Nutting Manual
Version 1.0 - Released Nov 04, 2000

This PDF document was originally scanned in by Frank Palazzo (Thank you!). It has been slightly changed-- the index has been re-typed and converted to PDF so that it looks cleaner courtesy of the *Bally Alley* newsletter. For other reprints and more information visit:

http://www.ballyalley.com

Corrections? Suggestions? Email Adam Trionfo at: ballyalley@hotmail.com

TABLE OF CONTENTS - SOFTWARE

TABLE OF CONTENTS - HARDWARE

LIST OF ILLUSTRATIONS

HOME VIDEO GAME SYSTEM

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

The major sub-sections of the system are:

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

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

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

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

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

USER PROGRAM INTERFACE

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

The basic rule for using the UPI is:

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

The mechanism for calling a system routine is:

```
RST     5
DEFB    (routine # + option)
```

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

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

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

```
        SYSTEM  (routine #)
```

(example)

```
        SYSTEM  FILL
```

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

```
        SYSSUK  (routine #)
```

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

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

Example:

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

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

All UPI routines are re-entrant.

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

SYSTEM ROUTINE CONVENTIONS

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

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

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

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

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

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

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

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

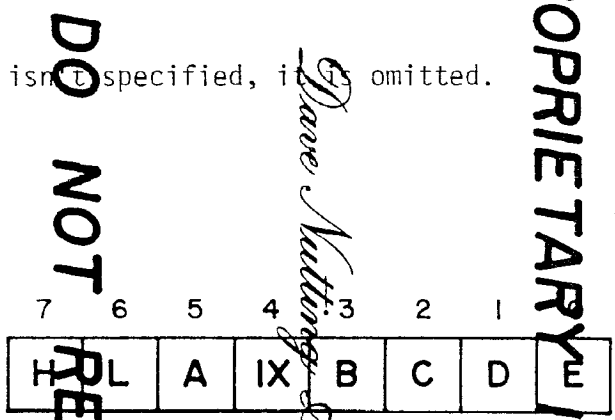
CONTEXT BLOCK FORMAT

IN LINE ARGUMENT MASK TABLE ENTRY
TABLES MRARGT and UMARGT

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

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

If an argument is not specified, it is omitted.



UPI INTPC
BEGIN INTERPRETING

Calling Sequence: SYSTEM INTPC
Aruguments: None
Notes: None
Description:

See UPI description for explanation of interpre

UPI XINTC
EXIT INTERPRETER

Calling Sequence: EXIT
Arguments: None

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

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

UPI RCALL
CALL ASSEMBLY LANGUAGE SUBROUTINE

Calling Sequence:

```
        DO      RCALL
          or
        DONT    RCALL
        DEFW    (routine address)
```

Arguments: HL=address of routine to call

Description:
RCALL may be used to call any assembly language subroutine from the interpreter. When the subroutine returns, interpretation proceeds at the next instruction.
When the assembly language routine receives control, HL will point at the routine's starting address; the other registers will contain their current values. Any changes made to the register set by the subroutine will not be passed along. To pass an output parameter, the subroutine must alter the context block, which is pointed at by IY.

Restrictions:
Assembler routine must not destroy IY.

Example:

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

UPI MCALL
CALL INTERPRETER SUBROUTINE

Calling Sequence:

```
SYSTEM  MCALL
     or
SYSSUK  MCALL
DEFW    (routine address)
```

Arguments: HL=Subroutine Address

Description:
MCALL is used to call an interpreter sequence or 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    ØFFFH
        DEFB    Ø
        DO      MRET            ;GO BACK TO CALLER
```

UPI MJUMP
INTERPRETER JUMP

Calling Sequence:

```
DO      MJUMP
   or
DONT    MJUMP
DEFW    (goto address)
```

Arguments: HL=Go to address

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

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

Example:

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

UPI MRET

RETURN FROM INTERPRETIVE SUBROUTINES

Calling Sequence: DO MRET

Arguments: None

Description:

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

SCREEN HANDLER

The screen handler is a group of routines for generating frame buffer images. Included are entries for filling sections of the screen with constant data, the animation of figures, and the display of 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 HORIZ (port 9H)
D=Data to output to VERT (port AH)

Output:

None

Description:

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

SCREEN FILL

FILL A CONTIGUOUS AREA WITH CONSTANT

Calling Sequence:

```
SYSTEM  FILL
     or
SYSSUK  FILL
DEFW    (first byte)
DEFW    (number of bytes)
DEFB    (data to fill with)
```

Arguments:

A =Data to fill with
BC=number of bytes to fill
DE=address to begin filling at

Description:

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

Notes:

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

SCREEN RECTAN

PAINT A RECTANGLE

Calling Sequence:

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

Arguments:

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

Description:

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

Example:

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

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

SCREEN WRITE ROUTINES

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

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

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

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

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

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

STANDARD CALLING SEQUENCE

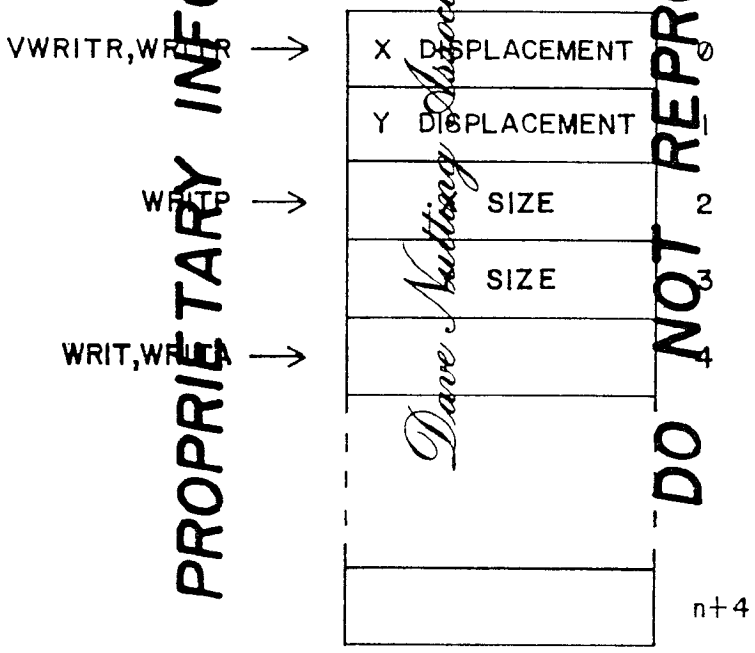
Every write routine uses a subset of the following argument/register assignment:

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

PATTERN REPRESENTATION

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

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



SCREEN WRITE VWRITR
WRITE RELATIVE FROM VECTOR

Calling Sequence:

```
SYSTEM  VWRITR
     or
SYSSUK  VWRITR
DEFW    (vector)
DEFW    (pattern)
```

Arguments: HL=Pattern address
IX=Vector Address

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

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

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

SCREEN WRITE WRITR
WRITE RELATIVE

Calling Sequence:

```
SYSTEM  WRITR
     or
SYSSUK  WRITR
DEFB    (X co-ordinate)
DEFB    (Y co-ordinate)
DEFB    (Magic Register)
DEFW    (Pattern address)
```

Arguments:

```
HL=Pattern address
A =Magic Register
D =Y co-ordinate
E =X co-ordinate
```

Output:

```
DE=Screen Address Used
A = Magic Register Used
```

Description:

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

Notes:

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

SCREEN WRITE WRITP

WRITE WITH PATTERN SIZE SCARE UP

Calling Sequence:

```
SYSTEM  WRITP
     or
SYSSUK  WRITP
DEFB    (X co-ordinate)
DEFB    (Y co-ordinate)
DEFB    (Magic Register)
DEFW    (Pattern address)
```

Arguments:

HL=Pattern Address
A =Magic Register
D =Y co-ordinate
E =X co-ordinate

Output:

DE=Screen Address Used
A =Magic Register Used

Description:

The pattern size is taken from the pattern.

Notes:

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

SCREEN WRITE WRIT

WRITE PATTERN

Calling Sequence:

```
SYSTEM  WRIT
     or
SYSSUK  WRIT
DEFB    (X co-ordinate)
DEFB    (Y co-ordinate)
DEFB    (X pattern size)
DEFB    (Y pattern size)
DEFB    (Magic Register)
DEFW    (Pattern address)
```

Arguments:

```
HL=Pattern Address
A =Magic Register to use
B =Y pattern size
C =X pattern size
D =Y co-ordinate
E =X co-ordinate
```

Output:

```
DE=Absolute address used
A =Magic Register used
```

Notes:

User must set ON/OFF color if using expansion.

SCREEN WRITE WRITA
WRITE ABSOLUTE

Calling Sequence:

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

Arguments:

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

Notes:

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

SCREEN SAVE

SAVE AREA

Calling Sequence:

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

Arguments:

B =Y size of area to save

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

DE=Address of save area

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

Description:

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

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

The save area may be MAGIC or non-MAGIC.

SCREEN RESTORE

RESTORE AREA

Calling Sequence:

```
SYSTEM  RESTOR
     or
SYSSUK  RESTOR
DEFW    (Save area)
DEFW    (Screen address)
```

Arguments:

DE=Save area to restore from

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

Description:

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

SCREEN VBLANK
BLANK FROM VECTOR

Calling Sequence:

```
SYSTEM  VBLANK
     or
SYSSUK  VBLANK
DEFW    (Vector address)
DEFB    (X size)
DEFB    (Y size)
```

Arguments:

D =Y size
E =X size (in bytes)
IX=Vector address

Description:

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

SCREEN BLANK

BLANK AREA

Calling Sequence:

```
SYSTEM  BLANK
     or
SYSSUK  BLANK
DEFB    (X size)
DEFB    (Y size)
DEFB    (Blank to)
DEFW    (Blank address)
```

Arguments:

HL=Blank address (not MAGIC)
B =Data to blank to
D =Y size
E =X size

Description:

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

SCREEN SCROLL
SCROLL WINDOW

Calling Sequence:

```
SYSTEM  SCROLL
     or
SYSSUK  SCROLL
DEFW    (line increment)
DEFB    (# of bytes)
DEFB    (# of lines)
DEFW    (first byte)
```

Arguments:

B =Number of lines to scroll
C =Number of bytes on line to scroll
DE=Line increment
HL=First byte to scroll

Description:

This routine copies NBYTES from first line +INC to first line. Thus to scroll upward, HL points at the first line (which is over-written) and the line increment would be positive. To scroll downward HL points at the last line and the line increment would be negative. The value in HL is an absolute address calculated by:
BASE OF SCREEN + #BYTES IN X OFFSET +(#lines offset*byte per line)

Note:

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

SCREEN ALPHANUMERIC
ALPHANUMERIC DISPLAY ROUTINES

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

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

Characters with hex equivalents in the range 1 - 1F are interpreted as tabulation codes which cause the character display routines to skip over N character positions before writing the following characters.

The characters 2ØH to 63H are displayed as 5 X 7 standard graphics with 3 pixels of horizontal spacing and 1 pixel of vertical spacing.

The characters between 64H and 7FH are interpreted by STRDIS as control codes which cause the contents of registers C, DE, and IX to be changed to the value that follow the string. See table accompanying STRDIS.

The characters between 8ØH and FFH are taken as references to a user supplied alternate character font.

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

Register C contains the options byte formatted as shown below.

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

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

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

OPTION BYTE

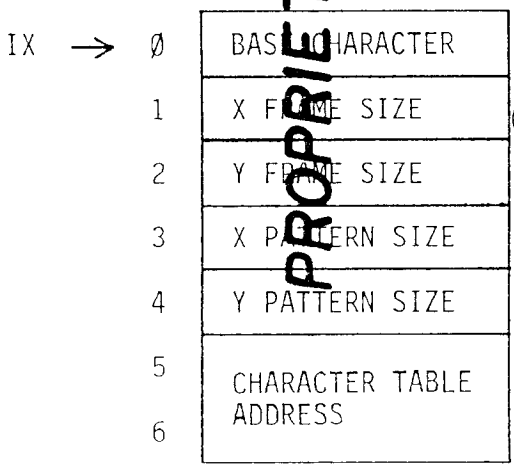
ENLARGE FACTOR	HOW MANY TIMES LARGER	ENLARGED SIZE OF SINGLE PIXEL
ØØ	1	1 X 1
Ø1	2	2 X 2
1Ø	4	4 X 4
11	8	8 X 8

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

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

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

The format of the alternate font descriptor is shown below.



EQUAL TO FIRST CHARACTER IN TABLE

CHARACTER SIZE IN BITS + X SPACING

CHARACTER SIZE IN BITS + Y SPACING

EACH CHARACTER TABLE ENTRY SHOULD BE OF SIZE X PATTERN*Y PATTERN SIZE

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
IX=Optional character font descriptor
```

*NOT LOADED

Outputs:

```
DE=Updated
```

Decription:

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

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

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

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

SCREEN ALPHANUMERIC DISTIM
DISPLAY TIME

Calling Sequence:

```
SYSTEM  DISTIM
     or
SYSSUK  DISTIM
DEFB    (X co-ordinate)
DEFB    (Y co-ordinate)
DEFB    (Options)
```

Arguments: DE=X,Y co-ordinates
X =Options (see note below)
IX=Alternate Font Descriptor (not loaded)

Outputs: DE=Updated

Description:
This routine displays the system time (GTMINS, 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=Ø to suppress colon and seconds.

SCREEN ALPHANUMERIC CHRDIS
DISPLAY CHARACTER

Calling Sequence:

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

Arguments:

A =ASCII character to display
C =Standard options byte
DE=Standard Y,X co-ordinates to begin at
IX=Optional alternate font descriptor address

*NOT LOADED

Outputs:

DE=Updated to next frame

Description:

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

Notes:

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

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

SCREEN ALPHANUMERIC DISPLAY STRING — STRDIS

Calling Sequences:

```
SYSTEM  STRDIS
     or
SYSSUK  STRDIS
DEFB    (X co-ordinate)
DEFB    (Y co-ordinate)
DEFB    (Options)
DEFW    (String)
```

Arguments:

HL=String address
C =Standard Options
DE=Standard Co-ordinates
IX=Alternate Font Descriptor Address

*NOT LOADED

Outputs: DE=Updated to next frame

Description:

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

Notes:

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

STRDIS INTERPRETATION OF CODES 64H to 7FH

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

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

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

SCREEN VECTORING - VECTORING ROUTINES

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

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

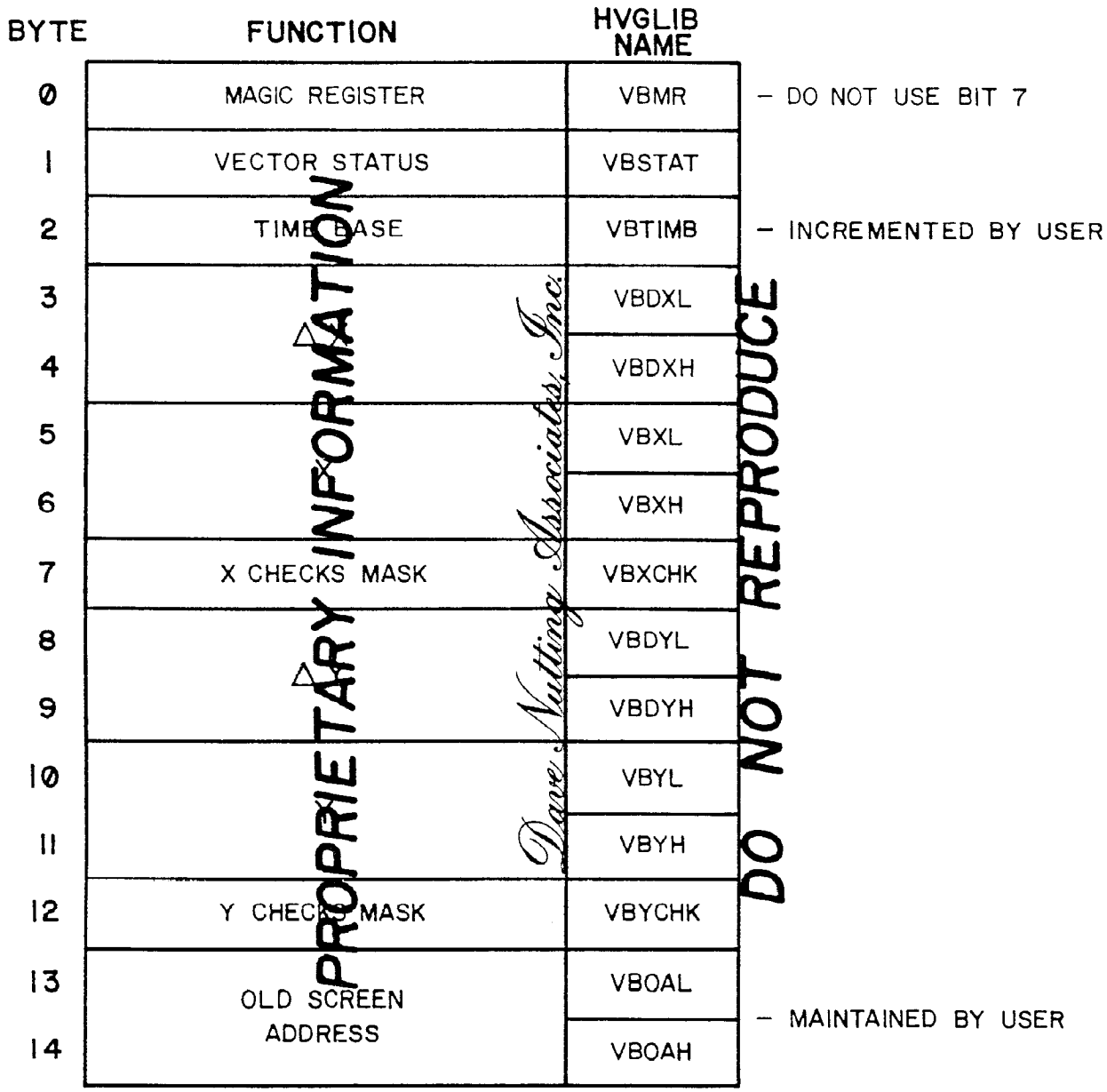
More formally, a vectored object possesses an X and Y co-ordinate. Associated with these co-ordinates are velocities, ΔX and ΔY, which are added to X and Y every time increment. Since the screen is finite, there also exists two upper and two lower limits X_{LU}, X_{LL}, Y_{LU}, and Y_{LL}, the attainment of which requires some response. The HVGSYS vectoring routine allows for two different responses to a limit attained. Either the sign of the delta is reversed or vectoring is stopped for this co-ordinate. This is specified by a flag byte. When attainment occurs this fact is indicated by a status byte. Also the co-ordinate is set equal to the limit that was attained, preventing over-shoot.

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

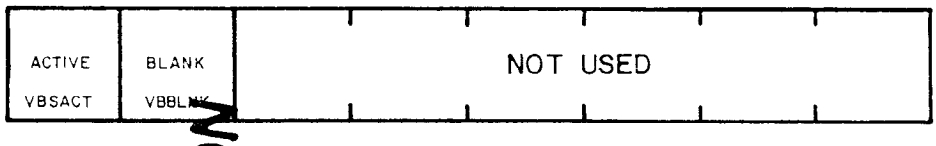
To insure high-accuracy, co-ordinates and deltas are double-precision. The assumed binary "decimal point" is between the high and low order byte.

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

VECTOR BLOCK



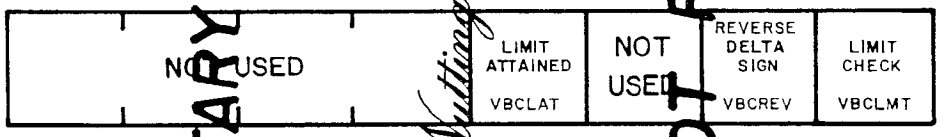
VECTOR STATUS DETAIL



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 VINIT and VBLANK system routines.

CHECKS MASK DETAIL



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

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

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

SCREEN VECTORING VECT
VECTOR OBJECT IN TWO DIMENSIONS

Calling Sequence:

```
SYSTEM  VECT
     or
SYSSUK  VECT
DEFW    (Vector address)
DEFW    (Limit table)
```

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

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

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

SCREEN VECTORING VECTC
VECTOR A CO-ORDINATE

Calling Sequence:

```
SYSTEM  VECTC
     or
SYSSUK  VECTC
DEFW    (co-ordinate address)
DEFW    (Limit table)
```

Arguments:

IX=Pointer to low-order byte of delta for co-ordinate
HL=Limits table for this co-ordinate (if required)
C =Time base to use

Description:

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

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

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

SCREEN RELABS

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

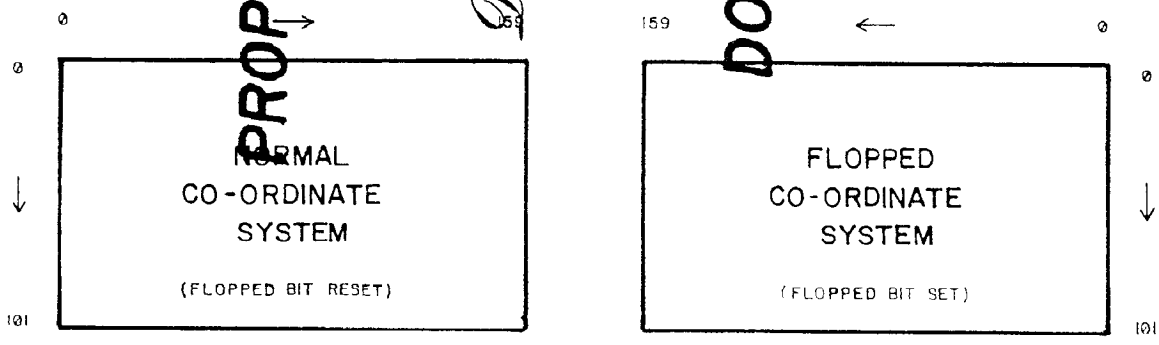
Calling Sequence: SYSTEM RELABS
or
SYSSUK RELABS
DEFB (Magic register value)

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

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

Description:

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



SCREEN RELAB1

CONVERT RELATIVE ADDRESS TO ABSOLUTE NORMAL ADDRESS

Calling Sequence: SYSTEM RELAB1

or

SYSSUK RELAB1

DEFB (Magic register value)

Arguments: A =Magic register value to combine with shift amount

D =Y co-ordinate

E =X co-ordinate

Output: A =Combined magic register value

DE=Absolute normal address (not magic)

Description:

This routine is identical to RELAB1 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 it

COLOR last [illegible]: 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 BREAKOUT for color restoration.

HUMAN INCSCR

INCREMENT SCORE AND COMPARE TO END SCORE

Calling Sequence: SYSTEM INCSCR

or

SYSSUK INCSCR

DEFW (address of score)

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

Output: Score incremented and optionally game over bit set

Description:

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

HUMAN PAWS

PAUSE

Calling Sequence: SYSTEM PAWS
or
SYSSUK PAWS
DEFB (number of interrupts)

Arguments: B=Number of interrupts to wait

Description:
This routine provides for a pause for certain number of interrupts. If used with ACT IM1, 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	↓	76
4	Percent	%	25
5	Recall	MR	52
6	Store	MS	53
7	Change sign	CS	3B
8	Divide	÷	F7
9	7	7	37
10	8	8	38
11	9	9	39
12	Times	×	2A
13	4	4	34
14	5	5	35
15	6	6	36
16	Minus	-	2D
17	1	1	31
18	2	2	32
19	3	3	33
20	Plus	+	2B
21	Clear Entry	CE	26
22	Ø	Ø	3Ø
23	Decimal point	.	2E
24	Equals	=	3D

HUMAN CONTROLS & KEYPAD SENTRY
SENSE TRANSITION

Calling Sequence: SYSTEM SENTRY
or
SYSSUK SENTRY
DEFW (Key mask address)

Arguments: DE=Keypad mask table

Description:
SENTRY checks for changes in the potentiometers (pots), control handles, triggers, keypad, semiphores and counter/timers. It also takes care of blackout. Blackout is the automatic blacking-out of the screen after 65 seconds without a change. If SENTRY isn't called then the game will not black out.

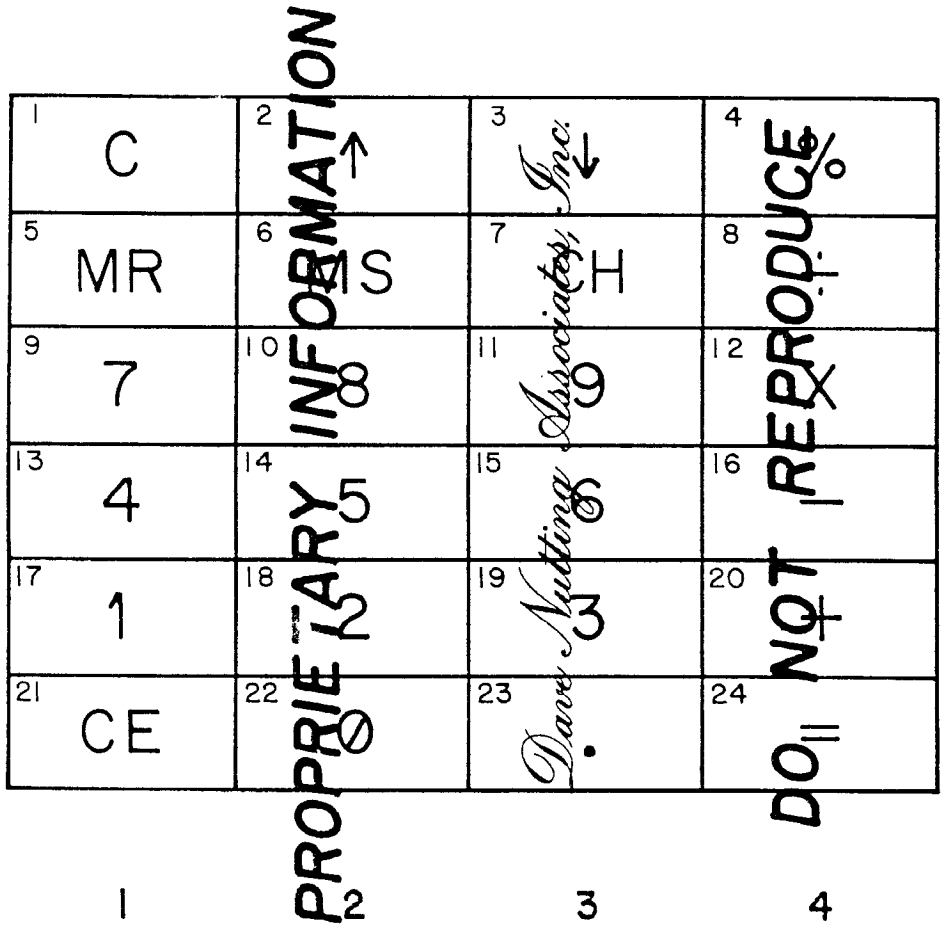
SENTRY checks if TIMOUT equals Ø on entry and if zero, it goes to PIZBRK. If a key has gone down or a control handle changed, then TIMOUT is set to FFH.

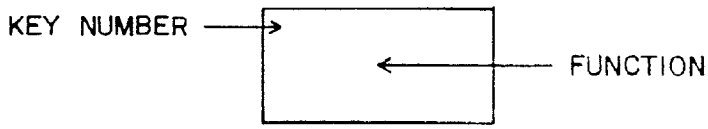
HL should point at a keypad mask. The keypad consists of 6 rows by 4 columns.

Example mask of just Ø - 9

```
DEFB    Ø111ØØB
DEFB    111ØØB
DEFB    Ø111ØØB
DEFB    ØØØØØØB
```

See diagram on following page.


1 C
2 ↑
3 ↓
4 %
5 MR
6 MS
7 CH
8
9 7
10 8
11 9
12 X
13 4
14 5
15 6
16
17 1
18 2
19 3
20 +
21 CE
22 0
23 .
24 =
0
1
2
3
4
5
MASK BIT NUMBER
1
2
3
4
MASK BYTE NUMBER
PROPRIETARY INFORMATION
Property of Dave Nutting Associates, Inc.
DO NOT REPRODUCE


KEY NUMBER
FUNCTION

Output: A=Return code
B=Extended code

PRIORITY	A=	MEANING
	SNUL	Nothing changed
1	SCTØ	Counter/timer Ø decremented to Ø
1	SCT7	Counter/timer 7 decremented to Ø
2	SFØ	SMI4S bit Ø went 1
2	SF7	SMI4S bit 7 went 1
4	SSEC	1 second has elapsed since the last SSEC
5	SKYU	Keypad went from down to up B=Ø
5	SKYD	Key is down B=key number
3	SPØ	Port Ø changed B=new value
3	SP3	Port 3 changed B=new value
6	SJØ	Joystick Ø changed B=new value
6	SJ3	Joystick 3 changed B=new value
6	STØ	Trigger Ø changed B=new value
6	ST3	Trigger 3 changed B=new value

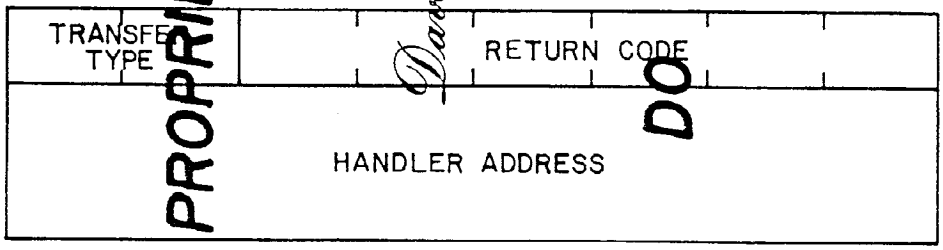
Notes:
The potentiometers (pots) are debounced. New trigger value=Trigger off (Ø) or trigger on (1ØH). When switches are actuated simultaneously the order of return is: SCT7 to SCTØ, SF7 to SFØ, SPØ to SP3, SSEC, SKYU, SKYD, SJØ, STØ, SJ1, ST1, SJ2, ST2, SJ3, ST3.

HUMAN CONTROL DOIT
RESPOND TO INPUT TRANSITION

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

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

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



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

HUMAN CONTROL PIZBRK
"COFFEE BREAK" BLACK OUT SCREEN AND WAIT FOR KEY

Calling Sequence: SYSTEM PIZBRK
or
SYSSUK PIZBRK

Input: NONE
Output: NONE

Description:
This routine blacks out the screen and waits for either a key press or a trigger or a joystick change.
This function should be called whenever a "hold until further notice" is needed.

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

HUMAN CONTROLS EXAMPLE

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

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

NUMBAS: DEFB    ØØ1ØØØB         ;NUMBER KEYS ONLY
        DEFB    ØØ1ØØ1B
        DEFB    ØØ1Ø1ØB
        DEFB    ØØ1Ø1Ø1B
        DEFB    Ø

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

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

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

INTERRUPT MUSIC PROCESSOR

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

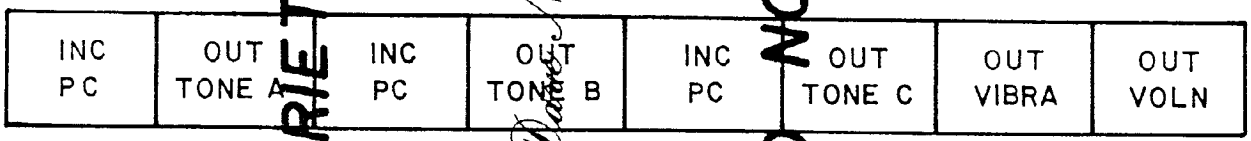
MPC: Like all program counters, points to the next data byte to fetch.

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

DURATION: Is loaded at the start of a note and then decremented every 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.



MUZCPU INSTRUCTION SET

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

Note: All durations are limited to a maximum of 127

MUSIC SCORE EXAMPLE

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

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

INTERRUPTS MUSIC EMUSIC
STOP MUSIC

Calling Sequence: SYSTEM EMUSIC
or
SYSSUK EMUSIC

Arguments: NONE
Outputs: NONE

Description:
Outputs Ø to volume ports and halts music processor.

INTERRUPTS ACTINT
ACTIVE INTERRUPTS

Calling Sequence: SYSTEM ACTINT
or
SYSSUK ACTINT

Input: NONE

Output: NONE

Function: Sets IM=2, INLIN=2ØØ, sets I reg + INFBK
Calls TIMEX and TIMEY
Enables interrupts

Description:

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

INTERRUPTS TIMERS DECCTS
DECREMENT COUNTER/TIMERS

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

Input: C=Mask indicative which counters to decrement.

Output: Sentry will notify the program.

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

INTERRUPTS TIMERS CTIMER

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

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

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

INTERRUPTS TIMERS STIMER
DECREMENT TIMERS

Calling Sequence:

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

Input: NONE

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

Uses: AF, BC, DE, HL

Calls: Music processor on note (duration) expiration.

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

MOVE MOVE BYTES

Calling Sequence:

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

Arguments:

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

Description:

MOVE uses LDIR to copy bytes from source to destination.

INDEXN INDEX NIBBLE

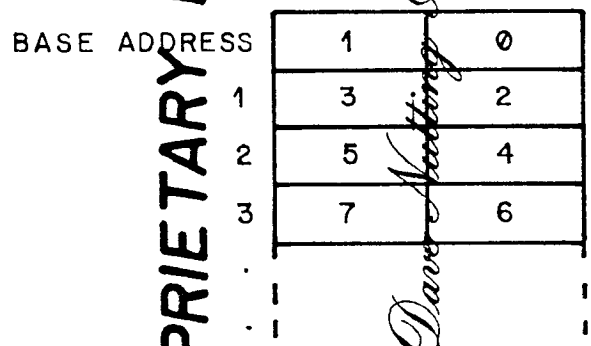
Calling Dequence: SYSTEM INDEXN
or
SYSSUK INDEXN
DEFW (Base Address)

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

Output: A =Nibble value

Description:

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



STOREN STORE NIBBLE

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

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

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

INDEXW INDEX WORD

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

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

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

Description: Indexing looks like: