplex device. However, since EOT occurs at the beginning of a request for input, we can transmit a BEL (7) to the device to tell the user at the keyboard that input is being requested:

```
X9T←2 3⍴0 7 128, 0 128 128
X9T ⎕YO[⎕IO]9 10
X9T ⎕YI[⎕IO]9 10
```

18. Now that the control segments have been set up, it will probably be useful to save them:

```
YOC←⎕YO[⎕IO](⍳11)-⎕IO
YIC←⎕YI[⎕IO](⍳11)-⎕IO
```

    Note that since in most cases the control segment is the same for both source and sink tables, it is usually easier to set up one (say, the sink table), and use the saved result (which we have called *YOC*) to load the other:

```
YOC ⎕YI[⎕IO](⍳11)-⎕IO
```

19. Further, now that the entire contents of both tables have been loaded, it may be useful to save them in toto, in order to make reloading easier than setting up each segment individually:

```
        'O' ⎕YR 'YYO'
        'I' ⎕YR 'YYI'
```

At any future time, the tables may be reloaded as follows:

```
        'O' ⎕YW 'YYO' ∘ 'O' ⎕YX ⍳0
        'I' ⎕YW 'YYI' ∘ 'I' ⎕YX ⍳0
```

The expunge (⎕*YX*) is usually necessary, because ⎕*YW* will <u>not</u> replace an existing table, as the contents may have not been saved.

II - The following special considerations should be noted in writing device tables for IBM 2741 terminals (or equivalent).

1. Since the 2741 can reside indefinitely in any one of three states, and since it is capable of printing output in only one of these states, the user must be careful never to transmit an EOT character (code 60) to the device, except via row 9 of the control segment. Similarly, row 10 must contain a BOT (code 52).

   If the device state is altered (either by power down or switching temporarily into LOCAL mode), it must be reset by issuing a request for ⍞ input, and pressing the return key on the terminal. If the terminal is equipped with a reverse Break, the same affect may be achieved in direct access mode with the following sequence:

   ```
   ⎕OU A,4    (where A is the device address)
   ⎕DL 0.5
   ⎕BO 52
   ```

   This issues a reverse break and a BOT to the device. This will only work if I/O to the device has already been attempted.

2. Since the 2741 has a two-level code set, the Shift bit (flag value 64) in the second element of row 0 of the control segment, must be set, and the shift codes (31 and 28) must be loaded into rows 7 and 8 of the control segment.

   Further, to maintain synchronization with respect to case, the newline sequence in row 2 must contain a downshift (31).

3. The appropriate newline idle count is 8. The user may have to tune this for his specific terminal.

4. A 2741 runs at 134.5 Baud, with a 6-bit data word, 1 stop bit, and odd parity.

5. When an EOT (60) is transmitted to a 2741, it responds with a BOT (52). Thus the ignore count in the second element of control segment row 0 should be 1.

6. If the form feed is being used, it should be simulated using linefeed/idle (46 61) or newline/idle (45 61). The former is to be preferred, but some 2741 type terminals do not respond to a linefeed.

7. The following is a list of the code values for 2741 control characters and their corresponding input translate segment values:

Newline	-	45	(¯2)
Backspace	-	29	(¯3)
Idle	-	61	(¯4)
Linefeed	-	46	
Downshift	-	31	(¯7)
Upshift	-	28	(¯8)
EOT	-	60	(¯9)
BOT	-	52	(¯10)

III - The following special considerations should be noted in communicating with external computer systems.

1. The most difficult problem to deal with in talking to an external computer system is the multi-line response. Since the Communications SubSystem will only accept one line of input at a time, steps must be taken to ensure that this condition is not violated, otherwise input will be lost.

   This is best done by having a monitor run in both the local and remote-systems, which hands across and acknowledges one line at a time. Included in this description are the listings of a pair of APL functions, *MΔIN* and *MΔOUT*, which implements this procedure for character matrices. With these functions it is possible to build a package which transfers data of an arbitrary nature by first converting it to a character matrix, then converting back again after transfer.

   Note the use of the function *XFER*, which guarantees that synchronization of the two systems is maintained.

2. Except in very unusual cases, the prompted input used for terminals is not meaningful for external computer systems. Thus the PROMPT bit in the answer-back of the hardware interface and/or the PROMPT flag in the first element of row 0 of the control segment should be off. Input should be requested using ⍞ ''.

3. If the two systems get out of synchronization, a request for input may wait indefinitely for a termination. In order to recover from this condition automatically, the time-out element of row 2 of the control segment should be set to a reasonable non-zero value (say, 30 seconds). This will cause the Communications SubSystem to terminate the input operation unilaterally if this time elapses without a response.

   At this time, all input which has been received up to this point is returned, and the program can then take action on the basis of the input which was received.

   Premature termination may also be triggered with an attention at the main keyboard (control, '←' for one second).

4. When first connecting to an external computer system, it may not be possible to have the monitors discussed in (1) active at that time. It is possible to deal in a very limited way with multi-line responses using the following technique.

   Pick a code which is known to be transmitted by the external system immediately prior to a request for input (for 2741 compatible systems, this consists of an EOT). Specify that location in the input translate segment as a newline ($^{-}2$).

   Specify the location in the input translate segment corresponding to the device code for newline to some other character (such as Bad Character).

   The lines contained in an input response will then be delimited by the character substituted for the newline character, and can be picked out and dealt with separately.

   The most serious limit to this technique is that the <u>entire</u> input response must not overflow the Communication SubSystem's input buffer of 133 characters.

5. Since the external system may respond to an EOT with a non-zero number of non-significant acknowledgment codes, the ignore count in the second element of row 0 of the control segment should be set to bypass that many characters.

6. Since some systems will not tolerate any delay between a newline and an EOT, it may be necessary to set the newline output code to nothing, and the EOT output code to newline/EOT.

CHARACTER MATRIX TRANSFER PACKAGE

```
      ∇ R←MΔIN X;I;N;L;E
[1]     →R←0×⍳2≠⍴N←NVEVAL L←XFER X
[2]     →0×⍳0=×/⍴R←N⍴' '⋄I←1⋄'ORIGIN 1'
[3]   LP:→ERR⌈⍳(N[2]<⍴L)∨0=⍴L←XFER L
[4]     R[I;]←N[2]↑L
[5]     →LP⌈⍳N[1]≥I←I+1
[6]     →0×⍳0=⍴L←XFER L
[7]   ERR:→LP⌈⍳(N[1]≥I)∧0<⍴L←XFER''
[8]     R←(1,⍴R)⍴R
      ∇

      ∇ R←MΔOUT X;I;N
[1]     →R←0×⍳(' '≠0\0⍴X)∨(2≠⍴⍴X)∨WIDTH<¯1↑1,⍴X
[2]     →R←VOUT FMT ⍴X
[3]     →END⌈⍳0=×/⍴X
[4]     N←1↑⍴X⋄I←1⋄'ORIGIN 1'
[5]   LP:→R←VOUT X[I;]
[6]     →LP⌈⍳N≥I←I+1
[7]   END:R←⍳0
      ∇

      ∇ R←XFER X
[1]     ⋄⎕IN 1↑⎕OU ⎕YA 1 31
[2]     R←⍞''⋄⎕←X
[3]     ⋄'IN CASE ATTENTION'
      ∇

      ∇ R←NVEVAL X
[1]     R←⍎X
[2]     →0×⍳(0=0\0⍴R)∧1=⍴⍴R
[3]     R←(0=⍴⍴R)/,R
      ∇

      ∇ R←VOUT X;Y
[1]     Y←XFER X
[2]     →R←0×⍳(⍴Y)≠⍴X
[3]     R←0×⍳Y∨.≠X
      ∇

      ∇ R←FMT X
[1]     R←⍕X
      ∇

      ∇ R←WIDTH
[1]     ⍝R←⎕PW
[2]     R←''⍴⎕YO[⎕IO]1
      ∇
```

APPENDIX

ERROR MESSAGES AND POSSIBLE CAUSES

COMM TABLE ERROR -

The device tables indicate a logically impossible operation:

a) Overstrike or mnemonic representation has been requested for an output character which has none (e.g. 'A')

b) A negative value other than ¯1 or ¯2 was fetched from the output translate segment for an output character.

c) A negative value was fetched from the output translate segment for one of the two characters being used in an overstrike representation of an output character.

d) A value >127 was specified as the output code for a character in a control sequence (N.B. this does not apply to the continuation sequence).

e) A width less than 3 plus the size of the continuation character plus the continuation indent was discovered at continuation time.

COMM DEVICE ERROR -

A hardware error has occured on the current comm device, or a physically impossible device operation was requested.

HYTYPE/QUME Driver:

a) Device not powered on.
b) Device out of paper.
c) Margins violated.
d) Paper or carriage motion of ≥1024 increments requested

EIA Driver:

a) Data Set (or Terminal) not ready.
b) Transmit buffer not empty within 1 second (usually caused by too low or zero data rate).
c) Modem Carrier Lost.
d) Break received (while enabled) during output.

COMM TYPE ERROR -

The current comm device has the wrong answer-back code for the current requested operation.

a) Input (or output) requested for a non-input (or non-output) device.

b) ⍞ input or ⎕ output requested for an unsupported device.

c) A device table input (or output) translate segment requested for a non-input (or non-output) device.

d) The current device answer-back is different from the type that existed at the time the current device table was created (that is, the table was built for another type of device).

COMM SYSDEV ERROR -

I/O was requested for a device which is used by AVS/EASY.

COMM BUFFER ERROR -

The comm buffer overflows on the current output operation. This occurs if the device width is set larger than 133, or if a long prompt (somewhat less than 133, depending on continuation sequence settings) is issued.

OVERSTRIKE CHARACTERS

⍱	~	∨
⍲	~	∧
⍟	○	*
!	'	.
⌿	/	−
⍀	\	−
⌽	○	|
⊖	○	−
⍉	\	○
⍋	|	∆
⍒	∇	|
⍝	∩	∘
⍎	∘	⊥
,̄	,	¯
‡⃞	‡	⎕
⍞	'	⎕
$	|	*S*
⍕	∘	⊤
⌷	]	[
⪽	⊂	∘

MNEMONIC REPRESENTATIONS - SORTED BY MNEMOMIC

$AL	α	ALPHA	$LG	⍟	LOG
$AN	∧	AND	$LP	⍝	LAMP
$BC	▯	BAD CHARACTER	$LT	<	LESS THAN
$BS	\	BACKSLASH	$MD	∣	MODULUS
$BT	⍀	BACKSLASH*	$ML	×	MULTIPLY
$BV	⊥	BASE VALUE	$MN	⌊	MINIMUM
$CB	]	CLOSE BRACKET	$MX	⌈	MAXIMUM
$CI	○	CIRCLE	$ND	⍲	NAND
$CL	:	COLON	$NE	≠	NOT EQUAL
$CN	⍪	COMMA*	$NG	¯	NEG
$CO	ɑ	CONTINUATION	$NL	∘	NULL
$DG	⍒	DOWNGRADE	$NR	⍱	NOR
$DL	∇	DEL	$OB	[	OPEN BRACKET
$DO	$	DOLLAR	$OM	⍵	OMEGA
$DQ	¨	DOUBLE QUOTE	$OR	∨	OR
$DR	↓	DROP	$QD	⎕	QUAD
$DT	∆	DELTA	$QP	⍞	QUAD-PRIME
$DV	÷	DIVIDE	$QT	'	QUOTE
$EP	∊	EPSILON	$QU	?	QUERY
$EQ	=	EQUAL	$RO	⍴	RHO
$EV	⍎	EVALUATE	$RP	⊤	REPRESENTATION
$EX	!	EXCLAMATION	$RT	⌽	ROTATE
$FM	⍕	FORMAT	$RU	⊖	ROTATE*
$GE	≥	GREATER OR EQUAL	$SC	;	SEMICOLON
$GO	→	GOTO	$SM	⌿	SLASH*
$GT	>	GREATER THAN	$TA	↑	TAKE
$ID	⊂	IMBED	$TL	~	TILDE
$IN	⊃	INCLUSION	$TP	⍉	TRANSPOSE
$IO	⍳	IOTA	$UG	⍋	UPGRADE
$IS	←	IS	$UL	_	UNDERLINE
$IX	∩	INTERSECTION	$UN	∪	UNION
$LE	≤	LESS OR EQUAL	$XD	⌹	MATRIX DIVIDE

* - ALONG 1ST CO-ORDINATE

MNEMONIC REPRESENTATIONS - SORTED BY CHARACTER

_	$UL	UNDERLINE	⍳	$IO	IOTA
∆	$DT	DELTA	⍴	$RO	RHO
⎕	$QD	QUAD	⌽	$RT	ROTATE
¯	$NG	NEG	⊖	$RU	ROTATE*
<	$LT	LESS THAN	⍉	$TP	TRANSPOSE
≤	$LE	LESS OR EQUAL	←	$IS	IS
=	$EQ	EQUAL	;	$SC	SEMICOLON
≥	$GE	GREATER OR EQUAL	→	$GO	GOTO
>	$GT	GREATER THAN	⍋	$UG	UPGRADE
≠	$NE	NOT EQUAL	⍒	$DG	DOWNGRADE
∨	$OR	OR	[	$OB	OPEN BRACKET
∧	$AN	AND	]	$CB	CLOSE BRACKET
⍱	$NR	NOR	∘	$NL	NULL
⍲	$ND	NAND	⍝	$LP	LAMP
×	$ML	MULTIPLY	'	$QT	QUOTE
÷	$DV	DIVIDE	:	$CL	COLON
⍟	$LG	LOG	∇	$DL	DEL
⌊	$MN	MINIMUM	⍺	$AL	ALPHA
⌈	$MX	MAXIMUM	⍵	$OM	OMEGA
|	$MD	MODULUS	∩	$IX	INTERSECTION
!	$EX	EXCLAMATION	∪	$UN	UNION
○	$CI	CIRCLE	⊃	$IN	INCLUSION
~	$TL	TILDE	⊂	$ID	IMBED
?	$QU	QUERY	¨	$DQ	DOUBLE QUOTE
⌿	$SM	SLASH*	⍎	$EV	EVALUATE
\	$BS	BACKSLASH	⍪	$CN	COMMA*
⍀	$BT	BACKSLASH*	⌹	$XD	MATRIX DIVIDE
↑	$TA	TAKE	⍞	$QP	QUAD-PRIME
↓	$DR	DROP	$	$DO	DOLLAR
⊥	$BV	BASE VALUE	⍕	$FM	FORMAT
⊤	$RP	REPRESENTATION	▯	$BC	BAD CHARACTER
∊	$EP	EPSILON	ɑ	$CO	CONTINUATION

* - ALONG 1ST CO-ORDINATE

APL SYSTEM CHARACTER CODE VALUES

0:0	30:*T*	60:|	90:⍝
1:1	31:*U*	61:!	91:'
2:2	32:*V*	62:○	92::
3:3	33:*W*	63:~	93:∇
4:4	34:*X*	64:?	94:⍺
5:5	35:*Y*	65:⌿	95:⍵
6:6	36:*Z*	66:/	96:∩
7:7	37:∆	67:\	97:∪
8:8	38:⎕	68:⍀	98:⊃
9:9	39:	69:↑	99:⊂
10:_	40:.̲	70:↓	100:¨
11:*A*	41:¯	71:⊥	101:⍎
12:*B*	42:<	72:⊤	102:⍕
13:*C*	43:≤	73:∊	103:⌹
14:*D*	44:=	74:⍳	104:⍞
15:*E*	45:≥	75:,	
16:*F*	46:>	76:⍴	106:$
17:*G*	47:≠	77:⌽	107:▼
18:*H*	48:∨	78:⊖	
19:*I*	49:∧	79:⍉	¯1:*CO*
20:*J*	50:⍱	80:←	¯2:*NL*
21:*K*	51:⍲	81:;	¯3:*BS*
22:*L*	52:+	82:→	¯4:*IDL*
23:*M*	53:-	83:⍋	¯5:*CRR*
24:*N*	54:×	84:⍒	¯6:*FF*
25:*O*	55:÷	85:[	¯7:*SO*
26:*P*	56:⋆	86:]	¯8:*SI*
27:*Q*	57:⍟	87:(	¯9:*EOT*
28:*R*	58:⌊	88:)	¯10:*BOT*
29:*S*	59:⌈	89:∘	

OUTPUT EXAMPLES

```
      ⎕←(4ρ2)ρι2*4
 1  2                              Numeric formatting
 3  4                              and spacing between
                                   subarrays.
 5  6
 7  8


 9 10
11 12

13 14
15 16

      WIDTH 30
30

      ⎕←'1234567 1234567 1234567 1234567'
1234567 1234567 1234567 α          Line broken on space.
      1234567

      ⎕←70ρ'ASDFG(12345)'
ASDFG(12345)ASDFG(12345)α          Lines broken on parentheses.
      ASDFG(12345)ASDFGα
      (12345)ASDFG(12345)α
      ASDFG(1234

      ⎕←50ρ' '
                                   Trailing spaces truncated.

      ⎕←(49ρ' '),'*'
                             α     Truncation of logical line
                          *        only, not physical line.

      ⎕←'A≠B'∘NE←¯1 XLATE '≠'∘⎕←'A≠B'
A≠B
A$NEB                              Mnemonic output representation

      A←1↑(¯1,NE)XLATE'A≠'
      ⎕←'A≠B'
COMM TABLE ERROR                   'A' has no mnemonic.
      ⎕←'A≠B'
      ^

      ∘A XLATE 'A'

      ⎕PCι0∘⎕←⎕PCι0
1 1
1 2                                Line counter incremented.
```

INPUT EXAMPLES

```
      X←⍞'12345'
12345678                                       Prompted input.

      X
12345678                                       Visual fidelity

      X←5⍞'1234 6'
1234X6                                         Cursor positioning.

      X
1234X6

      X←5⍞'A≠34 6' ∘ NE←¯1 XLATE '≠'
A$NE34+6                                       Cursor at logical position.

      X ∘ NE XLATE '≠'
A≠34+6

      X←⍞8ρ'○'
○⊖○⌽⍉⊛⌀○                                       Overstriking.

      X
○⊖○⌽⍉⊛⎕○                                       Result of illegal overstike.

      X←⍞8ρ'○'∘OVS 0
○⊖○⌽⍉⊛⌀○                                       Overstriking suppressed.
                                               (Visual fidelity for CRT's)
      X
○- |\*/○

      X←⍞'' ∘ OVS 1
A$NE B                                         Mnemonic input transformation.

      X,':',⍕ρX
A≠ B:4

      X←⍞''∘MNEM 0
A$NE B

      X,':',⍕ρX
A$NE B:6                                       Mnemonic transformation
                                               suppressed.
      ∘MNEM 1
```

`[ ... ]` - Input from device.

INPUT EXAMPLES (cont'd.)

```
      X←⍞''
1234⍺                                   Input continuation.
56

      X
123456

      X←⍞''
1234⍺5                                  Illegal continuation.

      X
1234⎕5

      X←⍞'HELLO' ∘ PROMPT 0
THERE                                   Prompt suppressed.

      X                                 Visual fidelity.
THERE

      X←⍞''
123467                                  EOT (Attn) destroys input.
567                                      (EOT at '_')

      X ∘ PROMPT 1
567

      X←⍞''
123467                                  With prompt, EOT breaks line.
1234567

      X
1234567

      ⍞''
A$ND1                                   Mnemonics not transfomed
A$NE12                                  until newline.

A≠12
```

World's first
portable full-service
APL computer.
World wide
distribution/service.

MICRO COMPUTER MACHINES, INC.
133 Dalton Street, Kingston, Ontario, Canada K7L-4W2
(613) 544-9860 • TWX: 610-523-1155

COMMUNICATION

SUBSYSTEM

REFERENCE MANUAL

CONTENTS:

1. GENERAL DESCRIPTION

GENERAL DESCRIPTION

I - INTRODUCTION:

This subsystem is responsible for handling all explicit (that is, ⎕ or ⍞) I/O for the APL system. It contains built in drivers for a DIABLO or QUME printer, and an EIA Asynchronous communications interface. The EIA driver is controlled by user-accessible tables which reside in the APL workspace, and can be modified to support almost any EIA compatible asynchronous device, including ASCII and IBM 2741 devices, with both APL and non-APL character sets. Communication with other computer systems is also possible. Synchronous communication protocol is not supported.

II - DIVERTED I/O:

For the purposes of this description, SINK is the logical device to which output is sent, and SOURCE is the logical device from which input is obtained.

In MCM/70 APL, the following types of output can occur:

1. Data assigned to Quad (Quad Output)
2. Quad input prompt
3. Quad prime input prompt
4. Data left unassigned at the end of a line of immediate execution or user defined function (Implicit output)
5. Error Messages
6. Function editing display

Normally, all output is directed to the MASTER SINK (on the MCM/70, the built-in display). However, it is possible to divert output of the first type (that is, Quad output), to a device attached to the MCM/70's Omniport. A device to which output is so directed is called a DIVERTED SINK.

Similarly, the following types of input can occur:

1. Literal (Quad-prime) input
2. Quad Input
3. Input requested by the system when no function is active (Implicit Input)
4. Function editing input
5. Input requested for control of the display on the MASTER SINK

Normally, all input is obtained from the MASTER SOURCE (on the MCM/70, the built-in keyboard). However, it is also possible to divert input of the first type (that is, Quad-prime input), to a device attached to the Omniport (which becomes a diverted source).

In addition, if the device is known to the system to be able to accept prompts, the prompt normally output on the built-in display will be diverted to this device.

The diversion of I/O is accomplished with the system functions ⎕*IN* and ⎕*OU*, (see the description of the Direct Device Access Functions Page 2.5, 2.6). In general, ⎕*IN A* (or ⎕*OU A*) causes the device whose omniport address is *A* to become the current source (or SINK) device. Normal devices may have any address in the range 1 through 199. Address 0 refers to the MASTER device, and addresses above 199 are reserved for special purposes. The address of a normal device is set with switches on the device itself. (N.B. The system is <u>not</u> able to cope with the situation where two devices on the omniport have the same address).

III - <u>I/O FORMATTING</u>:

All diverted output is formatted, and all diverted input is de-formatted.

The following rules apply to output:

1. All numeric output is converted to character form. That is, in all cases, the expression ⎕←*X* is equivalent to the expression ⎕← ⍕ *X*, except that the former takes less time and space.

2. Boundaries between subarrays of rank ≥2 of output arrays of rank ≥2 are marked by a single blank line. (See examples in Appendix).

3. Trailing space truncation may be enabled or disabled. If disabled, no special action is taken. If enabled, trailing spaces on each <u>logical</u> output line are removed before formatting.

4. Characters whose output code is MNEMONIC (¯1; see DEVICE CONTROL) are converted to their equivalent Mnemonic form (see appendix). If no such representation exists, or if the code for the ecape character ('*$*') is MNEMONIC, a COMM TABLE ERROR will result.

5. Lines whose width after mnemonic conversion exceeds the width declared for the device are broken at the right-most space, left parenthesis, or right parenthesis to the left of the right margin of the device. If no such character exists, the line is broken at the right margin. Lines broken in this fashion have a Continuation Character (see Continuation, page 1.6) appended to them, and the next line is preceded with the continuation indent declared for the device. The maximum legal device width is 133.

6. Each character is translated to its declared output code before transmission. Legal output codes are 0 through 127, MNEMONIC (¯1), and OVERSTRIKE (¯2). Characters whose output code is OVERSTRIKE are transformed to their equivalent overstrike form (see Appendix), and the resulting triplet is re-translated prior to output. If any of the resulting triplet translates to a code outside the range 0 to 127, a COMM TABLE ERROR will result.

7. Each physical line output causes the device line counter to be incremented (see Form Feed, pg. 1.5.). If this causes line counter to reach the page size declared for the device, the line counter is reset to 0 and the page counter is incremented.

The following rules apply to input:

1. Visual fidelity is maintained. This means that if the input device has an integral display, the line returned to the system is that which appears on that display at the time that the input operation is terminated. Devices without an integral display are treated as if they had one.

2. The rightmost character in the line corresponds to the rightmost character (including spaces) input from the device, subject to margin restrictions.

3. Backspaces received at the left margin are ignored. Non-control characters received at the right margin are ignored, and the rightmost valid character is replaced with the system canonical Bad Character. The right margin is <u>fixed</u> at 133.

4. Overstriking may be enabled or disabled (see Row 0, pg. 1.6) If disabled, a character received at a position containing another character replaces that character.

   If enabled, the following rules apply:
   a) Any non-control character replaces a blank
   b) Blanks replace no other character
   c) Any character replaces itself
   d) Non-blank, non-control characters form their appropriate overstrike (see appendix). If no such valid overstrike exists, the result is the system canonical Bad Character.

5. Idle characters are ignored. Input Codes which do not translate to recognizable characters are replaced by the system canonical Bad Character.

6. An END-OF-TRANSMISSION character received from a prompted device causes the line to be truncated immediately to the left of the current position. The remainder is re-transmitted on a new line as a prompt for further input. For non-prompted devices, all input prior to the EOT is discarded.

7. A NEWLINE character terminates the input operation.

8. Mnemonic processing may be enabled or disabled (see DEVICE CONTROL). If disabled, no special action is taken. If enabled, an escape character ('$') followed by an alphabetic causes them and the following character to be converted to the appropriate character. If no such valid mnemonic exists, the result is the system canonical bad character.

9. If, after mnemonic processing, a continuation character is found at the right end of the line, the system returns to the input device for the continuation (N.B.: at this time, the right margin position is reduced by the logical length of the previous accumulated input).
Any input may consist of an arbitrary number of continuations.

10. If a prompt is provided for the input, and the device is capable of accepting it, it is transmitted to the device according to the rules for output, and the device is left positioned at the logical position specified for the operation (i.e. The left argument to ⍞, if one exists, or else at the right end of the output). If the prompt was broken (see Output rule 5) at a point to the right of the requested position, the device is left positioned at the left margin. The prompt is then treated as if it were input from the device (see examples in Appendix).

IV - DEVICE CONTROL:

Device Control is achieved via two sets of tables, one each for the SOURCE and SINK devices. These tables reside in the workspace, and may be modified by the user. For information on tables access, see the description of the Table Access System Functions. The MASTER DEVICES do not use these tables.

Each table consists of three segments: a CONTROL segment, and an INPUT segment, and an OUTPUT segment. The INPUT and OUTPUT segments are simply translate tables. Each output character is translated to the corresponding code in the OUTPUT segment. Permissible codes are 0 through 127, ¯1 and ¯2. Non-negative codes are output verbatim (except for Shift Control; see EIA INTERFACE). A code of ¯1 indicates that this character has no direct representation on the device, and must be output as a MNEMONIC (see Output Format rule 3). A code of ¯2 means that this character must be output as an OVERSTRIKE (see Output rule 5, page 1.3).

Each input code is translated to the corresponding character in the INPUT segment. Control characters are represented by character values less than zero. Each control character corresponds to a row in the CONTROL segment. The number of the row referred to is the absolute value of the control character.

The Control segment is used to specify properties of the device which cannot be specified in either the INPUT or the OUTPUT segment.

Control table rows are numbered starting at 0, and are of varying length. Each row except 0 refers to action to be taken in transmitting a control character. In most cases, this information is used only by the device driver, and is discussed there. However, certain elements are used by the system as a whole. The structure of the control table is as follows:

ROW	LENGTH	USAGE
0	4	General Device Information
1	3	Continuation
2	3	Newline
3	1	Backspace
4	1	Idle
5	3	Carriage Return
6	4	Form Feed
7	1	Shift Down
8	1	Shift Up
9	~~3~~ 2	End of Transmission
10	~~3~~ 2	Beginning of Transmission

0 - General Device Information:

The value of the first element of row 0 is:

$(32\times PROMPT) + 2\perp TRUNCATE, OVERSTRIKE, MNEMONIC$

where:

PROMPT =	1 if input prompts are to be issued to the device.
TRUNCATE =	1 if trailing spaces are to be truncated on output.
OVERSTRIKE =	1 if overstriking is enabled on input.
MNEMONIC =	1 if Mnemonics are enabled on input.

The remaining 3 elements are used only by the driver.

1 - Continuation:

The structure of row 1 is:

WIDTH, CHAR, INDENT

where:

WIDTH	-	is the physical width to be used for the device (maximum 132).
CHAR	-	is the code for the continuation character: 0 through 127 means an actual device code; 128 means no character; 129 means mnemonic; 130 means overstrike.
INDENT	-	is the number of spaces which precede the text of continuation lines. (maximum 15).

6 - Form Feed:

Row 6 is used for paging control. The first element is the number of physical lines to be printed on a page. When this limit is reached the number of Form Feed characters specified by the second row element is issued, and the line counter is reset to zero. If the limit is 0, page control is disabled.

3 - Toggle Ribbon States:

There are two state bits which affect ribbon position: UP/DOWN, and RED/BLACK. If Data 2048 is 1, the UP/DOWN state is toggled. All other data bits are ignored.

The ribbon state is normally UP and BLACK. Thus, in order to drop the ribbon:

3 *COMMAND* 2048

Repeating this command will raise it again.

NOTE: All HYTYPES, and QUMES which do not have a red ribbon option, will drop the ribbon if the state is set to RED.

4 - Feed Paper:

The paper is moved the number of increments (48th's of an inch) specified by Data 1 through Data 512. If Data 1024 is 0, the paper is fed up (normal motion). If Data 1024 is 1, the paper is fed down (reverse motion). Data ½ and Data 2048 are ignored. Thus to feed the paper *N* inches (down if *N* is negative):

4 *COMMAND* (1024×*N*<0)+48×|*N*

5 - Enable Platens:

Printers with split platens may have paper feed for each platen enabled independently. If a platen is disabled, it will ignore paper feed commands. If Data 2048 is 1, the left platen is enabled. If Data 1024 is 1, the right platen is enabled. Printers with only one platen are considered to have only a right platen.

6 - Form Feed:

Printers with the top-of-form option will feed the paper to the top of the next form. Others will ignore this command. All data bits are ignored.

7 - Restore Printer:

If a check condition occurs, the printer must be restored to remove the condition. The carriage is restored to the left margin. No other functions are affected. All data bits are ignored.

The status returned from the printer is an 8 bit binary word, defined as follows:

Bit 7	-	Paper feed ready
Bit 6	-	Carriage ready
Bit 5	-	Character print ready
Bit 4	-	Ribbon up
Bit 3	-	Ribbon red
Bit 2	-	Paper Out
Bit 1	-	Check condition (see carriage motion)
Bit 0	-	Printer powered and ready

Bits 0, 1, and 2 should be checked prior to any operation. Bits 5, 6, and 7 should be checked prior to each operation for the corresponding bit (ex: Bit 5 before printing a character). Bits 3 and 4 are used for setting a desired ribbon state. To set the ribbon <u>unconditionally</u> UP, for example:

```
3 COMMAND 2048×~STATUS[⎕IO+7-4]
```

Note that the bits are numbered in reverse order to that in which STATUS presents them. Note also the difference between this method and the one discussed under Ribbon States, which <u>toggles</u> the ribbon state.

<u>DEVICE TABLES</u>:

The OUTPUT segment is set up by the system for an APL print wheel. If the user wishes to use another print wheel, he should replace the output segment with the translate codes appropriate to that wheel.

Usage of the CONTROL segment is as follows:

<u>Row 0</u>:

This row has four elements. Ths first element is described in the general documentation of the control tables. The second element has the value:

```
4 2 4 8 ⊥ PLATENS, FLAG, RIBBON, STEP
```

At the beginning of each output, the printer is set so that the platens are enabled according to PLATENS, and the printer flag and ribbon states are set to FLAG and RIBBON. STEP is the ribbon step which is issued with each print command.
The defaults are:

PLATENS	=	1 (right)
FLAG	=	1 (on)
RIBBON	=	2 (up, black)
STEP	=	7 (Max. Step)

The third and fourth elements of row 0 are the number of half increments per, horizontal and vertical space, respectively. For HYTYPEs the horizontal spacing should be even, and for all printers the vertical spacing should be even. The defaults are 12 (10 characters/inch) and 16 (6 lines/inch).

Row 1:

This row was described in the general documentation. Its form is:

WIDTH, *CHAR*, *INDENT*

The defaults are 120, 130 (overstrike) and 6.

Row 2:

Only the second element of this row is used. ALL output is indented the number of character spaces indicated by this element. The default is 0.

Row 6:

This row was also described in the general documentation. Its form is:

PAGE_LIMIT, *COUNT*, *CMD*, *DATA*

When the PAGE-LIMIT is reached, the 16-bit binary word represented by:

$CMD+DATA\times COUNT$
is issued to the printer. The defaults are 0 (paging off), 0, 128 (paper feed up), 16 (1 line).

To print M lines on a form N lines high, the first two elements should be $M,N-M$.

HYTYPE APL10 PRINT WHEEL CODES

0:	32:•	64:¯	96:◇
1:	33:¨	65:α	97:*A*
2:	34:)	66:⊥	98:*B*
3:	35:<	67:∩	99:*C*
4:	36:≤	68:⌊	100:*D*
5:	37:=	69:∊	101:*E*
6:	38:>	70:_	102:*F*
7:	39:⌋	71:∇	103:*G*
8:	40:∨	72:∆	104:*H*
9:	41:∧	73:⍳	105:*I*
10:	42:≠	74:∘	106:*J*
11:	43:÷	75:'	107:*K*
12:	44:,	76:⎕	108:*L*
13:	45:+	77:|	109:*M*
14:	46:.	78:⊤	110:*N*
15:	47:/	79:○	111:*O*
16:	48:0	80:*	112:*P*
17:	49:1	81:?	113:*Q*
18:	50:2	82:⍴	114:*R*
19:	51:3	83:⌈	115:*S*
20:	52:4	84:~	116:*T*
21:	53:5	85:↓	117:*U*
22:	54:6	86:∪	118:*V*
23:	55:7	87:⍵	119:*W*
24:	56:8	88:⊃	120:*X*
25:	57:9	89:↑	121:*Y*
26:	58:(	90:⊂	122:*Z*
27:	59:[	91:←	123:{
28:	60:;	92:⊢	124:⊣
29:	61:×	93:→	125:}
30:	62::	94:≥	126:*$*
31:	63:\	95:-	127:¬

<u>Note</u>: When used with ⎕*BO* , these codes should be multiplied by 2.

DEFAULT HYTYPE/QUME OUTPUT
TRANSLATE TABLE SEGMENT

0:	48	*Q*:	113	÷:	43	→:	93
1:	49	*R*:	114	*:	80	⍋:	¯2
2:	50	*S*:	115	⍟:	¯2	⍒:	¯2
3:	51	*T*:	116	⌊:	68	[:	59
4:	52	*U*:	117	⌈:	83	]:	39
5:	53	*V*:	118	|:	77	(:	58
6:	54	*W*:	119	!:	65	):	34
7:	55	*X*:	120	○:	79	∘:	74
8:	56	*Y*:	121	~:	84	⍝:	¯2
9:	57	*Z*:	122	?:	81	':	75
_:	70	Δ:	72	⌿:	¯2	::	62
A:	97	⎕:	76	/:	47	∇:	71
B:	98	.:	46	\:	63	⍺:	65
C:	99	¯:	64	⍀:	¯2	⍵:	87
D:	100	<:	35	↑:	89	∩:	67
E:	101	≤:	36	↓:	85	∪:	86
F:	102	=:	37	⊥:	66	⊃:	88
G:	103	≥:	94	⊤:	78	⊂:	90
H:	104	>:	38	∊:	69	¨:	33
I:	105	≠:	42	⍳:	73	⍎:	¯2
J:	106	∨:	40	,:	44	⍕:	¯2
K:	107	∧:	41	⍴:	82	⌹:	¯2
L:	108	⍱:	¯2	⌽:	¯2	⍞:	¯2
M:	109	⍲:	¯2	⊖:	¯2	$:	¯2
N:	110	+:	45	⍉:	¯2	⍫:	¯2
O:	111	-:	95	←:	91	⌷:	¯2
P:	112	×:	61	;:	60		