HDS

human designed systems, inc.
3700 Market Street, Philadelphia, Pennsylvania 19104 (215) 382-5000

CONCEPT INFORMATION UPDATE

NUMBER: 14

DATE: APRIL 1, 1980

SUBJECT: WINDOWING

The purpose of this update is to review the basic structure of windowing within the concept terminals, provide an introduction to a multiple-window application, and present the advanced user with a more detailed understanding of device windows.

Basic Definition-Window

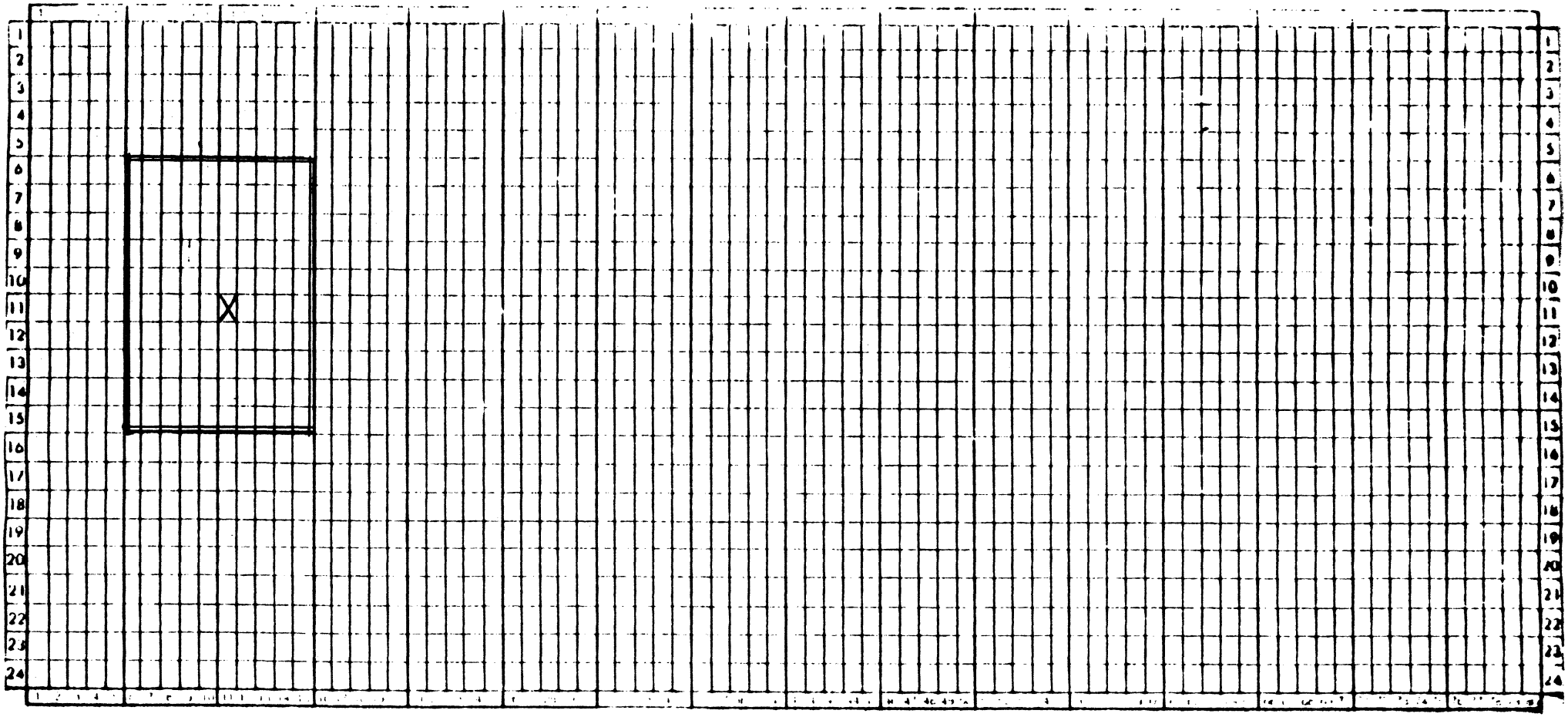
A "window" is a rectangular area of display memory, defined by the user for each device within the terminal (keyboard, line 1, and line 2).

A window can be of any size and starting location, limited only by the 80 columns and 24 lines (96 lines for a 4-page terminal) of the terminal's physical display memory.

A window is a "logical" definition which sets the boundaries for character display, in that characters cannot be written past the right-hand edge of the window, for example, or below the bottom line. When the terminal is powered on, all devices have a default window definition corresponding to all of the terminal's physical display memory.

At a given point in time each device has an "active" window definition, which may be changed at any time. Terminal operations are performed relative to the current, active window definition. Such operations include character display, cursor movement, cursor addressing, clear, send, etc. For example, moving the cursor to line 5, column 5 (relative to the window's home position of line 0, column 0) can be executed as long as the window defined is large enough to contain that address; however, the effect to the user may be different, since the windows could have different home positions. Figure 1-A shows the effects of addressing to line 5, column 5 for a one-page window defined as 5, 5, 10, 10 (that is: home position line, home position column, number of lines on the window, number of columns in the window); figure 1-B shows the same write address command with a window defined as 10, 20, 10, 10. In this example, in situation A the cursor is physically

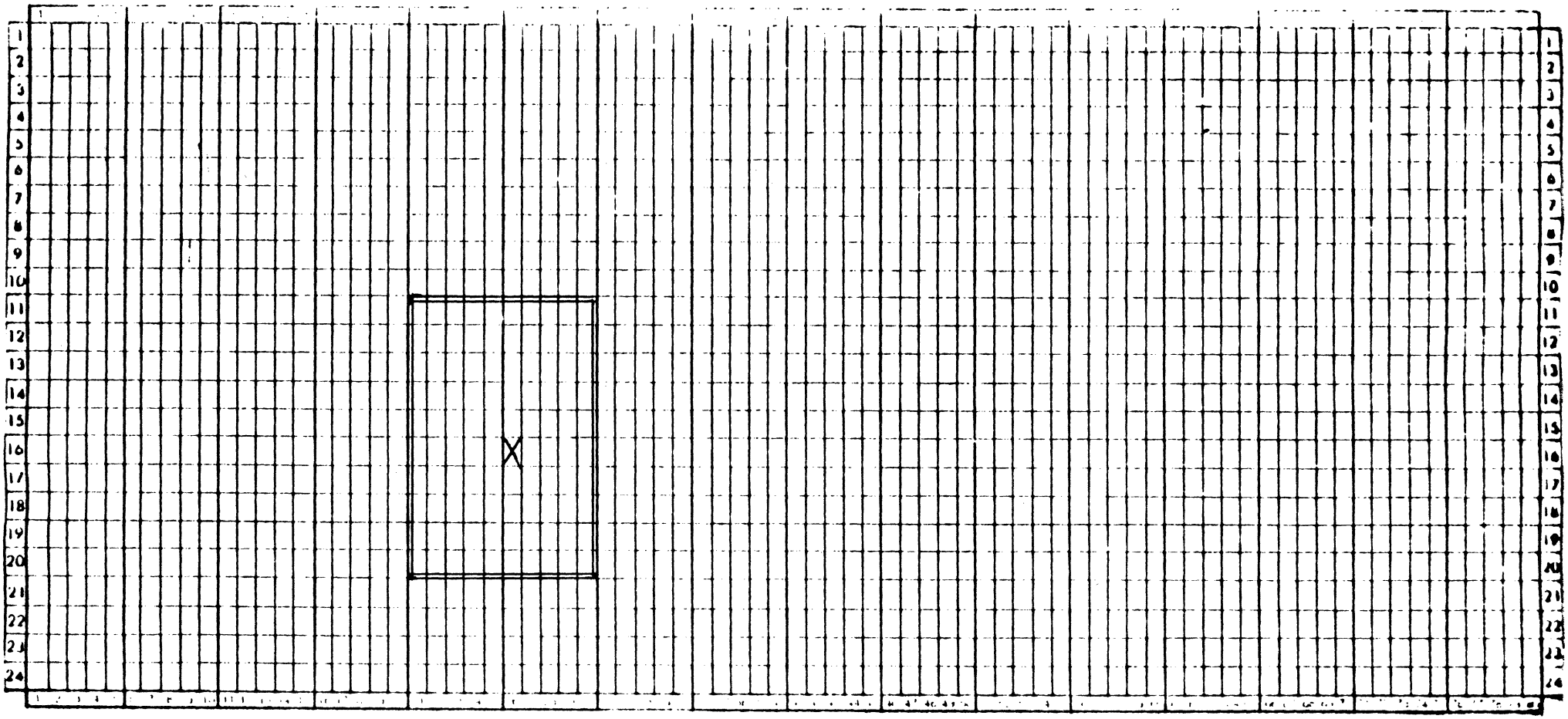
FIGURE 1-A



Window Definition = 5,5,10,10 (relative to 0)

Cursor Address = 5,5

FIGURE 1-B



Window Definition = 10,20,10,10 (relative to 0)

Cursor Address = 5,5

located (that is, relative to the actual screen display) at line 10, column 10; in situation B, the cursor is physically located at line 15, column 25.

Basic Definition-Cursor

Every window definition has associated with it a "cursor", which points to the location at which the next character is to be displayed. This cursor is updated whenever a character is displayed or the cursor is explicitly moved via a terminal command, for example. As mentioned above, each device within the terminal can have its own window definition; similarly, associated with each window definition and device is its cursor. The keyboard's cursor is represented as a blinking underline or blinking block and will always be visible to the user. The cursors for Line 1 and Line 2, if used, are "invisible"; that is, they are logical pointers to the next available character location, but are not represented directly on the screen.

Tied Windows

While each device may have its own window definition, in many applications (and when the terminal is powered on) only the keyboard's window definition and cursor are actually used; all other devices are "tied" to the keyboard. This means the characters and commands received from Line 1 and Line 2 will actually move the keyboard cursor, as will characters and commands for the keyboard. (In fact, Line 1 and Line 2 do have their "own" window definitions and cursors; however, they are not used when their windows are tied to the keyboard.)

Separate Windows

For many applications it is desirable for the keyboard and the communication line (Line 1) to be "independent" and have separate windows and cursors. For example, an application may have the user entering data in the keyboard window with appropriate error messages and responses appearing in a separate Line 1 window.

The windows for Line 1 and Line 2 can be separated from the keyboard (their default condition) by "tieing" the line to itself (that is, "un-tieing" it from the keyboard window). This is done by the Tie Window command (MC q device). When a device is tied to itself, it uses its own window definition, as opposed to the keyboard's window definition. The Tie Window command would then typically be followed by the line issuing a Define Window command for (presumably) a new window that is different than the keyboard window. Figure 2 shows the Escape sequences issued by Line 1 to configure a one-page terminal with two separate windows - the keyboard

Figure 2

SEPARATE WINDOWS EXAMPLE

Commands (*)	Decimal Equivalent	Comment
ESC v Ø Ø , p	27 118 32 32 44 112	Line 1 defines the keyboard window to be the "top" half of the screen (a home position of 0,0; 12 lines long and 80 columns wide). Note that after this command all display, from both the keyboard and Line 1, would appear only in the top half.
(**) ESC q !	27 113 33	Line 1 is tied to itself. After this command keyboard data would appear in the top half only; Line 1 data would appear in the entire screen (its default window definition)
ESC v , Ø , p	27 118 44 32 44 112	Line 1 defines its window to be the "bottom" half of the screen; a home position of 12, 0; 12 lines long and 80 columns wide). After this command all keyboard data appears in the top half and all Line 1 data in the bottom half.

(*) This assumes that Line 1 is issuing the command. To perform the same function from the keyboard, do the following (using Function Routing):

ESC v Ø Ø , p	27 118 32 32 44 112	Define keyboard window
ESC Q ! ESC q ! ↑W	27 81 33 27 113 33 23	Function route Line 1 tie window
ESC Q ! ESC v , Ø , p ↑w	27 81 33 27 118 44 32 44 112 23	Function route Line 1 window definition

(**) On some older terminals it may be necessary to Function Route a null command to a device other than Line 1 at this point; that is, send:

ESC q ! ESC Q Ø ↑@ ↑W 27 113 33 27 81 32 0 23

is the top half and the communications line is the bottom half. (Users with multiple-page terminals should be sure to remember that the keyboard cursor will always remain visible. If the keyboard were to be defined as page one and Line 1 as page four, the user would never be able to actually see any data sent by Line 1.)

Windowing - A Detailed Explanation

Each logical device within the terminal (keyboard/video, Line 1, and Line 2) has its own "device table", which is used to store information about that device. Included in the table is such information as:

- current cursor address
- current window definition
- "tie window" indicator
- attribute word
- network word

Of particular interest are the first three, which combine to control where characters received from that device are to be displayed. While each device logically can have separate cursors/windows, the "tie window" indicator actually controls where characters are to be displayed, in that it specifies the actual device table to be used (and therefore the cursor address and window definition). Figure 3 shows the logical structure when the terminal is powered on.

When a character is received over Line 1, for example, the following occurs:

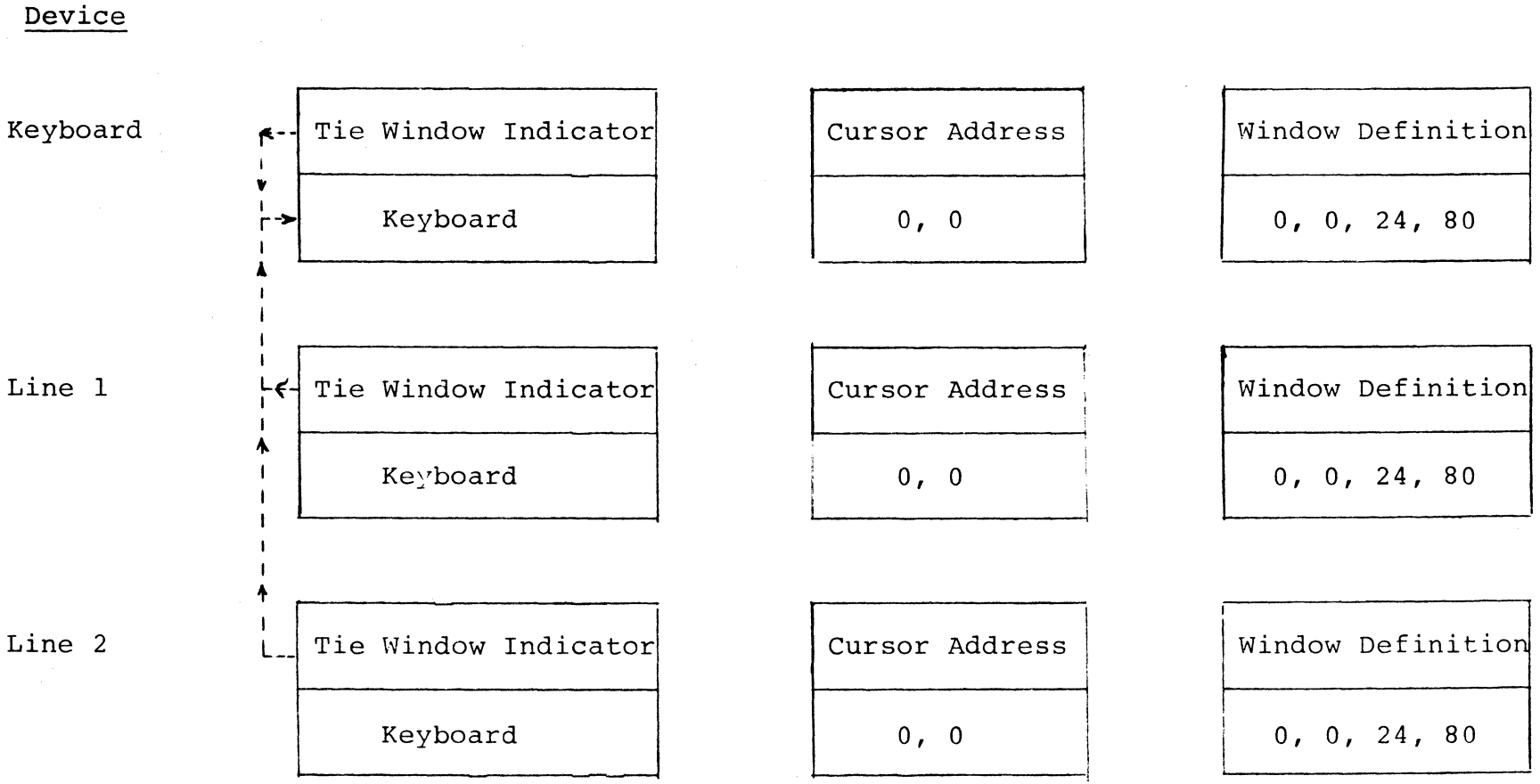
- The device table for Line 1 is determined
- The tie window indicator is obtained, and the cursor address to use is retrieved from that device table (that is, the keyboard device table).
- The character is placed in memory at the current cursor address.
- The current device attribute for the receiving device (Line 1) is also placed into memory at the current cursor address.
- The current cursor address (in the keyboard device table) is incremented by one, wrapping around to the next line of the right margin of the (keyboard) window is reached. (See Figure 4)

Note that even though data has been received from Line 1, its device table information has not been changed.

Let us assume now that the user has "separated" the keyboard and Line 1, as in Figure 2. Figure 5 shows the device

Figure 3

DEFAULT STRUCTURE (*)



(*) Assumes 1-page terminal

Figure 4

STRUCTURE AFTER RECEIPT OF ONE CHARACTER

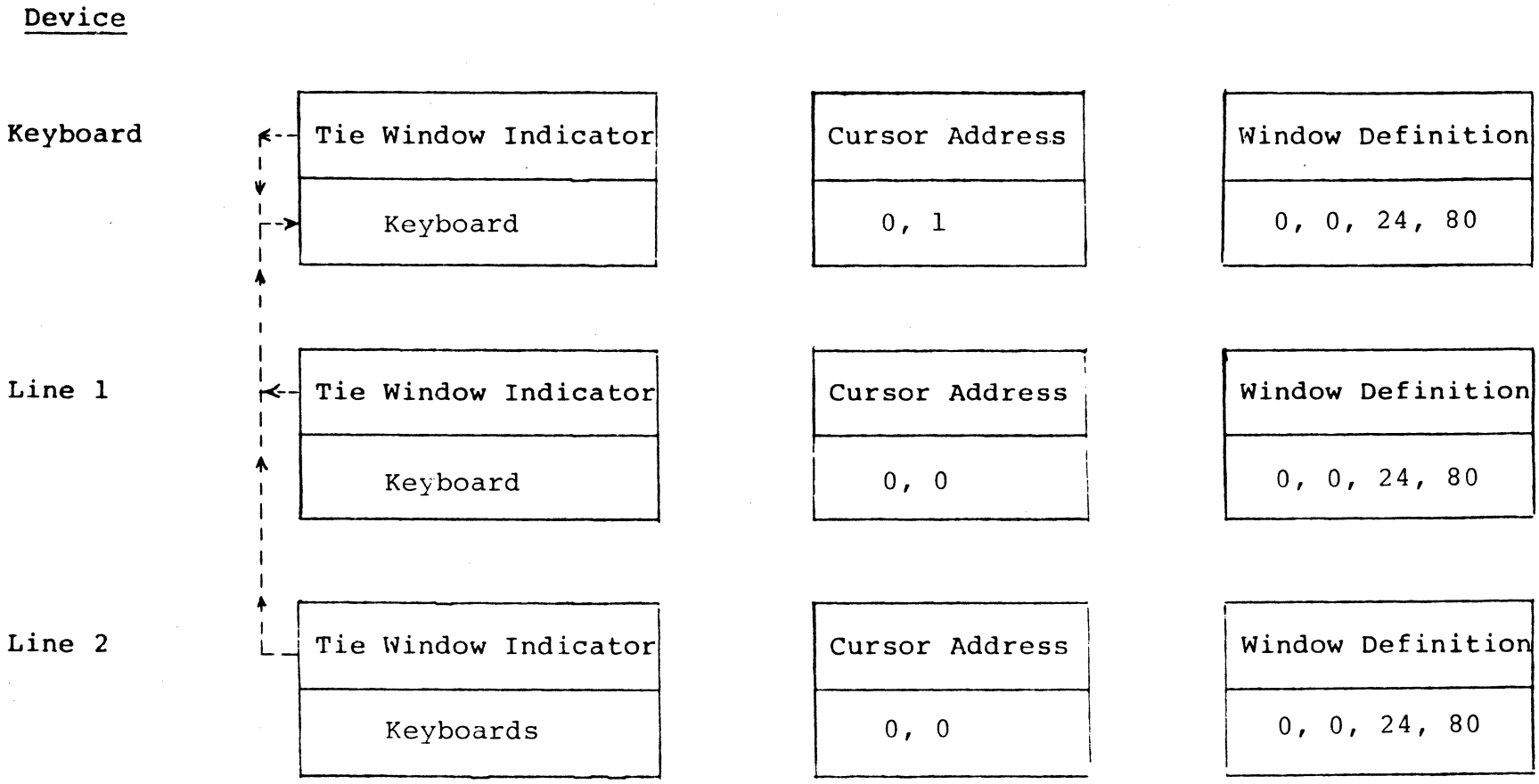


Figure 5

SEPARATED WINDOWS STRUCTURE

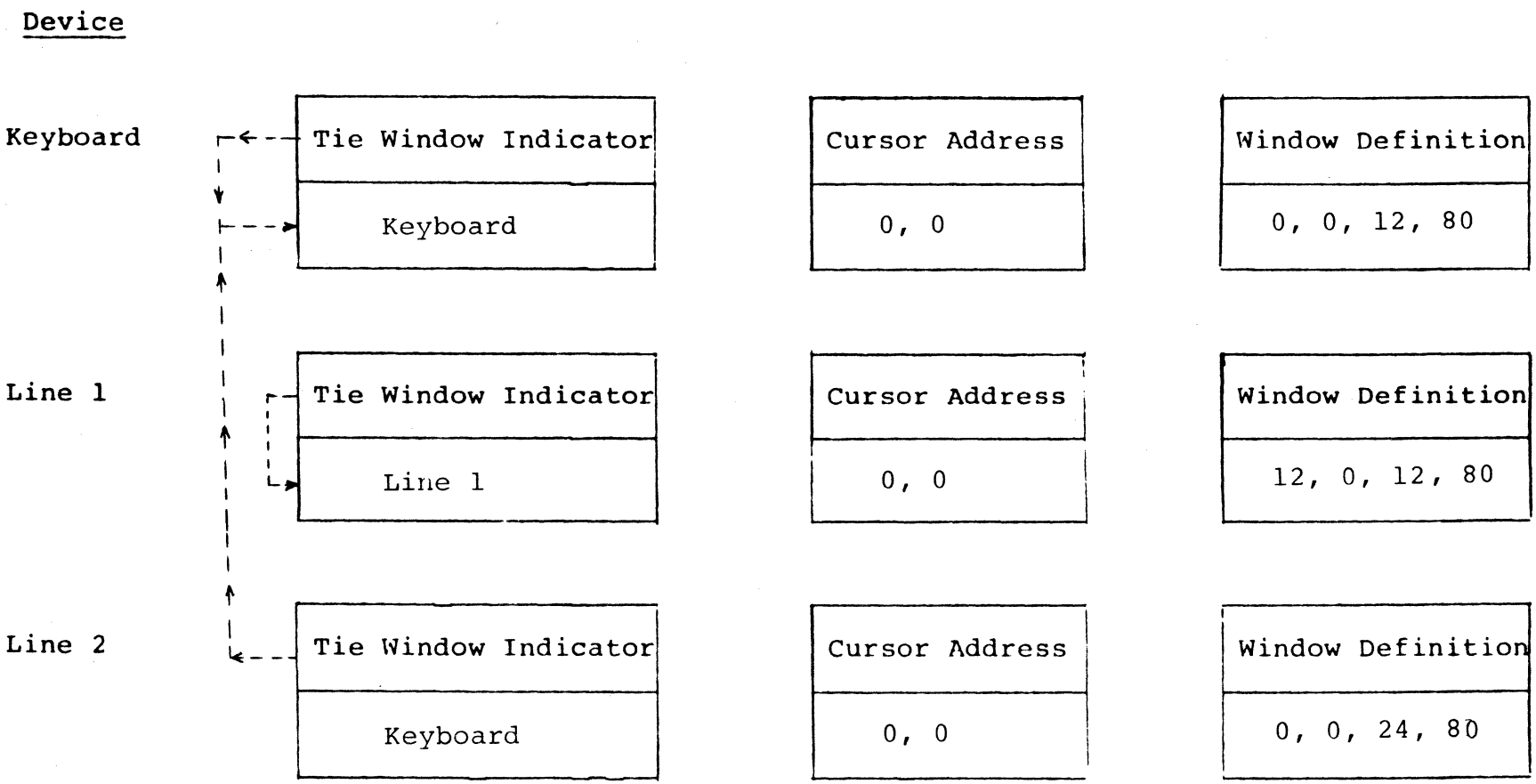


table structure after the Figure 2 escape sequences. Figure 6 shows the same structure after the recept of one character from the keyboard and one character from Line 1.

The user should be sure to note that many commands and functions, including Define Window, use the tie window indicator to determine which device table to change. That means, therefore, that a Define Window command received from Line 1 would normally change the keyboard window definition, unless Line 1 had previously been tied to itself.

Also, note that devices may only be tied to themselves or to the keyboard. The keyboard can only be attached to itself (not Line 1 or Line 2), and Line 1 cannot be tied to Line 2 and vice-versa.

Figure 6

SEPARATE WINDOWS AFTER RECEIPT OF ONE CHARACTER