THIS DOCUMENT AND ITS CONTENTS ARE THE PROPERTY OF DAVE NUTTING ASSOCIATES, INCORPORATED AND BALLY MANUFACTURING CORPORATION. THE INFORMATION CONTAINED HEREIN IS BOTH PROPRIETARY AND CONFIDENTIAL.

NO PART OF THIS DOCUMENT MAY BE REPRODUCED, STORED IN A RETRIEVAL SYSTEM, OR TRANSMITTED IN ANY FORM OR BY ANY MEANS ELECTRONIC, MECHANICAL, CHEMICAL, PHOTOGRAPHICAL, RECORDING, PHOTOCOPYING OR OTHERWISE.

DAVE NUTTING ASSOCIATES, INCORPORATED ASSUMES NO RESPONSIBILITY FOR THE USE OF ANY CIRCUITRY OTHER THAN CIRCUITRY EMBODIED IN A DAVE NUTTING ASSOCIATES, INCORPORATED DESIGNED PRODUCT.

THIS DOCUMENT MUST BE RETURNED TO DAVE NUTTING ASSOCIATES, INCORPORATED BY REGISTERED MAIL WITHIN 5 DAYS UPON WRITTEN DEMAND.

TABLE OF CONTENTS - SOFTWARE

1		Home Video Game System
2		User Program Interface
5		System Routine Conventions
7		Inline Argument Mask Table Entry
8	INTPC	Begin Interpreting
9	XINTC	Exit Interpreter
1Ø	RCALL	Call Assembly Language Subroutine
11	MCALL	Call Interpreter Subroutine
12	MJUMP	Interpreter Jump
13	MRET	Return From Interpretive Subroutines
14		Screen Handler
15	SETOUT	Set Display Ports
16	FILL	Fill A Contiguous Area With Constant
17	RECTAN	Paint A Rectangle
18		Screen Write Routines
19		Standard Calling Sequence
2Ø		Pattern Representation
21	VWRITR	Write Relative From Vector
22	WRITR	Write Relative
23	WRITP	Write With Pattern Size Scare Up
24	WRIT	Write Pattern
25	WRITA	Write Absolute
26	SAVE	Save Area
27	RESTOR	Restore Area
28	VBLANK	Blank From Vector
29	BLANK	Blank Area
3Ø	SCROLL	Scroll Window

31		Screen Alphanumeric Display Routines
34	DISNUM	Display BCD Number
35	DISTIM	Display Time
36	CHRDIS	Display Character
37	STRDIS	Display String
38		STRDIS Interpretation of Codes 64H to 7FH
39		Screen Vectoring - Vectoring Routines
42	VECT	Vector Object In Two Dimensions
43	VECTC	Vector A Co-ordinate
44	RELABS	Convert Relative Co-ordinates
45	RELAB1	Convert Relative Address To Absolute
46	COLSET	Set Color Registers
47	INCSCR	Increment Score And Compare To End Score
48	PAWS	Pause
49	KCTASC	Key Code to ASCII
5Ø	SENTRY	Sense Transition
53	DOIT	Respond To Input Transition
54	PIZBRK	Coffee Break, Black Out Screen, Wait For Key
55		Example
56		Interrupt - Music Processor
57		MUZCPU Instruction Set
58		Music Score Example
59	BMUSIC	Begin Playing Music
6Ø	EMUSIC	Stop Music
61	ACTINT	Active Interrupts
62	DECCTS	Decrement Counter/Timers
63	CTIMER	

64	STIMER	Decrement Timers
65	MOVE	Move Bytes
66	INDEXN	Index Nibble
67	STOREN	Store Nibble
68	INDEXW	Index Word
69	INDEXB	Index Byte
7Ø	SETB	Store Byte
71	SETW	Store Word
72		Cassette Conventions
75	GETPAR	Get Game Parameter
76	MENU	Display Menu And Branch On Selection
77	GETNUM	Get Number
79	MSKTD	Joystick Mask To Deltas
8Ø	RANGED	Ranged Random Number

TABLE OF CONTENTS - HARDWARE

81 Introduction
82 Memory Map
85 Screen Map
88 Color Mapping
89 Background Color
9Ø Vertical Blank
92 Interrupt Feedback
92 Interrupt Control Bits
93 Screen Interrupt
93 Light Pen Interrupt
94 Magic Register
95 Expand
96 Shifter
96 Flopper
98 Rotator
1ØØ OR And XOR
1ØØ Intercept
1Ø1 Player Input
1Ø3 Master Oscillator
1Ø4 Tones
1Ø4 Sound Block Transfer
1Ø6 Output Ports
1Ø7 Input Ports

1Ø9 Microcycler

111 Address Chip Description

114 Data Chip Description

117 I/O Chip Description

119 Music Processor

123 Custom Chip Timing

131 Video Timing

135 Electrical Specifications for Midway Custom Circuits

LIST OF ILLUSTRATIONS

6 Context Block Format

2Ø Pattern Representation

32 Option Byte

33 Alternate Font Descriptor

4Ø Vector Block

41 Vector Status Detail

41 Checks Mask Detail

44 Normal and Flopped Co-ordinate Systems

51 Keypad Mask Configuration

56 Voices Status Register

66 INDEXN

68 INDEXW

74 Cassette Map

78 Display Number Options

78 Character Display Options

83 Memory Map Low Resolution

84 Memory Map High Resolution

86 Screen Map Low Resolution

87 Screen Map High Resolution

91 Color Register Map

97 Shifter - Flopper

99 Rotator

1Ø2 Player Input

1Ø5 Audio Generator Block Diagram

1Ø6 Output Ports

1Ø7 Input Ports

1Ø8 System Block Diagram

11Ø Microcycler Block Diagram

113 Address Chip Block Diagram

116 Data Chip Block Diagram

118 I/O Chip Block Diagram

121 Master Oscillator

122 Tone Generators

124 Memory Write Without Extra Wait State

125 Memory Write With Video Wait State

126 Memory Read Without Extra Wait State

127 Memory Read With Video Wait State

128 I/O Read From Port 1ØH - 17H

129 I/O Read From Other Than Port 1ØH - 17H

13Ø I/O Write

132 Relationship Between 7M, Horiz Dr, Vert Dr, $\overline{\Phi}$G, $\overline{\text{PX}}$, and RAS

133 Relationship Between Horiz Dr, Horiz Blank, Horiz Sync, and Color Burst

134 Relationship Between Vertical Sync, Vertical Blank, and Vertical Drive

HOME VIDEO GAME SYSTEM

This documentation describes the Bally Home Video Game System. The description begins with a discussion of the major sub-sections of the system. Following this, each sub-section is presented in greater detail, with detailed particulars, such as calling sequences and resource use.

The major sub-sections of the system are:

The User Program Interface...which allows cassettes to reference the system routines through a standard interface. Includes an interpreter.

The Screen Handler...a complex of routines for creating screen images. Includes facilities for initialization, pattern, and character display, co-ordinate conversion, and object vectoring.

The Interrupt Processor...decrements timers, plays music, and produces sounds.

The Human Interface...reads keypad and control handles, inputs game selection and options.

Math Routines...a package of routines for manipulating floating BCD numbers.

USER PROGRAM INTERFACE

The User Program Interface (UPI) is a set of procedures and conventions, which are utilized by a cassette program to access the facilities provided by the home video game system. By adhering to these conventions a cassette program will be system independent, thus allowing improvements to be made to later versions of the system and on-board games, while maintaining upward compatability.

The basic rule for using the UPI is:

> With exception to the system DOPE vector, no cassette should ever address system ROM directly, or expect a given cell to always equal a certain value

The mechanism for calling a system routine is:

```
        RST     56
        DEFB    (routine # + option)
```

where routine number is an even number specifying which sub-routine to transfer to, symbolic identifiers, which are equated to routine numbers, are provided in HVGLIB.

Option is used to specify how arguments are being passed to the system routine. If option equals zero, the arguments are presumed to exist in CPU registers; if option equals 1, the arguments are taken to follow in line after the routine number/option byte. These arguments are loaded into the CPU registers automatically before the called routine is entered. The arguments required by each system routine are given in the routine's detail documentation.

The SYSTEM macro generates the sequence previously mentioned with option = Ø:

```
        SYSTEM  (routine #)
```

(example)

```
        SYSTEM  FILL
```

The SYSSUK macro generates the sequence previously mentioned with option = 1:

```
        SYSSUK  (routine #)
```

Frequently it is desirable to string several system routine calls together. If four or more calls follow in sequence, it is more efficient to utilize the interpreter. By using the interpreter we void the overhead of the RST 56 instruction by expecting a call index to immediately follow the call index or arguments used by the previous system routine.

Special call indexes are used to enter and exit interpretive mode:

Example:

```
        SYSTEM  INTPC             ;BEGIN INTERPRETING
        DO      FILL              ;DO FILL ROUTINE
        DEFW    NORMEM            ;STARTING AT TOP OF SCREEN
        DEFW    92*BYTEPL         ;CONTINUING FOR 92 LINES
        DEFB    Ø                 ;FILLED WITH ZEROES
        DO      CHRDIS            ;DO CHARACTER DISPLAY ROUTINE
        DEFB    Ø                 ;Y-AXIS POSITION OF CHARACTER
        DEFB    1Ø                ;X-AXIS POSITION OF CHARACTER
        DEFB    8                 ;OPTIONS-PLOP,1Ø-ON,ØØ-OFF
        DEFB    'A'               ;CHARACTER TO BE DISPLAYED
        EXIT                      ;EXIT INTERPRETER
```

A block of call indexes have been set aside for the internal use of cassette programs. If a negative call index is encountered, the user's macro routine address table and argument table are utilized. The user is responsible for storing the addresses of these tables into dedicated system RAM cells.

All UPI routines are re-entrant.

Registers which are not defined as containing output parameters will not change.

SYSTEM ROUTINE CONVENTIONS

A system routine is coded like a conventional machine language subroutine, with the exception that output parameters are not passed through registers, but rather through the context block.

The context block is created by the RST 56 call. The user's register set (AF, BC, DE, HL, IX, IY) is pushed onto the stack. Register IY is set to point at this stack frame. Thus a copy of the input arguments exists in RAM which the system routine may refer to as needed. These arguments are also present in the registers when the system routine is entered, hence it is only necessary to refer to the context block when one has clobbered an input argument.

An output argument is returned to the caller by setting it in the context block. If a register was changed, but the associated cell in the context block was not, then the register will have it's old value on return. Thus a system routine is free to use any of the registers it needs without concern to saving and restoring. Moreover, the user can assume that no registers will change except those defined as returning an output argument.

The following illustration describes the context block and equates provided in HVGLIB for each field.

Four tables are used by the UPI in the control transfer process. The first two tables gives the routines starting address indexed via call number. The systems table is named SYSDPT. The user may extend this table by storing the address of his extended table into USERTB, USERTB+1. This address should point 128 bytes before the first entry.

The other two tables describe what in line arguments a call that specifies in line arguments should expect. This table gives a one-byte bitstring, also indexed via call number. The systems name is MRARGT, the user's address is in UMARGT, UMARGT must point 64 bytes ahead. Arguments must follow the call in a specified order.

Note that the context contains additional information not shown. This information exists both above and below the context. User programs should never use this information or even assume that it exists. The user should only address this area by using IY.

DISPLACEMENT	MEMORY CELL	EQUATE NAME
0	IY	CBIYL
1		CBIYH
2	IX	CBIXL
3		CBIXH
4	E	CBE
5	D	CBD
6	C	CBC
7	B	CBB
8	FLAGS	CBFLAG
9	A	CBA
A	L	CBL
B	H	CBH

CONTEXT BLOCK FORMAT

IN LINE ARGUMENT MASK TABLE ENTRY

TABLES MRARGT and UMARGT

If a bit corresponding to a register is set, the register is loaded. The order in which the arguments must appear is:

IX (L then H), E, D, C, B, A, L, H

If an argument isn't specified, it is omitted.

7	6	5	4	3	2	1	[illegible]
H	L	A	IX	B	C	D	E

UPI INTPC

BEGIN INTERPRETING

Calling Sequence: SYSTEM INTPC

Aruguments: None

Notes: None

Description:

See UPI description for explanation of interprete

UPI XINTC

EXIT INTERPRETER

Calling Sequence: EXIT

Arguments: None

Description:

This code causes the interpreter to exit. Execution of machine instructions proceeds at the following location.

Restrictions:

This routine should only be called using the interpreter. A direct system call would produce unpredictable (and catastrophic) results.

UPI RCALL

CALL ASSEMBLY LANGUAGE SUBROUTINE

```
Calling Sequence:       DO      RCALL
                              or
                        DONT    RCALL
                        DEFW    (routine address)
Arguments:              HL=address of routine to call
```

Description:

RCALL may be used to call any assembly language subroutine from the interpreter. When the subroutine returns, interpretation proceeds at the next instruction.

When the assembly language routine receives control, HL will point at the routine's starting address; the other registers will contain their current values. Any changes made to the register set by the subroutine will not be passed along. To pass an output parameter, the subroutine must alter the context block, which is pointed at by IY.

Restrictions:

Assembler routine must not destroy IY.

```
Example:                DEFB    RCALL
                        DEFW    CLRAC
                              .
                              .
                              .
                CLRAC:  XOR     A
                        RET
```

UPI MCALL

CALL INTERPRETER SUBROUTINE

```
Calling Sequence:       SYSTEM  MCALL
                             or
                        SYSSUK  MCALL
                        DEFW    (routine address)

Arguments:              HL=Subroutine Address
```

Description:

MCALL is used to call an interpreter sequence as a subroutine. MCALL may be used from machine language as well as within an interpreted sequence. Calls may be nested infinitely, limited only by stack space (4 bytes per call)

To exit the interpreted subroutine, use MRET.

```
Example:                SYSSUK  MCALL
                        DEFW    ZAPALL
                             .
                             .
                             .
                ZAPALL: DO      FILL+1          ;DO FILL
                        DEFW    NORMEM
                        DEFW    ØFFFFH
                        DEFB    Ø
                        DO      MRET            ;GO BACK TO CALLER
```

UPI MJUMP

INTERPRETER JUMP

```
Calling Sequence:       DO      MJUMP
                             or
                        DONT    MJUMP
                        DEFW    (goto address)
```

Arguments: HL=Go to address

Description:
The current interpretive program counter is set to the contents of HL. The next instruction is fetched from that address.

Restrictions:
MJUMP must be called from the interpreter. The targets of all JUMPS must also be interpreted sequences.

Example:

```
                SYSTEM  INTPC           ;ENTER INTPC STEP
                     .
                     .
                     .
                DO      MJUMP           ;JUMP TO END OF
                DEFW    END             ;INTPC STEP
                     .
                     .
                     .
        END:    DEFB    XINTC           ;EXIT INTERPRETER
```

UPI MRET

RETURN FROM INTERPRETIVE SUBROUTINES

Calling Sequence: DO MRET

Arguments: None

Description:

MRET causes execution to proceed at the instruction following the corresponding MCALL instruction. See MCALL for more information.

SCREEN HANDLER

The screen handler is a group of routines for generating frame buffer images. Included are entries for filling sections of the screen with constant data, the animation of figures, and the display of alphanumerics.

Many of these routines utilize the MAGIC functions provided by the custom chips. Since the status of these chips cannot be context-switched, many of these routines are not re-entrant. The user is responsible for preventing conflicts. This can be done by disabling interrupt, or implementing a semaphore.

SCREEN SETOUT

SET DISPLAY PORTS

```
Calling Sequence:       SYSTEM  SETOUT
                             or
                        SYSSUK  SETOUT
                        DEFB    BLINE*2
                        DEFB    HORIZX/4
                        DEFB    INMOD
Arguments:              A=Data to output to INMOD  (port EH)
                        B=Data to output to HORCB  (port 9H)
                        D=Data to output to VERBL  (port AH)
Output:                 None

Description:            Outputs above data to ports
                        See hardware writeup for discussion of
                        above ports.
```

SCREEN FILL

FILL A CONTIGUOUS AREA WITH CONSTANT

```
Calling Sequence:      SYSTEM  FILL
                            or
                       SYSSUK  FILL
                       DEFW    (first byte)
                       DEFW    (number of bytes)
                       DEFB    (data to fill with)
Arguments:             A =Data to fill with
                       BC=number of bytes to fill
                       DE=address to begin filling at
```

Description:

This routine sets the memory range DE to (DE+BC-1) to the specified constant.

Notes:

Fill can be used for screen clearing, or initialization of scratchpad RAM. It is re-entrant.

SCREEN RECTAN

PAINT A RECTANGLE

Calling Sequence:

```
SYSTEM  RECTAN
     or
SYSSUK  RECTAN
DEFB    (X co-ordinate)
DEFB    (Y co-ordinate)
DEFB    (X size)
DEFB    (Y size)
DEFB    (color mask)
```

Arguments:

A =Color mask to write rectangle with
B =Y-size of rectangle in pixels
C =X-size of rectangle in pixels
D =Y co-ordinate for UL corner of rectangle
E =X co-ordinate for UL corner of rectangle

Description:

A rectangle of specified size of color mask is written at X,Y. RECTAN uses the MAGIC functions and is not re-entrant.

Example: Put up a 3 X 4 rectangle of color 2 at 15,13.

```
DO      RECTAN
DEFB    15
DEFB    13
DEFB    3
DEFB    4
DEFB    1Ø1Ø1Ø1ØB
```

SCREEN WRITE ROUTINES

Virtually every video game involves the manipulation of animated figures. These figures are composed of patterns which are arbitrary pixel arrays. The write routines are used to transfer such patterns to the screen.

Five hierarchical levels of call are supported. The levels differ in the amount of preprocessing required by the user before calling. The highest level assumes that most of the parameters reside in a standard data structure, while the lowest level presumes that all arguments are in registers with all attendant transformations (such as relative-to-absolute conversion) already accomplished. The five levels are:

(1) Write from a Vector
(2) Write Relative
(3) Write Variable Pattern
(4) Write
(5) Write Absolute

Two transformations of the pattern may be performed prior to writing. They are FLOP and EXPAND. FLOP is mirroring the pattern on the X-axis. EXPAND is the translation of a 1-bit per pixel pattern into a 2-bit per pixel pattern. Since many patterns are only two-color, this allows for more efficient pattern storage. FLOP and EXPAND can both be done at the same time.

Three writing modes may be used. They are PLOP, OR, and XOR. PLOP is a conventional store into RAM. If OR is optioned, the data being written is ORed bit by bit with whatever was already there. Similarly, if XOR is set, the pattern is XORed with that beneath. Use of OR or XOR takes slightly longer since a read before write must be performed.

Note that ROTATE is not currently supported in software due to space considerations.

STANDARD CALLING SEQUENCE

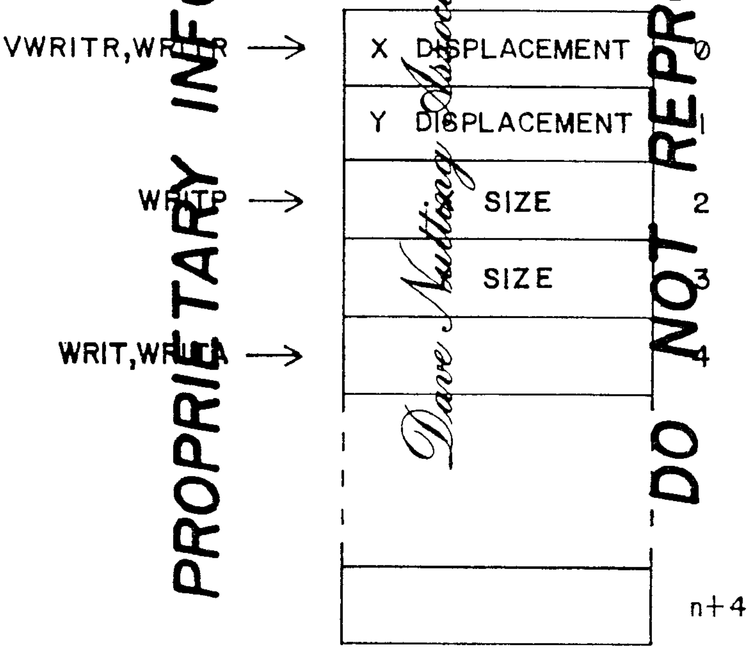
Every write routine uses a subset of the following argument/register assignment:

A	=	Magic Register
B	=	Y Pattern Size
C	=	X Pattern Size in Bytes
D	=	Y Co-ordinate (Ø - 1Ø1)
E	=	X Co-ordinate (Ø - 159)
HL	=	Pattern Address
IX	=	Vector Address

PATTERN REPRESENTATION

The higher the level of the write routine, the more ancillary information is stored with the pattern. The following diagram shows what each level expects. Any bytes of lower address than the pointer for a given level, need not be specified.

Use Restrictions:
None of the write routines are re-entrant due to Magic Register/Expander clobber.



SCREEN WRITE VWRITR
WRITE RELATIVE FROM VECTOR

Calling Sequence: SYSTEM VWRITR
or
SYSSUK VWRITR
DEFW (vector)
DEFW (pattern)

Arguments: HL=Pattern address
IX=Vector Address

Output: DE=Absolute address used
A =Magic register used

Description:
The co-ordinates and magic register are loaded from the specified vector. (See vector routine document) The relative co-ordinates stored with the pattern are added to the co-ordinates from the vector. The pattern size is also taken from the pattern and writing proceeds.

Notes:
If expansion is to be done, the ON/OFF color must be set by the user before calling VWRITR.

SCREEN WRITE WRITR

WRITE RELATIVE

Calling Sequence:	SYSTEM WRITR
	or
	SYSSUK WRITR
	DEFB (X co-ordinate)
	DEFB (Y co-ordinate)
	DEFB (Magic Register)
	DEFW (Pattern address)
Arguments:	HL=Pattern address
	A =Magic Register
	D =Y co-ordinate
	E =X co-ordinate
Output:	DE=Screen Address Used
	A = Magic Register Used

Description:

The relative co-ordinates stored with the pattern are added to the co-ordinates passed in DE. Pattern size is taken from the pattern.

Notes:

If expansion is to be done, the ON/OFF color must be set by the user before calling WRITR.

SCREEN WRITE   WRITP

WRITE WITH PATTERN SIZE SCARE UP

```
Calling Sequence:       SYSTEM  WRITP
                             or
                        SYSSUK  WRITP
                        DEFB    (X co-ordinate)
                        DEFB    (Y co-ordinate)
                        DEFB    (Magic Register)
                        DEFW    (Pattern address)
Arguments:              HL=Pattern Address
                        A =Magic Register
                        D =Y co-ordinate
                        E =X co-ordinate
Output:                 DE=Screen Address Used
                        A =Magic Register Used
```

Description:

The pattern size is taken from the pattern.

Notes:

User must worry about ON/OFF color if expansion is used.

SCREEN WRITE   WRIT

WRITE PATTERN

```
Calling Sequence:       SYSTEM  WRIT
                             or
                        SYSSUK  WRIT
                        DEFB    (X co-ordinate)
                        DEFB    (Y co-ordinate)
                        DEFB    (X pattern size)
                        DEFB    (Y pattern size)
                        DEFB    (Magic Register)
                        DEFW    (Pattern address)
Arguments:              HL=Pattern Address
                        A =Magic Register to use
                        B =Y pattern size
                        C =X pattern size
                        D =Y co-ordinate
                        E =X co-ordinate
Output:                 DE=Absolute address used
                        A =Magic Register used
```

Notes:

User must set ON/OFF color if using expansion.

SCREEN WRITE   WRITA

WRITE ABSOLUTE

Calling Sequence:

```
SYSTEM  WRITA
     or
SYSSUK  WRITA
DEFW    (Absolute address)
DEFB    (X pattern size)
DEFB    (Y pattern size)
DEFB    (Magic Register)
DEFW    (Pattern address)
```

Arguments:

```
HL=Pattern Address
A =Magic Register
B =Y Pattern size
C =X Pattern size
DE=Absolute screen address of upper left-
   hand corner of where to write
```

Notes:

This entry can be used for pattern writing to non-magic memory. The value in A is not output to (MAGIC); it is only interrogated to decide whether to FLOP or EXPAND.

SCREEN SAVE

SAVE AREA

Calling Sequence:

```
SYSTEM  SAVE
     or
SYSSUK  SAVE
DEFW    (save area)
DEFB    (X size)
DEFB    (Y size)
DEFW    (Screen address)
```

Arguments:

B =Y size of area to save

C =X size of area to save (in bytes)

DE=Address of save area

HL=Absolute address of upper left-hand corner of area to save

Description:

SAVE is used to preserve what is 'underneath' a moving pattern. SAVE copies the indicated area of the screen to the save area. The sizes of the area which was saved is preserved in the first two bytes of the save area.

The save area size must be greater than or equal to the X-size times the Y-size plus 2.

The save area may be MAGIC or non-MAGIC.

SCREEN RESTORE

RESTORE AREA

```
Calling Sequence:       SYSTEM  RESTOR
                             or
                        SYSSUK  RESTOR
                        DEFW    (Save area)
                        DEFW    (Screen address)
Arguments:              DE=Save area to restore from
                        HL=Absolute address of upper left-hand corner
                           of area to restore
```

Description:

RESTORE is the inverse of SAVE. The size of the area to restore is taken from the first two bytes of the save area.

SCREEN VBLANK

BLANK FROM VECTOR

```
Calling Sequence:       SYSTEM  VBLANK
                             or
                        SYSSUK  VBLANK
                        DEFW    (Vector address)
                        DEFB    (X size)
                        DEFB    (Y size)
Arguments:              D =Y size
                        E =X size (in bytes)
                        IX=Vector address
```

Description:

The BLANK bit in the vector status byte is tested. If it is not set, no blanking is done. If it is set, it is reset then the old screen address is taken from the vector and blanking is done. If FLOPPED is specified by the Magic Register byte in the vector, a flopped blank is done. VBLANK always blanks to zero.

SCREEN BLANK

BLANK AREA

```
Calling Sequence:       SYSTEM  BLANK
                             or
                        SYSSUK  BLANK
                        DEFB    (X size)
                        DEFB    (Y size)
                        DEFB    (Blank to)
                        DEFW    (Blank address)
Arguments:              HL=Blank address (not MAGIC)
                        B =Data to blank to
                        D =Y size
                        E =X size
```

Description:

The specified area is blanked to whatever is passed in B.

SCREEN SCROLL

SCROLL WINDOW

Calling Sequence: SYSTEM SCROLL

or

SYSSUK SCROLL

DEFW (line increment)

DEFB (# of bytes)

DEFB (# of lines)

DEFW (first byte)

Arguments: B =Number of lines to scroll

C =Number of bytes on line to scroll

DE=Line increment

HL=First byte to scroll

Description:

This routine copies NBYTES from first line +INC to first line. Thus to scroll upward, HL points at the first line (which is over-written) and the line increment would be positive. To scroll downward HL points at the last line and the line increment would be negative. The value in HL is an absolute address calculated by:

BASE OF SCREEN + #BYTES IN X OFFSET +(#lines offset*byte per line)

Note:

This routine can only be used to scroll one line at a time.

SCREEN ALPHANUMERIC
ALPHANUMERIC DISPLAY ROUTINES

HVGSYS provides several routines for the display of alphanumeric information. This section provides information which is common to all of the alphanumeric display routines.

The ASCII character code is used to represent all strings, with the following extensions:

> Characters with hex equivalents in the range 1 - 1F are interpreted as tabulation codes which cause the character display routines to skip over N character positions before writing the following characters.
>
> The characters 2ØH to 63H are displayed as 5 X 7 standard graphics with 3 pixels of horizontal spacing and 1 pixel of vertical spacing.
>
> The characters between 64H and 7FH are interpreted by STRDIS as control codes which cause the contents of registers C, DE, and IX to be changed to the value that follow the string. See table accompanying STRDIS.
>
> The characters between 8ØH and FFH are taken as references to a user supplied alternate character font.

The following argument/register combinations are used by all of the alphanumeric display routines.

Register C contains the options byte formatted as shown below.

ENLARGE FACTOR specifies if the character is to be enlarged in size. The table below defines the possible values for this parameter.

XOR/OR WRITE - All writes are performed through magic memory. Use of one of these options causes the character to be ORed/XORed with what was beneath it.

ON/OFF COLOR - all characters are stored one bit per pixel, but are written two bits per pixel by use of the expander. This field specifies the pixel values to translate the one bit per pixel representation into. For example, the value 11Ø1 specifies that the foreground color is 11, and the background color is Ø1.

OPTION BYTE

ENLARGE FACTOR	XOR WRITE	OR WRITE	ON COLOR	OFF COLOR

ENLARGE FACTOR	HOW MANY TIMES LARGER	ENLARGED SIZE OF SINGLE PIXEL
ØØ	1	1 X 1
Ø1	2	2 X 2
1Ø	4	4 X 4
11	8	8 X 8

D register contains the Y co-ordinate and the E register contains the X co-ordinate. These co-ordinates give the address of the upper left-hand corner where the first character will appear. Upon return, these registers are updated to give the address of the character to the right, (or below if no more space exists on the line). This simplifies the composition of complex messages.

IX register contains the Alternate Font Descriptor. It is required only if alternate font is referenced in call. Each character must be stored in one-bit per pixel format.

The small (3 X 5) character set is displayed using this facility. A word in the system DOPE vector points at a standard alternate font descriptor for this character set.

The format of the alternate font descriptor is shown below.

IX →	Ø	BASE CHARACTER	EQUAL TO FIRST CHARACTER IN TABLE
	1	X FRAME SIZE	CHARACTER SIZE IN BITS + X SPACING
	2	Y FRAME SIZE	CHARACTER SIZE IN BITS + Y SPACING
	3	X PATTERN SIZE	EACH CHARACTER TABLE ENTRY SHOULD BE OF SIZE X PATTERN*Y PATTERN SIZE
	4	Y PATTERN SIZE	
	5 6	CHARACTER TABLE ADDRESS	

SCREEN ALPHANUMERIC   DISNUM
DISPLAY BCD NUMBER

```
Calling Sequence:      SYSTEM  DISNUM
                            or
                       SYSSUK  DISNUM
                       DEFB    (X)
                       DEFB    (Y)
                       DEFB    (options)
                       DEFB    (extended options)
                       DEFW    (number address)
Arguments:             B =Extended options
                       C =Standard alphanumeric options byte
                       DE=Standard X,Y co-ordinate
                       HL=Address of BCD number
*NOT LOADED            IX=Optional character font descriptor
Outputs:               DE=Updated
```

Decription:

This routine displays the standard BCD codes Ø through 9. In addition, the codes AH through FH are also defined. The interpretation for these codes are:

```
                       A = *     B = +     C = '
                       D = -     E = .     F = /
```

If leading zero suppress is set, then instead of displaying a leading zero, a space is displayed. The first non-zero nibble encountered terminates leading zero suppression (including A - F). If the number is zero, a single zero is displayed.

If alternate font is set, the routine will display using codes between AAH and B9H (zero starting at BØH).

SCREEN ALPHANUMERIC   DISTIM
DISPLAY TIME

```
Calling Sequence:       SYSTEM  DISTIM
                             or
                        SYSSUK  DISTIM
                        DEFB    (X co-ordinate)
                        DEFB    (Y co-ordinate)
                        DEFB    (Options)
Arguments:              DE=X,Y co-ordinates
                        X =Options  (see note below)
                        IX=Alternate Font Descriptor  (not loaded)
Outputs:                DE=Updated
```

Description:
This routine displays the system time (GTMINS, GSECS) at the co-ordinates specified in the form MM:SS, where M=minutes, S=seconds. Seconds are optional.

Notes:
The small character set is used and one level of enlarge factor is permitted.
Options are the same as the alphanumeric display routine except that bit 7=1 to display colon and seconds; bit 7=Ø to suppress colon and seconds.

SCREEN ALPHANUMERIC   CHRDIS

DISPLAY CHARACTER

Calling Sequence:

```
SYSTEM  CHRDIS
     or
SYSSUK  CHRDIS
DEFB    (X co-ordinate)
DEFB    (Y co-ordinate)
DEFB    (options)
DEFB    (character)
```

Arguments:	A =ASCII character to display
	C =Standard options byte
	DE=Standard Y,X co-ordinates to begin at
*NOT LOADED	IX=Optional alternate font descriptor address
Outputs:	DE=Updated to next frame

Description:

This is the basic charcter display promative. If tabulation is specified, the co-ordinates are updated but no actual writing occurs.

Notes:

Observe that IX is not loaded by the UPI SUCK facility. If alternate font is used, IX must be loaded with alternate font descriptor address.

Since this routine uses magic memory, it is not re-entrant.

SCREEN ALPHANUMERIC STRDIS
DISPLAY STRING

Calling Sequences:	SYSTEM STRDIS
	or
	SYSSUK STRDIS
	DEFB (X co-ordinate)
	DEFB (Y co-ordinate)
	DEFB (Options)
	DEFW (String)
Arguments:	HL=String address
	C =Standard Options
	DE=Standard Co-ordinates
*NOT LOADED	IX=Alternate Font Descriptor Address
Outputs:	DE=Updated to next frame

Description:
The string pointed at by HL is displayed as optioned. The string is terminated by a zero byte.

Notes:
IX is not loaded by SUCK. STRDIS is not re-entrant.

STRDIS INTERPRETATION OF CODES 64H to 7FH

STRDIS responds to the charcter codes between 64H and 7FH. These codes are taken to specify that certain registers in the context block are to be set to new values. This facility is useful for changing size, write mode, screen co-ordinates, or fonts, during a single STRDIS call.

The following table specifies which registers are loaded for a given code. The order in which the new register data follows the code, is also represented.

64H	C	72H	IX,D
65H	E,C	73H	IX,E,D
66H	D,C	74H	IX,C
67H	E,D,C	75H	IX,E,C
68H	NONE	76H	IX,D,C
69H	E	77H	IX,E,D,C
6AH	D	78H	IX
6BH	E,D	79H	IX,E
6CH	C	7AH	IX,D
6DH	E,C	7BH	IX,E,D
6EH	D,C	7CH	IX,C
6FH	E,D,C	7DH	IX,E,C
7ØH	IX	7EH	IX,D,C
71H	IX,E	7FH	IX,E,D,C

SCREEN VECTORING - VECTORING ROUTINES

Most games involve moving patterns. Most moving patterns move along a line. The home video game operating system provides the vectoring routines to facilitate programming such pattern motion.

The vectoring routines work with a memory array called a vector. Represented within this vector are the co-ordinates of an object, the velocities of the object, and the necessary status information to control the object. By periodically invoking the vectoring routine, this data is updated and can be used to direct the motion of a pattern.

More formally, a vectored object possesses an X and Y co-ordinate. Associated with these co-ordinates are velocities ΔX and ΔY, which are added to X and Y every time increment. Since the screen is finite, there also exists two upper and two lower limits X_{LU}, X_{LL}, Y_{LU}, and Y_{LL}, the attainment of which requires some response.

The HVGSYS vectoring routine allows for two different responses to a limit attained. Either the sign of the delta is reversed or vectoring is stopped for this co-ordinate. This is specified by a flag byte. When attainment occurs this fact is indicated by a status byte. Also the co-ordinate is set equal to the limit that was attained, preventing over-shoot.

Utilization of the vectoring routines involves a number of user responsibilities. The user must properly initialize certain fields in the vector array. He must increment the time base byte, and periodically call the vectoring routine. Status bits must be checked and writing must be done.

To insure high-accuracy, co-ordinates and deltas are double-precision. The assumed binary "decimal point" is between the high and low order byte.

The following diagrams explain the layout of the vector array and the attendant user responsibilities.

VECTOR BLOCK

BYTE	FUNCTION	HVGLIB NAME	
0	MAGIC REGISTER	VBMR	– DO NOT USE BIT 7
1	VECTOR STATUS	VBSTAT	
2	TIME BASE	VBTIMB	– INCREMENTED BY USER
3	ΔX	VBDXL	
4		VBDXH	
5	X	VBXL	
6		VBXH	
7	X CHECKS MASK	VBXCHK	
8	ΔY	VBDYL	
9		VBDYH	
10	Y	VBYL	
11		VBYH	
12	Y CHECKS MASK	VBYCHK	
13	OLD SCREEN ADDRESS	VBOAL	– MAINTAINED BY USER
14		VBOAH	

VECTOR STATUS DETAIL

ACTIVE VBSACT	BLANK VBBLNK	NOT USED					

ACTIVE Set by user to indicate that vector is active. The vectoring routines will do no processing if reset.

BLANK Must be initialized by user to reset state. Thereafter this bit is maintained by the VWRIT and VBLANK system routines.

CHECKS MASK DETAIL

NOT USED				LIMIT ATTAINED VBCLAT	NOT USED	REVERSE DELTA SIGN VBCREV	LIMIT CHECK VBCLMT

LIMIT CHECK Set by user to indicate that this co-ordinate is to be limit checked.

REVERSE DELTA Set by user to indicate that when this co-ordinate attains it's limit, the sign of the associated delta is to be reversed. This can be used to cause objects to 'bounce' off barriers.

LIMIT ATTAINED Set by system if the limit was attained this call. Otherwise it is reset. If the delta was not changed, either by Reverse Delta or user, this bit will stay set.

SCREEN VECTORING VECT
VECTOR OBJECT IN TWO DIMENSIONS

Calling Sequence:	SYSTEM VECT or SYSSUK VECT DEFW (Vector address) DEFW (Limit table)

Arguments:	HL=Limit table address
	IX=Vector address (points at VBMR)

Output:	C =Time base used Z =True, if it did not move

Description:
If the vector is inactive, control is returned immediately. Otherwise VECTC is called for X, then Y. The zero status is determined by comparing the new co-ordinate value with it's old value. If the high-order byte changed, then the object moved. Zero status set if object did not move, reset if object moved.

SCREEN VECTORING VECTC

VECTOR A CO-ORDINATE

```
Calling Sequence:     SYSTEM  VECTC
                            or
                      SYSSUK  VECTC
                      DEFW    (co-ordinate address)
                      DEFW    (Limit table)
Arguments:            IX=Pointer to low-order byte of delta for co-ordinate
                      HL=Limits table for this co-ordinate (if required)
                      C =Time base to use
```

Description:

This routine operates on the subset of the vector array associated with a single co-ordinate. This subset consists of the delta co-ordinate and checks mask. This entry is provided so special vectoring schemes may be implemented such as 1 dimensional or 3 dimensional vectoring.

This entry adds the delta to the co-ordinate time base times. It then performs the limit checks for the co-ordinate if optioned.

Note that this entry does not interrogate or alter any bytes in the vector array outside of the defined subset. Hence the active bit isn't checked.

SCREEN RELABS

CONVERT RELATIVE CO-ORDINATES TO ABSOLUTE MAGIC ADDRESS AND SET UP MAGIC REGISTER

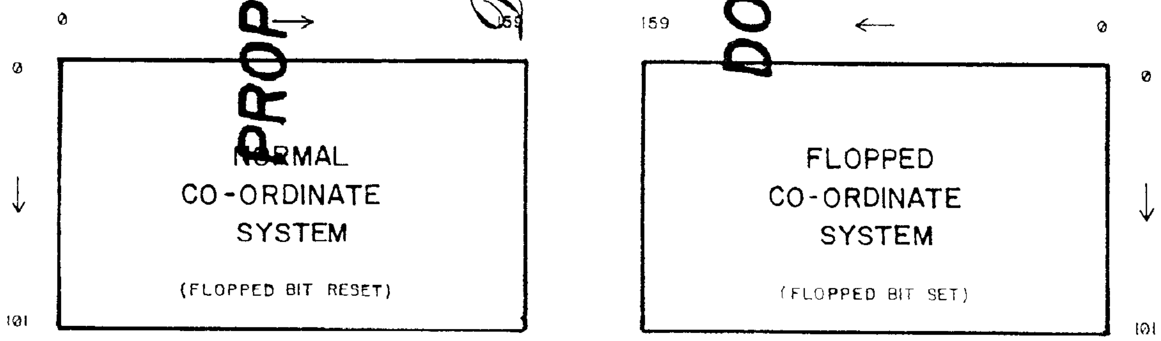
Calling Sequence: SYSTEM RELABS
or
SYSSUK RELABS
DEFB (Magic register value)

Arguments: A =Magic register value to set
D =Y co-ordinate
E =X co-ordinate

Output: A =Magic register value, with proper shift amount set
DE=Absolute memory address (MAGIC)

Description:

The low-order two bits of the X co-ordinate are inserted into the magic register value bit string. The absolute memory address corresponding to the co-ordinate is computed, taking into consideration the value of the flopped bit. The co-ordinate systems used are shown below.



SCREEN RELAB1

CONVERT RELATIVE ADDRESS TO ABSOLUTE NORMAL ADDRESS

Calling Sequence:	SYSTEM RELAB1
	or
	SYSSUK RELAB1
	DEFB (Magic register value)
Arguments:	A =Magic register value to combine with shift amount
	D =Y co-ordinate
	E =X co-ordinate
Output:	A =Combined magic register value
	DE=Absolute normal address (not magic)

Description:

This routine is identical to RELABS except that a non-magic address is returned and the hardware magic register is not set. The flopped bit is interrogated and the flopped co-ordinate system is used, if optioned.

SCREEN COLSET

SET COLOR REGISTERS

```
Calling Sequence:       SYSTEM  COLSET
                             or
                        SYSSUK  COLSET
                        DEFW    (Address of color list)
Inputs:                 HL=Color list laid out
                     COL3L=first to
                     COLOR last ie: COLOR would be at a higher
                                    address than COL3L
```

Description:

This routine sets color registers and saves address of colors for use by PIZBRK and BLAKOUT for color restoration.

HUMAN   INCSCR

INCREMENT SCORE AND COMPARE TO END SCORE

Calling Sequence: SYSTEM INCSCR
                    or
                    SYSSUK INCSCR
                    DEFW   (address of score)

Arguments: HL=Address of score (must be 3 bytes long)

Output: Score incremented and optionally game over bit set

Description:

The 3 byte score pointed at by HL (BCD with low order byte at lowest address) is incremented (by 1) and compared to the end score (ENDSCR). If the end score bit (GSBSCR) was set in the game status byte (GAMSTB) and end score has been reached, then the game over bit (GSBEND) is set in the game status byte.

HUMAN PAWS

PAUSE

```
Calling Sequence:       SYSTEM PAWS
                            or
                        SYSSUK PAWS
                        DEFB   (number of interrupts)
Arguments:              B=Number of interrupts to wait
```

Description:

This routine provides for a pause for certain number of interrupts. If used with ACT INT, 60 will be a 1-second pause. This routine does an EI upon entry and assumes interrupts will occur.

HUMAN KEYBOARD KCTASC
KEY CODE TO ASCII

Calling Sequence: SYSTEM KCTASC
Arguments: B=Key code (not loaded)
Output: A=ASCII equivalent of keycode
Description: This routine does a table look-up

KEYCODE	NAME	GRAPHIC	HEX VALUE
1	Clear	C	43
2	Up Arrow	↑	5E
3	Down Arrow	↓	5C
4	Percent	%	25
5	Recall	MR	52
6	Store	MS	53
7	Change sign	CH	3B
8	Divide	÷	2F
9	7	7	37
10	8	8	38
11	9	9	39
12	Times	X	2A
13	4	4	34
14	5	5	35
15	6	6	36
16	Minus	-	2D
17	1	1	31
18	2	2	32
19	3	3	33
20	Plus	+	2B
21	Clear Entry	CE	26
22	Ø	Ø	3Ø
23	Decimal point	.	2E
24	Equals	=	3D

HUMAN CONTROLS & KEYPAD   SENTRY
SENSE TRANSITION

```
Calling Sequence:       SYSTEM  SENTRY
                            or
                        SYSSUK  SENTRY
                        DEFW    (Key mask address)
Arguments:              DE=Keypad mask table
```

Description:
SENTRY checks for changes in the potentiometers (pots), control handles, triggers, keypad, semiphores and counter/timers. It also takes care of blackout. Blackout is the automatic blacking-out of the screen after 255 seconds without a change. If SENTRY isn't called then the game will not black out.

SENTRY checks if TIMOUT equals Ø on entry and if zero, it goes to PIZBRK. If a key has gone down or a control handle changed, then TIMOUT is set to FFH.

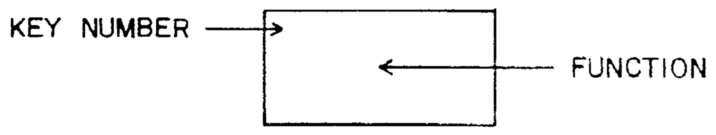
HL should point at a keypad mask. The keypad consists of 6 rows by 4 columns.

```
Example mask of         DEFB    Ø111ØØB
just Ø - 9              DEFB    1111ØØB
                        DEFB    Ø111ØØB
                        DEFB    ØØØØØØB
```

See diagram on following page.

MASK BIT NUMBER	1	2	3	4
Ø	1 C	2 ↑	3 ↓	4 %
1	5 MR	6 MS	7 CH	8 ÷
2	9 7	10 8	11 9	12 ×
3	13 4	14 5	15 6	16 −
4	17 1	18 2	19 3	20 +
5	21 CE	22 Ø	23 •	24 =

MASK BYTE NUMBER



Output: A=Return code
B=Extended code

PRIORITY	A=	MEANING	
	SNUL	Nothing changed	
1	SCTØ	Counter/timer Ø decremented to Ø	
	to		
1	SCT7	Counter/timer 7 decremented to Ø	
2	SFØ	SEMI4S bit Ø was 1	
	to		
2	SF7	SEMI4S bit 7 was 1	
4	SSEC	1 second has elapsed since the last SSEC	
5	SKYU	Keypad went from down to up	B=Ø
5	SKYD	Key is down	B=key number
3	SPØ	Pot Ø changed	B=new value
	to		
3	SP3	Pot 3 changed	B=new value
6	SJØ	Joystick Ø changed	B=new value
	to		
6	SJ3	Joystick 3 changed	B=new value
6	STØ	Trigger Ø changed	B=new value
	to		
6	ST3	Trigger 3 changed	B=new value

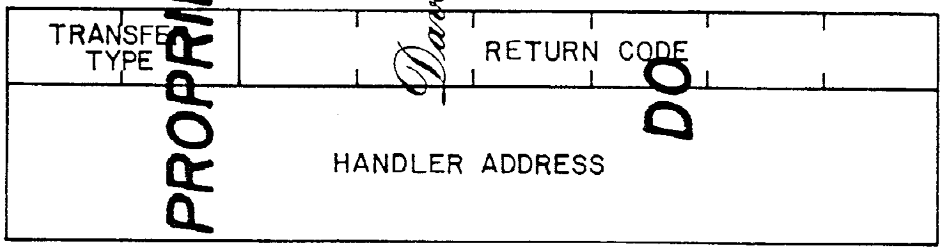
Notes:
The potentiometers (pots) are debounced. New trigger value=Trigger off (Ø) or trigger on (1ØH). When switches are actuated simultaneously the order of return is: SCT7 to SCTØ, SF7 to SFØ, SPØ to SP3, SSEC, SKYU, SKYD, SJØ, STØ, SJ1, ST1, SJ2, ST2, SJ3, ST3.

HUMAN CONTROL DOIT
RESPOND TO INPUT TRANSITION

Calling Sequence: SYSTEM DOIT
                         or
                     SYSSUK DOIT
                     DEFW   (Do table)

Arguments: A =SENTRY return code
                     B =Extended return code
                     HL=Do table address

Description:
The SENTRY return code is used to search the DOTABLE. If the transition is present in DOTABLE, then control is transferred to the associated handling routine. The handling routine may be MACRO or machine instructions. The routine receives registers as they are on DOIT entry. If no transition is found, execution continues at the first instruction following call. The DOTABLE is a linear list composed of 3 byte entries, 1 entry per SENTRY return code.



Where transfer type designates how handler address is to be transferred to. The codes are: ØØ=JMP to machine language routine and pop context; Ø1=RCALL machine language routine in current context; 1Ø=MCALL interpreter routine in current context. Mode Ø1 and 1Ø expect the returned-to point to be interpretive, mode Ø expects it to be machine instructions.
End of list is indicated by a terminator byte which is greater than or equal to CØH.

HUMAN CONTROL PIZBRK

"COFFEE BREAK" BLACK OUT SCREEN AND WAIT FOR KEY

Calling Sequence: SYSTEM PIZBRK
or
SYSSUK PIZBRK

Input: NONE

Output: NONE

Description:

This routine blacks out the screen and waits for either a key press or a trigger or a joystick change.

This function should be called whenever a "hold until further notice" is needed.

All keys on the keypad are enabled. Interrupts are disabled on entry and enabled on exit. It is a good idea to reset any 60th of a second timers on exiting PIZBRK.

HUMAN CONTROLS   EXAMPLE

This routine echoes number keys and takes a coffee break on trigger Ø being pulled. Assumes SP is set and screen erases.

```
        SYSTEM  INTPC
LOOP:   DO      SENTRY
        DEFW    NUMBAS
        DO      DOIT
        DEFW    DTAB
        DO      MJUMP
        DEFW    LOOP

NUMBAS: DEFB    Ø111ØØB          ;NUMBER KEYS ONLY
        DEFB    1Ø11ØØB
        DEFB    Ø111ØØB
        DEFB    Ø

DTAB:   MC      SKYD,SHOW        ;ON KEY DOWN MACRO CALL
        MC      STØ,PBREAK+END   ;ON TO MACRO CALL

SHOW:   DO      KCTASC           ;CONVERT TO ASCII
        DO      SUCK
        DEFB    ØØØØØ111B         ;X,Y=Ø=DE
        DEFB    11ØØ11ØØB         ;OPTIONS=C
        DONT    CHRDIS           ;DISPLAY CHAR
        MRET                     ;BACK TO LOOP

PBREAK: DO      PIZBRK           ;COFFEE BREAK
        DO      MRET             ;BACK TO LOOP
```

INTERRUPT MUSIC PROCESSOR

The music processor can be thought of as an independent CPU handling all output to the music/noise ports. The MUZCPU has 4 registers:

MPC: Like all program counters, points to the next data byte to fetch.

MSP: Like a stack pointer, points to return addresses on the stack.

DURATION: Is loaded at the start of a note and then decremented every 6Øth of a second

VOICES: Is a status register. It tells which voices (tones) to load with what data.

The voices status register is shown below. Execution proceeds right-to-left. Make sure that you always have at least one PC incrementing bit or load on.

INC PC	OUT TONE A	INC PC	OUT TONE B	INC PC	OUT TONE C	OUT VIBRA	OUT VOLN

MUZCPU INSTRUCTION SET

# OF BYTES	MNEMONIC	COMMENT
2	VOICES,(data)	;VOICES=(data)
2	MASTER,(data)	;TONEØ=(data)
3	CALL,(address)	;(SP)=(PC+3) PC=address
1	RET	;PC=(SP++)
3	JP,(address)	;PC=address
2	NOTE1	;Duration, note or data (D1)
3	NOTE2	;Duration, D1,D2
4	NOTE3	;Duration, D1,D2,D3
5	NOTE4	;Duration, D1,D2,D3,D4
6	NOTE5	;Duration, D1,D2,D3,D4,D5
2	REST	;Duration in 6Øths of a second
		;Pauses silently (except legato)
1	QUIET	;Stops music and sets volume=Ø
2	OUTPUT	;Port #, Data
9	OUTPUT	;SNDBX,DATA1Ø,D11,D12,D13,D14,D15,D16,D17
3	VOLUME	;(VOLAB),(VOLMC) sets volume for notes
1	PUSHN	;Push # between 1-16 onto the stack
1	CREL	;Call relative to next instruction
3	DSJNZ	;decrement stack top and jump
		;if not Ø, else pop stack
1	LEGSTA	;flips between STACATO and LEGATO modes
		;STACATO is clipped 1/60th before the
		;end of each note
		;LEGATO allows one note to run into
		;the next

Note: All durations are limited to a maximum of 127

MUSIC SCORE EXAMPLE

```
VOICES   11Ø1Ø1ØØB       ;ABC=Data 1
MASTER   ØA1H            ;ABC=½
VOLUME   88H,Ø8H
NOTE1    12,A1
NOTE1    12,C2
NOTE1    24,E2
NOTE1    12,C2
NOTE1    12,E2
REST     6
VOICES   1111Ø11ØB       ;Suck in Vibrato, AB and C bytes
NOTE3    12,14,A2,E[illegible]
QUIET
```

INTERRUPTS   MUSIC   BMUSIC

BEGIN PLAYING MUSIC

```
Calling Sequence:       SYSTEM  BMUSIC
                             or
                        SYSSUK  BMUSIC
                        DEFW    (Music stack)
                        DEFB    (voices byte)
                        DEFW    (Score)
Arguments:              A =Voices to start with
                        HL=Music PC (Score)
                        IX=Music SP
```

Description:

Quiets any previous music, then interprets "score". See music processor for more information.

INTERRUPTS MUSIC EMUSIC
STOP MUSIC

Calling Sequence: SYSTEM EMUSIC
or
SYSSUK EMUSIC

Arguments: NONE

Outputs: NONE

Description:

Outputs Ø to volume ports and halts music processor.

INTERRUPTS ACTINT

ACTIVE INTERRUPTS

Calling Sequence:	SYSTEM ACTINT or SYSSUK ACTINT
Input:	NONE
Output:	NONE
Function:	Sets IM=2, INLIN=2ØØ, sets I reg + INFBK Calls TIMEX and TIMEY Enables interrupts

Description:

Once ACTINT is called, it provides interrupt service completely automatically. It runs the seconds timer, the game timer, the music processor, and black-out timers, plus CTØ, CT1, CT2, CT3. Functions as 6Øth of a second timers.

INTERRUPTS   TIMERS   DECCTS

DECREMENT COUNTER/TIMERS

Calling Sequence: SYSTEM DECCTS
                    or
                    SYSSUK DECCTS
                    DEFB   (Mask)

Input: C=Mask indicative which counters to decrement.

Output: Sentry will notify the program.

Description:

Decrements counters if they are not zero. If any go from 1 to Ø, sentry is notified.

INTERRUPTS TIMERS CTIMER

Calling Dequence: CALL CTIMER

Input: HL=Address of custom time base

B =Value to load into time base 1 to Ø transition

C =CT mask as in DECCTS

Description:

HL is loaded and decremented. If it is not = Ø, then a return is executed. Else, HL is loaded with B and DECCTS is called.

Registers HL, DE, BC, and AF are undefined upon exit.

INTERRUPTS TIMERS STIMER

DECREMENT TIMERS

Calling Sequence:

```
PUSH    AF
PUSH    BC
PUSH    DE
PUSH    HL
CALL    STIMER
POP     HL
POP     DE
POP     BC
POP     AF
```

Input: NONE

Description: STIMER keeps track of game time. If it hits Ø, then the GSBEND bit in the game status byte is set.

Uses: AF, BC, DE, HL

Calls: Music processor on note (duration) expiration.

Note: Sets bit 7 of key sex to 1 on every second.

MOVE   MOVE BYTES

Calling Sequence:

```
SYSTEM  MOVE
     or
SYSSUK  MOVE
DEFW    (Destination)
DEFW    (Number of bytes)
DEFW    (Source)
```

Arguments:

DE=Destination address
HL=Source address
BC=Number of bytes to transfer

Description:

MOVE uses LDIR to copy bytes from source to destination.

INDEXN   INDEX NIBBLE

Calling Dequence: SYSTEM INDEXN
                              or
                         SYSSUK INDEXN
                         DEFW   (Base Address)

Arguemnts: C =Nibble displacement (Ø - 255)
                         HL=Base address of table

Output: A =Nibble value

Description:

INDEXN is used to look up a given nibble in a linear list.
The indexing works like:

BASE ADDRESS	1	0
1	3	2
2	5	4
3	7	6
.		
.		

STOREN   STORE NIBBLE

Calling Dequence: SYSTEM STOREN
or
SYSSUK STOREN
DEFW   (Base address)

Arguments: C =Nibble displacement   *NOT LOADED
HL=Base address
A =Nibble value to store   *NOT LOADED

Description: STOREN is the inverse of INDEXN.
STOREN works as with INDEXN.

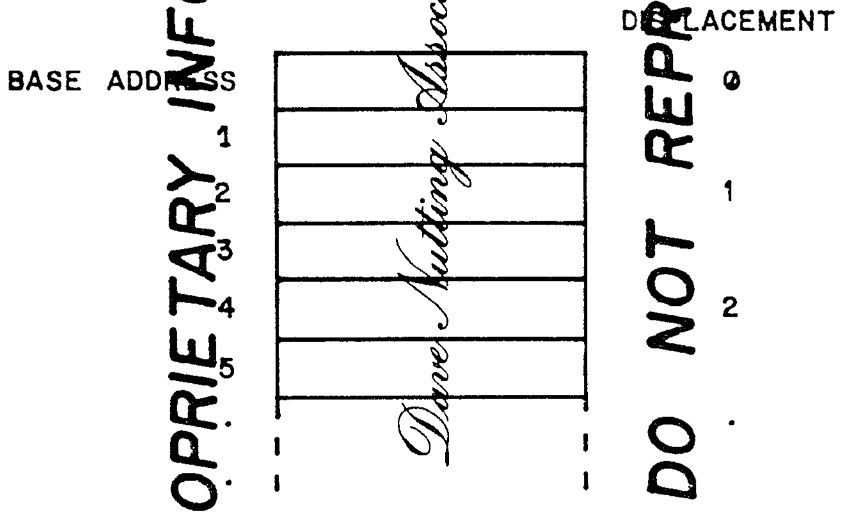
INDEXW   INDEX WORD

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

Arguments: A =Displacement (Ø - 255)   *NOT LOADED
HL=Base address of table

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

Description: Indexing looks like:



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 ACTINT interrupt routine is enabled
3) The MENU colors are set to 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	NEXT MENU NODE	MENU NODE FOR FIRST GAME ON CASSETTE
2		
3	STRING ADDRESS FOR FIRST GAME	
4		
5	START ADDRESS FOR FIRST GAME	
6		
7	RST 8 JUMP VECTOR	THESE CELLS MAY BE USED FOR PROGRAM IF THE ASSOCIATED RST OR HOOK IS NOT USED
8		
9		
A	RST 16	
B		
C		
D	RST 24	
E		
F		
2010	RST 32	
1		
2		
3	RST 40	
4		
5		
6	RST 48	
7		
8		
9	SENTRY HOOK TRANSFER VECTOR USED FOR DEMO PROGRAMS	
A		
B		

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 mode 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, creams 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.

DISPLAY NUMBER OPTIONS

ZERO SUPP	ALT FONT	NUMBER OF DIGITS TO DISPLAY/ACCEPT

CHARACTER DISPLAY OPTIONS

ENLARGE FACTOR	XOR	OR	ON COLOR	OFF COLOR

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                    *NOT LOADED
                        C =Flop flag
                        DE=X positive delta
                        HL=Y positive delta
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 Z-80 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.

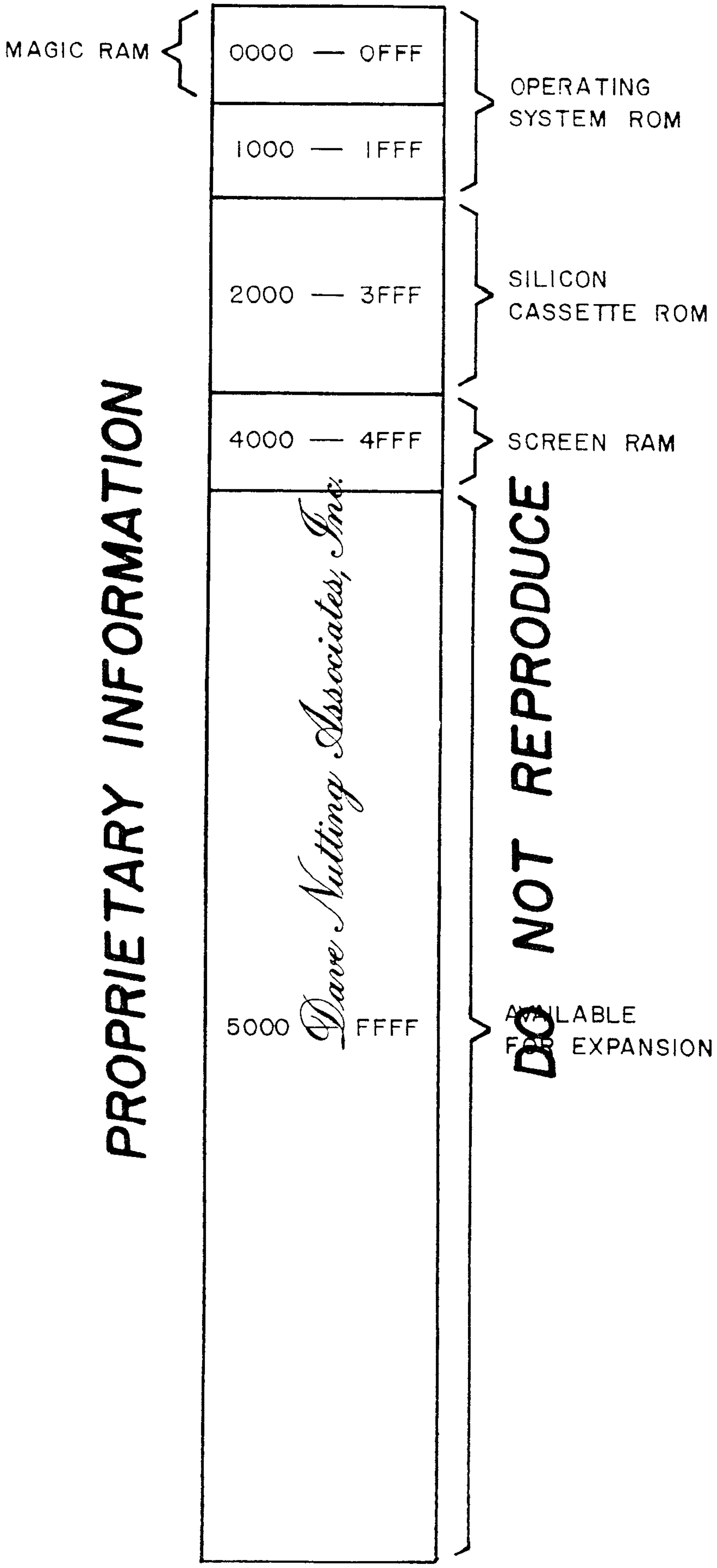
The system exists in both high-resolution and low-resolution models. The three custom chips can operate in either mode. The mode of operation is determined by bit Ø of output port 8H. It must be set to Ø for low-resolution and 1 for high-resolution. This bit is not set to Ø at power up and must be set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

MEMORY MAP

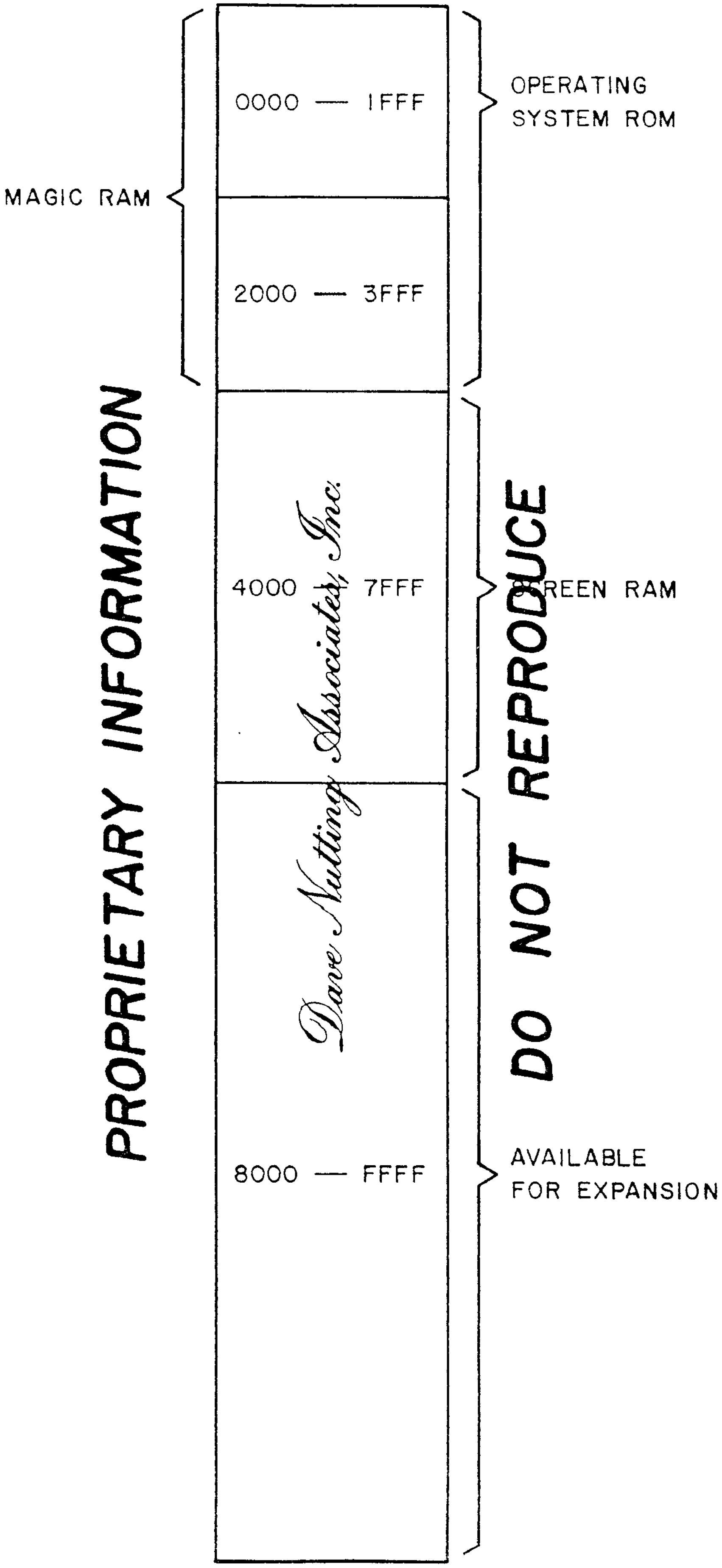
In both the low and high resolution models, the operating system ROM is in the first 8K of memory space. The silicon cassette ROM is in the space from 8K to 16K. The standard screen RAM begins at 16K. In the low-resolution unit, standard screen RAM is 4K bytes; in the high-resolution unit it is 16K bytes. 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



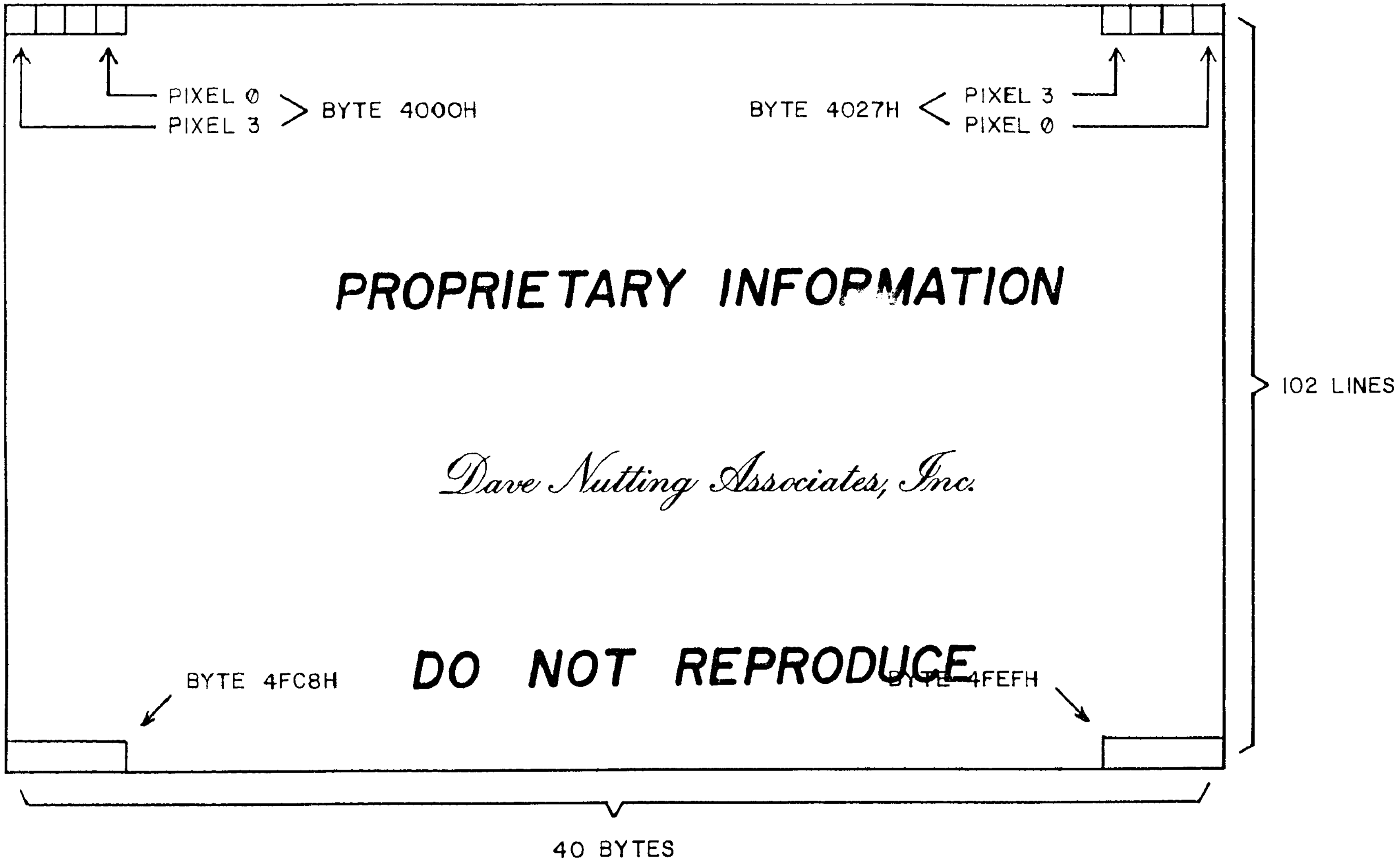
SCREEN MAP

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.

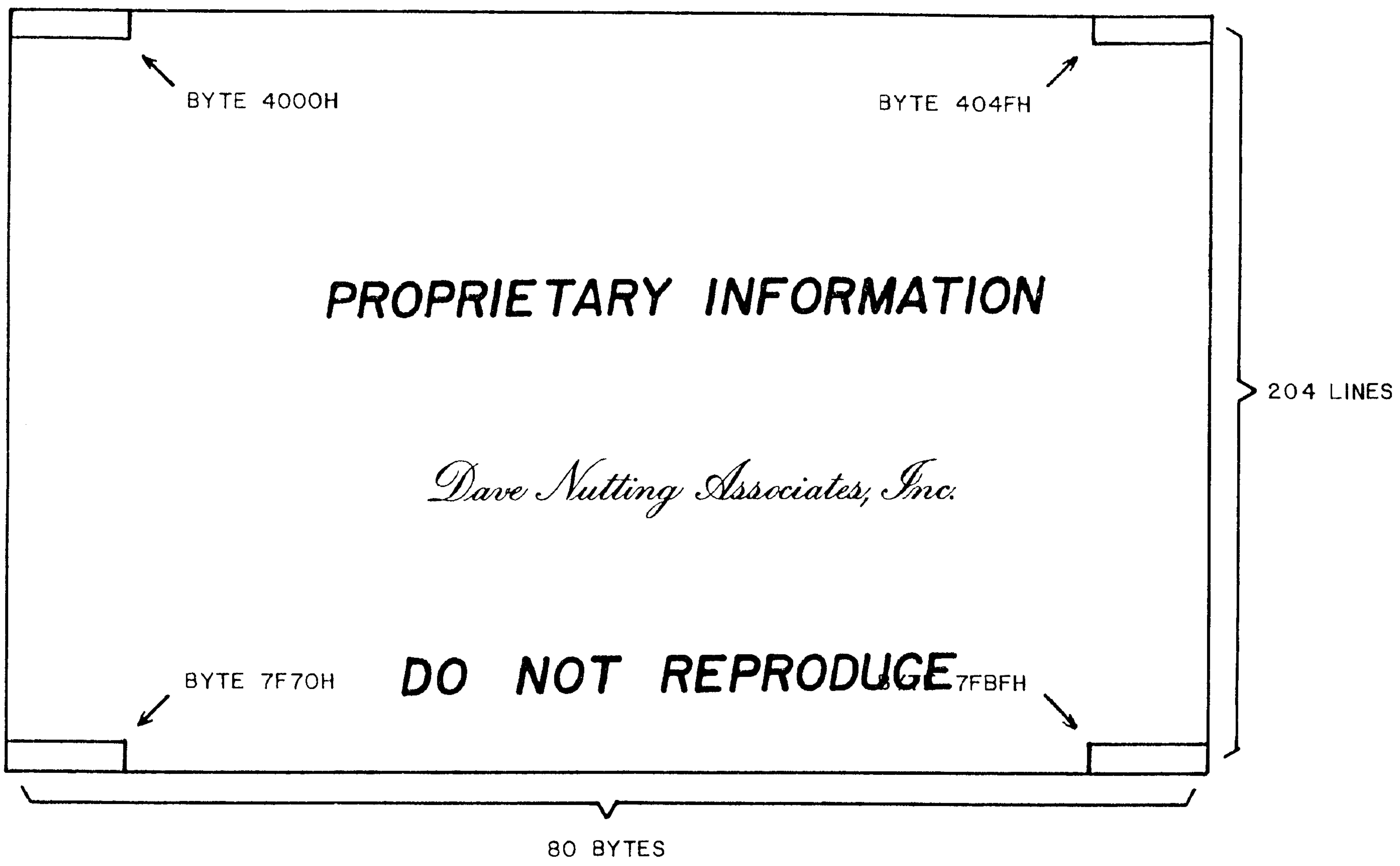


PROPRIETARY INFORMATION

Dave Nutting Associates, Inc.

DO NOT REPRODUCE

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 ar all eight can be accessed by one OTIR instruction. The OTIR instruction is to port BH (register C=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

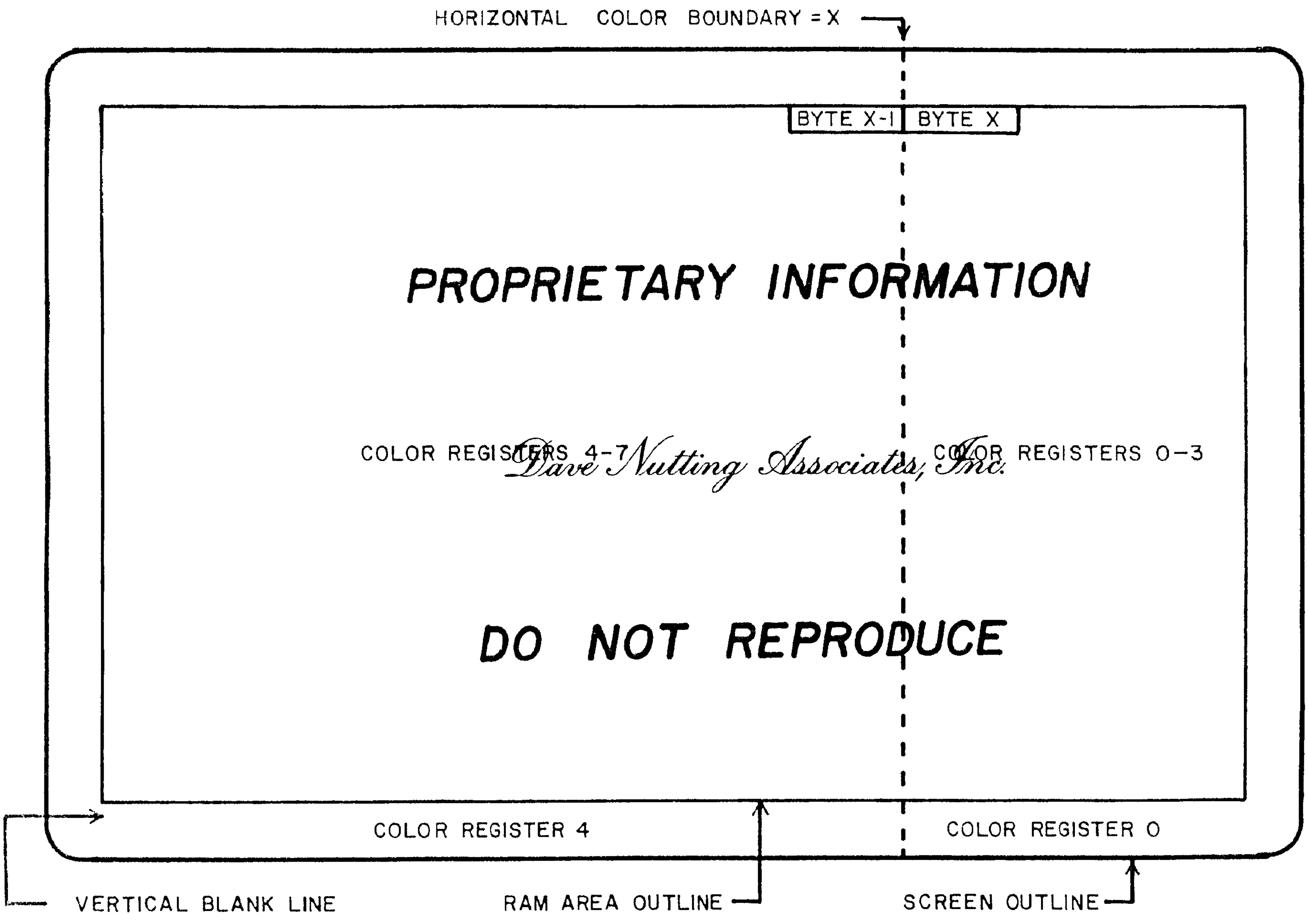
On most television the area defined by RAM is slightly smaller than the screen. There is generally extra space on all four sides of the RAM area. The color and intensity of this area is defined by the background color number (bits 6 and 7 of port 9). These two bits, along with the LEFT/RIGHT bit point 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 2Ø3 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 one or two of the external interrupt enable bits must be set (output port EH). 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 Ø 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 19H) 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

BYTE 0				BYTE 1				BYTE 2				
P3	P2	P1	P0	P7	P6	P5	P4	P11	P10	P9	P8	ORIGINAL DATA
0	P3	P2	P1	P0	P7	P6	P5	P4	P11	P10	P9	SHIFT 1
0	0	P3	P2	P1	P0	P7	P6	P5	P4	P11	P10	SHIFT 2
0	0	0	P3	P2	P1	P0	P7	P6	P5	P4	P11	SHIFT 3
P0	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11	FLOPPED

ROTATOR

The rotator is used to rotate a 4 X 4 pixel image 90° 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° from the original image.

The rotator can only be used in commercial mode.

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

P 3	P 2	P 1	P 0
P 7	P 6	P 5	P 4
P11	P10	P 9	P 8
P15	P14	P13	P12

ORIGINAL

P15	P11	P 7	P 3
P14	P10	P 6	P 2
P13	P 9	P 5	P 1
P12	P 8	P 4	P 0

ROTATED

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 occured 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

PORT ↓	7	6	5	4	3	2	1	0	← BIT
10H				TRIG	RIGHT	LEFT	DOWN	UP	PLAYER 1
11H				TRIG	RIGHT	LEFT	DOWN	UP	PLAYER 2
12H				TRIG	RIGHT	LEFT	DOWN	UP	PLAYER 3
13H				TRIG	RIGHT	LEFT	DOWN	UP	PLAYER 4
14H			=	+	−	X	÷	%	KEYPAD
15H			.	3	6	9	M+	∨	KEYPAD
16H			0	2	5	8	MS	∧	KEYPAD
17H			CE	1	4	7	MR	C	KEYPAD
1CH	← POT →								PLAYER 1
1DH	← POT →								PLAYER 2
1EH	← POT →								PLAYER 3
1FH	← POT →								PLAYER 4

MASTER OSCILLATOR

The frequency of the master oscillator is determined by the contents of several output ports. Port 1ØH sets the master frequency. It is given by the following formula:

$$F_m = \frac{1789}{\text{PORT 10H}+1}\ \text{Khz}$$

If bit 4 of output port 15H is set to 1, the master oscillator frequency will be modulated by noise. The amount of modulation will be set by the 8-bit noise volume register, output port 17H.

If bit 4 of output port 15H is set to Ø, the frequency of the master oscillator will be modulated by a constant value to give a vibrato effect. The amount of modulation will be set by the vibrato depth register (the first 6 bits of output port 14H). The speed of modulation is set by the vibrato speed register (upper 2 bits of output port 14H); ØØ for fastest and 11 for slowest.

Frequency modulation is accomplished by adding a modulation value to the contents of port 1ØH and sending the result to the master oscillator frequency generator. In noise modulation, the modulation value is an 8-bit word from the noise generator. If a bit in the noise volume register is set to Ø, the corresponding bit in the modulation value word will be set to Ø. In vibrato modulation, the modulation value alternates between Ø and the contents of the vibrato volume register.

Modulation can be completely disabled by setting the master volume to Ø if noise modulation is being used, or by setting the vibrato depth to Ø when vibrato is used.

TONES

The system contains three tone generators each clocked by the same master oscillator. The frequency of Tone A is set by output port 11H, Tone B by output port 12H, and Tone C by output port 13H. The frequency is given by the following formula:

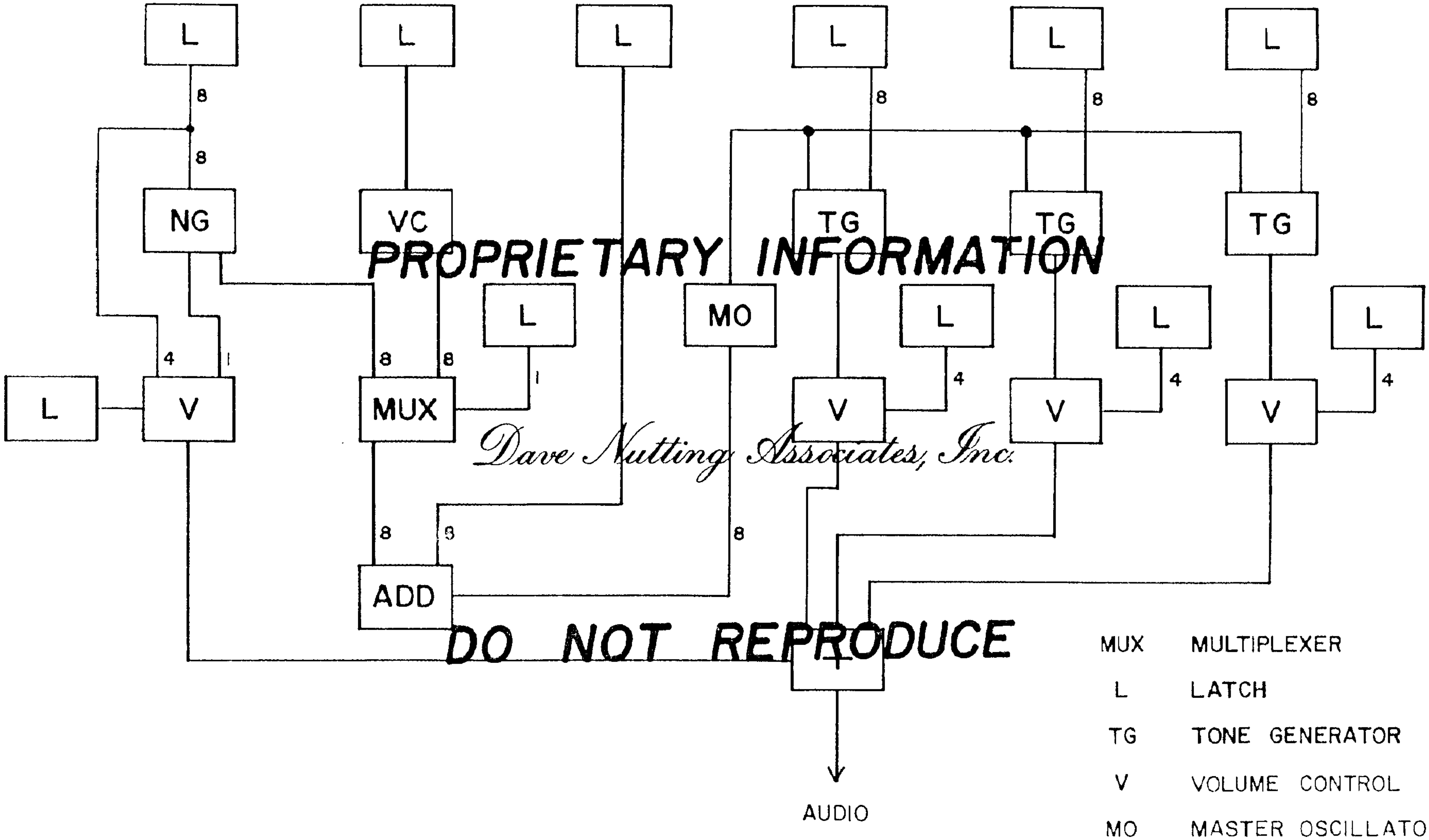
$$F_t = \frac{F_m}{2(\text{contents of TONE PORT}+1)} = \frac{894}{(\text{PORT 10H}+1)(\text{contents of TONE PORT}+1)} \text{ Khz}$$

The tone volumes are controlled by output ports 15H and 16H. The lower 4 bits of port 16H set Tone A Volume, the upper 4 bits sets Tone B Volume. The lower 4 bits of port 15H sets Tone C Volume. Noise can be mixed with the tones by setting bit 5 of port 15H to 1. In this case the noise volume is given by the upper 4 bits of port 17H. In all cases a volume of Ø is silence and a volume of all 1's is loudest.

SOUND BLOCK TRANSFER

All 8 bytes of sound control can be sent to the audio circuit with one OTIR instruction. Register C should be sent to 18H, register B to 8H and HL pointing to the 8 bytes of data. The data pointed to by HL goes to port 17H and the next 7 bytes of data goes to ports 16H through 1ØH.

HL →	Memory Location	X	Data-to-port	17H
		X+1	Data-to-port	16H
		X+2	Data-to-port	15H
		X+3	Data-to-port	14H
		X+4	Data-to-port	13H
		X+5	Data-to-port	12H
		X+6	Data-to-port	11H
		X+7	Data-to-port	1ØH



MUX	MULTIPLEXER
L	LATCH
TG	TONE GENERATOR
V	VOLUME CONTROL
MO	MASTER OSCILLATOR
VC	VIBRATO CONTROL
NG	NOISE GENERATOR
ADD	ADDER

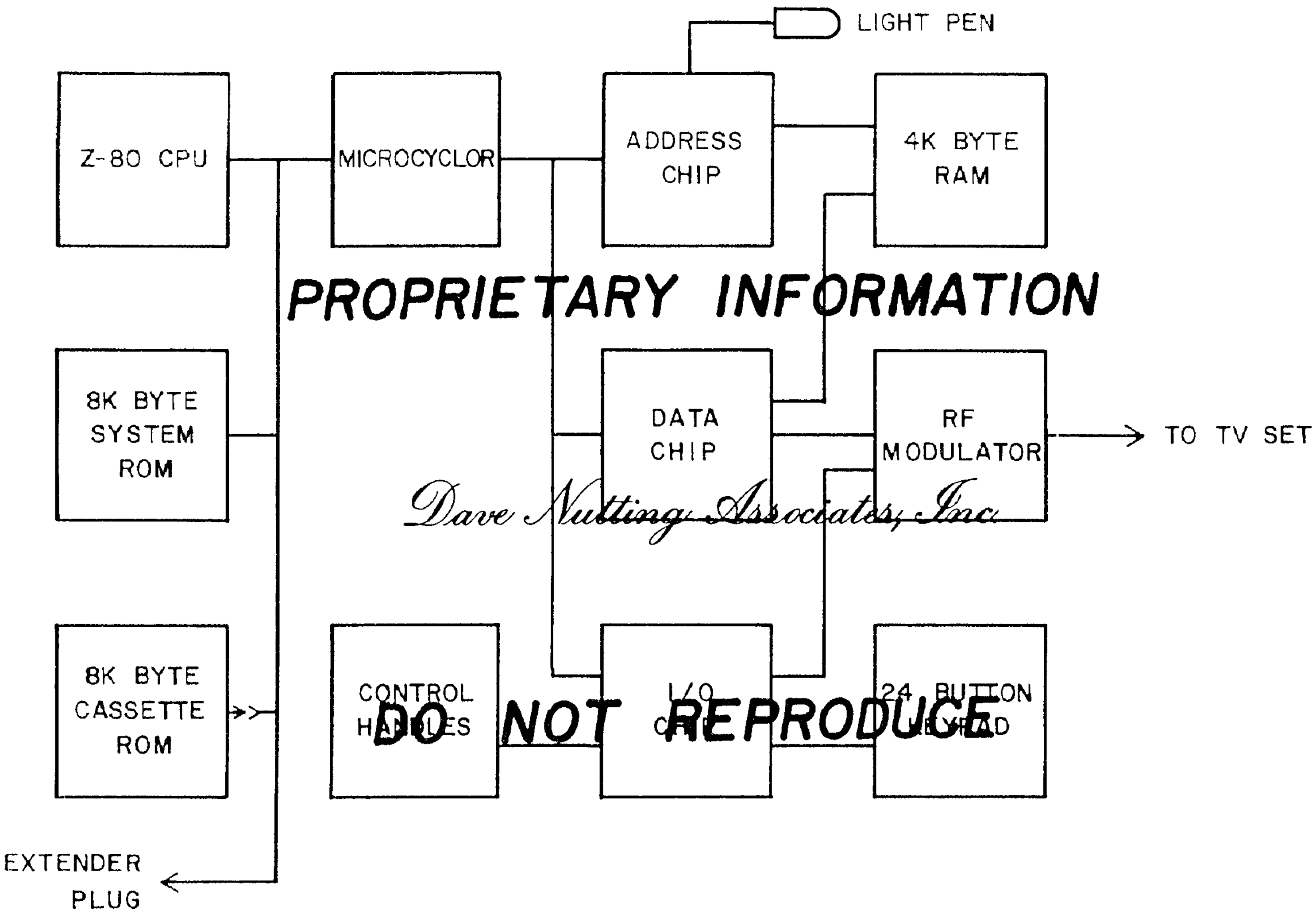
AUDIO GENERATOR BLOCK DIAGRAM

OUTPUT PORTS

PORT NUMBER	FUNCTION
ØH	Color Register Ø
1H	Color Register 1
2H	Color Register 2
3H	Color Register 3
4H	Color Register 4
5H	Color Register 5
6H	Color Register 6
7H	Color Register 7
8H	Low/High Resolution
9H	Horizontal Color Boundary, Background Color
AH	Vertical Blank Register
BH	Color Block Transfer
CH	Magic Register
DH	Interrupt Feedback Register
EH	Interrupt Enable and Mode
FH	Interrupt Line
1ØH	Master Oscillator
11H	Tone A Frequency
12H	Tone B Frequency
13H	Tone C Frequency
14H	Vibrato Register
15H	Tone C Volume, Noise Modulation Control
16H	Tone A Volume, Tone B Volume
17H	Noise Volume Register
18H	Sound Block Transfer
19H	Expand Register

INPUT PORTS

PORT NUMBER	FUNCTION
8H	Intercept Feedback
EH	Vertical Line Feedback
FH	Horizontal Address Feedback
1ØH	Player 1 Handle
11H	Player 2 Handle
12H	Player 3 Handle
13H	Player 4 Handle
14H	Keypad Column Ø (right)
15H	Keypad Column 1
16H	Keypad Column 2
17H	Keypad Column 3 (left)



SYSTEM BLOCK DIAGRAM

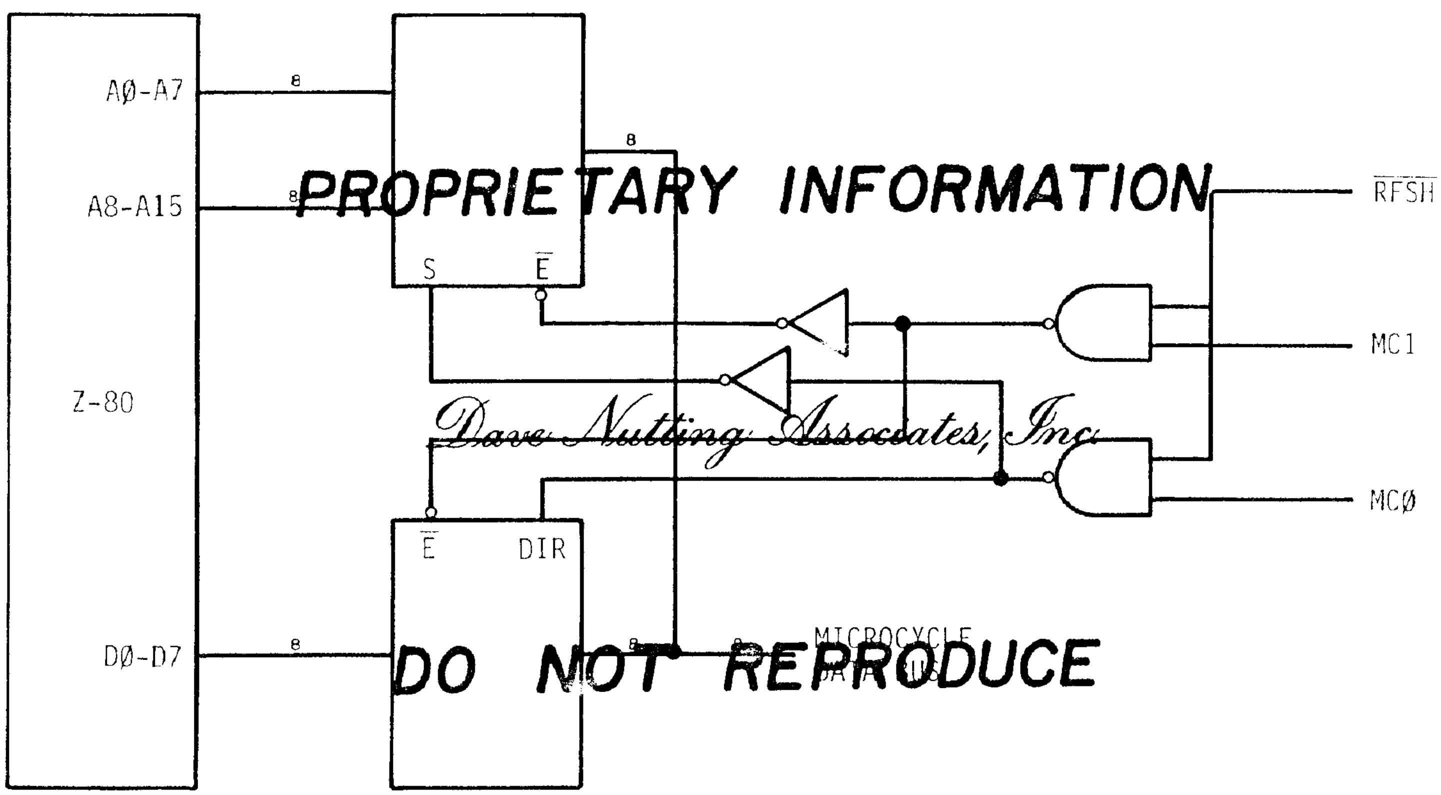
MICROCYCLER

The purpose of the microcycler is to combine the 16-bit Address Bus and the 8-bit Data Bus from the Z-8Ø into one 8-bit Microcycle Data Bus to the Data Chip, Address Chip, and I/O Chip. This was done to reduce the pin count on the custom chips.

The Microcycle Data Bus can be in any of four modes. Its mode is controlled by MCØ and MC1 coming from the Data Chip and $\overline{\text{RFSH}}$ from the Z-8Ø. The modes are shown below.

$\overline{\text{RFSH}}$	MCØ	MC1	Microcycle Data Bus Contents
Ø	Ø	Ø	AØ - A7 from Z-8Ø
Ø	Ø	1	AØ - A7 from Z-8Ø
Ø	1	Ø	AØ - A7 from Z-8Ø
Ø	1	1	AØ - A7 from Z-8Ø
1	Ø	Ø	AØ - A7 from Z-8Ø
1	Ø	1	A8 - A15 from Z-8Ø
1	1	Ø	DØ - D7 from Z-8Ø
1	1	1	DØ - D7 to Z-8Ø

MCØ and MC1 change 140 nsec after the rising edge of Φ. Their changes are shown in the timing diagrams of various instruction cycles.



MICROCYCLER BLOCK DIAGRAM

ADDRESS CHIP DESCRIPTION

The Microcycle Decoder generates twelve bits of Z-8Ø address from the 8-bit Microcycle Data Bus. This address is then fed through MUX I and MUX II to MAØ-5 which go to the RAM. The Scan Address Generator generates a 12-bit address which is used to read video data from the RAM. This address goes from Ø to FFFH once every frame (1/60 sec.).

MUX I sends either the Scan Address or Z-8Ø Address to its 12 outputs. An output of the Scan Address Generator controls MUX I. If the Scan Address Generator and the Z-8Ø request a memory cycle at the same time, the Scan Address Generator will have higher priority and the Z-8Ø will be required to wait (by the $\overline{\text{WAIT}}$ output). The Scan Address Generator never requires the memory for more than one consecutive memory cycle, so the Z-8Ø is never required to wait for the memory for more than one cycle. HORIZ DR and VERT DR synchronize the Scan Address Generator with the Data Chip and the TV Scan.

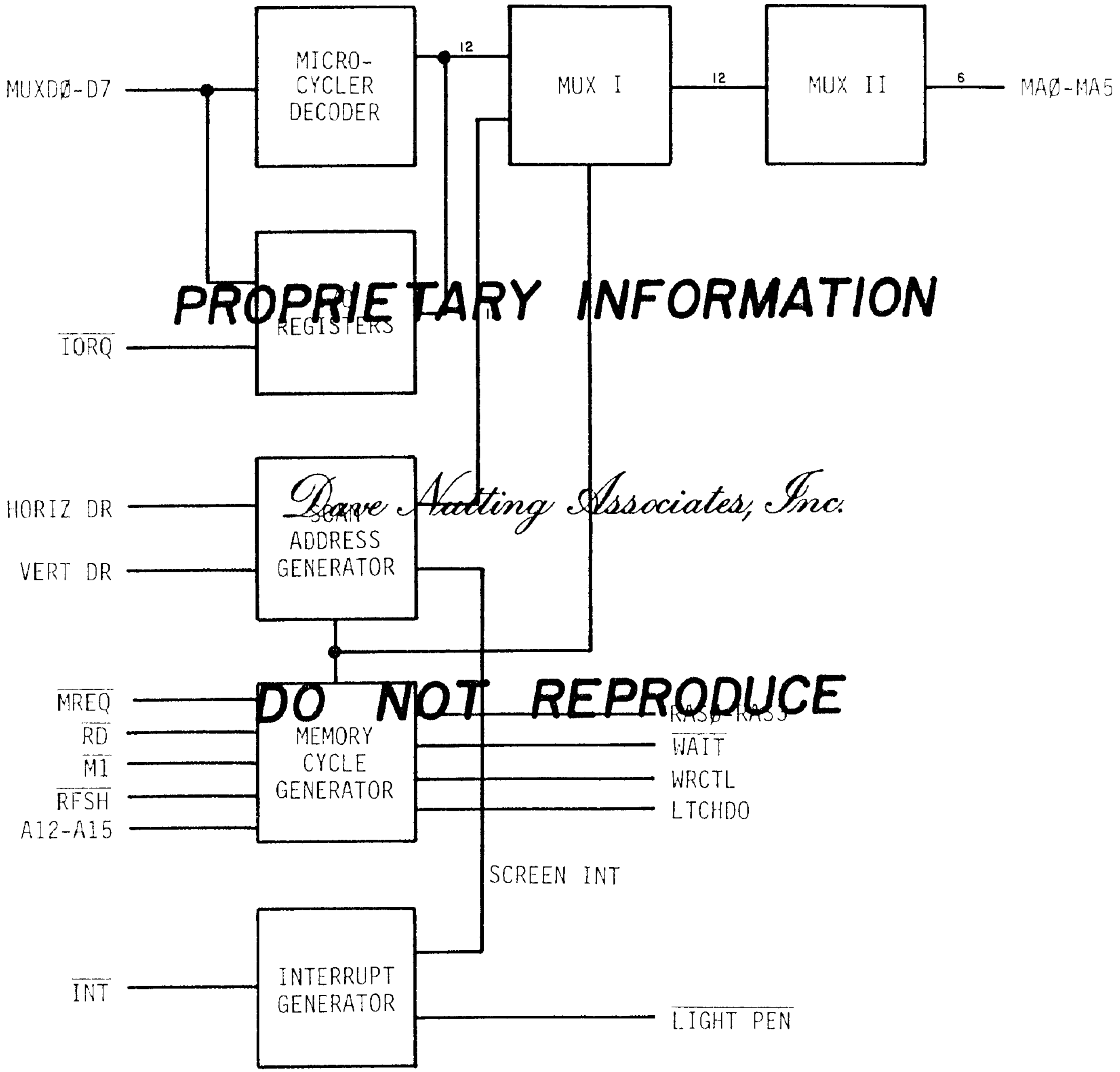
The purpose of MUX II is to multiplex its 12 inputs to the six address bits in the two time slices required for 4K x 1 16 pin RAMS.

The Memory Cycle Generator controls memory cycles generated by either the Z-8Ø or Scan Address Generator. $\overline{\text{MREQ}}$, $\overline{\text{RD}}$, $\overline{\text{M1}}$, $\overline{\text{RFSH}}$, and A12-A15 are from the Z-8Ø. A12-A15 are fed directly from the Z-8Ø because if they were brought out of the microcycle decoder, they would arrive too late in the memory cycle. The RASØ - RAS3 outputs are used to activate memory cycles. In the consumer game, only RASØ is used to one bank of RAM (4K x 8). In the commercial game, all four RAS's are used to control four banks of RAM (16K x 8). WRCTL and LTCHDO are control signals to the Data Chip. WRCTL tells the Data Chip when to place data to be written to memory on the memory data bus. LTCHDO tells the Data Chip when valid data from RAM is present on the memory data bus.

As mentioned earlier, WAIT is generated when the Z-8Ø and Scan Address Generator both request memory at the same time. WAIT is also generated for one cycle every time the Z-8Ø requests a memory access, even if there is no conflict with the Scan Address. This is because the microcycler slows down Z-8Ø memory accesses. The Z-8Ø address bus and data bus must time share the microcycle bus so the Z-8Ø data reaches the microcycle bus very late in the memory cycle.

The INT Generator generates two types of interrupts to the Z-8Ø; Light Pen and Screen interrupts. A screen interrupt is generated when screen interrupts are enabled and the TV scan completes a certain line on the screen (from Ø to 255). The line at which the interrupt will occur is determined by the Z-8Ø. This interrupt can be used for timing since the TV rescans every line once every 1/60 sec. A light pen interrupt occurs when the light pen interrupt is enabled and LIGHT PEN goes low. The current scan address is saved in latches in the Scan Address Generator. The Z-8Ø can read the contents of these latches to determine the scan address at the time LIGHT PEN was activated and thus the position of the light pen on the screen.

The I/O Decode circuit is used during Z-8Ø input and output instructions. Z-8Ø input instructions are used to read the scan address after light pen interrupts. Output instructions are used to enable the two interrupts and set the line number for screen interrupts.



ADDRESS CHIP BLOCK DIAGRAM

DATA CHIP DESCRIPTION

The TV Sync Generator uses 7M and $\overline{7M}$ (7.159090 Mhz square waves) to generate NTSC standard sync and blank to be sent to the Video Generator. It also generates HORIZ DR and VERT DR for synchronization with the Address Chip. HORIZ DR occurs once every horizontal line (63.5 usec), and VERT DR occurs once every frame (16.6 msec).

The Shift Register loads parallel data from the memory data bus (MDØ - MD7) and shifts it out of its two serial outputs. The TV sync Generator controls when data is loaded or shifted. In a consumer game, the two outputs of the shift register are sent through MUX I to MUX II. In a commercial game, SERIAL Ø and SERIAL 1 are sent through the MUX I to MUX II. The two bits from MUX I select 8 bits to be sent through MUX II to the Video Generator. These 8 bits then determine the analog values of VIDEO, R-Y, and B-Y. 2.5V is a 2.5V D C reference level.

The Clock Generator generates ØG and $\overline{PX}$ from 7M. These are the clocks for the rest of the system. The frequency of $\overline{PX}$ is half that of 7M and the frequency of ØG is half that of $\overline{PX}$.

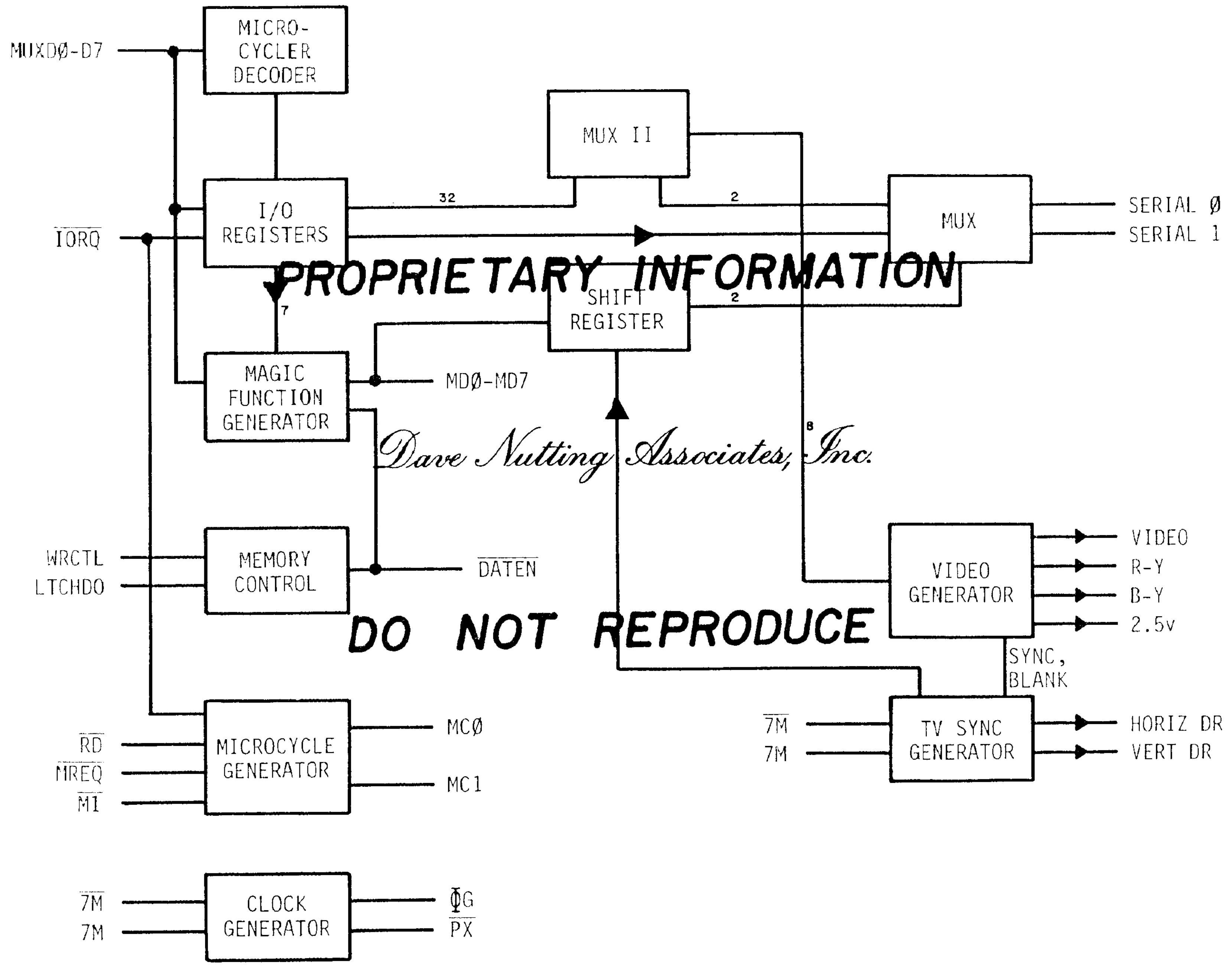
The Microcycle Generator generates the microcycle control bits, MCØ and MC1, from $\overline{IORQ}$, $\overline{MREQ}$, $\overline{RD}$, and $\overline{M1}$, all from the Z-8Ø.

In memory write cycles WRCTL is activated and the Memory Control circuit generates $\overline{DATEN}$. The Magic Function Generator takes the data from the Z-8Ø on MUXDØ - D7 and transfers it to MDØ - MD7. If a Magic write is being done, the Magic Function Generator will modify the data as required before it places it on the memory data bus.

A Magic write is a memory write cycle in which data is written to a location, (X) from Ø to 16K. All memory from Ø to 16K is ROM and cannot be modified. The data is modified by the Magic Function Generator and is written to location X + 16K. The way in which the data is modified is determined by the 7 bits coming from the I/O registers.

In memory reads, data is transferred from MDØ - MD7 to MUXDØ - MUXD7. Also, LTCHDO is activated which causes the data from RAM to be latched up in a register in the Magic Function Generator. This latched data is used in some magic functions.

The I/O registers are loaded by output instructions from the Z-8Ø just as in the Address Chip.



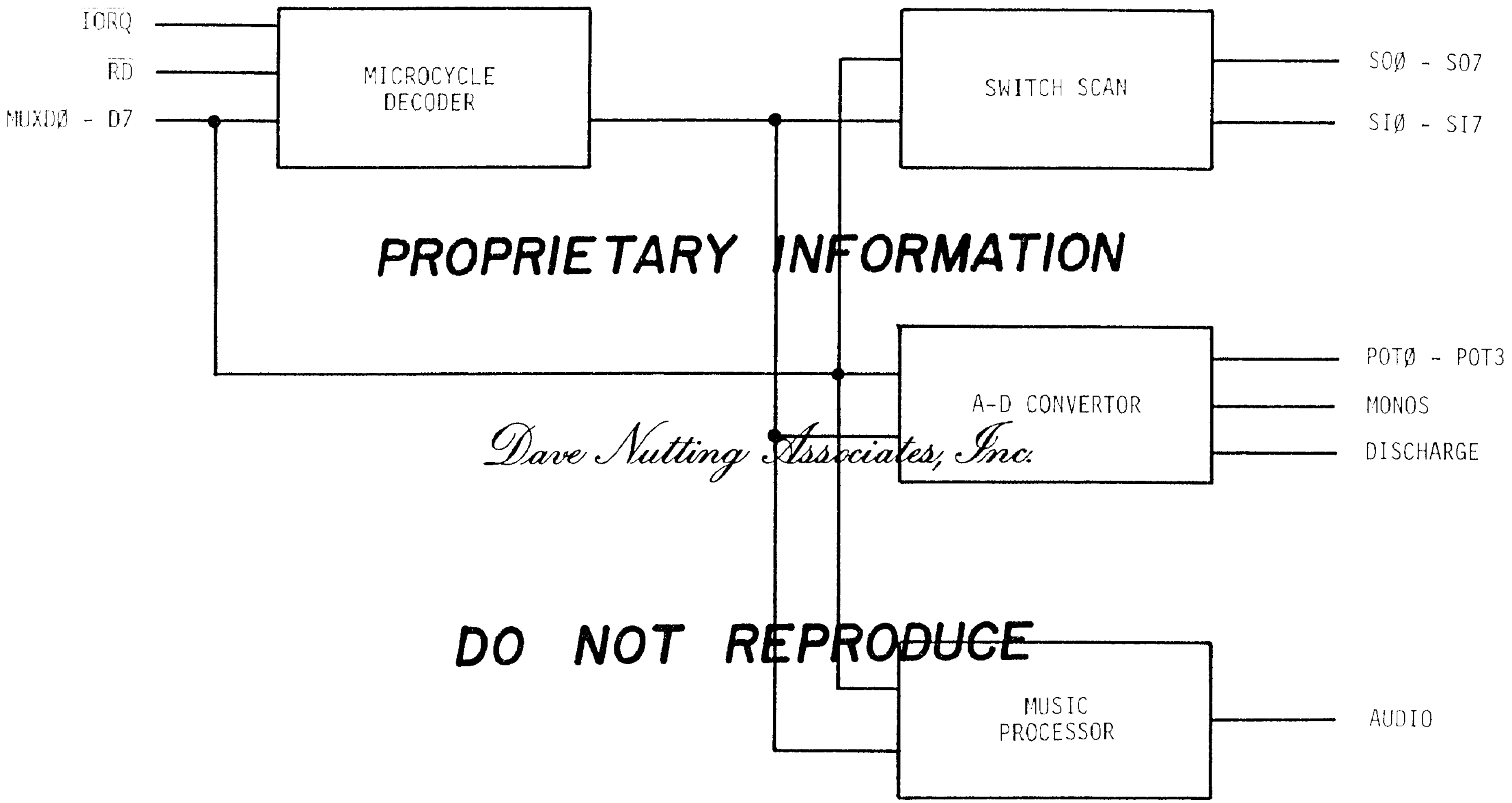
DATA CHIP BLOCK DIAGRAM

I/O CHIP DESCRIPTION

The Z-8Ø communicates with the I/O Chip through input and output instructions. The state of an 8 x 8 switch matrix can be read through the Switch Scan circuit. When an input instruction is executed, one of the SOØ-SO7 lines will be activated. When a line is activated, the switch matrix will feed back eight bits of data on SIØ-SI7. This data is in turn fed to the Z-8Ø through MUXDØ - MUXD7.

The Z-8Ø can read the position of four potentiometers (pots) through the A-D Converter circuit. The pots are continuously scanned by the A-D Converter and the results of the conversions are stored in a RAM in the A-D Converter circuit. The Z-8Ø simply reads this RAM with input instructions.

The Z-8Ø loads data into the Music Processor with output instructions. This data determines the characteristics of the audio that is generated. The Music Processor is described in detail below.



I/O CHIP BLOCK DIAGRAM

MUSIC PROCESSOR

The music processor can be divided into two sections. The first section generates the Master Oscillator Frequency and the second section uses the Master Oscillator Frequency to generate tone frequencies and the analog audio output. The contents of all registers in the Music Processor are set by output instructions from the Z-8Ø.

Master Oscillator Frequency is a square wave whose frequency is determined by the 8 binary inputs to the Master Oscillator. The 8-bit word is the sum of the contents of the Master Oscillator Register and the output of the MUX. The MUX is controlled by MUX REG.

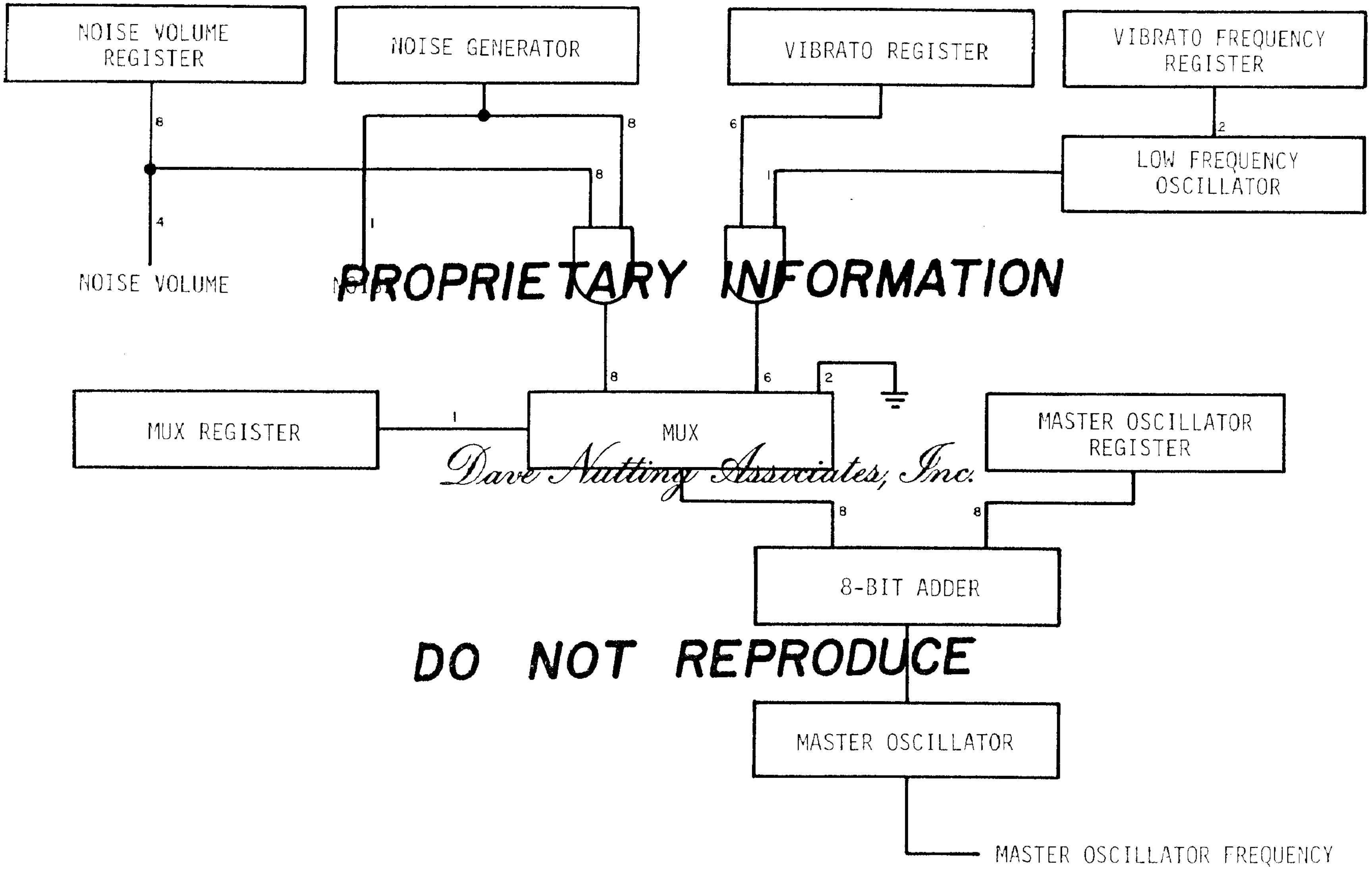
If MUX REG contains Ø, then data from the Vibrato System will be fed through the MUX. The two bits from the Vibrato Frequency Register determine the frequency of the square wave output of the Low Frequency Oscillator. The 6-bit word at the output of the AND gates oscillates between Ø and the contents of the Vibrato Register. The frequency of oscillation is determined by the contents of the Vibrato Frequency Register. The 6-bit word, along with two ground bits are fed through the MUX to the Adder. This causes the Master Oscillator Frequency to be modulated between two values thus giving a vibrato effect.

If MUX REG contains 1, then data from the Noise System will be fed through the MUX. The 8-bit word from the Noise Volume Register determines which bits from the Noise Generator will be present at the output of the AND gates.

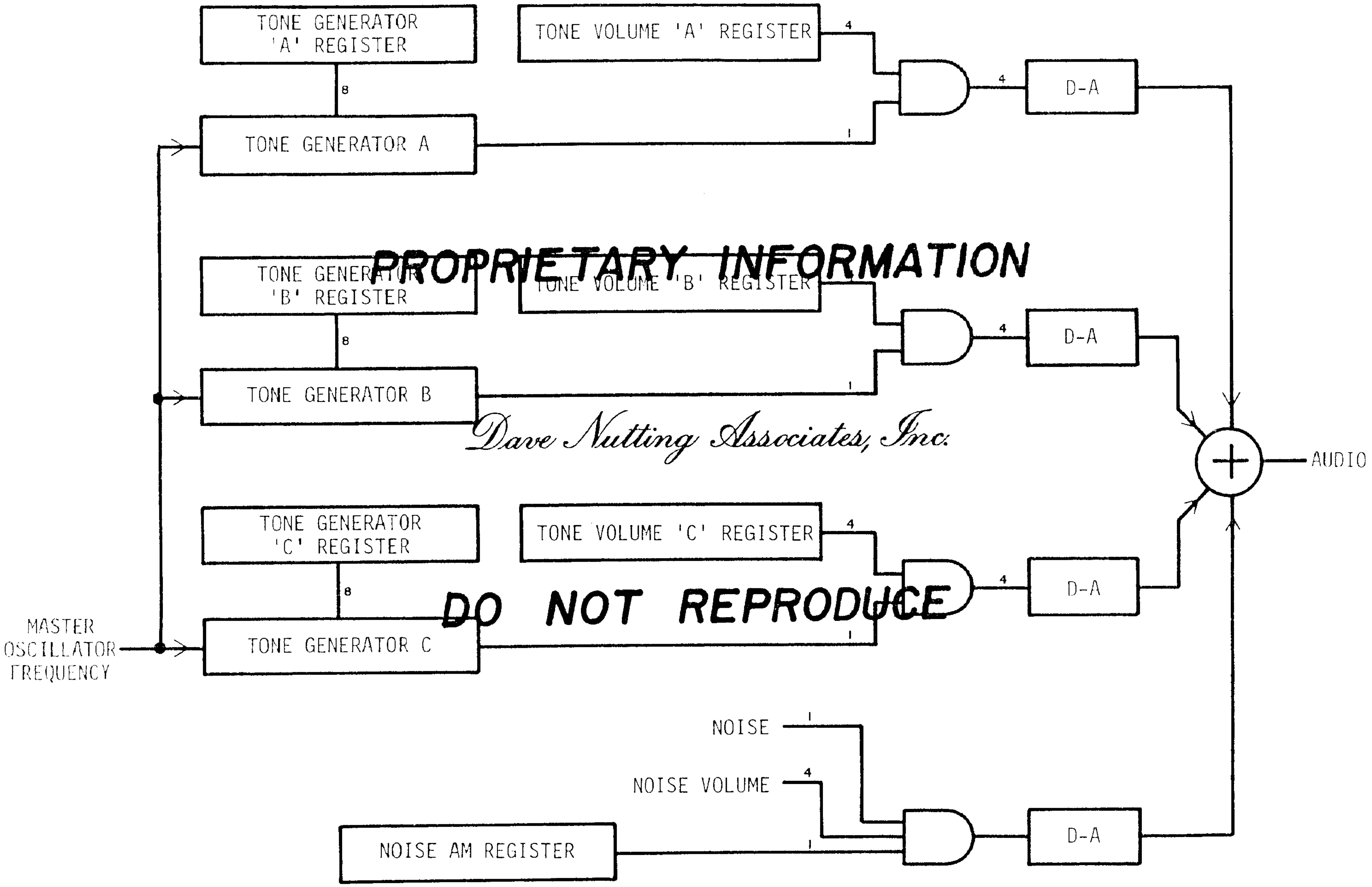
If a bit in the Noise Volume Register is Ø, then the corresponding bit at the output of the AND gates will be Ø. If a bit in the Noise Volume Register is 1, then the corresponding bit at the output of the AND gates will be noise from the Noise Generator. This 8-bit word is sent through the MUX to the Adder. The Master Oscillator Frequency is modulated by noise.

In the second part of the Music Processor, the square wave from the Master Oscillator is fed to three Tone Generator circuits which produce square waves at their outputs. The frequency of their outputs is determined by the contents of their Tone Generator Register and Master Oscillator Frequency. The 4-bit words at the output of the AND gates oscillate between Ø and the contents of the Tone Volume Register. These 4-bit words are sent to D-A Converters whose outputs oscillate between GND and a positive analog voltage determined by the contents of the Tone Volume Register.

One Noise bit and four Noise Volume bits from the first section of the Music Processor are fed to a set of AND gates. This set of AND gates operates the same way as the AND gates for the tones except that the Noise AM Register must contain a 1 for the outputs of the AND gates to oscillate. The analog outputs of the four D-A Converters are summed to produce the single audio output.



MASTER OSCILLATOR



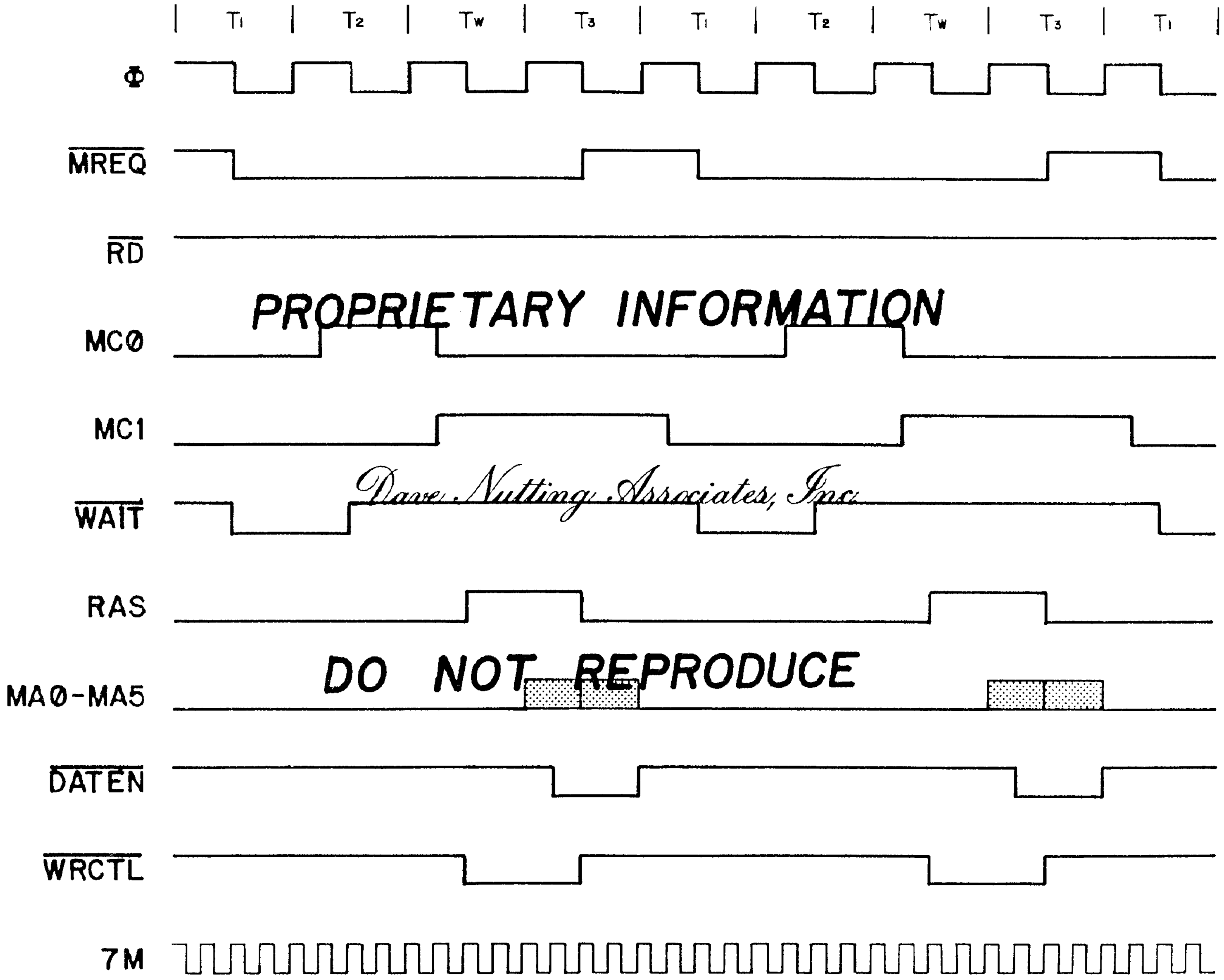
TONE GENERATORS

CUSTOM CHIP TIMING

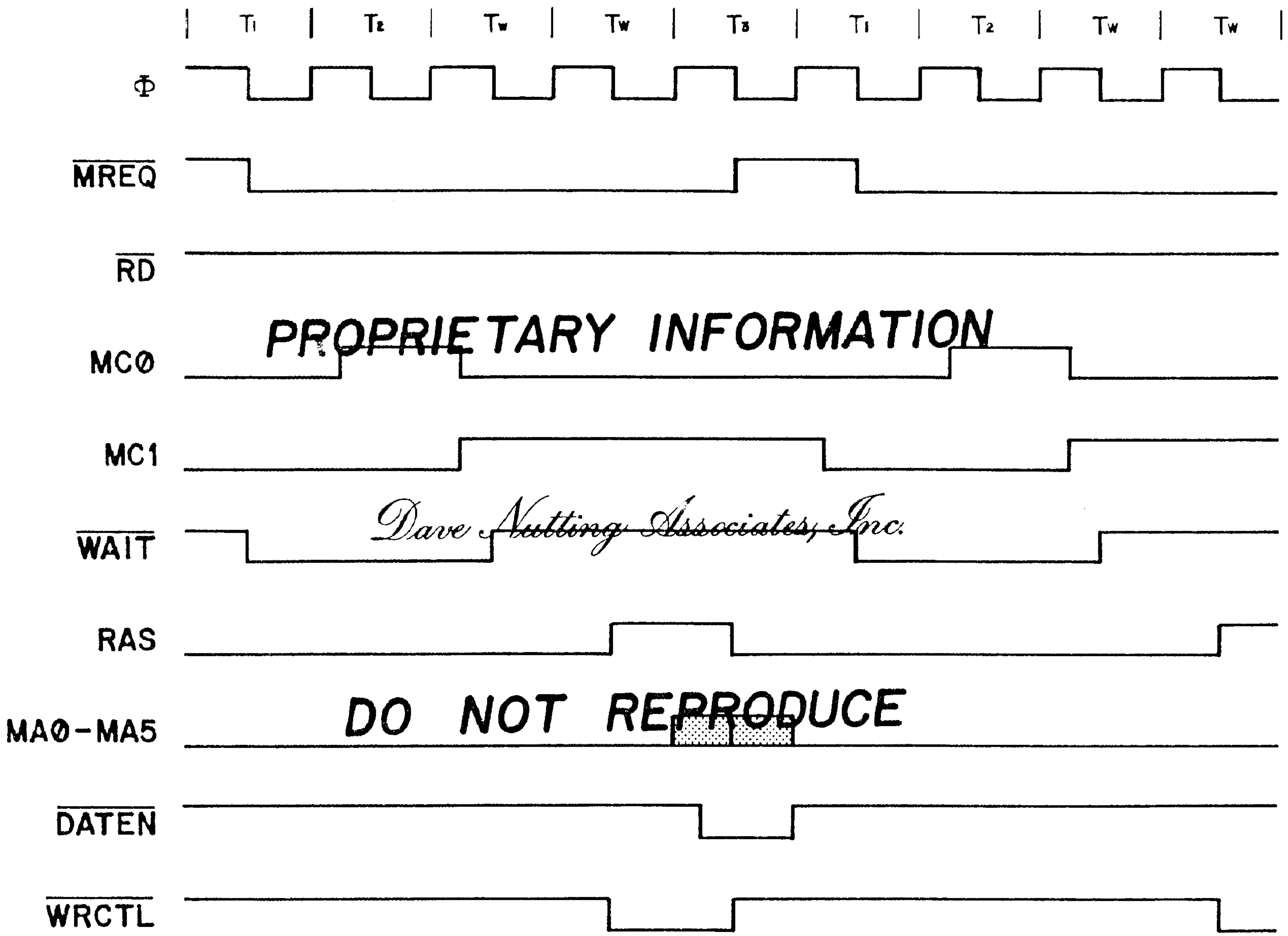
The following diagrams show the relationship of various signals in the system during different types of operations. Delays are shown to be zero nsec from the clock edge which causes the transition. The actual delay is given in "Electrical Specification for Midway Custom Circuits".

MUXDØ - MUXD7 is a 8-bit bidirectional address and data bus for the custom chips. By using this technique 16 bits of address and 8 bits of data can be sent to the custom chips on 8 wires. The state of the bus is determined by MCØ and MC1 from the data chip and $\overline{\text{RFSH}}$ from the Z-80.

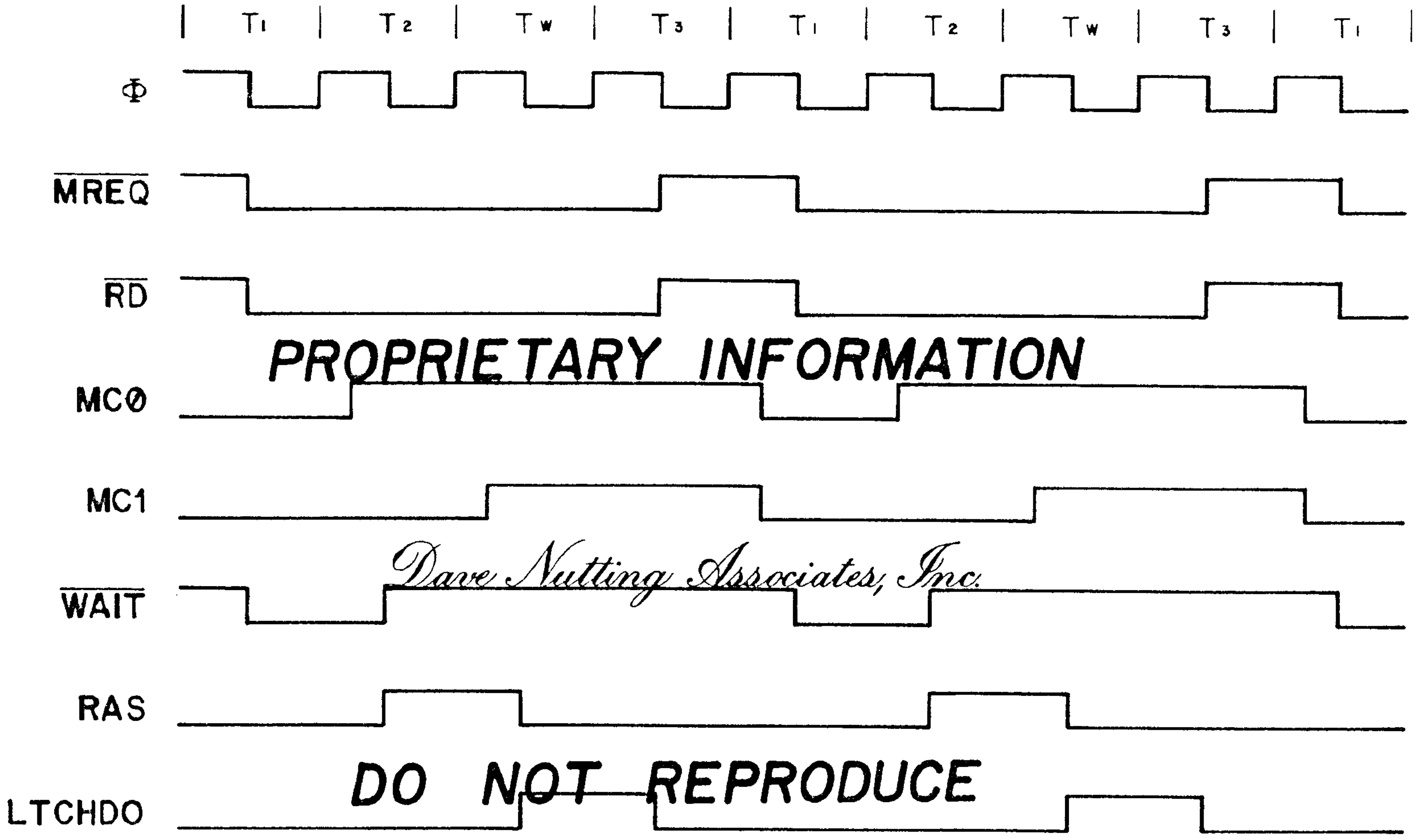
$\overline{\text{RFSH}}$	MC1	MCØ	
L	L	L	AØ - A7 to custom chips.
L	L	H	AØ - A7 to custom chips
L	H	L	AØ - A7 to custom chips
L	H	H	AØ - A7 to custom chips
H	L	L	AØ - A7 to custom chips
H	L	H	A8 - A15 to custom chips
H	H	L	DØ - D7 to custom chips
H	H	H	DØ - D7 from custom chips



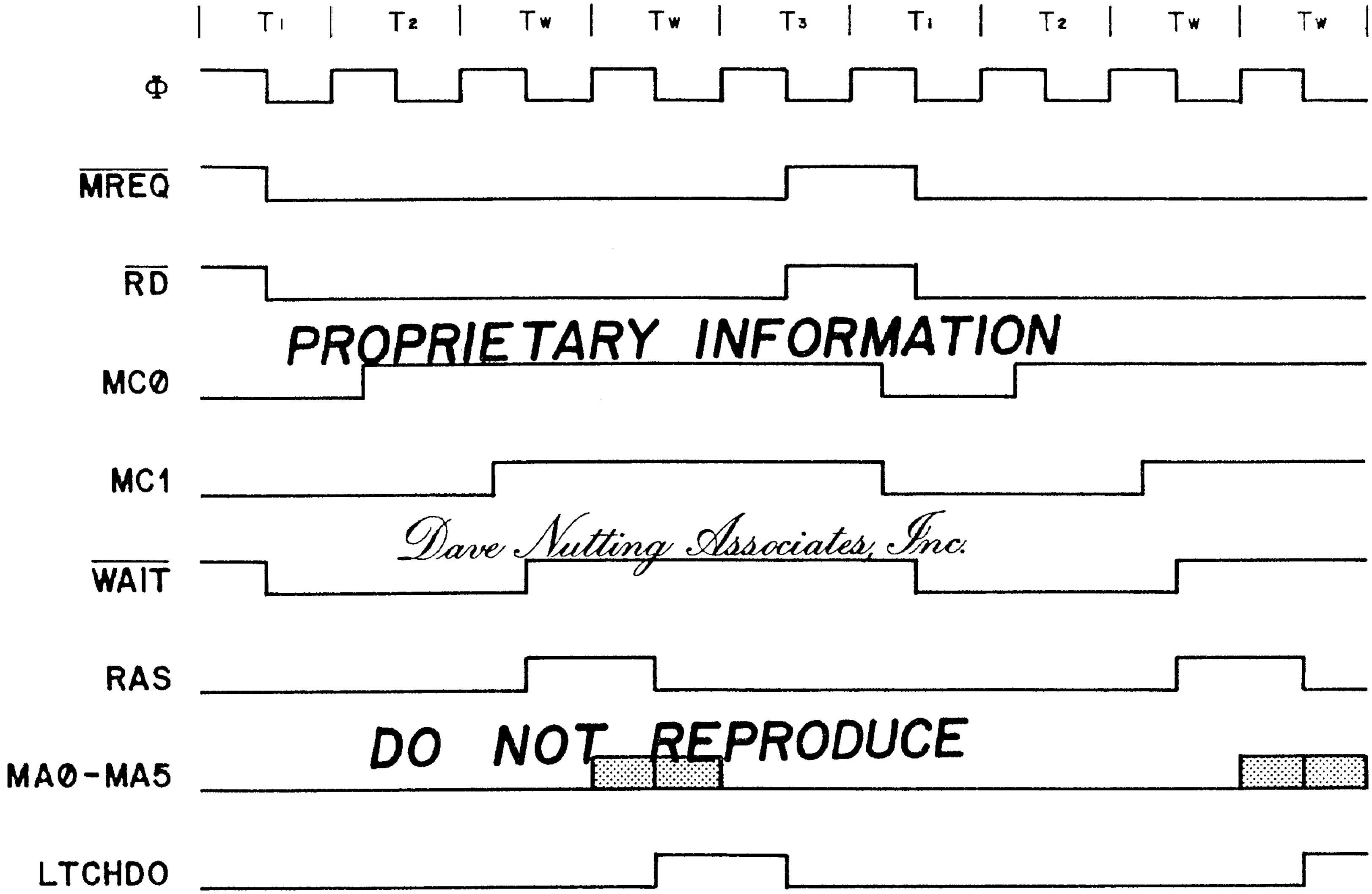
MEMORY WRITE WITHOUT EXTRA WAIT STATE



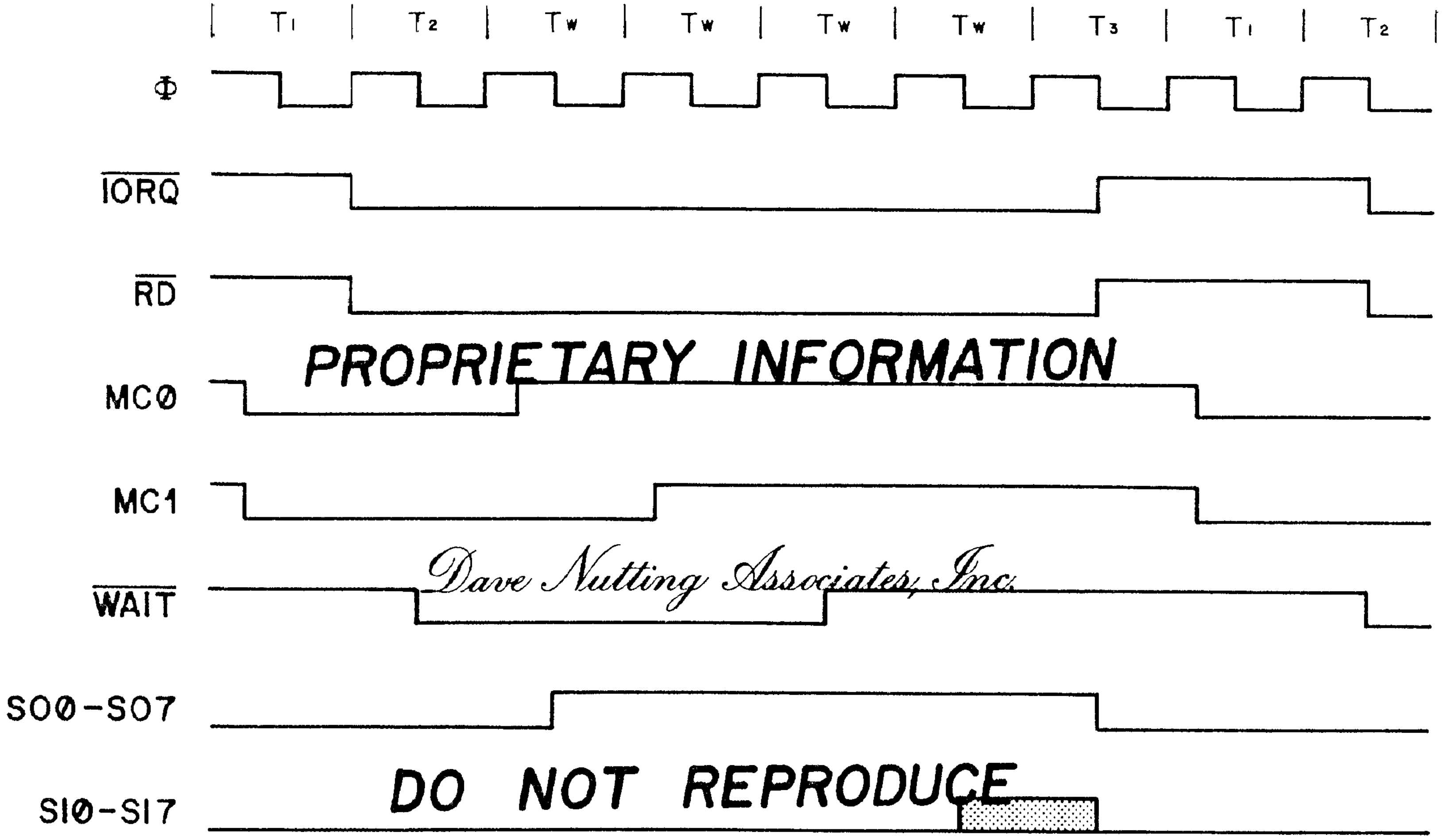
MEMORY WRITE WITH VIDEO WAIT STATE



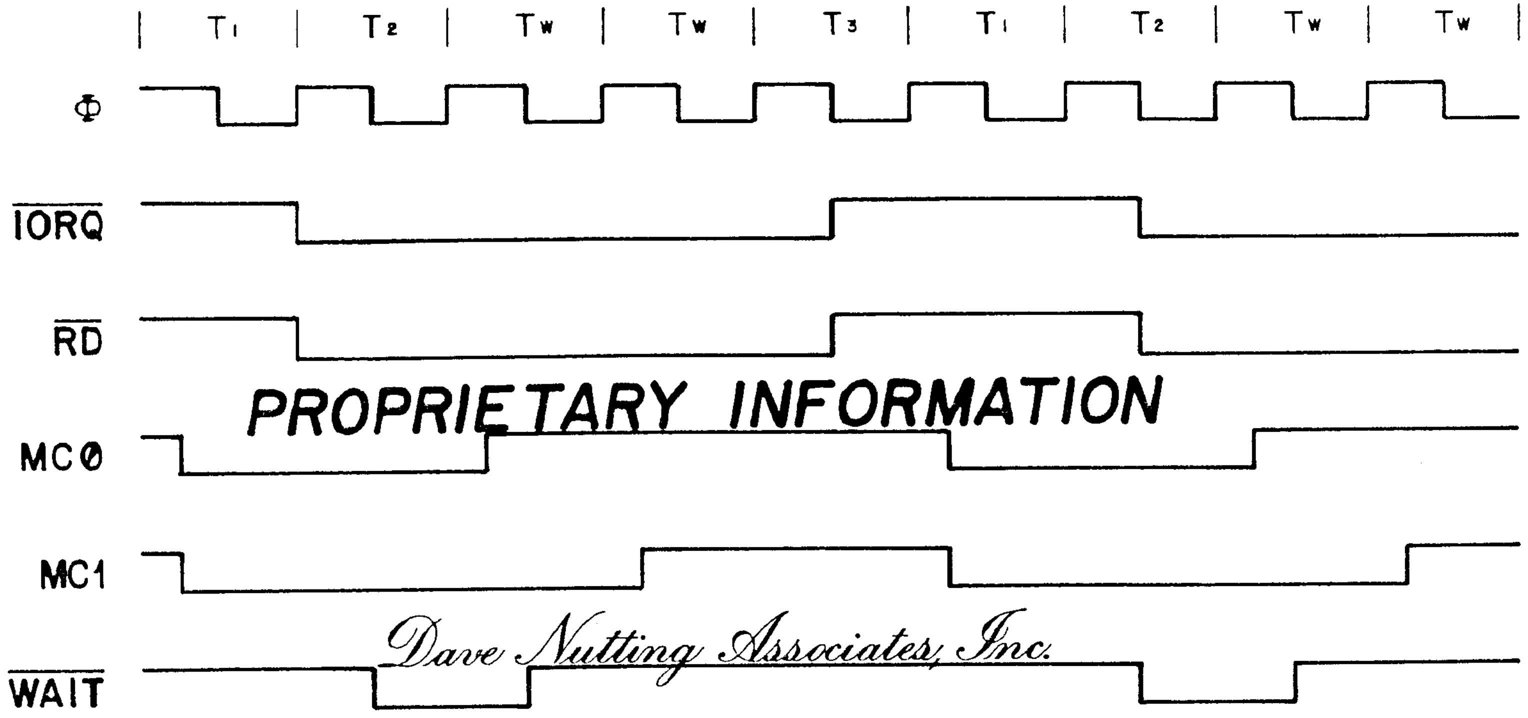
MEMORY READ WITHOUT EXTRA WAIT STATE



MEMORY READ WITH VIDEO WAIT STATE



I/O READ FROM PORT 10H-17H

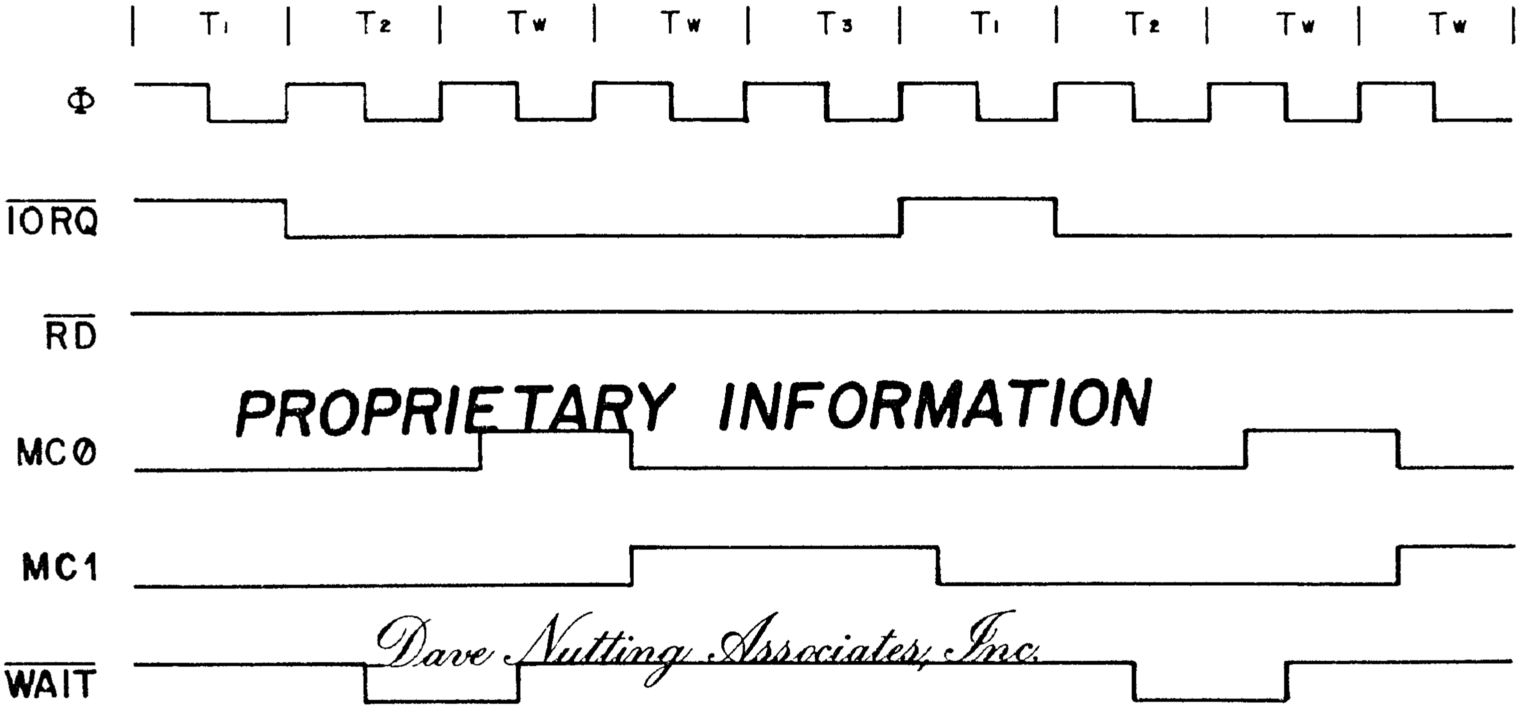


PROPRIETARY INFORMATION

Dave Nutting Associates, Inc.

DO NOT REPRODUCE

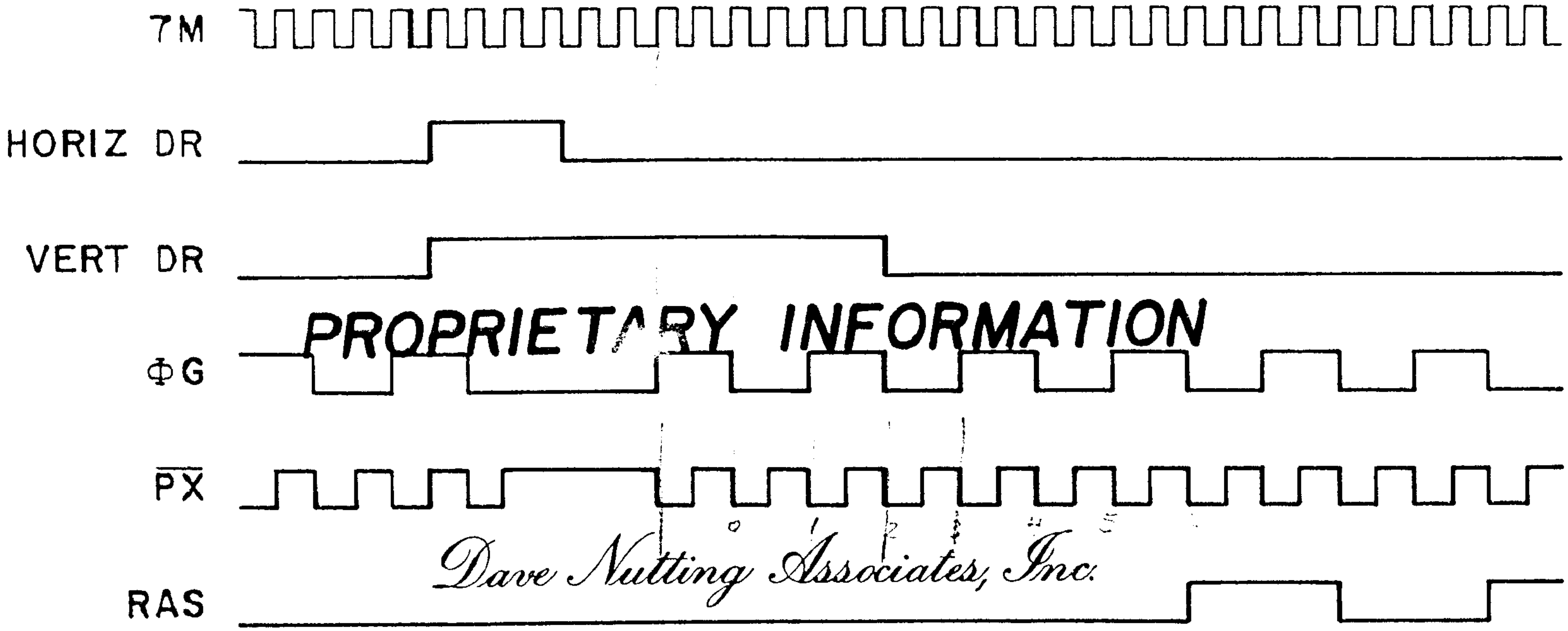
I/O READ FROM OTHER THAN PORT 10H-17H



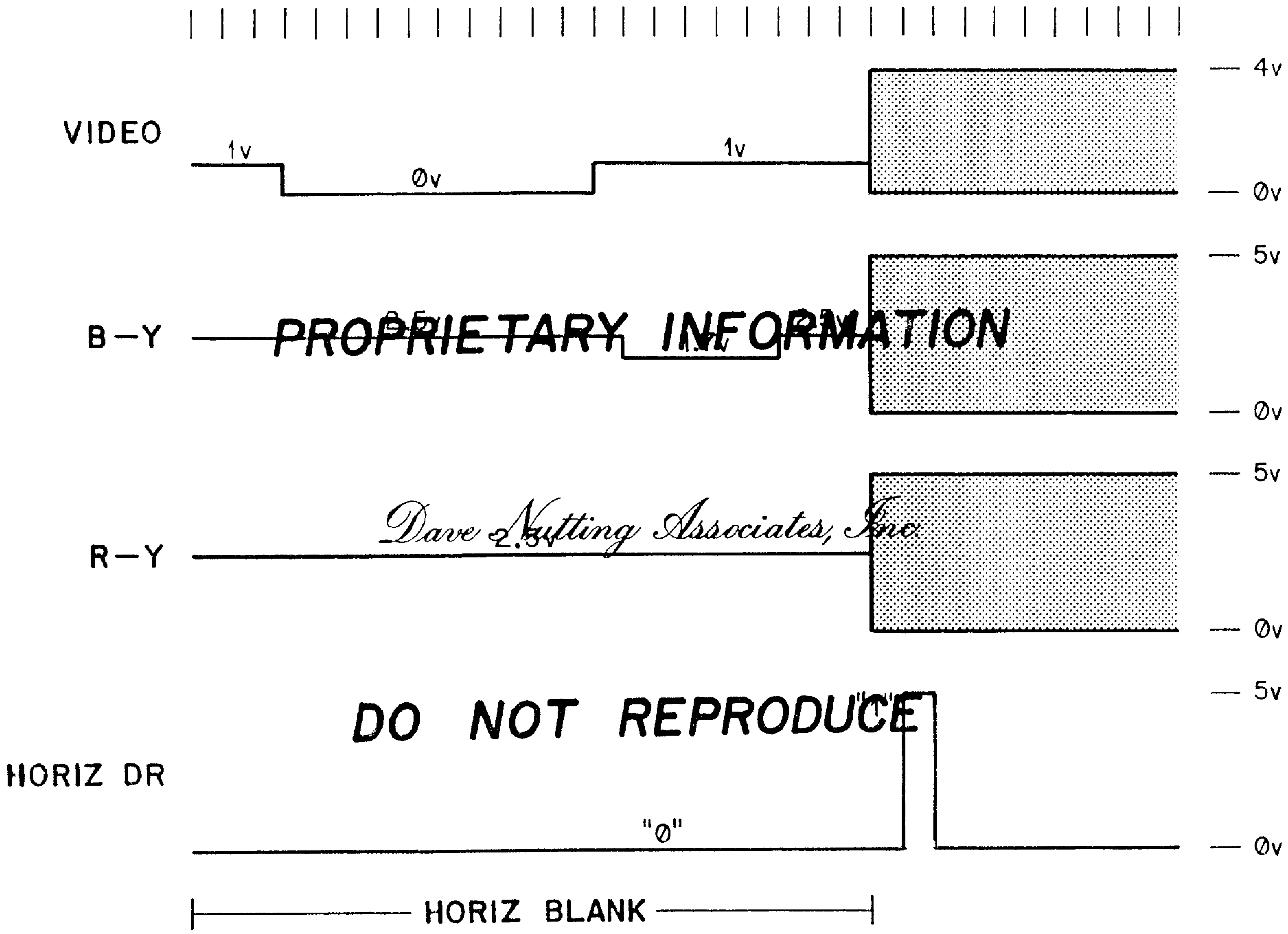
I/O WRITE

VIDEO TIMING

The frequency of $\overline{PX}$ is half that of 7M and the Ø is one-fourth 7M. There are 455 cycles of 7M per horizontal line and 113 3/4 Φ cycles per line. Because of the extra 3/4 cycle Ø must be resynchronized at the beginning of each line. This is done by stalling Ø for 3 cycles of 7M. $\overline{PX}$ is also stalled for the same amount of time. The timing relationship is shown below. The diagram also shows the relationship of VERT DR to HORIZ DR. The two RAS pulses shown are the first two video RAS signals of a line, each line contains forty.



RELATIONSHIP BETWEEN 7M, HORIZ DR, VERT DR, ΦG, $\overline{PX}$ AND RAS

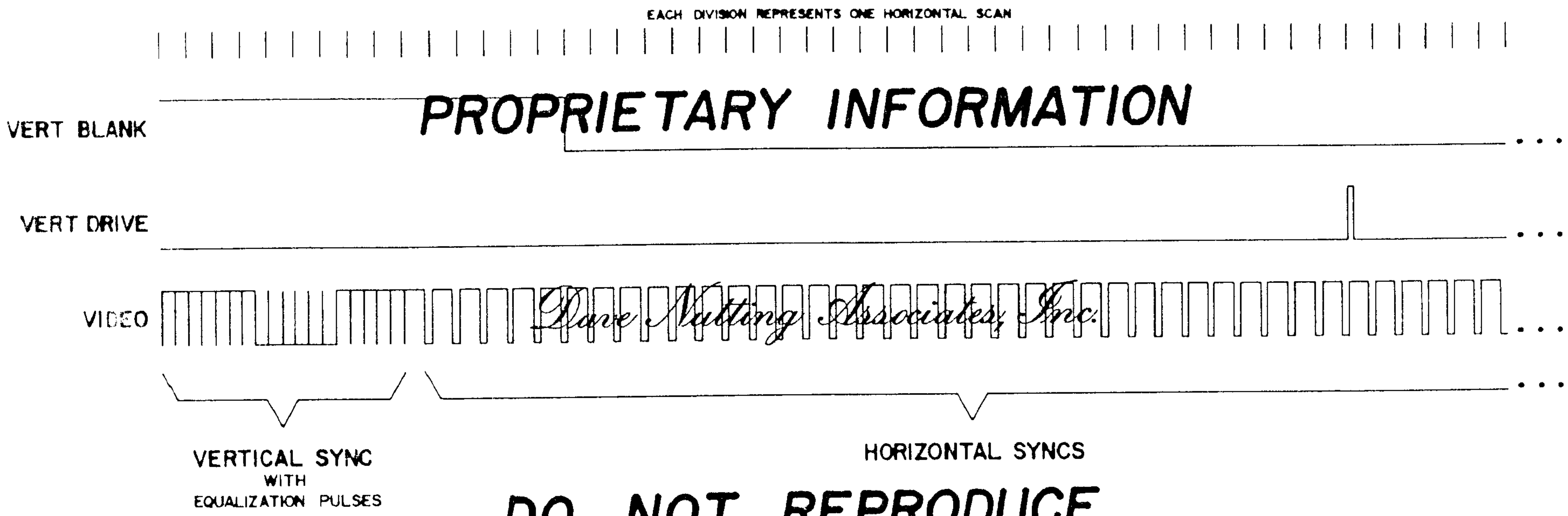


RELATIONSHIP BETWEEN HORIZ DR, HORIZ BLANK, HORIZ SYNC AND COLOR BURST

EACH HORIZONTAL DIVISION IS EQUAL TO 3½ CYCLES OF 7M

THE PATTERN REPEATS EVERY 455 CYCLES OF 7M

SHADED AREA VOLTAGE DETERMINED BY THE DATA IN RAM



RELATIONSHIP BETWEEN VERTICAL SYNC, VERTICAL BLANK AND VERTICAL DRIVE
EACH HORIZONTAL DIVISION REPRESENTS ONE HORIZONTAL SCAN

1/14/77	N/C
1/27/77	A
3/25/77	B
7/6/77	C

ELECTRICAL SPECIFICATION FOR MIDWAY CUSTOM CIRCUITS

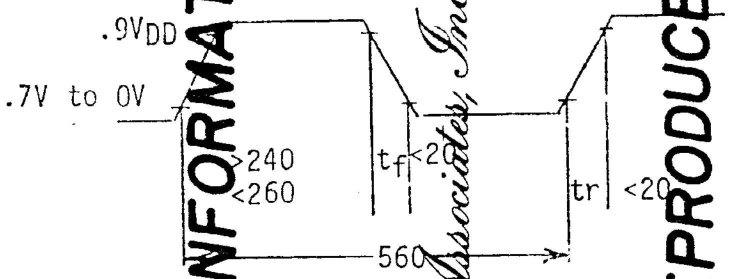
I. GENERAL SYSTEM PARAMETERS

I. A. Power Supplies

1. VDD=+5.0V ±5%
2. VGG=+10.0V ±5%
3. VSS=0.0V

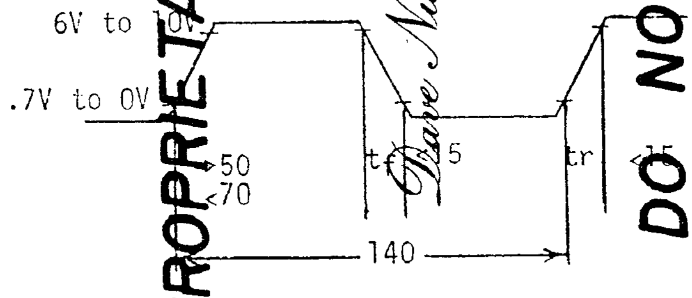
I. B. Timing Signals

1. Ø & $\overline{Ø}$: Period = 560nsec, High time* 240nsec to 260nsec.
   Ø and $\overline{Ø}$ have zero level crossover +1 volt -0 volts
   t_r, t_f* less than 20nsec



(Times are in nsec)

2. 7M & $\overline{7M}$; Period = 140nsec, High time+ 50nsec to 70nsec
   7M & $\overline{7M}$ have zero level crossover +1 volt -0 volt
   t_r, t_f+ less than 15nsec



(Times are in nsec)

Dead time < 5nsec
Max C Load = 20pf

+Note
1) High time is time clock at ≥.6V.
2) Rise time from zero level to one level.

I. B. (Continued)

*Note:

1. High time is time between 50% points.

2. Clock signals are generated by low power Shottky Logic (series 74LS). Full level swing on clock signals to be achieved through external resistor to V_{DD}. Zero level .7V to 0V.

3. Rise time from zero level to $.9V_{DD}$.

I. C. Z80 Data Bus (MUXDØ-MUXD7)

1. Z80 Data Bus interface requires a three-state output/input buffer. The three states are defined below.

2. Logic 0: .5V + noise generated by chip, noise for address chip is .15V @ -430μA

3. Logic 1: 2.7V @ +70μA

4. High Impedance: Leakage at either logic 0 or 1 to be less than 5μA.

5. Transient Response: Transition from High Impedance to 0 or 1 will be complete within 442nsec of the 90% point of $\overline{\emptyset}$ of the last wait state of input cycle or 442nsec of the 90% point of Ø of the second wait state of the interrupt acknowledge cycle. The maximum load will be 80pf. This includes 14pfd for two custom chips.

6. Exception: The path through the Data chip connecting the RAM bus with the Z80 bus shall introduce a maximum of 160nsec of delay.

7. The low address byte will be valid on the Z80 Data Bus at least 62nsec before $\overline{\emptyset}$. The high address byte will be valid at least 79nsec before $\overline{\emptyset}$. The data byte will be valid 55nsec before $\overline{\emptyset}$.

I. D. RAM Data Bus (MDO-MD7) - Home Game

1. The RAM Data Bus will require three state logic buffers.
2. Logic 0: .5V @ -25μA
3. Logic 1: 2.7V @ +25μA
4. High Impedance: 5μA maximum leakage at either logic 0 or 1.
5. Transient Response: The outputs shall transition from High Impedance to 0 or 1 within 120nsec of 7M. The outputs shall transition from 1 or 0 to high impedance within 20nsec of 7M. Maximum load will be 20pf.

I. E. RAM Data Bus (MDO-MD7) - Commercial Game

1. The RAM Data Bus will require three state logic buffers.
2. Logic 0: .5V @ -200μA
3. Logic 1: 2.7V @ +25μA
4. High Impedance: 5μA maximum leakage of either logic 0 or 1.
5. Transient Response: The output shall transition from High Impedance to 0 or 1 within 120nsec of 7M. The output shall transition from 1 or 0 to High Impedance within 2nsec of 7M. Maximum load will be 10pf.

I. F. Ambient operating temperature ≥ 0°C, ≤ [illegible]°C.

I. G. Storage temperature ≥ -65°C, ≤ 150°C.

I. H. Packing 40 pin plastic.

II. CUSTOM CIRCUIT SPECIFICATION

This specification defines the terminal characteristics for each of the custom circuits. These specifications shall take precedence in case of conflict. All Ø references refer to the Ø and $\overline{Ø}$ inputs to the address and I/O chip.

II. A. Data Chip

1. Input Pin List

Input Pin List	VO (V)	V1 (V)	td (Low)[1] (nsec)	td (High)[1] (nsec)	Ref.
$\overline{\text{MREQ}}$	.5	2.45	132	6	7M
$\overline{\text{RD}}$	.5	2.45	12	6	7M
$\overline{\text{IORQ}}$	.5	2.45	112	126	7M
7M	See Section I.B.				
$\overline{\text{7M}}$	"	"			
$\overline{\text{WRCTL}}$	.5	3.1	82	82	7M
$\overline{\text{M1}}$	.5	2.45	12	82	7M
LTCHDO	.5	3.1	120	120	7M
Serial 0	.5	2.45	30	30	7M
Serial 1	.5	2.45	30	30	7M

2. Power Supplies

See Section I. A.

3. Bus Connections

MXD0	See Z80 Data Bus Spec. Section	I.C.
MXD1	"	"
MXD2	"	"
MXD3	"	"
MXD4	"	"
MXD5	"	"
MXD6	"	"
MXD7	"	"
MD0	See RAM Data Bus Spec Section	I.D.
MD1	"	"
MD2	"	"
MD3	"	"
MD4	"	"
MD5	"	"
MD6	"	"
MD7	"	"

4. Outputs	VO (V)	IO (µA)	VI (V)	II (µA)	CAP (pf)	t_p (nsec)	Ref.
VIDEO*	*				10	100	7M
R-Y*	*				10	600	
B-Y*	*				10	600	
HORIZ DR	Note 4	400	2.7	20	20	20	7M
VERT DR	Note 4	400	2.7	20	20	20	7M
2.5V[6]	--	--	--	--	--	DC	
Ø	Note 4	400	2.7	20	10	100	7M
$\overline{\text{PXCLK}}$	Note 4	400	2.7	20	10	100	7M
MC0	Note 4	400	2.7	20	10	120	7M
MC1	Note 4	400	2.7	20	10	120	7M
$\overline{\text{DATEN}}$	Note 4	400	2.7	20	10	90	7M

*Video, R-Y, B-Y are analog outputs at 140nsec rate. Video, must switch from 10% to 90% of black to white in 140nsec. R-Y and B-Y transitions not to exceed .6µsec.

1 t_d (Low) and t_d (High) is maximum time in nsec except where a minimum is shown.
2 For $\overline{\text{IORQ}}$ Ref. to Ø t_d (Low)=132nsec t_d (High)=6nsec.
3 Serial 0 and Serial 1 will operate at 7MHz.
4 .5V + noise generated by chip.
5 Tap on both resistor chains for a capacitor. Will become test input with voltage applied > 8V.
6 The Z80 Ø is generated by this signal with a clock driver which introduces a delay of <20nsec.

II. B. I/O Chip

1. Input Pin List

	V0	V1	Ref	t_d (High) (nsec)	t_d (Low) (nsec)
Reset	.5	2.45			
MONOS	Note 1				
$\overline{\text{RD}}$	.5	2.45	Ø or $\overline{Ø}$	166	172 Ø or $\overline{Ø}$
$\overline{\text{IORQ}}$	.5	2.45	$Ø^6$	146 $\overline{Ø}$	132 Ø
Ø	See Section I.B.				
$\overline{Ø}$	"	"	"		
SIØ	.5	3.3			Note 3
SI1	.5	3.3			Note 3
SI2	.5	3.3			Note 3
SI3	.5	3.3			Note 3
SI4	.5	3.3			Note 3
SI5	.5	3.3			Note 3
SI6	.5	3.3			Note 3
SI7	.5	3.3			Note 3
TEST	.5	5.0			DC

2. Power Supplies

See Section I.A.

3. Bus Connections

MUXD0	See Z80 Data Bus Spec Section I.C.
MUXD1	" "
MUXD2	" "
MUXD3	" "
MUXD4	" "
MUXD5	" "
MUXD6	" "
MUXD7	" "

4. Outputs

		V0 (V)	I0 (µA)	V1 (V)	I1 (µA)
Audio	Note 4	Fmax - 20KHz			
Discharge	Note 5	.5V	4V		
SØØ	Note 3	Note 7	200	4V	1650
SØ1	Note 3	Note 7	200	4V	1650
SØ2	Note 3	Note 7	200	4V	1650
SØ3	Note 3	Note 7	200	4V	1650
SØ4	Note 3	Note 7	200	4V	1650
SØ5	Note 3	Note 7	200	4V	1650
SØ6	Note 3	Note 7	200	4V	1650
SØ7	Note 3	Note 7	200	4V	1650
POT Ø	Note 2		5	VDD-.5	50
POT 1	Note 2		5	VDD-.5	50
POT 2	Note 2		5	VDD-.5	50
POT 3	Note 2		5	VDD-.5	50

Note 1 MONOS triggers at 2.1 volts ±2% ± noise voltage when the supply is 5.25V.

Note 2 Open source-Voltage measured with 0.2ma.

Note 3 Time from load of address into microcycle register to date valid on MUX data bus from SI inputs (data path through address decoder, out on SO outputs, through closed switch and isolation diode, into SI input to MUX Data Bus) shall be 2μsec max. Drop of isolation diode will be 0.7V max. SO must drive 2kΩ in the high level. Max C load of SO shall be 300 pf. SI input shall have kill device enabled by $\overline{\text{INPUT}}$.

Note 4 Audio voltage oscillates between 0V and one of the following voltages; .33, .67, 1.00, 1.33, 1.67, 2.00, 2.33, 2.67, 3.00, 3.33, 3.67, 4.00, 4.33, 4.67 and 5.00. These voltages should be ±6%. The load shall be 1000pf and 100kΩ.

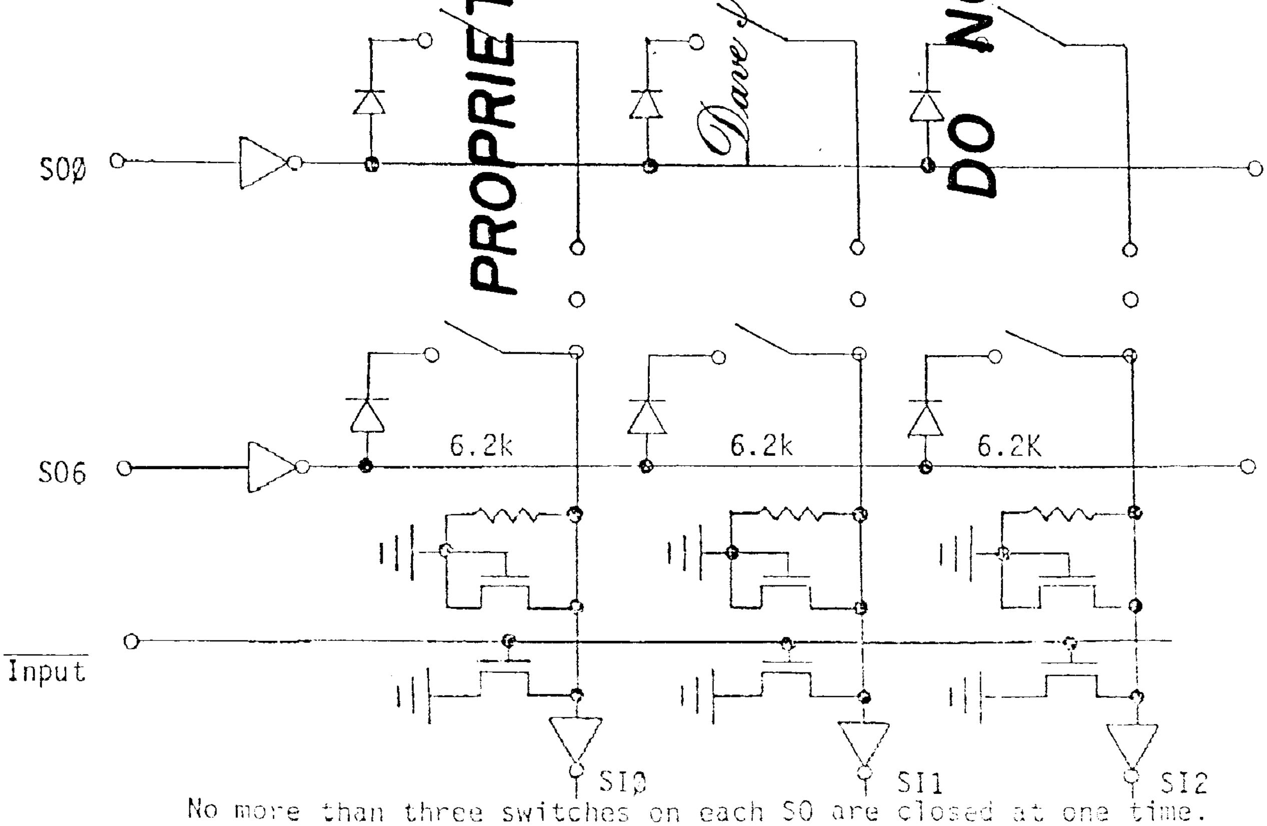
Note 5 Discharge is open drain to Vss. Discharges .01μfd capacitor to .2V in 144μsec.

Note 6 For $\overline{\text{IOREQ}}$ Ref. to $\bar{\text{Ø}}$ td (Low)=152nsec td (High)=166nsec.

Note 7 .5V + noise generated by I/O chip.

Miscellaneous Timing

Time for [illegible] Adder - 2Ø max



No more than three switches on each SO are closed at one time.

II. C. Address Chip

1. Input Pin List

	VO (V)	V1 (V)	tpd (Low) (nsec)	tpd (High) (nsec)	REF
$\overline{\text{RFSH}}$	.5	2.45	222 Ø	216	Ø
$\overline{\text{MREQ}}$	.5	2.45	152 $\overline{\text{Ø}}$	166	Ø or $\overline{\text{Ø}}$
$\overline{\text{RD}}$	.5	2.45	172 Ø or $\overline{\text{Ø}}$	166	Ø or $\overline{\text{Ø}}$
$\overline{\text{MI}}$	.5	2.45	176 Ø	242	Ø
A12[1]	.5	2.45			Ø
A13[1]	.5	2.45			Ø
A14[1]	.5	2.45			Ø
A15[1]	.5	2.45			Ø
$\overline{\text{IORQ}}$	.5	2.45	132 Ø	146	$\overline{\text{Ø}}$[2]
$\overline{\text{LIGHT PEN}}$	.5	2.45	Asyn		
$\overline{\text{TEST}}$	.5	5.0	DC		
HORIZ. DR.	.5	2.45	Note 3		$\overline{\text{Ø}}$
VERT. DR.	.5	2.45	Note 4		Ø
Ø	See Section I.B.				
$\overline{\text{Ø}}$	"	"			

2. Power Supplies

See Section I.A.

3. Bus Connections

MXD0	See Z80 Data Bus Spec Section I.E.
MXD1	" "
MXD2	" "
MXD3	" "
MXD4	" "
MXD5	" "
MXD6	" "
MXD7	" "

4. Outputs

	VO (V)	IO (μA)	V1 (V)	I1 (μA)	CAP (pf)	tpd(Low) (nsec)	tpd(High) (nsec)	REF
LATCHDO	Note 7	Note 6	3.1	Note 6	10	280	140	$\overline{\text{Ø}}$[5]
$\overline{\text{WAIT}}$	"	" 400	2.4	20	25	490	490	$\overline{\text{Ø}}$
MA0-MA5	"	" 400	2.4	20	20	242	240	$\overline{\text{Ø}}$ or Ø
$\overline{\text{INT}}$	"	" 400	2.4	20	25	490	572	$\overline{\text{Ø}}$
RAS0-RAS3	"	" 400	2.4	20	20	382	382	$\overline{\text{Ø}}$
$\overline{\text{WRCTL}}$	"	" Note 6	3.1	Note 6	10	382	382	$\overline{\text{Ø}}$

1. Time from High Impedance to 1 or 0 is 200nsec. (from $Ø_1$ of T_1)
2. For $\overline{\text{IORQ}}$ Ref to $\overline{\text{Ø}}$ td (Low)=152nsec td (High)=166nsec. Ø
3. Horizontal Drive time from low to high is 40nsec after $\overline{\text{Ø}}$.
   Time from high to low is 100nsec before rising edge of Ø.
4. Vertical Drive will transition from low to high 40nsec after falling edge of Ø. Its width will be 2.1 μsec max. 1.54μsec min. It will go from high to low 100nsec before falling edge of Ø.
5. Reference tpd (High) is Ø.
6. MOS to MOS signal.
7. .5V + noise generated by Address Chip (.15V) = .65V

III. I/O MODE DECODE

I/O Parts

HEX	Out	Input
0	Color Ø Right	
1	" 1 "	
2	" 2 "	
3	" 3 "	
4	" 0 Left	
5	" 1 "	
6	" 2 "	
7	" 3 "	
8	Consumer/Commercial	Intercept Feedback
9	Horiz Color Bndry	
A	Vertical Blank	
B	Color Block TX	
C	Magic Reg	
D	Interrupt Feedback	
E	Interrupt Mode	Vertical Addr Feedback
F	Interrupt Line.	Horizontal Addr Feedback
10	Tone Master OSC	SW Bank 0
11	Tone A	1
12	" B	2
13	" C	3
14	Tremello	4
15	Tone C Volume	5
16	Tone A,B Volume	6
17	Noise Volume	7
18	Sound Block TX	
19		
1A		
1B		
1C		POT 0
1D		" 1
1E		" 2
1F		" 3
20		
21		
22		
23		
24		
•		
•		
2F		