INDEXB INDEX BYTE

Calling Sequence: SYSTEM INDEXB
or
SYSSUK INDEXB
DEFW (Base address)

Arguments: A =Displacement (Ø - 255)
HL=Base address of table

Output: A =Entry looked up
HL=Address of entry looked up

Notes:
INDEXB returns the byte at address
(Base address) + (Displacement)

SETB STORE BYTE

Calling Sequence:

```
SYSTEM  SETB
     or
SYSSUK  SETB
DEFB    (Value to store)
DEFW    (Address)
```

Arguments:

A =Byte value to store
HL=Address to be set

Description:

Stores an 8-bit value at a specified address.

SETW STORE WORD

Calling Sequence:

```
SYSTEM  SETW
     or
SYSSUK  SETW
DEFW    (Value to store)
DEFW    (Address)
```

Arguments:

DE=Word value to store
HL=Address to be set

Description:

Stores a 16-bit value at a specified address.

CASSETTE CONVENTIONS

Two types of cassettes may be used with the Bally Professional Arcade. The first type, called an autostart cassette, is entered immediately after reset. The only initialization that is performed before entry is the set-up of the stack pointer to point just below system RAM and the establishment of "consumer mode" in the custom chips. RAM is not altered in this mode.

The second type, called a standard cassette, is entered after a game selection process is completed. Considerably more initialization is done by the system before control transfer.

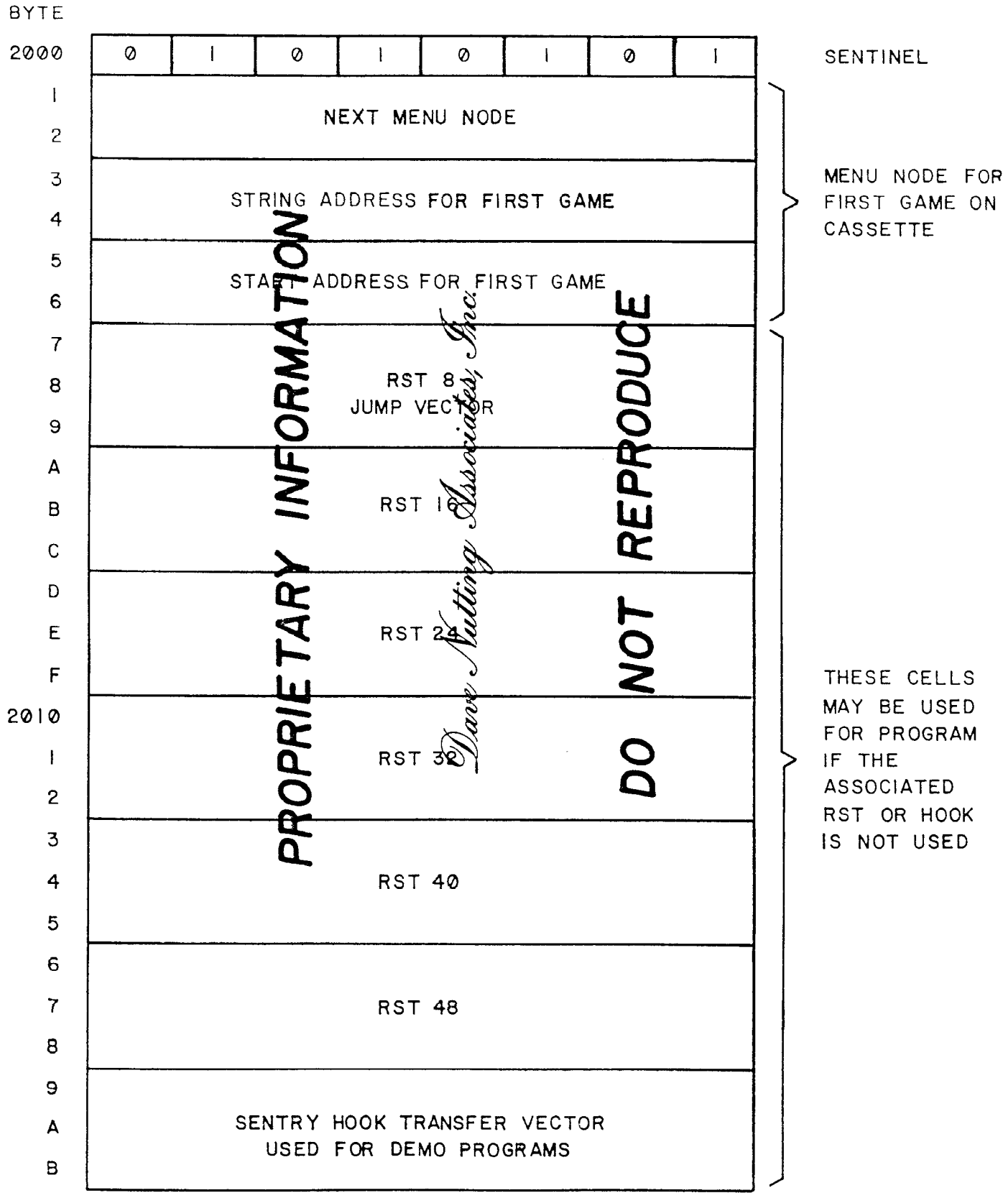
1) System RAM is cleared to Ø
2) The ACTIN interrupt routine is enabled
3) The MENU colors are set in the left color map
4) Vertical blank is set at line 96, horizontal boundary at 41, and interrupt mode at 8.
5) The screen displays the menu frame.
6) The shifter is cleared.

An autostart cassette is indicated by a jump instruction (opcode C3H) at location 2ØØØH. This jump instruction should branch to the starting address of the cassette.

A standard cassette is indicated by a sentinel byte of 55H at location 2ØØØH. Following this byte is the first node of the cassette's menu data structure. This data structure gives the name and starting address of each program in the cassette. (See MENU)

When the user has selected a cassette game, control is transferred to the starting address with the address of the program name string in the registers. The cassette program will use the GETPAR system routine to prompt for game parameters such as score to play to, game time limit or number of layers.

The cassette has access to the six unused restart instructions. See the following cassette diagram for the transfer vectors.


BYTE
2000
0 1 0 1 0 1 0 1
SENTINEL
1
2
NEXT MENU NODE
3
4
STRING ADDRESS FOR FIRST GAME
5
6
START ADDRESS FOR FIRST GAME
MENU NODE FOR FIRST GAME ON CASSETTE
7
8
9
RST 8
JUMP VECTOR
A
B
C
RST 16
D
E
F
RST 24
2010
1
2
RST 32
3
4
5
RST 40
6
7
8
RST 48
9
A
B
SENTRY HOOK TRANSFER VECTOR
USED FOR DEMO PROGRAMS
THESE CELLS MAY BE USED FOR PROGRAM IF THE ASSOCIATED RST OR HOOK IS NOT USED
PROPRIETARY INFORMATION
Dave Nutting Associates, Inc.
DO NOT REPRODUCE

HUMAN GETPAR

GET GAME PARAMETER

Calling Sequence:

```
SYSTEM  GETPAR
     or
SYSSUK  GETPAR
DEFW    (Prompt)
DEFB    (Digits)
DEFW    (Parameter)
```

Arguments:

A =Number of digits to get
BC=Address of prompt string
DE=Title string address *NOT LOADED
HL=Address of parameter to get

Description:

A menu frame is created displaying the title passed in DE at the top. The message "ENTER" is displayed in the center of the screen followed by the prompt string. GETNUM is entered with feedback specified in 2X enlarged characters. After entry is complete, GETPAR pauses for ¼ second to allow user to see his entry and then returns.

Notes:

See entry conditions and resource requirements for menu.
Prompt string example: "# OF PLAYERS"
The title string address (DE) is usually the title returned from MENU.
The address of parameter to get (HL), HL points at the low-order byte of BCD number in RAM.

HUMAN MENU

DISPLAY MENU AND BRANCH ON SELECTION

Calling Sequence:

```
SYSTEM  MENU
     or
SYSSUK  MENU
DEFW    (Title)
DEFW    (List)
```

Arguments: DE=Address of menu title string
HL=Address of menu list

Output: DE=String address of selection mode

Description:

The title is displayed at the top of the screen. Each entry in the menu list is then displayed with a preceding number supplied by MENU. GETNUM is called to get the selection number. The menu list is searched for the selected node and it is jumped to.

Notes:

A maximum of eight entries may appear.

On entry, MENU expects interrupts to be enabled, colors and boundaries to be set up. MENU uses 96 lines of screen, creates the alternate set, and requires three levels of context. MENU calls SENTRY and thus 'eats' all irrelevant transitions.

NEXT	ADDRESS OF NEXT NODE ON LIST ZERO IF THIS NODE IS LAST
STRING	ADDRESS OF NAME OF THIS SELECTION THIS IS WHAT IS PASSED IN DE
GO TO	WHERE TO BRANCH TO IF THIS SELECTION IS SELECTED

HUMAN GETNUM
GET NUMBER

Calling Sequence:

```
SYSTEM  GETNUM
     or
SYSSUK  GETNUM
DEFB    (X address)
DEFB    (Y address)
DEFB    (CHRDIS options)
DEFB    (DISNUM options)
DEFW    (Number address)
```

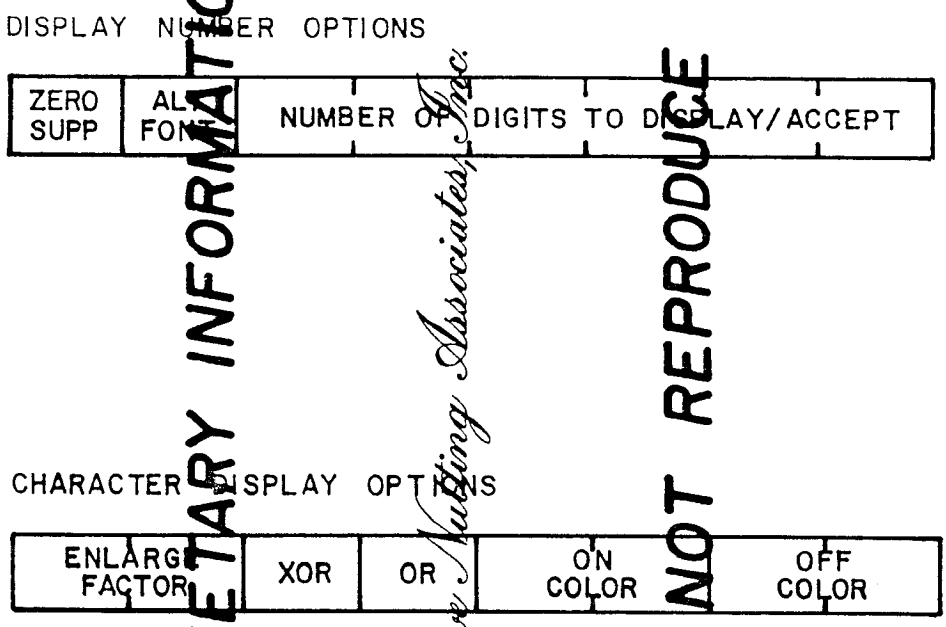
Arguments:

B =Display number routine options
C =Character display routine options
DE=Y,X co-ordinate for feedback
HL=Address of where to put entered number

Description:

This routine inputs a number from either the keypad or the pot on control handle of player one. Keypad entry has priority. The routine exits when the specified number of digits were entered or = is pressed on the keypad.

Pot entry is enabled by pressing the trigger. The current pot value is then shown. Twist the pot until the number you want is shown. Then press the trigger again to complete entry. The pot can only enter 1 or 2 digits. If a group of numbers is being entered, the user must enable entry for each new number.



HUMAN MSKTD

JOYSTICK MASK TO DELTAS

Calling Sequence:

```
SYSTEM  MSKTD
    or
SYSSUK  MSKTD
DEFW    (X Delta)
DEFB    (Flop-flag)
DEFW    (Y Delta)
```

Arguments:

```
B =Joystick mask
C =Flop flag
DE=X positive delta
HL=Y positive delta
```

*NOT LOADED

Output:

```
DE=X Delta
HL=Y Delta
```

Description:

This routine uses the joystick mask and flop flag to conditionally modify the passed deltas. If negative direction is indicated, the delta is 2's complemented; if no direction is indicated, Ø is returned.

Note: B is not sucked.

MATH RANGED

RANGED RANDOM NUMBER

Calling Sequence: SYSTEM RANGED

or

SYSSUK RANGED

DEFB (N)

Arguments: A=N where Ø is less than or equal to a random numner less than N

(ie: for a random number of Ø,1,or 2, N=3)

Output: A=Random Number

Notes;

If N is a power of 2 it is considerably faster to use N=Ø which causes an 8-bit value to be returned without ranging. Use an AND instruction to range it yourself.

This routine uses a polynomial shift register RANSHT in system RAM. RANGED is called in GETNUM while waiting for game selection/parameter entry. Thus each execution of a program will receive different random numbers. For 'predictable' random numbers, alter RANSHT yourself after parameter acceptance.

INTRODUCTION

The Bally Professional Arcade is a full-color video game system based on the mass-ram-buffer technique. A mass-ram-buffer system is one in which one or more bits of RAM are used to define the color and intensity of a pixel on the screen. The picture on the screen is defined by the contents of RAM and can easily be changed by modifying RAM.

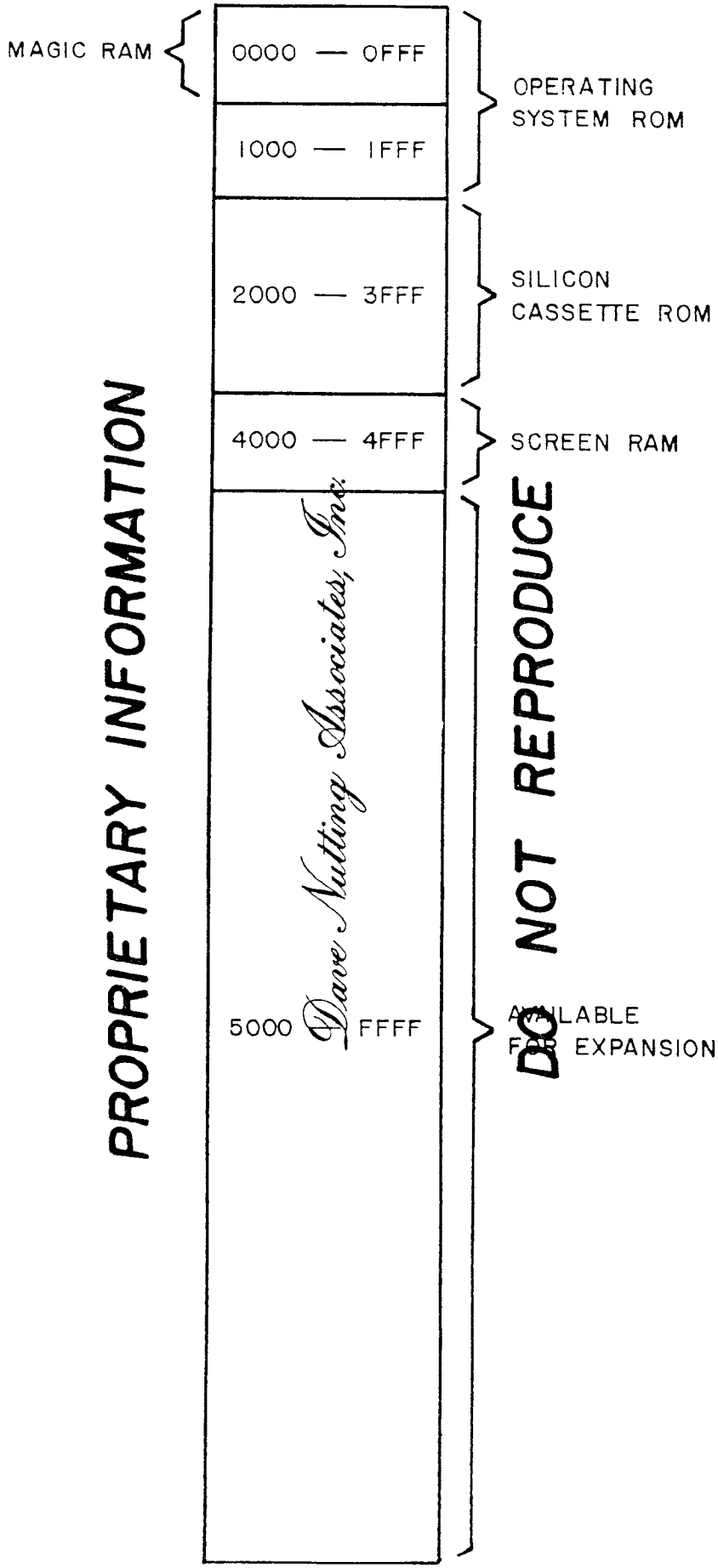
The system uses a Z80 Microprocessor as it's main control unit. The system ROM has software for four games: Gunfight, Checkmate, Scribbling, and Calculator. Additional ROM can be accessed through the silicon cassette connector. Three custom chips are used for the video interface, special video processing functions, keyboard and control handle interface, and audio generation.

The system exists in both high-resolution and low-resolution models. The three custom chips can operate in either mode. The mode of operation is determined by bit Ø of output port 8H. It must be set to Ø for low-resolution and 1 for high-resolution. This bit is not set to Ø at power up and must be set by software before any RAM operations can be performed.

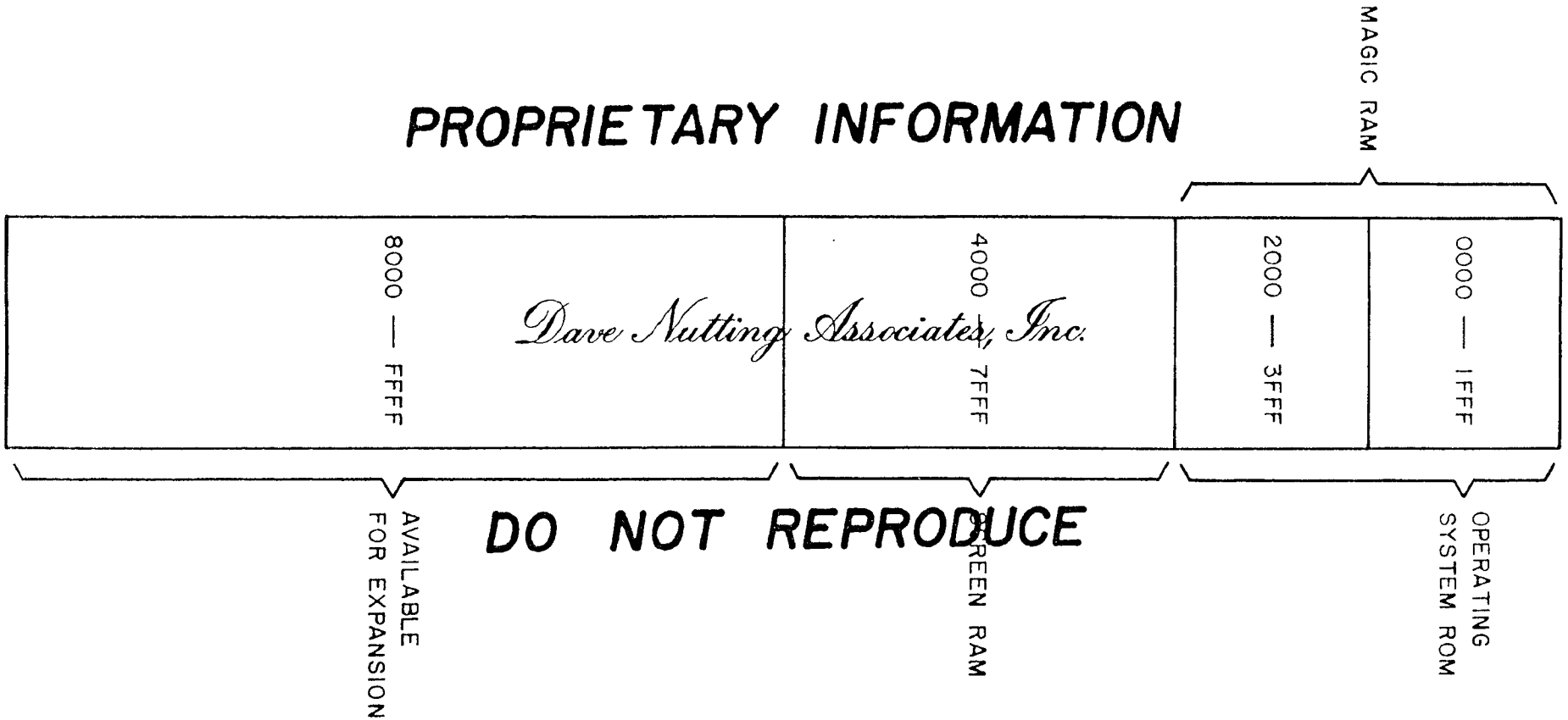
MEMORY MAP

In both the low and high resolution models, the operating system ROM is in the first 8K of memory space. The silicon cassette ROM is in the space from 8K to 16K. The standard screen RAM begins at 16K. In the low-resolution unit, standard screen RAM is 4K bytes; in the high-resolution unit it is 16K bytes. Magic screen RAM begins at location Ø. It is the same size as standard screen RAM. All memory above 32K is available for expansion. In the low-resolution unit, memory space 20K - 32K is available for expansion.

When data is read from a memory location between Ø and 16K the data comes from the ROM. When data is written in a memory location (X) between Ø and 16K, the system actually writes a modified from of the data in location X+16K. The modification is performed by the magic system in the Data Chip and Address Chip. Thus the RAM from 0 to 16K is called Magic Memory.



MEMORY MAP LOW RESOLUTION



MEMORY MAP HIGH RESOLUTION

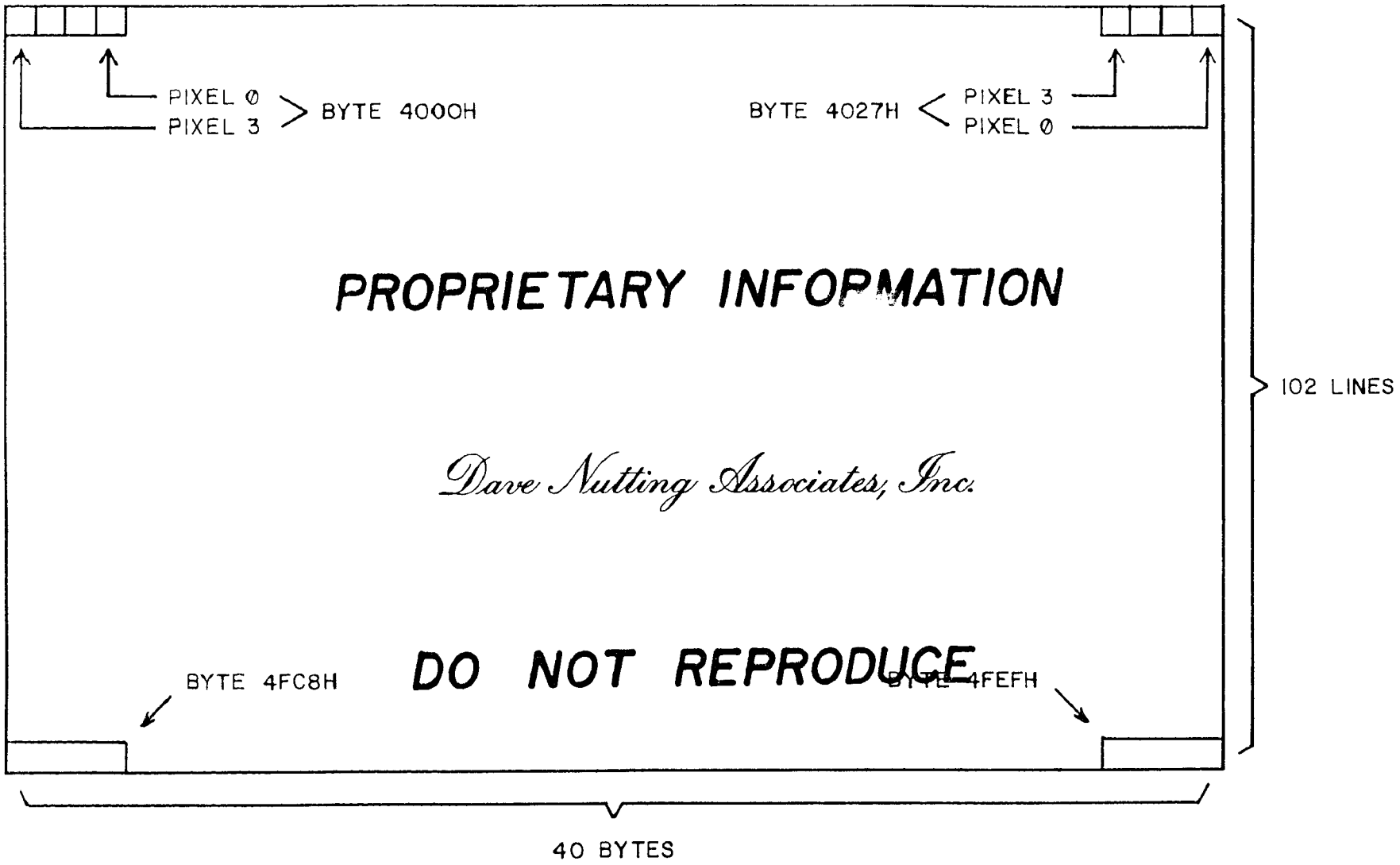
SCREEN MAP

In the Bally Professional Arcade, two bits of RAM are used to define a pixel on the screen. One 8-bit byte of RAM therefor defines four pixels on the screen.

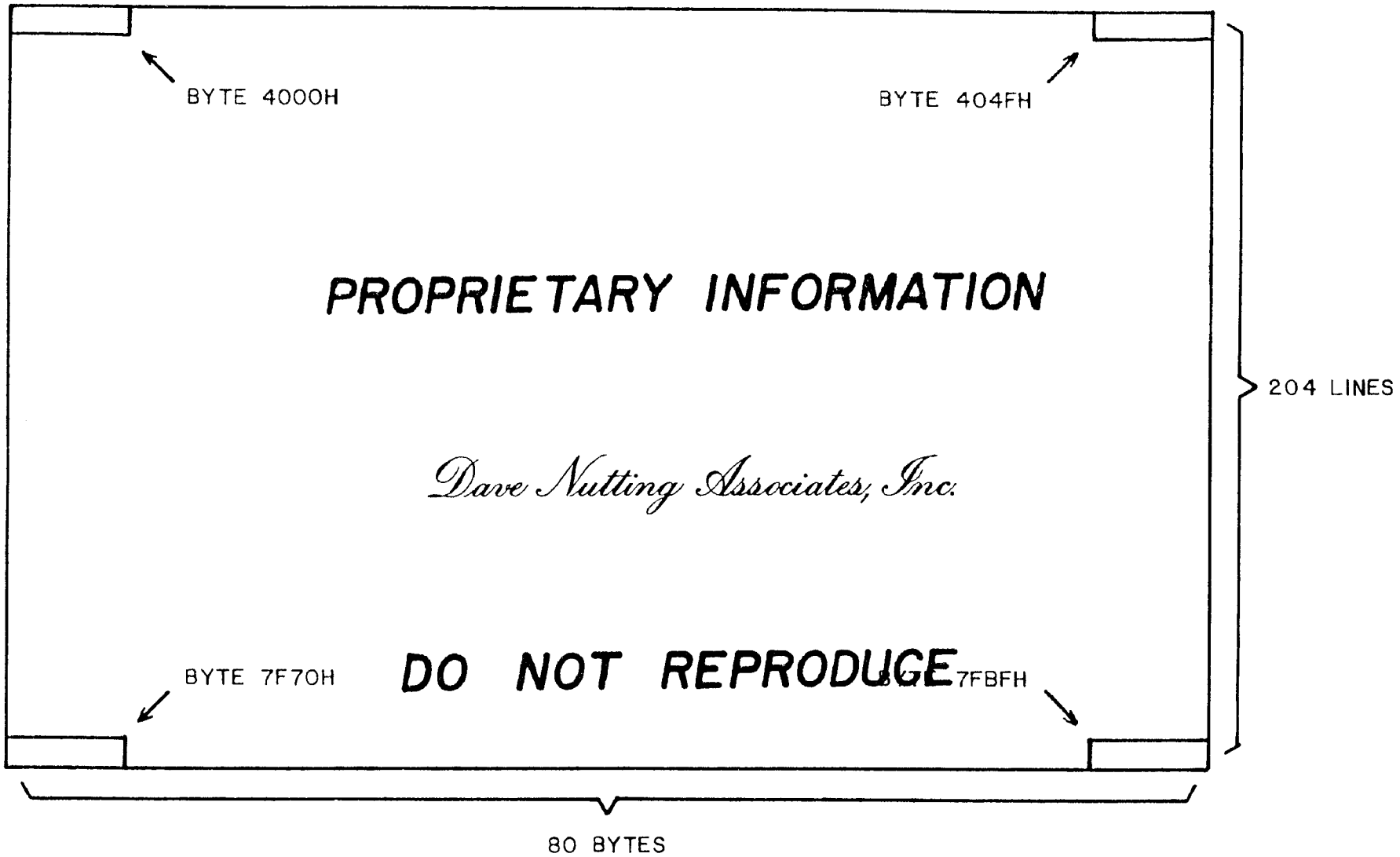
In the low-resolution model there are 40 bytes used to define a line of data. This gives a horizontal resolution of 160 pixels. The vertical resolution is 102 lines. The screen therefor requires 102 X 40 = 4,080 bytes. The remaining 16 bytes of the 4K RAM are used for scratch pad. More of the RAM can be used for scratchpad by blanking the screen before the 102nd line. This will be described later.

In the high-resolution model there are 80 bytes and 320 pixels per line. The 204 lines require 16,320 bytes of RAM. 64 bytes of the 16K RAM are left for scratch pad.

In both models the first byte of RAM is in the upper left-hand corner of the screen. As the RAM address increases, the position on the screen moves in the same directions as the TV scan; from left-to-right and from top-to-bottom. The four pixels in each byte are displayed with the least significant pixel, the one defined by bits Ø and 1, on the right.



SCREEN MAP LOW RESOLUTION



SCREEN MAP HIGH RESOLUTION

COLOR MAPPING

Two bits are used to represent each pixel on the screen. These two bits, along with the LEFT/RIGHT bit which is set by crossing the horizontal color boundary, map each pixel to one of eight different color registers. The value in the color register then defines the color and intensity of the pixel on the screen. The intensity of the pixel is defined by the three least significant bits of the register, ØØØ for darkest and 111 for lightest. The color is defined by the five most significant bits. The color registers are at output ports Ø through 7; register Ø at port Ø, register 1 at port 1, etc.

The color registers can be accessed as individual ports or all eight can be accessed by the OTIR instruction. The OTIR instruction is to port BH (register =BH) and register B should be set to 8. The eight bytes of data pointed to by HL will go to the color registers

HL → Memory Location X Color Register 7
X+1 Color Register 6
X+2 Color Register 5
X+3 Color Register 4
X+4 Color Register 3
X+5 Color Register 2
X+6 Color Register 1
X+7 Color Register Ø

The horizontal color boundary (bits Ø-5 of port 9) defines the horizontal position of an imaginary vertical line on the screen. The boundary line can be position between any two adjacent bytes in the low-resolution system. The line is immediately to the left of the byte whose number is sent to bits Ø-5 of port 9. For example, if the horizontal color boundary is set to Ø, the line will be just to the left of byte Ø; if it is set to 20, the line will be between bytes 19 and 20 in the center of the screen.

If a pixel is to the left of the boundary, its LEFT/RIGHT bit is set to 1. The LEFT/RIGHT bit is set to Ø for pixels to the right of the boundary. Color registers Ø-3 are used for pixels to the right of the boundary and registers 4-7 are used for pixels to the left of the boundary.

In the high-resolution system, the boundary is placed in the same position on the screen but between different bytes. If the value X is sent to the horizontal color boundary, then the boundary will be between bytes 2X and 2X-1. If the value 20 is sent, the boundary will be between 39 and 40, in the center of the screen.

To put the entire screen, including the right side background, on the left side of the boundary, set the horizontal color boundary to 44.

BACKGROUND COLOR

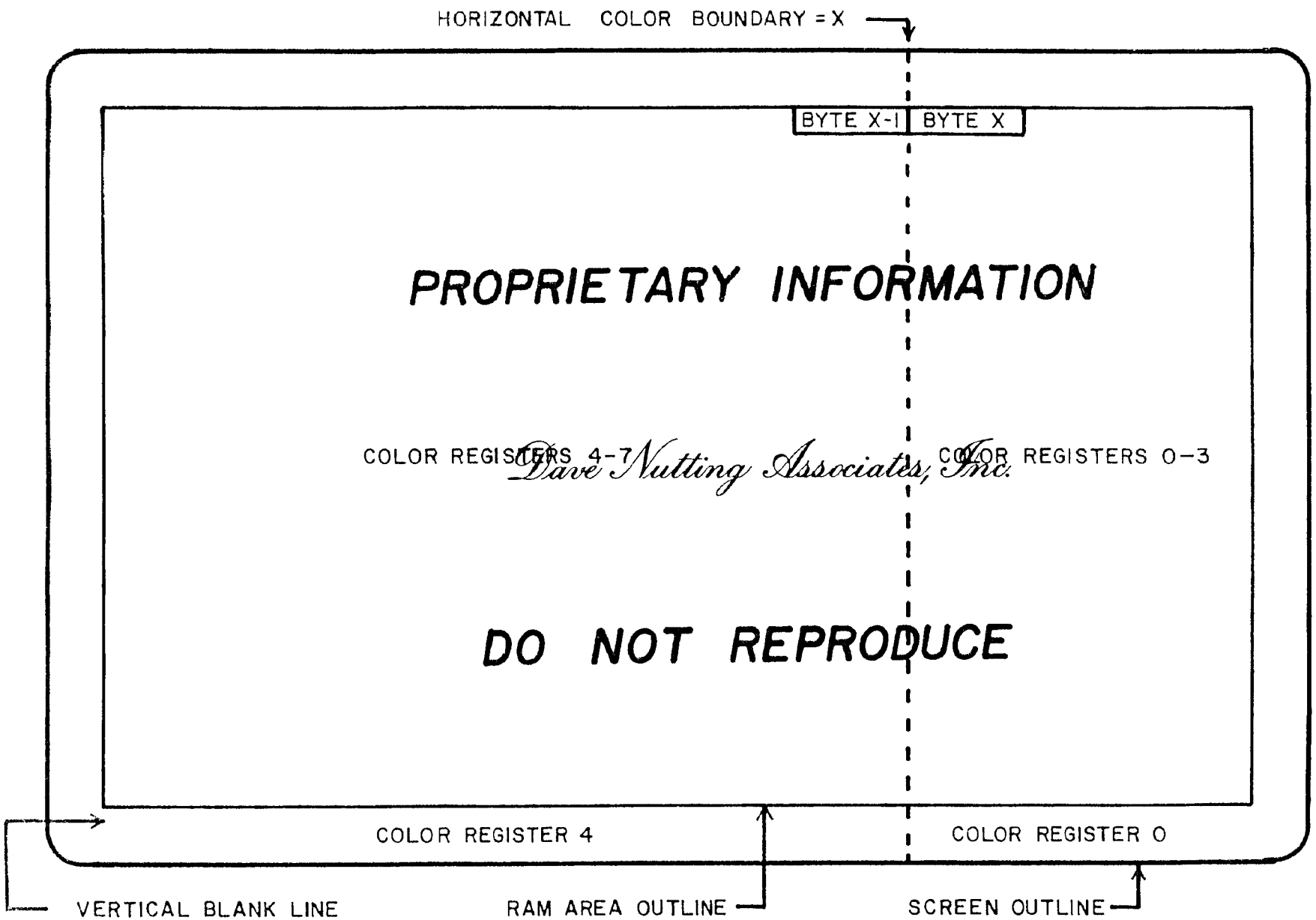
On most televisions the area defined by RAM is slightly smaller than the screen. There is generally extra space on all four sides of the RAM area. The color and intensity of this area is defined by the background color number (bits 6 and 7 of port 9). These two bits, along with the LEFT/RIGHT bit point to one of the color registers which determines the color and intensity.

VERTICAL BLANK

The Vertical Blank Register (output port AH) contains the line number on which vertical blanking will begin. In the low-resolution system bit Ø should be set to Ø and the line number should be in bits 1-7. In the high-resolution system the line number is in bits Ø-7. The background color will be displayed from the vertical blank line to the bottom of the screen. This allows the RAM that would normally be displayed in that area to be used for scratch pad. If the vertical blank register is set to Ø the entire RAM can be used for scratch pad. In a low-resolution system the register must be set to 101 or less; in a high-resolution system it must be set to 2Ø1 or less.

SUMMARY

The following color register map shows which color registers are used to define colors in different areas of the screen. The map assumes the background color is set to Ø. If it were set to 1 then color registers 1 and 5 would be used for background instead of Ø and 4. In the low-resolution system the color boundary is between bytes X and X-1. In the high-resolution system the boundary is between bytes 2X and 2X-1.



COLOR REGISTER MAP

INTERRUPT FEEDBACK

When the Z-80 acknowledges an interrupt it reads 8 bits of data from the data bus. It then uses this data as an instruction or an address. In the Bally Professional Arcade this data is determined by the contents of the interrupt feedback register (output port DH). In responding to a screen interrupt the contents of the interrupt feedback register are placed directly on the data bus. In responding to a light pen interrupt the lower four bits of the data bus are set to Ø and the upper four bits are the same as the corresponding bits of the feedback register.

INTERRUPT CONTROL BITS

In order for the Z-80 to be interrupted the internal interrupt enable flip-flop must be set by an EI instruction and bit one or two of the external interrupt enable bits must be set output port DH. If bit 1 is set, light pen interrupts can occur. If bit 3 is set, screen interrupts can occur. If both bits are set, both interrupts can occur and the screen interrupt has higher priority.

The interrupt mode bits determine what happens if an interrupt occurs when the Z-80's interrupt enable flip-flop is not set. Each of the two interrupts may have a different mode. In mode 0 the Z-80 will continue to be interrupted until it finally enables interrupts and acknowledges the interrupt. In mode 1 the interrupt will be discarded if it is not acknowledged by the next instruction after it occured. If mode 1 is used the software must be designed such that the system will not be executing certain Z-80 instructions when the interrupt occurs. The opcodes of these instructions begin with CBH, DDH, EDH, and FDH.

The mode bit for light pen interrupt is bit Ø of port EH and the mode bit for screen interrupt is bit 2 of port EH.

SCREEN INTERRUPT

The purpose of the screen interrupt is to synchronize the software with the video system. The software must send a line number to the interrupt line register (output port FH). In the low-resolution system bit Ø is set to Ø and the line number is sent to bits 1-7. In the high-resolution system the line number is sent to bits 0-7. If the screen interrupt enable bit is set, the Z-80 will be interrupted when the video system completes scanning the line in the interrupt register. This interrupt can be used for timing since each line is scanned 60 times a second. It can also be used in conjunction with the color registers to make as many as 256 color-intensity combinations appear on the screen at the same time.

LIGHT PEN INTERRUPT

The light pen interrupt occurs when the light pen trigger is pressed and the video scan crosses the point on the screen where the light pen is. The interrupt routine can read two registers to determine the position of the light pen. The line number is read from the vertical feedback register (input port EH). In the high-resolution system the line number is in bits Ø-7. In the low-resolution system the line number is in bits 1-7, bit Ø should be ignored. The horizontal position of the light pen can be determined by reading input port FH and subtracting 8. In the low-resolution system the resultant value is the pixel number, Ø to 159. In the high-resolution system the resultant must be multiplied by two to give the pixel number, Ø to 358.

MAGIC REGISTER

As described earlier, the Magic System is enable when data is written to a memory location (X) from Ø to 16K. A modified form of the data is actually written in memory location X+16K. The magic register (output port CH) determines how the data is modified. The purpose of each bit of the magic register is shown below.

Bit Ø	LSB of shift amount
1	MSB of shift amount
2	Rotate
3	Expand
4	OR
5	XOR
6	Flop

The order is which magic functions are performed is as follows: Expansion is done first; rotating or shifting; flopping; OR or XOR. As many as four can be used at any one time and any function can be bypassed. Rotate and shift as well as OR and XOR cannot be done at the same time.

EXPAND

The expander is used to expand the 8 bit data bus into 8 pixels (or 16 bits). It expands a Ø on the data bus into a two-bit pixel and a 1 into another two-bit pixel. Thus, two-color patterns can be stored in ROM in half the normal memory space.

During each memory write instruction using the expander, either the upper half or the lower half of the data bus is expanded. The half used is determined by the expand flip-flop. The flip-flop is reset by an output to the magic register and is toggled after each magic memory write. The upper half of the data bus is expanded when the flip-flop is Ø, and the lower half when the flip-flop is 1.

The expand register (output port 09H) determines the pixel values into which the data bus will be expanded. A Ø on the data bus will be expanded into the pixel defined by bits Ø and 1 of the expand register. A 1 on the data bus will be expanded into the pixel defined by bits 2 and 3 of the expand register.

The pixels generated by bit Ø or 4 of the data bus will be the least significant pixel of the expanded byte. The most significant pixel will come from bit 3 or 7.

SHIFTER

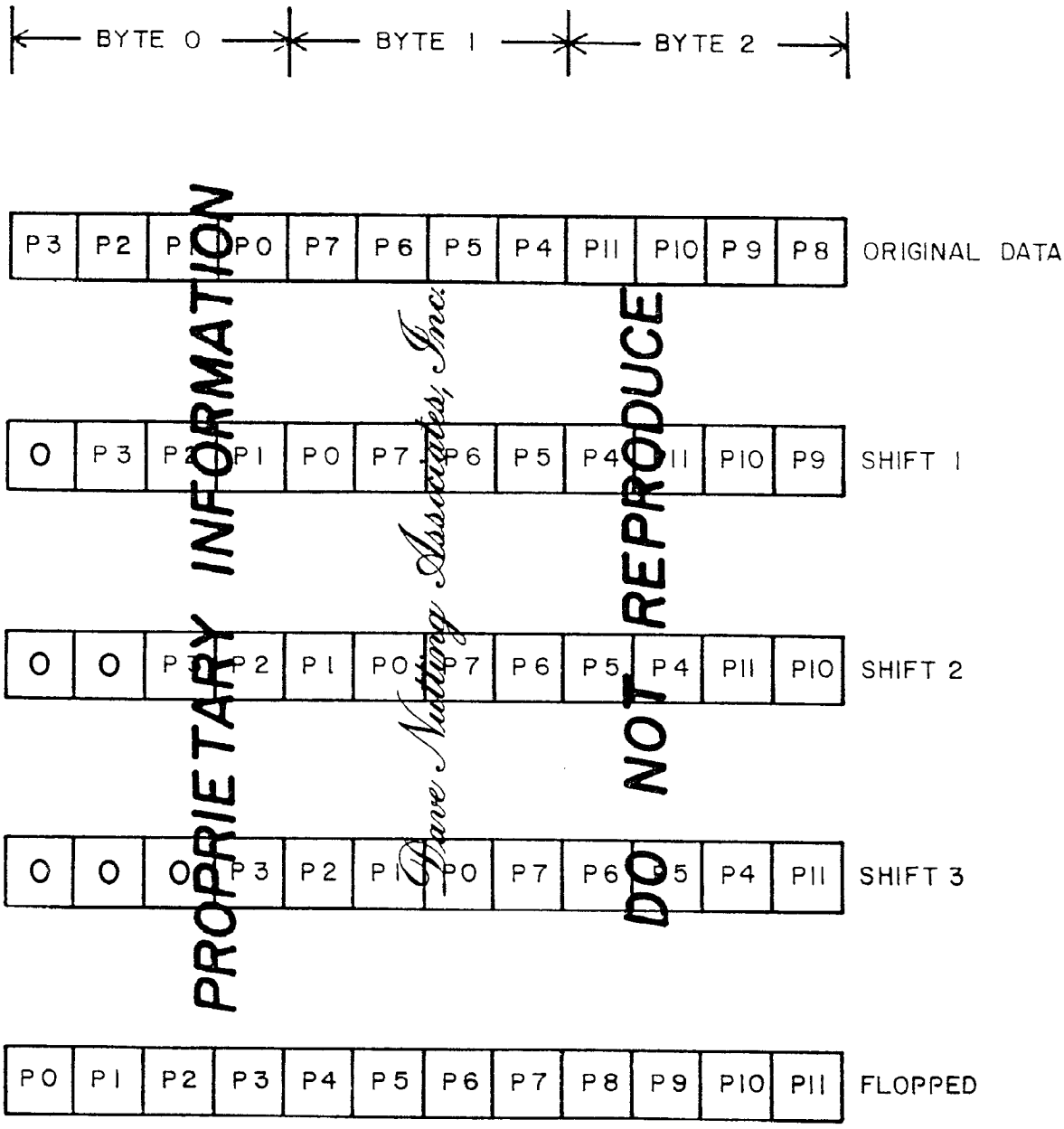
The shifter, flopper, and rotator operate on pixels rather than bits. Each byte is thought of as containing four pixels, each of which has one of four values. The four pixels are referred to as PØ, P1, P2, and P3. PØ is composed of the first two bits of the byte.

The shifter shifts data Ø, 1, 2, or 3 pixels to the right. The shift amount is determined by bits Ø and 1 of the magic register. The pixels that are shifted out of one byte are shifted into the next byte. Ø's are shifted into the first byte of a sequence. The shifter assumes the first byte of a sequence is the first magic memory write after an output to the magic register. Each sequence must be initialized by an output to the magic register and data cannot be sent to the magic register in the middle of a sequence.

FLOPPER

The output of the flopper is a mirror image of it's input. Pixel Ø and 3 exchange values as do pixel 1 and 2.

The diagrams on the following page show examples of shifting and flopping.



SHIFTER — FLOPPER

ROTATOR

The rotator is used to rotate a 4 X 4 pixel image 90^{o} in a clockwise direction. The rotator is initialized by an output to the magic register and will re-initialize itself after every eight writes to magic memory. To perform a rotation, the following procedure must be performed twice. Write the top byte of the unrotated image to a location in magic memory. Write the next byte to the first location plus 80, the next byte to the first location plus 160, and the last byte to the first location plus 240. After eight writes the data will appear in RAM and on the screen rotated 90^{o} from the original image.

The rotator can only be used in commercial mode.

The diagram on the following page shows an example of rotating.

ROTATOR

OR AND XOR

These functions operate on a byte as 8-bits rather than four pixels. When the OR function is used in writing data to RAM, the input to the OR circuit is ORed with the contents of the RAM location being accessed. The resultant is then written in RAM.

The XOR function operates in the same way except that the data is XORed instead of ORed.

INTERCEPT

Software reads the intercept register (input port 8H) to determine if an intercept occurred on an OR or XOR write. An intercept is defined as the writing of a non-zero pixel in a pixel location that previously contained a non-zero pixel. A non-zero pixel is a pixel with a value of Ø1, 1Ø, or 11. A 1 in the intercept register means an intercept has occured. Bits Ø - 3 give the intercept information for all OR or XOR writes since the last input from the intercept register. An input from the intercept register resets these bits. A bit is set to 1 if an intercept occurs in the appropriate position and will not be reset until after the next intercept register input.

Bit	
Ø	Intercept in pixel 3 in an OR or XOR write since last reset
1	Intercept in pixel 2 in an OR or XOR write since last reset
2	Intercept in pixel 1 in an OR or XOR write since last reset
3	Intercept in pixel Ø in an OR or XOR write since last reset
4	Intercept in pixel 3 in last OR or XOR write
5	Intercept in pixel 2 in last OR or XOR write
6	Intercept in pixel 1 in last OR or XOR write
7	Intercept in pixel Ø in last OR or XOR write

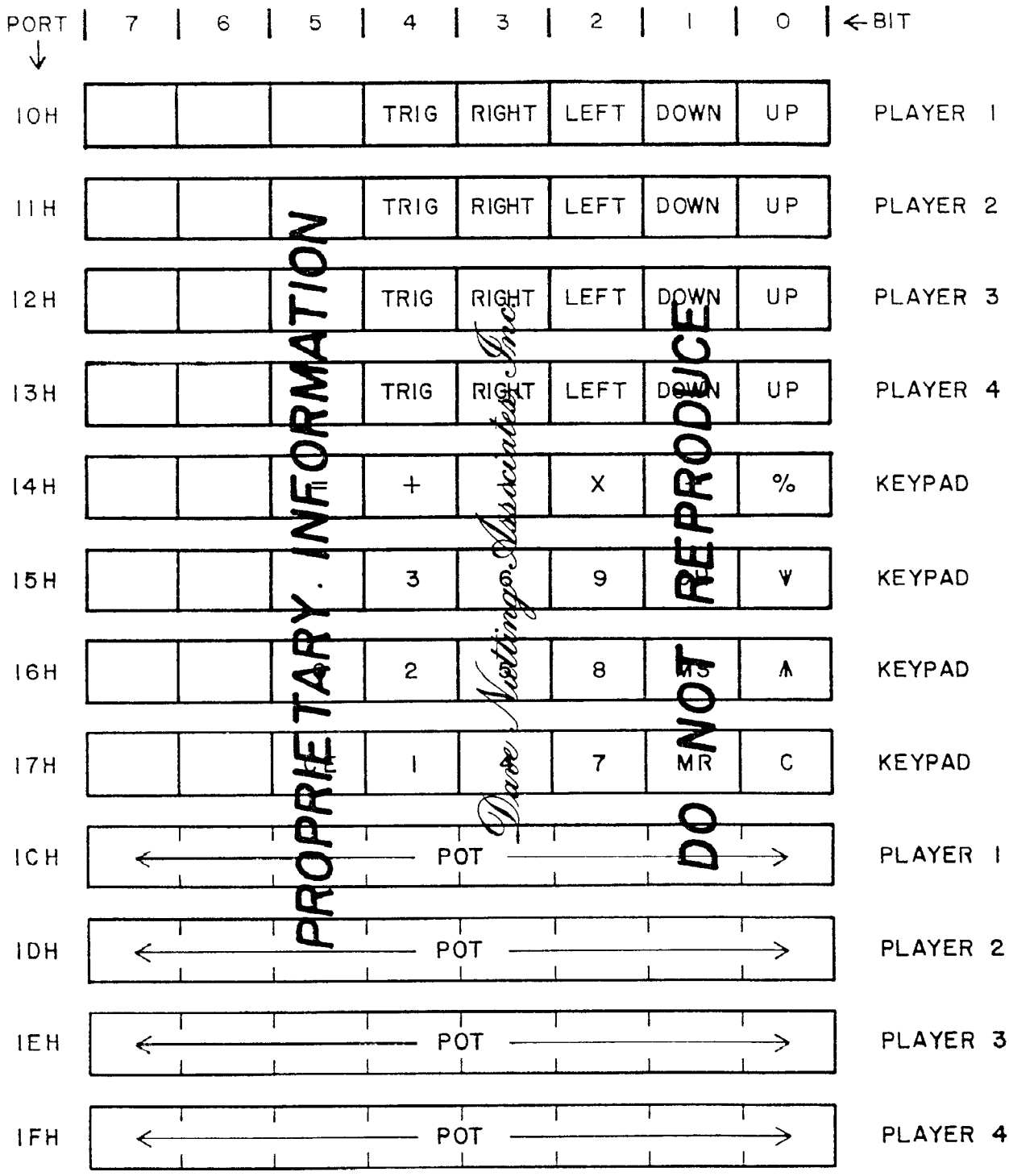
PLAYER INPUT

The system will accomodate up to four player control handles at once. Each handle has five switches and a potentiometer. The switches are read by the Z-80 on input ports 10H - 13H and are not debounced. The switches are normally open and normally feedback Ø's.

The signals from the potentiometers are changed to digital information by an 8-bit Analog-to-Digital Convertor. The four pots are on input ports 1CH - 1FH. All Ø's are fedback when the pot is turned fully counter-clockwise and all 1's when turned fully clockwise.

The 24-button keypad is read on bits Ø-5 of ports 14H-17H. The data is normally Ø and if more than one button is depressed, the data should be ignored. The keypad will not send back the proper data if any of the player control switches are closed. Here again, the buttons are not debounced.

Player control inputs are shown on the following page.



PLAYER INPUT