Home Video Game Manual for the Bally Astrocade

Dave Nutting Associates

Text Release 2.1 - February 8, 2002

This document, officially called the 'Home Video Game System' manual, is better known as the 'Nutting' or 'DNA' manual.

Conventions:

1) To match the original manual as closely as possible (for table of contents reasons), two blanks lines separate each 'page.' This is followed by the page number and then one more blank line.
2) The Page Number appears at the top of each page, not the bottom (don't get confused); it takes up the first line.
3) Every instance of the word 'cassette' has been replaced by the word 'cartridge' to avoid any confusion with the BASIC Cassette Interface (which isn't mentioned in this manual at all).
4) Special Character Representations:

   a. $\overline{\text{IORQ}}$, $\overline{\text{MREQ}}$ = IORQ#, MREQ#
   b. Subscripts = '_' (underscore) - contextual
   c. Superscripts = '^' (caret) - contextual
   d. The Greek symbol Phi ('O' over-striked with an 'I') is replaced with the word 'Phi'

From the original manual:

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 five days of written demand.

(c) 1978 Dave Nutting Associates, Incorporated
(c) 1978 Bally Manufacturing Corporation

This page intentionally left blank for double-sided print purposes

TABLE OF CONTENTS - SOFTWARE

TABLE OF CONTENTS - HARDWARE

LIST OF ILLUSTRATIONS

This page intentionally left blank for double-sided print purposes

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 - Allows cartridges 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 keyboard 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 cartridge program to access the facilities provided by the home video game system. By adhering to these conventions a cartridge program will be system independent, thus allowing improvements to be made to later versions of the system and on board games, while maintaining upward compatibility.

The basic rule for using the UPI is:

> With exception to the system DOPE vector, no cartridge 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 by 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 = 0:

```
        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 in 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    0             ;FILLED WITH ZEROES
        DO      CHRDIS        ;DO CHARACTER DISPLAY ROUTINE
        DEFB    0             ;Y-AXIS POSITION
        DEFB    10            ;X-AXIS POSITION
        DEFB    8             ;OPTIONS-PLOP,10-ON,00-OFF
        DEFB    'A'           ;CHARACTER TO BE DISPLAYED
        EXIT                  ;EXIT INTERPRETER
```

A block of call indexes have been set aside for the internal use of cartridge 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 its 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 give the routines starting address indexed via call number. The systems table is called 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

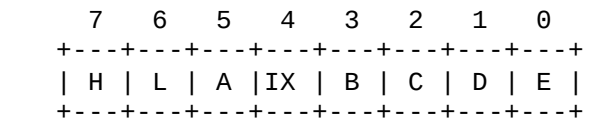
IN LINE ARGUMENT MASK ENTRY

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.



```
UPI INTPC
BEGIN INTERPRETING

Calling Sequence:       SYSTEM  INTPC
Arguments:              NONE
Notes:                  NONE
Description:

See UPI description for explanation of interpreter.
```

UPI XINTC
EXIT INTERPETER

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 to by IY.

Restrictions:
Assembler routines 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 in 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    0FFFH
                        DEFB    0
                        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 alpha-numerics.

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    10101010B
```

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.

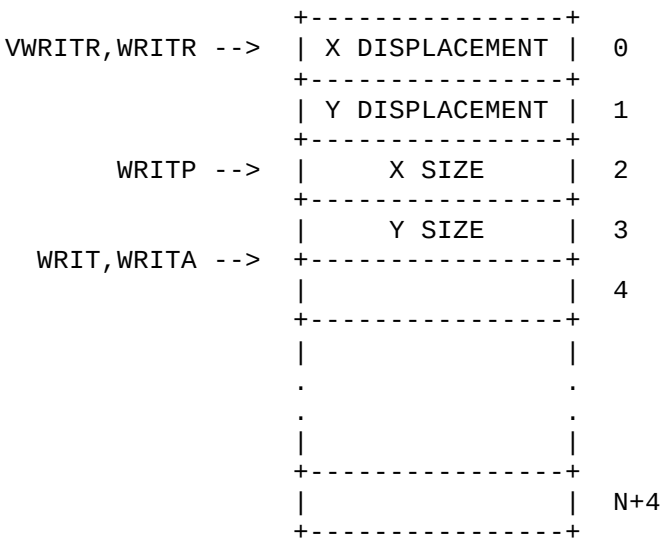
Note that ROTATE is not currently supported in software due to space considerations.

```
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 (0 - 101)
                E  = X Co-ordinate (0 - 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 were 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 20H 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 80H 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 1101 specifies that the foreground color is 11, and the background color is 01.

OPTION BYTE

ENLARGE FACTOR	XOR WRITE	OR WRITE	ON COLOR	OFF COLOR

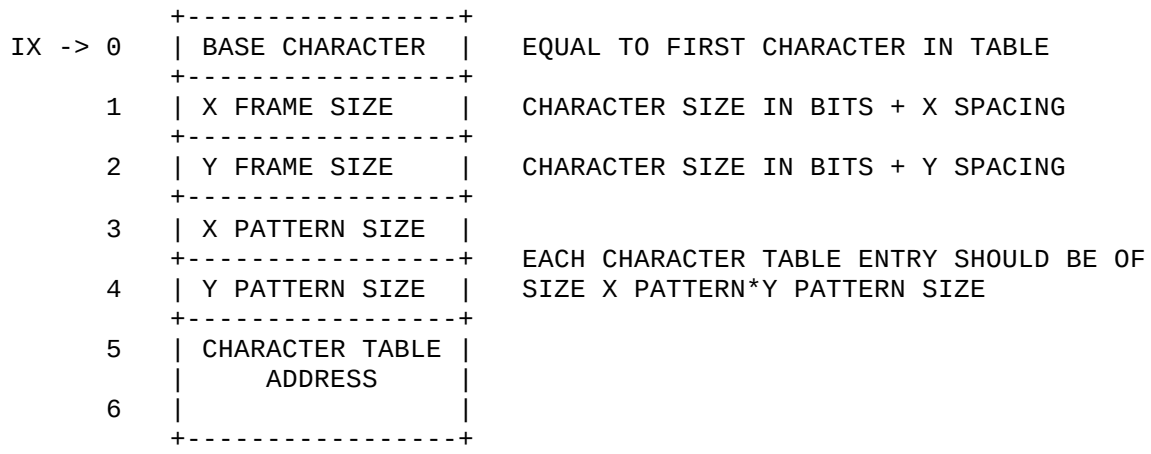
ENLARGE FACTOR	HOW MANY TIMES LARGER	ENLARGED SIZE OF SINGLE PIXEL
00	1	1 X 1
01	2	2 X 2
10	4	4 X 4
11	8	8 X 8

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 reference 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.



```
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


Description:
This routine displays the standard BCD codes 0 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 B0H).
```

```
SCREEN ALPHANUMERIC   DISTIM
DISPLAY TIME

Calling Sequence:       SYSTEM  DISTIM
                             or
                        SYSSUK  DISNUM
                        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, GTSECS) 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=0 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 character display primitive.  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 descriptor address.

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

```
SCREEN ALPHANUMERIC   STRDIS
DISPLAY STRING

Calling Sequence:       SYSTEM  STRDIS
                             or
                        SYSSUK  STRDIS
                        DEFB    (X co-ordinate)
                        DEFB    (Y co-ordinate)
                        DEFB    (options)
                        DEFW    (String)
Arguments:              HL = String address
                        C  = Standard options byte
                        DE = Standard Co-ordinates
*NOT LOADED             IX = Alternate Font descriptor dddress
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 character 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
70H	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 DELTA X and DELTA Y, which are added to X and Y every time increment. Since the screen is finite, there also exists two upper and 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	DELTA X	VBDXL	
4		VBDXH	
5	X	VBXL	
6		VBXH	
7	X CHECKS MASK	VBXCHK	
8	DELTA Y	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	BLANK	NOT					
VBSACT	VBBLNK	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	LIMIT ATTAINED	NOT	REVERSE DELTA SIGN	LIMIT CHECK
	USED	VBCLAT	USED	VBCREV	VBCLMT

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

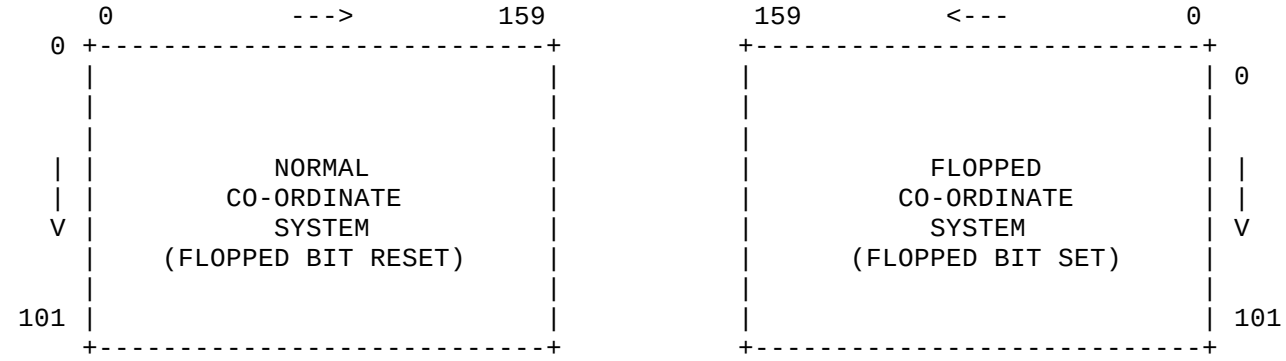
REVERSE DELTA — Set by user to indicate that when this co-ordinate attains it's limit, the sign of the associated delta is to be reversed. This can be used to cause objects to 'bounce' off barriers.

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 bitstring.  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.
```




```
Proofing Note: 160/102 = 1.57 - Actual Screen Aspect Ratio
```

```
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  i.e.: 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.

[Proofing Note: BLAKOUT is seven letters (limit is six).  I can not find
                anything close to this in the manual.  Thoughts?  Dec 16, 2001]
```

```
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 a 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

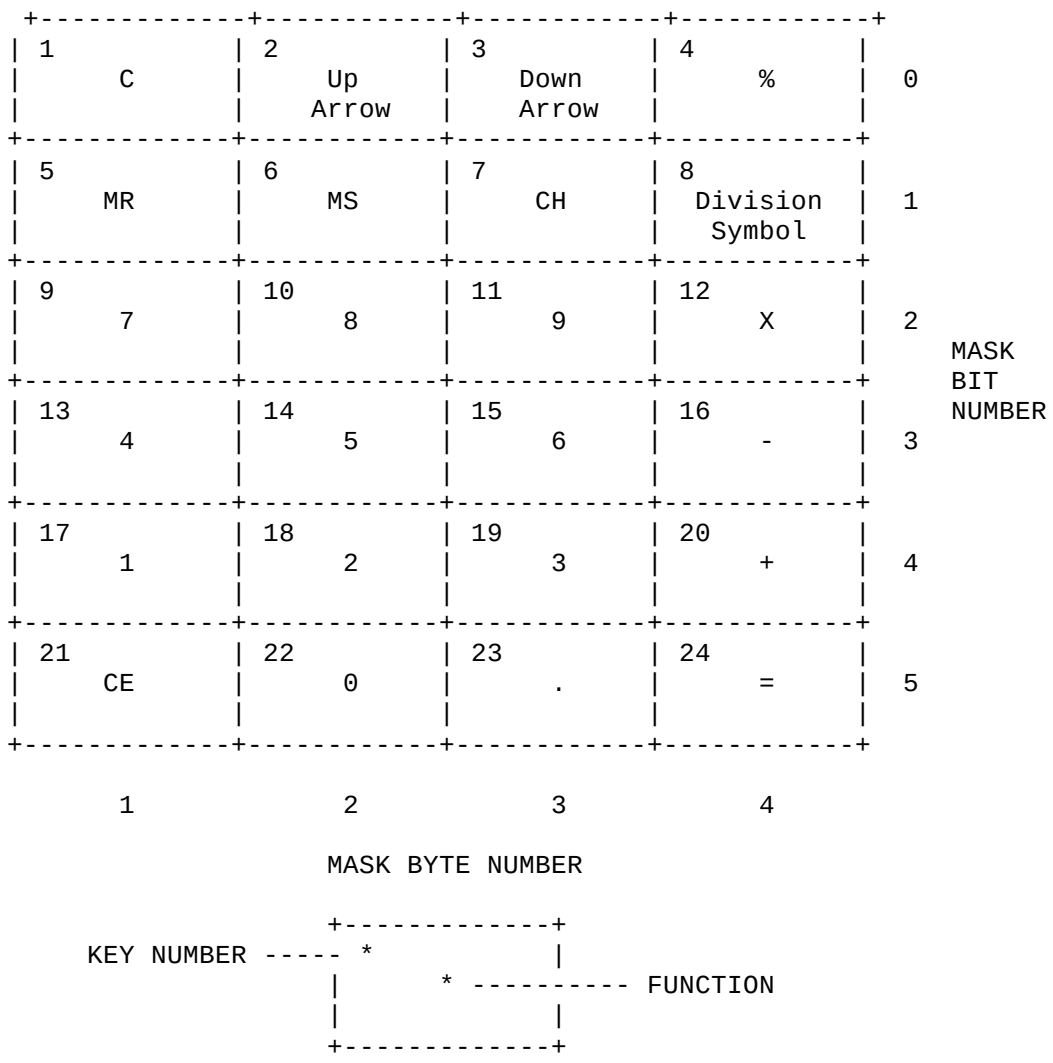
Description: This routine does a table look-up

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	0	0	30
23	Decimal Point	.	2E
24	Equals	=	3D

* - Three names ('Up Arrow,' 'Down Arrow,' and 'Divide') do not have ASCII equivalent graphic marks. An asterisk is NOT printed on screen. Instead, the BPA uses three different non-ASCII symbols. Each graphic looks as the name describes.

```
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, semaphores 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 0 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    011100B
just 0-9                DEFB    111100B
                        DEFB    011100B
                        DEFB    000000B
```

See diagram on following page.


1
C
2
Up Arrow
3
Down Arrow
4
%
0
5
MR
6
MS
7
CH
8
Division Symbol
1
9
7
10
8
11
9
12
X
2
13
4
14
5
15
6
16
-
3
17
1
18
2
19
3
20
+
4
21
CE
22
0
23
.
24
=
5
MASK BIT NUMBER
1
2
3
4
MASK BYTE NUMBER
KEY NUMBER
FUNCTION

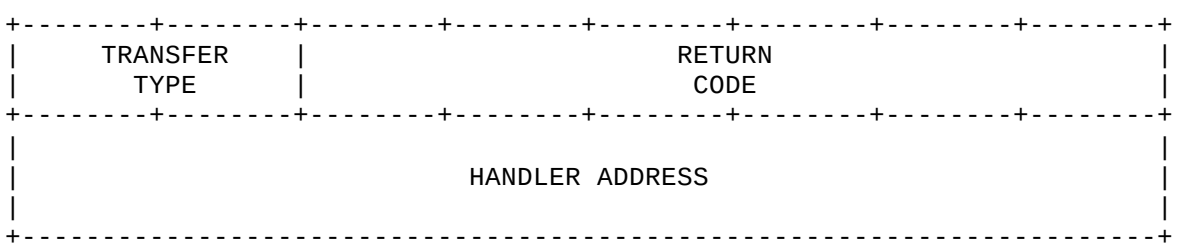
```
Output:               A = Return Code
                      B = Extended Code

PRIORITY        A=              MEANING
--------        --              -------

                SNUL            Nothing Changed
1               SCT0            Counter/Timer 0 decremented by 0
                 to
1               SCT7            Counter/Timer 7 decremented to 0
2               SF0             SEMI4S bit 0 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=0
5               SKYD            Key is down     B=key number
3               SP0             POT 0 changed   B=new value
                 to
3               SP3             Pot 3 changed   B=new value
6               SJ0             Joystick 0 changed    B=new value
                 to
6               SJ3             Joystick 3 changed    B=new value
6               ST0             Trigger 0 changed     B=new value
                 to
6               ST3             Trigger 3 changed    B = new value
```

Notes:

The potentiometers (pots) are debounced. New trigger value=Trigger off (0) or trigger on (10H). When switches are actuated simultaneously the order of return is: SCT7 TO SCT0, SF7 TO SF0, SP0 TO SP3, SSEC, SKYU, SKYD, SJ0, ST0, 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: 00=JMP to machine language routine and pop context; 01=RCALL machine language routine in current context; 10=MCALL interpreter routine in current context. Mode 01 and 10 expect the returned-to point to be interpretive, mode 0 expects it to be machine instructions.
End of list is indicated by a terminator byte which is greater than or equal to C0H.

```
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 0 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    011100B         ;NUMBER KEYS ONLY
                DEFB    111100B
                DEFB    011100B
                DEFB    0

        DTAB:   MC      SKYD,SHOW       ;IN KEY DOWN MACRO CALL
                MC      ST0,PBREAK+END  ;ON TO MACRO CALL

        SHOW:   DO      KCTASC          ;CONVERT TO ASCII
                DO      SUCK
                DEFB    00000111B       ;X,Y=0=DE
                DEFB    11001100B       ;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 address in the stack.
Duration:	Is loaded at the start of a note and then decremented every 60th of a second
Voice:	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)        ;TONE0=(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 60ths of a second
                                     ;Pauses silently (except legato)
    1           QUIET                ;Stops music and sets volume=0
    2           OUTPUT               ;Port #, Data
    9           OUTPUT               ;SNDBX,DATA10,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 0, 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  11010100B       ;ABC=DATA 1
                MASTER  0A1H            ;ABC=1/2
                VOLUME  88H,08H
                NOTE1   12,A1
                NOTE1   12,C2
                NOTE1   24,E2
                NOTE1   12,C2
                NOTE1   12,E2
                REST    6
                VOICES  11110110B       ;Suck in Vibrato, AB and C bytes
                NOTE3   12,14,A2,E2
                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 0 to volume ports and halts music processor.
```

```
INTERRUPTS   ACTINT
ACTIVE INTERUPTS

Calling Sequence:       SYSTEM  ACTINT
                               or
                        SYSSUK  ACTINT
Input:                  NONE
Output:                 NONE
Function:               Sets IM=2, INLIN=200, 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 CT0, CT1, CT2, CT3.  Functions
as 60th 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 counter if they are non-zero.  If any go from 1 to 0,
sentry is notified.
```

```
INTERRUPTS   TIMERS   CTIMER

Calling Sequence:       CALL    CTIMER
Input:                  HL = Address of custom time base
                        B  = Value to load into time base 1 to 0 transition
                        C  = CT mask as in DECCTS

Description:
HL is loaded and decremented.  If it is not = 0, 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 0,
                        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 Sequence:       SYSTEM  INDEXN
                                or
                        SYSSUK  INDEXN
                        DEFW    (Base Address)
Arguments:              C  = Nibble displacement (0 - 255)
                        HL = Base address of table
Output:                 A  = Nibble value

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

                        +-------+-------+
          BASE ADDRESS  |   1   |   0   |
                        +-------+-------+
                     1  |   3   |   2   |
                        +-------+-------+
                     2  |   5   |   4   |
                        +-------+-------+
                     3  |   7   |   6   |
                        +-------+-------+
                     .  |       |       |
                     .  |       |       |
```

```
STOREN   STORE NIBBLE

Calling Sequence:       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 (0 - 255)     *NOT LOADED
                        HL = Base address of table
Output:                 DE = Entry looked up
                        HL = Address of entry looked up
Description:            Indexing looks like:

                                              DISPLACEMENT
                        +---------------+
          BASE ADDRESS  |               |       0
                        +---------------+
                     1  |               |
                        +---------------+
                     2  |               |       1
                        +---------------+
                     3  |               |
                        +---------------+
                     4  |               |       2
                        +---------------+
                     5  |               |
                        +---------------+
                     .  |               |       .
                     .  |               |
```

```
INDEXB   INDEX BYTE

Calling Sequence:       SYSTEM  INDEXB
                             or
                        SYSSUK  INDEXB
                        DEFW    (Base Address)
Arguments:              A  = Displacement (0 - 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.
```

CARTRIDGE CONVENTIONS

Two types of cartridges may be used with the Bally Professional Arcade. The first type, called an autostart cartridge, 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 cartridge, is entered after a game selection process is completed. Considerably more initialization is done by the system before control transfer.

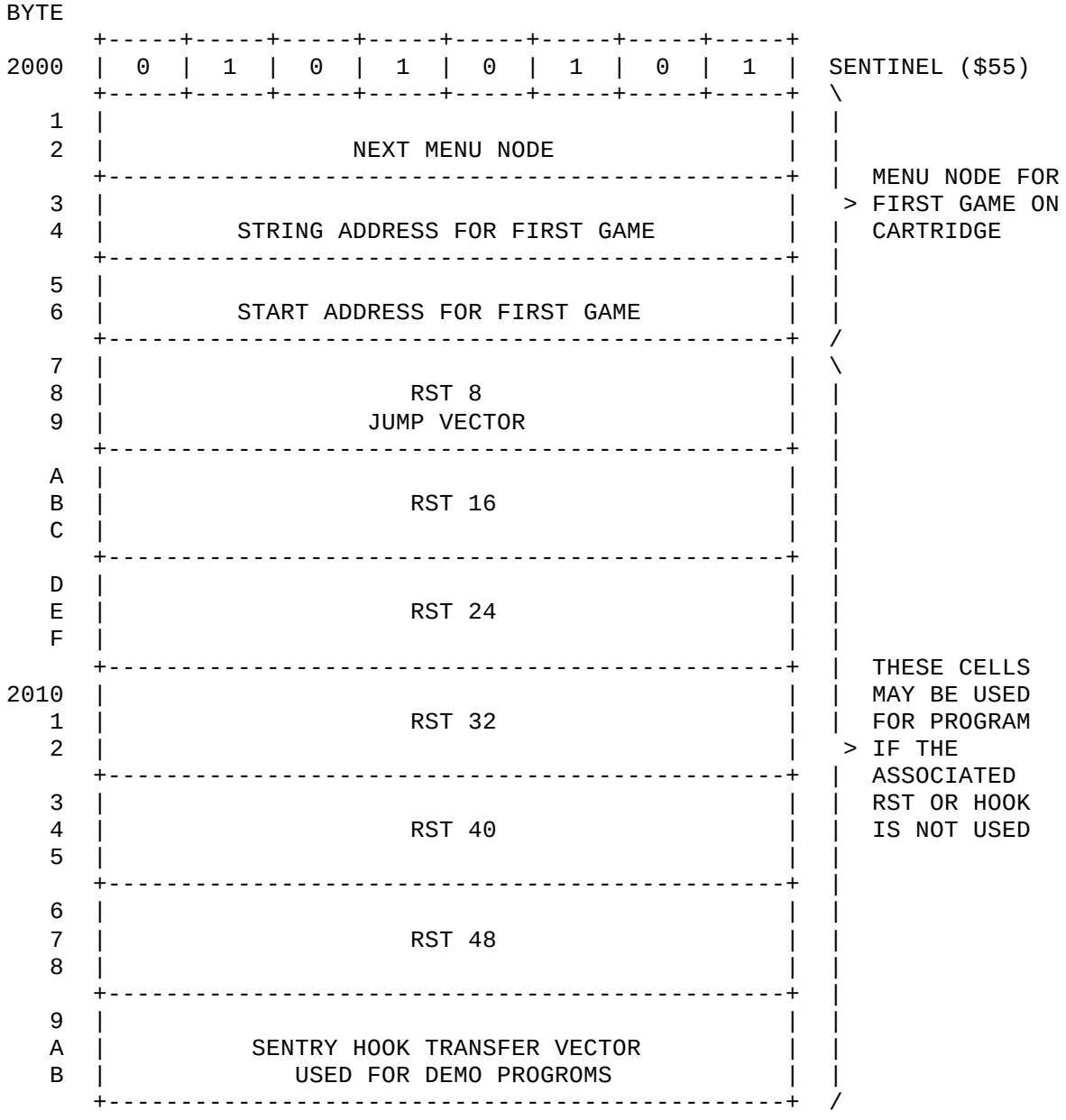
1) System RAM is cleared to 0
2) The ACTINT interrupt routine is enabled
3) The MENU colors are set in the left color map
4) Vertical blank is set at line 96, horizontal boundary at 41, and interrupt mode at 8.
5) The screen displays the menu frame.
6) The shifter is cleared.

An autostart cartridge is indicated by a jump instruction (opcode C3H) at location 2000H. This jump instruction should branch to the starting address of the cartridge.

A standard cartridge is indicated by a sentinel byte of 55H at location 2000H. Following this byte is the first node of the cartridge's menu data structure. This data structure gives the name and starting address of each program in the cartridge. (See MENU)

When the user has selected a cartridge game, control is transferred to the starting address with the address of the program name string in the registers. The cartridge 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 cartridge has access to the six unused restart instructions. See the following cartridge diagram for the transfer vectors.


BYTE
2000
SENTINEL ($55)
NEXT MENU NODE
MENU NODE FOR
FIRST GAME ON
CARTRIDGE
STRING ADDRESS FOR FIRST GAME
START ADDRESS FOR FIRST GAME
RST 8
JUMP VECTOR
RST 16
RST 24
THESE CELLS
MAY BE USED
FOR PROGRAM
IF THE
ASSOCIATED
RST OR HOOK
IS NOT USED
2010
RST 32
RST 40
RST 48
SENTRY HOOK TRANSFER VECTOR
USED FOR DEMO PROGROMS

```
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 1/4 second to allow user to see his entry and then returns.

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

```
HUMAN   MENU
DISPLAY MENU AND BRANCH ON SELECTION

Calling Sequence:       SYSTEM  MENU
                             or
                        SYSSUK  MENU
                        DEFW    (Title)
                        DEFW    (List)
Arguments:              DE = Address of menu title string
                        HL = Address of menu list
Output:                 DE = String address of selection mode

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

Notes:
A maximum of eight entries may appear.
On entry, MENU expects interrupts to be enabled, colors and boundaries
to be set up.  MENU uses 96 lines of screen, 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
             +----------+

Proofing Note: 'creams the alternate set' = 'creates the alternate set?'
```

```
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 the 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, 0 is returned.

Note:                   B is not sucked [by SYSSUK].
```

```
MATH   RANGED
RANGED RANDOM NUMBER

Calling Sequence:       SYSTEM  RANGED
                             or
                        SYSSUK  RANGED
                        DEFB    (N)
Arguments:              A = N where 0 is less than or equal to a random
                            number less than N
                            (ie: for a random number of 0,1,or 2, N=3)
Output:                 A = Random Number

Notes:
If N is a power of 2, it is considerably faster to use N=0 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 cartridge connector. Three custom chips are used for the video interface, special video processing functions, keyboard and control handle interface, and audio generation.

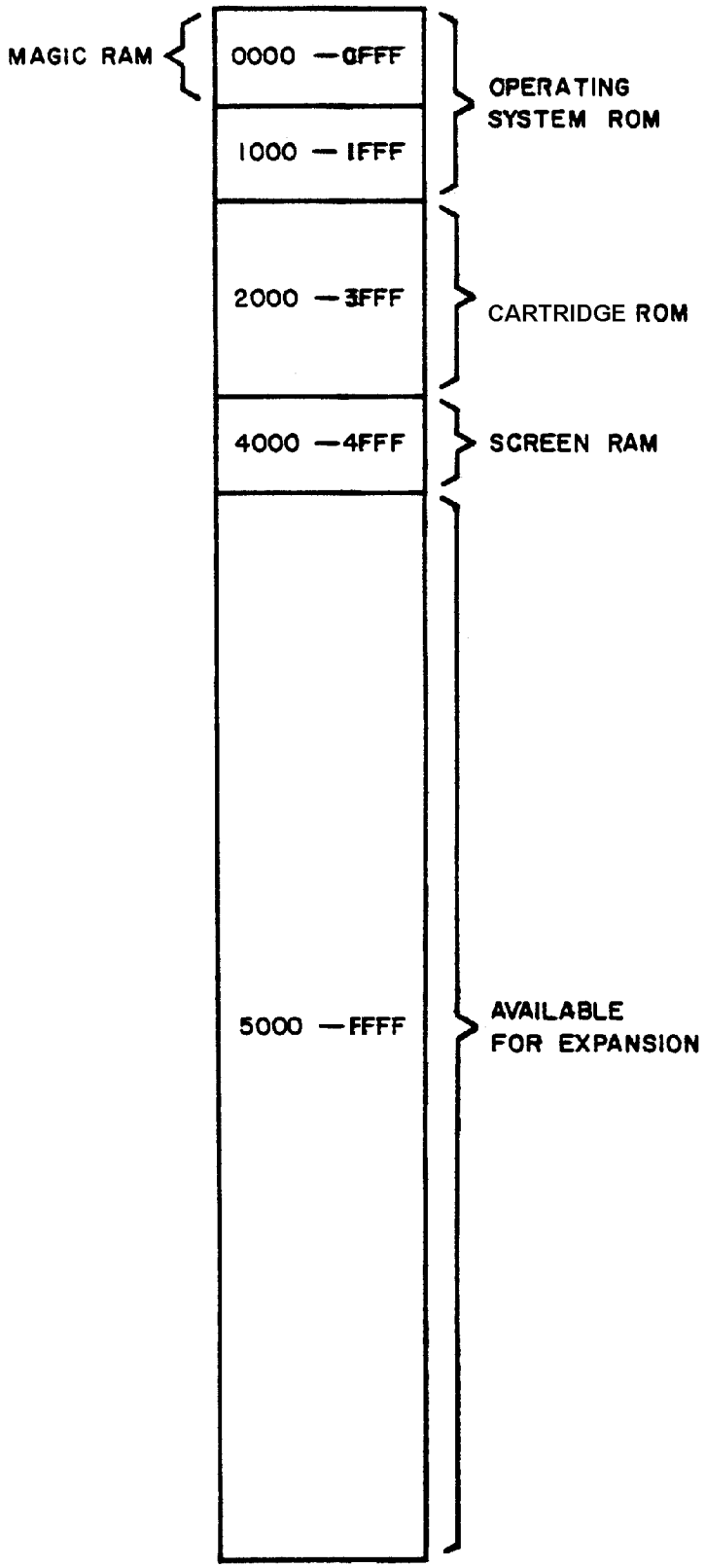
The system exists in both high-resolution and low-resolution models. The three custom chips can operate in either mode. The mode of operation is determined by bit 0 of output port 8H. It must be set to 0 for low-resolution and 1 for high-resolution. This bit is not set to 0 at power up and must be set by software before any RAM operations can be performed.

Memory Map

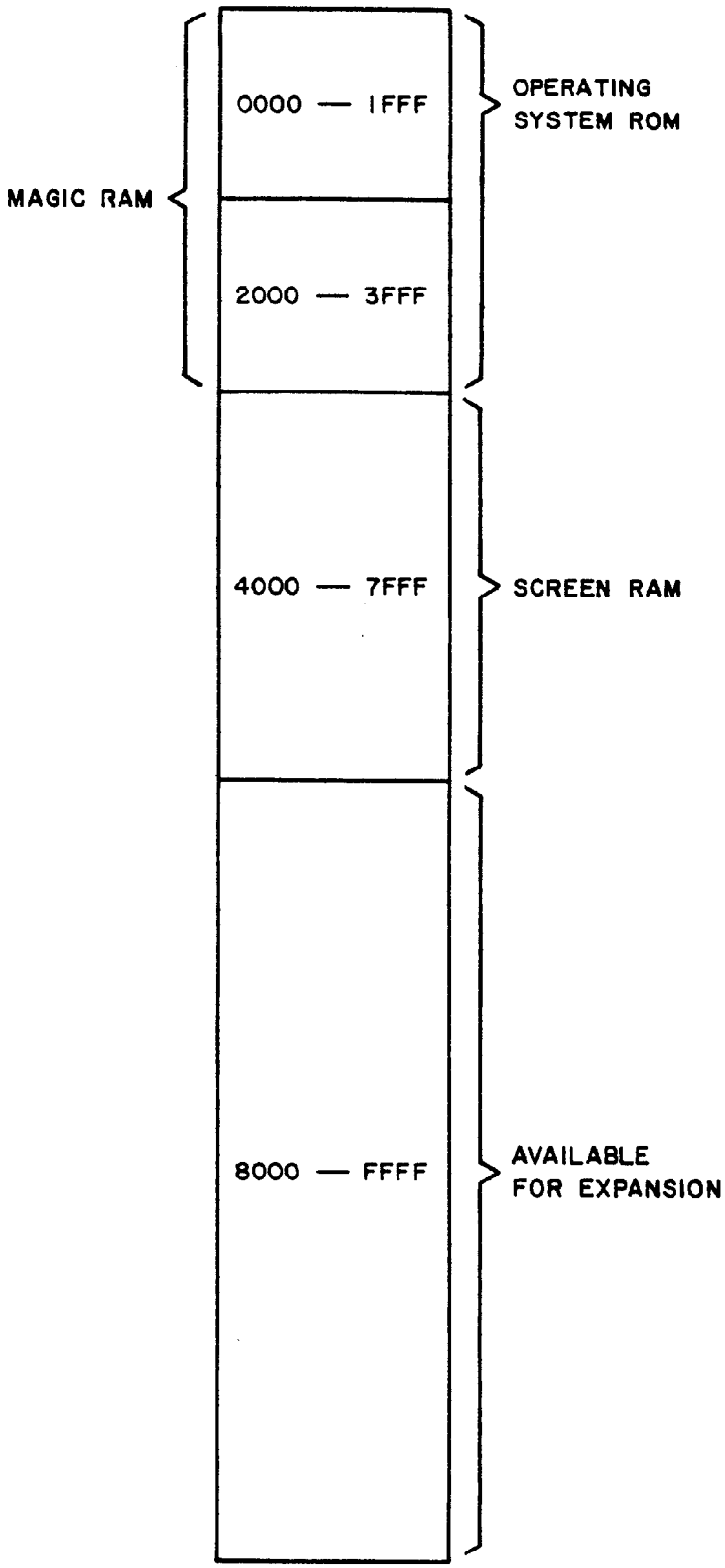
In both the low and high resolution models, the operating system ROM is in the first 8K of memory space. The cartridge ROM is in the space from 8K to 16K. The standard screen RAM begins at 16K. In the low-resolution unit, standard screen RAM is 4K bytes; in the high-resolution unit it is 16K bytes. Magic screen RAM begins at location 0. 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 0 and 16K the data comes from the ROM. When data is written in a memory location (X) between 0 and 16K, the system actually writes a modified form 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

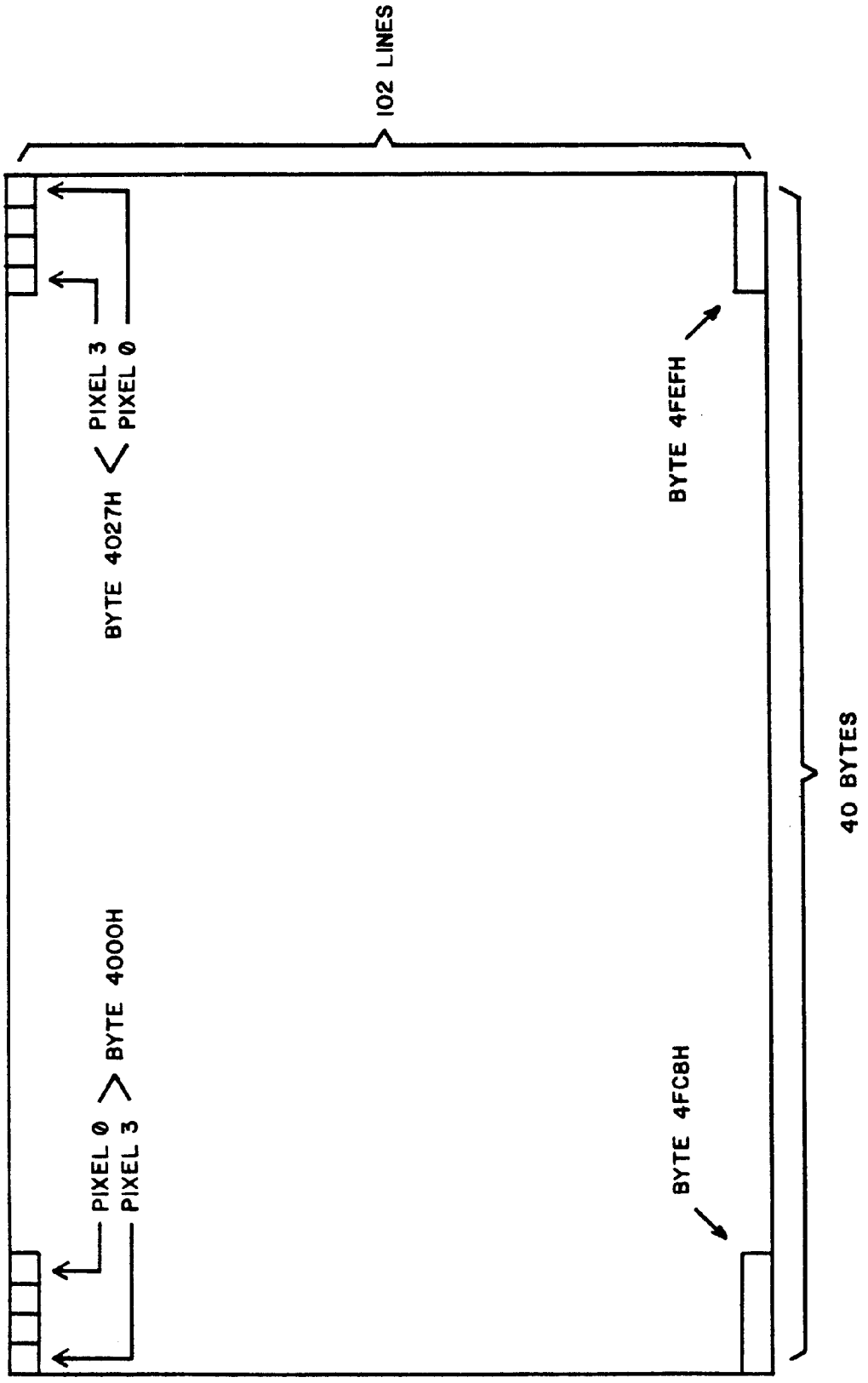
In the Bally Professional Arcade, two bits of RAM are used to define a pixel on the screen. One 8-bit byte of RAM therefor defines four pixels on the screen.

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 may be used for scratchpad by blanking the screen before the 102nd line. This will be described later.

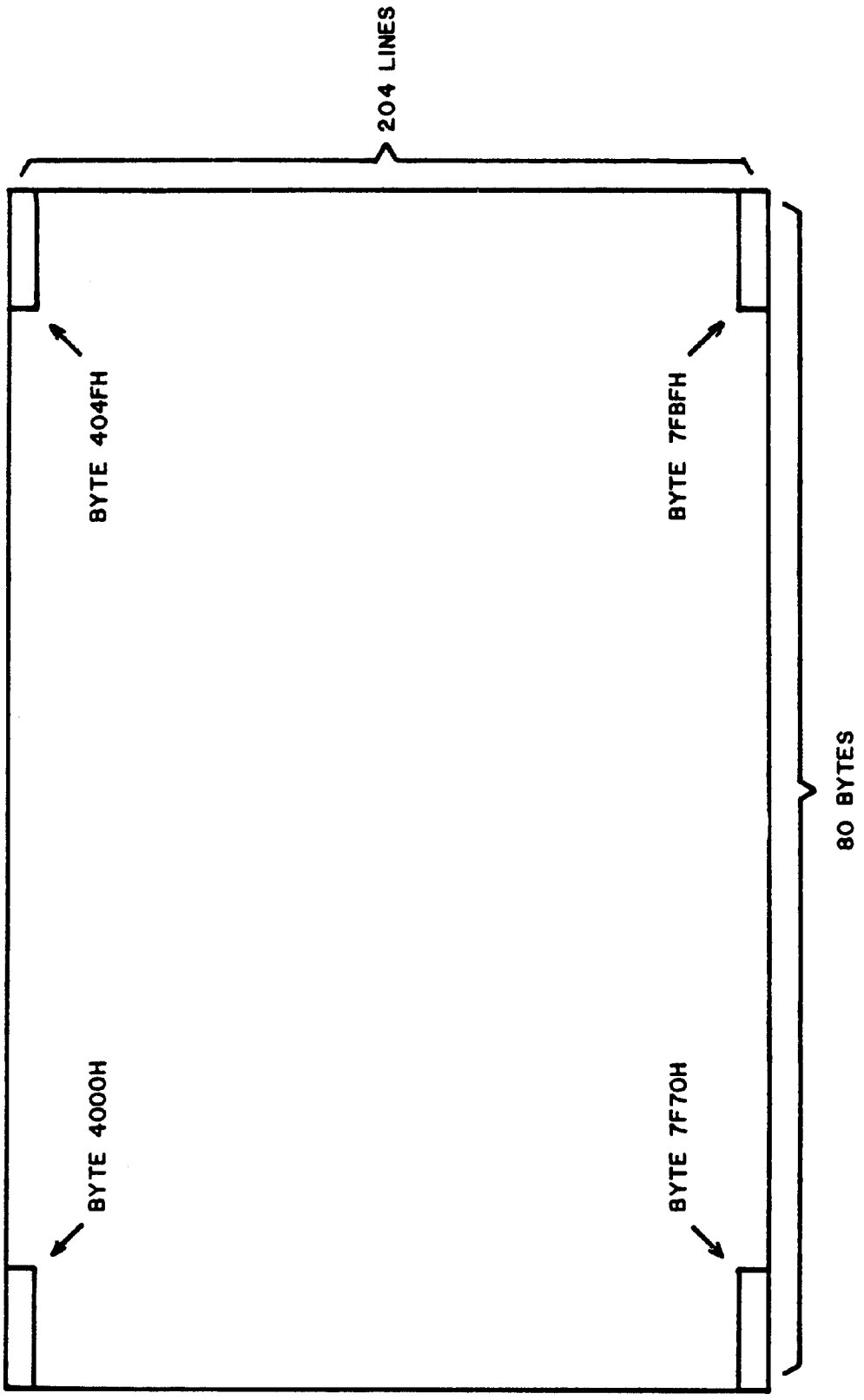
In the High-resolution model there are 80 bytes and 320 pixels per line. The 204 lines require 16,320 bytes of RAM. 64 bytes of the 16K RAM are left for scratch pad.

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 0 and 1, on the right.

Screen Map - Low Resolution



Screen Map - High Resolution



Color Mapping

Two bits are used to represent each pixel on the screen. These two bits, along with the LEFT/RIGHT bit which is set by crossing the horizontal color boundary, map each pixel to one of eight different color registers. The value in the color register then defines the color and intensity of the pixel on the screen. The intensity of the pixel is defined by the three least significant bits of the register, 000 for darkest and 111 for lightest. The color is defined by the five most significant bits. The color registers are at output ports 0 through 7; register 0 at port 0, register 1 at port 1, etc.

The color registers can be accessed as individual ports or all eight can be accessed by the OTIR instruction. The OTIR instruction is to port BH (register 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 0
```

The horizontal color boundary (bits 0-5 of port 9) defines the horizontal position of an imaginary vertical line on the screen. The boundary line can be positioned 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 0-5 of port 9. For example, if the horizontal color boundary is set to 0, the line will be just to the left of byte 0; 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 0 for pixels to the right of the boundary. Color registers 0-3 are used for pixels to the right of the boundary and registers 4-7 are used for pixels to the left of the boundary.

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

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

BACKGROUND COLOR

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

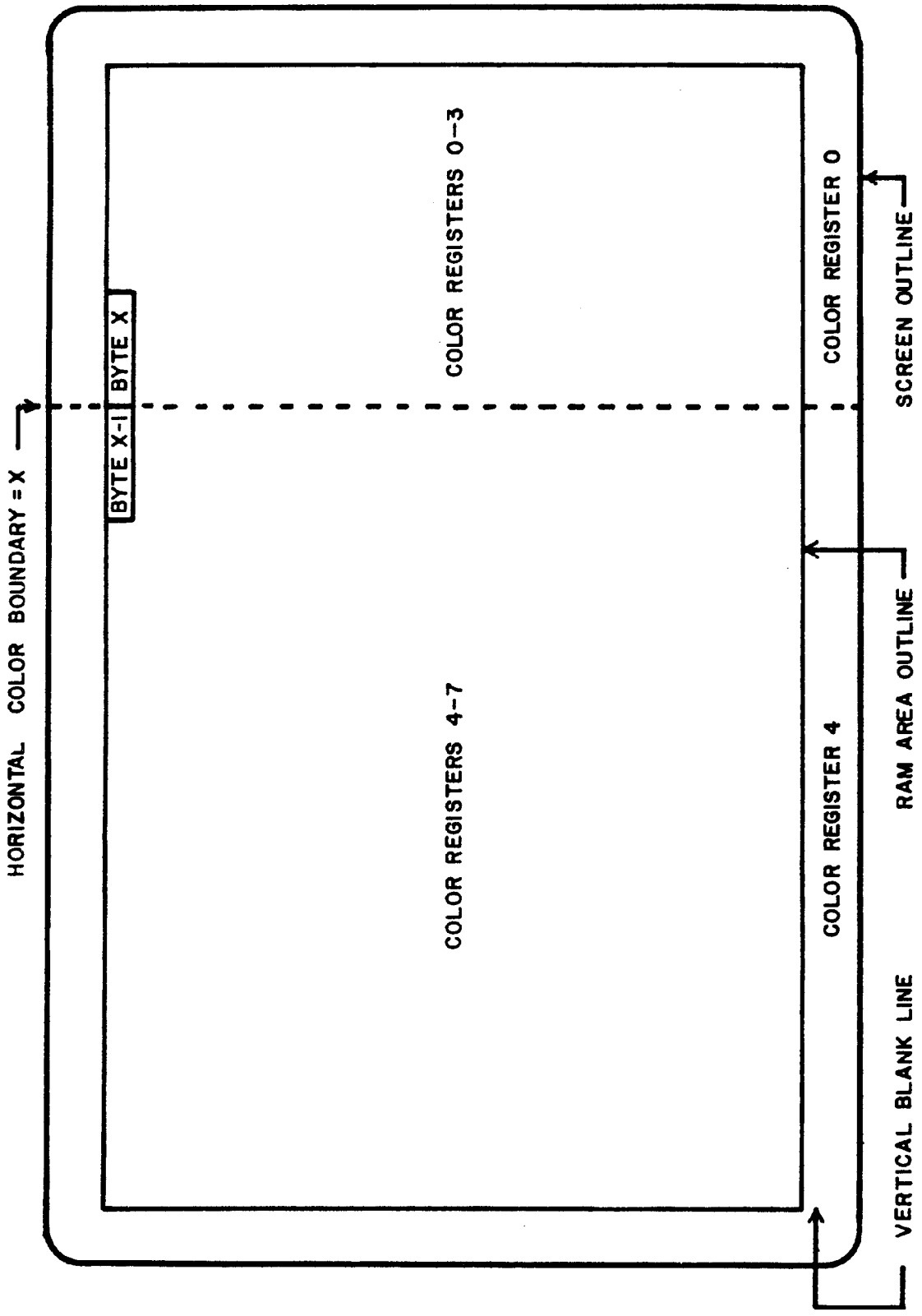
VERTICAL BLANK

The Vertical Blank Register (output port AH) contains the line number on which vertical blanking will begin. In the low-resolution system bit 0 should be set to 0 and the line number should be in bits 1-7. In the high-resolution system the line number is in bits 0-7. 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 0 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 203 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 0. If it were set to 1 then color registers 1 and 5 would be used for background instead of 0 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 0 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 0 the Z-80 will continue to be interrupted until it finally enables interrupts and acknowledges the interrupt. In mode 1 the interrupt will be discarded if it is not acknowledged by the next instruction after it occurred. 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 the light pen interrupt is bit 0 of port EH and the mode bit for screen interrupt is bit 2 of 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 0 is set to 0 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 0-7. In the low resolution system the line number is in bits 1-7, bit 0 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, 0 to 159. In the high-resolution system the resultant must be multiplied by two to give the pixel number, 0 to 358.

MAGIC REGISTER

As described earlier, the Magic System is enabled when data is written to a memory location (X) from 0 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	0	LSB of shift amount
	1	MSB of shift amount
	2	Rotate
	3	Expand
	4	OR
	5	XOR
	6	Flop

The order in 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 0 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 0, 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 0 on the data bus will be expanded into the pixel defined by bits 0 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 0 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 rotater 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 P0, P1, P2, and P3. P0 is composed of the first two bits of the byte.

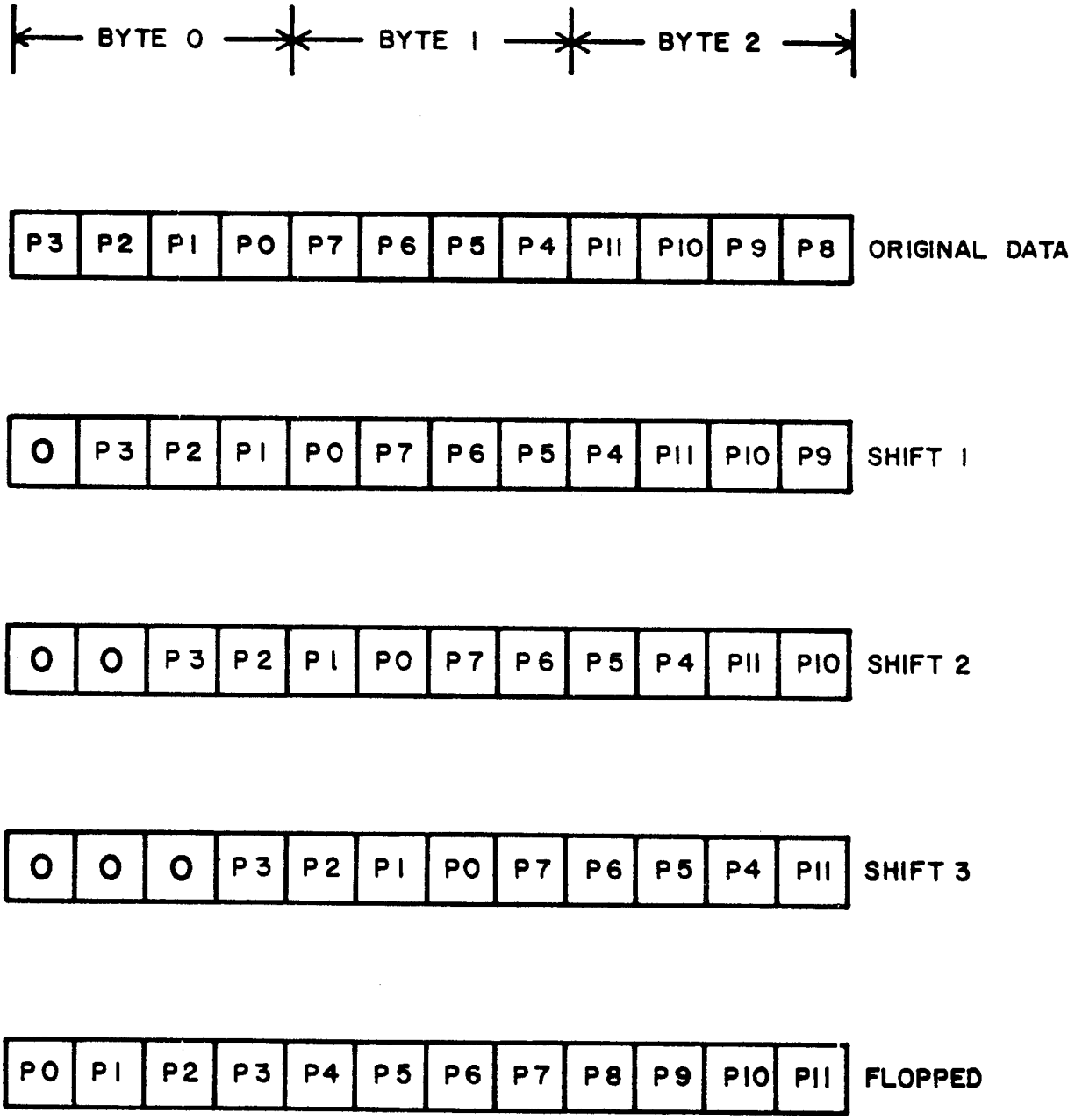
The shifter shifts data 0, 1, 2, or 3 pixels to the right. The shift amount is determined by bits 0 and 1 of the magic register. The pixels that are shifted out of one byte are shifted into the next byte. Zero's are shifted into the first byte of a sequence. The shifter assumes the first byte of a sequence 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 0 and 3 exchange values as do pixel 1 and 2.

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

SHIFTER - FLOPPER



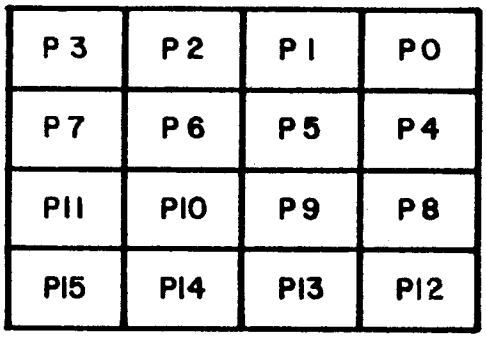
ROTATOR

The rotater is used to rotate a 4 X 4 pixel image 90 degrees in a clock-wise 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 degrees from the original image.

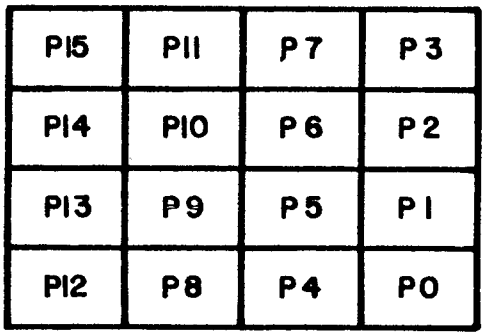
The rotator can only be used in commercial mode.

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

ROTATOR



ORIGINAL



ROTATED

OR AND XOR

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

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

INTERCEPT

Software reads the intercept register (input port 8H) to determine if an intercept occurred on an OR or XOR write. An intercept is defined as the writing of a non-zero pixel in a pixel location that previously contained a non-zero pixel. A non-zero pixel is a pixel with a value of 01, 10, or 11. A 1 in the intercept register means an intercept has occurred. Bits 0 - 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	
0	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 0 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 0 in last OR or XOR write

PLAYER INPUT

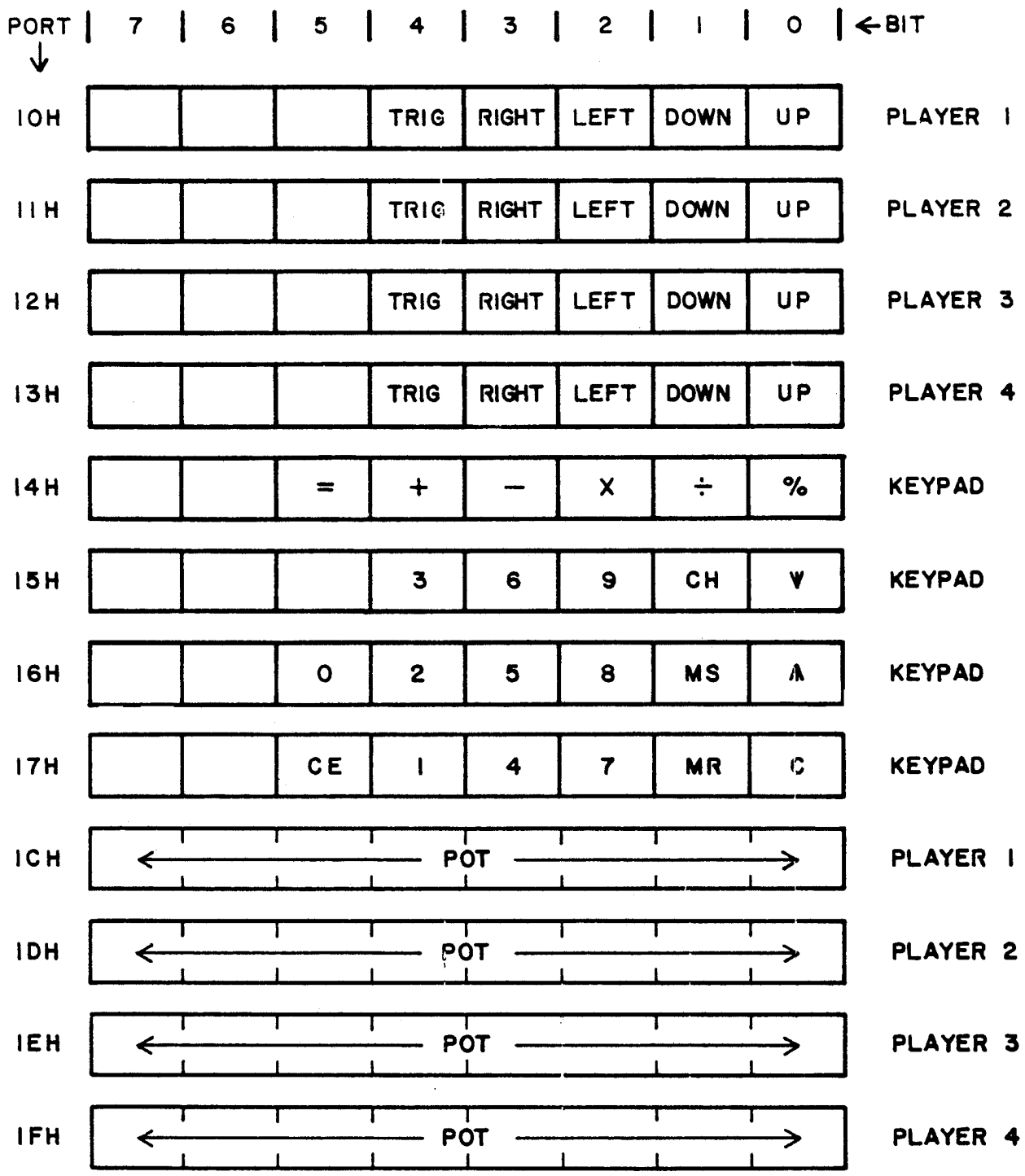
The system will accommodate up to four player control handles at once. Each handle has five switches and a potentiometer. The switches are read by the Z-80 on input ports 10H - 13H and are not debounced. The switches are normally open and normally feedback 0's.

The signals from the potentiometers are changed to digital information by an 8-bit Analog-to-Digital Converter. The four pots are on input ports 1CH - 1FH. All 0'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 0-5 of ports 14H-17H. The data is normally 0 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



MASTER OSCILLATOR

The frequency of the master oscillator is determined by the contents of several output ports. Port 10H 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 0, 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 the modulation is set by the vibrato speed register (upper 2 bits of output port 14H); 00 for fastest and 11 for slowest.

Frequency modulation is accomplished by adding a modulation value to the contents of port 10H 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 0, the corresponding bit in the modulation value word will be set to 0. In vibrato modulation, the modulation value alternates between 0 and the contents of the vibrato volume register.

Modulation can be completely disabled by setting the master volume to 0 if noise modulation is being used, or by setting the vibrato depth to 0 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:

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

Proofing Note: 'F_.' = 'F Subscript Up-Arrow'

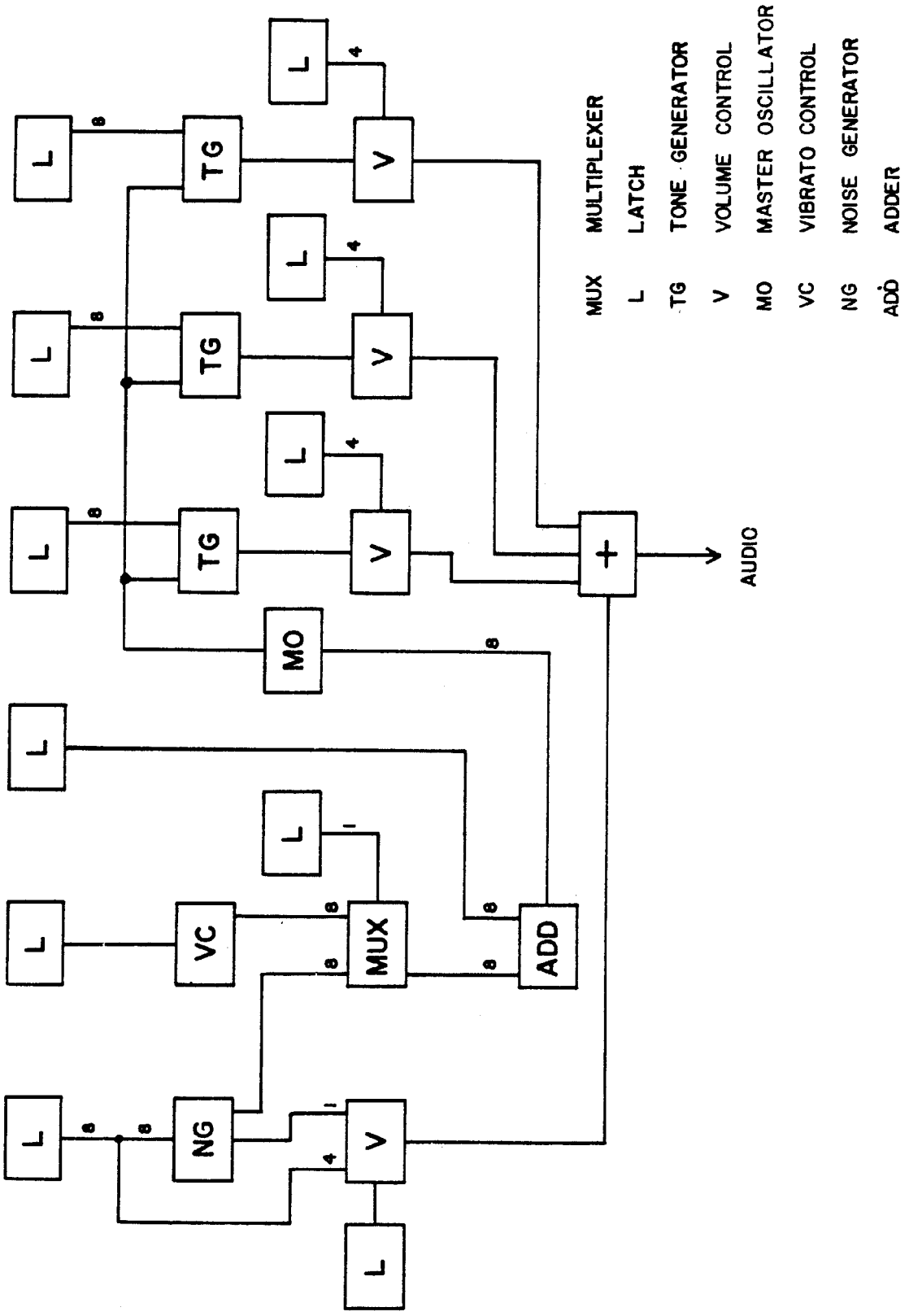
The tone volumes are controlled by the output ports 15H and 16H. The lower 4 bits of port 16H set Tone A Volume, the upper 4 bits set Tone B Volume. The lower 4 bits of port 15H set Tone C Volume. Noise can be mixed with the tones by setting bit 5 of port 15H to 1. In this case the noise volume is given by the upper 4 bits of port 17H. In all cases a volume of 0 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 10H.

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	10H

AUDIO GENERATOR BLOCK DIAGRAM



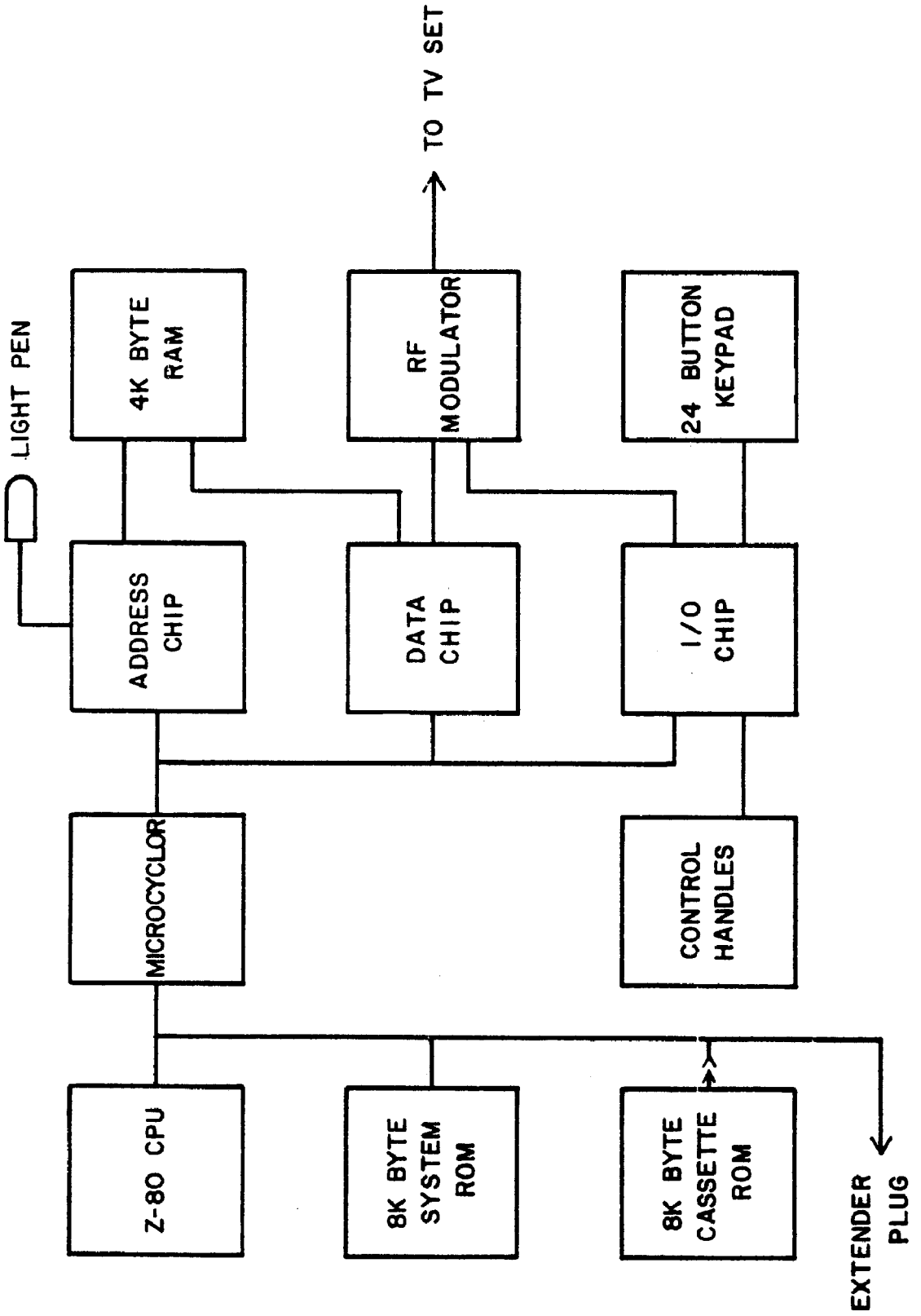
OUTPUT PORTS

PORT NUMBER	FUNCTION
0H	Color Register 0
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
10H	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
10H	Player 1 Handle
11H	Player 2 Handle
12H	Player 3 Handle
13H	Player 4 Handle
14H	Keypad Column 0 (right)
15H	Keypad Column 1
16H	Keypad Column 2
17H	Keypad Column 3 (left)

SYSTEM BLOCK DIAGRAM



MICROCYCLER

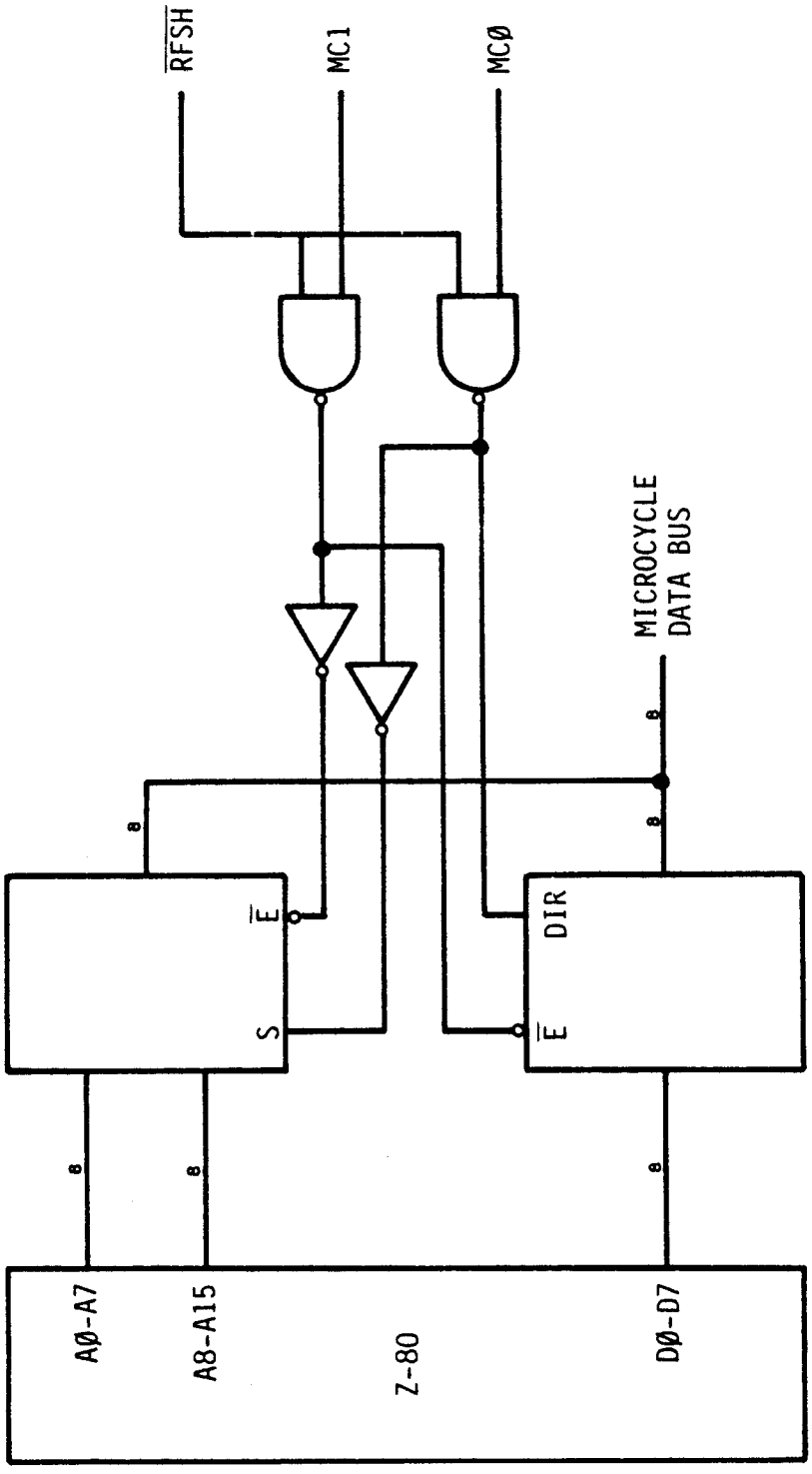
The purpose of the microcycler is to combine the 16-bit Address Bus and the 8-bit Data Bus from the Z-80 into one 8-bit Microcycle Data Bus to the Data Chip, Address Chip, and I/O Chip. This was done to reduce the pin count on the custom chips.

The Microcycle Data Bus can be in any of four modes. Its mode is controlled by MC0 and MC1 coming from the Data Chip and RFSH# from the Z-80. The modes are shown below.

RFSH#	MC0	MC1	Microcycle Data Bus Contents
0	0	0	A0 - A7 from Z-80
0	0	1	A0 - A7 from Z-80
0	1	0	A0 - A7 from Z-80
0	1	1	A0 - A7 from Z-80
1	0	0	A0 - A7 from Z-80
1	0	1	A8 - A15 from Z-80
1	1	0	D0 - D7 from Z-80
1	1	1	D0 - D7 to Z-80

MC0 and MC1 change 140 usec after the rising edge of Phi. 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-80 address from the 8-bit Microcycle Data Bus. This address is then fed through MUX I and MUX II to MA0-5 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 0 to FFFH once every frame (1/60 sec.).

MUX I sends either the Scan Address or Z-80 Address to its 12 outputs. An output of the Scan Address Generator controls MUX I. If the Scan Address Generator and the Z-80 request memory cycle at the same time, the Scan Address Generator will have higher priority and the Z-80 will be required to wait (by the WAIT# output). The Scan Address Generator never requires the memory for more than one consecutive memory cycle, so the Z-80 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.

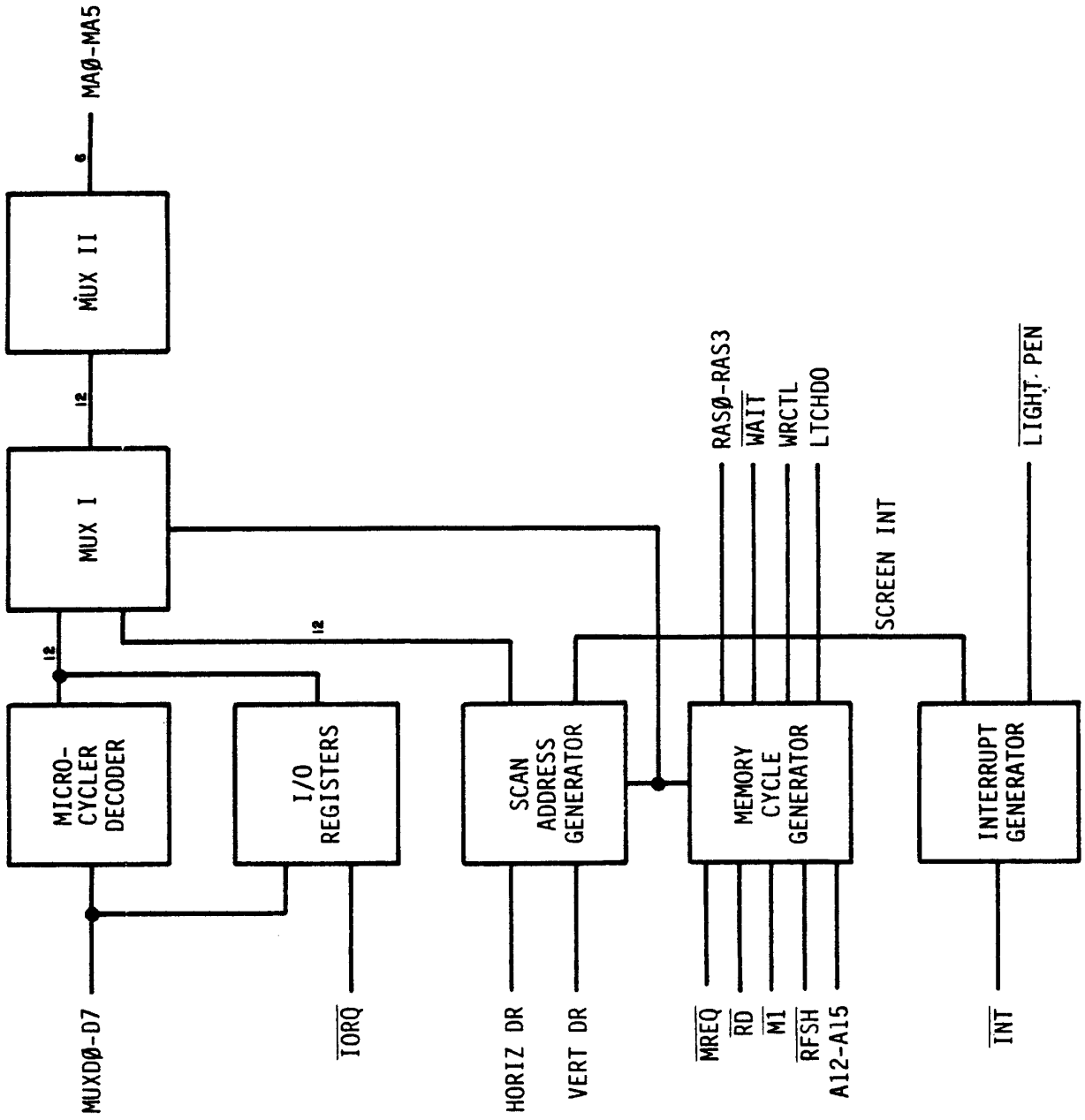
The Memory Cycle Generator controls memory cycles generated by either the Z-80 or Scan Address Generator. MREQ#, RD#, M1#, RFSH#, and A12-A15 are from the Z-80. A12-A15 are fed directly from the Z-80 because if they were brought out of the microcycle decoder, they would arrive too late in the memory cycle. The RAS0 - RAS3 outputs are used to activate memory cycles. In the consumer game, only RAS0 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 LTCHD0 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. LTCHD0 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-80 and Scan Address Generator both request memory at the same time. WAIT# is also generated for one cycle every time the Z-80 requests a memory access, even if there is no conflict with the Scan Address. This is because the microcycler slows down Z-80 memory accesses. The Z-80 address bus and data bus must time share the microcycle bus so the Z-80 data reaches the microcycle bus very late in the memory cycle.

The INT Generator generates two types of interrupts to the Z-80; 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 0 to 255). The line at which the interrupt will occur is determined by the Z-80. This interrupt can be used for timing since the TV rescans every line once every 1/60th 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-80 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-80 input and output instructions. Z-80 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 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 (MD0 - 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 0 and SERIAL 1 are sent through MUX I and 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 voices of VIDEO, R-Y, and B-Y. 2.5V is a 2.5V D C reference level.

The Clock Generator generates 0G and PX# from 7M. These are the clocks for the rest of the system. The Frequency of PX# is half that of 7M and the frequency of 0G is half that of PX#.

The Microcycle Generator generates the microcycle control bits, MC0 and MC1 from IORQ#, MREQ#, RD#, and M1#, all from the Z-80.

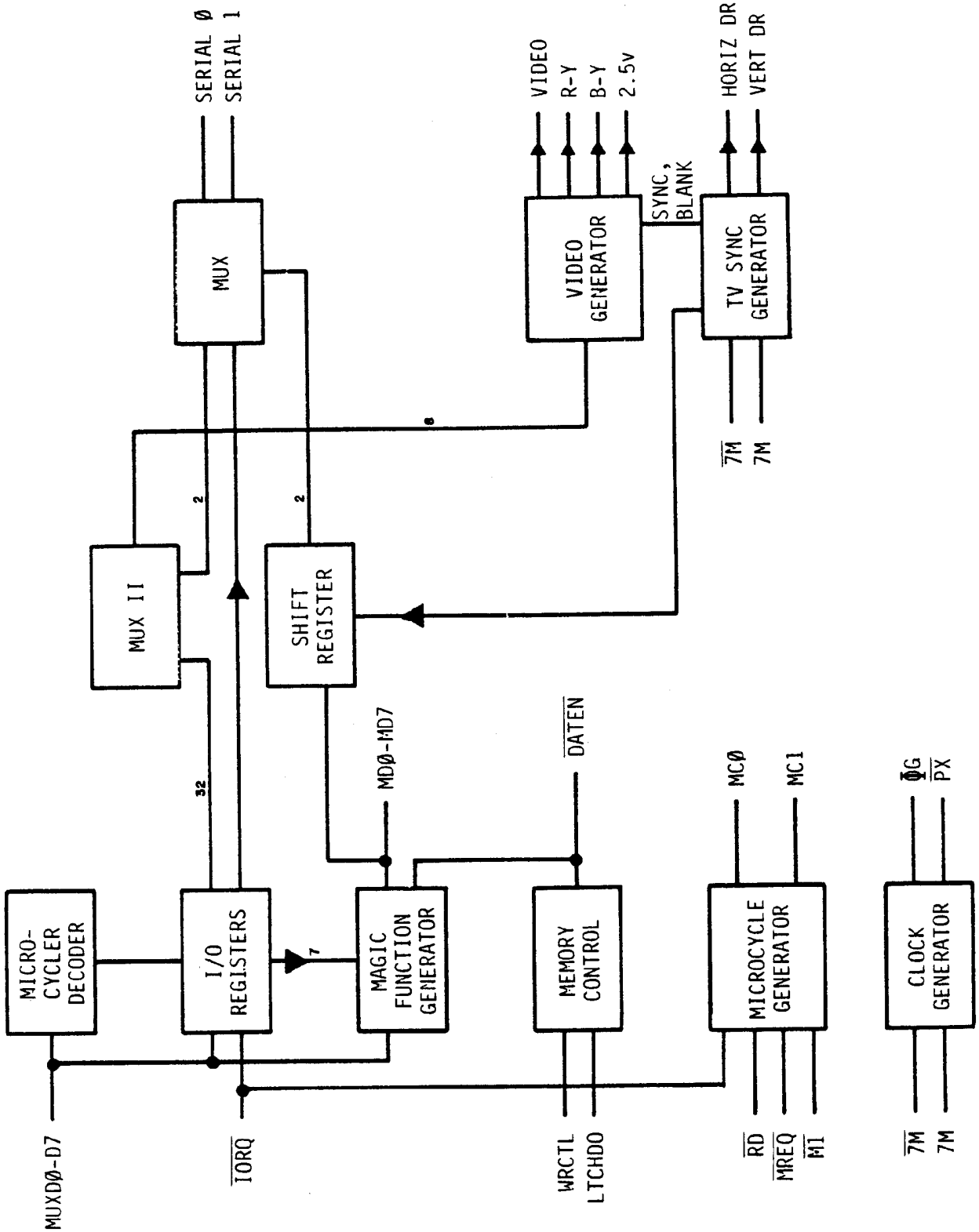
In memory write cycles WRCTL is activated and the Memory Control circuit generates DATEN#. The Magic Function Generator takes the data from the Z-80 on MUXD0 - D7 and transfers it to MD0 - MD7. If a Magic write is being done, the Magic Function Generator will modify the data as required before it places it on the memory data bus.

A Magic write is a memory write cycle in which data is written to a location, (X) from 0 to 16K. All memory from 0 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 MD0 - MD7 to MUXD0 - MUXD7. Also, LTCHD0 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-80 just as in the Address Chip.

DATA CHIP BLOCK DIAGRAM



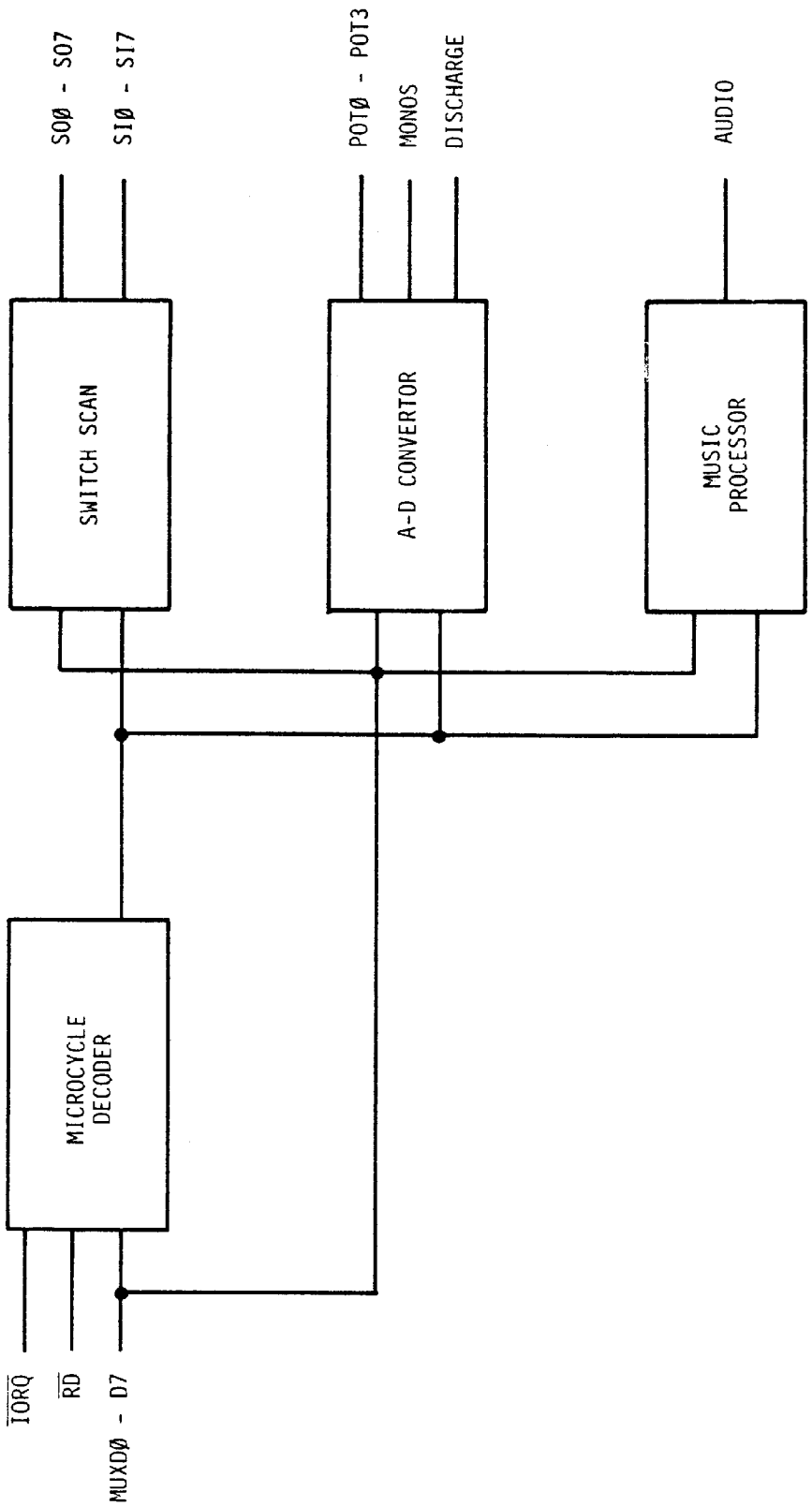
I/O CHIP DESCRIPTION

The Z-80 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 SO0-SO7 lines will be activated. When a line is activated, the switch matrix will feed back eight bits of data on SI0-SI7. This data is in turn fed to the Z-80 through MUXD0 - MUXD7.

The Z-80 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-80 simply reads this RAM with input instructions.

The Z-80 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-80.

Master Oscillator Frequency is a square wave whose frequency is determined by the 8 binary inputs to the Master Oscillator. This 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 0, 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 0 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.

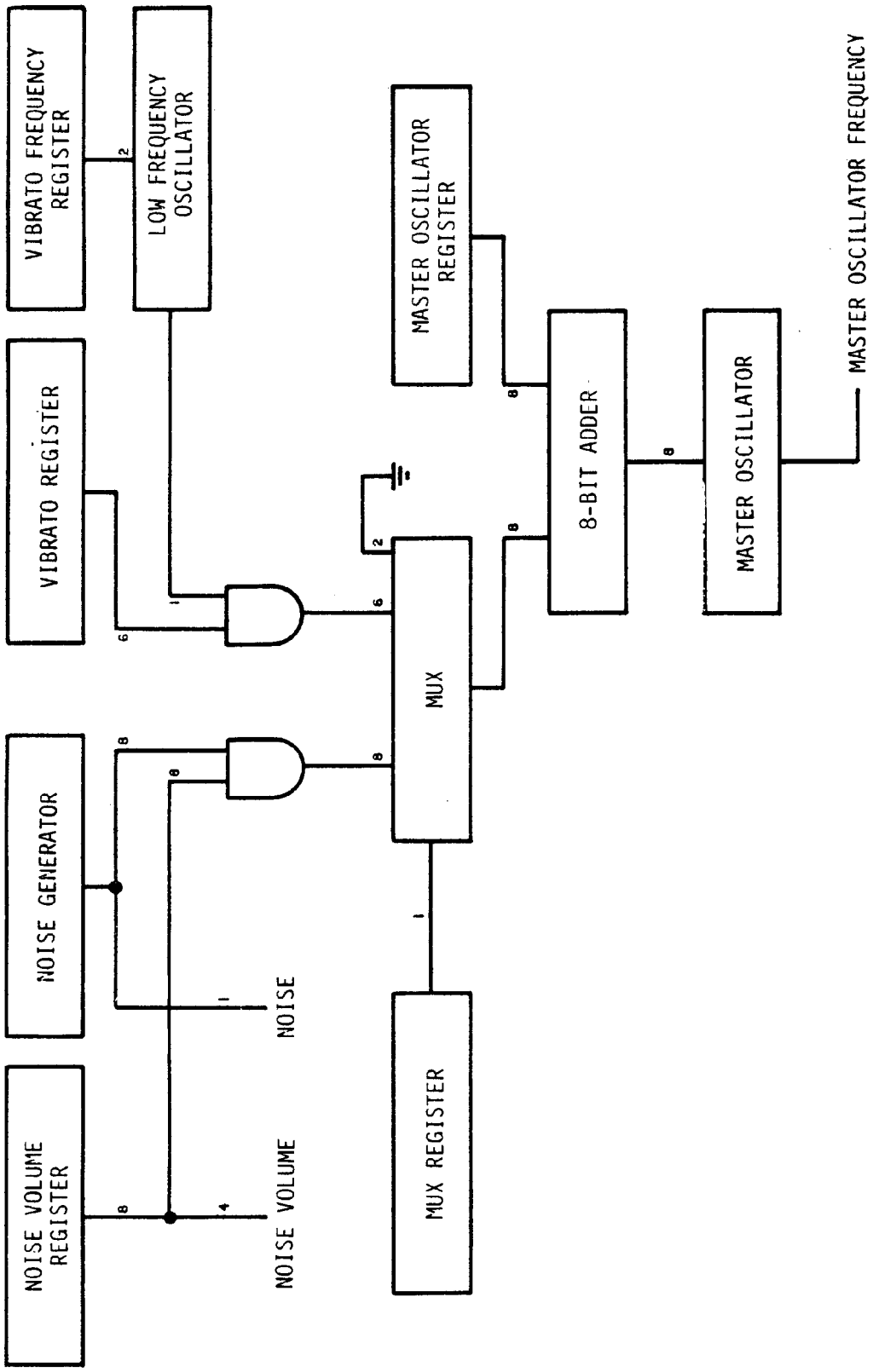
If MUX REG contains 1, then data from the Noise System will be fed through the MUX. The 8-bit word from the Noise Volume Register determines which bits from the Noise Generator will be present at the output of the AND gates.

If a bit in the Noise Volume Register is 0, then the corresponding bit at the output of the AND gates will be 0. 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.

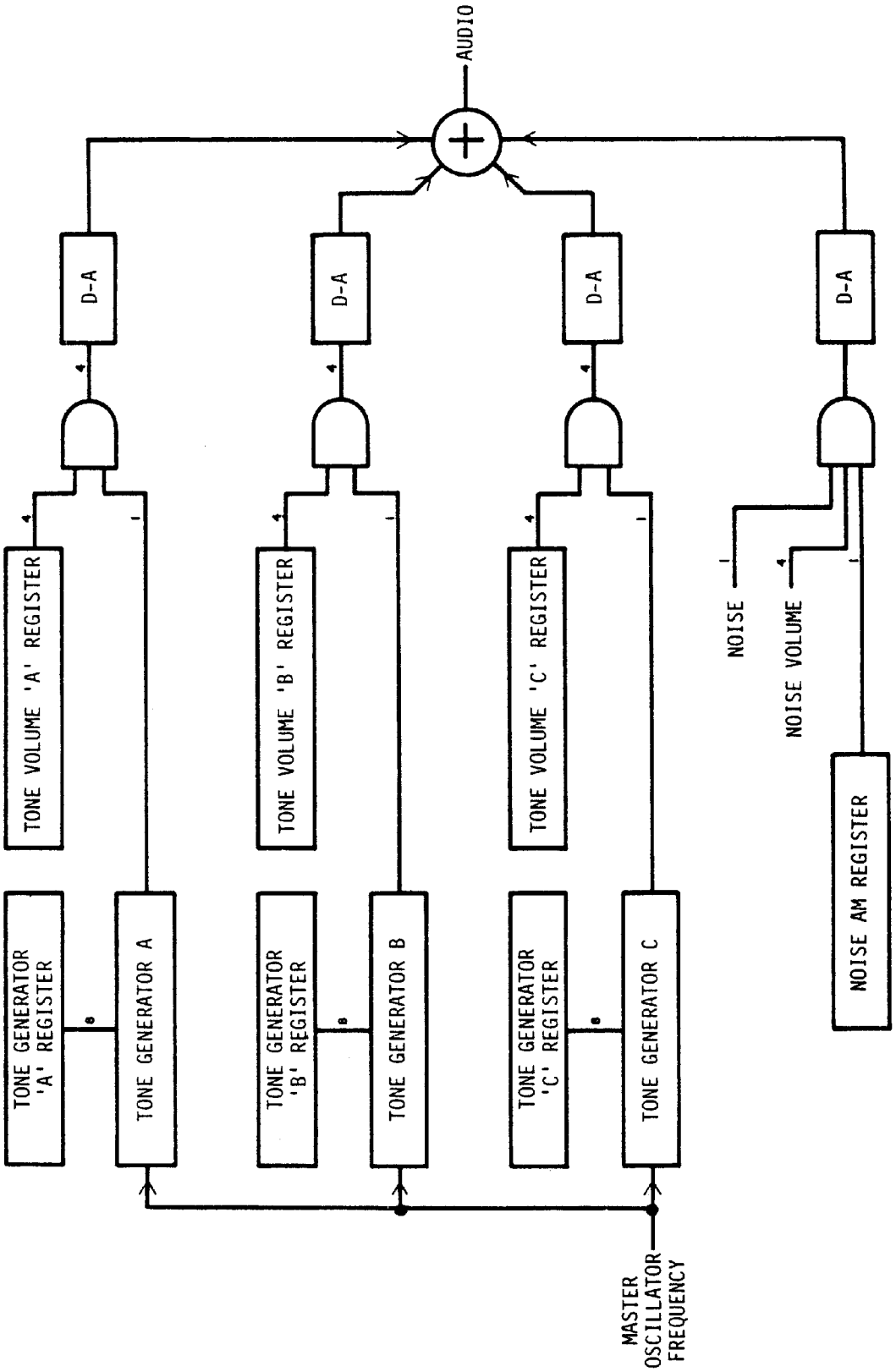
In the second part of the Music Processor, the square wave from the Master Oscillator is fed to three Tone Generator circuits which produce square waves at their outputs. The frequency of their outputs is determined by the contents of their Tone Generator Register and Master Oscillator Frequency. The 4-bit words at the output of the AND gates oscillate between 0 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 BLOCK DIAGRAM



TONE GENERATORS



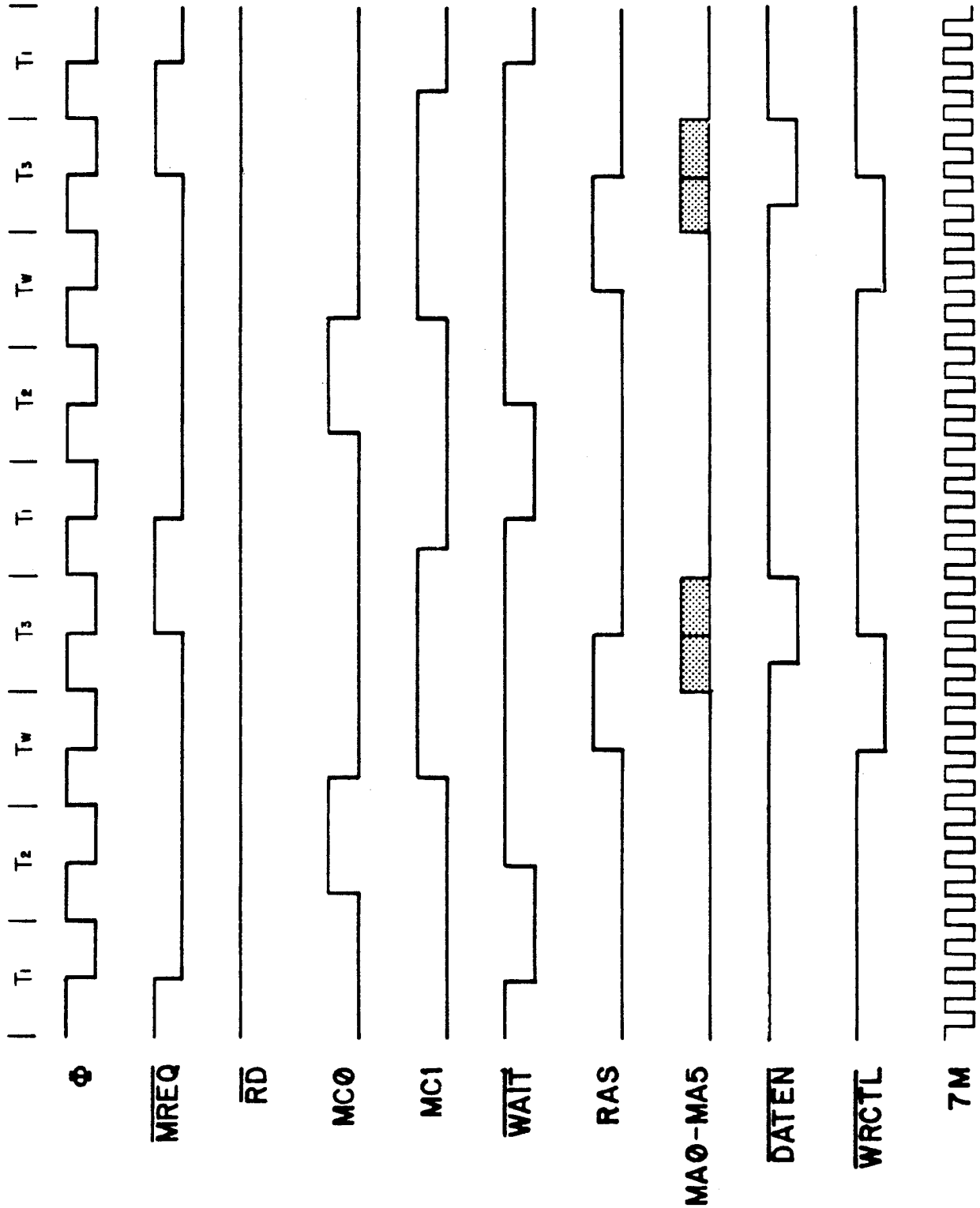
CUSTOM CHIP TIMING

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 cause the transition. The actual delay is given in "Electrical Specifications for Midway Custom Circuits."

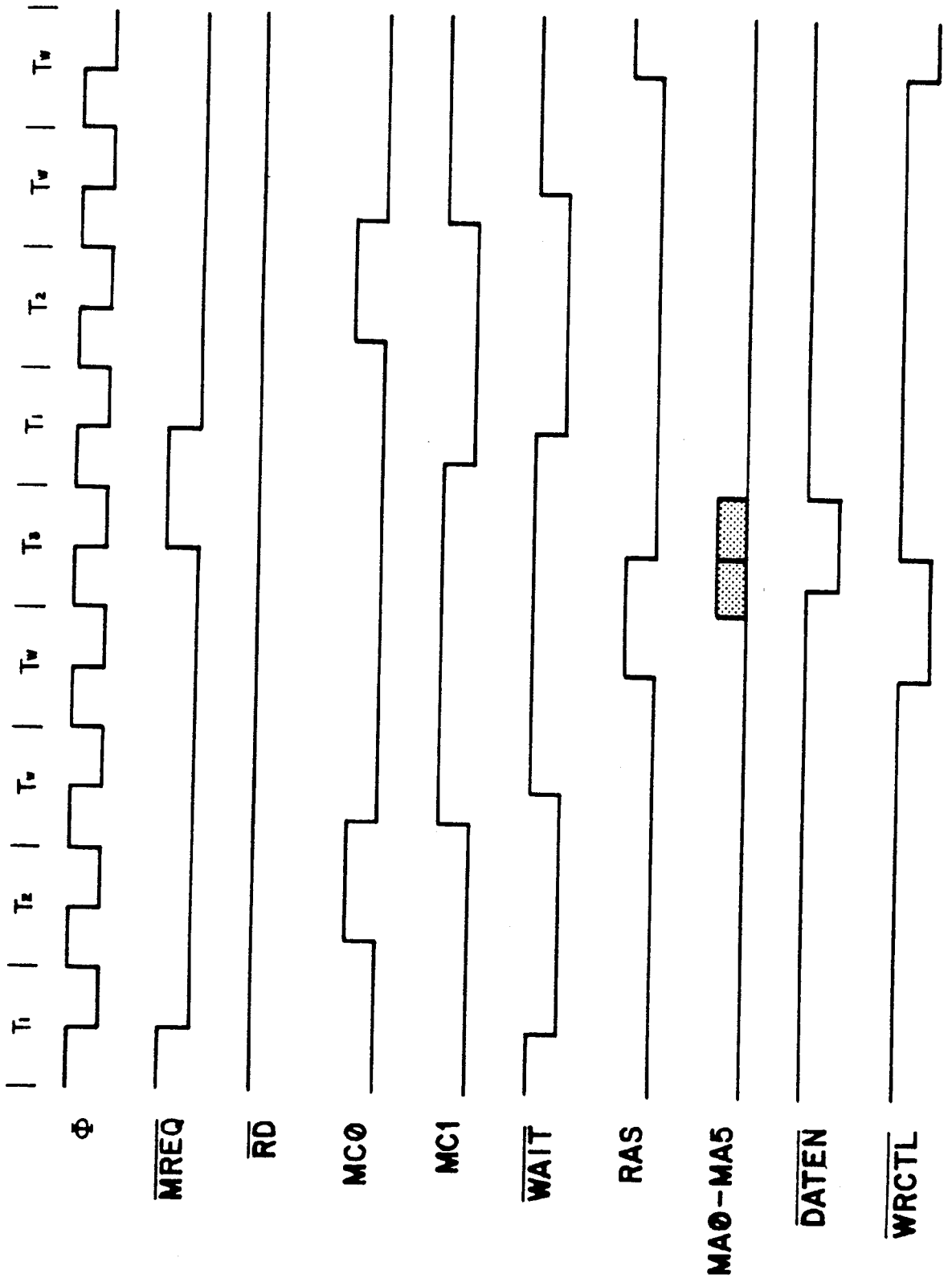
MUXD0 - MUXD7 is an 8-bit bidirectional address and data bus for the custom chips. By using this technique, 16 bits of address and 8 bits of data can be sent to the custom chips on 8 wires. The state of the bus is determined by MC0 and MC1 from the data chip and RFSH# from the Z-80.

RFSH#	MC1	MC0	
L	L	L	A0 - A7 to custom chips
L	L	H	A0 - A7 to custom chips
L	H	L	A0 - A7 to custom chips
L	H	H	A0 - A7 to custom chips
H	L	L	A0 - A7 to custom chips
H	L	H	A8 - A15 to custom chips
H	H	L	D0 - D7 to custom chips
H	H	H	D0 - D7 from custom chips

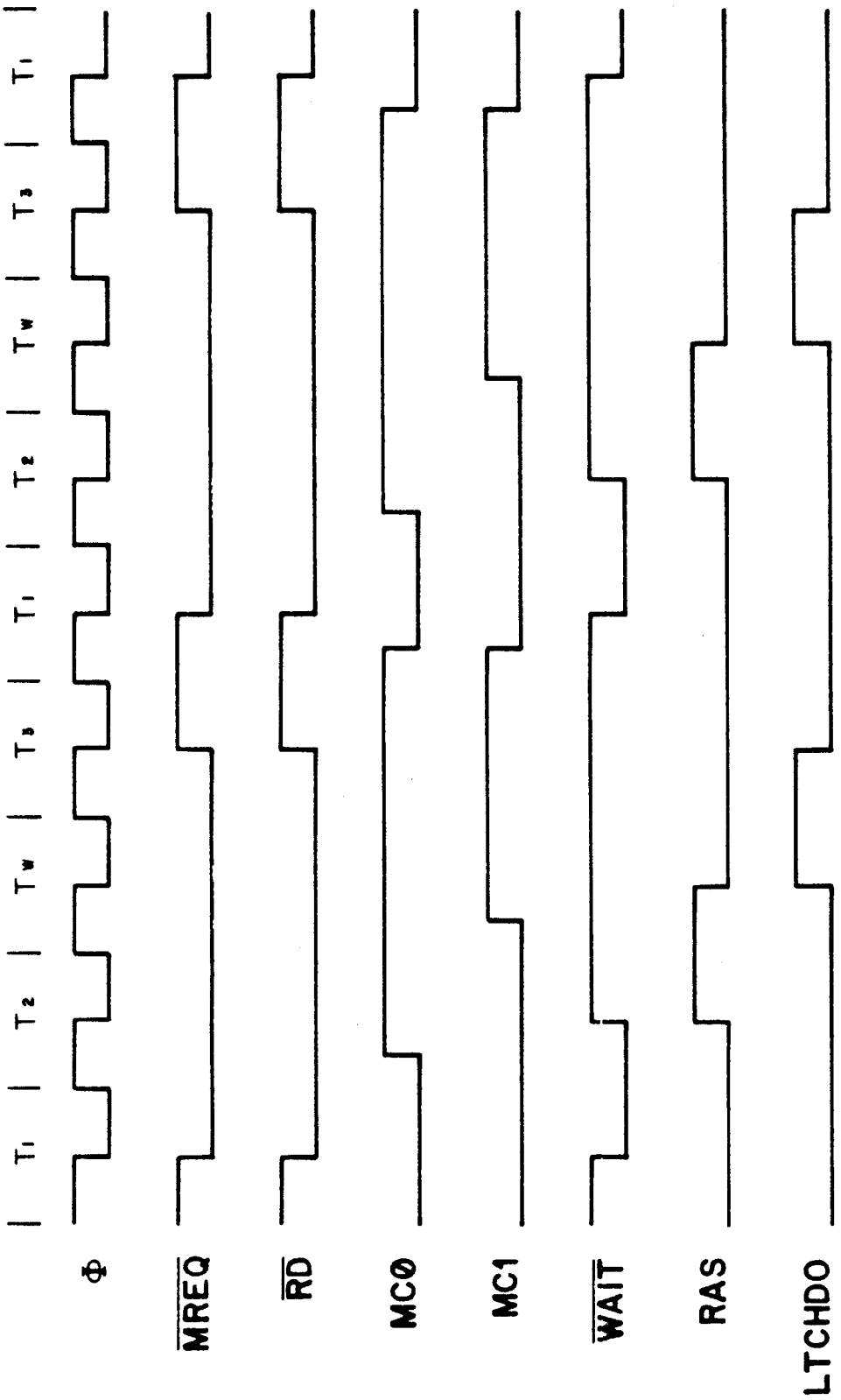
MEMORY WRITE WITHOUT EXTRA WAIT STATE



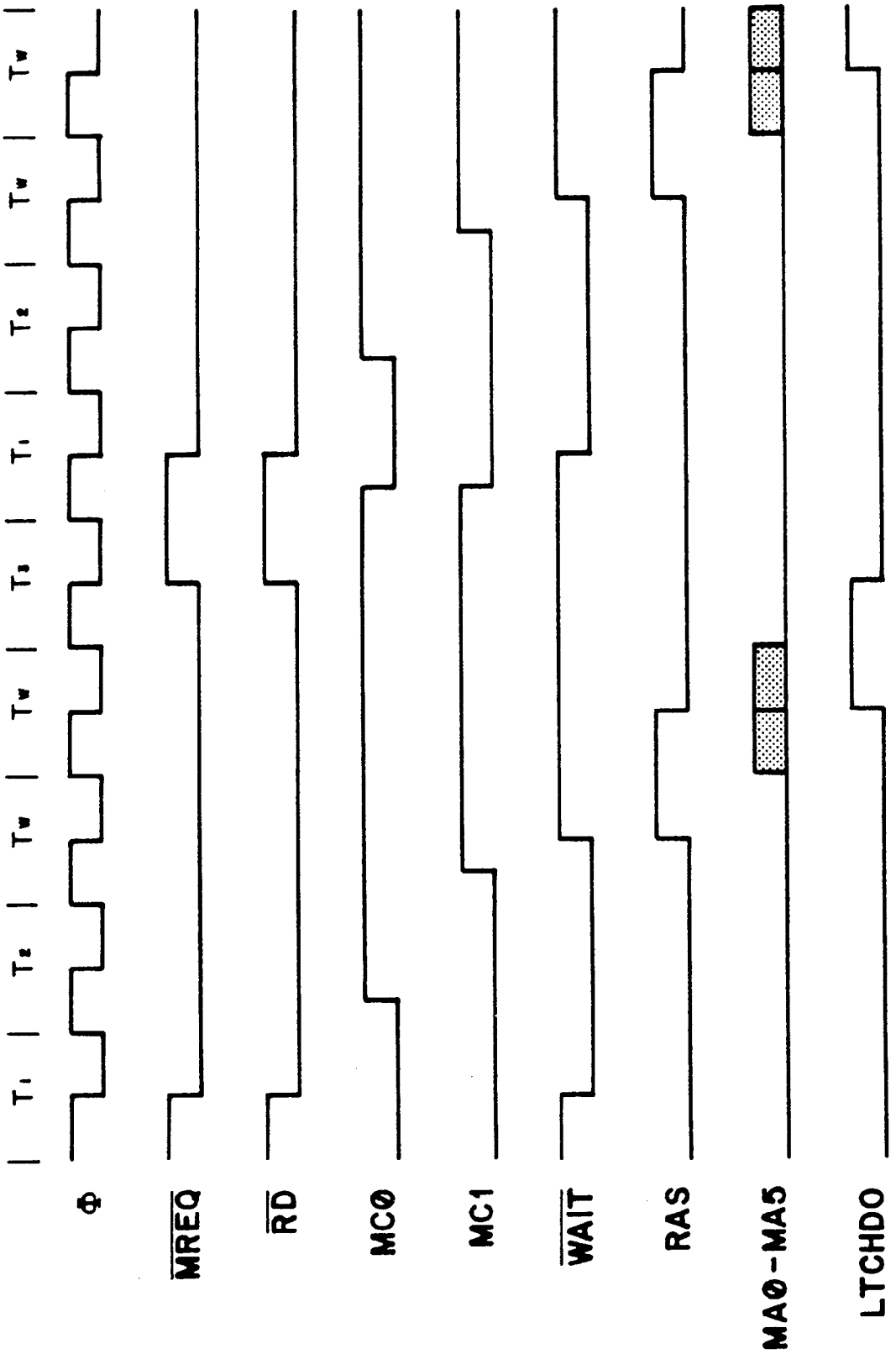
MEMORY WRITE WITH VIDEO WAIT STATE



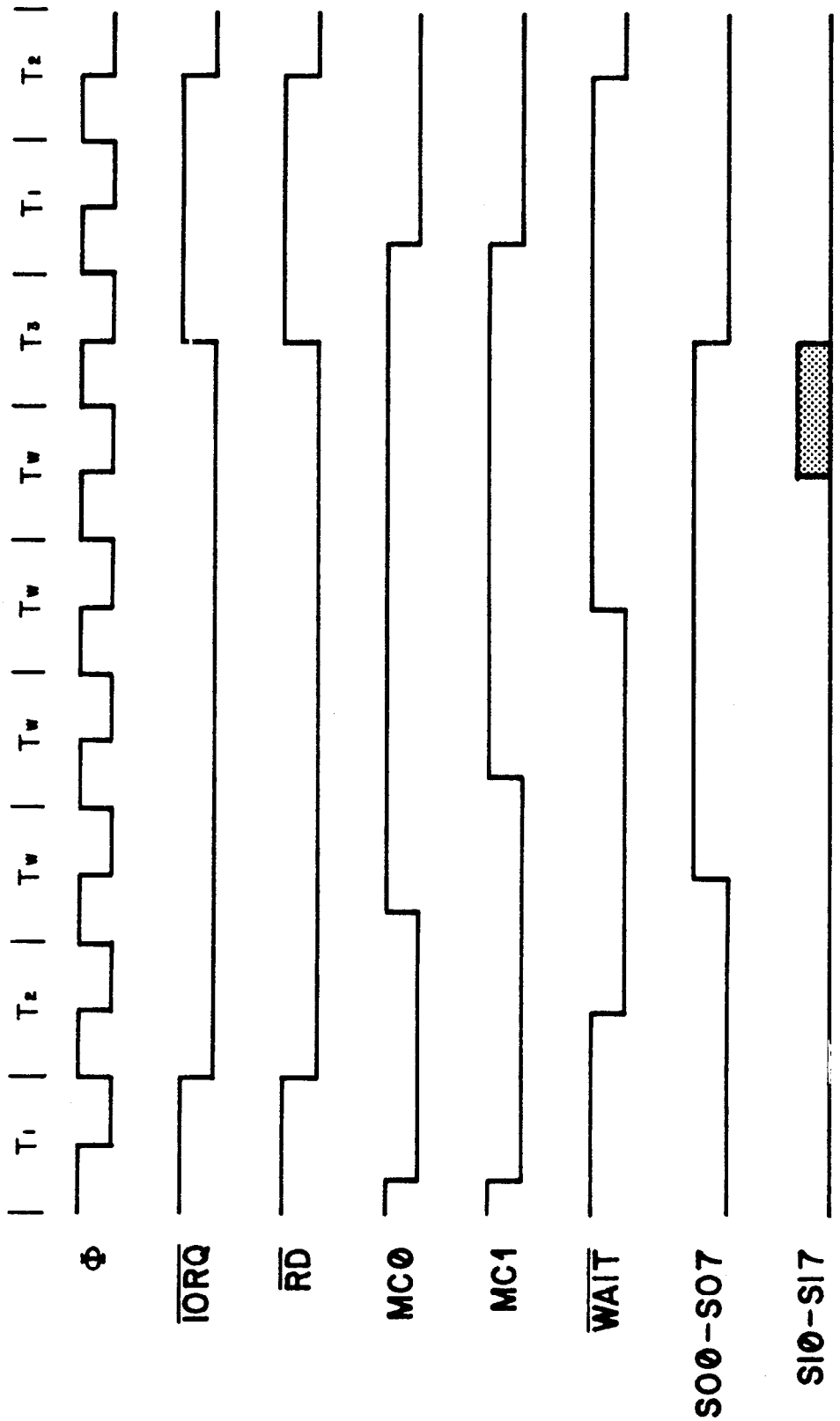
MEMORY READ WITHOUT EXTRA WAIT STATE



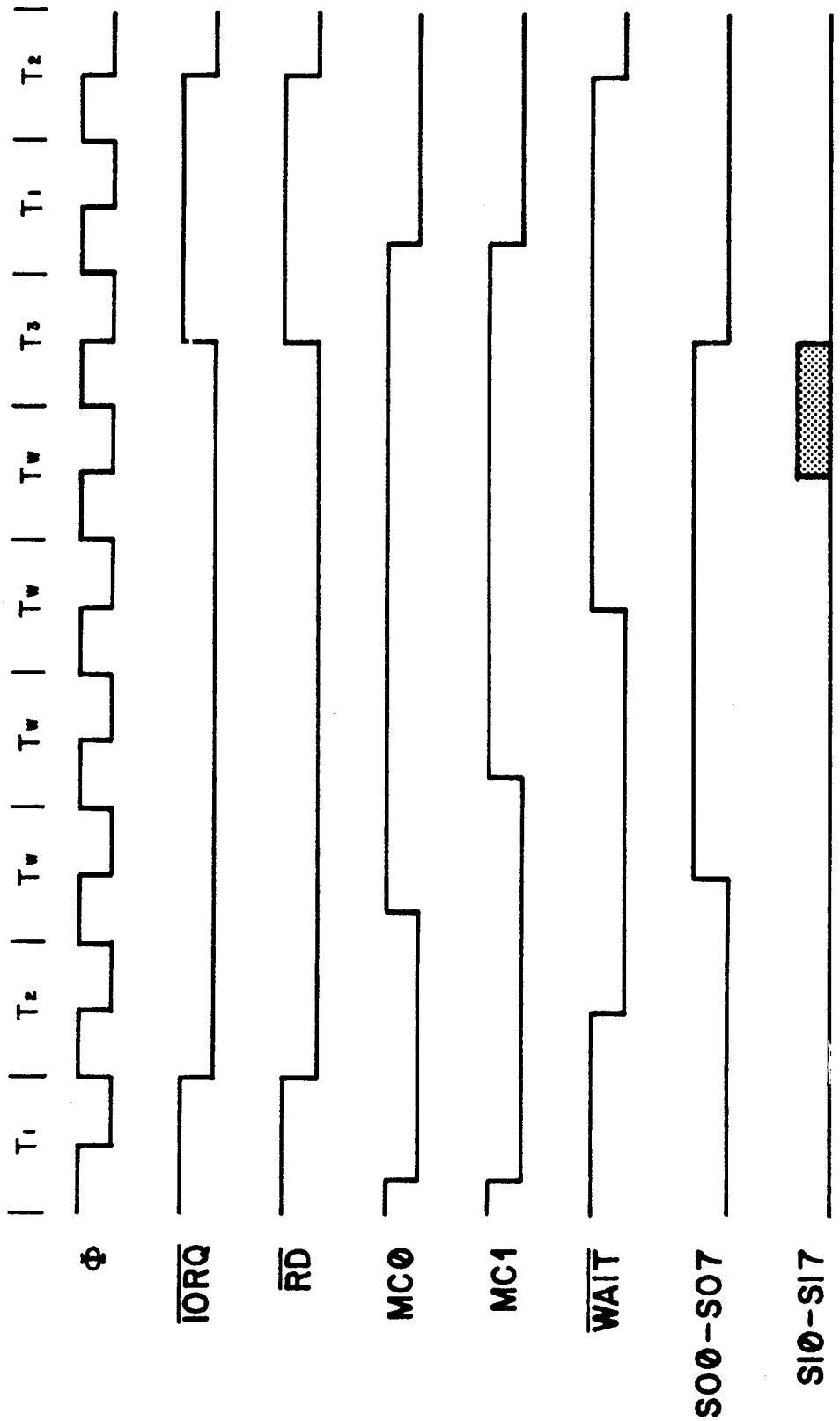
MEMORY READ WITH VIDEO WAIT STATE



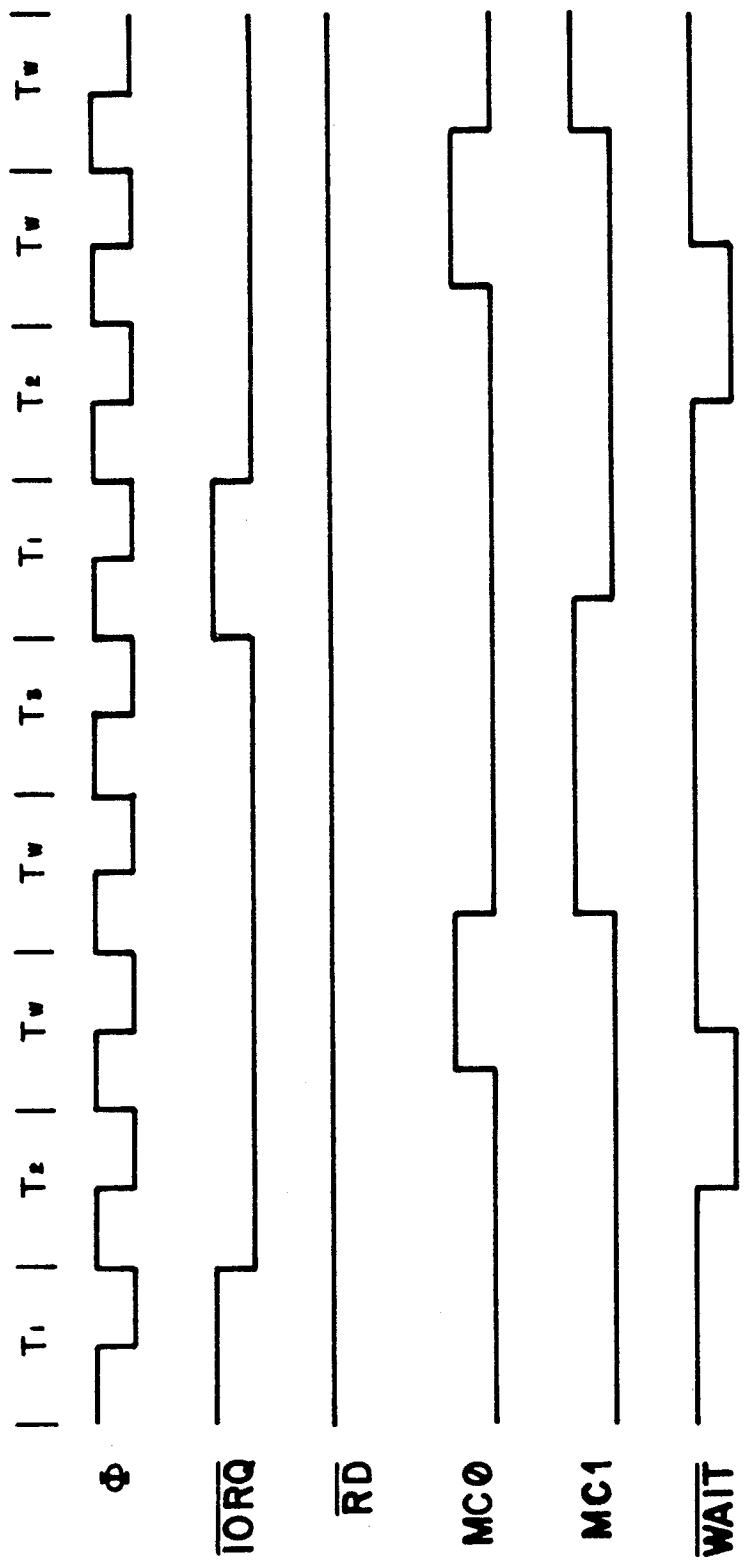
I/O READ FROM PORT 10H - 17H



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



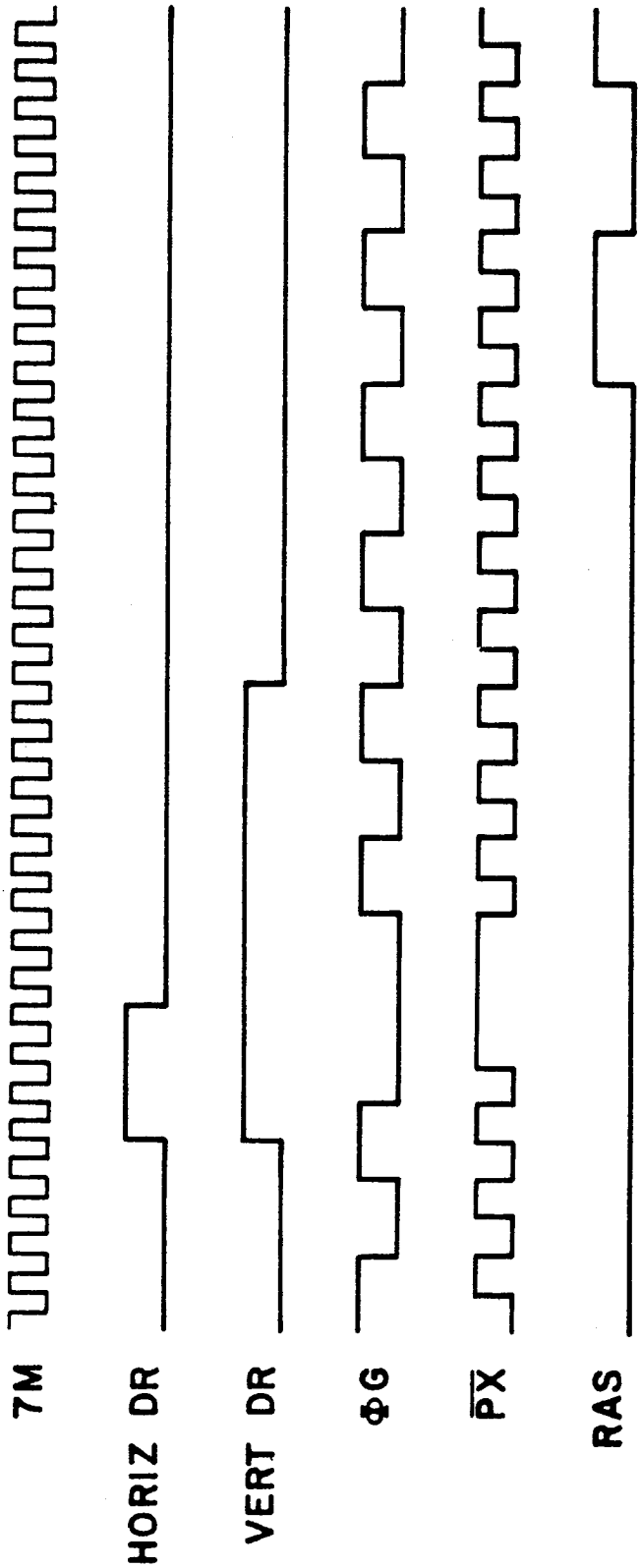
I/O WRITE



VIDEO TIMING

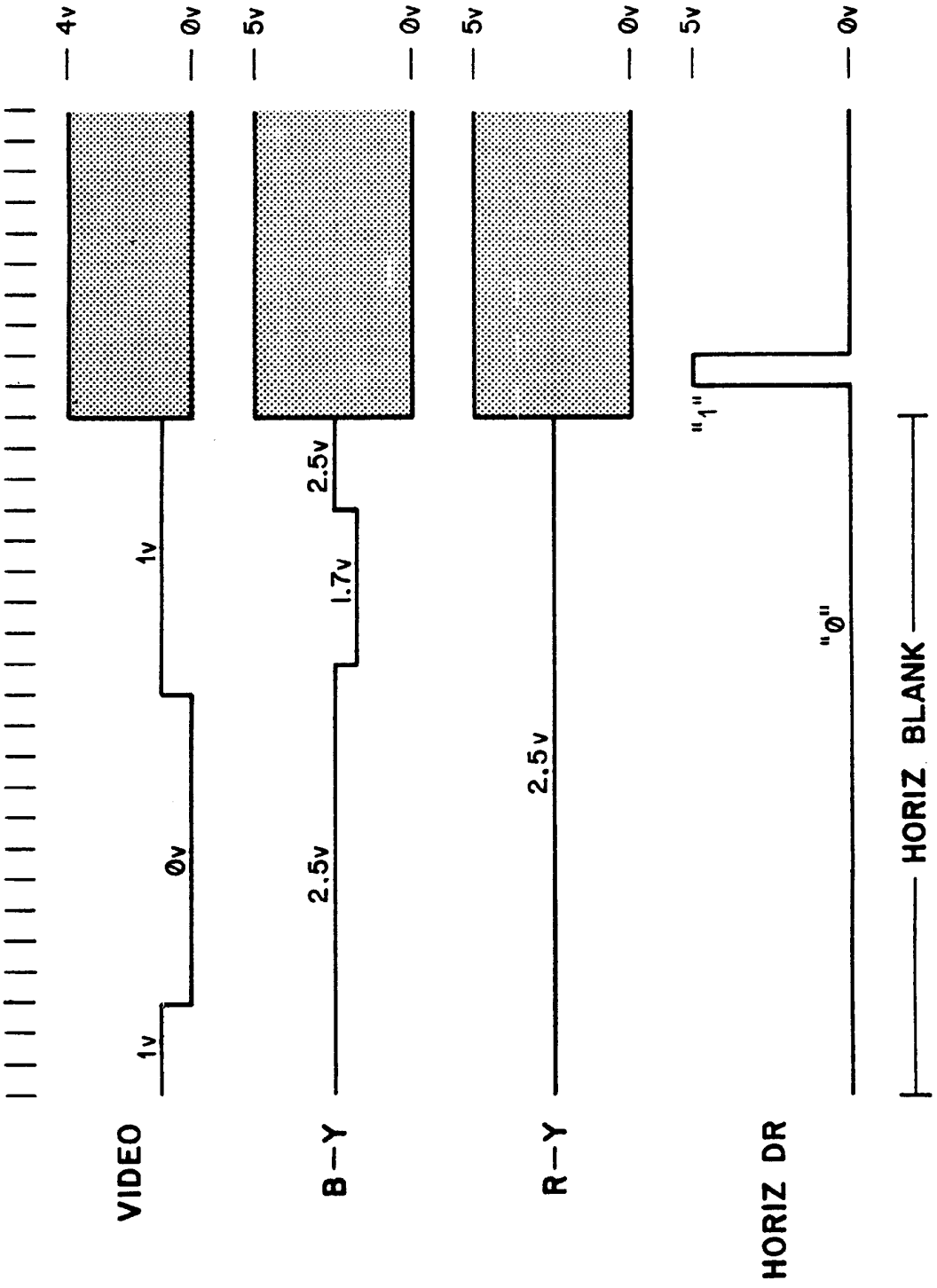
The frequency of PX# is half that of 7M and the 0 is one-fourth 7M. There are 455 cycles of 7M per horizontal line and 133 3/4 Phi cycles per line. Because of the extra 3/4 cycle, 0 must be resynchronized at the beginning of each line. This is done by stalling 0 for 3 cycles of 7M. PX# is also stalled 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, PHI G, PX AND RAS



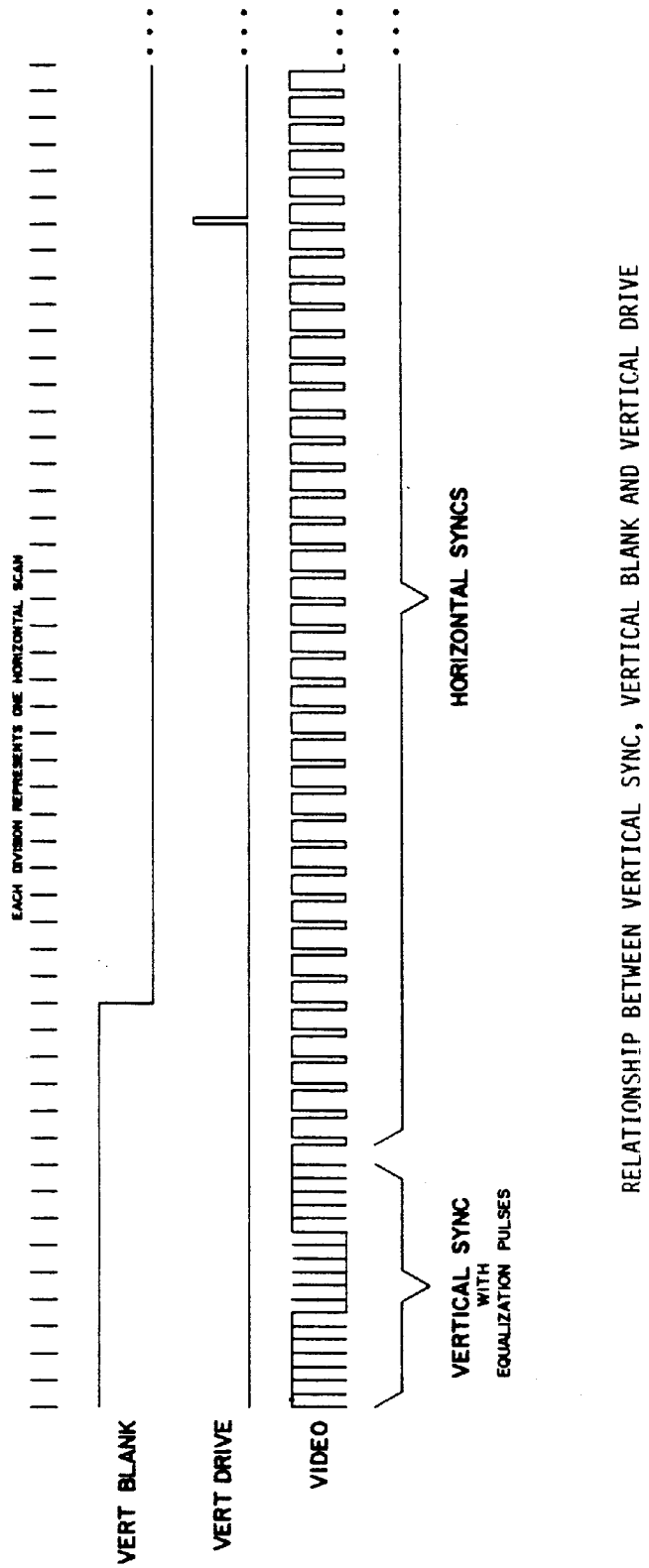
RELATIONSHIP BETWEEN 7M, HORIZ DR, VERT DR, $\overline{\Phi G}$, $\overline{PX}$ AND RAS

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



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



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

- 1 -

ELECTRICAL SPECIFICATION FOR MIDWAY CUSTOM CIRCUITS

I. GENERAL SYSTEM PARAMETERS

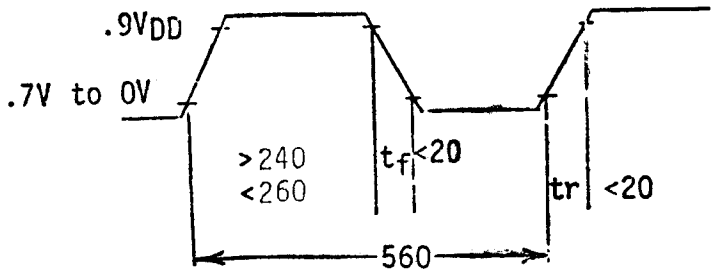
REVISIONS:	
1/14/77	N/C
1/27/77	A 135
3/25/77	B
7/6/77	C

I. A. Power Supplies

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

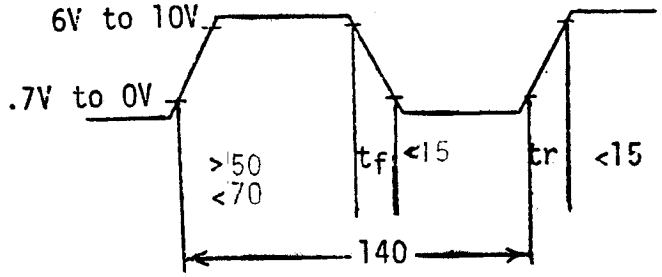
I. B. Timing Signals

1. 0 & 0#; Period = 560nsec, High time^* 240nsec to 260nsec.
   0 & 0# have zero level crossover +1 volt -0 volts.
   t_r, t_f^* less than 20nsec



(Times are in nsec)

2. 7M & 7M#; Period = 140nsec, High time^+ 50nsec to 70nsec
   7M & 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 (MUXD0-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 @ -430uA

3. Logic 1: 2.7V @ +70uA

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

5. Transient Response: Transition from High Impedance to 0 or 1 will be complete within 442nsec of the 90% point of 0# of the last wait state of input cycle or 442nsec of the 90% point of the 0 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 0#. The high address byte will be valid at least 79nsec before 0#. The data byte will be valid 55nsec before 0#.

I. D. RAM Data Bus (MD0-MD7) - Home Game

1. The RAM Data Bus will require three state logic buffers.

2. Logic 0: .5V @ -25uA

3. Logic 1: 2.7V @ +25uA

4. High Impedance: 5uA 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 (MD0-MD7) - Commercial Game

1. The RAM Data Bus will require three state logic buffers.

2. Logic 0: .5V @ -200uA

3. Logic 1: 2.7V @ +25uA

4. High Impedance: 5uA 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 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, <= 55'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 0 references refer to the 0 and 0# inputs to the address and I/O chip.

II. A. Data Chip

1. Input Pin List

	V0 (V)	V1 (V)	t_d (Low)^1 (nsec)	t_d (High)^1 (nsec)	Ref.
MREQ#	.5	2.45	132	6	7M
RD#	.5	2.45	12	6	7M
IORQ#	.5	2.45	112	126	7M
7M	See Section I.B.				
7M#	See Section I.B.				
WRCTL#	.5	3.1	82	82	7M
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	V0	I0	V1	I1	CAP	t_p	Ref.
	--	--	--	--	---	----	----
	(V)	(uA)	(V)	(uA)	(pf)	(nsec)	
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	
0	Note 4	400	2.7	20	10	100	7M
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
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 .6usec.

1 t_d (Low) and t_d (High) is maximum time in nsec except where a minimum is shown.
2 For IORQ# Ref. to 0# t_d (Low)=132nsec t_d (High)=6nsec.
3 Serial 0 and Serial 1 will operate at 7MHz
4 .5 + noise generated by chip.
5. Tap on both resistor chains for a capacitor. Will become test input with voltage applied > 8V.
6 The Z80 0 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				
RD#	.5	2.45	0 or 0#	166	172 0 or 0#
IORQ#	.5	2.45	0^6	146 0#	132 0
0	See Section I.B.				
0#	See Section I.B.				
SI0	.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 (uA)	V1 (V)	I1 (uA)
Audio	Note 4	Fmax - 20KHz			
Discharge	Note 5	.5V	4V		
SO0	Note 3	Note 7	200	4V	1650
SO1	Note 3	Note 7	200	4V	1650
SO2	Note 3	Note 7	200	4V	1650
SO3	Note 3	Note 7	200	4V	1650
SO4	Note 3	Note 7	200	4V	1650
SO5	Note 3	Note 7	200	4V	1650
SO6	Note 3	Note 7	200	4V	1650
SO7	Note 3	Note 7	200	4V	1650
POT 0	Note 2		5	V_DD-.5	50
POT 1	Note 2		5	V_DD-.5	50
POT 2	Note 2		5	V_DD-.5	50
POT 3	Note 2		5	V_DD-.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 2usec max. Drop of isolation diode will be 0.7V max. SO must drive 2kohm in the high level. Max C load of SO shall be 300 pf. SI input shall kill device enabled by 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 100kohm.

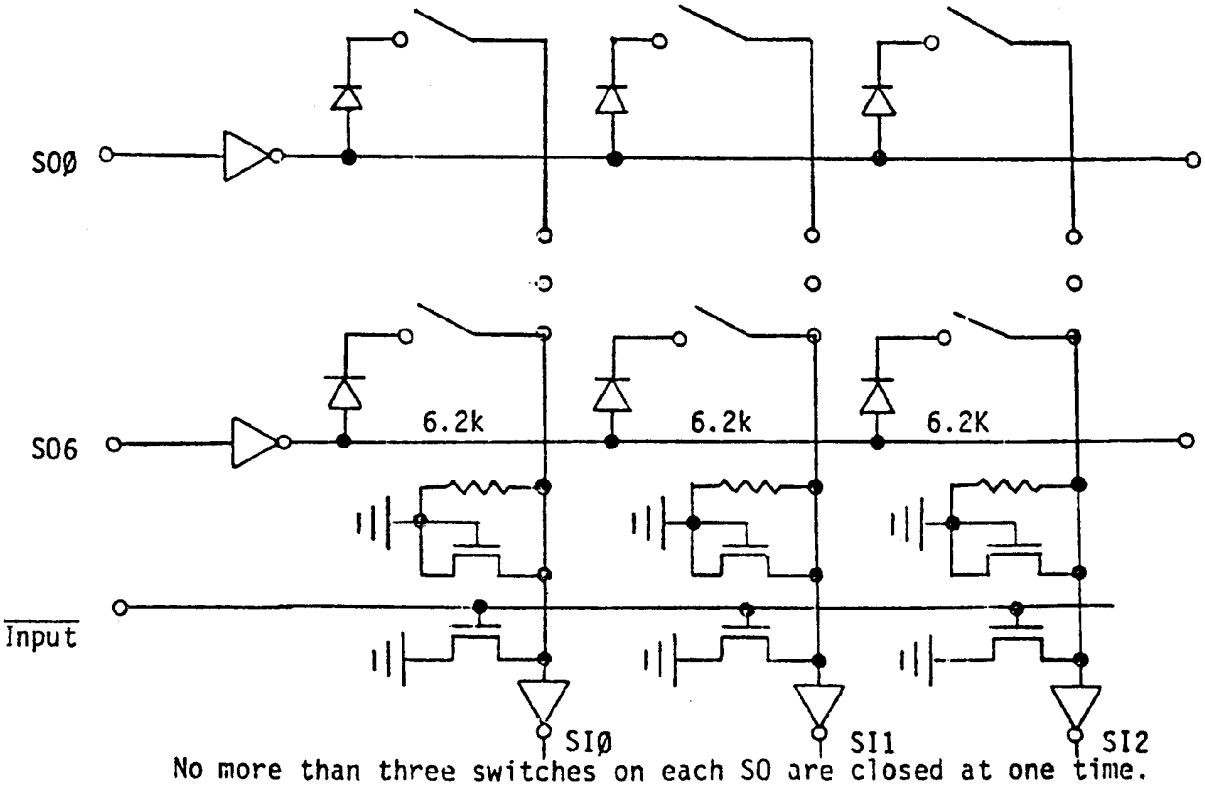
Note 5 Discharge is open drain to V_SS. Discharges .01ufd capacitor to .2V in 144usec.

Note 6 For IOREQ# Ref. to 0# t_d (Low)=152nsec t_d(High)=166nsec.

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

Miscellaneous Timing

Time for MO Adder - 20 max



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

II. C. Address Chip

1. Input Pin List

	V0 (V)	V1 (V)	t_pd (Low) (nsec)	t_pd (High) (nsec)	REF
RFSH#	.5	2.45	222 0	216	0
MREQ#	.5	2.45	152 0#	166	0 or 0#
RD#	.5	2.45	172 0 or 0#	166	0 or 0#
MI#	.5	2.45	176 0	242	0
A12^1	.5	2.45			0
A13^1	.5	2.45			0
A14^1	.5	2.45			0
A15^1	.5	2.45			0
IORQ#	.5	2.45	132 0	146	0#^2
LIGHT PEN#	.5	2.45	Asyn		
TEST#	.5	5.0	DC		
HORIZ. DR.	.5	2.45	Note 3		0#
VERT. DR.	.5	2.45	Note 4		0
0	See Section I.B.				
0#	See Section I.B.				

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. Output

	V0 (V)	I0 (uA)	V1 (V)	I1 (uA)	CAP (pf)	T_pd(Low) (nsec)	T_pd(High) (nsec)	REF
LATCHD0	Note 7	Note 6	3.1	Note 6	10	280	140	0#^
WAIT#	Note 7	400	2.4	20	25	490	490	0#
MA0-MA5	Note 7	400	2.4	20	20	242	240	0# or 0
INT#	Note 7	400	2.4	20	25	490	572	0
RAS0-RAS3	Note 7	400	2.4	20	20	382	382	0#
WRCTL#	Note 7	Note 6	3.1	Note 6	10	382	382	0#

1. Time from High Impedance to 1 or 0 is 200nsec. (from 0_1 of T_1)
2. For IORQ# Ref to 0# t_d (Low)=152nsec t_d (High)=166nsec. 0
3. Horizontal Drive time from low to high is 40nsec after 0#.
   Time from high to low is 100nsec before rising edge of 0.
4. Vertical Drive will transition from low to high 40nsec after falling edge of 0. Its width will be 2.1 usec max, 1.54usec min. It will go from high to low 100nsec before falling edge of 0.
5. Reference t_pd (High) is 0.
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 0 Right	
1	Color 1 Right	
2	Color 2 Right	
3	Color 3 Right	
4	Color 0 Left	
5	Color 1 Left	
6	Color 2 Left	
7	Color 3 Left	
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	SW Bank 1
12	Tone B	SW Bank 2
13	Tone C	SW BANK 3
14	Tremolo	SW BANK 4
15	Tone C Volume	SW BANK 5
16	Tone A,B Volume	SW BANK 6
17	Noise Volume	SW BANK 7
18	Sound Block TX	
19		
1A		
1B		
1C		Pot 0
1D		Pot 1
1E		Pot 2
1F		Pot 3
20		
21		
22		
23		
24		
.		
.		
2F		

End of 'Nutting' Manual - Continues with ROM Source

This page intentionally left blank for double-sided print purposes

```
                ;  BALLY.H - Version 2.2
                ;  Bally Astrocade Equates and Macros Header File
                ;
                ;  Retyped and proofread by Adam Trionfo and Lance F. Squire
                ;  Version 1.0 - January 17, 2002
                ;  Version 2.2 - February 6, 2002
                ;    This ROM file contains the equates and macros that the
                ;  Bally ROM requires for assembly (the header file is
                ;  available separately too).  This file has been written to
                ;  assemble with ZMAC 1.3 (a little known, freely distribut-
                ;  able Z-80 assembler (with C source), that has a 25-year
                ;  history. ZMAC can be compiled under just about any O.S.
                ;  in existence, so try it out.  This file will probably
                ;  require changes to be assembled under other assemblers.
                ;
                ;  To assemble your Z-80 source code using ZMAC:
                ;
                ;  zmac -d -o <outfile> -x <listfile> <filename>
                ;
                ;  For example, assemble this Astrocade Z-80 ROM file:
                ;
                ;  zmac -d -o BallyROM.bin -x BallyROM.lst BallyROM.asm
                ;
                ;    Currently the Listing file is full of 'Undeclared'
                ;  errors.  When all of the source is typed-in, these will
                ;  vanish. I'm leaving the others until all the source is
                ;  re-typed.
                ;
```

```
  30:               ; **************************
  31:               ; * HOME VIDEO GAME EQUATES *
  32:               ; **************************
  33:               ;
  34:               ;  ASSEMBLY CONTROL
  35:               ;
  36: 0001              XPNDON  EQU    1          ; ** SET TO 1 WHEN HARDWARE EXP
  37: 0001              NWHDWR  EQU    1          ; ** SET TO 1 WHEN NEW HARDWARE
  38:               ;
  39:               ;  GENERAL GOODIES
  40: 4000              NORMEM  EQU    4000H
  41: 2000              FIRSTC  EQU    2000H      ; FIRST ADDRESS IN CARTRIDGE
  42: 0000              SCREEN  EQU    0
  43: 0028              BYTEPL  EQU    40         ; BYTES PER LINE
  44: 00A0              BITSPL  EQU    160        ; BITS PER LINE
  45:               ;  STUFF IN SYSTEM DOPE VECTOR
  46: 0200              STIMER  EQU    200H       ; SECONDS AND GAME TIME,MUSIC
  47: 0203              CTIMER  EQU    203H       ; CUSTOM TIMERS
  48: 0206              FNTSYS  EQU    206H       ; SYSTEM FONT DESCRIPTOR
  49: 020D              FNTSML  EQU    20DH       ; SMALL FONT DESCRIPTOR
  50: 0214              ALKEYS  EQU    214H       ; KEYMASK OF ALL KEYS
  51: 0218              MENUST  EQU    218H       ; HEAD OF ONBOARD MENU
  52: 021E              MXSCR   EQU    21EH       ; ADDRESS OF 'MAX SCORE'
  53: 0228              NOPLAY  EQU    228H       ; ADDRESS OF '# OF PLAYERS'
  54: 0235              NOGAME  EQU    235H       ; ADDRESS OF '# OF GAMES'
  55:               ;  BITS IN PROCESSOR FLAG BYTE
  56: 0007              PSWSGN  EQU    7          ; SIGN BIT
  57: 0006              PSWZRO  EQU    6          ; ZERO BIT
  58: 0002              PSWPV   EQU    2          ; PARITY        OVERFLOW
  59: 0000              PSWCY   EQU    0          ; CARRY
  60:               ;  BITS IN GAME STATUS BYTE
  61: 0000              GSBTIM  EQU    0
  62: 0001              GSBSCR  EQU    1
  63: 0007              GSBEND  EQU    7
  64:               ;  STANDARD VECTOR DISPLACEMENTS AND BITS
  65: 0000              VBMR    EQU    0          ; MAGIC REGISTER
  66: 0001              VBSTAT  EQU    1          ; STATUS
  67: 0002              VBTIMB  EQU    2          ; TIME BASE
  68: 0003              VBDXL   EQU    3          ; DELTA X LO
  69: 0004              VBDXH   EQU    4          ; DELTA X HI
  70: 0005              VBXL    EQU    5          ; X COORD LO
  71: 0006              VBXH    EQU    6          ; X COORD HI
  72: 0007              VBXCHK  EQU    7          ; X CHECK FLAGS
  73: 0008              VBDYL   EQU    8          ; DELTA Y LO
  74: 0009              VBDYH   EQU    09H        ; DELTA Y HI
  75: 000A              VBYL    EQU    0AH        ; Y COORD LO
  76: 000B              VBYH    EQU    0BH        ; Y COORD HI
  77: 000C              VBYCHK  EQU    0CH        ; Y CHECK FLAGS
  78: 000D              VBOAL   EQU    0DH        ; OLD ADDRESS OF L.O.
  79: 000E              VBOAH   EQU    0EH        ; OLD ADDRESS OF H.O.
  80:               ;  DISPLACEMENTS FROM START OF COORDINATE AREA
  81: 0000              VBDCL   EQU    0          ; LO DELTA
  82: 0001              VBDCH   EQU    1          ; HI DELTA
  83: 0002              VBCL    EQU    2          ; LO COORD
  84: 0003              VBCH    EQU    3          ; HI COORD
  85: 0004              VBCCHK  EQU    4          ; CHECK BITS
```

```
 86:                ;  BITS IN STATUS BYTE
 87: 0007               VBSACT  EQU    7          ; VECTOR ACTIVE STATUS
 88: 0006               VBBLNK  EQU    6          ; BLANK STATUS
 89:                ;  BITS IN CHECK BIT MASK
 90: 0000               VBCLMT  EQU    0          ; DO LIMIT CHECKING
 91: 0001               VBCREV  EQU    1          ; REVERSE DELTA ON LIMIT ATTAIN
 92: 0003               VBCLAT  EQU    3          ; COORDINATE IS AT LIMIT
 93:                ;  FONT TABLE DISPLACEMENTS FOR NEW CHARACTER DISPLAY ROU
 94: 0000               FTBASE  EQU    0          ; BASE CHARACTER
 95: 0001               FTFSX   EQU    1          ; X FRAME SIZE
 96: 0002               FTFSY   EQU    2          ; Y FRAME SIZE
 97: 0003               FTBYTE  EQU    3          ; X SIZE FOR CHAR IN BYTES
 98: 0004               FTYSIZ  EQU    4          ; Y SIZE IN BITS
 99: 0005               FTPTL   EQU    5          ; PATTERN TABLE ADDRESS LO
100: 0006               FTPTH   EQU    6          ; PATTERN TABLE ADDRESS HI
101:               ;  BITS FOR MAGIC REGISTER     WRITE OPTION BYTE
102: 0006               MRFLOP  EQU    6          ; WRITE WITH FLOP
103: 0005               MRXOR   EQU    5          ; WRITE WITH EXCLUSIVE OR
104: 0004               MROR    EQU    4          ; WRITE WITH OR
105: 0003               MRXPND  EQU    3          ; WRITE WITH EXPAND
106: 0002               MRROT   EQU    2          ; WRITE WITH ROTATE
107: 0003               MRSHFT  EQU    03H        ; MASK OF SHIFT AMOUNT
108:               ;  BITS OF CONTROL HANDLE INPUT PORT
109: 0004               CHTRIG  EQU    4          ; TRIGGER
110: 0003               CHRIGH  EQU    3          ; JOYSTICK RIGHT
111: 0002               CHLEFT  EQU    2          ; JOYSTICK LEFT
112: 0001               CHDOWN  EQU    1          ; DOWN
113: 0000               CHUP    EQU    0          ; UP
114:               ; CONTEXT BLOCK REGISTER DISPLACEMENTS
115: 0000               CBIYL   EQU    0          ; IY
116: 0001               CBIYH   EQU    1
117: 0002               CBIXL   EQU    2          ; IX
118: 0003               CBIXH   EQU    3
119: 0004               CBE     EQU    4          ; DE
120: 0005               CBD     EQU    5
121: 0006               CBC     EQU    6          ; BC
122: 0007               CBB     EQU    7
123: 0008               CBFLAG  EQU    8          ; AF
124: 0009               CBA     EQU    9
125: 000A               CBL     EQU    0AH        ; HL
126: 000B               CBH     EQU    0BH
127:               ;  SENTRY RETURN CODES EQUATES:
128: 0000               SNUL    EQU    0          ; NOTHING HAPPENED
129: 0001               SCT0    EQU    1          ; COUNTER-TIMER 1 THRU 8
130: 0002               SCT1    EQU    2
131: 0003               SCT2    EQU    3
132: 0004               SCT3    EQU    4
133: 0005               SCT4    EQU    5
134: 0006               SCT5    EQU    6
135: 0007               SCT6    EQU    7
136: 0008               SCT7    EQU    8
137: 0009               SF0     EQU    9          ; FLAG BIT 0
138: 000A               SF1     EQU    0AH
139: 000B               SF2     EQU    0BH
140: 000C               SF3     EQU    0CH
141: 000D               SF4     EQU    0DH
```

```
 142: 000E              SF5     EQU    0EH
 143: 000F              SF6     EQU    0FH
 144: 0010              SF7     EQU    10H
 145: 0011              SSEC    EQU    11H         ; SECONDS TIMER HAS COUNTED DOWN
 146: 0013              SKYD    EQU    13H         ; KEY IS DOWN
 147: 0012              SKYU    EQU    12H         ; YES IS UP
 148: 001C              SP0     EQU    1CH         ; POT IS 0
 149: 001D              SP1     EQU    1DH         ; POT IS 1
 150: 001E              SP2     EQU    1EH         ; POT IS 2
 151: 001F              SP3     EQU    1FH         ; POT IS 3
 152: 0014              ST0     EQU    14H         ; TRIGGER 0
 153: 0015              SJ0     EQU    15H         ; JOYSTICK 0
 154: 0016              ST1     EQU    16H         ; SIMILARLY FOR 1-3
 155: 0017              SJ1     EQU    17H
 156: 0018              ST2     EQU    18H
 157: 0019              SJ2     EQU    19H
 158: 001A              ST3     EQU    1AH
 159: 001B              SJ3     EQU    1BH
```

```
                ; ********************************
                ; * HOME VIDEO GAME PORT EQUATES *
                ; ********************************
                ; OUTPUT PORTS FOR VIRTUAL COLOR
0000               COL0R   EQU    0          ; COLOR 0 RIGHT
0001               COL1R   EQU    1          ; COLOR 1 RIGHT
0002               COL2R   EQU    2          ; COLOR 2 RIGHT
0003               COL3R   EQU    3          ; COLOR 3 RIGHT
0004               COL0L   EQU    4          ; COLOR 0 LEFT
0005               COL1L   EQU    5          ; COLOR 1 LEFT
0006               COL2L   EQU    6          ; COLOR 2 LEFT
0007               COL3L   EQU    7          ; COLOR 3 LEFT
000B               COLBX   EQU    0BH        ; COLOR BLOCK OUTPUT PORT
0009               HORCB   EQU    9          ; HORIZONTAL COLOR BOUNDARY
000A               VERBL   EQU    0AH        ; VERTICAL BLANKING LINE
                ; OUTPUT PORTS FOR MUSIC AND SOUNDS
0010               TONMO   EQU    10H        ; TONE MASTER OSCILLATOR
0011               TONEA   EQU    11H        ; TONE A OSC.
0012               TONEB   EQU    12H        ; TONE B OSC.
0013               TONEC   EQU    13H        ; TONE C OSC.
0014               VIBRA   EQU    14H        ; VIBRATO
0016               VOLAB   EQU    16H        ; TONES A,B VOLUME
0015               VOLC    EQU    15H        ; TONE C VOLUME
0017               VOLN    EQU    17H        ; NOISE VOLUME
0018               SNDBX   EQU    18H        ; SOUND BLOCK OUTPUT PORT
                ; INTERRUPT AND CONTROL OUTPUT PORTS
000D               INFBK   EQU    0DH        ; INTERRUPT FEEDBACK
000E               INMOD   EQU    0EH        ; INTERRUPT MODE
000F               INLIN   EQU    0FH        ; INTERRUPT LINE
0008               CONCM   EQU    8          ; CONSUMER      COMMERCIAL
000C               MAGIC   EQU    0CH        ; MAGIC REGISTER
0019               XPAND   EQU    19H        ; EXPANDER PIXEL DEFINITION PORT
                ; INTERRUPT AND INTERCEPT INPUT PORTS
0008               INTST   EQU    8          ; INTERCEPT STATUS
000E               VERAF   EQU    0EH        ; VERTICAL ADDRESS FEEDBACK
000F               HORAF   EQU    0FH        ; HORIZONTAL ADDRESS FEEDBACK
                ; HAND CONTROL INPUT PORTS
0010               SW0     EQU    10H        ; PLAYER 0 HAND CONTROL
0011               SW1     EQU    11H        ; PLAYER 1 HAND CONTROL
0012               SW2     EQU    12H        ; PLAYER 2 HAND CONTROL
0013               SW3     EQU    13H        ; PLAYER 3 HAND CONTROL
001C               POT0    EQU    1CH        ; PLAYER 0 POT
001D               POT1    EQU    1DH        ; PLAYER 1 POT
001E               POT2    EQU    1EH        ; PLAYER 2 POT
001F               POT3    EQU    1FH        ; PLAYER 3 POT
                ; KEYBOARD INPUT PORTS
0014               KEY0    EQU    14H        ; KEYBOARD COLUMN 0
0015               KEY1    EQU    15H        ; KEYBOARD COLUMN 1
0016               KEY2    EQU    16H        ; KEYBOARD COLUMN 2
0017               KEY3    EQU    17H        ; KEYBOARD COLUMN 3
```

```
                ; ************************************
                ; * HOME VIDEO GAME SYSTEM CALL INDEXES *
                ; ************************************
                ; USER PROGRAM INTERFACE
0000                UPISTR  EQU     0
0000                INTPC   EQU     UPISTR      ; INTERPRET WITH CONTEXT CREATE
0002                XINTC   EQU     INTPC+2     ; EXIT INTERPRETER WITH CONTEXT
0004                RCALL   EQU     XINTC+2     ; CALL ASM LANGUAGE SUBROUTINE
0006                MCALL   EQU     RCALL+2     ; CALL INTERPRETER SUBROUTINE
0008                MRET    EQU     MCALL+2     ; RETURN FROM INTERPRETER SUBRO
000A                MJUMP   EQU     MRET+2      ; MACRO JUMP
000C                SUCK    EQU     MJUMP+2     ; SUCK INLINE ARGS INTO CB
                ; SCHEDULER ROUTINES
000C                SCHEDR  EQU     SUCK
000E                ACTINT  EQU     SCHEDR+2    ; SET SUB TIMER
0010                DECCTS  EQU     ACTINT+2    ; DEC CT'S UNDER MASK
                ; MUSIC AND SOUNDS
0012                MUZAK   EQU     DECCTS+2
0012                BMUSIC  EQU     MUZAK       ; BEGIN PLAYING MUSIC
0014                EMUSIC  EQU     BMUSIC+2    ; STOP PLAYING MUSIC
                ; SCREEN HANDLER ROUTINES
0016                SCRSTR  EQU     EMUSIC+2
0016                SETOUT  EQU     SCRSTR      ; SET SCREEN SIZE
0018                COLSET  EQU     SETOUT+2    ; SET COLORS
001A                FILL    EQU     COLSET+2    ; FILL MEMORY WITH DAT
001C                RECTAN  EQU     FILL+2      ; PAINT RECTANGLE
001E                VWRITR  EQU     RECTAN+2    ; WRITE RELATIVE FROM VECTOR
0020                WRITR   EQU     VWRITR+2    ; WRITE RELATIVE
0022                WRITP   EQU     WRITR+2     ; WRITE WITH PATTERN SIZE LOOKUP
0024                WRIT    EQU     WRITP+2     ; WRITE WITH SIZES PROVIDED
0026                WRITA   EQU     WRIT+2      ; WRITE ABSOLUTE
0028                VBLANK  EQU     WRITA+2     ; BLANK AREA FROM VECTOR
002A                BLANK   EQU     VBLANK+2    ; BLANK AREA
002C                SAVE    EQU     BLANK+2     ; SAVE AREA
002E                RESTOR  EQU     SAVE+2      ; RESTORE AREA
0030                SCROLL  EQU     RESTOR+2    ; SCROLL AREA OF SCREEN
                ;
0032                CHRDIS  EQU     SCROLL+2    ; NEW DISPLAY CHARACTER
0034                STRDIS  EQU     CHRDIS+2    ; NEW DISPLAY STRING
0036                DISNUM  EQU     STRDIS+2    ; DISPLAY NUMBER
                ;
0038                RELABS  EQU     DISNUM+2    ; RELATIVE TO ABSOLUTE CONVERSI
003A                RELAB1  EQU     RELABS+2    ; NONMAGIC RELABS
003C                VECTC   EQU     RELAB1+2    ; VECTOR SINGLE COORDINATE
003E                VECT    EQU     VECTC+2     ; VECTOR COORDINATE PAIR
                ; HUMAN INTERFACE ROUTINES
0040                HUMANR  EQU     VECT +2
0040                KCTASC  EQU     HUMANR      ; KEY CODE TO ASCII
0042                SENTRY  EQU     KCTASC+2    ; SENSE TRANSITION
0044                DOIT    EQU     SENTRY+2    ; BRANCH TO TRANSITION HANDLER
0046                DOITB   EQU     DOIT+2      ; USE B INSTEAD OF A
0048                PIZBRK  EQU     DOITB+2     ; TAKE A BREAK
004A                MENU    EQU     PIZBRK+2    ; DISPLAY A MENU
004C                GETPAR  EQU     MENU+2      ; GET GAME PARAMENTER FROM USER
004E                GETNUM  EQU     GETPAR+2    ; GET NUMBER FROM USER
0050                PAWS    EQU     GETNUM+2    ; PAUSE
```

```
 268: 0052              DISTIM  EQU    PAWS+2     ; DISPLAY TIME
 269: 0054              INCSCR  EQU    DISTIM+2   ; INC SCORE
 270:                ; MATH ROUTINES
 271: 0056              MATH    EQU    INCSCR+2
 272: 0056              INDEXN  EQU    MATH       ; INDEX NIBBLE
 273: 0058              STOREN  EQU    INDEXN+2   ;
 274: 005A              INDEXW  EQU    STOREN+2   ; INDEX WORD
 275: 005C              INDEXB  EQU    INDEXW+2   ; INDEX BYTE
 276: 005E              MOVE    EQU    INDEXB+2   ; BLOCK TRANSFER
 277: 0060              SHIFTU  EQU    MOVE+2     ; SHIFT UP A DIGIT
 278: 0062              BCDADD  EQU    SHIFTU+2   ; BCD ADD
 279: 0064              BCDSUB  EQU    BCDADD+2   ; BCD SUBTRACT
 280: 0066              BCDMUL  EQU    BCDSUB+2   ; BCD MULTIPLY
 281: 0068              BCDDIV  EQU    BCDMUL+2   ; BCD DIVIDE
 282: 006A              BCDCHS  EQU    BCDDIV+2   ; BCD CHANGE SIGN
 283: 006C              BCDNEG  EQU    BCDCHS+2   ; BCD NEGATE
 284: 006E              DADD    EQU    BCDNEG+2   ; DECIMAL ADD
 285: 0070              DSMG    EQU    DADD+2     ; CONVERT TO SIGN MAGNITUDE
 286: 0072              DABS    EQU    DSMG+2     ; DECIMAL ABSOLUTE VALUE
 287: 0074              NEGT    EQU    DABS+2     ; NEGATE
 288: 0076              RANGED  EQU    NEGT+2     ; RANGED RANDOM NUMBER
 289: 0078              QUIT    EQU    RANGED+2   ; QUIT CASSETTE EXECUTION
 290: 007A              SETB    EQU    QUIT+2     ; SET BYTE
 291: 007C              SETW    EQU    SETB+2     ; SET WORD
 292: 007E              MSKTD   EQU    SETW+2     ; MASK TO DELTAS
```

```
                ; **********
                ; * MACROS *
                ; **********
                ; MACROS TO DEFINE PATTERNS
DEF2    MACRO  AA, AB
        DEFB   AA
        DEFB   AB
        ENDM
DEF3    MACRO  BA, BB, BCC
        DEFB   BA
        DEFB   BB
        DEFB   BCC  ; 'BC' reserved, so used 'BCC'
        ENDM
DEF4    MACRO  CA, CB, CC, CD
        DEFB   CA
        DEFB   CB
        DEFB   CC
        DEFB   CD
        ENDM
DEF5    MACRO  DA, DBB, DC, DD, DEE
        DEFB   DA
        DEFB   DBB  ; 'DB' reserved, so used 'DBB'
        DEFB   DC
        DEFB   DD
        DEFB   DEE  ; 'DE' reserved, so used 'DEE'
        ENDM
DEF6    MACRO  EA, EB, EC, ED, EE, EF
        DEFB   EA
        DEFB   EB
        DEFB   EC
        DEFB   ED
        DEFB   EE
        DEFB   EF
        ENDM
DEF8    MACRO  GA, GB, GC, GD, GEE, GF, GG, GH
        DEFB   GA
        DEFB   GB
        DEFB   GC
        DEFB   GD
        DEFB   GEE  ; 'GE' reserved, so used 'GEE'
        DEFB   GF
        DEFB   GG
        DEFB   GH
        ENDM
  ; MACROS TO COMPUTE CONSTANT SCREEN ADDRESSES
XYRELL  MACRO  p1, p2, p3  ; RELATIVE LOAD
        LD     p1,. RES. (p3). SHL. 8+(p2)
        ENDM
 ; MACRO TO GENERATE SYSTEM CALL
SYSTEM  MACRO  NUMBA
        RST    56
        DEFB   NUMBA
        IF     NUMBA = INTPC
INTPCC   DEFL   1
        ENDIF
        ENDM
```

```
 350:                ; MACRO TO GENERATE SYSTEM CALL WITH SUCK OPTION ON
 351:               SYSSUK  MACRO  UMBA
 352:                       RST    56
 353:                       DEFB   UMBA+1
 354:                       IF     UMBA = INTPC
 355:               INTPCC  DEFL   1
 356:                       ENDIF
 357:                       ENDM
 358:                ; MACROS TO GENERATE MACRO INSTRUCTION CALLS
 359:                ; FILL SCREEN WITH CONSTANT DATA (was 'FILL?')
 360:               FILLq   MACRO  START, NBYTES, DATA
 361:                       DEFB   FILL+1
 362:                       DEFW   START
 363:                       DEFW   NBYTES
 364:                       DEFB   DATA
 365:                       ENDM
 366:                ; EXIT INTERPRETER WITH CONTEXT RESTORE
 367:               EXIT    MACRO
 368:                       DEFB   XINTC
 369:               INTPCC  DEFL   0
 370:                       ENDM
 371:                ; INTERPRET WITH INLINE SUCK
 372:               DO      MACRO  CID
 373:                       DEFB   CID+1
 374:                       ENDM
 375:                ; INTERPRET WITHOUT INLINE SUCK
 376:               DONT    MACRO  CID
 377:                       DEFB   CID
 378:                       ENDM
 379:                ; MACRO CALL FROM DOIT TABLE
 380: 00C0                ENDx    EQU    0C0H
 381:               MC      MACRO  AA, BB, EE
 382:                       DEFB   AA+80H
 383:                       DEFW   BB
 384:                       IF     EE
 385:                       DEFB   EE
 386:                       ENDIF
 387:                       ENDM
 388:                ; REAL CALL FROM DOIT TABLE
 389:               RC      MACRO  AA, BB, EE
 390:                       DEFB   AA+40H
 391:                       DEFW   BB
 392:                       IF     EE
 393:                       DEFB   EE
 394:                       ENDIF
 395:                       ENDM
 396:                ; REAL JUMP FROM DOIT TABLE
 397:               JMPd    MACRO  AA, BB, EE
 398:                       DEFB   AA
 399:                       DEFW   BB
 400:                       IF     EE
 401:                       DEFB   EE
 402:                       ENDIF
 403:                       ENDM
 404:                ; DISPLAY A STRING
 405:               TEXTD   MACRO  AA, BB, CC, DD
```

```
                        DEFB   STRDIS+1
                        DEFB   BB
                        DEFB   CC
                        DEFB   DD
                        DEFW   AA
                        ENDM



                  ;**************
                  ; MUSIC MACROS
                  ; NOTE DURATION, FREQ(S)
                NOTE1   MACRO  DUR, N1
                        DEFB   (DUR)&(7FH)
                        DEFB   N1
                        ENDM
                NOTE2   MACRO  DUR, N1, N2
                        DEFB   (DUR)&(7FH)
                        DEFB   N1
                        DEFB   N2
                        ENDM
                NOTE3   MACRO  DUR, N1, N2, N3
                        DEFB   DUR
                        DEFB   N1
                        DEFB   N2
                        DEFB   N3
                        ENDM
                NOTE4   MACRO  DUR, N1, N2, N3, N4
                        DEFB   DUR
                        DEFB   N1
                        DEFB   N2
                        DEFB   N3
                        DEFB   N4
                        ENDM
                NOTE5   MACRO  DUR, N1, N2, N3, N4, N5
                        DEFB   DUR
                        DEFB   N1
                        DEFB   N2
                        DEFB   N3
                        DEFB   N4
                        DEFB   N5
                        ENDM
                MASTER  MACRO  OFFSET
                        DEFB   80H
                        DEFB   OFFSET
                        ENDM
                  ; STUFF OUTPUT PORT#, DATA  OR
                  ; OUTPUT SNDBX, DATA10, D11,..., DATA17
                OUTPUT  MACRO  PORT, D0, D1, D2, D3, D4, D5, D6, D7
                        IF     .NOT. (PORT=18H)
                        DEFB   80H+((PORT)&(7FH))
                        DEFB   D0
                        ENDIF
                        IF     PORT=18H
                        DEFB   88H
```

```
                        DEF8   D7, D6 ,D5, D4, D3, D2, D1, D0
                        ENDIF
                        ENDM
                  ; SET VOICE BYTE
                  ; THE FORMAT OF THE VOICE BYTE IS
                  ; *I*A*I*B*I*C*V*N
                  ; WHERE N = LOAD NOISE WITH DATA AT PC AND INC PC
                  ; V = LOAD VIBRATO AND INC PC
                  ; I = INC PC
                  ; A,B,C = LOAD TONE A,B,C WITH DATA AT PC
                VOICEM  MACRO  MASK  ;  'VOICES' TO 'VOICEM'
                        DEFB   90H
                        DEFB   MASK
                        ENDM
                ; PUSH NUMBER ONTO STACK
                PUSHN   MACRO  NUMB
                        DEFB   0A0H+((NUMB-1). AND. 0FH)
                        ENDM
                  ; SET VOLUMES
                VOLUME  MACRO  P1, P2
                        DEFB   0B0H
                        DEFB   P1
                        DEFB   P2
                        ENDM
                ; CALL RELATIVE 0-15 BEYOND SELF+1
                CREL    MACRO  BY
                        DEFB   0D0H+(BY.AND.0FH)
                        ENDM
                ; DEC STACK TOP AND JNZ
                DSJNZ   MACRO  ADD_IT
                        DEFB   0C0H
                        DEFW   ADD_IT
                        ENDM
                ; FLIP LEGATO  STACATO
                LEGSTA  MACRO
                        DEFB   0E0H
                        ENDM
                REST    MACRO  TIME
                        DEFB   0E1H
                        DEFB   TIME
                        ENDM
                QUIET   MACRO
                        DEFB   0F0H
                        ENDM
                  ; *****************
                  ; * MUSIC EQUATES *
                  ; *****************
                  ; NOTE VALUES
00FD                 G0      EQU    253
00EE                 GS0     EQU    238
00E1                 A0      EQU    225
00D4                 AS0     EQU    212
00C8                 B0      EQU    200
00BD                 C1      EQU    189
00B2                 CS1     EQU    178
00A8                 D1      EQU    168
```

```
 515: 009F              DS1     EQU    159
 516: 0096              E1      EQU    150
 517: 008D              F1      EQU    141
 518: 0085              FS1     EQU    133
 519: 007E              G1      EQU    126
 520: 0077              GS1     EQU    119
 521: 0070              A1      EQU    112
 522: 006A              AS1     EQU    106
 523: 0064              B1      EQU    100
 524: 005E              C2      EQU    94
 525: 0059              CS2     EQU    89
 526: 0054              D2      EQU    84
 527: 004F              DS2     EQU    79
 528: 004A              E2      EQU    74
 529: 0046              F2      EQU    70
 530: 0042              FS2     EQU    66
 531: 003E              G2      EQU    62
 532: 003B              GS2     EQU    59
 533: 0037              A2      EQU    55
 534: 0034              AS2     EQU    52
 535: 0031              B2      EQU    49
 536: 002E              C3      EQU    46
 537: 002C              CS3     EQU    44
 538: 0029              D3      EQU    41
 539: 0027              DS3     EQU    39
 540: 0025              E3      EQU    37
 541: 0022              F3      EQU    34
 542: 0020              FS3     EQU    32
 543: 001F              G3      EQU    31
 544: 001D              GS3     EQU    29
 545: 001B              A3      EQU    27
 546: 001A              AS3     EQU    26
 547: 0018              B3      EQU    24
 548: 0017              C4      EQU    23
 549: 0015              CS4     EQU    21
 550: 0014              D4      EQU    20
 551: 0013              DS4     EQU    19
 552: 0012              E4      EQU    18
 553: 0011              F4      EQU    17
 554: 0010              FS4     EQU    16
 555: 000F              G4      EQU    15
 556: 000E              GS4     EQU    14
 557: 000D              A4      EQU    13
 558: 000B              C5      EQU    11
 559: 000A              CS5     EQU    10
 560: 0009              DS5     EQU    9
 561: 0008              F5      EQU    8
 562: 0007              G5      EQU    7
 563: 0006              A5      EQU    6
 564: 0005              C6      EQU    5
 565: 0004              DS6     EQU    4
 566: 0003              G6      EQU    3
 567: 0002              C7      EQU    2
 568: 0001              G7      EQU    1
 569: 0000              G8      EQU    0
 570:               ; MASTER OSCILATOR OFFSETS
```

```
 571: 00FE              OB0     EQU    254
 572: 00F1              OC0     EQU    241
 573: 00D6              OD1     EQU    214
 574: 00BF              OE1     EQU    191
 575: 00B4              OF1     EQU    180
 576: 00A0              OG1     EQU    160
 577: 008F              OA1     EQU    143
 578: 0047              OA2     EQU    71
 579: 0023              OA3     EQU    35
 580: 0011              OA4     EQU    17
 581: 0008              OA5     EQU    8
```

```
                ; ***************************
                ; * SYSTEM RAM MEMORY CELLS *
                ; ***************************
0FFF               WASTE   EQU    0FFFH
0FFF               WASTER  EQU    WASTE
                ;
                ; THE FOLLOWING ORG SHOULD BE SET TO THE VALUE OF
                ; THE TAG 'SYSRAM', THIS WILL CAUSE SYSTEM RAM
                ; TO RESIDE AT THE HIGEST POSSIBLE ADDRESS
                ;
              ;        ORG    4FC8H
              ;        DEFS   6          ; GOT SOME LEFT STILL
4FCE               BEGRAM  EQU    4FCEH
                ; USED BY MUSIC PROCESSOR
4FCE               MUZPC   EQU    4FCEH      ; MUSIC PROGRAM COUNTER
4FD0               MUZSP   EQU    4FD0H      ; MUSIC STACK POINTER
4FD2               PVOLAB  EQU    4FD2H      ; PRESET VOLUME FOR TONES A AND B
4FD3               PVOLMC  EQU    4FD3H      ; PRESET VOLUME FOR MASTER OSC
4FD4               VOICES  EQU    4FD4H      ; MUSIC VOICES
                ; COUNTER TIMERS (USED BY DECCTS,ACTINT,CTIMER)
4FD5               CT0     EQU    4FD5H      ; COUNTER TIMER 0
4FD6               CT1     EQU    4FD6H      ; 1
4FD7               CT2     EQU    4FD7H      ; 2
4FD8               CT3     EQU    4FD8H      ; 3
4FD9               CT4     EQU    4FD9H      ; 4
4FDA               CT5     EQU    4FDAH      ; 5
4FDB               CT6     EQU    4FDBH      ; 6
4FDC               CT7     EQU    4FDCH      ; 7
                ;USED BY SENTRY TO TRACK CONTROLS
4FDD               CNT     EQU    4FDDH      ; COUNTER UPDATE&NUMBER TRACKING
4FDE               SEMI4S  EQU    4FDEH      ; FLAG BITS
4FDF               OPOT0   EQU    4FDFH      ; POT 0 TRACKING
4FE0               OPOT1   EQU    4FE0H      ; POT 1 TRACKING
4FE1               OPOT2   EQU    4FE1H      ; POT 2 TRACKING
4FE2               OPOT3   EQU    4FE2H      ; POT 3 TRACKING
4FE3               KEYSEX  EQU    4FE3H      ; KEYBOARD TRACKING BYTE
4FE4               OSW0    EQU    4FE4H      ; SWITCH 0 TRACKING
4FE5               OSW1    EQU    4FE5H      ; SWITCH 1 TRACKING
4FE6               OSW2    EQU    4FE6H      ; SWITCH 2 TRACKING
4FE7               OSW3    EQU    4FE7H      ; SWITCH 3 TRACKING
4FE8               COLLST  EQU    4FE8H      ; COLOR LIST ADDRESS FOR P.B.A
                ; USED BY STIMER
4FEA               DURAT   EQU    4FEAH      ; NOTE DURATION
4FEB               TMR60   EQU    4FEBH      ; SIXTIETHS OF SEC
4FEC               TIMOUT  EQU    4FECH      ; BLAKOUT TIMER
4FED               GTSECS  EQU    4FEDH      ; GAME TIME SECONDS
4FEE               GTMINS  EQU    4FEEH      ; GAME TIME MINUTES
                ; USED BY MENU
4FEF               RANSHT  EQU    4FEFH      ; RANDOM NUMBER SHIFT REGISTER
4FF3               NUMPLY  EQU    4FF3H      ; NUMBER OF PLAYERS
4FF4               ENDSCR  EQU    4FF4H      ; SCORE TO 'PLAY TO'
4FF7               MRLOCK  EQU    4FF7H      ; MAGIC REGISTER LOCK OUT FLAG
4FF8               GAMSTB  EQU    4FF8H      ; GAME STATUS BYTE
4FF9               PRIOR   EQU    4FF9H      ; MUSIC PROTECT FLAG
4FFA               SENFLG  EQU    4FFAH      ; SENTRY CONTROL SEIZURE FLAG
4FFB               UMARGT  EQU    4FFBH
```

```
Feb 08 16:23 2002 bally.h      Page 15


 639: 4FFD              USERTB  EQU    4FFDH
 640: 9FCD              SYSRAM  EQU    (5000H-($-BEGRAM+1))
**** bally.h ****



Statistics:

   426      symbols
     0      bytes

     0      macro calls
  2744      macro bytes
     0      invented symbols
```

Feb 08 16:23 2002 ** Symbol Table **   Page 16

Symbol Table:

```
a0              =  e1+    fntsml          = 20d+    ranged          =  76
a1              =  70+    fntsys          = 206+    ransht          =4fef+
a2              =  37+    fs1             =  85+    rc                521+
a3              =  1b+    fs2             =  42+    rcall           =   4
a4              =   d+    fs3             =  20+    rectan          =  1c
a5              =   6+    fs4             =  10+    relab1          =  3a
actint          =   e     ftbase          =   0+    relabs          =  38
alkeys          = 214+    ftbyte          =   3+    rest              a60+
as0             =  d4+    ftfsx           =   1+    restor          =  2e
as1             =  6a+    ftfsy           =   2+    save            =  2c
as2             =  34+    ftpth           =   6+    schedr          =   c
as3             =  1a+    ftptl           =   5+    screen          =   0+
b0              =  c8+    ftysiz          =   4+    scroll          =  30
b1              =  64+    g0              =  fd+    scrstr          =  16
b2              =  31+    g1              =  7e+    sct0            =   1+
b3              =  18+    g2              =  3e+    sct1            =   2+
bcdadd          =  62     g3              =  1f+    sct2            =   3+
bcdchs          =  6a     g4              =   f+    sct3            =   4+
bcddiv          =  68     g5              =   7+    sct4            =   5+
bcdmul          =  66     g6              =   3+    sct5            =   6+
bcdneg          =  6c     g7              =   1+    sct6            =   7+
bcdsub          =  64     g8              =   0+    sct7            =   8+
begram          =4fce     gamstb          =4ff8+    semi4s          =4fde+
bitspl          =  a0+    getnum          =  4e     senflg          =4ffa+
blank           =  2a     getpar          =  4c     sentry          =  42
bmusic          =  12     gs0             =  ee+    setb            =  7a
bytepl          =  28+    gs1             =  77+    setout          =  16
c1              =  bd+    gs2             =  3b+    setw            =  7c
c2              =  5e+    gs3             =  1d+    sf0             =   9+
c3              =  2e+    gs4             =   e+    sf1             =   a+
c4              =  17+    gsbend          =   7+    sf2             =   b+
c5              =   b+    gsbscr          =   1+    sf3             =   c+
c6              =   5+    gsbtim          =   0+    sf4             =   d+
c7              =   2+    gtmins          =4fee+    sf5             =   e+
cba             =   9+    gtsecs          =4fed+    sf6             =   f+
cbb             =   7+    horaf           =   f+    sf7             =  10+
cbc             =   6+    horcb           =   9+    shiftu          =  60
cbd             =   5+    humanr          =  40     sj0             =  15+
cbe             =   4+    incscr          =  54     sj1             =  17+
cbflag          =   8+    indexb          =  5c     sj2             =  19+
cbh             =   b+    indexn          =  56     sj3             =  1b+
cbixh           =   3+    indexw          =  5a     skyd            =  13+
cbixl           =   2+    infbk           =   d+    skyu            =  12+
cbiyh           =   1+    inlin           =   f+    sndbx           =  18+
cbiyl           =   0+    inmod           =   e+    snul            =   0+
cbl             =   a+    intpc           =   0     sp0             =  1c+
chdown          =   1+    intst           =   8+    sp1             =  1d+
chleft          =   2+    jmpd              58a+    sp2             =  1e+
chrdis          =  32     kctasc          =  40     sp3             =  1f+
chrigh          =   3+    key0            =  14+    ssec            =  11+
chtrig          =   4+    key1            =  15+    st0             =  14+
chup            =   0+    key2            =  16+    st1             =  16+
```

Feb 08 16:23 2002 ** Symbol Table **   Page 17

```
cnt             =4fdd+    key3            =  17+    st2             =  18+
col0l           =   4+    keysex          =4fe3+    st3             =  1a+
col0r           =   0+    legsta            a3d+    stimer          = 200+
col1l           =   5+    magic           =   c+    storen          =  58
col1r           =   1+    master            821+    strdis          =  34
col2l           =   6+    math            =  56     suck            =   c
col2r           =   2+    mc                4b8+    sw0             =  10+
col3l           =   7+    mcall           =   6     sw1             =  11+
col3r           =   3+    menu            =  4a     sw2             =  12+
colbx           =   b+    menust          = 218+    sw3             =  13+
collst          =4fe8+    mjump           =   a     sysram          =9fcd+
colset          =  18     move            =  5e     syssuk            376+
concm           =   8+    mret            =   8     system            309+
crel              9d8+    mrflop          =   6+    textd             5ef+
cs1             =  b2+    mrlock          =4ff7+    timout          =4fec+
cs2             =  59+    mror            =   4+    tmr60           =4feb+
cs3             =  2c+    mrrot           =   2+    tonea           =  11+
cs4             =  15+    mrshft          =   3+    toneb           =  12+
cs5             =   a+    mrxor           =   5+    tonec           =  13+
ct0             =4fd5+    mrxpnd          =   3+    tonmo           =  10+
ct1             =4fd6+    msktd           =  7e+    umargt          =4ffb+
ct2             =4fd7+    muzak           =  12     upistr          =   0
ct3             =4fd8+    muzpc           =4fce+    usertb          =4ffd+
ct4             =4fd9+    muzsp           =4fd0+    vbblnk          =   6+
ct5             =4fda+    mxscr           = 21e+    vbcchk          =   4+
ct6             =4fdb+    negt            =  74     vbch            =   3+
ct7             =4fdc+    nogame          = 235+    vbcl            =   2+
ctimer          = 203+    noplay          = 228+    vbclat          =   3+
d1              =  a8+    normem          =4000+    vbclmt          =   0+
d2              =  54+    note1             65e+    vbcrev          =   1+
d3              =  29+    note2             699+    vbdch           =   1+
d4              =  14+    note3             6e6+    vbdcl           =   0+
dabs            =  72     note4             73d+    vbdxh           =   4+
dadd            =  6e     note5             7a6+    vbdxl           =   3+
deccts          =  10     numply          =4ff3+    vbdyh           =   9+
def2               0+     nwhdwr          =   1+    vbdyl           =   8+
def3              33+     oa1             =  8f+    vblank          =  28
def4              97+     oa2             =  47+    vbmr            =   0+
def5              ee+     oa3             =  23+    vboah           =   e+
def6            195+      oa4             =  11+    vboal           =   d+
def8            210+      oa5             =   8+    vbsact          =   7+
disnum          =  36     ob0             =  fe+    vbstat          =   1+
distim          =  52     oc0             =  f1+    vbtimb          =   2+
do              474+      od1             =  d6+    vbxchk          =   7+
doit            =  44     oe1             =  bf+    vbxh            =   6+
doitb           =  46     of1             =  b4+    vbxl            =   5+
dont            497+      og1             =  a0+    vbychk          =   c+
ds1             =  9f+    opot0           =4fdf+    vbyh            =   b+
ds2             =  4f+    opot1           =4fe0+    vbyl            =   a+
ds3             =  27+    opot2           =4fe1+    vect            =  3e
ds4             =  13+    opot3           =4fe2+    vectc           =  3c
ds5             =   9+    osw0            =4fe4+    veraf           =   e+
ds6             =   4+    osw1            =4fe5+    verbl           =   a+
dsjnz             a08+    osw2            =4fe6+    vibra           =  14+
dsmg            =  70     osw3            =4fe7+    voicem            927+
durat           =4fea+    output            855+    voices          =4fd4+
```

```
Feb 08 16:23 2002 **  Symbol Table  **     Page 18


e1              =  96+    paws            =  50     volab           =  16+
e2              =  4a+    pizbrk          =  48     volc            =  15+
e3              =  25+    pot0            =  1c+    voln            =  17+
e4              =  12+    pot1            =  1d+    volume            991+
emusic          =  14     pot2            =  1e+    vwritr          =  1e
endscr          =4ff4+    pot3            =  1f+    waste           = fff
endx            =  c0+    prior           =4ff9+    waster          = fff+
exit              43f+    pswcy           =   0+    writ            =  24
f1              =  8d+    pswpv           =   2+    writa           =  26
f2              =  46+    pswsgn          =   7+    writp           =  22
f3              =  22+    pswzro          =   6+    writr           =  20
f4              =  11+    pushn             95b+    xintc           =   2
f5              =   8+    pvolab          =4fd2+    xpand           =  19+
fill            =  1a     pvolmc          =4fd3+    xpndon          =   1+
fillq             3e4+    quiet             a95+    xyrell            2cf+
firstc          =2000+    quit            =  78
```