COLECOVISION ™

PROGRAMMER'S MANUAL

restored by

Richard F. Drushel

12 February 1992

My thanks to Barry Wilson, A.N.N., for the generous (and prolonged) use of his original copy of this manual.

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

TABLE OF CONTENTS

APPENDICES

SECTION III

GRAPHICS GENERATION SOFTWARE

The graphics generation process is structured on three levels of software. A typical application will use routines from all three levels. These are the chip driver level, the table level and the object level. Figure 3-1 shows the program flow, software structure and its relationship with the outside world.

3.1 Chip Driver Level

The graphics hardware consists of the VDP and 16K VRAM. The VDP has eight write-only control registers and one read-only status register. The chip driver level software interfaces with the VDP registers and VRAM through the VDP. For detailed configuration of the registers, refer to the TMS 9928A VDP Data Manual.

The chip driver level software consists of six subroutines:

READ_VRAM, WRITE_VRAM, READ_REGISTER, WRITE_REGISTER, FILL_VRAM and MODE_1. The first five routines allow programs to access the VDP registers and transfer information to and from VRAM blocks. The sixth routine, MODE_1, initializes the VDP into a standard configuration.

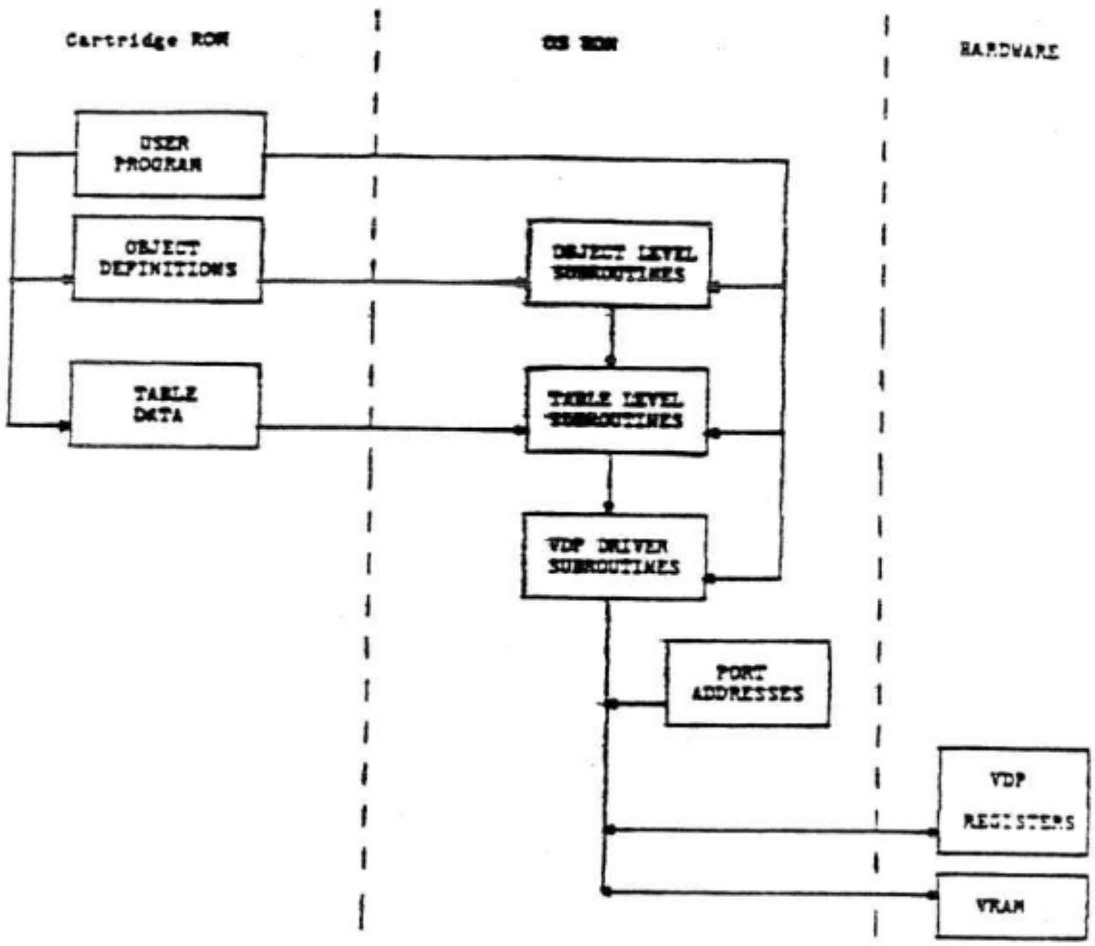


Figure 3-1
OS Graphics Software/VDP Interface

3.1.1. READ_VRAM

Calling Sequence:

```
LD    HL, BUFFER
LD    DE, SRCE
LD    BC, COUNT
CALL  READ_VRAM
```

Description:

READ_VRAM reads COUNT bytes from VRAM starting at SRCE and puts them in BUFFER.

Parameters:

BUFFER	This is the starting address of a CRAM buffer which is to receive the data read from VRAM.
SRCE	VRAM starting address to be read from.

COUNT	Number of bytes to be read from VRAM.

Side Effects:

- Destroys AF, BC, DE and HL.
- Cancels any previously initiated VDP operations.

3.1.2 WRITE_VRAM

Calling Sequence:

```
LD    HL, BUFFER
LD    DE, DEST
LD    BC, COUNT
CALL  WRITE_VRAM
```

Description:

WRITE_VRAM takes COUNT bytes from BUFFER and sends them through the VDP to VRAM. The starting address in VRAM for the write operation is given as DEST.

Parameters:

BUFFER	This is the starting address of a buffer where data to be sent to the VDP is located.

DEST	This is the VRAM address where the data is to be sent.
COUNT	This is the number of bytes that are to be transferred to VRAM. Count should be either less than 256 (100H) or even multiples of 256. (Ref. ColecoVision Bulletin No. 0002).

Side Effects:

- Destroys AF, BC, DE and HL.
- Cancels any previously initiated VDP operations.

3.1.3 READ_REGISTER

Calling Sequence:

```
CALL  READ_REGISTER
```

Description:

READ_REGISTER reads and returns the contents of the VDP status register in the accumulator. This value should be stored at VDP_STATUS_BYTE in CRAM. The information in this register can only be guaranteed valid during the vertical retrace time.

Return value:

Returns the contents of the VDP status register which has the following form (see VDP manual for further details):

Bit 7	Bit 6	Bit 5	Bits 4..0
Interrupt	Fifth Sprite	Coincidence	Fifth Sprite No.

Figure 3-2

VDP Status Register

Side Effects:

This routine has no effect at all in the processor memory or register space. However, a status read has a significant side effect to the VDP.

It acts as an interrupt acknowledge operation, i.e., it clears the interrupt flag and enables further generation of interrupts.

Thise side effect must be treated with care for two reasons. First of all, as is pointed out in the VDP manual, asynchronous reads may cause the interrupt flag in the status register to be reset before it is detected; this may cause problems in systems that expect to perform synchronization using the interrupt flag.

The second reason concerns interrupts which halt the execution of routines while they are accessing VRAM. In order to re-enable interrupts, a service routine must read the status register. However, to prevent the NMI from re-interrupting the service routine, the user should avoid reading the status register until all of its work is done. A defer interrupt routine, DEF_INT, has been developed to assist the user in handling this situation. Refer to ColecoVision Bulletin No. 0010 for additional information.

3.1.4 WRITE_REGISTER

Calling Sequence:

```
LD    B, REGISTER
LD    C, VALUE
CALL  WRITE_REGISTER
```

Description:

WRITE_REGISTER takes VALUE and writes it to the VDP register numbered REGISTER.

WRITE_REGISTER also maintains two bytes in CRAM starting at address VDP_MODE_WORD. The first is intended to duplicate the current contents of VDP Register 0, and the second to duplicate Register 1. When writing to a register using WRITE_REGISTER, the appropriate half of VDP_MODE_WORD is updated.

Parameters:

REGISTER	This is the VDP register number (0 - 7) to be written.
VALUE	This is the value to be written to REGISTER.

Side Effects:

- Destroys the AF register pair.

3.1.5 FILL_VRAM

Calling Sequence:

```
LD    HL, ADDRESS
LD    DE, COUNT
LD    A, VALUE
CALL  FILL_VRAM
```

Description:

FILL_VRAM writes COUNT copies of VALUE to VRAM starting at ADDRESS.

Parameters:

ADDRESS	VRAM address to start fill operation.
COUNT	Number of bytes to fill.

VALUE 8-bit value to fill with.

Side Effects:

- Destroys AF and DE.
- Cancels all previously initiated VRAM operations.

Calls to other OS routines:

- READ_REGISTER (Ref. Sec. 3.1.3)

3.1.6 MODE_1

Calling Sequence:

```
CALL  MODE_1
```

Description:

MODE_1 sets the VDP to graphics mode 1 and sprite size 0. It also uses the INIT_TABLE routine to define the VRAM table addresses as follows:

- Sprite Generator Table - 3800H
- Patter Color Table - 2000H
- Sprite Attribute Table - 1B00H
- Pattern Name Table - 1800H
- Pattern Generator Table - 0000H

When MODE_1 returns, the screen is blanked and the backdrop plane color is set to black.

Side Effects:

- Destroys AF, BC and HL.

Calls to other OS routines:

- WRITE_REGISTER
- INIT_TABLE

3.2 Table Level

The VDP requires various table areas within VRAM to operate. These tables are interrelated, each controlling its own aspect of the graphics generation process. The table level software provides routines which will read or write VRAM with respect to these table areas. The routines also provide the capability of reading and writing entire tables entries or sections of these entries up to and including the whole table. This level also has special functions which were found helpful.

The major difference between the table level and the chip driver level is that the applications programmer is no longer required to manipulate VRAM addresses on the table level. Instead, each of the VRAM tables is assigned a number or table code as listed in Table 3-1.

Table Name	Code
Sprite attribute table	0
Sprite generator table	1
Pattern name table	2
Pattern generator table	3
Pattern color table	4

Table 3-1

VRAM Table Code

When an applications program needs to operate on a table, only a table code needs to be passed to the applicable table processing the routine.

Furthermore, in graphics mode 1 and graphics mode 2, which are supported by the OS graphics software, the tables have more or less fixed shapes. The entry numbers and bytes per entry for each of the five tables, as well as their boundaries, is given in Table 3-2.

TABLE CODE	MODE(S)	ENTRIES	BYTES/ ENTRY	HEX EQUIVALENTS
0	1 & 2	32	4	80H
1	1 & 2	256	8	800H
2	1 & 2	768	1	400H
3	1	256	8	800H
3	2	768	8	800H
4	1	32	1	40H
4	2	768	8	2000H

Table 3-2
Table Entries and Boundaries

The table management software takes advantage of this regularity by letting application programs address table entries as integral entities. Let us take, for example, the task of getting the 14th sprite attribute entry from VRAM. In terms of the chip driver software, the task appears as follows:

- Get sprite attribute table address.
- Calculate offset into table (14 * table row length).
- Add offset to address.
- Read one table entry (4 bytes) from VRAM at off set + attribute table address.

On the other hand, when using the table level software, the task is now reduced to the following:

- Give offset into table (14).
- Give table code.
- Give item count (1).
- Call GET_VRAM (places the desired bytes at a user-defined area).

In a video program that requires accessing the sprite attribute table frequently (for example, an action-oriented game), the table level method constitutes a significant savings in cartridge code.

Software in the table level may be further subdivided into three groups of routines as follows:

- Table Managers
- Table-oriented Graphics Routines
- Sprite Reordering Software

3.2.1 Table Managers

There are three routines in this group: INIT_TABLE, GET_VRAM and PUT_VRAM. As the names imply, they deal with table initialization, getting data from tables and placing data into tables, respectively.

Table initialization is a very simple operation which involves assigning a base address to a table. The base addresses are "saved" for later use by GET_VRAM and PUT_VRAM for address calculations and remain fixed until they are reinitialized. GET_VRAM and PUT_VRAM both take a table code, an entry number, as well as an element count and a buffer address in CRAM as parameters when they perform their respective transfers of information between CRAM and VRAM.

3.2.1.1 INIT_TABLE

Calling Sequence:

```
LD    A, TABLE_CODE
LD    HL, ADDRESS
CALL  INIT_TABLE
```

Description:

INIT_TABLE takes a table code and a VRAM address at which that table is to reside, and initializes the VDP base address register for the given table. It also stores the unconverted form of the address in an array called VRAM_ADDR_TABLE for later use in address arithmetic. This address is stored at VRAM_ADDR_TABLE [TABLE_ CODE].

INIT_TABLE makes use of the current graphics mode in determining the actual value written to the base address register in some cases. It determines the graphics mode by looking at the VDP_MODE_WORD. Thus, it is imperative that the graphics mode be set up using WRITE_ REGISTER before INIT_TABLE.

Parameters:

TABLE_CODE	Number of the table to be initialized. TABLE_CODE must be one of the legal table codes defined in (Table 3-1).
ADDRESS	Intended VRAM address of table. Each table has its own boundary defined by the table base address in the VDP control register. The user should refer to Table 3-2 for the proper table boundary.

Side Effects:

- Destroys AF, BC, HL, IX and IY.

Calls to other OS routines:

- REG_WRITE

3.2.1.2 GET_VRAM

Calling Sequence:

```
LD    A, TABLE_CODE
LD    DE, START_INDEX
LD    HL, DATA
LD    IY, COUNT
CALL  GET_VRAM
```

Description:

GET_VRAM reads into the CRAM buffer DATA, COUNT entries from the table specified by TABLE_CODE, which starts at the table entry number START_INDEX.

GET_VRAM uses the VDP_MODE_WORD and VRAM_ADDR_TABLE to calculate VRAM addresses and byte counts. It is imperative, before calling GET_VRAM, that the graphics mode be initialized using WRITE_REGISTER, and that the table being accessed be initialized using INIT_TABLE.

Parameters:

TABLE_CODE	VRAM table code (Table 3-1) to be read.
START_INDEX	START_INDEX is a two-byte number that indicates the starting entry of the table.

The range of START_INDEX is table dependent. However, no boundary checking is done; therefore, if an index is given that is outside the range of the table, but still a legal VRAM address, the specified number of "entries" will be extracted from that location in VRAM.

Both the pattern generator and the color tables in graphics mode 2 are 768 entries long and they are segmented into three sections corresponding to the three sections of the display. When addressing these tables, the high order byte (D) of the two-byte START_INDEX value is a "segment specifier" (0 <= D <= 2), while the low order byte (E) specifies the index of the entry in that segment.

In the case of the sprite generator table, please note that COUNT refers to 8-byte shape for entries whether one is using size 0 or size 1 sprites.

DATA	Starting address of a CRAM data buffer to receive data from VRAM.
COUNT	Number of entries to be read from the VRAM table.

The restrictions on COUNT are again table dependent. In other words, it should always be the case that START_INDEX + COUNT <= Table Size.

Side Effects:

- Destroys AF, BC, DE, HL, IX and IY.
- This routine uses the local storage area SAVED_COUNT and is therefore not re-entrant.

Calls to other OS routines:

- READ_VRAM

3.2.1.3 PUT_VRAM

Calling Sequence:

```
LD    A, TABLE_CODE
LD    DE, START_INDEX
LD    HL, DATA
LD    IY, COUNT
CALL  PUT_VRAM
```

Description:

PUT_VRAM w ites from the buffer DATA, COUNT entries to the table specified by TABLE_CODE, which starts at the table entry number START_INDEX.

PUT_VRAM uses the VDP_MODE_WORD and VRAM_ADDR_TABLE to calculate VRAM address and byte counts. It is imperative that the graphics mode be set up using WRITE_ REGISTER and the table being accessed be initialized using INIT_TABLE before PUT_VRAM is called.

The table level of graphics software contains a sprite reordering feature where the major effect is in the operation of PUT_VRAM. When the MUX_SPRITES flag is set to TRUE (1), PUT_VRAM writes sprite entries to a CRAM copy of the sprite attribute table instead of writing them to VRAM. It locates this table through a pointer in low cartridge ROM called LOCAL_SPR_TBL. The sprite entries will then be re-ordered before being written to VRAM.

Parameters:

TABLE_CODE	VRAM table code (Refer to Table 3-1) to be written.
START_INDEX	START_INDEX is a two-byte number which indicates the starting entry number of the table. For other considerations, refer to the START_ INDEX parameter of GET_VRAM in Section 3.2.1.2.

DATA	Starting address of a data buffer where data to be written to VRAM resides.
COUNT	Number of entries to be put to the VRAM table. The restrictions on COUNT are again table dependent. In other words, it should always be the case that START_INDEX + COUNT <= Table Size.

Side Effects:

- Destroys AF, BC, DE, HL, IX and IY.
- Uses local storage locations, SAVE_TEMP and SAVED_COUNT.

Calls to other OS routines:

- WRITE_VRAM

3.2.2 Table-Oriented Graphics Routines

A number of routines are included in the table level graphics software that perform useful operations on generators. Each of these takes a table code, a source index from that table, a destination index in the same table, and the number of entries to be processed. The routines work in read-modify-write mode, that is, they pull the generators out of the table one at a time, process them and put them back. They use a CRAM buffer for their scratch area. This buffer is allocated by the applications programmer and accessable only through the pointer at WORK_BUFFER in cartridge ROM.

With one exception, the routines in this package always process generators one at a time, and write them to the destination block in the same order in which they are extracted from the source block. This has important implications for their use with size 1 sprites.

When the sprite size is 1, the hardware accesses four generators at the index found in a sprite's attribute table entry and displays them so that they appear on the screen as shown in Figure 3-3.

Sprite Screen Location

first generator	third generator
second generator	fourth generator

Figure 3-3
Sprite Size 1 Orientation

Thus, OS routines operating on the individual generators for a size 1 sprite will not be sufficient to orient the entire object. The four generators that make up the sprite will have to be permuted as well. The applications program will have to include a small routine that performs the required permutation in tandem with the OS call.

The following operations are available in the table-oriented graphics package:

- Reflection about the vertical axis
- Reflection about the horizontal axis
- 90-degree rotation
- Enlargement by a factor of two

3.2.2.1 REFLECT_VERTICAL

Calling Sequence:

```
LD    A, TABLE_CODE
LD    DE, SOURCE
LD    HL, DESTINATION
LD    BC, COUNT
CALL  REFLECT_VERTICAL
```

Description:

REFLECT_VERTICAL takes each generator in a block of COUNT generators following SOURCE in the table indicated by TABLE_CODE and modifies it in such a way that the new generator thus created will appear to be a reflection about the vertical screen axis of the old. The created generators are put back into a block of COUNT generators following DESTINATION in the same table.

The user must provide the permutation for size 1 sprite generators as diagrammed in Figure 3-4 below:

Block indicated by sprite name:

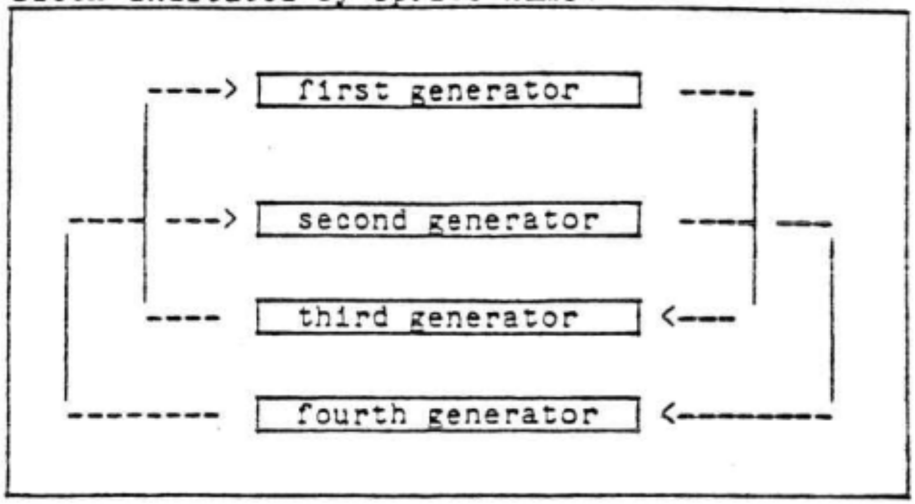


Figure 3-4
REFLECT_VERTICAL Size 1 Sprite Permutation

If TABLE_CODE is 3 (indicating the pattern generator table) and graphics mode 2 is used, REFLECT_VERTICAL also copies the color table entries for each generator it processes. Thus, when it is complete, the two-color table blocks indexed by SOURCE and DESTINATION will be identical. This means that the color scheme for the reflected generators will be the same as that for the originals.

Parameters:

TABLE_CODE	VRAM table code (Ref. Table 3-1) to be operated upon.
SOURCE	SOURCE is the two-byte index of the first entry in the specified table to be operated on.
	For table operations of sprite generator or pattern generator in graphics mode 1, SOURCE should be in the range 0 <= SOURCE <= 255. For pattern generators in mode 2, it should be in the range 0 <= SOURCE <= 767. In either case, if a value of SOURCE supplied is outside the table's range but still is a legal VRAM address, the specified number of "entries" will be read and modified from the VRAM location (table location) + 8 *

SOURCE. For the proper table entries and table boundary, refer to Table 3-2.

Sprite size has no effect on the range of SOURCE.

DESTINATION (HL) — DESTINATION indexes the place where REFLECT_VERTICAL will start putting generators back into VRAM after modifying them.

The same restrictions apply to the value of DESTINATION as to the value of SOURCE. They are both intended to be indices into the same generator table.

COUNT (BC) — A two-bytes count of the number of entries to be processed sequentially after SOURCE.

The legal value for COUNT is dependent on the size of the table being operated on and the values of SOURCE and DESTINATION. In general, both of the following statements should be true:

COUNT + SOURCE <= (table size)
COUNT + DESTINATION <= (table size)

Side Effects:

- Destroys AF, AF', BC, DE, DE', HL, HL', IX and IY.
- Uses the first 16 bytes of the data area pointed to by WORK_BUFFER.

Calls to other OS routines:

- GET_VRAM
- PUT_VRAM

3.2.2.2 REFLECT_HORIZONTAL

Calling Sequence:

```
LD    A, TABLE_CODE
LD    DE, SOURCE
LD    HL, DESTINATION
LD    BC, COUNT
CALL  REFLECT_HORIZONTAL
```

Description:

REFLECT_HORIZONTAL takes each generator in a block of COUNT generators following SOURCE in the table indicated by TABLE_CODE and modifies it in such a way that the new generator created will appear to be a reflection about the horizontal screen axis of the old. The created generators are placed back into a block of COUNT generators following DESTINATION in the same table.

The user has to provide the permutation for size 1 sprite generators as diagrammed in Figure 3-5.

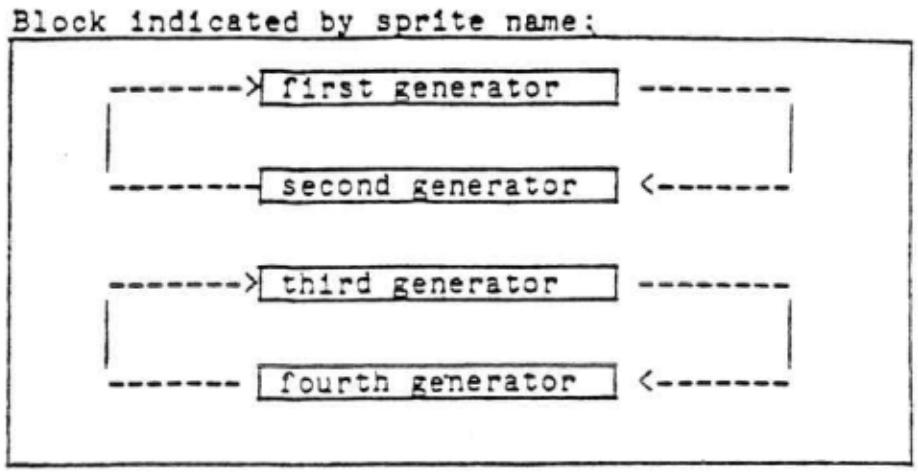


Figure 3-5
REFLECT_HORIZONTAL Size 1 Sprite Permutation

If TABLE_CODE is 3 (indicating the pattern generator table) and the graphics mode is 2, REFLECT_HORIZONTAL also performs the identical reflection on the corresponding color table entry for each generator it processes. This means that the reflected generators will be colored in a way that is consistent with their unreflected counterparts. When in mode 1, the color table is untouched.

Parameters:

TABLE_CODE	VRAM table code (Ref. Table 3-1) to be operated upon.
SOURCE	SOURCE is the two-byte index of the first entry in the specified table to be operated on. For table operations on sprite generator or pattern generator in graphics mode 1, SOURCE should be in the range 0 <= SOURCE <= 255. For pattern generators in mode 2, it should be in the range 0 <= SOURCE <= 767. In either case, if a value of SOURCE is supplied and is outside the table's range but still a legal VRAM address, the specified number of "entries" will be read and modified from the VRAM location (table location) + 8 * SOURCE. For the proper table entries and table boundary, refer to Table 3-2.

	Sprite size has no effect on the range of SOURCE.
DESTINATION	DESTINATION indexes the place where REFLECT_VERTICAL will start putting generators back into VRAM after modification.
	The same restrictions apply to the value of DESTINATION as to the value of SOURCE. They are both intended to be indices into the same generator table.
COUNT	A two-byte count of the number of entries to be processed sequentially after SOURCE.
	A legal value for count depends on the size of the table being operated on and the values of SOURCE and DESTINATION. In

general, both of the following statements should be true:

COUNT + SOURCE <= (table size)

COUNT + DESTINATION <= (table size)

Side Effects:

- Destroys AF, AF', BC, DE, DE', HL, HL', IX and IY.
- Uses the first 16 bytes of the data area pointed to by WORK_BUFFER.

Calls to other OS routines:

- GET_VRAM
- PUT_VRAM

3.2.2.3 ROTATE_90

Calling Sequence:

```
LD    A, TABLE_CODE
LD    DE, SOURCE
LD    HL, DESTINATION
LD    BC, COUNT
CALL  ROTATE_90
```

Description:

ROTATE_90 takes each generator in a block of COUNT generators following SOURCE in the table indicated by TABLE_CODE and modifies it in such a way that the new generator thus created will appear to be a 90-degree clockwise rotation of the old. The created generators are put back into a block of COUNT generators following DESTINATION in the same table.

The user must provide the permutation for size 1 sprite generators as diagrammed in Figure 3-6 below:

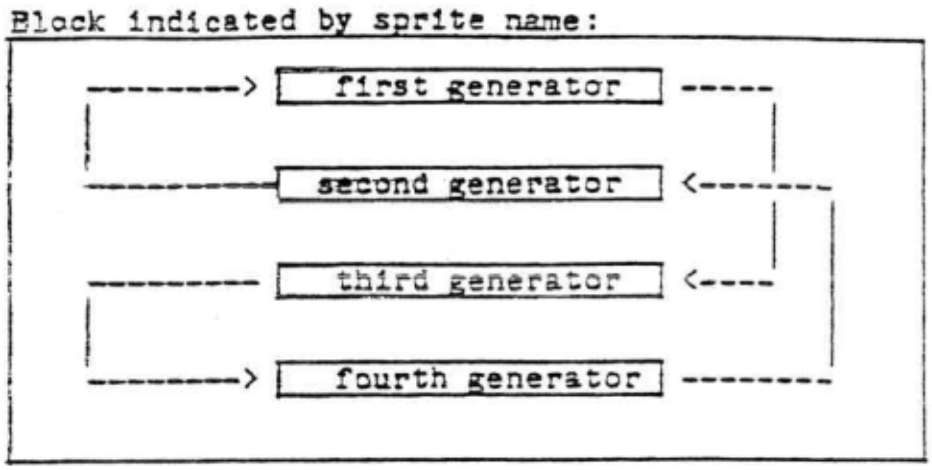


Figure 3-6
ROTATE_90 Size 1 Sprite Permutation

This routine should be used with great care when applied to pattern generators in mode 2. In this mode, the VDP allows arbitrary color combinations along vertical lines while it is still limited to two colors along a given 8-pixel horizontal line. The problem is that if the user attempts to rotate a figure that has more than two colors on a vertical line, ROTATE_90 will exhibit color problems after rotation. There is no way around this problem except to keep any generators that are intended for rotation simple. If the TABLE_CODE is 3 (pattern generator table) and the mode is 2, ROTATE_90 will copy the corresponding color table entries indexed by SOURCE to the block indexed by DESTINATION.

Parameters:

TABLE_CODE	VRAM table code (Ref. Table 3-1) to be operated upon.
SOURCE	SOURCE is the two-byte index of the first entry in the specified table to be operated on. For table operations of sprite generator or pattern generator in graphics mode 1, SOURCE should be in the range 0 <= SOURCE <= 255. For pattern generators in mode 2, it should be in the range 0 <= SOURCE <= 767. In either case, if a value of SOURCE is supplied and is outside the table's range but

still is a legal VRAM address, the specified number of "entries" will be read and modified from the VRAM location (table location) + 8 * SOURCE. For the proper table entries and table boundary, refer to Table 3-2.

Sprite size has no effect on the range of SOURCE.

DESTINATION

DESTINATION indexes the place where REFLECT_VERTICAL will start putting generators back into VRAM after modifying them.

The same restrictions apply to the value of DESTINATION as to the value of SOURCE. They are both intended to be indices into the same generator table.

COUNT	A two-byte count of the number of entries to be processed sequentially after SOURCE.
	The legal value for count is dependent on the size of the table being operated on and the values of SOURCE and DESTINATION. In general, both of the following statements should be true:
	COUNT + SOURCE <= (table size) COUNT + DESTINATION <= (table size)

Side Effects:

- Destroys AF, AF', BC, DE, DE', HL, HL' IX and IY.
- Uses the first 16 bytes of the data area pointed to by WORK_BUFFER.

Calls to other OS routines:

- GET_VRAM
- PUT_VRAM

3.2.2.4 ENLARGE

Calling Sequence:

```
LD    A, TABLE_CODE
LD    DE, SOURCE
LD    HL, DESTINATION
LD    BC, COUNT
CALL  ENLARGE
```

Description:

ENLARGE takes each generator in a block of COUNT generators following SOURCE in the table indicated by TABLE_CODE and from it creates four generators as shown below in Figure 3-7.

first generator	third generator
second generator	fourth generator

Figure 3-7
ENLARGE Generators Layout

The enlarged object will appear to be a double-sized version of the original. The created generators are put back into a block of 4 * COUNT generators following DESTINATION in the same table.

Note that since the ordering of the expanded generators is the same as that for the four generators needed to produce a size 1 sprite, ENLARGE lends itself well to use with sprites as long as the programmer is willing to dedicate four times as many sprites to the expanded object as to the orginal.

If TABLE_CODE is 3 (indicating the pattern generator table) and the graphics mode is 2, ENLARGE makes four copies of the color table entry for each source generator and places them in the color table so that they correspond to the four destination generators. This should mean that the color scheme for the enlarged object will be the same as that of the original. If the mode is 1, the color table is untouched.

Parameters:

TABLE_CODE	VRAM table code (Ref. Table 3-1) to be operated upon.
SOURCE	SOURCE is the two-byte index of the first entry in the specified table to be operated on. For table operations on a sprite generator or a pattern generator in graphics mode 1, SOURCE should be in the range 0 <= SOURCE <= 255. For pattern generators in mode 2, it should be in the range

0 <= SOURCE <= 767. In either case, if a value of SOURCE is supplied and is outside the table's range but still a legal VRAM address, the specified number of "entries" will be read and modified from the VRAM location (table location) + 8 * SOURCE. For the proper table entries and table boundary, refer to Table 3-2.

Sprite size has no effect on the range of SOURCE.

DESTINATION	DESTINATION indexes the place where ENLARGE will start placing generators back into VRAM after modifying them.

The same restrictions apply to the value of DESTINATION as to the value of SOURCE. They are both intended to be indices into the same generator table.

COUNT — A two-byte count of the number of entries to be processed sequentially after SOURCE.

The most important factor limiting the size of COUNT in the case of the ENLARGE routine is that ENLARGE actually produces four generators for every generator that it reads.

The legal value for count depends on the size of the table being operated on and the values of SOURCE and DESTINATION. Both of the following statements should be true:

COUNT + SOURCE <= (table size)

DESTINATION + 4 * COUNT) <= (table size)

Side Effects:

- Destroys AF, AF', BC, DE, DE', HL, HL', IX and IY.
- Uses the first 40 bytes of the data area pointed to by WORK_BUFFER.

Calls to other OS routines:

- GET_VRAM
- PUT_VRAM

3.2.3 Sprite Reordering Software

Probably the most significant hardware limitation of the VDP is the so-called "fifth sprite problem." This problem arises when more than four sprites occur on a single horizontal scan line. Because the chip only has four registers for dealing with the lower order sprites, the sprites with the higher sprite attribute indices cannot be generated on that scan line and therefore disappear.

One solution to this problem is to use a reordering scheme on the offending sprites which involves swapping the priorities of the sprite that is being blanked out with that of one of the higher order sprites in the group on successive video fields. The result is that while the sprites that are being reordered tend to flicker in the area of overlap, they are still quite visible. The degree of flicker depends on many factors including the color of the sprites in question and the background color and complexity.

The OS supports this solution by allowing the application to adjust the order of sprite attribute entries with minimum effort.

Two tables are used in implementing the sprite reordering feature. The first of these is simply a local CRAM version of the VRAM sprite attribute table. It must be allocated by the application program and made accessible to the OS by placing a pointer to it at the predetermined cartridge ROM location LOCAL_SPR_TBL. This local sprite attribute table need only contain the active sprite entries needed by the application and therefore may be shorter than the 128 bytes required for the VRAM version. The other table is called the sprite order table. It is also allocated by the application program through a pointer, SPRITE_ORDER, located in cartridge ROM. The sprite order table should contain one byte for each entry in the local sprite attribute table, and the bytes should take on values in the range 0 <= b <= 31.

When the flag MUX_SPRITES is false (0), PUT_VRAM writes sprite attribute entries directly to VRAM. However, when this flag becomes true (1), they are written instead to the local sprite attribute table. Then, a routine called WR_SPR_NM_TBL will map the local sprite attribute entries to VRAM according to the sprite order table.

An example of the relationship between the three tables may be illustrated as follows:

```
                                                      Sprite
Local Sprite              Sprite Order               Attribute
Table (CRAM)              Table (CRAM)              Table (VRAM)

| entry 0 | ----------->     0     | ---------->  entry 0 |
| entry 1 | ----------->     3     | -----        entry 1 |
| entry 2 | ----------->    17     | -----|  -|   entry 2 |
      .                      .            |   -|-> entry 3 |
      .                      .                 |     .
      .                      .                 |     .
      .                      .                 |     .
      .                      .                 --->| entry 17 |
      .                      .                       .
      .                      .                       .
```

Figure 3-8

Sprite Reordering Table Mapping

The advantage of this method lies in the fact that it takes a lot less work to reorder the bytes in the sprite order table than it does to move around the entries in the VRAM or CRAM sprite attribute tables.

3.2.3.1 INIT_SPR_ORDER

Calling Sequence:

```
LD    A, SPRITE_COUNT
CALL  INIT_SPR_ORDER
```

Description:

INIT_SPR_ORDER looks at the pointer SPRITE_ORDER in low cartridge ROM which should contain the address of a free area SPRITE_COUNT bytes long in CRAM. It sets this area up as a sprite order table by initializing it with zero through SPRITE_COUNT - 1.

Parameters:

SPRITE_COUNT	The length of the sprite order table, which whould be the same as the intended number of entries in the local sprite attribute table.

This number must always be in the range 1 <= SPRITE_COUNT <= 32.

Side Effects:

- Destroys AF, BC, and HL.

3.2.3.2 WR_SPR_NM_TBL

Calling Sequence:

```
LD    A, COUNT
CALL  WR_SPR_NM_TBL
```

Description:

WR_SPR_NM_TBL writes COUNT entries from the local sprite attribute table, which it accesses through the pointer LOCAL_SPR_TBL in low cartridge ROM, to the VRAM sprite attribute table. The transfer is mapped through the sprite order table which it accesses through the pointer SPRITE_ORDER in low cartridge ROM.

Parameters:

COUNT	This is the number of sprite attribute entries to be written to VRAM.

COUNT should not be larger than the initialized length of the sprite order table.

Side Effects:

- Destroys AF, BC, DE, HL, IX and IY.
- Cancels any previously established VDP operations.

3.3 Object Level

The object level software constitutes the top level of the graphics generation software, which appears to the user as a collection of screen objects with well-defined shape, color scheme, and location at any given moment. The software supports four distinct object types, each of which has its own capabilities and limitations. Once objects are defined, however, the rules for manipulating them are fairly type-independent. In fact, only one routine (PUTOBJ),is used to display objects of all types.

Brief descriptions are given in the following sections in regard to object types, object data structures and two user-accessible routines (ACTIVATE, PUTOBJ). For further information, refer to Appendix B.

3.3.1 Object Types

There are four different types of objects defined by the OS. A brief description for each type is given below.

3.3.1.1 Semi-Mobile

Semi-mobile objects are rectangular arrays of pattern blocks which are always aligned on pattern boundaries. Their animation capability is limited. In most cases they are used to set up background pattern graphics.

3.3.1.2 Mobile

The size of a mobile object is fixed in two-by-two pattern blocks. They belong to the pattern plane but can be moved from pixel to pixel in X,Y directions like a sprite superimposed on the background. However, the speed of mobile objects are too slow when compared to the sprites.

3.3.1.3 Sprite

Sprite objects are composed of an individual sprite.

3.3.1.4 Complex

Complex objects are collections of other "component" objects which may be of any type including other complex objects.

3.3.2 Object Data Structure

Each of the above mentioned objects has its definition in cartridge ROM. This high-level definition links together several different data areas which specify all aspects of an object. The data structure is described in detail in Appendix B.

3.3.2.1 Graphics Data Area

This data area is located in cartridge ROM. Pattern and color generators for semi-mobile, mobile and sprite objects and frame data for all objects are located in the graphics data area. The data structure within each graphics area depends on the type of object with which it is associated. If, however, two or more objects of the same type are graphically identical, they may share the same graphics area. This will reduce the amount of graphics data that needs to be stored in cartridge ROM.

3.3.2.2 Status Area

Each object will have its own status area in CRAM. The game program uses this area to manipulate the object. It does this by altering the location within status which determines which frame is to be displayed as well as the locations which define the position of the object on the display. The graphics routine, PUTOBJ, when called, will access the object's status area and place the object accordingly.

3.3.2.3 OLD_SCREEN

Mobile and semi-mobile objects appear in the pattern plane. They are displayed by altering some of the names in the pattern name table. The original names represent a background which is "underneath" the object. When the object moves or is removed from the pattern plane, the original names must be restored to the name table.

Before placing a semi-mobile or mobile object on the display, PUTOBJ will restore any previously saved names and also save the names which constitute the background underneath the new location of the object. Sprite and complex objects do not need OLD_SCREEN areas.

3.3.3 ACTIVATE

Calling Sequence:

```
LD    HL, OBJ_DEF
SCF
CALL  ACTIVATE
```

or

```
LD    HL, OBJ_DEF
OR    A
CALL  ACTIVATE
```

Description:

The primary purpose of this routine is to move the pattern and color generators from the graphics data area into the pattern and color generator tables in VRAM. Each object must be "activated" before it can be displayed. ACTIVATE also initializes the first byte in an object's OLD_SCREEN data area with the value 80H. PUTOBJ tests this location before restoring the background names to the name table. If the value 80H is found, it is an indication that there are no background names to restore.

Parameters:

OBJ_DEF	High level definition of an object. See Appendix B for further details.
SCF	Carry flag should be set if user wishes to load the generators specified for this object.

```
OR A                   Carry flag should be reset if user
                       knows that the generators are
                       already in VRAM.
```

3.3.4 PUTOBJ

Calling Sequence:

```
LD    IX, OBJ_DEF
LD    B, BKGND_SELECT
CALL  PUTOBJ
```

Description:

PUTOBJ is called when an object's frame or its location on the display is to be changed. The routine tests the type of object and then branches to one of several subroutines designed to handle that particular object type. These routines are not accessible to the user. Their functions are as follows:

1. PUT_SEMI

   Semi-mobile objects are placed on the display by writing the generator names specified by one of the object's frames into the pattern name table in VRAM. The pattern and color generators which are needed to create the frame must already be in their respective generator tables.

2. PUT_MOBILE

   Mobile objects are displayed by producing a new set of pattern and color generators which depict the frame to be displayed on the background. These new generators are then moved to the locations in the VRAM pattern and color generator tables which are reserved for the object; the names of the new generators are then written into the pattern name table.

3. PUT_SPRITE0

   PUT_SPRITE0 handles the display of size 0 sprite objects.

4. PUT_SPRITE1

PUT_SPRITE1 handles the display of size 1 sprite objects.

5. PUT_COMPLEX

PUT_COMPLEX calls PUTOBJ for each of its component objects.

Parameters:

OBJ_DEF	High level definition of an object. See Appendix B for further details.
BCKGND_SELECT	Used with mobile objects or complex objects with a mobile-type component. Can be ignored otherwise. For methods of selecting background colors in a mobile object. Refer to Appendix B for additional information.

SECTION IV

INTERRUPT HANDLING AND WRITE DEFERRAL

The 60Hz (50Hz for European version) non-maskable interrupt (NMI) in the ColecoVision system has a wide variety of applications such as providing a fixed time base for the timing software, and a natural debounce interval for the controller interface. However, interrupts can cause problems if not handled properly.

Let us say, for example, that the system is in the midst of a call to PUTOBJ and is, in fact, writing to VRAM when the interrupt occurs. If the interrupt service routine calls for transferring data to another area of VRAM by setting up the VDP address register (auto-increment) to a different value, the pending VRAM operation cannot resume properly after the interrupt is serviced.

The OS contains software which allows graphics operations on the object level to be protected against damage by asynchronous interrupts. It should be stressed that the OS protects ONLY the object level. Routines on the table and chip driver levels could be deferred against interrupt by using the application library routine, DEF_INT, suggested in ColecoVision Bulletin No. 0010 (Appendix D).

In order to implement this protection for graphics operations, the application program must allocate space for a deferral queue. The size of this queue depends on the number of graphics operations that are expected to be performed between NMIs, but usually fifteen entries of three bytes each will prove sufficient. The address of the queue should be passed on to the OS using a routine called INIT_WRITER which also empties the queue and prepares it for operation. Thereafter, whenever the flag byte DEFER_WRITES is set to true (1), calls to PUTOBJ are deferred by placing them on the queue where they may be performed using a single OS call from the interrupt service routine.

In addition to the buffer in which the queue resides, the deferral routines use several defined storage areas in the course of their operation. These are: QUEUE_SIZE, QUEUE_HEAD, QUEUE_TAIL, HEAD_ADDRESS, TAIL_ADDRESS and BUFFER. They are all related to the state of the queue.

4.1 INIT_WRITER

Calling Sequence:

```
LD    A, SIZE
LD    HL, LOCATION
CALL  INIT_WRITER
```

Description:

INIT_WRITER initializes the queue. It does not, in fact, do anything to the "physical" queue in RAM. Instead, it merely sets up its description by setting QUEUE_SIZE to SIZE, HEAD_ADDRESS and TAIL_ADDRESS to the beginning of the buffer and QUEUE_HEAD and QUEUE_TAIL to 0.

Parameters:

SIZE	The size in entries of the queue. SIZE should be equal to the amount

	of space allocated for the queue divided by three. Range for SIZE is 1 to 255.
LOCATION	The location of CRAM area allocated for the queue.

Side Effects:

- Destroys AF.

4.2 WRITER

Calling Sequence:

```
CALL  WRITER
```

Description:

WRITER performs any deferred PUTOBJ operations that may be on the queue emptying the queue as it goes. WRITER should be called by the interrupt service routine.

WRITER uses a "back door" into the PUTOBJ software without ever making an explicit call to PUTOBJ.

Side Effects:

Destroys AF, BC, DE, HL, IX, IY, BC', DE' and HL'.

SECTION V

TIMING SOFTWARE

Timing software enables the user to specify a preset length of time and to signal the user when that time has elapsed. In theory, up to 255 software timers are available to the user.

The Z80-CPU's non-maskable interrupt (NMI) input, which comes from the VDP interrupt output, forms the time base for all the timers. In the U.S., it is about every 1/60 second. In the European version, it is about every 1/50 second. TIME_MGR is the routine responsible for generating the time base at the desired intervals.

All timer routines use a CRAM area designated as the TIMER_TABLE. The size of this table depends on the number of timers in use and their types. There are two types of timers, non-repeating and repeating.

The user will be notified of the status of the timer only when he checks it.

5.1 Non-Repeating Timers

These timers will not repeat themselves after time out. The user will be notified and their timers are set free.

5.2 Repeating Timers

These timers only need to be set once. After each time out they will notify the user and repeat themselves.

For both types of timers, the timer length can be either short or long:

(a) Short - 1 to 255 units of time base.
(b) Long - 256 to 65535 units of time base.

5.3 TIMER_TABLE:

As a timer is requested, it is placed into TIMER_TABLE. Each timer consists of a Mode_Byte and a two-byte Value_Word.

The appropriate amount of CRAM should be reserved using the following formula:

```
TIMER_TABLE          DEFS      Num_of_Timers * 3
TIMER_DATA_BLOCK     DEFS      Num_of_Long_Repeating * 4
```

NOTE: Num_of_Timers is the total number of timers.

5.3.1 Mode_Byte

Mode_Byte is one byte of data for each timer containing the information of done bit, repeat bit, free bit, long bit and last-timer-in-table bit (Refer to Appendix G for details).

5.3.2 Value_Word

A two-byte value which can have several meanings depending on the type of timer.

(a) Short Timers:
The Value_Word is the actual timer in this case.

The first byte is the value to be decremented and the second byte is the initial timer value. In the case of a repeating timer, the second byte is used to restart the timer.

(b) Long Non-Repeating Timers:
The Value_Word is the value of the timer and is decremented as a tyo-byte quantity.

(c) Long Repeating Timers:
The Value_Word is the address of the location in the TIMER_DATA_BLOCK where the first word is the value to be decremented and the second word is the initial timer value.

5.3.3 TIMER_DATA_BLOCK

This is the data area in CRAM where four bytes are designated for each long repeating timer. The table's size is expandable under user control, so care should be taken not to write over other pertinent data.

5.4 INIT_TIMER

Calling Sequence:

```
LD   HL, TIMER_TABLE
LD   DE, TIMER_DATA_BLOCK
CALL INIT_TIMER
```

Description:

INIT_TIMER initializes timer data areas to the locations defined as inputs.

Parameters:

TIMER_TABLE	This is the CRAM address where the timer information will be placed.
TIMER_DATA_BLOCK	This is the address where long repeating timer data will be placed.

Side Effects:

- Destroys DE and HL.

5.5 TIME_MGR

Calling Sequence:

```
CALL  TIME_MGR
```

Description:

TIME_MGR is responsible for maintaining all OS software timers. The task of maintenance is defined as updating all timers, setting the proper signal code when a timer times out, and restarting repeating timers.

Each call to TIME_MGR will cause all active timers to be decremented by one. There is no limit as to when the routine could be called, but typically it is every NMI from VDP which forms the system time base.

An active timer is defined as a timer with its repeat bit set or its done bit not set, or both.

If an entire timer value decrements to zero, the done bit will be set in Mode_Byte. In addition, the timer will be restarted if it is a repeating type.

Parameters: None.

Side Effects:

- Destroys AF, DE and HL.

5.6 REQUEST_SIGNAL

Calling Sequence:

```
LD    HL, TIMER_LENGTH
LD    A, REPEAT
CALL  REQUEST_SIGNAL
```

Description:

REQUEST_SIGNAL accepts a time interval and a repeat code (Boolean) as input. The REPEAT parameter, when set, instructs TIME_MGR to re-initialize the timer when it times out instead of relinquishing the timer memory locations.

REQUEST_SIGNAL sets up a timer and assigns that timer a number in the accumulator. The routine also determines the type of timer and allocates space in the TIMER_TABLE accordingly.

Short Timer:

A short timer has a counter value of 255 or less and uses one Mode_Byte and Value_Word.

Long Timer:

A long timer has counter values greater than 255.

(a) Non-Repeating:

A non-repeating timer uses a Mode_Byte and Value_Word.

(b) Repeating:

A repeating timers uses a Mode_Byte and Value_Word in addition to four bytes starting at the first available location in the TIMER_DATA_BLOCK.

The user should save the timer number. This value, referred to as SIGNAL_NUM, should subsequently be used when calling TEST_SIGNAL to find the status of the signal or when calling FREE_SIGNAL to release a timer.

Parameters:

TIMER_LENGTH	The number of the time base units of a timer. Values range from 1 (shortest) to OFFFFH (longest).
REPEAT	1 = repeating timer; 0 = non-repeating timer.
Output:	Value of timer number returned in accumulator. User should save it in CRAM location SIGNAL_NUM.

Side Effects:

- Destroys AF, BC, DE, and HL.

5.7 TEST_SIGNAL

Calling Sequence:

```
LD    A, SIGNAL_NUM
CALL  TEST_SIGNAL
```

Description:

TEST_SIGNAL takes a signal number and tests to see whether the indicated timer has timed out since the last time it was tested. If so, it returns with "true" in the accumulator; otherwise, it returns "false". The zero flag reflects the contents of the accumulator.

If the timer of SIGNAL_NUM tested has its Done bit set and the timer is non-repetitive, then the Free bit will be set to release the timer for further use.

If no timer of a particular signal number exists then the routine will return a false.

Although it has been defined that testing a non-existing signal number will return a false value, a common error in use of timing routines is the testing of an undefined signal.

The error occurs when one module, with a given SIGNAL_NUM, calls TEST_SIGNAL with that SIGNAL_NUM as input. If this module receives a "true" from TEST_SIGNAL, then another module which is rightfully using a timer with that SIGNAL_NUM will not receive a "done" signal.

Parameters:

SIGNAL_NUM Timer number.

Sido Effects:

- Destroys AF (output), BC, DE, and HL.

5.8 FREE_SIGNAL

Calling Sequence:

```
LD    A, SIGNAL_NUM
CALL  FREE_SIGNAL
```

Description:

FREE_SIGNAL takes a SIGNAL_NUM value as input and upon finding a timer assigned to that number, releases it to the free list. If no timer of that SIGNAL_NUM is found, no action will be taken. This routine will free a timer regardless of its current value or its REPEAT parameter.

Special case of long repeating timer:

This routine will release a portion of the TIMER_DATA_BLOCK that a particular timer uses and moves the remaining contents up. The Value_Words of other repeating timers will also be modified to reflect this move.

NOTE: In this special case, TIME_MGR, or any other routine that accesses or modifies the TIMER_TABLE, should NOT be called during the execution of FREE_SIGNAL. (This may occur if TIME_MGR was called on interrupt). ColecoVision Bulletin No. 0010 (Appendix D) suggests the solution of using DEF_INT to defer interrupts.

Parameters:

SIGNAL_NUM	Previously defined output from REQUEST_SIGNAL.

Side Effects:

- Destroys AF, BC, DE and HL.

SECTION VI

CONTROLLER INTERFACE

Most applications involving the hand controller require similar needs in decoding and debouncing those inputs. The operating system addresses those needs in one general purpose routine, POLLER. POLLER will decode and debounce either all or selected portions of the hand controller hardware and place the processed data in the Controller Data Area selected by the pointer in CONTROLLER_MAP.

Special applications may require non-standard decoding of the inputs available from the hardware; therefore, entry points to lower level routines are available.

There are four routines available to access controller inputs:

- POLLER
- DECODER
- CONT_SCAN
- UPDATE_SPINNER

6.1 Controller Data Area

The pointer in CONTROLLER_MAP points to the user-defined CRAM area which is accessed and/or modified when POLLER is called. Users define this address by placing the location of the 12 bytes of the CRAM Controller Data Area at cartridge location CONTROLLER_MAP. They are defined as follows:

+0	Player 1 enable	
+1	Player 2 enable	
+2	Fire button (left button)	Player 1
+3	Joystick	Player 1
+4	Spinner count (for interface modules)	Player 1
+5	Arm button (right button)	Player 1
+6	Keyboard	Player 1
+7	Fire button	Player 2
+8	Joystick	Player 2
+9	Spinner count	Player 2
+10	Arm button	Player 2
+11	Keyboard	Player 2

Player Enable (+0, +1):

Bit 7							Bit 0
	X	X			X		

Where bit = 1: Function enabled.

bit = 0: Function disabled.

X = Don't care

While functions are as follows:

Bit 7 = Controller Enable

Bit 4 = Keypad

Bit 3 = Arm Button

Bit 1 = Joystick

Bit 0 = Fire Button

Status of individual portions of the controller map area when enabled is described as follows:

Fire button:

Status = 040H, if fire button pressed

Status = 0H, if fire button not pressed

Joystick:

Status	Direction
01H	N
03H	NE
02H	E
06H	SE
04H	S
0CH	SW
08H	W
09H	NW

Spinner Switch:

SPIN_SW_CNT is added to the value for position offset.
(Ref to Sec. 6.5)

Arm Button:

Status = 0040H if arm button pressed

Status = 0000H if arm button not pressed

Keypad:

Value	Key
00H	0
01H	1
02H	2
03H	3
04H	4
05H	5
06H	6
07H	7
08H	8
09H	9
0AH	*
0BH	#

6.2 POLLER

Calling Sequence:

```
CALL  POLLER
```

Description:

Reads, decodes and debounces all active portions of both controllers. The results are placed in the Controller Data Area.

POLLER's debounce algorithm waits until it finds the data the same for two successive passes before it modifies the Controller Data Area. If a particular portion is disabled, then this routine will still be looking for the second occurrence upon re-enabling. Please note that the POLLER routine cannot interrupt itself.

Side Effects:

-Destroys all except alternate register pairs, does not destroy alternate AF pair.

- Zero's SPIN_SW_CNT if that portion of the controller is enabled. (See UPDATE_SPINNER).

Calls to other OS routines:

- CONT_SCAN

6.3 DECODER

Calling Sequence:

```
LD   H, CNTRLR NO.
LD   L, CNTRLR SEGMENT NO.
CALL DECODER
```

Description:

DECODER calls CONT_SCAN; decodes and returns as output according to the controller segment requested. Decoding uses the same format as the individual status bytes in Controller Data Area.

Parameters:

CNTRLR NO.	0 = Player 1's controller only 1 = Player 2's controller only
CNTRLR SEGMENT NO.	The value found in segment number will decode these respective portions of the controller:

0 = Fire, Joystick, Spinner

1 = Arm, Keypad

OUTPUTS:	IF SEGMENT CHOSEN WAS:	
	Segment 0	Segment 1
Register H	Fire	Arm
Register L	Joystick	Keyboard
Register E	Spinner	

The decoded values are listed in the Controller Data Area.

Side Effects:

- Destroys AF, BC, DE and HL.

Calls to other OS routines:

- CONT_SCAN

6.4 CONT_SCAN

Calling Sequence:

```
CALL  CONT_SCAN
```

Description:

Reads the actual ports to both controllers and places the data in an OS-defined CRAM area. These locations are labeled as SO_CO, SO_C1, S1_CO and S1_C1.

Side Effects:

- Destroys AF.

6.5 UPDATE_SPINNER

Calling Sequence:

```
ORG   801EH
JP    UPDATE_SPINNER
```

Description:

For use with expansion modules only. Interrupt service routine which processes controller spinner switch interrupts (maskable). Decrements OS reserved byte SPIN_SWO_CNT for Controller No. 0 or SPIN_SW1_CNT for Controller No. 1 if spinner is going in one direction; increments byte if spinner is going in the other direction (Ref. Table 10-1).

NOTE: SPIN_SW_CNT is accessed and modified by both DECODER and POLLER if they are called.

SECTION VII

SOUND GENERATION SOFTWARE

The OS provides sound generation routines that output frequency, attenuation and control data to the TI SN76489 sound generator controller. The "sound" described in this section can be represented as both music or noise.

There is at least one ten-byte block of CRAM called a "Sound Data Area" reserved for each sound channel. This area contains a record of the current values "playing" on that sound channel. These values are the timing and descriptive information which generate musical notes that are originally stored in cartridge ROM. In total, there should be a minimum of four sound data areas reserved by the user, one for each channel. More data areas are needed if there are sounds to be played concurrently. For an average video game, seven is the required number.

Basically, in order to generate sound effects, the user has to prepare music notes and call the sound generation routines. The notes table, pointer and four routines are described below. For detailed information, refer to the Sound Users' Manual in Appendix C.

7.1 LST_OF_SND_ADDRS and PTR_TO_LST_OF_SND_ADDRS

All the music notes for an application program starts at the address called LST_OF_SND_ADDRS in cartridge ROM. There is another dedicated CRAM pointer located at address PTR_TO_LST_OF_SND_ADDRS which points to the LST_OF_SND_ADDRS. It is the user's responsibility to set up the pointer before passing control to any sound generation software.

7.2 SOUND_INIT

This routine should be called immediately after power on, before any sound processing can occur. It turns off the sound generators, initializes the CRAM locations to be used as sound data areas, sets up the four channel data area pointers and initializes PTR_TO_LST_OF_SND_ADDRS.

INPUT:	n
TYPE:	8-bit constant
PASSED:	in B
DESCRIPTION:	Number of sound data areas used by the game.

INPUT:	LST_OF_SND_ADDRS
TYPE:	16-bit address
PASSED:	in HL
DESCRIPTION:	LST_OF_SND_ADDRS is the base address of a list of the starting addresses for each sound's note list and data area.

OUTPUT:

1. Turns off all sound generators.
2. Initializes PTR_TO_LST_OF_SND_ADDRS.
3. Writes the inactive code (OFFH) to byte 0 of the n sound data areas.
4. Stores 00 at end of sound data areas.
5. Sets the 4 channel sound pointers to a dummy inactive area.
6. Sets SAVE_CTRL to OFFH. (See "Noise Notes" discussion in

ColecoVision Sound Users' Manual in Appendix C).

7.3 PLAY_IT

PLAY_IT is called to start a sound. Using a sound number passed in B, PLAY_IT loads the data for the sound's first note into the appropriate sound data area, thereby truncating whatever sound had been "playing" in that data area. (The address of the appropriate area is found by using the sound number as an index into the LST OF_SND_ADDRS table). It also formats the data area's header and sets up the next note pointer. If the sound is a special sound effect, its next note pointer is set to the address of the special effect routine. The next time PLAY_SONGS is called, that sound's first note will be played.

If PLAY_IT is called with a sound number of a sound which is already in progress, it returns immediately (i.e., it doesn't restart the sound).

INPUT:	Sound number to be started.
TYPE:	8-bit constant, 1 to 61.
PASSED:	In B.
CALLS:	PT_IX_TO_SxDATA, LOAD_NEXT_NOTE PTR, UP_CH_DATA_PTRS.
OUTPUT:	1. Moves the sound's first note data to the appropriate sound data area. 2. Formats byte 0 header of the sound's data area. 3. Points next note pointer in data area (bytes 1 & 2) to address of first note in sound, or address of special sound effect routine.

7.4 SOUND_MAN

SOUND_MAN should be called every VDP interrupt. For each data area, SOUND_MAN processes the appropriate timer and sweep counters and modifies the frequency and attenuation data accordingly. If the data area is assigned to a special effect, SOUND_MAN calls that effect. When a note is finished, SOUND_MAN, using the data area's next note pointer, moves data for the next note of the sound into the area. If SOUND_MAN reads a header byte (in Cart ROM) that has bits 3 and 4 set, indicating repeat sound, it will start the sound again by reloading the first note in the sound.

After the operations upon a data area have been performed, if necessary, the channel data area pointers (PTR_TO_S_ON_x) are updated. The following data areas are processed in the same fashion, in order of occurrence, until the end of data area code, 00, is reached.

SOUND_MAN does not output the modified frequency and attenuation data. PLAY_SONGS is called just before SOUND_MAN to do this.

Special codes in byte 0 of the sound data area indicate:

255: Data area inactive, do no processing;
62: A special effect is to be played; SOUND_MAN calls the effect routine;
0: End of sound data areas (SOUND_MAN processes data areas until it sees 0 in byte 0).

NOTE: Sound number 1 MUST use the first area in the block of sound data areas. SOUND_INIT uses this address to find the sound data area.

INPUT:	None.
CALLS:	PT_IX_TO_SxDATA, PROCESS_DATA_AREA.
OUTPUT:	Calls routines which: 1. Decrement sound duration and sweep timers.

2. Modify swept frequency and attenuation values.
3. Call special effects routines where necessary.
4. Update the channel data area pointers if necessary.
5. Restart the sound if indicated.

7.5 PLAY_SONGS:

PLAY_SONGS takes the frequency and attenuation data pointed to by the four channel data area pointers (PTR_TO_S_ON_X) and outputs it to the four sound chip generators.

INPUT: None.

CALLS: TONE_OUT, UPATNCTRL.

OUTPUT:

1. Current frequency and attenuation data is output to

each tone generator, if sound on that channel is active; if sound on that channel is inactive, that generator is turned off.

2. Noise generator is sent current attenuation data and control data, if new.
3. Modifies SAVE_CTRL if necessary.

7.6 Application

These four routines would normally be called as follows:

Begin

Power on inits done by OS

Cartridge program receives control

LD B, number of song data areas used in the game

LD HL, address where LST_OF_SND_ADDRS is store in ROM.

```
        CALL SOUND_INIT to initialize sound data areas
        Whatever other power on inits you want to do
        Start game
        .
        .
        .
        LD B, number of sound you want to start
        CALL PLAY_IT to set up for start of sound
        .
        .
        VDP interrupt occurs:
        CALL PLAY_SONGS to output data
        CALL SOUND_MAN to process sound data
        Whatever else you want to do during VDP
        interrupt
        RETN to game
    End
```

SECTION IX

MISCELLANEOUS UTILITIES

9.1 ADD816

Calling Sequence:

```
LD    A, VALUE
LD    HL, ADDRESS
CALL  ADD816
```

Description:

ADD816 adds an 8-bit signed number in accumulator to a 16-bit unsigned number pointed to by HL; returns with altered 16-bit number at the HL address.

Parameters:

VALUE	8-bit signed number.

ADDRESS	Address pointing to a 16-bit unsigned number
Output:	Two-byte value at the address pointed to by the HL register pair.

Side Effects:

Destroys registers A, F and B.

9.2 DECLSN

Calling Sequence:

```
LD    HL, ADDRESS
CALL  DECLSN
```

Description:

DECLSN decrements least significant nibble of a byte pointed to by HL without affecting most significant nibble or HL. Returns with altered 8-bit number at HL address. Sets Z-flag if 0, C-flag if -1.

Parameters:

ADDRESS	Address pointing to an8-bit unsigned number.
Output:	A one-byte value at the address pointed to by the HL register pair.

Side Effects:

Destroys A and F.

9.3 DECMSN

Calling Sequence:

```
LD    HL, ADDRESS
CALL  DECMSN
```

Description:

DECMSN decrements the most significant nibble of byte pointed to by HL without affecting the least significant nibble or HL. Returns with altered 8-bit number at HL address. Sets Z-flag if 0, C-flag if -1.

Parameters:

ADDRESS	Address pointing to 8-bit unsigned number.
Output:	A one-byte value at the address pointed to by the HL register pair.

Side Effects:

Destroys A and F.

9.4 MSNTOLSN

Calling Sequence:

```
LD    HL, ADDRESS
CALL  MSNTOLSN
```

Description:

MSNTOLSN copies the most significant nibble of byte pointed to by HL to the least significant nibble of that byte. The routine returns the results at the location pointed to by HL.

Parameters:

ADDRESS	Address pointing to an 8-bit unsigned number.
Output:	A one-byte value pointed to by HL register pair.

Side Effects:

Destroys A, F and B.

9.5 RAND_GEN

Calling Sequence:

```
CALL  RAND_GEN
```

Description:

RAND_GEN is a 16-bit psuedo random number generator. It "exclusive OR's" the 15th and 8th bit together and then rotates the entire quantity to the left and inserts the "exclusive OR'ed" bit into the rightmost bit. Upon leaving, it stores the random number at global location RAND_NUM.

Output: The random number can be found in the HL register pair or RAND_GEN because RAND_GEN contains the value of L while RAND_GEN + 1 has the value of H, or in the accumulator because A = L before RET.

Side Effects:

Destroys registers AF and HL (return values).

9.6 LOAD ASCII

Calling Sequence:

```
CALL  LOAD_ASCII
```

Description:

LOAD_ASCII writes out the ASCII generator set to the pattern generator table. The ASCII table is located in Cartridge ROM starting at ASC_TABLE. INIT_TABLE must be called to set up the table addresses before using this routine.

Side Effects:
Destroys AF, DE, HL and IY.

Calls to other OS routines:
- PUT_VRAM

SECTION X

DEFINED REFERENCE LOCATIONS

10.1 OS ROM

In the OS ROM area, it is IMPORTANT to know that the application programs should only use the OS entry points listed in the OS_SYMBOLS file. Accessing to the OS otherwise is illegal and may cause program malfunction when hardware configuration changes or OS routines relocated due to update. The jump table starts from location JUMP_TABLE through the end of OS ROM. It contains all the subroutine entry points released to the user.

At the beginning of the cartridge, there are eight programmable restarts at addresses 0008H, 0010H, 0018H, 0020H, 0028H and 0030H. Each of the restarts jump to a location in Cartridge ROM where a vector can be provided to access an OS entry point. The Z80A-CPU hardware also designates location 0038H to service maskable interrupt

(MI) and location 0066H to service non-maskable interrupt (NMI). Jump instructions are provided for these two reference locations for the user to implement interrupt vectors in Cartridge ROM. Starting at location 0069H is the OS ROM data area which contains the AMERICA byte, ASCII table address and numeric table address. Figure 10-1 is the OS ROM map showing all the reference locations mentioned above. Appendix E lists all entry points of the Jump Table.

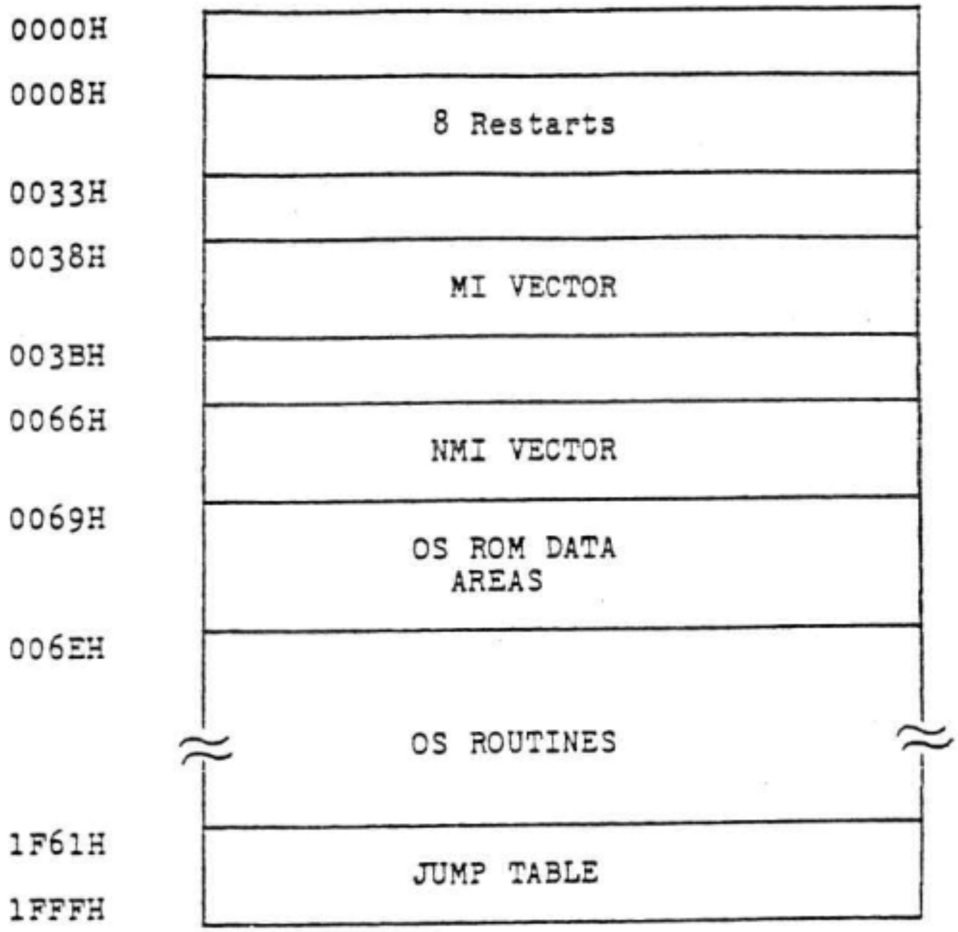


Figure 10-1

OS ROM MAP

10.1.1 Europe/America Byte:

The European TV uses PAL system (625-line format) which requires interrupt at the end of each active-display scan every 1/50 second, as opposed to every 1/60 second for the US model (NTSC, 525-line format).

ColecoVision cartridges must be interchangeable between both systems, the Europe/America byte at AMERICA in OS ROM, has been established to detect which version of the unit is in use. If a real-time display (such as a clock) must be implemented, the program will have to access the Europe/America byte to determine the current line frequency. For America-based units, this location will contain 60 (3CH) and for European-based units, it will contain 50 (32H).

10.1.2 Restart Vectors

Figure 10-2 shows the eight programmable restarts their addresses and corresonding locations in Cartridge ROM.

OS ADDRESS	JUMP TO CART. ROM ADDR.
0008H	800CH
0010H	800FH
0018H	8012H
0020H	8015H
0028H	8018H
0030H	801BH

Figure 10-2
OS RESTARTS

For each of the restart locations above, there should be a vector in Cartridge ROM provided by the user. To use a restart, the user must place a jump instruction to the address of the routine which he or she wishes to access through the Cartridge ROM vector; for example, JP WRITE_ VRAM at 800CH. These routines are usually the ones most frequently used in order to save application program space.

10.1.3 Graphics Tables

There are two graphics tables in the OS available to the user. The pointers for the ASCII table and Number table are defined in the locations of ASCII-TABLE and NUMBER_TABLE.

The ASCII table contains pattern generators for all 26 upper and psuedo-lower (half-size upper) case letters plus eleven special characters in 5x7 dot matrix form. The number table contains pattern generators for the numbers from 0 to 9 plus seven special characters.

10.2 Cartridge ROM

At the beginning of Cartridge ROM, locations are reserved for testing cartridge presence (Section 8-3), plus a number of pointers which point to tables, buffers and start of the game. On top of the pointers there are spaces allocated for restart (Ref. Figure 10-2) and interrupt vectors. There are up to 60 bytes available to the user starting at location GAME_NAME, to name the cartridge, their format has been described in the title screen in section 8.2. Figure 10-3 shows the cartridge ROM map.

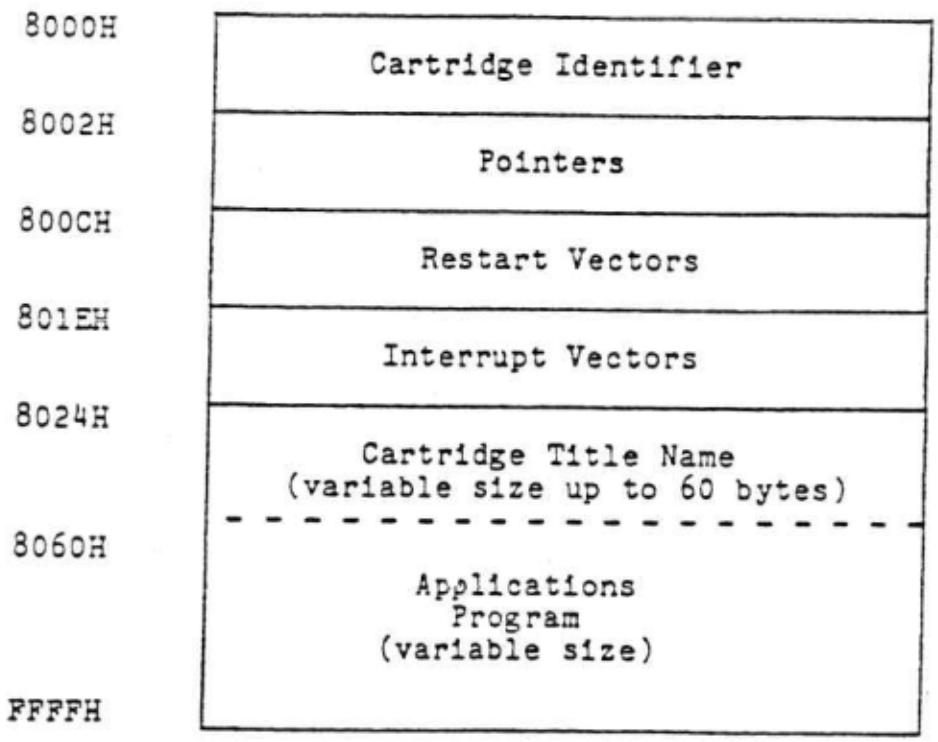


Figure 10-3

CARTRIDGE ROM MAP

10.3 CRAM Areas

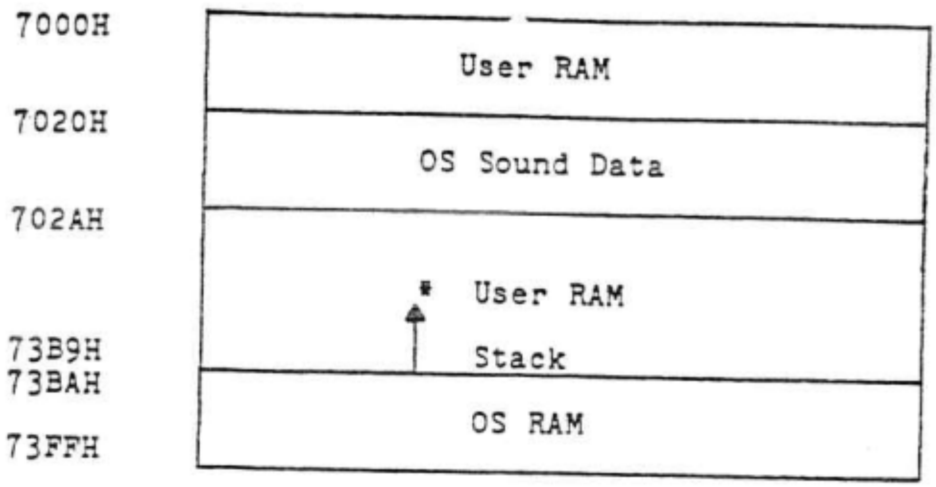


Figure 10-4

CRAM MAP

Figure 10-4 is the CRAM Map. Eleven bytes are reserved for OS sound data starting at 7020H; seventy-one bytes at the high end of memory are used by various OS routines. The top of the stack is sitting at address 73B9H which grows in the decrementing direction. Between stack and user buffer there are 942 bytes available for the application program. However, care should be exercised in both size and boundary when using CRAM as scratch pad.

Table 10-1 lists all reserved CRAM areas for user reference.

Table 10-1
DETAILED CRAM REFERENCE LOCATIONS

	7000H	(Start of user RAM)
PTR_TO_LST_OF_SND_ADDRS	7020H	(OS Sound Data Area)
+1	7021H	
PTR_TO_S_ON_0	7022H	
+1	7023H	
PTR_TO_S_ON_1	7024H	
+1	7025H	
PTR_TO_S_ON_2	7026H	
+1	7027H	
PTR_TO_S_ON_3	7028H	
+1	7029H	
SAVE_CTRL	702AH	
	702BH	(Resume user RAM)
STACK	73B9H	(Top of Stack)
PARAM_AREA	73BAH	(Parameter passing area for
+1	73BBH	Pascal calls to OS routines)
+2	73BCH	
+3	73BDH	

+4	73BEH
+5	73BFH
+6	73C0H
+7	73C1H
+8	73C2H
VDP_MODE_WORD	73C3H
+1	73C4H
VDP_STATUS BYTE	73C5H
DEFER_WRITES	73C6H
MUX_SPRITES	73C7H
RAND_NUM	73C8H
+1	73C9H
QUEUE_SIZE	73CAH
QUEUE_HEAD	73CBH
QUEUE_TAIL	73CCH
HEAD_ADDRESS	73CDH
+1	73CEH
TAIL_ADDRESS	73CFH
+1	73D0H
BUFFER	73D1H
+1	73D2H
TIMER_TABLE_BAS	73D3H

+1	73D4H
NEXT_TIMER_DATA	73D5H
+1	73D6H
DBNCE_BUFF	73D7H (FIRE_OLD - Player 0)
+1	73D8H (FIRE_STATE - Player 0)
+2	73D9H(JOY_OLD - Player 0)
+3	73DAH (JOY_STATE - Player 0)
+4	73DBH(SPIN_OLD - Player 0)
+5	73DCH(SPIN_STATE - Player 0)
+6	73DDH(ARM_OLD - Player 0)
+7	73DEH(ARM_STATE - Player 0)
+8	73DFH(KBD_OLD - Player 0)
+9	73E0H(KBD_STATE - Player 0)
+10	73E1H(FIRE_OLD - Player 1)
+11	73E2H(FIRE_STATE - Player 1)
+12	73E3H(JOY_OLD - Player 1)
+13	73E4H(JOY_STATE - Player 1)
+14	73E5H(SPIN_OLD - Player 1)
+15	73E6H(SPIN_STATE - Player 1)
+16	73E7H(ARM_OLD - Player 1)
+17	73E8H(ARM_STATE- Player 1)
+18	73E9H(KBD_OLD - Player 1)

+19	73EAH	(KBD_STATE - Player 1)
SPIN_SW0_CT	73EBH	
SPIN_SW1_CT	73ECH	
STROBE_FLG	73EDH	
S0_C0	73EEH	
S0_C1	73EFH	
S1_C0	73F0H	
S1_C1	73F1H	
VRAM_ADDR_TABLE	73F2H	
SPRITENAMETBL	73F2H	
+1	73F3H	
SPRITEGENTBL	73F4H	
+1	73F5H	
PATTRNNAMETBL	73F6H	
+1	73F7H	
PATTRNGENTBL	73F8H	
+1	73F9H	
COLORTABLE	73FAH	
+1	73FBH	
SAVE_TEMP	73FCH	
+1	73FDH	
SAVED_COUNT	73FEH	
+1	73FFH	

APPENDIX A

BIBLIOGRAPHY

Z80 Microcomputer Data Book, Mostek, 1981

Z80 Users Manual, Joseph J. Carr, Reward Books, 1980

How to Program the Z80, Rodnay Zaks, Radio Shack, 1979

Z80 Assembly Language Programming, Lance A. Leventhal, Osborne/McGraw-Hill, 1979

TMS 9918A Video Display Processor Data Manual, Texas Instruments, 1980

SN76489AN Data Sheet, Texas Instruments, 1980

APPENDIX B

COLECOVISION

GRAPHICS USERS' MANUAL

11/30/82 V1.0

COLECOVISION
GRAPHICS USERS' MANUAL

Table of contents

1.0 INTRODUCTION

The material in this manual assumes that the reader is familiar with Texas Instruments' 9918/9928 Video Display Processor (VDP). Since the system routines assume that the VDP will be operating in Graphics Mode I or II, particular attention should be given to the discussion of these modes.

The Colecovision operating system includes several graphics routines to help facilitate the creation and manipulation of images on the display. These routines and associated data structures enable the cartridge programmer to treat graphic elements as conceptual entities called "objects", of which there are four types. Each object may consist of one or more "frames". A frame is a single, visual manifestation of the object.

The graphics routines provide a high-level means of altering the frame displayed and the position of objects.

1.1 SUMMARY OF OBJECT TYPES

The four possible object types are:

1. Semi-Mobile

   Semi-Mobile objects are rectangular arrays of pattern blocks They are always aligned on pattern boundaries but may bleed on and off the pattern plane in any direction. Semi-Mobile objects may be used either for moving images (when incremental motion by pattern plane positions is acceptable) or for setting up background patterns.

2. Mobile

   The primary purpose of Mobile objects is to overcome a particular limitation of the VDP which prevents more than 4 sprites from being displayed on a givwn horizontal TV line. Mobile objects are always 2 by 2 pattern blocks in size. They may be placed anywhere on the pattern plane with pixel resolution and will appear as superimposed upon the background. They also may bleed on and off the pattern plane in any direction.

3. Sprite

   Sprite objects are composed of individual sprites.

4. Complex

   Complex objects are collections of other "component" objects. The component objects may be of any type including other Complex objects.

1.2 DATA STRUCTURE OVERVIEW

Each object is defined by a "high level definition" in cartridge ROM (CROM) which links together several different data areas. The data contained within these areas completely specifies all aspects of an object. The following is a brief description of the different areas.

GRAPHICS
This data area is also located in CROM. Pattern and color generators for Semi-Mobile, Mobile and Sprite objects, and frame data for all objects are located in the GRAPHICS data area. The data structure within each GRAPHICS area depends on the type of object with which it is associated. If, however, two or more objects of the same type are graphically identical, they may share the same GRAPHICS data area. This will reduce the amount of graphics data that needs to be stored in CROM.

STATUS
Each object must have its own STATUS area in CPU RAM. The cartridge program uses this area to manipulate the object. This is done by altering the location within STATUS which determines which frame is to be displayed as well as the locations which define the position of the object on the display. The graphics routine, PUT_OBJECT, when called, will access the object's status area and place the object accordingly.

OLD_SCREEN
Mobile and Semi-Mobile objects utilize the pattern plane. They are displayed by overwriting an array of the names in the pattern name table with an array of names which represents a particular frame of the object. The overwritten names can be thought of as representing a background frame which is "underneath" the object. If the background frame will need to be restored to the display (e.g. when the object moves or is removed from the display) then it is necessary to save that frame. OLD_SCREEN is a save area for background frames. The graphics routines PUT_SEMI and PUT_MOBILE will automatically save and restore background frames when an object is moved or its frame is changed. OLD_SCREEN may be located in CRAM or in VRAM.

1.3 GRAPHICS ROUTINES OVERVIEW

ACTIVATE
The primary purpose of this routine is to move the pattern and color generators from the GRAPHICS data area into the pattern and color generator tables in VRAM. Each object must be "activated" before it can be displayed. ACTIVATE also initializes the first byte in an object's OLD_SCREEN data area with the value 80H. PUT_OBJECT tests this location before restoring the background names to the name table. If the value 80H is found, it is an indication that the object has not yet been displayed and therefore, there are no saved background names in OLD_SCREEN. ACTIVATE initializes a byte (in the case of Mobile objects, a word) which indicates where additional generators should be

located if such generators are to be created at game-on time by other routines, such as REFLECT_VERTICAL (described elsewhere in the COLECOVISION USERS MANUAL). Finally, ACTIVATE will initialize the FRAME variable in the object's STATUS area to 0.

PUT_OBJECT
This routine is called when an object's frame or its location on the display is to be changed. The routine tests the type of object and then branches to one of four other routines designed to handle that particular object type. These routines function as follows:

1. PUT_SEMI
Semi-Mobile objects are placed on the display by writing the generator names specified by one of the object's frames into the pattern name table in VRAM. The pattern and color generators which are needed to create the frame must already be in their respective generator tables.

2. PUT_MOBILE
Mobile objects are displayed by producing a new set of pattern and color generators which depict the frame to be displayed on the background. These new generators are then moved to the locations in the VRAM pattern and color generator tables which are reserved for the object; the names of the new generators are then written into the pattern name table.

3. PUT_SPRITE0
4. PUT_SPRITE1
These routines handle the display of size 0 and size 1 Sprite objects.

5. PUT_COMPLEX
Complex objects are aggregates of other "component" objects. The positional relationship of these compontent objects is defined in an offset list. For each of the component objects, PUT_COMPLEX calculates the correct X and Y location, then calls PUT_OBJECT with the address of the high-level definition for that component object passed in the IX register.

1.4 SOME KEY CONCEPTS

META-PLANE
In order to facilitate moving objects on and off the pattern plane, a conceptually larger "meta-plane" has been implemented. Positions on the meta-plane are defined with respect to two orthogonal axes, X and Y. The pattern plane is contained within the meta-plane and its origin is coincident with the origin of the meta-plane (see fig. 1).

X_LOCATION
Y_LOCATION
These two variables are part of each object's STATUS area Each variable contains a 16 bit signed number which represents the distance

in pixels from the origin of the meta-plane. The two variables together form the coordinate location of the object's upper left corner on the meta-plane. Sprite and Mobile objects will be displayed at the exact location indicated by their X and Y_LOCATIONs. However, since Semi-Mobile objects are always aligned on pattern position boundaries, they will be displayed aligned with the nearest pattern boundary above and to the left of the indicated X and Y_LOCATIONs. When a Complex object is to be displayed, the X and Y_LOCATION of each of its component objects is computed by adding a displacement for that component to the Complex object's X and Y_LOCATIONs. Each component object is then displayed at the computed location.

To move an object, the X and or Y_LOCATIONs in the CRAM STATUS area are changed and then PUT_OBJECT is called with the address of the object's high-level definition passed in the IX register. When moving Mobile objects an additional parameter is passed in register B. This is discussed further in the description of PUT_MOBILE.

FRAME
This variable is also part of each object's STATUS area. The value contained in FRAME is used by PUT_OBJECT to select one of several pointers (or, in the case of Complex objects, pairs of pointers) which point to the data defining the graphic content of the frame. The pointers may either point to frame data which is part of the original ROMed GRAPICS, or they may point to an area in CPU RAM. In the latter case, the frame data must be created by the cartridge program before that frame can be displayed.

The frame of an object is changed by altering the FRAME variable in the object's STATUS area. PUT_OBJECT is then called in the same manner as when moving an object. When changing the frame number of a Mobile object, bit 7 of FRAME should not be altered. This bit is reserved for use by the PUT_MOBILE routine.

[page 5 is missing]

Since Semi-Mobile objects are displayed by altering names in the pattern name table, it may be necessary to save the pattern plane graphic which is lost in the process of displaying the Semi-Mobile object (see previous discussion of OLD_SCREEN). The third address in the object's high level definition designates a location for saving the overwritten names. If that address is greater than or equal to 7000H, then the OLD_SCREEN will reside in CPU RAM. If the address is less than 7000H, then OLD_SCREEN will be in VRAM. Finally, if the address is 8000H or greater, then the overwritten names will not be saved.

2.3 DETAILED DESCRIPTION OF SEMI-MOBILE OBJECT DATA STRUCTURE

(Identifiers in caps refer to symbols in the data structure summary)

Each Semi-Mobile object is defined in cartridge ROM by:
  1) SMO - the object's "high level definition"
  2) GRAPHICS - an area containing the object's graphics data, divided into three subsections:
     Parameters and Pointers
     Frames
     Generators

and in CPU RAM by:
  1) STATUS - a status area
  2) OLD_SCREEN - an optional location for saving overwritten backgounds. (OLD_SCREEN may alternatively be stored in VRAM)

A detailed description of each structure follows.

*** SMO - cartridge ROM

The high level definition, at SMO, is composed of three 16 bit addresses; these are stored in the normal manner with the low order byte first:

byte:

0 address of GRAPHICS (the start of the ROMed graphics data for the object)

2 address of STATUS (the object's RAM status area)

4 address of OLD_SCREEN (VRAM or CPU RAM area for saving background information; bit 15 of the OLD_SCREEN address is a flag indicating whether or not backgrounds are to be saved. If bit 15 is set, backgrounds will not be saved).

*** GRAPHICS - cartridge ROM

The ROMed graphics data for a Semi-Mobile object can be thought of conceptually as three chunks. Each chunk is stored as follows:

*** Parameters and Pointers

byte:

- Parameters -

0 OBJ_TYPE - OBJ_TYPE is divided into two parts:

LSN - specifies the object type which, for Semi-Mobile objects, must equal 0.

MSN - only meaningful when the VDP is operating in graphics mode II. Bits 5, 6 and 7 indicate which third or thirds of the pattern plane ~~the object~~ (or any part of the object) may be required to appear. This information is used by routines which move the object's pattern and color generator data to VRAM.

bit 7 - if set, generators will be moved to the 1st third of the generator tables.

bit 6 - if set, generators will be moved to the 2nd third.

bit 5 - if set, generators will be moved to the 3rd third.

bit 4 - indicates the number of color generator bytes per 8 byte pattern generator which are included as part of the ROM graphics data. If this bit is 0, then there must be 8 color generator bytes per pattern generator (the normal case for graphics mode II). If bit 4 is 1, then only 1 color generator byte per pattern generator will be expected, giving the programmer the option of reducing the number of ROMed color generator bytes needed when operating in graphics mode II. The single color byte will be expanded to 8 bytes by the routine which moves the color generators to VRAM.

1 FIRST_GEN_NAME - an index from the start of the pattern and color generator tables in VRAM. This index specifies the location to which the ROMed pattern and color generators will be moved (i.e. the base address of the pattern generator table + 8 * FIRST_GEN_NAME = the address within the pattern generator table where the object's 1st pattern generator will be stored. This is also true for the color generator table. The first pattern generator will be moved to the location in the pattern generator table indexed by FIRST_GEN_NAME and the rest of the generators will be loaded sequentially. The color generators are loaded in a similar fashion.

2 NUMGEN - indicates how many pattern/color generator pairs are defined (stored) in the graphics data area. This is equal to the number of generator pairs which will be moved to the VRAM generator tables.

- Pointers -

3 address of GENERATORS - the start of the ROMed pattern and color generators in the object's graphics data area. Color generators must be stored immediately after pattern generators; therefore, the address of the first color generator within the graphics data area can be computed from NUMGEN and GENERATORS.

5 address of FRAME_0 - FRAME_0 is the address at which the data specifying the object's first frame is stored. As indicated by the frame address, the data for a given frame may be stored in ROM (as part of the graphics data area) or it may be stored in CPU RAM. RAM storage of frame data allows for the algorithmic generation of additional frames at game on time: e.g., rather than storing both a frame and its rotated version in ROM, ROM space can be saved by storing only one frame and generating the rotated frame data in RAM. Also, frame data stored in RAM can be dynamically modified, allowing for special frame effects (e.g. color modification).

7 address of FRAME_1 - the address at which the data specifying the object's second frame is stored.

:

:

2n+5 address of FRAME_n - the object's last frame

*** Frames - cartridge ROM or CPU RAM

Since the frame pointers (FRAME_0...FRAME_n) are 16 bit addresses, the frame data for an object may be located anywhere in cartridge ROM or RAM. The format of a frame's data is as follows:

byte:

0 X_EXTENT - specifies the width of the frame in pattern plane positions, its "X_EXTENT", which must be > 0 and <= 255. As indicated by the range of values for the X_EXTENT, a frame may be much greater in width than can be displayed within the pattern plane. When a frame with a X_EXTENT which is greater than 32 is to be displayed, the section actually visible will depend on the specified X position (i.e. if a frame of an object has a X_EXTENT of 64 and the X position of the object is -8*32 pixels, then the right half of the frame will be displayed on the pattern plane). This feature allows objects to be scrolled horizontally by creating a frame greater in width than the pattern plane and then displaying the object with varying values of the X position.

1 Y_EXTENT - specifies the height of the frame in pattern plane positions, its "Y_EXTENT", which must be > 0 and <= 255. As indicated by the range of values for the Y_EXTENT, a frame may be much greater in height than can be displayed within the pattern plane. When a frame with a Y_EXTENT which is greater than 24 is to be displayed, the section actually visible will depend on the specified Y position (i.e. if a frame of an object has a Y_EXTENT of 48 and the Y position of the object is -8*24 pixels, then the bottom half of the frame will be

displayed on the pattern plane). This feature allows objects to be scrolled vertically by creating a frame greater in height than the pattern plane and then displaying the object with varying values of the Y position.

2... for X_EXTENT * Y_EXTENT bytes
Following the X and Y_EXTENT is an array of the pattern names (length in bytes = X_EXTENT * Y_EXTENT) which specify the generators used to create that frame. The names are arranged in row major order (i.e. the first X_EXTENT names are the names of the generators which will be displayed in the first row of the frame, the second X_EXTENT names are the names of the generators which will be displayed in the second row of the frame, etc.). There must be exactly X_EXTENT by Y_EXTENT names in the array.

For Semi-Mobile objects the names stored in the above described name list are "names" in the TI sense of the word: i.e., they are values ranging from 0 to 255 that directly point to (or "name") a generator location within the pattern generator table.

*** Generators - cartridge ROM

All the ROMed pattern and color generators must be grouped together in a contiguous block (starting at location GENERATORS). Each pattern generator must be 8 bytes long, and the number of pattern generators must conform to the value stored in NUMGEN:

byte:
0 bb,bb,bb,bb,bb,bb,bb,bb - the first 8 byte pattern generator (bb = binary graphic data)
8 bb,bb,bb,bb,bb,bb,bb,bb - the second 8 byte pattern generator
etc. for a total of 8*NUMGEN bytes

The color generators are stored immediately following the pattern generators. The format of the color generators depends on which graphics mode is being used and whether bit 4 of OBJ_TYPE is set or not.

GRAPHICS MODE II color generator storage:

When OBJ_TYPE bit 4 = 0, there must be eight color generator bytes per pattern generator.

byte:
0-7 bb,bb,bb,bb,bb,bb,bb,bb - color generator bytes for 1st pattern
8-15 bb,bb,bb,bb,bb,bb,bb,bb - color generator bytes for 2nd pattern
etc., for a total of 8*NUMGEN bytes

When OBJ_TYPE bit 4 = 1, there must be only 1 color generator byte per pattern generator. It will be expanded to 8 bytes when moved to the VRAM color generator table. This feature is useful if each generator for a particular object can use the same ~~two~~ color combination for all 8 lines.

byte:
0 bb - color generator byte for 1st pattern
1 bb - color generator byte for 2nd pattern
etc., for a total of NUMGEN bytes.

GRAPHICS MODE I color generator storage:

In Graphics Mode I the pattern generator table is divided into 32 "groups" of 8 (contiguous) generators (see TI VDP manual page 18). All the generators within a group share the same color generator byte. Therefore, there must be one color generator byte stored per group occupied by the object's pattern generators. The first color generator byte will be placed in the color generator table at an offset of FIRST_GEN_NAME/8 and the rest will be placed in sequential locations.

NOTE 1: The exact number of color generator bytes needed by an object depends both on the number of pattern generators contained in the graphics data and the location in the pattern generator table to which the generators will be moved. For example, if an object has 8 pattern generators and they are moved to the pattern generator table starting at location 0 (offset from the start of the table), then only 1 color generator is needed. However, if the same 8 pattern generators are loaded into the pattern generator table starting at location 20H, then 2 color generators will be needed, since the first four generators are in one group and the second four are in another group.

NOTE 2: Due to an error in the ACTIVATE routine, ACTIVATE cannot be used to move pattern and color generators of Semi-Mobile objects to VRAM when the VDP is operating in graphics mode I, and when FIRST_GEN_NAME is equal to or greater than 80H. In this situation the cartridge program must fulfill the functions of ACTIVATE (see discussion of ACTIVATE).

*** STATUS - CPU RAM

An object's status area consists of 4 elements:

byte:
0 1 byte for FRAME - indicates which of the object's frames is to be displayed.

1 2 bytes for X_LOCATION

3 2 bytes for Y_LOCATION - X_LOCATION and Y_LOCATION give the coordinate position of the upper left corner of the object on a "metaplane" which includes the pattern plane (see fig. 1). The origin of the pattern plane is coincident with the origin of the metaplane. The values at X_LOCATION and Y_LOCATION are 16 bit signed numbers which permit the positioning of an object anywhere within, or outside of, the pattern plane. This enables an object to be bled on or off the pattern plane in any direction.

5 1 byte for NEXT_GEN - an index which points to the next generator location which can be used when adding new generators to an object's generator tables. After the object's ROMed generators have been moved to its VRAM tables, ACTIVATE will set NEXT_GEN equal to FIRST_GEN_NAME + NUMGEN.

Some objects may require generators which are essentially modified versions of other generators (e.g. a generator which represents the pattern of another generator which has been rotated). ROM space can be conserved by including only the "unmodified" generators in the graphics data and using system routines to generate the modified versions. NEXT_GEN points to the next free location in that object's generator table to which a modified generator may be added. When adding new generators in this manner, NEXT_GEN should be updated in order to prevent new generators overwriting old ones.

*** OLD_SCREEN - VRAM or CPU RAM

OLD_SCREEN is a buffer for saving the pattern names which are overwritten in the process of displaying an object. These names constitute a "background frame" which has the same X and Y_EXTENTs as the frame of the object which is currently being displayed. The data structure within OLD_SCREEN is the same as the data structure for a frame with two extra bytes tacked on at the beginning. These bytes, X_PAT_POS and Y_PAT_POS, indicate where on the pattern plane this background frame belongs, and are expressed in terms of pattern plane positions. The next time the position or frame of this object is changed PUT_OBJECT will restore the background frame to the display and save a "new" background frame before placing the object on the pattern plane.

byte:

0 1 byte for X_PAT_POS - the column in which the upper left corner of the saved background screen (frame) lies

1 1 byte for Y_PAT_POS - the row in which the upper left corner of the saved background screen (frame) lies

2 1 byte for X_EXTENT of the saved screen - set by PUT_OBJECT to equal X_EXTENT of the frame which eclipses it

3 1 byte for Y_EXTENT of the saved screen - set by PUT_OBJECT to equal Y_EXTENT of the frame which eclipses it

4 n bytes for storage of pattern names; where n = the number of patterns contained in the largest frame of the object (i.e. n = X_EXTENT * Y_EXTENT for the object's largest frame)

3.0 MOBILE OBJECTS

Mobile objects emulate size 1 sprites with 0 magnification (i.e. they are always 16 by 16 pixels in size and appear as if they were superimposed upon the background). They may be positioned anywhere on the pattern plane with pixel resolution and may also be made to bleed on and off the pattern plane in any direction.

Each frame of a Mobile object requires exactly four pattern generators and one color generator. These generators, however, are not moved to VRAM. In order to display the object anywhere with pixel resolution, a new set of nine pattern and color generators must be created by the OS graphics routine PUT_MOBILE. These new generators represent graphically the superposition of the Mobile object upon the background at which it is to be displayed. The new generators are then moved to the VRAM pattern and color generator tables, and next the names of these generators are written into the pattern name table, thus displaying the object at the desired location. Nine generators are used in order to cover all positional relationships between the desired position of the object and the boundries of the pattern positions in the pattern plane (see figure 1).

The pattern and color generator table space used by PUT_MOBILE depends on the graphics mode being used and, in graphics mode II, the location of the object on the display.

In graphics mode I, PUT_MOBILE will use 18 pattern generator locations. The first pattern generator will be located at FIRST_GEN_NAME (i.e. the actual VRAM address will be the pattern generator base address + FIRST_GEN_NAME * 8). Three or four color generator bytes will be required depending on the value of FIRST_GEN_NAME. If FIRST_GEN_NAME MOD 8 < 7, then there needs to be space for three color generators; if FIRST_GEN_NAME MOD 8 = 7, then there needs to be space for four color generators. The first of the color generators will be located at FIRST_GEN_NAME/8 (i.e. VRAM address equals the color table base address + FIRST_GEN_NAME/8).

When operating in graphics mode II, Mobile objects will require generator space for 18 pattern and 18 color generators in each third of the pattern and color generator tables which corresponds to that third of the pattern plane in which any part of the object may appear. The location within each third of the tables is determined by FIRST_GEN_NAME. When any part of the object is in the top third of the pattern plane, pattern generators will be located at the VRAM address given by the pattern generator base address + FIRST_GEN_NAME * 8. If any part is in the middle third of the pattern plane, pattern generators will be located at the VRAM address given by the pattern generator base address + 800H + FIRST_GEN_NAME * 8, and if any part is in the bottom third, pattern generators will be located at the VRAM address given by the pattern generator base address + 1000H + FIRST_GEN_NAME * 8. The color generators are located in a similar manner.

Even though only 9 pattern and 9 color generators (2 color generator bytes in graphics mode I) are "active" (being displayed) at a given time, the additional generator space is required to enable PUT_MOBILE to move new sets of pattern and color generators to VRAM without disturbing the display. While one set of 9 pattern and color generators are being displayed, the other set of 9 can be changed. After the change is completed, the new generators are displayed by writing the new pattern names into the pattern name table.

Each Mobile object requires a STATUS area in CPU RAM which contains the frame of the object to be displayed, its location on the display, and a pointer to the area for adding new generators (these new generators are used in the same manner as the object's ROMed generators). Each object also requires an OLD_SCREEN area which serves the same purpose as OLD_SCREEN areas for Semi-Mobile objects.

Even though a Mobile object's generators are not moved directly to VRAM, each object must be "activated" in order to initialize the OLD_SCREEN and STATUS areas.

To place a Mobile object on the display, the desired location should be loaded into the X and Y_LOCATION variables in its STATUS area. In addition, the FRAME variable must contain the desired frame number. When loading the frame number, however, only bits 0-6 should be used. Bit 7 should not be altered. This bit is used by PUT_MOBILE (see description of PUT_MOBILE).

PUT_OBJECT is then called, passing the address of the high-level definition in the IX register. In addition, register B is used to pass two parameters which determine the method for combining the background graphics with that of the object (see description of PUT_MOBILE).

3.1 DETAILED DESCRIPTION OF MOBILE OBJECT DATA STRUCTURE

(Identifiers in caps refer to symbols in the data structure summary)

Each Mobile object is defined in cartridge ROM by:
  1) MOB - the object's "high level definition"
  2) GRAPHICS - an area containing the object's graphic data, divided into three subsections:
    Parameters and pointers
    Frames
    Generators

and in CPU RAM by:
  1) STATUS - a status area
  2) OLD_SCREEN - a location for saving overwritten background names (OLD_SCREEN may be either in CPU RAM or VRAM).

A detailed description of each structure follows.

*** MOB - cartridge ROM

The high level definition at MOB is composed of four 16 bit addresses stored in the normal manner with the low order byte first:

byte:

0	address of GRAPHICS (the start of the ROMed graphics data for the object). Any number of Mobile objects may use the same graphics data. However, each Mobile object must have its own STATUS and OLD_SCREEN areas.
2	address of STATUS (the object's RAM status area)
4	address of OLD_SCREEN (VRAM or CPU RAM area for saving background information)
6	FIRST_GEN_NAME (index to the start of the object's generator tables within the VRAM pattern and color generator tables)

*** GRAPHICS - cartridge ROM

The ROMed graphics for Mobile objects may be thought of as three separate segments. The data in each segment is defined as follows:

*** Parameters and Pointers

byte:

- Parameters -

0	OBJ_TYPE - for Mobile objects OBJ_TYPE = 1
1	NUMGEN - indicates how many pattern generators are contained within the graphics data area.
2	address of NEW_GEN (an area for storing new generators created at game-on time)
4	address of GENERATORS - the start of the ROMed pattern generators for the object

- Pointers -

6	address of FRAME_0 - this is the address of the start of the data for the object's first frame. This data, as well as the data for any of the object's frames, may be located anywhere in cartridge ROM or in CPU RAM. Frame data stored in RAM allows for the creation of new frames at game-on time and therefore reduces the amount of data which needs to be stored as part of the object's ROMed frame data.
8	address of FRAME_1 - address of the data for the object's second frame.
:	
:	
2n+6	address of FRAME_n - the object's frame.

*** Frames - each frame may be either in cartridge ROM or CPU RAM

FRAME_0...FRAME_n - cartridge ROM or CPU RAM

Since the frame pointers (FRAME_0...FRAME_n) are 16 bit addresses, the frame data for an object may be located anywhere in cartridge ROM or CPU RAM. The format for a Mobile object's frame data is as follows:

byte:

0 list of 4 pattern names. The four names specify which of the object's patterns are to be displayed in each of the object's four quadrants as follows:

name	quadrant
1	upper left
2	lower left
3	upper right
4	lower right

The value of the name specifies the object's generators as follows:

0 = first ROMed pattern
1 = second ROMed pattern
:
NUMGEN-1 = last ROMed pattern

Name values greater than or equal to NUMGEN refer to patterns stored in the location starting at NEW_GEN in CPU RAM. A value of NUMGEN refers to the first pattern in that area. A value of NUMGEN + 1 refers to the second pattern etc.
The MSN of this byte determines the color of the object (i.e. the color1 of the object) and the LSN must be 0.

*** Generators - cartridge ROM

All of a Mobile object's pattern generators must be grouped together in a contiguous block starting at location GENERATORS. Each pattern generator must be 8 bytes long. The number of generators must equal the number stored in NUMGEN:

byte:

0 bb,bb,bb,bb,bb,bb,bb,bb - first generator (bb = binary graphic data)
8 bb,bb,bb,bb,bb,bb,bb,bb - second generator
for a total of 8*NUMGEN bytes

*** STATUS - CPU RAM

A Mobile object's status area consists of 4 elements

byte:

0 FRAME - bits 0-6 indicate which frame is to be displayed. Bit 7 is used by the PUT_MOBILE routine and should not be altered when changing the frame number.

1 X_LOCATION

3 Y_LOCATION - The X and Y_LOCATION variables specify the coordinate position of the upper left corner of the object on a "metaplane" which includes the pattern plane.

5 pointer to NEW_GEN area - This pointer points to the next available space for adding new generators. It will be initialized by ACTIVATE with the 16 bit value NEW_GEN from the parameter segment of the object's GRAPHICS area. Routines which add generators to this area should increment the pointer by 8 each time a new generator is added if subsequent generators are not to overwrite previous ones.

*** OLD_SCREEN - VRAM or CPU RAM

OLD_SCREEN is an area for saving pattern names which are overwritten in the process of displaying the object. Since all Mobile objects are displayed by writing 9 names into the patten name table (except in cases in which part of the object is off the pattern plane) the size of the OLD_SCREEN area for any Mobile object is always 11 bytes. The first two bytes specify where (in pattern plane positions) the names came from and the next 9 bytes contain the 9 saved names. ACTIVATE initializes the first byte to 80H which indicates to PUT_MOBILE that no names have yet been saved.

byte:

0 X_PAT_POS - pattern position column in which the upper left corner of the saved "background frame" lies

1 Y_PAT_POS - pattern position row in which the upper left corner lies

2-11 the nine background names

4.0 SPRITE OBJECTS

Sprite objects are composed of individual sprites. They may be either size0 or size1 sprites, and may or may not be magnified.

The data areas that make up sprite objects are similar to those for the other object types. The high level definition contains two addresses which point to the object's GRAPHICS data and the STATUS area respectively. Following these addresses there is a byte which determines which of the 32 VDP sprites will be used to implement this object.

Each Sprite object must include a set of pattern generators which will be used to create an image on the display. These generators may be stored as part of the object's GRAPHICS data in ROM and moved to the sprite generator table in VRAM. The location within the sprite generator table to which the generators should be moved is determined by FIRST_GEN_NAME, a byte within the GRAPHICS data area (i.e. the first generator should be located at VRAM address = sprite generator base address + FIRST_GEN_NAME * 8).

Frames for a Sprite object are defined in the FRAME_TABLE. The FRAME_TABLE contains a pair of bytes for each frame of the object. The first byte specifies the color that the frame will be and the second byte determines which generator (or generators in the case of size1 sprites) will be used to define the shape.

Once the pattern generators have been moved to VRAM, a Sprite object may be displayed by setting up it's STATUS area and then calling PUT_OBJECT. To set up the STATUS area, the following must be done:

1. Set the first byte, FRAME, to the desired frame number to be displayed. This number is used to pick one of the pairs of bytes in the FRAME_TABLE which determines the color and shape to be displayed.

2. The 16 bit signed values at X_LOCATION and Y_LOCATION must be set to the desired x and y pixel positions of the upper left corner of the object.

To place the Sprite object on the screen, the following calling sequence is used:

```
LD IX,OBJECT_NAME
CALL PUT_OBJECT
```

Where OBJECT_NAME is the address of the high level definition for the object.

4.1 DETAILED DESCRIPTION OF SPRITE OBJECT DATA STRUCTURE

(Identifiers in caps refer to symbols in the data structure summary)

Each Sprite object is defined in cartridge ROM by:
1) SPROBJ - the object's "high level definition"
2) GRAPHICS - an area containing the object's graphics data, divided into three subsections:
   Parameters and Pointers
   Frame_Table
   Generators

and in CPU RAM by:
1) STATUS - a status area

A detailed description of each structure follows.

*** SPROBJ cartridge ROM

The high level definition, at SPROBJ, is composed of two 16 bit addresses stored in the normal manner with the low order byte first. Following the addresses is a byte which indicates which actual sprite number is used to implement the object.

byte:

0	address of GRAPHICS (the start of the ROMed graphics data for the object)
2	address of STATUS (the object's RAM status area)
4	SPRITE_INDEX (determines the sprite to be used for this object, i.e. 0-31)

*** GRAPHICS - cartridge ROM

The ROMed graphics data for a Sprite object can be thought of in three conceptual chunks. Each chunk is stored as follows:

*** Parameters and Pointers

byte:

0 OBJ_TYPE - OBJ_TYPE is equal to 3 for all Sprite objects.

1 FIRST_GEN_NAME - an index into the sprite pattern generator table in VRAM. This index specifies the location to which the ROMed pattern generators will be moved (i.e. the base address of the sprite pattern generator table + 8 * FIRST_GEN_NAME = the address within the pattern generator table where the object's 1st pattern generator will be stored). The first pattern generator will be moved to the location in the pattern

generator table indexed by FIRST_GEN_NAME and the rest of the generators will be loaded sequentially. When using size1 sprites, FIRST_GEN_NAME must be a multiple of 4.

3 address of GENERATORS - the start of the ROMed pattern generators in the object's graphics data area.

4 NUMGEN - indicates how many pattern generators are defined (stored) in the graphics data area. This is the number of generators which will be moved to the VRAM generator table.

5 address of FRAME_TABLE - This is a pointer to the table containing shape and color information for each frame of the object.

*** FRAME_TABLE - cartridge ROM or CPU RAM

The FRAME_TABLE contains a pair of bytes for each frame. The first byte of each pair determines the color and the second byte points to the generator (or set of four generators in the case of size1 sprites) to be used for that frame.

byte:

0	COLOR for frame 0
1	SHAPE - index to generator(s) for frame 0, e.g. the VRAM address of the generator(s) for this frame = sprite generator base address + 8 * (FIRST_GEN_NAME + SHAPE). When using size1 sprites, the value of SHAPE must be a multiple of 4.
2	COLOR for frame 1
3	SHAPE for frame 1
:	: :
2n	COLOR for frame n
2n+1	SHAPE for frame n

*** GENERATORS - cartridge ROM

All the ROMed pattern generators must be grouped together in a contiguous block (starting at location GENERATORS). Each pattern generator must be 8 bytes long, and the number of pattern generators must conform to the value stored in NUMGEN:

byte:

0	bb,bb,bb,bb,bb,bb,bb,bb - the first 8 byte pattern generator (bb = binary graphic data)
8	bb,bb,bb,bb,bb,bb,bb,bb - the second 8 byte pattern generator
etc.	for a total of 8*NUMGEN bytes

*** STATUS - CPU RAM

An object's status area consists of 4 elements:

byte:

0 1 byte for FRAME - indicates which of the object's frames is to be displayed.

1 2 bytes for X_LOCATION

3 2 bytes for Y_LOCATION - X_LOCATION and Y_LOCATION give the coordinate position of the upper left corner of the object on a "metaplane" which includes the pattern plane (see fig. 1). The origin of the pattern plane is coincident with the origin of the metaplane. The values at X_LOCATION and Y_LOCATION are 16 bit signed numbers which permits the positoning of an object anywhere within or outside of the pattern plane. This enables an object to be slid on or off the pattern plane in any direction.

5 1 byte for NEXT_GEN - an index which points to the next generator location which can be used when adding new generators to an object's generator tables. After the object's ROMed generators have been moved to its VRAM tables, ACTIVATE will set NEXT_GEN to equal FIRST_GEN_NAME + NUMGEN.

Some objects may require generators which are essentially modified versions of other generators (e.g. a generator which represents the pattern of another generator which has been rotated). ROM space can be conserved by including only the "unmodified" generators in the graphics data and using system routines to generate the modified versions. NEXT_GEN points to the next free location in that object's generator table to which a modified generator may be added. When adding new generators in this manner, NEXT_GEN should be updated in order to prevent new generators overwriting old ones.

5.0 COMPLEX OBJECTS

Complex objects are aggregates of other "component" objects which may be of any type (including other Complex objects) except Mobile objects. This object type gives the game programmer the ability to combine several objects in order to create a higher order graphic entity. Complex objects may have multiple frames and may be moved about on the display in the same manner as any other object.

Before a Complex object can be displayed, all of its component objects must be activated. This can be done by activating the complex object itself (ACTIVATE will then process all the component objects). However, if any of the component objects are type Semi-Mobile and the VDP is operating in graphics mode I, then all the component objects must be individually activated. If ACTIVATE is not used, then in addition to moving the generators to VRAM, the FRAME variable in the STATUS area for each of the component objects must be initialized to 0.

Once all the component objects are activated, a Complex object may be placed on the display in the same manner as any other object type. The object's STATUS area is initialized with the desired frame number and position, and then PUT_OBJECT is called. PUT_OBJECT will then determine the correct frame number and position for all of the component objects and place them on the display.

Each frame of the Complex object points to a list of frame numbers and a list of offsets. The list of frame numbers specifies the frame number to be used for each of the component objects. The list of offsets specifies the positional offset of each of the component objects from the Complex object's X and Y_LOCATIONs.

5.1 DETAILED DESCRIPTION OF COMPLEX OBJECT DATA STRUCTURE

Complex objects are defined in cartridge ROM by:

1) COM_OB - the object's high level definition

2) GRAPHICS - which contains frame and offset parameters. For each frame of the Complex object, these parameters define the frame numbers for the component objects and the positional relationship between the component objects.

and in CPU ram by:

1) STATUS - which contains the frame and position variables for the object.

The following is a detailed description of the data structure:

*** COM_OB - cartridge ROM

The high level definition of a Complex object contains pointers to the GRAPHICS and STATUS areas for the object, and a list of addresses pointing to the high level definition of the component objects.

byte:	
0	address of GRAPHICS (the start of the ROMed graphics data for the object)
2	address of STATUS (the object's RAM status area)
4	address of the 1st component object high level definiton
6	" " " 2nd " " " " "
:	:
2n+2	" " " nth " " " " "

*** GRAPHICS - cartridge ROM

The graphics segment of a Complex object can be thought of as divided into three sections. The first section contains the following:

byte:	
0	OBJ_TYPE - This byte is divided into two parts: LSN - specifies the object type and must be equal to 4 for complex objects. MSN - contains the number of component objects that make up the complex object.
1	pointer to FRAME_0 (list of frame numbers)
3	pointer to OFFSET_0 (list of offsets)
5	pointer to FRAME_1
7	pointer to OFFSET_1
:	: :
2n+1	pointer to FRAME_n
2n+2	pointer to OFFSET_n

*** FRAME_0 ... FRAME_n - cartridge ROM or CPU RAM

Each frame of a complex object specifies the frame numbers to be used for each of its component objects. The frame numbers to be used for any given frame are arranged in a list; the first entry being the frame number for the first component, the second entry the frame number for the sencond component, etc. There may be as many frame lists as there are frames, or several or all frames may share the same frame list. For example, if the only difference between frames of a complex object were the positional relationship of the component objects, then one frame list would be sufficient. The format of the frame list is as follows:

byte:	
0	frame number for 1st component
1	" " " 2nd "
:	:
n-1	" " " nth "

*** OFFSET_0 ... OFFSET_n - cartridge ROM or CPU RAM

Each frame of a Complex object also must have an offset list. This list determines the position of each of the component objects with respect to the position of the Complex object (its X and Y_LOCATIONs). Each entry in the offset list has two components, a one byte X displacement followed by a one byte Y displacement. These displacements are unsigned 8 bit integers and are added to the 16 bit values contained in the Complex object's X and Y_LOCATIONs to form the coordinate position for each of the component objects. As with the frame lists, there may be as many offset lists as there are frames, or fewer if several or all frames use the same offset list. The format of the offset lists is as follows:

byte:	
0	X displacement for the 1st component object
1	Y " " " 1st " "
2	X " " " 2nd " "
3	Y " " " 2nd " "
:	:
2n	X " " " nth " "
2n+1	Y " " " nth " "

*** STATUS - CPU RAM

The STATUS area for a Complex object is similar to the STATUS area for any other object type. The only difference is that the 5th byte, NEXT_GEN, is not used.

byte:	
0	FRAME - the value in this byte indicates the frame to be displayed.
1	X_LOCATION - a two byte value determining the X position of the Complex object on the display.
3	Y_LOCATION - a two byte value determining the Y position of the Complex object on the display.

6.0 ACTIVATE

Calling sequence:

```
LD HL,OBJECT_NAME
SCF                    ; MOVES GENERATORS TO VRAM
CALL ACTIVATE
 :

-or-

 :
LD HL,OBJECT_NAME
OR A                   ; DOESN'T MOVE GENERATORS TO VARM
CALL ACTIVATE
 :
```

Registers used:

Uses all registers

RAM Used (starting at WORK_BUFFER):

If the object type is Semi_Mobile, the VDP is operating in Graphics Mode II, and bit 4 of OBJ_TYPE = 1, then 8 bytes starting at WORK_BUFFER are used. Otherwise no RAM is used.

6.1 DESCRIPTION OF ACTIVATE ROUTINE

The primary purpose of the ACTIVATE routine is to move the pattern and color generators from an object's GRAPHICS data area to the locations in the VRAM pattern and color generator tables assigned to it. In addition ACTIVATE will initialize certain variables in the object's STATUS and OLD_SCREEN areas. How the routine actually functions is dependent on the type of object being "activated", the graphics mode being used, and the state of the carry flag.

1) Activating Semi-Mobile objects

If the third address pointer in the object's high level definition is less than 8000H, then that address is taken to be the address of an OLD_SCREEN area for that object. The first byte in the OLD_SCREEN area is initialized with an 80H. This value indicates that no data has yet been saved in OLD_SCREEN. When PUT_OBJECT sees this value it will not attempt to restore the contents of OLD_SCREEN to the display.

The first byte in the object's STATUS area, the FRAME variable, is set to 0.

The 6th byte in the object's STATUS area will be initialized with a value equal to FIRST_GEN_NAME + NUMGEN. Routines which add new generators to the object's pattern and color generator tables may access this byte to determine where to put them.

If the carry flag is set when ACTIVATE is called, then the object's pattern and color generators will be moved to the pattern and color generator tables in VRAM. If the carry flag is not set, then the generators will not be moved; in this case only the STATUS and OLD_SCREEN areas will be affected. This feature is used when several objects share the same generators. ACTIVATE needs to be called with the carry flag set for only one of the objects, to get the generators to VRAM, and the other objects are ACTIVATEd with the carry flag not set to prevent the generators from being moved again. The generators are moved as follows:

a) In Graphics Mode I
All the pattern generators (the number of which is given by the NUMGEN entry) are moved to the pattern generator table in VRAM. The first generator is moved to a location within the table specified by FIRST_GEN_NAME (i.e. the VRAM address to which the first pattern generator is moved = pattern generator base address + 8 * FIRST_GEN_NAME). The others are loaded sequentially.

In Graphics Mode I the pattern generator table is divided into 32 eight-byte blocks. The color for each block of 8 generators is defined by one color generator byte. Therefore, ACTIVATE needs to determine the number of color generator bytes that must be moved to the VRAM color generator table. It does this in the following manner (see NOTE below):

The first pattern generator will require a color generator byte at FIRST_GEN_NAME/8 offset from the start of the color generator table. The last pattern generator will require a color generator byte at (FIRST_GEN_NAME + NUMGEN -1)/8. Therefore, the total number of color generators needed will be (FIRST_GEN_NAME + NUMGEN -1)/8 - FIRST_GEN_NAME/8 +1. ACTIVATE will move this number of color generator bytes to the color generator table starting at VRAM address = color table base address + FIRST_GEN_NAME/8.

b) In Graphics Mode II
In this mode the pattern and color generator tables are divided into three sections. Each section contains the generators which will be displayed in the corresponding third of the pattern plane. A Semi-Mobile object needs to have it's generators moved to one or more sections depending on which thirds of the pattern plane it may appear in. The MSN of OBJ_TYPE in the object's GRAPHICS data area indicates which sections of the generator tables the pattern and color generators should be moved to as follows:

if bit 7 = 1 then move generators to the 1st section
if bit 6 = 1 then move generators to the 2nd section
if bit 5 = 1 then move generators to the 3rd section

The starting location within each section to which the generators will be moved is specified by FIRST_GEN_NAME. The first pattern/color generator will be located at FIRST_GEN_NAME and the rest will be loaded sequentially (e.g. if the MSN of a Semi-Mobile object = 1010B, then the pattern generators will be moved to VRAM address = pattern base address + FIRST_GEN_NAME * 8, and also to VRAM address = pattern base address + 1000H + FIRST_GEN_NAME * 8).

Bit 4 of OBJ_TYPE indicates whether there are 8 or 1 color generator bytes per pattern generator. If this bit is 0, then ACTIVATE will expect 8 color generator bytes per pattern generator. If bit 4 is 1, then only one color generator byte will be expected. This byte will be used to fill all eight bytes of the appropriate color generator in VRAM. The location within the color table to which the generators are moved is determined in the same fashion as the pattern generators (see above).

2) Activating Mobile objects

The first byte of the object's OLD_SCREEN area is initialized with 80H.

The frame variable, the first byte in the object's STATUS area, is initialized to 0.

Bytes 6 and 7 of the object's STATUS area are initialized with the value NEW_GEN from the object's GRAPHICS data. This value is used as a pointer to the beginning of an area reserved for the addition of generators created at game-on time.

3) Activating Sprite objects

The frame variable, the first byte in the object's STATUS area, is initialized to 0.

Byte 6 of the object's STATUS area is initialized with the value FIRST_GEN_NAME + NUMGEN.

All the generators in the GRAPHICS data area are moved to the sprite generator table in VRAM. The first generator is moved to the location specified by FIRST_GEN_NAME and the rest are loaded in sequentially.

4) Activating Complex objects

The number of component objects to activate is determined by the value contained in the MSN of OBJ_TYPE, the first byte in the object's GRAPHICS data area. Following the pointer to the object's STATUS area is a list of addresses. Each address in the list points to the high level definition for one of the component objects. ACTIVATE is called once for each of the component objects, with the state of the carry flag set to the condition it was in when the routine was initially called.

NOTE:
An error in the ACTIVATE routine shows up when ACTIVATEing Semi-Mobile objects, or Complex objects containing Semi-Mobile objects, and the VDP is operating in Graphics Mode I. This error causes an incorrect number of color generator bytes to be moved to the color table in VRAM when FIRST_GEN_NAME of the object is 80H or greater. In these instances, the cartridge program will have to fulfill the functions of ACTIVATE, and move the pattern and color generators to VRAM itself. Initialization of the STATUS and OLD_SCREEN areas can still be done by ACTIVATE; make sure the carry flag is NOT set when calling ACTIVATE on such objects.

7.0 PUT_OBJECT

Calling sequence (for all object types except Mobile objects):

```
:
LD IX,OBJECT_NAME
CALL PUT_OBJECT
:
```

Registers used:

Uses all registers

RAM used (starting at WORK_BUFFER):

1. Semi_Mobile
   If OLD_SCREEN is not used, then no RAM area is used by PUT_OBJECT. If OLD_SCREEN is used, then the maximum space, in bytes, used by PUT_OBJECT will be equal to the number of pattern blocks in the largest frame (i.e. the largest value of X_EXTENT * Y_EXTENT) plus 4.

2. Mobile
   See discussion of PUT_MOBILE below.

3. Sprite
   4 bytes.

4. Complex
   The amount of RAM used is equal to the largest amount used by any of its component objects.

7.1 DESCRIPTION OF PUT_OBJECT

PUT_OBJECT places a frame of an object on the display. Which frame is displayed and where it is placed on the display are specified in that object's STATUS area. When PUT_OBJECT is called, it will determine what type of object is to be "put" on the display and then branch to one of four subroutines designed to handle that particular object type. These subroutines are: PUT_SEMI, PUT_MOBILE, PUT_SPRITE and PUT_COMPLEX. Below is a description of each routine.

7.2 DESCRIPTION OF PUT_SEMI

As the name implies, this routine handles the display of Semi-Mobile objects. A frame for a Semi-Mobile object is put on the display by writing the list of generator names that define the frame into the pattern name table in VRAM. In addition, the names in the name table that are overwritten may optionally be saved and later restored to the name table when the object is moved, or removed from the display.

The following algorithm is used to place Semi-Mobile objects on the display:

```
        IF the object does NOT have an OLD_SCREEN (i.e. if bit 15 of
        the OLD_SCREEN address in the object's high level definition
        is set), THEN

             DISPLAY the frame indicated by FRAME in STATUS at location
             specified by X_LOCATION and Y_LOCATION

        ELSE

             IF 1st BYTE of OLD_SCREEN is NOT 80H (there is valid data
             in OLD_SCREEN), THEN

                  DISPLAY OLD_SCREEN, first 2 bytes in OLD_SCREEN give X
                  and Y coordinates of upper-left corner (in pattern
                  plane positions) of OLD_SCREEN frame, third and
                  fourth bytes give X and Y_EXTENTs of OLD_SCREEN frame

             READ background pattern names over which frame of object
             will be displayed and save in OLD_SCREEN along with X and
             Y positions and X and Y_EXTENTS

             DISPLAY new frame of object at called for X and Y position

        ENDIF
```

7.3 PUT_MOBILE

Calling sequence:

```
         :
        LD IX,OBJECT_NAME
        LD B,MODE               ; MODE = 0-3, DETERMINES THE METHOD
                                ; FOR COMBINING BACKGROUND AND OBJECT
        CALL PUT_MOBILE         ; PUT_MOBILE ENTRY IS IN CARTRIDGE ROM
         :                      ; (SEE DISCUSSION BELOW)
```

Registers used:

Uses all registers

RAM Used (starting at WORK_BUFFER):

In graphics mode I, 141 bytes
In graphics mode II, 203 bytes

7.4 DESCRIPTION OF PUT_MOBILE

This routine will place the specified frame of a Mobile object on the display. The location of the object and the frame to be displayed are determined by the variables FRAME, X_LOCATION and Y_LOCATION in its STATUS area. The X and Y_LOCATION variables specify the pixel position of the upper left corner of the object on the meta-plane. The object therefore may displayed as entirely on the pattern plane or as bleeding on or off the pattern plane in any direction.

In general, PUT_MOBILE produces the image of a Mobile object on the display by creating a new set of pattern and color generators which depict the object superimposed on the background. Since all frames of Mobile objects will be 2 by 2 pattern blocks in size, the number of new generators which need to be created is 9. These 9 generators constitute a "surround" for the object within which the object will be displayed.

The 9 pattern generators which constitute the surround may be thought of as an array of 3 by 24 "surround" bytes. Similarly, the pattern generators which constitute the frame of the object to be displayed may be thought of as an array of 2 by 16 "frame" bytes. PUT_MOBILE uses one of two methods for combining these two arrays of generator bytes. Which method is used is selected by the state of bit 0 in register B when PUT_OBJECT is called.

If this bit is zero then an "additive" method is used to combine the object's graphics with that of the surround. In this method the "1" bits in the object's pattern generators are "moved" into the appropriate locations in a new set of surround generator bytes; these create a graphic of the Mobile object superimposed on the background. New surround color generator bytes are created as follows:

> For each byte, if the corresponding pattern generator byte has not been changed (i.e. no "1" bits have been moved to it) then that color byte is left alone. If the corresponding pattern generator byte has had one or more "1" bits added to it, then the color1 portion of that byte is changed to the color of the Mobile object. Therefore, any parts of the background graphic represented by "1" bits will take on the color of the Mobile object when both the background and the object have "1" bits within the same pattern generator byte. The overall effect of this method is to maintain the "structural" integrity of the background pattern, even though the color of the background graphic will change to that of the object whenever elements of the object and the background exist within the same generator byte. Figure 2 illustrates the graphic effect of this mode of operation.

If bit 0 of register B is set to 1, then another method is applied. As in the above method, the "1" bits of the object's pattern generator bytes are again moved to the appropriate locations within the new set of surround generator bytes. However, surround generator bytes to which bits are to be added will be cleared first (i.e. any elements of the background graphics which exist within the same byte as object graphics will be lost). The color generators are treated in the same manner as above. This gives the effect of the Mobile object "breaking holes" in the background as it moves through a pattern. See figure 3 for an illustration of this mode.

The limitation of only being able to display two colors within the same line of a pattern position force the above compromises when combining object and background generators. Which method should be used in a given situation will depend on the graphics and color of both the background and the object. Experimentation will be necessary to determine which method produces the least disruption of the background graphics.

Another option gives the programmer control over the choice of color0 for any color generator bytes which have had their color1 changed to the object's color.

If bit 1 of register B is 0, then the color0 of the above-mentioned color bytes will not be changed.

If bit 1 of register B is 1, then the color0 will be changed to transparent (thereby causing the backdrop color to be displayed as the color0).

Figures 4 and 5 illustrate these last two modes of operation respectively.

The new pattern and color generators are moved to the object's generator tables as follows:

> Each Mobile-Object will have table space assigned to it (within the VRAM pattern and color generator tables) for 18 pattern and color generators. These tables are divided into an upper and a lower half. When a Mobile-Object is displayed, the "active" generators (i.e. those currently being used to display the object) will reside in either the upper or lower half of its generator tables. The half which is not in use is indicated by bit 7 of FRAME in that object's status area. This bit will be 0 when the lower half is not in use and 1 when the upper half is not in use. PUT_MOBILE moves the new generators to the half of the table not in use and also complements bit 7 of FRAME. This procedure enables the new generators to be moved to VRAM without disturbing the current display. If this were not done, it would be possible for one TV field to display partially updated generators and thereby cause the object to flicker.

Once the new set of pattern and color generators are moved to the VRAM pattern and color tables, PUT_MOBILE displays the Mobile-Object by writing the names of these new generators to the pattern name table in order to display the object at the called-for location. In addition the pattern names which are overwritten in the process of displaying the Mobile-Object are saved, and then restored to the name table, if necessary, when the object moves.

NOTE:
Due to an error near the beginning of the PUT_MOBILE routine, the beginning part of PUT_MOBILE will have to be included as part of the cartridge program. Instead of calling PUT_OBJECT for mobile objects, it will be necessary to call the version of PUT_MOBILE which will reside in cartridge ROM. This has the following two side effects:

1. Mobile-Objects may not be components of a Complex object.
2. The defered write condition will not be recognised by PUT_MOBILE.

The following is the section of PUT_MOBILE which must be incorporated as part of the cartridge program:

```
WORK_BUFFER       EQU     8006H
FLAGS             EQU     3
FRM               EQU     4
YDISP             EQU     0
XDISP             EQU     1
YP_BK             EQU     18
XP_BK             EQU     17
PX_TO_PTRN_POS    EQU     07E8H
GET_BKGRND        EQU     0898H
PM2               EQU     0AE0H

PUT_MOBILE        LD IY,[WORK_BUFFER]        ; IY :=  WORK_BUFFER

     if in GRAPHICS MODE I
                  RES 7,B
     if in GRAPHICS MODE II
                  SET 7,B

                  LD [IY+FLAGS],B
                  PUSH HL
                  LD H,[IX+3]
                  LD L,[IX+2]
                  LD A,[HL]
                  LD [IY+FRM],A
                  XOR 80H
                  LD [HL],A
                  INC HL
                  LD E,[HL]
                  LD A,E
                  AND 7
                  NEG
                  ADD A,8
                  LD [IY+XDISP],A
                  INC HL
                  LD D,[HL]
```

```
                CALL PX_TO_PTRN_POS
                LD [IY+XP_BK],E
                INC HL
                LD E,[HL]
                LD A,E
                AND 7
                LD [IY+YDISP],A
                INC HL
                LD D,[HL]
                CALL PX_TO_PTRN_POS
                LD [IY+YP_BK],E
                LD HL,[WORK_BUFFER]
                LD DE,YP_BK+1
                ADD HL,DE
                LD D,[IY+YP_BK]
                LD E,[IY+XP_BK]
                LD BC,303H
                PUSH IX
                CALL GET_BKGRND
                POP IX
                JP PM2
```

The calling sequence for Mobile-Objects is:

```
                LD IX,HIGH_LEVEL_DEFINITION
                LD HL,GRAPHICS
                LD B,MODE                ; see above discussion for
                                         ; parameter passed in reg B
                CALL PUT_MOBILE
```

An additional patch is needed when operating in GRAPHICS MODE I. In this case the last instruction in the above code (JP PM2) is replaced with the following code:

```
                PUSH IX            ; Save another copy of object pointer
                CALL PM2           ; Call rest of OS PUT_MOBILE routine
                POP IX             ; Restore object pointer
                LD IY,3            ; Set up for 3 item VRAM write
                LD A,[IX+6]        ; Get FIRST_GEN_NAME
                LD B,A             ; And save another copy
                AND A,7            ; Evaluate MOD 8
                CP 7               ; If NE 7 then
                JR NZ,THREE_GEN    ;   3 generators to move
                LD IY,4            ; Else, move 4 generators
THREE_GEN       LD A,B             ; A := FIRST_GEN_NAME
                SRL A              ; Divide by 8
                SRL A              ;   to get index into
                SRL A              ;   color table
                LD E,A             ; DE gets pointer to object's
                LD D,0             ;   color gens in VRAM
                LD HL,V_BUF+18H    ; Point to 4th gen
                PUSH HL            ; Save pointer
```

```
                LD A,[HL]
                LD B,3          ; Copy this generator 3 times
COPY3           INC HL
                LD [HL],A
                DJNZ COPY3
                POP HL          ; Get back pointer
                LD A,4          ; Code for color table
                CALL PUT_VRAM
                RET
```

V_BUF is the address of the work area for OS routines (i.e. the address stored at WORK_BUFFER, 8006H, in cartridge ROM).

NOTE: In applications where the background color (color0) of the patterns over which the mobile object is to move and the color of the mobile object itself does not change, the above patch will not be needed. Instead it is sufficient to initialize the color generators for the mobile object to the desired color1/color0 combination.

7.5 PUT_SPRITE

This routine handles the display of all sprite objects; size0, size1, magnified and unmagnified.

The routine uses SPRITE_INDEX in the object's high level definition to determine which of the 32 sprites to use to implement the object. Positional information from X and Y_LOCATION in the object's STATUS area is used to update the vertical and horizontal position entries in the sprite attribute table in VRAM. The routine facilitates Sprite objects bleeding off to the left as well as completely off the screen by making use of the early clock bit.

Frame information for Sprite objects comes from the FRAME_TABLE in the object's GRAPHICS data area. Each frame is specified by a COLOR and a SHAPE byte. The SHAPE byte is added to the object's FIRST_GEN_NAME (the second entry in the object's GRAPHICS data area) and this sum is then moved to the name entry position in the sprite attribute table. Bit 7 of the COLOR byte is modified to reflect the required state of the early clock bit and then moved to color entry in the sprite attribute table.

7.6 PUT_COMPLEX

A Complex object is a collection of "component" objects. Each frame of a Complex object specifies the positional relationship and frame to be used for each of the component objects. The positional relationship for all the component objects is defined in an offset list and the frame for each component is defined in a frame list. There is one offset list and one frame list for each frame of the Complex object.

PUT_COMPLEX takes the following steps to display Complex objects:

1. Using the FRAME_LIST for the particular frame of the Complex object to be displayed, the frame of each of the component objects is updated.

2. X and Y_LOCATION for each of the component objects is formed by adding the X and Y offsets for each component, as specified in the OFFSET_LIST, to the X and Y_LOCATIONs from the Complex object's STATUS area. Note: the offsets are 8 bit unsigned numbers, so the location of the component objects will always be to the right and/or below the coordinate position indicated by X and Y_LOCATION in the Complex object's STATUS area.

3. Each of the component objects is displayed by calling PUT_OBJECT for each of the component objects.

8.0 DATA STRUCTURE SUMMARY FOR SEMI_MOBILE OBJECTS

*** ROM DATA AREAS ***

High level definition for a Semi-Mobile object; the address of the high level definition (SMO) is passed to the graphics routines when called on to process the object:

```
SMO             DEFW    GRAPHICS    ;pointer to graphic data for object
                DEFW    STATUS      ;pointer to status area
                DEFW    OLD_SCREEN  ;pointer to save area for OLD_SCREEN
```

Graphics for semi_mobile objects are defined as follows:

```
GRAPHICS        DEFB    OBJ_TYPE    ;LSN = 0, MSN used only in Graphics Mode II
                                    ;MSN bits 5,6,7 indicate which thirds of
                                    ;generator tables to move generators to
                                    ;if bit 7 then move gens to upper third
                                    ;"    " 6  "    "    "    "  middle  "
                                    ;"    " 5  "    "    "    "  lower   "
                                    ;bit 4 indicates how many color generator
                                    ;bytes per pattern generator to expect, if
                                    ;bit 4 = 0 then 8 bytes/gen else 1 byte/gen
                DEFB    FIRST_GEN_NAME  ;index into VRAM tables where object's
                                        ;generators are located
                DEFB    NUMGEN      ;number of ROMed generators for object
                DEFW    GENERATORS  ;pointer to first of ROMed patterns
                DEFW    FRAME_0     ;pointer to first frame data
                DEFW    FRAME_1     ;pointer to second frame
                 :        :
                 :        :
                DEFW    FRAME_n     ;pointer to Nth frame
```

The data for each frame is organized as follows:

```
FRAME_n         DEFB    X_EXTENT    ;X_EXTENT and Y_EXTENT are the width and
                DEFB    Y_EXTENT    ;height of the frame in pattern plane positions
                DEFB    NAME_0      ;NAME_0 through NAME_n are the names of the
                DEFB    NAME_1      ;patterns used for this frame.  There must
                 :        :         ;exactly X_EXTENT * Y_EXTENT names and they
                DEFB    NAME_n      ;must be arranged in a ROW major array, as
                                    ;they are to be displayed on the screen (i.e.
                                    ;the pattern representing NAME_0 will appear
                                    ;at the upper-left corner of the frame, NAME_n
                                    ;will appear at the lower-right corner).
```

The pattern generators are stored as follows:

```
GENERATORS      DEFB    bb,bb,bb,bb,bb,bb,bb,bb ;where bb = binary graphic data
                DEFB    bb,bb,bb,bb,bb,bb,bb,bb
                 :              :
                 :              :
                DEFB    bb,bb,bb,bb,bb,bb,bb,bb
```

Each group of 8 bytes corresponds to one generator. These generators are all moved to VRAM by ACTIVATE. The first generator will be located at FIRST_GEN_NAME (in Graphics Mode II, the MSN of OBJ_TYPE indicates which thirds of the generator tables will be initialized).

There are three possible formats for color generators. The first two are used in Graphics Mode II, the third in Graphics Mode I:

GRAPHICS MODE II:

1. If bit 4 of OBJ_TYPE is 0, then there must be 8 color generator bytes per pattern generator. The MSN of each byte specifies color1 for the corresponding pattern generator byte. The LSN specifies color0. (i.e. if the first color generator looks like: DEFB 1F,1F,1F,1F,39,39,39,39 , then the first 4 lines of the corresponding generator will have a color1=BLACK and color0=WHITE and the last 4 lines will have color1=LIGHT GREEN and color0=LIGHT RED.)

```
DEFB    cc,cc,cc,cc,cc,cc,cc,cc ;Color generator for 1st pattern
DEFB    cc,cc,cc,cc,cc,cc,cc,cc ; "        "        " 2nd   "
 :              :
 :              :
DEFB    cc,cc,cc,cc,cc,cc,cc,cc ; "        "        " last  "
```

2. If bit 4 of OBJ_TYPE is 1, then only one color generator byte per pattern generator will be expected. This byte will be duplicated 8 times by the ACTIVATE routine when moving the generators to VRAM. Each byte has the same format as above. There must be one byte per pattern generator.

```
DEFB    cc      ;Color generator for 1st pattern
DEFB    cc      ; "        "        " 2nd   "
 :      :
 :      :
DEFB    cc      ; "        "        " last  "
```

GRAPHICS MODE I:

3. In Graphics Mode I the pattern generator table can be thought of as divided up into 32 groups of 8 generators. Each group of pattern generators share the same color generator byte. Therefore, there must be one color generator byte for each group which contains pattern generators for the object.

*** RAM DATA AREAS ***

The status area for semi_mobile objects is as follows:

```
STATUS          DEFS    1       ;Frame number to be displayed
                DEFS    2       ;X_LOCATION, low byte first
                DEFS    2       ;Y_LOCATION, low byte first
                                ;X and Y_LOCATION used by game program to
                                ;position object on screen and are 16 bit
                                ;signed numbers
                DEFS    1       ;NEXT_GEN, index to area for adding new
                                ;generators
```

Old_screen data has the following structure (if OLD_SCREEN < 7000H, then it is in VRAM, else it is in CRAM):

```
OLD_SCREEN      DEFS    1       ;X_PAT_POS, column in which the upper left corner
                                ;of saved screen lies.  This byte initialised to
                                ;80H by ACTIVATE to indicate no data saved yet.
                DEFS    1       ;Y_PAT_POS, row in which upper left corner of
                                ;saved screen lies.
                DEFS    1       ;X_EXTENT of saved screen
                DEFS    1       ;Y_EXTENT of saved screen
                DEFS    n       ;Where n = the largest screen that will need to
                                ;be saved (i.e. the largest value of X_EXTENT *
                                ;Y_EXTENT for any of the frames of this object).
```

9.0 DATA STRUCTURE SUMMARY FOR MOBILE OBJECTS

*** ROM DATA AREAS ***

High level definition for Mobile objects; the address of this data block (MOB) is passed to the various graphics routines when processing the object:

```
MOB              DEFW    GRAPHICS      ;pointer to graphic data for object
                 DEFW    STATUS        ;pointer to status area
                 DEFW    OLD_SCREEN    ;pointer to save area for OLD_SCREEN
                 DEFB    FIRST_GEN_NAME  ;index into VRAM tables where object's
                                       ;"active" generators are located, i.e.
                                       ;those being used to display the object
                                       ;space for 18 generators must be reserved
                                       ;for each Mobile object
```

Graphics for a mobile object are defined as follows:

```
GRAPHICS         DEFB    OBJ_TYPE      ;LSN = 1
                 DEFB    NUMGEN        ;number of ROMed generators for object
                 DEFW    NEW_GEN       ;pointer to space for creation of new
                                       ;generators
                 DEFW    GENERATORS    ;pointer to first of ROMed patterns
                 DEFW    FRAME_0       ;pointer to first frame data
                 DEFW    FRAME_1       ;pointer to second frame
                   :        :
                   :        :
                 DEFW    FRAME_n       ;pointer to Nth frame
```

The data for each frame is organized as follows:

```
FRAME_x          DEFB    NAME_A,NAME_B,NAME_C,NAME_D,COLOR
                                       ;where the pattern
                                       ;for NAME_A will appear in the upper left
                                       ;corner, NAME_B in the lower left, NAME_C
                                       ;in the upper right and NAME_D in the lower
                                       ;right.  The MSN of COLOR specifies the
                                       ;color for the frame (e.g. if MSN=7 then
                                       ;frame will be CYAN) the LSN must = 0.
```

The pattern generators are stored as follows:

```
GENERATORS       DEFB    bb,bb,bb,bb,bb,bb,bb,bb ;where bb = binary graphic data
                 DEFB    bb,bb,bb,bb,bb,bb,bb,bb
                   :               :
                   :               :
                 DEFB    bb,bb,bb,bb,bb,bb,bb,bb
```

Each group of 8 bytes corresponds to one generator. Each generator is referenced by its position in the the table. The value of the first generator's name is 0, the value of the second generator's name is 1 etc. New generators created at game-on time and stored at (NEW_GEN) continue in the numbering sequence. (i.e. if there are 8 generators in ROM, numbers 0-7, then generator number 8 reffers to the first generator in a table starting at (NEW_GEN), etc.)

*** RAM DATA AREAS ***

The structure for a mobile object's status is as follows:

```
STATUS          DEFS    1       ;the lower 7 bits specify the frame to be
                                ;displayed, bit 7 indicates which VRAM table
                                ;area is in use and must not be altered when
                                ;changing the frame number
                DEFS    2       ;X_LOCATION, low byte first
                DEFS    2       ;Y_LOCATION, low byte first
                                ;X and Y_LOCATION used by game program to
                                ;position object on screen and are 16 bit
                                ;signed numbers
                DEFS    2       ;NEW_GEN, points to area for adding new
                                ;generators
```

Old_screen data has the following structure (if OLD_SCREEN < 7000H, then it is in VRAM, else it is in CRAM):

```
OLD_SCREEN      DEFS    1       ;X_PAT_POS, column in which the upper left
                                ;corner of saved screen lies.  This byte
                                ;is initialized to 80H by ACTIVATE to indicate
                                ;that there is no data saved yet.
                DEFS    1       ;Y_PAT_POS, row in which upper left corner of
                                ;saved screen lies.
                DEFS    9       ;Nine background names, arranged in ROW major
                                ;order.
```

10.0 DATA STRUCTURE SUMMARY FOR SPRITE OBJECTS

*** ROM DATA AREAS ***

High level definition for Sprite objects; the address of this data block (SP_OBJ) is passed to the various graphics routines when processing the object:

```
SP_OBJ          DEFW    GRAPHICS        ;pointer to graphic data for object
                DEFW    STATUS          ;pointer to status area
                DEFB    SPRITE_INDEX    ;sprite number for this object (0-31)
```

Graphics for a sprite object are defined as follows:

```
GRAPHICS        DEFB    OBJ_TYPE        ;OBJ_TYPE = 3
                DEFB    FIRST_GEN_NAME  ;name of first sprite generator
                DEFW    GENERATORS      ;pointer to ROMed generators
                DEFB    NUMGEN          ;number of ROMed generators for object
                DEFW    FRAME_TABLE     ;pointer to table of frame data
```

The data for each frame is organised as follows:

```
FRAME_TABLE     DEFB    COLOR           ;sprite's color for this frame
                DEFB    SHAPE           ;offset from FIRST_GEN_NAME (generators
                                        ;to be used for this frame)

                DEFB    COLOR           ;second frame color
                DEFB    SHAPE           ;second frame generator
                  :        :
                  :        :
                DEFB    COLOR           ;last frame color
                DEFB    SHAPE           ;last frame generator
```

The pattern generators are stored as follows:

```
GENERATORS      DEFB    bb,bb,bb,bb,bb,bb,bb,bb ;where bb = binary graphic data
                DEFB    bb,bb,bb,bb,bb,bb,bb,bb
                  :              :
                DEFB    bb,bb,bb,bb,bb,bb,bb,bb
```

*** RAM DATA AREAS ***

The structure for a mobile object's status is as follows:

```
STATUS          DEFS    1               ;Frame number to be displayed
                DEFS    2               ;X_LOCATION, low byte first
                DEFS    2               ;Y_LOCATION, low byte first
                                        ;X and Y_LOCATION used by game program to
                                        ;position object on screen and are 16 bit
                                        ;signed numbers
                DEFS    1               ;NEXT_GEN, index of free space in
                                        ;generator table
```

11.0 DATA STRUCTURE SUMMARY FOR COMPLEX OBJECTS

*** ROM DATA AREAS ***

High level definition for Complex objects; the address of this data block (COM_OB) is passed to the various graphics routines when processing the object:

```
COM_OB          DEFW    GRAPHICS         ;pointer to graphic data for object
                DEFW    STATUS           ;pointer to status area
                DEFW    OBJECT_1         ;pointer to component object 1
                DEFW    OBJECT_2         ; "              "      "    2
                 :       :
                DEFW    OBJECT_n         ; "              "      "    n
```

Graphics for a Complex object are defined as follows:

```
GRAPHICS        DEFB    OBJ_TYPE         ;MSN of OBJ_TYPE = number of component
                                         ;              objects (must equal number
                                         ;              of high level object addr)
                                         ;LSN of OBJ_TYPE = 4
                DEFW    FRAME_LIST_0     ;pointer to frame list for first frame
                DEFW    OFFSET_LIST_0    ;pointer to list of offsets for first
frame
                 :       :
                 :       :
                DEFW    FRAME_LIST_n     ;pointer to frame list for last frame
                DEFW    OFFSET_LIST_n    ;pointer to list of offsets for last
frame
```

Each frame of a complex object is defined by a list of frame numbers and another list of offsets. Each entry in the FRAME_LIST specifies the frame to be displayed for one of the component objects. The first entry specifies the frame number for the first component object, the second entry specifies the frame number for the second, etc.

```
FRAME_LIST_x    DEFB    f1               ;frame number for first component
                DEFB    f2               ;frame number for second component
                 :       :
                 :       :
                DEFB    fn               ;frame number for last component
```

The OFFSET_LIST specifies the location of each of the component objects with respect to the X and Y_LOCATION given in the complex object's STATUS area as follows:

```
OFFSET_LIST_n   DEFB    Xdisp        ;Xdisp = amount first component displaced
                                     ;horizontally from X_LOCATION of complex
object
                DEFB    Ydisp        ;Ydisp = amount first component displaced
                                     ;vertically from Y_LOCATION of complex
object
                DEFB    Xdisp        ;same for second component
                DEFB    Ydisp
                 :       :
                 :       :
                DEFB    Xdisp        ;same for last component
                DEFB    Ydisp
```

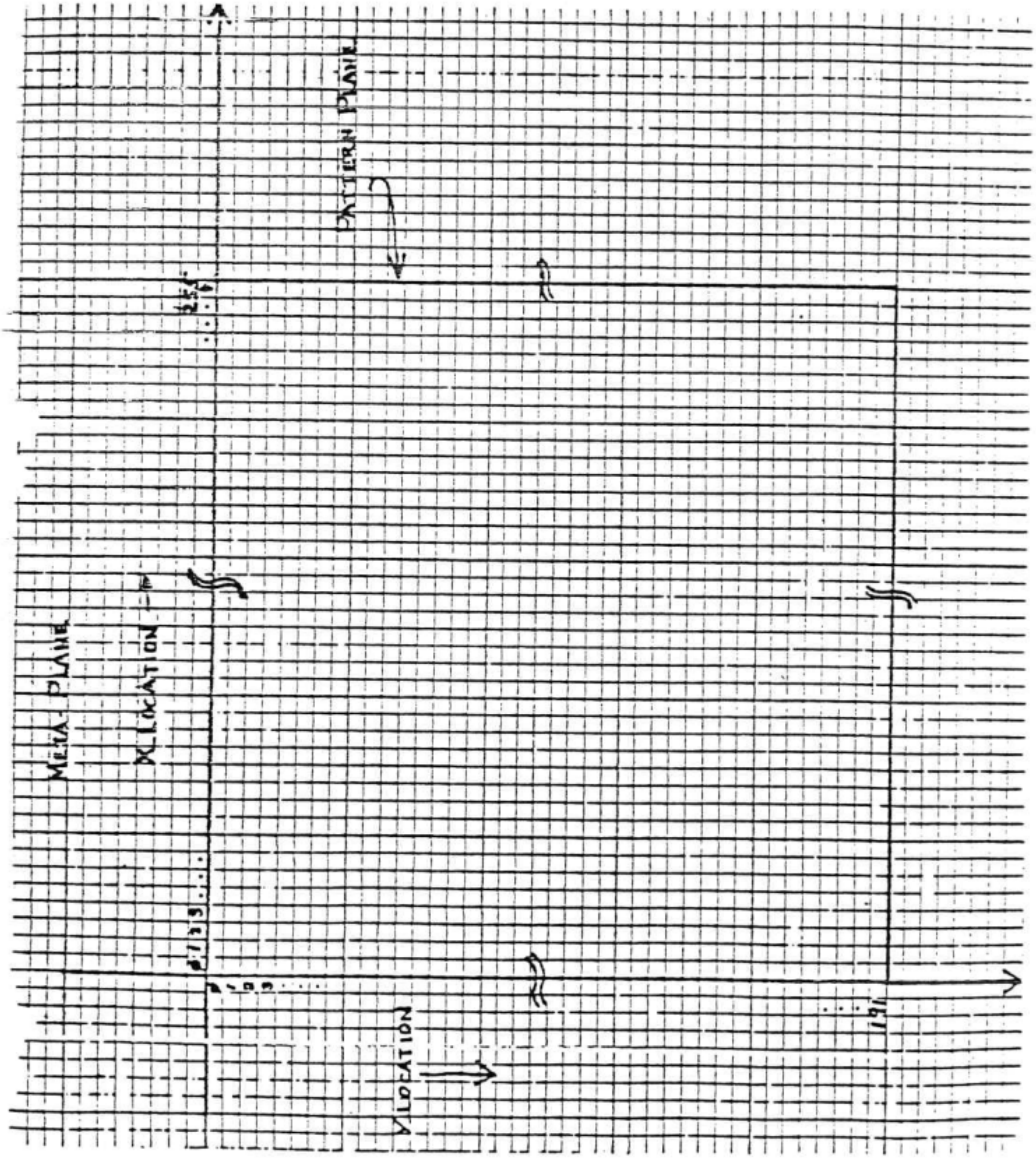
*** RAM DATA AREA ***

Status area for a Complex object:

```
STATUS          DEFS    1            ;Frame number to be displayed
                DEFS    2            ;X_LOCATION of object
                DEFS    2            ;Y_LOCATION of object
```

Figure 1

This figure illustrates the relationship between the Meta-Plane and the Pattern Plane. X_LOCATION and Y_LOCATION are sixteen bit signed variables which determin the location of an object's upper-left corner. These variables are part of each object's STATUS area in CPU RAM.



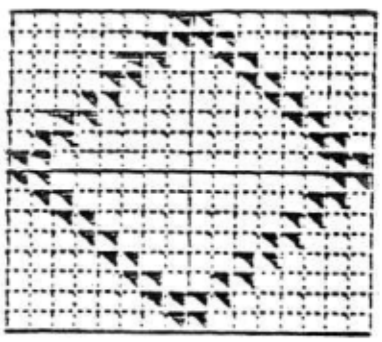
The following figures represent the four methods which PUT_MOBILE uses to superimpose a Mobile object upon a background. A parameter passed to PUT_MOBILE in register B, selects which of the four methods will be used.

Legend: represents the color0 of the background

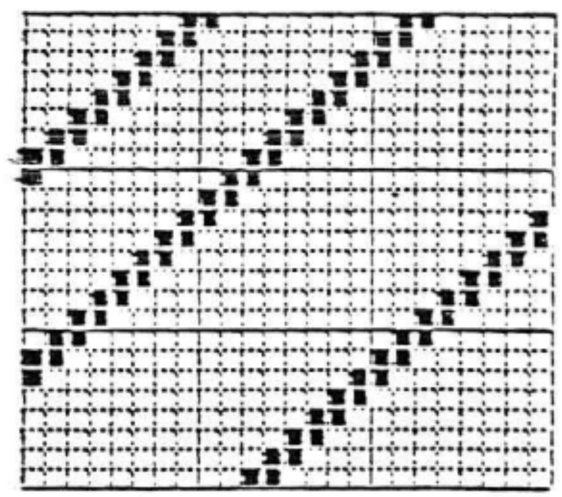
represents the color1 of the background

represents the color of the Mobile object

represents the backdrop color



Mobile object.



Three by three pattern position "surround" onto which the Mobile object will be placed.

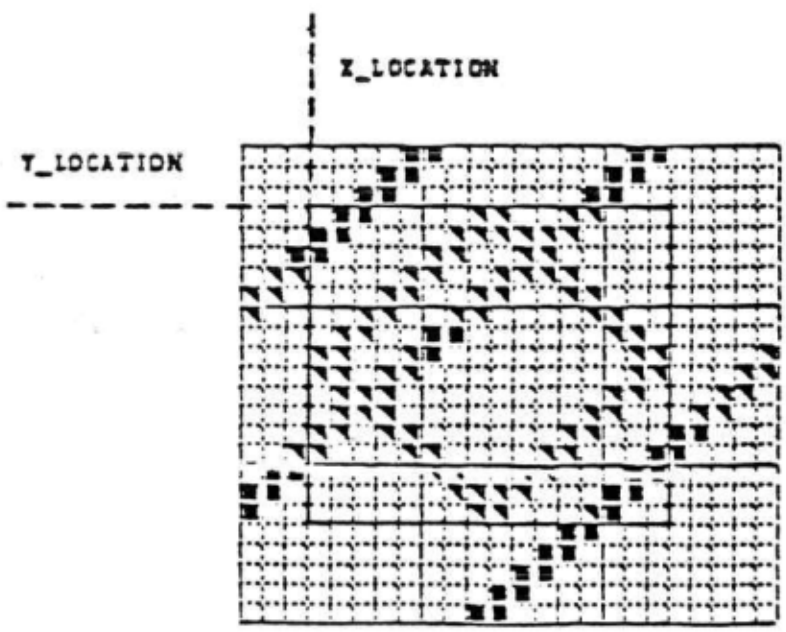


Figure 2: Mobile object superimposed on surround when parameter passed in register B = 0.

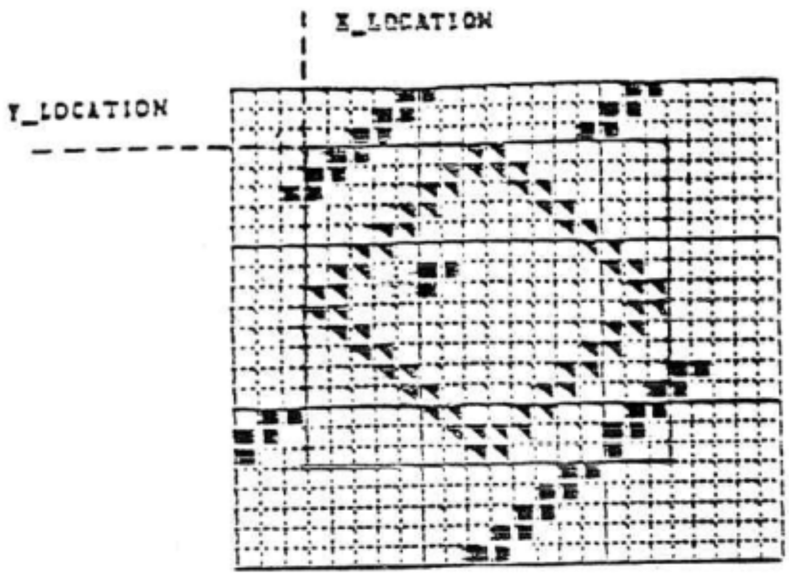


Figure 3: Mobile object superimposed on surround when parameter passed in register B = 1.

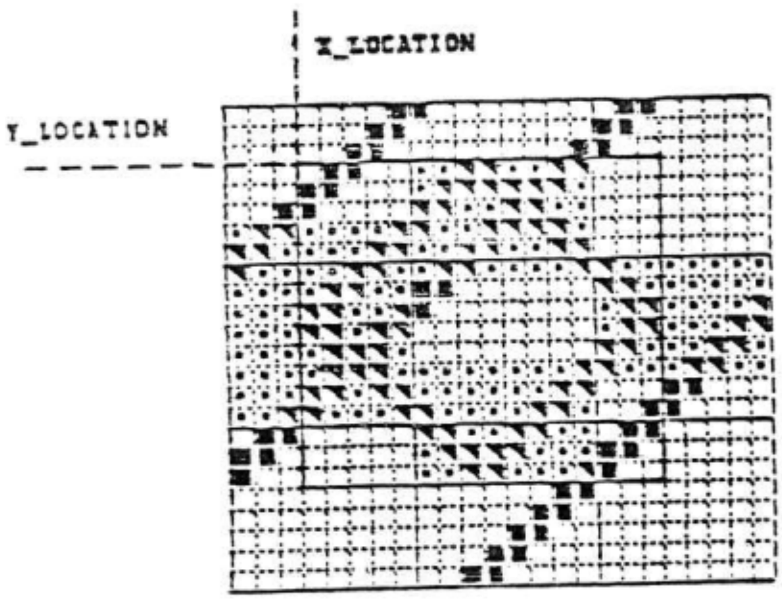


Figure 4: Mobile object superimposed on surround when parameter passed in register B = 2.

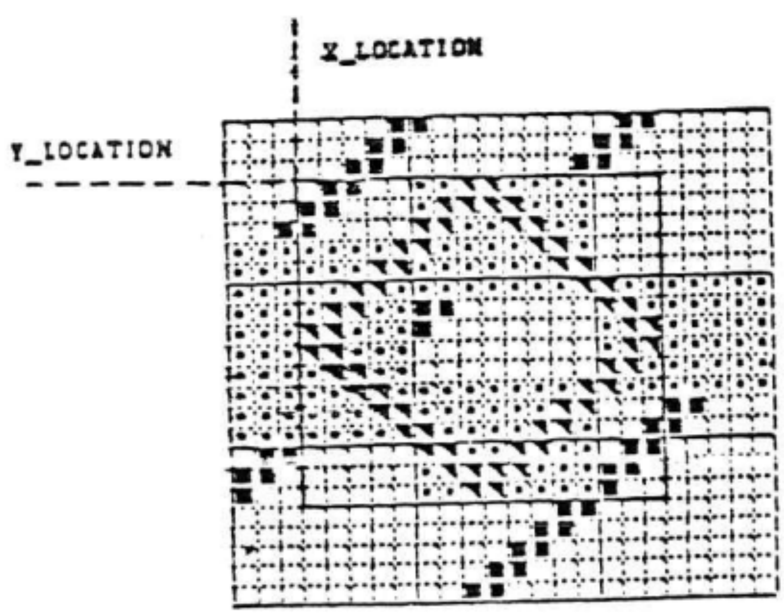


Figure 5: Mobile object superimposed on surround when parameter passed in register B = 3.

APPENDIX C

COLECOVISION SOUND USERS' MANUAL

Version 1.1
May 3,1982

also, half way down the page:
"...pointed to by CPU RAM word LST_OF_SND_ADDRS"
- should read -
"...pointed to by CPU RAM word PTR_TO_LST_OF_SND_ADDRS"

FIG 4 second line:
"Length in bytes: 2"
- should read -
"Length in bytes: 1"

also, about 6 lines down from that:
"B4 - B0= duration, 1 to 31"
- should read -
"B4 - B0= duration, 1 to 30"

*** NOTES ***

1) After reading through page 6 in the Users' Manual, it may be helpful to review the following summary of dedicated pointers and data structures (discussion refers to Figure 10, included as part of these notes):

DEDICATED RAM

Prior to calling any OS sound routines (except INIT_SOUND), the 11 CRAM locations 7020H through 702AH must be initialized to meaningful data. Ten of the locations are two byte pointers:

PTR_TO_LST_OF_SND_ADDRS:
7020-21H - Points to the start of a list of pointers in cartridge ROM, LST_OF_SND_ADDRS (see later). The OS sound routines know where the cartridge-dependent LST_OF_SND_ADDRS is stored through this pointer. It is shown pointing to the first byte in this ROM list (as it must).

PTR_TO_S_ON_1:
7022-23H - This and the following three pointers are used by OS sound routines to store the addresses of the four song data areas which currently contain the sound data to be modified/output to the four sound generators in the TI sound chip. This pointer stores the address for the song currently playing on tone generator #1.

-- EXAMPLE -- PTR_TO_S_ON_3 is shown pointing to the second song data area. I.e., data for the song currently playing on tone generator #3 happens to be stored in the second song data area. The second data area is used for purposes of illustration only: other songs may very well require that data for tone generator #3 be stored in a different song data area.

PTR_TO_S_ON_2:
7024-25H - As above, for tone generator #2.

PTR_TO_S_ON_3:
7026-27H - As above, for tone generator #3.

PTR_TO_S_ON_0:
7028-29H - As above, for tone generator #0 (the noise generator).

The final byte at 702AH, SAVE_CTRL, is used by the OS sound routines to store data necessary for smooth operation of the noise generator (see bottom of page 11 in the Users' Manual).

All 11 bytes should be initialised before the OS routines which operate upon them are called: this is done by calling INIT_SOUND and passing the appropriate cartridge-dependent information (see Users' Manual pages 5 and 13).

ROM

Cartridge ROM to be used by OS sound routines is divided into two sections: LST_OF_SND_ADDRS and the Note List.

LST_OF_SND_ADDRS:
A contiguous list of 4 bytes per each song (or special effect) used by the game. In each 4 byte section, the first 2 bytes are a pointer to the beginning of the song's note list (also in ROM). The second two bytes are a pointer to the song data area in RAM to be used by that song (review pages 2 and 3 in the Manual). Of course, another song may also use the same data area, since there can be (and usually are) more songs than there are data areas. NOTE, however, that Song #1 MUST be the first entry in LST_OF_SND_ADDRS for the OS routines to operate properly.

-- EXAMPLE -- The first two bytes in this list are a pointer to song #1's note list, as they MUST be. The second two bytes are a pointer to the song's data area in RAM, shown here pointing to the first song data area as also MUST be the case (see later). As can also be seen from the figure, the last song happens to use the second song data area.

In summary, there is a note list for every note list pointer in LST_OF_SND_ADDRS, but, since songs may share RAM data areas, there are almost always more data areas than there are data area pointers.

NOTE LIST:

The Note List is a contiguous block of ROM containing the data which comprise the notes of each song. The number of bytes per song of course varies with the length of the song. The first byte of the note list for a song is pointed to by a two byte entry in LST_OF_SND_ADDRS (see above). The last byte of each song's note list is a single byte end of song/repeat code (see page 2 and Figure 2 in the Users' Manual).

USER RAM

This is the area in CRAM that the cartridge programmer has chosen to hold the ten byte song data areas which contain sound and timing information to be processed by the OS sound routines. These data areas must be stored as contiguous blocks of ten bytes each. In all cases but one, the programmer may choose to "play" a song in any data area; however, song #1 MUST use the FIRST song data area for the OS routines to work properly. Also, the byte immediately following the last byte in the last data area MUST be zero (this code tells the OS routine SND_MANAGER to stop looking for more song data areas; see bottom of page 5, Users' Manual). This byte will automatically be set to zero by proper invocation of the INIT_SOUND routine before any other OS sound routines are called (see pages 5 and 13).

2) The ColecoVision OS entry point names of some of the sound routines and dedicated locations are different from their names given in the Sound Users' Manual. They are:

Entry Point	Sound Users' Manual
PLAY_IT	JUKE_BOX
SOUND_MAN	SND_MANAGER
SOUND_INIT	INIT_SOUND
NOTES	PTR_TO_LST_OF_SND_ADDRS

You should use the entry point names.

3) Page 22, last paragraph: It is mentioned that a special effect routine may want to call the OS sound routines FREQ_SWEEP and ATN_SWEEP to operate upon data within the effect's data area, which "require that data be ordered appropriately within a song data area". This means:

Whether or not the special effect uses FREQ_SWEEP or ATN_SWEEP: bytes 3 and 4 (see FIGURE 1) MUST contain the frequency and attenuation data as specified. This is because PLAY_SONGS (called every interrupt) will output

For FREQ_SWEEP used by itself - in addition to bytes 3 and 4, bytes 5, 6, and 7 must contain data as specified. Bytes 8 and 9 may be used for whatever (since FREQ_SWEEP doesn't look at them).

For ATN_SWEEP used by itself - in addition to bytes 3 and 4, bytes 5, 8, and 9 must contain data as specified. Bytes 6 and 7 may be used for whatever (since ATN_SWEEP doesn't look at them).

If both FREQ_SWEEP and ATN_SWEEP are used, all bytes in the data area must look as specified in FIGURE 1.

The Colecovision operating system includes routines which allow the cartridge programmer to store "songs" and simple sound effects in tabular form in cartridge ROM, and play them on request during the game. More complex, special sound effects can also be created and played within the same data structure and procedural format. This Sound Users' Manual describes the data structures expected by and the use of the operating system sound routines.

SUMMARY OF FEATURES

* six song note types: combinations of fixed or variable frequency and attenuation, "noise" (percussion) notes, and rests

* hierarchical structure of local data areas assigned to each sound channel, which allows the temporary "overwriting" of lower priority songs (i.e., lower priority songs are not truncated by higher priority songs or sound effects that use the same channel, but continue unheard until the higher priority songs finish)

* the ability to easily include a special sound effect (say, a cymbal crash) as part of a song composed primarily of musical tones

* both songs and special, independent sound effects (e.g., an explosion sound) can utilize the same data structures and output procedures

* sweep routines, which automatically create frequency or attenuation sweeps, simplify note data storage and can be used by special sound effects

* song end codes allow songs to be played once, or automatically restarted upon completion (repeat forever)

GENERAL DESCRIPTION

* Song data areas, SxDATA: RAM map mode of sound chip operation

The Colecovision operating system provides sound routines which output frequency, attenuation, and control data to the TI 76489 sound chip. Data to be sent to a particular sound generator channel is expected to be stored within a ten byte block of CPU RAM called a "song data area". A song data area, then, contains a RAM record of the current values "playing" on a sound channel.

Each song data area can also contain timing and descriptive information which allows for simple generation of musical notes. A "song" can be created by storing in CART ROM a list of note parameters which specify note duration, frequency, and attenuation. When a song is started, O/S routines are provided to load the data describing the first note of the song into a song data area. Each song data area is then processed at regular intervals by routines which modify and output the area's data to the sound chip. When a note is completed, the next note in the song is automatically loaded and the process continues.

O/S routines also exist which facilitate the creation of "special effects": sound routines written by the cartridge programmer which algorithmically generate data to be sent to the sound chip (as opposed to the table look-up, song approach).

RAM space for at least four song data areas must be reserved by the cartridge programmer: one each to describe the current status of the four sound chip channels. More than four song data areas will be required if the ability to "overwrite" lower priority songs is desired, and some songs may share the same data area (see the following discussion, "Hierarchy of song data areas: priority, truncation, overwriting"). The first byte in each song data area, byte 0 (its offset from the beginning of the data block = 0), contains the channel number upon which the song is to be played (0: noise generator, 1 to 3: tone generator) and the song's identification number (SONGNO: 1 to 61). A song data area is referred to throughout the rest of this manual as "SxDATA", where x = the song's SONGNO. For a detailed description of each byte in a song data area, see the following discussion, "NOTES", and refer to Figure 1.

* Note list storage and note headers

A note list is a sequential list of frequency and timer data stored in cartridge ROM that, when processed and output to the sound chip, create the notes that comprise a song. Each block of 1 to 8 bytes of data that describes a note in the list must begin with a one byte header which contains information (bit flags or values) that indicate (see Figure 2):

1) The number of the channel upon which to play the note
2) The note type (one of 4 combinations of fixed or swept frequency and attenuation, plus a rest).

The single byte header can also be used as an end-of-song marker, a repeat-song indicator, or an indicator that a "note" is to be determined algorithmically by a special sound effect routine (the starting address of which immediately follows).

A 16 bit pointer to the location of the header or the next note to be played (NEXT_NOTE_PTR) is maintained by the O/S routine SND_MANAGER in each song's data area (SxDATA, offsets 1 and 2).

* LST_OF_SND_ADDRS and PTR_TO_LST_OF_SND_ADDRS

The O/S routines expect the ten byte long song data areas to be stored contiguously in CPU RAM, starting with the data area used by song number one. The beginning addresses of each of these data areas, as well as the addresses of the headers of the first note in each song, are stored in a ROM table called LST_OF_SND_ADDRS (see Figure 3). The cartridge programmer may place this table wherever desired in ROM. The O/S routines know its starting address through a dedicated CPU RAM location, PTR_TO_LST_OF_SND_ADDRS, which must be loaded with the 16 bit address of the table by the cartridge program before calling any O/S routines which use it (see description of INIT_SOUND).

* Hierarchy of song data areas: truncation, priority, overwriting

The routine that does the processing of the note data stored in the song data areas, SND_MANAGER, and the routine that outputs the modified data to the sound chip, PLAY_SONGS, are designed to be called by the cartridge program every Video Display Processor (VDP) interrupt (every 16.7 ms). Starting with the data area for song number one, SND_MANAGER processes the appropriate timer and sweep counters and modifies the frequency and attenuation data accordingly. If the data area is assigned to a special effect, SND_MANAGER calls that effect. When a note is finished, SND_MANAGER, using the data area's next note pointer, moves data for the next note of the song into the area.

After the operations upon a data area have been performed, the sound chip channel number (CH#) stored in byte 0 of that data area is consulted and the appropriate "channel data area pointer" (PTR_TO_S_ON_x) is set to point to the beginning of the data area just processed (four of these pointers exist at dedicated 16 bit locations in CPU RAM, one for each of the sound generator channels). The following data areas are then processed in the same fashion, in order of occurence, until the end of data area code, 00, is reached. If a data area is inactive, i.e., if the song(s) that use it aren't playing at the moment, SND_MANAGER simply passes it over, doing no processing or channel data area pointer modification.

PLAY_SONGS, usually called immediately prior to SND_MANAGER, outputs data to the sound chip from the four song data areas pointed to by the channel data area pointers. Thus, a channel output priority is established on the basis of ordinal position within the data area block: the last data area processed that uses a given channel is the one that will be played on that channel. E.g.,

order of data area within data block containing all song data areas	songs that use this data area: SONGNO/CH# song is to be played on	
1st	1/CH#x	(remember: although other songs may use this data area also , song 1 MUST use it)
...		
5th	6/CH#2	
6th	3/CH#2	
7th	11/CH#2	
...		
10th	2/CH#3; 4/CH#3; 7/CH#3	

First, consider channel 2. Let's say that the only songs which use channel 2 are assigned three contiguous song data areas, 5th through 7th (grouping songs which use the same channel isn't necessary as far as the code is concerned, but it makes it simpler to think about). SND_MANAGER, as it makes its way through the song data areas in order, will process the 5th data area (which "belongs" to song 6) and set PTR_TO_S_ON_2 to the address of byte 0 in the 5th data area. Then, the 6th area will be processed, resulting in resetting PTR_TO_S_ON_2 to the 6th data area (song number 3). Likewise, the 7th data area will be processed, finally leaving PTR_TO_S_ON_2 pointing to the 7th data area (song number 11). The next time PLAY_SONGS is called, it will send to sound chip channel 2 frequency and attenuation data in the data area pointed to by PTR_TO_S_ON_2, namely, the data for song number 11.

Note that although only song 11 will be heard on channel 2 this pass through PLAY_SONGS, the timers and data for songs 6 and 3 were nonetheless modified, regardless of the existence of the higher priority song 11. That is, all songs "keep going", whether or not their data will be output during PLAY_SONGS. Songs 6 and 3 are said to have been "overwritten" by song 11. Should song 11 become inactive (end) before song 6 and/or song 3, then the highest priority of the remaining active songs (i.e., the last data area within the block of data areas to use a given channel) will be heard.

Thus, assigning several data areas to songs which use the same channel allows the creation of "background" songs which can be momentarily interrupted (overwritten) by a higher priority song or sound effect (e.g., an explosion, or a bonus song) and continue on after the overwriting song is over.

Now examine the 10th song data area. Again, let's say that the only songs that use channel 3 are shown here and that they all share the 10th data area. In this case, the programmer may have arranged things such that songs 2, 4, and 7 are never active simultaneously: i.e., there was no reason to assign three different data areas for songs which never overlap. If, however, this is not the case and, say, song 4 may be started before song 7 is finished, the O/S routines would stop song 7 in favor of song 4. That is, for songs that share the same data area, the most recent song started is the song heard, and interrupted songs do not continue: i.e., songs sharing the same data area truncate each other. In many cases this may be both acceptable and desirable, as it saves RAM space.

NOTE: The preceeding description states that a channel data area pointer is updated every time SND_MANAGER processes a song data area: this is actually not the case. To save processing time, a routine which updates all the pointers is called only when, after loading the next note in a song, SND_MANAGER detects that the data area's CH# or SONGNO has changed. This happens whenever the next note: 1) uses a different channel (see "Noise notes: special case Type 2 notes), 2) is a special effect note, or 3) is an end-of-song indicator. It is only necessary to update the channel data area pointers in these cases, and when a new song is started (in JUKE_BOX). See "Pseudo code versions of main routines".

* The four basic routines, briefly

The following four O/S routines are the only ones that need be called to create songs which use the six standard note types (more complete descriptions of each routine can be found in the "OPERATING SYSTEM ROUTINES" section):

INIT_SOUND: This routine should be called immediately after power on, before any sound processing can occur. It turns off the sound generators, initializes the CART RAM locations to be used as song data areas, sets up the four channel data area pointers, and initializes PTR_TO_LST_OF_SND_ADDRS.

INPUT: n
TYPE: 8 bit constant
PASSED: in B
DESCRIPTION: number of song data areas used by the game

INPUT: LST_OF_SND_ADDRS
TYPE: 16 bit address
PASSED: in HL
DESCRIPTION: LST_OF_SND_ADDRS is the base address of a list of the starting addresses of each song's data area and note list.

OUTPUT: 1) turns off all sound generators
2) initializes PTR_TO_LST_OF_SND_ADDRS
3) writes the inactive code (0FFH) to byte 0 of the n song data areas
4) stores 00 at end of song data areas
5) sets the 4 channel song pointers to a dummy inactive area
6) sets SAVE_CTRL to 0FFH (see "Noise notes" discussion)

JUKE_BOX: JUKE_BOX is called to start a song. Using a song number passed in B, JUKE_BOX loads the data for the song's first note into the appropriate song data area, thereby truncating whatever song had been "playing" in that data area. (The address of the appropriate area is found by using the song number as an index into the LST_OF_SND_ADDRS table). It also formats the data area's header and sets up the next note pointer. If the song is a special sound effect, its next note pointer is set to the address of the special effect routine. The next time PLAY_SONGS is called, that song's first note will be played.

If JUKE_BOX is called with a song number of a song already in progress, it returns immediately (i.e., it doesn't restart the song).

INPUT: song number to be started
TYPE: 8 bit constant, 1 to 61
PASSED: in B

CALLS: PT_IX_TO_SxDATA, LOAD_NEXT_NOTE_PTR, UP_CH_DATA_PTRS

OUTPUT: 1) moves the song's first note data to the appropriate song data area
1) formats byte 0 header of the song's data area
2) points next note pointer in data area (bytes 1&2) to address of first note in song, or address of special sound effect routine

SND_MANAGER: SND_MANAGER should be called every VDP interrupt (every 16.7 ms). For each data area, SND_MANAGER processes the appropriate timer and sweep counters and modifies the frequency and attenuation data accordingly. If the data area is assigned to a special effect, SND_MANAGER calls that effect. When a note is finished, SND_MANAGER, using the data area's next note pointer, moves data for the next note of the song into the area. If SND_MANAGER reads a header byte (in CART ROM) that has bits 3&4 set, indicating repeat song, it will start the song again by reloading the first note in the song.

After the operations upon a data area have been performed, if necessary, the channel data area pointers (PTR_TO_S_ON_X) are updated. The following data

SND_MANAGER does not output the modified frequency and attenuation data. PLAY_SONGS is called just before SND_MANAGER to do this.

Special codes in byte 0 of the song data area indicate:

255: data area inactive, do no processing
62: a special effect is to be played; SND_MANAGER calls the effect routine
0: end of song data areas (SND_MANAGER processes data areas until it sees 0 in byte 0)

NOTE: Song number 1 MUST use the first area in the block of song data areas.

INPUT: none

CALLS: PT_IX_TO_SxDATA, PROCESS_DATA_AREA

OUTPUT: Calls routines which:
1) decrement song duration and sweep timers
2) modify swept frequency and attenuation values
3) call special effects routines where necessary
4) update the channel data area pointers if necessary
5) restart the song if indicated

PLAY_SONGS: PLAY_SONGS takes the frequency and attenuation data pointed to by the four channel data area pointers (PTR_TO_S_ON_x) and outputs it to the four sound chip generators.

INPUT: none

CALLS: TONE_OUT, UPATNCTRL

OUTPUT:
1) current freq and atn data is output to each tone generator, if song/effect on that channel is active; if song on that channel is inactive, that generator is turned off
2) noise generator is sent current atn data, and control data, if new
3) modifies SAVE_CTRL if necessary

These four routines would normally be called as follows:

power on inits done by O/S
cartridge program receives control:
LD B, # of song data areas used in the game
LD HL, address where LST_OF_SND_ADDRS is stored in ROM
CALL INIT_SOUND to initialize song data areas
whatever other power on inits you want to do
start game:
.
.
.
LD B, # of song you want to start
CALL JUKE_BOX to set up for start of song
.
.
VDP interrupt occurs:
CALL PLAY_SONGS to output data
CALL SND_MANAGER to process song data
whatever else you want to do during VDP interrupts
RETN to game

NOTES

* Terminology

Each note in a song has an associated 10 bit frequency (except for noise notes) and 4 bit attenuation which is output to the sound chip every time PLAY_SONGS is called. The initial frequency and attenuation values (stored in 2 bytes) are part of a block of 4 to 8 bytes that describe a single note within a song's ROM note list. The remaining bytes are used to indicate sound channel, note type, duration, and various timers and values associated with swept notes.

The following are explanations of names and symbols used throughout this manual to refer to bytes, or segments of bytes, within both a note's ROM note list and a RAM song data area (see Figure 1):

|: | is a symbol used to graphically separate bits or nibbles within a byte

Bx: means bit x of a byte, bit 7 being the most significant bit

byte x: refers to the offset of a byte within a data block, byte 0 being the first byte in the block

MSN, LSN: MSN means the most significant nibble of a byte, LSN is the least significant nibble

CH#: the sound channel upon which a note is to be played; 0 = noise generator, 1 to 3 = the 3 tone generators

SONGNO: the song number of the song playing in a song data area; 1 to 61; SONGNO 62 means a special sound effect is using the data area

NEXT_NOTE_PTR: 2 byte address of the ROM location of the data block for the next note to be played in a song

F0 - F9: the 10 bit frequency data to be sent to a sound chip tone generator; F0 is the most significant bit, see data sheets, TI 76489

ATN: 4 bit attenuation data to be sent to any of the four sound generators

CTRL: 3 bit control data for sound chip noise genrator; FB NF0 NF1 (called SHIFT in this manual), see data sheets for details

NLEN: 1 byte that directly or indirectly determines the duration of a note

FPS: 4 bit frequency prescaler, used with NLEN to determine the length of a frequency sweep

FPSV: 4 bit temporary storage location for FPS; this variable gets decremented every VDP interrupt, and is reloaded from FPS

FSTEP: the size of a step in a frequency sweep, an 8 bit two's complement signed value that is added to the current 10 bit frequency at a rate determined by NLEN and FPS, 1 to 128, -1 to -128

ALEN: 4 bit number of steps in an attenuation sweep

ASTEPS: ASTEPS is the signed, 4 bit size of a step in an attenuation sweep; can take values 1 to 7, -1 to -8 (MSB = 1 means negative)

APS: 4 bit attenuation prescaler, used with ALEN to determine the length of a frequency sweep

APSV: 4 bit temporary storage location for APS; this variable gets decremented every VDP interrupt, and is reloaded from APS

* Frequency sweeps and note duration

The time duration of a note = the number of passes by PLAY_SONGS through the note times 16.7ms (the VDP interrupt period). (note that the time the note is HEARD could be shorter: see "Attenuation sweeps" discussion) The number of PLAY_SONGS passes is always determined directly or indirectly by NLEN. NLEN, however, has two meanings, depending upon whether or not a note's frequency is swept:

Fixed frequency notes - In this case, NLEN is decremented every VDP interrupt and therefore directly determines the length of a note:

duration = NLEN * 16.7ms

NLEN should have values in the range 0 to 255 (0 => 256), giving a maximum duration of ~ 4.25 seconds.

Swept frequency notes - Here, the prescaler variable, FPSV, is decremented until zero before NLEN is decremented. Once FPSV goes to zero, it's reloaded from FPS; however, an initial value for FPSV, which enters into the calculation of the the length of the first step in the sweep, is stored in ROM along with the rest of the note's initial data, and it may be different from the reload value, FPS. For a frequency swept note:

note duration = [((NLEN -1) * FPS) + initial FPSV] * 16.7ms

So, NLEN again determines note duration, but in an indirect fashion (in concert with FPS and the initial FPSV).

In the case of a frequency swept note, NLEN can be thought of as one of four parameters that describe the sweep: 1) the starting frequency, 2) FSTEP, a signed step size, i.e., a delta frequency that is periodically added to the current frequency, 3) the prescaler value (FPS) which determines the length of time at any one frequency step, and 4) NLEN, the number of steps in the sweep.

The duration of each step in the sweep is given by the following:

duration 1st step = initial FPSV * 16.7ms
duration all others = FPS * 16.7ms

Frequency sweep parameter ranges:

FSTEP - signed 8 bit two's complement number: 1 to 127, -1 to -128; an FSTEP of 0 tells FREQ_SWEEP that the note is not frequency swept, and the note's duration is determined by directly decrementing NLEN (prescaler is disregarded)

FPS - 4 bit frequency prescaler, used with NLEN to determine the length of a frequency sweep: 0 to 15 (0 => 16)

FPSV - 4 bit temporary storage location for FPS; this variable gets decremented every VDP interrupt, and is reloaded from FPS. 0 to 15 (0 => 16)

NLEN - 8 bit note duration for a fixed frequency note: 0 to 255 (0 => 256)
8 bit number of steps for a swept frequency note: 2 to 255 (0 =>256)

Note durations:

Fixed frequency -	NLEN * 16.7ms
Swept frequency -	[((NLEN -1) * FPS) + initial FPSV] * 16.7ms
duration 1st step =	initial FPSV * 16.7ms
duration all others =	FPS * 16.7ms

* Attenuation sweeps

Volume attacks and decays can be thought of as attenuation sweeps: a sweep from low to higher volume is an attack, a sweep in the other direction is a decay. Attenuation sweeps are created in a similiar fashion to the frequency sweeps described above, the primary difference being that attenuation sweep parameters don't take on 8 bit values. The full volume range for the attenuation registers on the 76489 chip is 0 (ON) to 15 (OFF), so step sizes and number of sweep steps greater than 4 bits aren't generally useful.

Just as in the case of a frequency swept note, attenuation sweeps have four parameters that describe the sweep: 1) the starting attenuation, 2) ASTEP, a signed step size, i.e., a delta attenuation that is periodically added to the current attenuation, 3) the prescaler value (APS) which determines the length of time at any one attenuation step, and 4) ALEN, the number of steps in the sweep.

The prescaler parameters, APS and APSV, are the same size (4 bits) and mean exactly the same thing as their frequency counterparts. ALEN, the number of steps in the sweep, is only 4 bits (compared to an FLEN of 8 bits), but 15 steps of even the smallest step size (+/-1) can sweep a generator from full on to full off. ASTEP, in order to squeeze it into a nibble, has been limited to a 4 bit signed number (3 bits data, 1 bit sign). This gives a range of step values from 1 to 7, -1 to -8. This shouldn't be too limiting, since most attenuation sweeps are implemented with the smallest step size.

NOTE: Recall that, as far as SND_MANAGER is concerned, the length of a note is determined, directly or indirectly, only by NLEN, a timer/counter that is decremented during FREQ_SWEEP; i.e., the duration of a note is independent of what is happening to its attenuation. Therefore, the programmer should take care to see that an attenuation sweep isn't inadvertently created that ramps the volume down to off before SND_MANAGER, through NLEN, has decided that the note is over. However, since ATN_SWEEP simply leaves the attenuation alone once it's finished a sweep, the independence of attenuation sweep length and note length may be put to good use: e.g., a sforzando can be accomplished by making an attenuation sweep (to a still audible volume) end before the rest of the note.

ALEN, like NLEN for a frequency sweep, is the number of steps in an attenuation sweep and can take on values from 0 to 15. However, since a "step" consists of a tone (which may be frequency swept) played at a fixed attenuation level, a sweep of 1 step doesn't make sense. ALEN values, then, should range from 2 to 15. An ALEN value of 0 causes a sweep of sixteen steps (NOTE: ATN_SWEEP "wraps around" at 0 and 15, i.e., subtracting 1 from 0 results in 15, and adding 1 to 15 results in 0).

The duration of an attenuation sweep can be calculated as follows:

```
duration entire sweep = [((ALEN -1) * APS) + initial APSV] * 16.7ms
duration 1st step  =         initial APSV * 16.7ms
duration all others =        APS * 16.7ms
```

Attenuation sweep parameter ranges:

ALEN: 4 bit number of steps in an attenuation sweep; can take values from 2 to 15 (0 => 16 steps).

ASTEP: ASTEP is the 4 bit signed (two's complement) size of a step in an attenuation sweep; can take values from 1 to 7, -1 to -8.

APS: 4 bit attenuation prescaler, used with ALEN to determine the length of a frequency sweep: 0 to 15 (0 => 16).

APSV: 4 bit temporary storage location for APS; this variable gets decremented every VDP interrupt, and is reloaded from APS: 0 to 15 (0 => 16)

Descriptions of each of the six note types follow:

* **Rests**

See Figure 4.

byte 0: B5 set indicates a rest, to be played on CH# in B7 - B6
duration = (B4 - B0) * 16.7ms, can take values 1 to 31

* **Type 0: Fixed frequency, fixed attenuation**

See Figure 4.

byte 0: header, CH# in B7 - B6, note type = 0 in B1 - B0
byte 1: least significant 8 bits of the 10 bit frequency data (constant)
byte 2: MSN = 4 bit ATN data (constant throughout the note)
LSN = 0 0 F0 F1, the top 2 bits of the frequency data (constant)
byte 3: NLEN, duration of the note = NLEN * 16.7ms

* **Type 1: Swept frequency, fixed attenuation**

See Figure 5.

byte 0: header, CH# in B7 - B6, note type = 1 in B1 - B0
byte 1: least significant 8 bits of the initial 10 bit frequency data
byte 2: MSN = 4 bit ATN data (constant throughout the note)
LSN = 0 0 F0 F1, the top 2 bits of the initial frequency data
byte 3: NLEN, number of steps in the frequency sweep, 1 to 255 (0 => 256)
byte 4: FPS : FPSV, prescaler reload value and initial FPSV
byte 5: FSTEP, sweep step size, 1 to 127, -1 to -128

* **Type 2: Fixed frequency, swept attenuation**

See Figure 6.

byte 0: header, CH# in B7 - B6, note type = 2 in B1 - B0
byte 1: least significant 8 bits of the 10 bit frequency data (constant)

byte 2: MSN = 4 bit ATN data (initial value)
LSN = 0 0 F0 F1, the top 2 bits of the frequency data (constant)
byte 3: NLEN, duration of the note = NLEN * 16.7ms
byte 4: ALEN | ASTEP
ALEN = number of steps in the attenuation sweep
ASTEP = step size, 1 to 7, -1 to -8
byte 5: APS | APSV, prescaler reload value and initial APSV, 1 to 15 (0 => 16)

* Type 3: Swept frequency, swept attenuation

See Figure 7.

byte 0: header, CH# in B7 - B6, note type = 3 in B1 - B0
byte 1: least significant 8 bits of the initial 10 bit frequency data
byte 2: MSN = 4 bit ATN data (initial value)
LSN = 0 0 F0 F1, the top 2 bits of the initial frequency data
byte 3: NLEN, number of steps in the frequency sweep, 2 to 255 (0 => 256)
byte 4: FPS | FPSV, prescaler reload value and initial FPSV, 0 - 15 (0 => 16)
byte 5: FSTEP, sweep step size, 1 to 127, -1 to -128
byte 6: ALEN | ASTEP
ALEN = number of steps in the attenuation sweep
ASTEP = step size, 1 to 7, -1 to -8
byte 7: APS | APSV, prescaler reload value and initial APSV, 1 to 15 (0 => 16)

* Noise notes: special case Type 2 notes

See Figure 8.

Noise notes are notes that are played on the sound chip noise generator (CH#0). They are stored in ROM as a special case of a Type 2 note, fixed frequency and swept attenuation. They consist of white noise with superimposed attenuation decay, which creates a percussive effect, such as a snare drum note.

Instead of frequency information, noise notes are stored with three bits of noise control data: FB NF0 NF1 (see TI data sheets 76489), which remain constant throughout the note. Experimentation with various values can result in credible percussion effects.

NLEN, as is the case for a regular Type 2 note, directly determines the duration of a noise note.

The sound chip noise generator is unlike the other generators in that sending it redundant data (i.e., the same data that it has stored in its internal registers) has an audible effect on its output. In particular, whenever control data, redundant or not, is sent to the noise generator, its internal shift register is reset, causing a short pop or click to be heard. This isn't annoying on an occasional basis, or when a new noise starts, but remember: PLAY_SONGS is sending data to all four channels every 16.7ms. This would cause a noticeable lack of "whiteness" in the noise generator's output.

PLAY_SONGS avoids this problem by referencing a dedicated CART RAM location, SAVE_CTRL (see Figure 9) each time before it sends control data to the noise generator. SAVE_CTRL contains the control data that was output to the noise generator the last time through PLAY_SONGS. If the noise control data in the pointed to song data area = SAVE_CTRL, PLAY_SONGS doesn't send it out again. If there is new data to be sent, that data is output and SAVE_CTRL is updated.

byte 0: header, CH#0 in B7 - B6, note type = 2 in B1 - B0
byte 1: MSN = 4 bit ATN data (initial value)
LSN = 0 FB NF0 NF1, noise control data
byte 2: NLEN, duration of note: NLEN * 16.7ms
byte 3: ALEN : ASTEP
ALEN = number of steps in the attenuation sweep
ASTEP = step size, 1 to 7, -1 to -8
byte 4: APS : APSV, prescaler reload value and initial APSV, 1 to 15 (0 => 16)

OPERATING SYSTEM ROUTINES

INIT_SOUND

Contains ENTRY POINT: ALL_OFF

INIT_SOUND, usually called right after power on, turns off the sound generators, initializes the CART RAM locations to be used as song data areas, and sets up the four channel data area pointers. Specifically, it:

1) directly turns off all four sound generators.
2) initializes PTR_TO_LST_OF_SND_ADDRS, a dedicated 16 bit CPU RAM pointer which other sound routines expect to contain the base address of a list in CART ROM (called LST_OF_SND_ADDRS) of the starting addresses of each song's data area and note list. The address of LST_OF_SND_ADDRS is passed to INIT_SOUND in HL.
3) stores the sound-inactive code (0FFH) into byte 0 of n song data areas. n is passed in B and = the total number of song data areas used by the game.
4) stores an end of data area code (00) following the last data area.
5) sets the four pointers to the data areas for the songs to be played on each channel, PTR_TO_S_ON_x (x = 0-3), to a dummy inactive area (DUM_AREA, which is actually a single 0FFH byte within INIT_SOU).
6) sets SAVE_CTRL to an initial value of 0FFH

INPUT: n
TYPE: 8 bit constant
PASSED: in B
DESCRIPTION: number of song data area used by the game

INPUT: LST_OF_SND_ADDRS
TYPE: 16 bit address
PASSED: in HL
DESCRIPTION: LST_OF_SND_ADDRS is the base address of a list of the starting addresses of each song's data area and note list.

OUTPUT: 1) turns off all sound generators
2) initializes PTR_TO_LST_OF_SND_ADDRS
3) writes inactive code to byte 0 of n song data areas
4) stores 00 at end of song data areas
5) sets the 4 channel song pointers to the inactive DUM_AREA
6) sets SAVE_CTRL to 0FFH

ALL_OFF

ALL_OFF directly turns off all four sound generators, but does nothing to any song data areas or the 4 channel data pointers.

INPUT: none

OUTPUT: turns off all sound generators

JUKE_BOX

JUKE_BOX is called to start a song. Using a song number passed in B, JUKE_BOX loads the data for the song's first note into the appropriate song data area (the address of the area is found by using the song number as an index into the LST_OF_SND_ADDRS table). It also formats the data area's header and sets up the next note pointer. If the song is a special sound effect, its next note pointer is set to the address of the special effect routine. The next time PLAY_SONGS is called, that song's first note will be processed (thereby truncating whatever song had been "playing" in that data area), and the song will have started.

Since starting a new song may have altered the priority structure within the song data areas, JUKE_BOX also calls UP_CH_DATA_PTRS to modify the channel data pointers accordingly.

If JUKE_BOX is called with a song number of a song already in progress, it returns immediately (i.e., it doesn't restart the song).

INPUT: song number to be started
TYPE: 8 bit constant, 1 to 61
PASSED: in B

CALLS: PT_IX_TO_SxDATA, LOAD_NEXT_NOTE, UP_CH_DATA_PTRS

OUTPUT: 1) moves the song's first note data to the appropriate song data area
1) formats byte 0 header of the song's data area
2) points next note pointer in data area (bytes 1&2) to address of first note in song, or address of special sound effect routine
3) updates the channel data pointers on basis of song priorities

SND_MANAGER

SND_MANAGER should be called every VDP interrupt (every 16.7 ms). It assumes that the song data areas are stored contiguously in a data block beginning with the data area assigned to song number one. For each data area, SND_MANAGER, or routines which it calls, processes the appropriate timer and sweep counters and modifies the frequency and attenuation data accordingly. If the data area is assigned to a special effect, SND_MANAGER simply calls that effect, and doesn't modify any data. When a note is finished, SND_MANAGER, using the data area's next note pointer, moves data for the next note of the song into the area and fills in keys bytes within the area to allow proper processing of the data area by the sweep routines it calls (FREQ_SWEEP and ATN_SWEEP). (SND_MANAGER considers a note finished when its frequency duration timers have timed out; see the descriptions of the FREQ_SWEEP and ATN_SWEEP routines) A special effect is responsible for deciding when its over and initiating the next note in the song.

After the operations upon a data area have been performed, the channel data area pointers (PTR_TO_S_ON_x) may be updated (see description of UP_CH_DATA_PTRS in "UTILITIES" section).

If SND_MANAGER reads a header byte (in CART ROM) that has bits 3&4 set, indicating repeat song, it will start the song again by reloading the first note in the song, using the SONGNO portion (B5-B0) of byte 0 in the song's data and the LST_OF_SND_ADDRS to find it.

SND_MANAGER does not output the modified frequency and attenuation data. PLAY_SONGS is usually called just before SND_MANAGER to do this.

Special codes in byte 0 of the song data area indicate:

0FFH	-	data area inactive, do no processing, do not modify channel data area pointer
B5-B0 = 62	-	a special effect is to be played; SND_MANAGER calls the effect routine
00H	-	end of song data areas (SND_MANAGER processes data areas until it sees 0 in byte 0)

NOTE: Song number 1 MUST use the first data area in the block of song data areas.

INPUT: none

CALLS: PROCESS_DATA_AREA, PT_IX_TO_SxDATA

OUTPUT:
1) decrements song duration and sweep timers
2) modifies swept frequency and attenuation values
3) calls special effects routines where necessary
4) restarts the song if indicated
5) may update the channel data area pointers (PTR_TO_S_ON_x)

PLAYSONGS

PLAY_SONGS takes the frequency and attenuation data pointed to by the four channel data area pointers (PTR_TO_S_ON_x) and outputs it to the four sound chip generators. Action is taken on the basis of the each data area's byte 0:

1) If the pointed to data area is active, the frequency and attenuation data are sent to the channel indicated by B7-B6 (CH#) of byte 0 of the pointed to data area.
2) If byte 0 is 0FFH (inactive), the channel to which that pointer is dedicated is sent the OFF attenuation code.
3) If CH# = 0 (noise), the attenuation data is output. If there is no new noise control data to be output (determined by checking dedicated CART RAM location SAVE_CTRL), no control data is sent out. Otherwise, the new control data is output and SAVE_CTRL is updated.

INPUT: none

OUTPUT: through SOUND_PORT,
1) current freq and atn data is output to each tone generator, if song/effect on that channel is active
2) noise generator is sent current atn data, and control data, if new
3) modifies SAVE_CTRL if necessary

FREQ_SWEEP

FREQ_SWEEP is used by SND_MANAGER and special effects routines to create frequency sweeps. It operates upon frequency data stored within a song data area, and is normally called (by SND_MANAGER or a special effect routine) once every VDP interrupt (16.7ms). The start of the data area (address of byte 0) is passed in IX.

FREQ_SWEEP assumes data has been stored as follows (names which may be used to describe the various bytes or byte segments within the data area are indicated; see Figure 1):

byte 3: the least significant 8 bits of that note's frequency (F2 - F9)

byte 4: top 2 bits of that note's frequency: B1 = F0, B2 = F1

byte 5: NLEN - determines the note's duration:
1) if frequency is to be swept, NLEN = number of steps in the sweep: 2 to 255 (0 => 256)
2) if fixed frequency, NLEN * 16.7 ms = duration of the note: 1 to 255 (0 => 256)

byte 6: FPS ¦ FPSV - frequency sweep duration prescaler:
FPS = prescaler reload value: 0 to 15 (0 => 16)
FPSV = temp storage nibble for FPS: init ROM value, 0 to 15 (0 => 16)
duration of sweep (& note) = [((NLEN-1) * FPS) + initial FPSV] * 16.7ms
duration 1st step = initial FPSV * 16.7ms
duration all other steps = FPS * 16.7ms

byte 7: FSTEP - frequency sweep step size: signed 8 bit number, two's complement: 1 to 127, -1 to -128
if FSTEP = 00, frequency is not to be swept, but NLEN is decremented each time called

Parameter limitations:
1) In a frequency sweep, a "step" consists of a single fixed frequency tone; therefore, the minimum number of steps a frequency sweep can have is two (otherwise the frequency wouldn't have "swept").
2) If a note is to be frequency swept, FSTEP must not be 0.
3) The minimum length fixed frequency note has NLEN = 1.
4) Maximum NLEN 0, which is equivalent to 256.

FREQ_SWEEP returns with the Z flag SET if the note (swept or fixed) is over, RESET if the note is not over. (PROCESS_DATA_AREA decides that a note is over when FREQ_SWEEP returns with the Z flag set)

INPUT: 16 bit address of a song data area in CPU RAM
PASSED: in IX
DESCRIPTION: FREQ_SWEEP operates upon frequency data within this song data area

OUTPUT: 1) duration and sweep counters are decremented
2) freq data in bytes 3&4 is modified if note is freq swept
3) returns with Z flag SET if note over, RESET if note not over

ATN_SWEEP

ATN_SWEEP is used to create attenuation sweeps. It operates upon attenuation data stored within a song data area, and is normally called (by PROCESS_DATA_AREA or a special effect routine) once every VDP interrupt (16.7ms). The start of the data area (address of byte 0) is passed in IX.

ATN_SWEEP assumes data has been stored as follows (see Figure 1):

byte 4: ATN - the MSN = 4 bit attenuation

byte 8: ALEN ! ASTEP - no-sweep code or sweep length and step size:
1) if byte 8 = 00, ATN is not to be swept and counters aren't changed
2) if byte 8 non zero, attenuation is to be swept:
  a) ALEN = number of steps in the sweep: 1 to 15 (0 => 16)
  b) ASTEP = sweep step size: 1 to 7, -1 to -8 (signed, 4 bit two's complement)

byte 9: APS ! APSV - attenuation sweep duration prescaler:
1) if attenuation is not swept, byte 9 is not used by ATN_SWEEP
2) if attenuation is to be swept:
APS = prescaler reload value: 1 to 15 (0 => 16)
APSV = temp storage nibble for APS: init ROM value, 1 to 15 (0 => 16)
duration of swept attenuation part of note =
[((ALEN - 1) * APS) + initial APSV] * 16.7 ms
  duration 1st step = initial APSV * 16.7ms
  duration all other steps = APS * 16.7ms

Parameter limitations:
1) In an attenuation sweep, a "step" consists of a tone (swept or not) played at a fixed attenuation level; so, the minimum number of steps an attenuation sweep can have is two (otherwise the attenuation wouldn't have "swept"). Therefore, the minimum ALEN value is 2 (0 => 16)
2) If a note is to be attenuation swept, byte 8 must not be 00.
3) The absolute value of ASTEP must be > = 1.

If byte 8 is 00, ATN_SWEEP returns immediately with Z flag SET (the sweep is over or the note was never swept), and doesn't modify any counters. When a sweep finishes, ATN_SWEEP sets byte 8 to 00 and returns with the Z flag SET. If a sweep is in progress, ATN_SWEEP returns with the Z flag RESET. (NOTE: PROCESS_DATA_AREA decides that a note is over when FREQ_SWEEP returns with Z set: the length of a note has nothing to do with when its attn sweep is over)

INPUT: 16 bit address of a song data area in CPU RAM
PASSED: in IX
DESCRIPTION: ATN_SWEEP operates upon frequency data within this song data area

OUTPUT: 1) duration and sweep counters are decremented if sweep in progress
2) atn data in byte 4 is modified if note is atn swept
3) RETs C SET, byte 8 = 0 if sweep is over or note was never swept
RETs C RESET if sweep in progress

PROCESS_DATA_AREA

PROCESS_DATA_AREA is called by SND_MANAGER. For an active data area (address of byte 0 passed in IX), PROCESS_DATA_AREA modifies the timers, sweep counters, frequency, and attenuation data by calling FREQ_SWEEP and ATN_SWEEP. If a note finishes during the current pass through PROCESS_DATA_AREA, the next note in the song is examined and its data is loaded into the data area (calls LOAD_NEXT_NOTE). Then, in order to maintain the song data area priority structure, the CH# : SONGNO of the newly loaded note is compared to the CH# : SONGNO of the previous note: if there is a difference, UP_CH_DATA_PTRS is called to adjust the channel data area pointers in response to the change caused by loading the next note.

If the data area is being used by a special sound effect, PROCESS_DATA_AREA calls the sound effect routine whose address is stored in bytes 1&2 of the data area (the actual address called is routine + 7: see discussion of special sound effects).

If the data area is inactive, PROCESS_DATA_AREA returns immediately (no processing occurs).

INPUT: address of byte 0 of a song data area
PASSED: in IX

CALLS: ATN_SWEEP, FREQ_SWEEP, LOAD_NEXT_NOTE, UP_CH_DATA_PTRS, AREA_SONG_IS

OUTPUT: 1) if active, modifies song data area's timer, freq, and atn data
2) loads the next note's data when a note is finished
3) if special sound effect routine using data area, calls it
4) when necessary, updates the channel data area pointers

LOAD_NEXT_NOTE

Called by PROCESS_DATA_AREA and JUKE_BOX, LOAD_NEXT_NOTE examines the next note to be played in a data area (address byte 0 passed in IX) and moves its data into the area. It fills in bytes (e.g., to indicate swept or not swept) where appropriate, based upon note type. If the next "note" is a special sound effect, its address is saved in bytes 1&2 and the address of the routine + 0 is called, with the address of the note to follow the effect passed in HL and SONGNO passed in A. This will cause the special effect routine to save both these values. Then, the special effect routine + 7 is called, which allows the routine to initialize the song data area for the first pass through PLAY_SONGS. (see discussion of special sound effects)

Prior to moving the next note data, LOAD_NEXT_NOTE saves the data area's byte 0 (CH# : SONGONO) and stores the song inactive code (0FFH) there. The last thing LOAD_NEXT_NOTE does is restore byte 0, loading CH# with the CH# : SONGNO of the new note (usually the same as the old note). If the new note is a special sound effect, 62 is returned as the SONGNO part of byte 0.

INPUT: address of byte 0 of a song data area
PASSED: in IX

OUTPUT:
1) sets up song data area with data from next note to be played
2) for next note = special sound effect, calls the effect twice, first with the address of the following note in the song and the song's SONGNO, and then once more to allow the effect to initialize the song data area
3) if next note is "normal", loads CH# : SONGNO in byte 0 with CH# : SONGNO of new note
4) returns with byte 0 = 0FFH if song over, SONGNO = 62 if next note is a sound effect

UTILITIES

The following are O/S utility routines, used by the main O/S sound programs, that may be of use to the cartridge programmer:

*** AREA_SONG_IS ***

The address of byte 0 of a song data area is passed in IX. The song number of the song using that area is returned in A (0FFH if inactive). If a special effect was using that area, 62 is returned in A and HL is returned with the address of the special sound effect routine.

*** UPATNCTRL ***

Perform single byte update of the snd chip noise control register or any attenuation register. IX is passed pointing to byte 0 of a song data area, MSN register C = formatted channel attenuation code.

*** UPFREQ ***

Perform double byte update of a sound chip frequency register. IX is passed pointing to byte0 of a song data area, MSN register D = formatted channel frequency code.

*** DECLSN ***

Without affecting the MSN, decrement the LSN of the byte pointed to by HL. HL remains the same.
RET with Z flag set if dec LSN results in 0, reset otherwise.
RET with C flag set if dec LSN results in -1, reset otherwise.

*** DECMSN ***

Without affecting the LSN, decrement the MSN of the byte pointed to by HL. HL remains the same.
RET with Z flag set if dec MSN results in 0, reset otherwise.
RET with C flag set if dec MSN results in -1, reset otherwise.

*** MSNTOLSN ***

Copy MSN of the byte pointed to by HL to the LSN of that byte. HL remains the same.

*** ADD816 ***

Adds 8 bit two's complement signed value passed in A to the 16 bit location pointed to by HL. Result is stored in the 16 bit location.

*** PT_IX_TO_SxDATA ***

A SONGNO is passed in B. PT_IX_TO_SxDATA returns with IX pointing to the song data area which is used by that SONGNO.

*** UP_CH_DATA_PTRS ***

UP_CH_DATA_PTRS adjusts each channel data pointer to point to the highest priority (ordinal last) song data area that uses that channel. It is called whenever a change has been made to a song data area that requires modification of the channel data pointers.

All 4 channel data pointers (PTR_TO_S_ON_x) are initially pointed to a dummy inactive area, DUM_AREA. Then, moving in order from the first data area to the last, CH# in byte 0 of each data area is examined, and the corresponding channel data pointer is pointed to that data area. Thus, by the time the routine is done, each channel data pointer is pointing to the last active data area that contains data to be sent to that channel. If none of the active data areas used a particular channel, then that channel remains pointing to DUM_AREA (and therefore its generator will be turned off next time through PLAY_SONGS).

*** LEAVE_EFFECT ***

LEAVE_EFFECT, called by a special sound effect routine when it's finished, restores the SONGNO of the song to which the effect belongs to B5 - B0 of byte 0 in the effect's data area, and loads bytes 1 & 2 with the address of the next note in the song. The address of the 1 byte SONGNO (saved by the effect when it was first called) is passed in DE. The 2 byte address of the next note in the song, also saved by the effect, is passed in HL. IX is assumed to be pointing to byte 0 of the data area to which the song number is to be restored. Bits 7 & 6 of the saved SONGNO byte are not stored into byte 0, and therefore may be used during the course of the effect to store any useful flag information.

SPECIAL SOUND EFFECTS

* Sound effects as notes within a song

Sounds which do not fit one of the six categories of "normal" musical notes can be created and played throughout the course of a song as "special effect" notes. Unlike normal musical notes, which are stored in ROM as tables of frequency/control and attenuation data, a special effect's data are determined algolrithmically by a custom routine written by the cartridge programmer. Special effect notes can also be used to generate sounds that could have been comprised of many normal notes, but which are more efficiently (in terms of ROM space used) computed by a short program.

These notes use the same song data area as the song within which they are contained, and they are stored in the song's ROM note list with a one byte header as are normal notes. However, the bytes following the ROM header do not contain data to be directly loaded into the song data area. The header (see Figure 2), which specifies the channel upon which to play the effect (which is usually the same as the channel used by the rest of the notes in the song), is followed by a two byte address of a routine written by the cartridge programmer which will be called every 16.7ms by PROCESS_DATA_AREA. When called, this special affect routine should compute data values and store them at the appropriate locations within the song data area. (In fact, many effect routines may call the O/S routines FREQ_SWEEP or ATN_SWEEP, which also require that data be ordered appropriately within a song data area) This computed data will then be output on the next pass through PLAY_SONGS (assuming that this song data area has the highest priority of any data area using the same channel).

Variables required by the effect which will not be output may be stored wherever the programmer desires. Free locations within the song's data area might as well be used for effect variable storage, since the entire ten byte area is reserved for the song anyway. If no free locations exist within a data area, which would be the case if an effect required both frequency and attenuation to be swept, the effect can store the remaining needed variables wherever convenient.

In order to interact properly with the O/S sound routines, each special effect routine must conform to a certain format. A description of that format, and how an effect interacts with the O/S routines, follows:

WHEN AN EFFECT BEGINS - When loading a new note, if LOAD_NEXT_NOTE sees that the note to be loaded is a special effect:

1) It stores in byte 0 of the song's data area the effect's CH# and a SONGNO of 62. SONGNO = 62 is used later by PROCESS_DATA_AREA to detect the fact that an effect is using the data area.

2) It then takes the address of the special effect routine (lets's call it SFX) from ROM and puts it into bytes 1&2 (NEXT_NOTE_PTR).

3) LOAD_NEXT_NOTE then calculates the ROM address of the header of the next note in the song, stores that address in HL, puts the song's SONGNO in A, and calls SFX + 0. In every special effect routine at SFX + 0, there MUST be the following code which saves the two passed values (see Figure 9):

```
SFX:      LD (SAVE_x_NNP),HL
          LD (SAVE_x_SONGNO),A
          RET
SFX+7:    code for sound effect starts here
          ...
```

where SAVE_x_NNP is a two byte location used by all the sound effect notes in the current song to save the address of the next note in the song, and SAVE_x_SONGNO is the address of a byte where the song number is saved. The programmer may put SAVE_x_NNP and SAVE_x_SONGNO wherever desired, including somewhere within the song data area.

Thus, calling SFX + 0 allows each effect routine to save the next note's address and the song's SONGNO.

1ST PASS THROUGH EFFECT - After calling SFX + 0, LOAD_NEXT_NOTE calls SFX + 7 for the first pass through the body of the routine. At this location, there should be code which initializes the appropriate bytes within the song data area, as the next pass through PLAY_SONGS, subject to the data area priority system, will output this initial data in normal fashion.

As will be seen below, this same location (SFX + 7) will be called every 16.7ms by PROCESS_DATA_AREA to modify the data within the area. Therefore, the code at SFX + 7 must know which pass is in effect, so that the song data area will be initialized only on the first pass. A convenient way of doing this is to test bit 7 of SAVE_x_SONGNO, the byte which contains the saved song number. On the 1st pass through the effect, bit 7 (and bit 6) will be zero, since the largest possible SONGNO (62) would not set this bit. If bit 7 is zero, then, code to initialize the data area can be executed and bit 7 reset to prevent re-initialization. I.e.,

```
SFX+7:    LD HL,SAVE_x_SONGNO
          BIT 7,(HL)
          JR NZ,NOT_PASS_1
            SET 7,(HL)        ;to prevent further passes thru inits
            ...               ;initialize bytes within the data area here
            RET               ;to LOAD_NEXT_NOTE
NOT_PASS_1: ....              ;code for pass 2 or greater starts here
```

PRIORITY UPDATE - After calling SFX + 0 and SFX + 7, LOAD_NEXT_NOTE will return to PROCESS_DATA_AREA, which checks to see if loading a new note has caused a change in either the channel used by the song (this happens with noise notes within a musical song) or the song number. If a change has occured, UP_CH_DATA_PTRS will be called, which updates the data pointers on the basis of priority within the block of song data areas (see description of this routine in the preceeding "UTILITIES" section). Since a special effect note will cause a change in the song number (from whatever it was to 62), UP_CH_DATA_PTRS will always be called whenever an effect note is loaded.

SECOND PASS OR GREATER - The next time PROCESS_DATA_AREA is called (from SND_MANAGER), which will be 16.7ms after PLAY_SONGS has sent out the effect's initial data, it will detect the fact that an effect is using the song data area (by seeing a SONGNO of 62) and will JUMP to SFX + 7, rather than calling the frequency and attenuation sweep routines as it would for a normal note. This will result in the first pass through the part of the body of the effect routine that actually does computation and adjusts the data values within the data area. When the effect routine has completed its modifications to the data area and performs a RET, control is transferred back to SND_MANAGER, which then moves on to the next song data area to be processed.

This process will be repeated every 16.7ms until the effect routine itself decides that it's over and takes action to load the next note in the song.

WHEN AN EFFECT IS OVER - Prior to performing a RET, the effect routine must decide whether the effect note has finished. If it has, NEXT_NOTE_PTR within the data area must be set to the address of the next note in the song and SONGNO must be restored to byte 0. This can be done by calling the O/S routine LEAVE_EFFECT which does this. The address of SAVE_x_NNP must be passed in HL and the address of SAVE_x_SONGNO must be passed in DE. Finally, the effect should JUMP to EFXOVER, a location within PROCESS_DATA_AREA which would normally be reached once a note is over. The code there takes care of loading the next note in the song. Thus, the final code of each effect routine will look as follows:

```
RET if effect not over
  LD HL,(SAVE_x_NNP)       ;HL := addr next note in song
  LD DE,SAVE_x_SONGNO      ;DE := addr saved song number
  CALL LEAVE_EFFECT        ;to restore them to bytes 0 - 2 in data area
  JP EFXOVER               ;in PROCESS_DATA_AREA to load song's next note
```

The entire above described sequence is summarized in Figure 9.

* A sound effect as a single sound

A stand alone sound effect can be implemented within the previously mentioned structures simply by creating a single note song. The single note is the effect and would be followed by an end of song code (or repeat code if you wish the effect to go on forever).

Many stand alone effects may want to use more than one tone generator channel: e.g., a special laser zap that momentarily requires all three tone generators, or, as is often the case, a white noise effect of particular character that requires the noise channel shift rate to be modified by channel three (see TI data sheets). In these cases, the effect's routine will have to modify data in several data areas whenever called. The song data areas used by such effects are subject to the normal priority structure. E.g., if you wish a two channel effect to temporarily overwrite the harmony and bass lines of a repeating song, the effect must have been assigned two data areas of higher priority (ordinally later in the block of song data areas). If it is not necessary to maintain any underlying songs, an effect can share data areas to conserve RAM space, with the understanding that, as usual, songs or sounds that share the same song data area truncate each other. A multi-channel effect (a chord note, say) may be included as a note within a song, but, again, the song data area priority structure determines what will finally be heard.

Providing for a typical game's sound generation needs might require eight song data areas: four for an underlying, repeating song(s) (three areas for the three tone generators and one for the noise generator used for percussion notes), and four for higher priority, occasional sound effects (which would temporarily overwrite the repeating songs, but truncate each other).

* Pseudo code listings of main routines

The following two pages contain pseudo code descriptions of most of the O/S sound routines. Some computational details are not shown, but all jumps, calls, returns, pushes and pops are listed.

Terminology:
":=" is used as the assignment statement, and "(xx)" means the contents of the memory location pointed to by xx, where "xx" is HL, IX, etc.

The structure of each description is as follows:

```
*** name of routine ***
the value expected for passed parameters (if any)

pseudo code description
of the routine, uninterrupted
by blank lines
RET
```

```
*** INIT_SOUND ***
HL = addr of LST_OF_SND_ADDRS
B = number of song data areas used by game

set RAM word PTR_TO_LST_OF_SND_ADDRS to value passed in HL
pt HL to byte 0 in 1st song data area
B1  (HL) := inactive code (0FFH)
HL := HL + 10
DJNZ B1 (set B areas inactive)
(HL) := end of data area code (0)
load all 4 channel data area pointers with the
addr of a dummy inactive area (DUMAREA)
SAVE_CTRL := 0FFH
ALL_OFF: turn off all 4 generators directly
RET

DUMAREA DEFB 0FFH
```

```
*** UP_CH_DATA_PTRS ***

PUSH IX to save it
set all 4 ch data ptrs to a dummy, inactive area
CALL PT_IX_TO_SxDATA, song # 1
LOOP
  IF byte 0 indicates the end of the song data areas JR DONE
    IF byte 0 indicates an active area
      set HL to address of this area's channel data pointer
      (i.e., HL := addr PTR_TO_S_ON_0 + (CH# this area  x 2))
      PUSH IX
      POP DE (DE := addr byte 0 this area)
      (HL) := E, (HL+1) := D
    ENDIF
    IX := IX + 10
  ENDIF
REPEAT LOOP
DONE  POP IX to restore it
RET
```

```
*** FREQ_SWEEP ***
IX = addr byte 0 of a song data area

IF FSTEP = 0 note is not to be swept
  A := NLEN
  DEC A
  RET Z (leave if note over, Z flag SET)
    NLEN := A (store decremented NLEN)
    RET (note not over, Z flag RESET)
ENDIF
PUSH IX, POP HL (pt HL to byte 0)
HL := HL + offset within data area of FPSV
CALL DECLSN to decrement FPSV
IF Z flag SET, FPSV has timed out
  CALL MSNTOLSN to reload FPSV
  A := NLEN
  DEC A
  RET Z (leave if sweep over with Z flag SET)
    NLEN := A (store decremented NLEN)
    point HL to FREQ
    A := FSTEP
    CALL ADD816 to add FSTEP to FREQ
    RESET bit 2 in hi byte FREQ
    (in case of overflow from addition)
    OR 0FFH to RESET Z flag
ENDIF
RET
```

```
*** ATN_SWEEP ***
IX = addr byte 0 of a song data area

RET with Z flag SET if byte 8 = 0
(i.e., note atn not to be swept)
  PUSH IX, POP HL (pt HL to byte 0)
  HL := HL + offset within data area of APSV
  CALL DECLSN to decrement APSV
  IF Z flag SET, APSV has timed out
    CALL MSNTOLSN to reload APSV
    pt HL to ALEN (# of steps in atn sweep)
    CALL DECLSN to decrement ALEN
    IF Z flag RESET, sweep not over yet
      ATN := ATN + ASTEP
      (4 bit add, overflow ignored)
      OR 0FFH to RESET Z flag
    ELSE Z flag is SET (sweep is over)
      byte 8 := 0 to indicate sweep over
    ENDIF
  ENDIF
RET
```

```
*** TONE_OUT ***

IX = (PTR_TO_S_ON_x), i.e., IX pts to byte 0 data area of song for CHx
A set for CHx OFF code
MSN C set for CHx attenuation
MSN D set for CHx frequency

IF area INACTIVE
  turn off CHx
ELSE
  CALL UPATNCTRL (send out attenuation)
  CALL UPFREQ (send out frequency)
ENDIF
RET
```

```
*** PT_IX_TO_SxDATA ***
B = a song number

HL := addr of LST_OF_SND_ADDRS
HL := HL - 2
BC := 4 x SONGNO
HL := HL + BC (i.e., HL now points to SxDATA's entry in LST_OF_SND_ADDRS)
E := (HL), D := (HL+1)
PUSH DE
POP IX (IX := addr byte 0 of this song's data area)
RET
```

```
*** JUKE_BOX ***
B = SONGNO to be started

PUSH BC
CALL PT_IX_TO_SxDATA (IX set)
POP BC
RET if song in progress
  byte 0 := SONGNO
  set NEXT_NOTE_PTR to 1st note in song
  CALL LOAD_NEXT_NOTE
  CALL UP_CH_DATA_PTRS
  RET


*** PLAY_SONGS ***

set A for CH1 OFF code
set MSN C for CH1 attenuation
set MSN D for CH1 frequency
IX := (PTR_TO_S_ON_1)
(i.e., pt IX to byte 0 data area of song for CH1)
CALL TONE_OUT
set A for CH2 OFF code
set MSN C for CH2 attenuation
set MSN D for CH2 frequency
IX := (PTR_TO_S_ON_2)
(i.e., pt IX to byte 0 data area of song for CH2)
CALL TONE_OUT
set A for CH3 OFF code
set MSN C for CH3 attenuation
set MSN D for CH3 frequency
IX := (PTR_TO_S_ON_3)
(i.e., pt IX to byte 0 data area of song for CH3)
CALL TONE_OUT
set A for CH0 OFF code
set MSN C for CH0 attenuation
IX := (PTR_TO_S_ON_0)
(i.e., pt IX to byte 0 data area of song for CH0)
IF area INACTIVE
  turn off CH0
ELSE
  CALL UPATNCTRL (send current atn)
  set LSN A for current ctrl data
  IF current ctrl data diff from last
    reload SAVE_CTRL
    CALL UPATNCTRL (send new ctrl)
  ENDIF
ENDIF
RET


*** SND_MANAGER ***

CALL PT_IX_TO_SxDATA, song 01
LOOP
  RET if end of song data areas
    CALL PROCESS_DATA_AREA
    pt IX to byte 0 next song data area
REPEAT LOOP


*** PROCESS_DATA_AREA ***
IX = addr byte 0 of a song data area

CALL AREA_SONG_IS
RET if area INACTIVE
  IF SONGNO = 62
    JP SFX+7 (RET from SFX)
  ENDIF
    CALL ATN_SWEEP
    CALL FREQ_SWEEP
    IF note over
EFXOVER: PUSH CH# : SONGNO note just over
      CALL LOAD_NEXT_NOTE
      POP CH# : SONGNO note just over
      IF CH# : SONGNO newly loaded note not =
      CH# : SONGNO note just over
        CALL UP_CH_DATA_PTRS
      ENDIF
    ENDIF
    RET


*** LOAD_NEXT_NOTE ***
IX = addr byte 0 of a song data area
byte 0 = CH# (or 00) : SONGNO
(SFX = addr of a special effect note's routine)

PUSH 0 B : SONGNO (CH# not pushed)
deactivate area (byte 0 := FF)
A := (NEXT_NOTE_PTR) (header new ROM note)

CASE header type of
1)rest
  PUSH header of new ROM note
  set NEXT_NOTE_PTR for next note in song
  (i.e., the note after this new note)
  set bytes in song data area:
    ATN := off
    NLEN := 5 bit rest duration
    FSTEP := 0 (i.e., no freq sweep)
    ASTEP := 0 (no atn sweep)
  JP MODB0
2)end of song:
  IF end repeat
    POP BC (B := SONGNO)
    CALL JUKE_BOX to reload 1st note of this song
    RET (to PROCESS_DATA_AREA)
  ENDIF
    PUSH inactive code
    JP MODB0
3)special effect:
  POP IY (IY := SONGNO)
  PUSH IY to put SONGNO back on stack
  PUSH header new ROM note
  set NEXT_NOTE_PTR, bytes 1&2, to SFX
  (address special effect routine)
  set DE to SFX
  HL := addr next note in song
  (i.e, the note after this new effect note)
  PUSH IY, POP AF (A := SONGNO)
  PUSH DE, POP IY (IY := SFX)
  DE := PASS1, PUSH DE
  JP (IY), i.e., "CALL (IY)", RET to PASS1
  (SFX saves SONGNO & addr next note)
  PASS1: IY := IY + 7
  DE := MODB0, PUSH DE
  JP (IY), i.e., "CALL (IY+7)", RET to MODB0
  (SFX+7 loads initial effect data)
4)normal note:
  CASE note type
    0)NEXT_NOTE_PTR := adr song's nxt note
      move 3 bytes ROM note data to RAM
      FSTEP := 0 (no freq sweep)
      ASTEP := 0 (no atn sweep)
      JR MODB0
    1)NEXT_NOTE_PTR := adr song's nxt note
      move 5 bytes ROM note data to RAM
      ASTEP := 0 (no atn sweep)
    2)NEXT_NOTE_PTR := adr song's nxt note
      move 5 bytes ROM note data to RAM
      FSTEP := 0 (no freq sweep)
      JR MODB0
    3)NEXT_NOTE_PTR := adr song's nxt note
      move 7 bytes ROM note data to RAM
  ENDCASE
ENDCASE

MODB0: PUSH IX
POP HL to point to byte 0
POP AF (A := header new note)
POP BC (B := SONGNO)
RET if header is inactive (i.e., song is over)
  IF header is for a special effect
    B := 62, the SONGNO for all effect notes
  ENDIF
  byte 0 := CH# (from header new note) : SONGNO (from B)
  RET
```

S x DATA

FIGURE 1

Description: Storage area for the various timers and output data for song number x. The song data areas MUST be stored in a contiguous block of CPU RAM and the data area used by song number one MUST be the first data area in the block. Song data area storage is allocated according to addresses stored in LST_OF_SND_ADDRS, a table stored in CART ROM.

Byte 0 of each data area, in addition to giving CH# and song number, can indicate two special conditions:
  byte 0 = OFFH: song(s) using this data area are inactive
  byte 0 = OOH: indicates end of song data areas

If SONGNO = 62, the address of a special sound effect routine is stored in bytes 1 and 2.

Length in bytes: 10
Location: CPU RAM
Beginning address: pointed to by a 16 bit entry in LST_OF_SND_ADDRS

```
       !        Contents          !
Offset ! B7 B6 B5 B4 B3 B2 B1 B0 ! Description
--------------------------------------------------------------------------

         -------------------------- B7 - B6: song channel number, 0 to 3
  0      ! CH# !      SONGNO       ! B5 - B0: song number, 1 to 61
         -------------------------- SONGNO = 62, snd effect adr in next 2 bytes

         -------------------------- usually, the addr of the next note in song;
  1      ! the LSB of an address...! if SONGNO = OFEH, this is the LSB of the
         -------------------------- addr of the special sound.effect routine

         -------------------------- usually, the addr of the next note in song;
  2      ! the MSB of an address...! if SONGNO = OFEH, this is the MSB of the
         -------------------------- addr of the special sound effect routine

         --------------------------
  3      ! F2 F3 F4 F5 F6 F7 F8 F9 ! bottom 8 bits of 10 bit freq data
         --------------------------

         -------------------------- if CH# = 0 (noise): ATN ! 0 FB SHIFT
  4      !ATN!CTRL or ATN!0 0 F0 F1! if CH# = 1 - 3 (tone): MSN = 4 bit ATN,
         -------------------------- LSN = top 2 bits freq (0  0 F0 F1)

         -------------------------- determines duration of note:
  5      !          NLEN           ! if freq swept, = # of steps in the sweep
         -------------------------- if not, NLEN * 16.7ms  = duration of note

         -------------------------- freq sweep duration prescaler:
  6      !    FPS     !    FPSV    ! FPS = prescaler reload value
         -------------------------- FPSV = temp FPS variable storage

         -------------------------- freq sweep step size: 1 to 127, -1 to -128
  7      !         FSTEP           ! if FSTEP = 0, freq is not to be swept
         --------------------------

         -------------------------- ALEN = # steps in atnswp: 2 - 15 (0 => 16)
  8      !   ASTEP    !    ALEN    ! ASTEP = step size: 1 to 7, -1 to -8
         -------------------------- if whole byte = 00, atn not to be swept

         -------------------------- atn duration prescaler:
  9      !    APS     !    APSV    ! APS = prescaler reload value
         -------------------------- APSV = temp APS variable storage
```

DURATIONS:

```
fixed frequency =        NLEN * 16.7ms
frequency sweep =        [((NLEN - 1) * FPS) + initial FPSV] * 16.7ms
duration 1st step =      initial FPSV * 16.7ms
duration all others =    FPS
  FPS: 0 to 15 (0 => 16)
  FPSV: 0 to 15 (0 => 16)
  NLEN: 0 to 255 (0 => 256)
attenuation sweep =      [((ALEN - 1) * APS) + initial APSV] * 16.7ms
duration 1st step =      initial APSV * 16.7ms
duration all others =    APS
  APS: 0 to 15 (0 => 16)
  APSV: 0 to 15 (0 => 16)
  ALEN: 0 to 15 (0 => 16)
```

FIGURE 2

Note Header

Length in bytes: 1
Location: begins each block of 1 to 10 bytes of note data in CART ROM

```
        !        Contents         !
Offset ! B7 B6 B5 B4 B3 B2 B1 B0 ! Description
--------------------------------------------------------------------------
                                   *** REST

        -------------------------- if B7 = 1, header describes a rest:
0       ! CH# ! 1!   duration   ! B7 -B6 = channel number, 0 - 3
        -------------------------- B4 - B0 = duration, 1 to 31

or, if B5 = 0, header preceeds note data or is special indicator:

                                   *** NOTE

        -------------------------- note data follows:
0       !  CH# ! 0! 0! 0! 0! type ! B7 - B6 = channel number, 0 - 3
        -------------------------- B1 - B0 = note type, 0 -3.

or                                 *** END OF SONG / REPEAT SONG

        -------------------------- if B4 = 1, end of song on channel in B7-B6:
0       ! CH#  ! 0! 1! R! 0! 0! 0 ! if B3 = 1, repeat song forever
        -------------------------- if B3 = 0, don't repeat

or                                 *** SPECIAL EFFECT

        -------------------------- this note is to be "played" by a special
0       !  CH# ! 0! 0! 0! 1! 0! 0 ! sound effect routine whose addr is
        -------------------------- contained in the following 2 bytes
```

REST DURATION = duration * 16.7ms
duration: 1 to 31

FIGURE 3

LST_OF_SND_ADDRS

Description: LST_OF_SND_ADDRS is a list of the starting addresses of each song's data area and note list. They are used by JUKE_BOX as source (note list) and destination (song data area) pointers. Each song's entries are stored as follows:

Byte 1: LSB of the address of the start of that song's note list
Byte 2: MSB of the address of the start of that song's note list
Byte 3: LSB of the address of the start of that song's data area
Byte 4: MSB of the address of the start of that song's data area

The beginning address of LST_OF_SND_ADDRS is stored in a dedicated CPU RAM 16 bit word, PTR_TO_LST_OF_SND_ADDRS (xxxxH), allowing the cartridge programmer to place LST_OF_SND_ADDRS wherever desired.

NOTE: In other data structures, six bits are allocated for the song number (SONGNO). However, song numbers 0, 62, and 63 are used as special indicators, leaving song numbers 1 - 61 available. Therefore, the first entry in LST_OF_SND_ADDRS is for song number 1.

Length in bytes: 4 x total number of songs
Location: CART ROM
Beginning address: pointed to by CPU RAM word @LST_OF_SND_ADDRS (xxxxH)

```
        !        Contents          !
 Offset ! B7 B6 B5 B4 B3 B2 B1 B0 ! Description
-------------------------------------------------------------------------
        -------------------------- of starting adr of the note list for
   0    !          LSB             ! song
        -------------------------- number 1

        -------------------------- of starting adr of the note list for
   1    !          MSB             ! song
        -------------------------- number 1

        -------------------------- of starting adr of the data area for
   2    !          LSB             ! song
        -------------------------- number 1

        -------------------------- of starting adr of the data area for
   3    !          MSB             ! song
        -------------------------- number 1

        -------------------------- of starting adr of the note list for
   4    !          LSB             ! song
        -------------------------- number 2
 etc...
        -------------------------- of starting adr of the data area for
 4 x n  !          MSB             ! song
        -------------------------- number n (n = total number of songs)
```

FIGURE 4

Rests

Length in bytes: 2
Location: CART ROM
Beginning address: pointed to by bytes 1&2 in that song's data area

```
        |        Contents         |
 Offset | B7 B6 B5 B4 B3 B2 B1 B0 | Description
-------------------------------------------------------------------------
           ________________________ if B7 = 1, header describes a rest:
   0    |  CH#  | 1|   duration   | B4 - B0 = duration, 1 to 30
           ________________________
```

REST DURATION = duration * 16.7ms
duration: 1 to 30

Note Type 0:
fixed frequency, fixed attenuation

Length in bytes: 4
Location: CART ROM
Beginning address: pointed to by bytes 1&2 in that song's CPU RAM data area

```
        |        Contents         |
 Offset | B7 B6 B5 B4 B3 B2 B1 B0 | Description
-------------------------------------------------------------------------
          _________________________
   0    |  CH#  | 0| 0| 0| 0| 0| 0 | header
          _________________________

          _________________________
   1    | F2 F3 F4 F5 F6 F7 F8 F9 | least significant 8 bits of 10 bit frq data
          _________________________

          _________________________
   2    |     ATN     | 0  0 F0 F1 | ATN = 4 bit atn data, LSN = top 2 bits freq
          _________________________

          _________________________
   3    |          NLEN            | NLEN * 16.7ms  = duration of note
          _________________________
```

NOTE DURATION = NLEN * 16.7ms
NLEN: 1 to 255 (0 => 256)

FIGURE 7

Note Type 3:
swept frequency, swept attenuation

Length in bytes: 8
Location: CART ROM
Beginning address: pointed to by bytes 1&2 in that song's CPU RAM data area

```
       !        Contents          !
Offset ! D7 D6 D5 D4 D3 D2 D1 D0  ! Description
--------------------------------------------------------------------------

        --------------------------
  0     !  CH#  ! 0! 0! 0! 0! 1! 1 ! header
        --------------------------

        --------------------------
  1     ! F2 F3 F4 F5 F6 F7 F8 F9 ! least sig 8 bits of init 10 bit frq data
        --------------------------

        --------------------------
  2     !    ATN    ! 0  0 F0 F1 ! ATN = init atn data, LSN = top 2 bits freq
        --------------------------

        --------------------------
  3     !          NLEN           ! NLEN = number of steps in the sweep
        --------------------------

        -------------------------- freq sweep duration prescaler:
  4     !    FPS    !  FPSU       ! FPS = prescaler reload value
        -------------------------- FPSU = initial FPSU

        --------------------------
  5     !          FSTEP          ! freq sweep step size: 1 to 127, -1 to -128
        --------------------------

        -------------------------- ALEN = # steps in atnswp: 2 - 15 (0 => 16)
  6     !   ASTEP    !    ALEN    ! ASTEP = step size: 1 to 7, -1 to -8
        -------------------------- if whole byte = 00, atn not to be swept

        -------------------------- atn duration prescaler:
  7     !    APS     !   APSU     ! APS = prescaler reload value
        -------------------------- APSU = initial APSU
```

```
NOTE DURATION =          [((NLEN - 1) * FPS) + initial FPSU] * 16.7ms
duration 1st step =      initial FPSU * 16.7ms
duration all others =    FPS
  FPS: 0 to 15 (0 => 16)
  FPSU: 0 to 15 (0 => 16)
  NLEN: 0 to 255 (0 => 256)

ATN SWEEP DURATION =     [((ALEN - 1) * APS) + initial APSU] * 16.7ms
duration 1st step =      initial APSU * 16.7ms
duration all others =    APS
  APS: 0 to 15 (0 => 16)
  APSU: 0 to 15 (0 => 16)
  ALEN: 0 to 15 (0 => 16)
```

FIGURE 8

Noise notes:
special case type 2 notes

Length in bytes: 5
Location: CART ROM
Beginning address: pointed to by bytes 1&2 in that song's CPU RAM data area

```
       !        Contents           !
Offset ! B7 B6 B5 B4 B3 B2 B1 B0 ! Description
------------------------------------------------------------------------------

        -------------------------
  0     ! 0 ! 0! 0! 0! 0! 0! 1! 0 ! header (CH0 = 0, indicates noise note)
        -------------------------

        ------------------------- MSN = 4 bit noise ATN data (init if swept)
  1     !    ATN     ! 0 FB SHIFT ! LSN = noise control data (SHIFT = NF0 NF1)
        -------------------------

        -------------------------
  2     !          NLEN           ! NLEN x 16.7ms = duration of note
        -------------------------

        ------------------------- ALEN = # steps in atnswp: 2 - 15 (0 => 16)
  3     !   ASTEP    !    ALEN    ! ASTEP = step size: 1 to 7, -1 to -8
        ------------------------- if whole byte = 00, atn not to be swept

        ------------------------- atn duration prescaler:
  4     !    APS     !    APSU    ! APS = prescaler reload value
        ------------------------- APSU = temp APS variable storage
```

```
NOTE DURATION =          NLEN x 16.7ms
  NLEN: 1 to 255 (0 => 256)

ATN SWEEP DURATION =     [((ALEN - 1) x APS) + initial APSU] x 16.7ms
duration 1st step =      initial APSU x 16.7ms
duration all others =    APS
  APS: 0 to 15 (0 => 16)
  APSU: 0 to 15 (0 => 16)
  ALEN: 0 to 15 (0 => 16)
```

FIGURE 9

Dedicated cartridge RAM locations and Special Effect format

Length in bytes: 11
Location: CPU RAM
Beginning address: 7020H

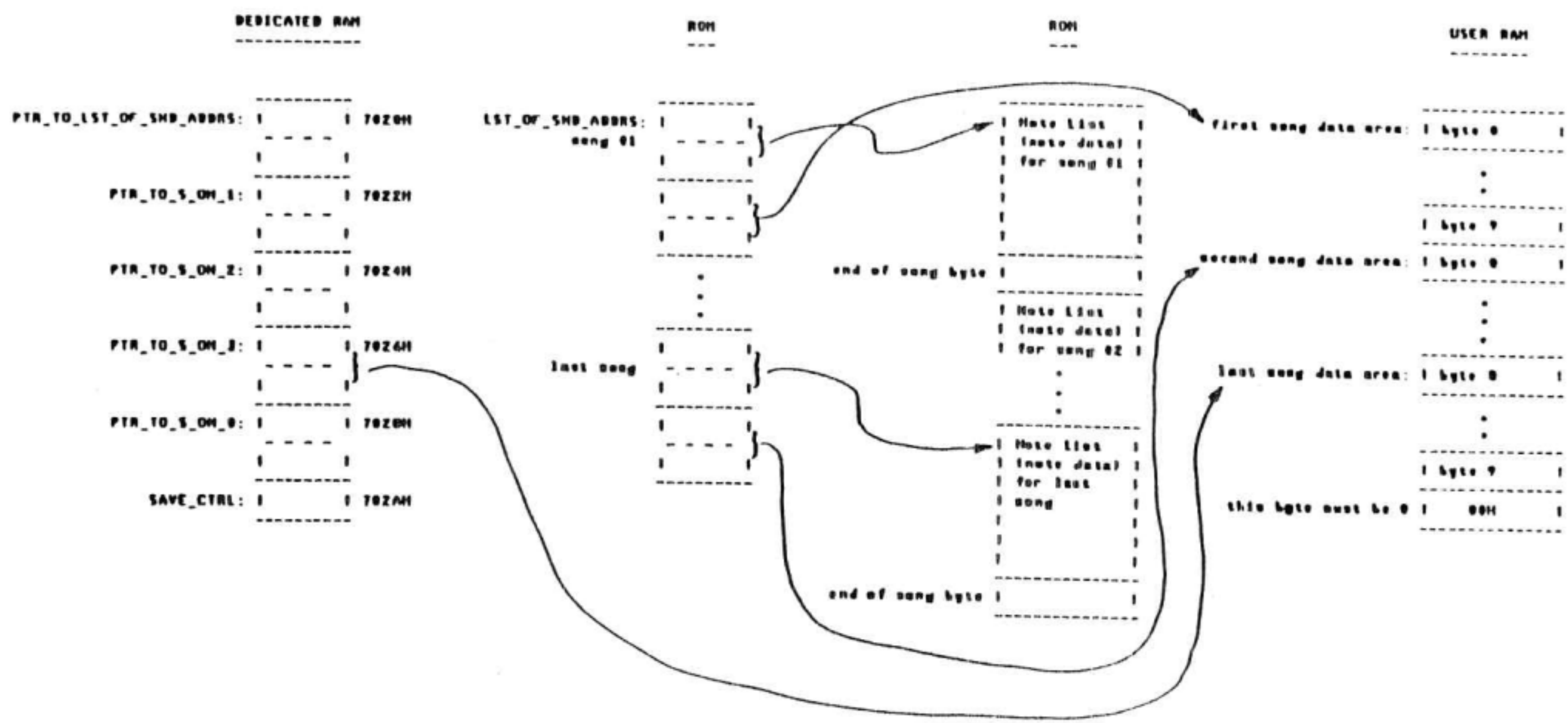
```
PTR_TO_LST_OF_SND_ADDRS   DS 2  ;pointer to start of LST_OF_SND_ADDRS

PTR_TO_S_ON_1   DS 2    ;pointer to data area of song to be played on CH# 1
PTR_TO_S_ON_2   DS 2    ;pointer to data area of song to be played on CH# 2
PTR_TO_S_ON_3   DS 2    ;pointer to data area of song to be played on CH# 3
PTR_TO_S_ON_0   DS 2    ;pointer to data area of song to be played on CH# 0

SAVE_CTRL       DS 1    ;LSN = last control data sent to noise generator
```

Special Effect format

All special effect routines should be written in the following format (SFX = the address of the effect routine, stored in ROM after the effect's header, IX is passed pointing to the song's data area):

```
SFX:    LD (SAVE_x_NNP),HL          ;save address of next note in song
        LD (SAVE_x_SONGNO),A        ;save song's SONGNO
        RET                         ;to LOAD_NEXT_NOTE
SFX47:  LD HL,SAVE_x_SONGNO         ;test for 1st pass through effect
        BIT 7,(HL)
        JR NZ,NOT_PASS_1
          SET 7,(HL)        ;to prevent further passes thru inits
          ...               ;initialize bytes within the data area here
          RET               ;to LOAD_NEXT_NOTE
NOT_PASS_1: ...             ;code for pass 2 or greater starts here,
        .                   ;which algorithmically modifies freq, atn,
        .                   ;or control data within song data area
        .                   ;pointed to by IX
        RET (to PROCESS_DATA_AREA) if effect not over
          ;if here, effect is over, so restore SONGNO and addr nxt note in song
          LD HL,(SAVE_x_NNP)     ;HL := addr next note in song
          LD DE,SAVE_x_SONGNO    ;DE := addr saved song number
          CALL LEAVE_EFFECT      ;to restore them to bytes 0 - 2 in data area
          JP EFXOVER             ;in PROCESS_DATA_AREA to load song's next note
```

FIGURE 10



DEDICATED RAM
ROM
ROM
USER RAM
PTR_TO_LST_OF_SND_ADDRS:
7020H
PTR_TO_S_ON_1:
7022H
PTR_TO_S_ON_2:
7024H
PTR_TO_S_ON_3:
7026H
PTR_TO_S_ON_0:
7028H
SAVE_CTRL:
702AH
LST_OF_SND_ADDRS:
song 01
last song
Note List (note data) for song 01
end of song byte
Note List (note data) for song 02
Note List (note data) for last song
end of song byte
first song data area:
byte 0
byte 9
second song data area:
byte 0
last song data area:
byte 0
byte 9
this byte must be 0
00H

APPENDIX D

DIRECTORY OF COLECOVISION SOFTWARE BULLETINS

BULLETIN NO. 0001
May 25, 1982

TO: DISTRIBUTION

FROM: DAVID HWANG

cc: Eric Bromely
Marshall Caras
Robert Schenck

SUBJECT: COLECOVISION SOFTWARE BULLETIN

The Colecovision Software Bulletin has been set up to assist Colecovision programming users to understand, maintain and develop the system/application software.

More specifically, its purposes are:

(1) Part of the continuing effort to document the operating system (currently OS_7:OS);

(2) Keep users updated regarding any patches and revisions of the operating system;

(3) Function as a user's library for information exchange. Any proven routines or modules which can be used as tools to facilitate software development will be properly documented here with author(s) duly credited.

BULLETIN NO. 0002

May 25, 1982

TO: DISTRIBUTION

FROM: Z. SMITH/D. HWANG

SUBJECT: ERROR IN WRITE_VRAM ROUTINE

cc: Eric Bromley
Marshall Caras
Robert Schenck

WRITE_VRAM has a problem:

- It works as advertised for byte counts less than 100H and for byte counts that are even multiples of 100H. For other values, it subtracts 100H from the actual byte count that is written.

- Cartridge programmers should deal with this problem (and corresponding problems it will cause in any OS routine that writes VRAM, except for WR_SPR_NM_TBL) by always sending numbers of bytes that are less than or even multiples of 100H.

- They should not deal with it by padding their byte counts as this may lead to cartridges that fail when the bug is fixed.

BULLETIN NO. 0003

June 7, 1982

TO: DISTRIBUTION

FROM: Z. SMITH/D. HWANG

cc: Eric Bromley
Marshall Caras
Robert Schenck

SUBJECT: ERROR IN OS SOUND PACKAGE

There is a bug in the OS sound software:

- The data structure PTR_TO_LST_OF_SND_ADDR, which takes up 11 RAM bytes, is not located in OS RAM above 73BAH as it should be, but instead has been placed in the cartridge programmer's RAM at 702OH. Cartridge programmers should avoid using RAM from 702OH thru 702AH when the sound software is in operation.

[BULLETIN 4 MISSING]

BULLETIN NO. 0005

6/18/82

TO: DISTRIBUTION

FROM: Z. Smith/D. HWANG

cc: Eric Bromley
Marshall Caras
Robert Schenck

SUBJECT: BUG IN OS ACTIVATE ROUTINE

There is a bug in the OS Activate routine which surfaces when Activate is called on a Semi-Mobile object in Graphics Mode 1.

In this mode, Activate writes the pattern generators for a Semi-Mobile object to VRAM properly, but miscalculates the number and placement in VRAM of the corresponding color bytes when operating on generators in the upper half of the stable.

This leads to 2 problems:

- The upper half of the color table is not written by Activate.
- The color bytes intended for this half of the table are written elsewhere in VRAM possibly overwriting some other table.

Cartridge programmers should avoid using Activate to write pattern generators to VRAM in Graphics Mode 1 whenever possible. Or, if it is absolutely necessary to use Activate in this way they should count, first of all, on having to write the color table separately, and second, on guarding against the second problem listed above by isolating the calor table.

BULLETIN NO. 0806

SEPTEMBER 17, 1982

TO: DISTRIBUTION

FROM: K. LAGACE/D. HWANG

cc: Eric Bromley
Marshall Caras
Robert Schenck

SUBJECT: RELEASE OF ADDITIONAL OS ENTRY POINTS
OS_SYMBOLS:OS REV. 1

Module Name	Address	Description	Inputs	Outputs	Regs. Destroyed
ADDB16	001B1H	Adds 8 bit signed number in "A" to 16 bit number pointed to by "HL".	- 8 bit # in A - 16 bit # addr in HL	Altered 16 bit # at HL addr.	A,F,B
DECLSN	00190H	Decrements LSN of byte pointed to by "HL" without affecting MSN or "HL".	- 8 bit # addr. in HL	Altered 8 bit # at HL addr. Z flag if 0 C flag if -1 .	A,F
DECMSN	0019BH	Decrements MSN of byte pointed to by "HL" without affecting LSN or "HL".	- 8 bit # addr. in HL	Altered 8 bit # at HL addr. Z flag if 0 C flag if -1	A,F
[illegible]LSN	001A6H	Copies MSN of byte pointed to by "HL" to LSN of that byte.	- 8 bit # addr. in HL	MSN!LSN of # at HL addr. becomes MSN!MSN	A,F,B

FOR SOUND USE ONLY

ATN_SWEEP	0012FH	Creates attenuation sweeps by altering attenuation data stored in song data area.	See Sound Users Manual	See Sound Users Manual	All
FREQ_SWEEP	000FCH	Creates frequency sweeps by altering frequency data stored in song data area.	See Sound Users Manual	See Sound Users Manual	All
EFXOVER	002EEH	See Sound Users	See Sound Users	See Sound Users	All
LEAVE_EFFECT	001D5H	See Sound Users	See Sound Users	See Sound Users	All

[BULLETIN 7 MISSING]

M E M O R A N D U M

NO. 0008
OCTOBER 27, 1982

TO: DISTRIBUTION

cc: Robert Schenck

FROM: MUSIC AND SOUND DEPT./D. HWANG

SUBJECT: MUSIC TABLES

LST_OF_SND_ADDRS has formerly been used in all games to denote the starting address of a list of pointers to song tables and work areas. This label will not be used in the future games. Instead a label with postfix "NOTES" will be used.

For example, in the upcoming games:

DONKEY KONG JR	will use	KONGJRNOTES
OMEGA RACE	will use	OMEGANOTES
GORF	will use	GORFNOTES

M E M O R A N D U M

NO. 0009
OCTOBER 27, 1982

TO: DISTRIBUTION cc: Robert Schenck

FROM: MUSIC AND SOUND DEPT./D. HWANG

SUBJECT: SONGBIRD FILE

Effective immediately all work pertaining to music and sounds will be done in the SONGBIRD file. To play a song, a call to a descriptive label, which is supplied by the music group, will be used. For example:

```
CALL   BELL_SOUND
```

Where BELL_SOUND is a global label in the SONGBIRD file which will contain all that is necesary to play that sound or song. This one call approach is replacing the former procedure such as:

```
LD     B,3      ; THE SONG NUMBER
CALL   PLAY_IT
LD     B,4
CALL   PLAY_IT
```

Within the SONG_BIRD file, song numbers will be EQUAT to descriptive labels instead of using absolute numbers.

[BULLETIN 10 MISSING]

BULLETIN NO. 11
DECEMBER 22, 1982

TO: DISTRIBUTION

FROM: ARD SOFTWARE ENGINEERING David Yang

SUBJECT: RELEASE OF COLECOVISION PROGRAMMERS MANUAL REV. 5

cc: Eric Bromley
Robert Schenck
Marshall Caras
Tom Helmer

The ColecoVision Programmer's Manual Rev. 5 has been released. This manual is written for the applications programmer and is intended as both a day-to-day reference source as well as a training document for programmers new to ColecoVision.

This new edition contains the overview for both hardware and software. Subsequently, detail descriptions are given in the areas of:

- Graphics Generation Software
- Interrupt Handling
- Timing
- Controller Software
- Sound Generation Software
- Boot up Software and Utilities
- Defined Reference Locations

The Rev. 5 manual pertains to the current production OS_7. Fundamental knowledge of the OS is presented in the manual without elaborating on application examples and design approaches. These materials will be documented in the proposed ColecoVision Applications Manual, scheduled to be released in second quarter 1983.

In the Appendix B you will find the graphics documentation (Rev. 1.0) has been updated with addition of materials describing PUT_SPRIT and PUT_COMPLEX.

The Sound documentation also received updates in the form of Notes and Errata attached at the end of Appendix C.

User feedback should be addressed to the Manager of Software Engineering of Coleco ARD. All adopted changes will be brought to your attention via ColecoVision Bulletin announcements.

This manual is confidential and should not be copied. All releases have to be signed out through the ARD Engineering secretary S. Kakowski.

DISTRIBUTION:

C. Baldyga	K. Lagace
R. Dionne	J. Michaels
A. Godfrey	M. Minto
L. Gray	A. Nguyen
C. Hager	L. Olbrych
R. Harris	D. Schulze

CONFIDENTIAL

COLECO INDUSTRIES, INC. 945 ASYLUM AVENUE HARTFORD, CT. 06105 (203) 278-0260
Executive Offices

ColecoVision Software Bulletin

BULLETIN NO. 0012
March 17, 1983

TO: DISTRIBUTION

FROM: ARD SOFTWARE ENGINEERING DKH KAL

RE: CORRECTIONS IN REGARD TO BULLETIN NO. 0004

---

(1) The statement that "Sound Data Areas are off limits to programmers" is not true.

(2) The "Null Song" method wastes CROM space. Writing OFFH to the first byte of the song's sound area IS recommended.

Since the ColecoVision Operating System turns off sounds by placing OFFH into the first byte of the Sound Data Areas anyway and changing the data structures of the Sound Data Areas would entail changing the operating system. It has been proven that the above method is the fastest and most direct way to abort sounds.

The "null song" method may still be used, but each additional song uses at least five bytes of CROM; four for the LST_OF_SND_ADDRS and one for the END code.

DISTRIBUTION:

C. Baldyga
R. Dionne
A. Godfrey
L. Gray
C. Hager
R. Harris
L. Henderson
R. Jepson
D. Jonker

K. Lagace
J. Michaels
M. Minto
A. Nguyen
L. Olbrych
R. Rizzo
D. Schulze
D. Stern
D. Thompson
B. Zawislak

CC: E. Bromley
R. Schenck
C. M. Caras
T. Helmer

CONFIDENTIAL

COLECO
INDUSTRIES, INC.

Executive Offices

945 ASYLUM AVENUE HARTFORD, CT 06105 (203) 278-0280

ColecoVision Software Bulletin

Bulletin No. 0013
April 4, 1983

TO: Distribution

FROM: ARD Software Engineering DKH RFS

RE: Release of Additional ColecoVision OS Entry Points

The following is a list of additional entry points to the ColecoVision OS ROM.

PX_TO_PTRN_POS	EQU	07E8H
PUT_FRAME	EQU	080BH
GET_BKGRND	EQU	0898H
CALC_OFFSET	EQU	08C0H

Attached is a brief description of the routines which correspond to the entry points released.

DISTRIBUTION

C. Baldyga
R. Dionne
L. Gray
C. Hager
R. Harris
L. Henderson
R. Jepson
D. Jonker
K. Lagace
J. Michaels
M. Minto
A. Nguyen
L. Olbrych
R. Rizzo
B. Rochette
D. Schulze
D. Stern
D. Thompson
B. Zawislak

CC: E. Bromley
R. Schenck
D. Hwang
C. M. Caras
T. Helmer
File

CONFIDENTIAL

Here are the graphic subroutines which would be useful to have access to, along with a brief description of what each one does.

****

PX_TO_PTRN_POS (Pixel to pattern plane position)
(entry point xxxxH)

This routine divides the 16 bit signed value in the DE register pair by 8. An 8 bit signed result is returned in register E. Results of less than -127 are returned as -128, results of greater than +126 are returned as +127.

If this routine is passed the X(or Y) pixel coordinate position of a point on the pattern plane, the X(or Y) coordinate in pattern positions will be returned.

INPUT: DE = N (16 bit signed number)

OUTPUT:	N/8 < -128	E = -128
	-128 <= N/8 <= 127	E = N/8
	N/8 > +126	E = +127

REGISTERS AFFECTED:

FLAGS
DE

****

PUTFRAME
(entry point xxxxH)

CONFIDENTIAL

PUTFRAME moves data from cpu RAM to the Pattern Name Table in VRAM. The data is assumed to be an array of Pattern Generator Names which when moved to the Pattern Name Table, will produce a rectangular graphic, or frame, composed of the patterns specified by these Pattern Generator Names. The array must be arranged in row major order.

The dimensions of array are passed to the routine in the BC register pair. These dimensions also define the height and width (in pattern plane positions) of the frame when displayed.

The upper left corner of the frame will appear on the pattern plane at a position determined by Y_PAT_POS and X_PAT_POS which are passed in the DE register pair. Y and X_PAT_POS are row and column coordinates in pattern plane positions as measured from the upper left corner of the pattern plane. Y and X_PAT_POS are interpreted as 8 bit signed values and, therefore, the corner of the frame may placed anywhere within or outside the boundaries of the pattern plane. Therefore, the frame itself may be placed partially off screen in any direction.

The HL register pair must contain the address of the start of the array of pattern names.

INPUT: HL = Address of array in CPU RAM
B = Y dimension of array and Y_EXTENT of frame
C = X dimension of array and X_EXTENT of frame
D = Y_PAT_POS of upper left corner of frame
E = X_PAT_POS of upper left corner of frame

OUTPUT: Modifies VRAM name table

REGESTERS AFFECTED:

All registers used

As an example, if an array exists in CPU memory space which looks like...

```
ARRAY:  DB      0,1,2,3,4,5
```

and the first six pattern generators in VRAM have been initialized with the following patterns...

Pattern Generator #	Graphic
0	A
1	B
2	C
3	D
4	E
5	F

Then the following code sequence...

```
        LD HL,ARRAY
        LD B,2          ;B := Y_EXTENT
        LD C,3          ;C := X_EXTENT
        LD D,2          ;D := Y_PAT_POS
        LD E,-1         ;E := X_PAT_POS
        CALL PUT_FRAME
```

will produce this display...

```
              0 X_PAT_POS ->
Y_PAT_POS    . . . . . . . . . . . .
        V   0. . . . . . . . . . . .
             . . . . . . . . . . . .
             .B.C. . . . . . . . . .
             .E.F. . . . . . . . . .
             . . . . . . . . . . . .
```

(diagram of upper left corner of pattern plane)

Note: Patterns A and D are not seen, since they would be to the left of the left-hand edge of the pattern plane.

****

GET_BKGRND
(entry point xxxxH)

This routine is the inverse of the PUT_FRAME routine described above. GET_BKGRND moves an array of names from the pattern name table in VRAM into CPU RAM. The dimensions of the array and the position of the upper left corner of the frame it defines, are passed to the routine in same manner as in PUT_FRAME. The names are moved to the location in CPU RAM specified by the contents of the HL register pair.

If part of the frame extends beyond the pattern plane, the names that correspond to positions which are not on the pattern plane will not be defined.

INPUTS: HL = Destination address in CPU RAM to which names will be moved
B = Y_EXTENT of frame
C = X_EXTENT of frame
D = Y_PAT_POS of upper left corner of frame
E = X_PAT_POS of upper left corner of frame

OUTPUTS: CPU RAM from HL to HL+(B*C)-1 filled with names from pattern name table

REGISTERS AFFECTED:

All registers used

****

CALC_OFFSET
(entry point xxxxH)

This routine calculates the offset from the start of the pattern name table corresponding to a pattern plane position specified by the coordinates Y_PAT_POS and X_PAT_POS.

The coordinates are passed to, and the result is passed back in the DE register pair.

INPUTS: D = Y_PAT_POS
E = X_PAT_POS

OUTPUTS: DE = Offset from start of pattern name table

REGISTERS AFFECTED:
FLAGS
DE

CONFIDENTIAL

COLECO
INDUSTRIES, INC.

Executive Offices

945 ASYLUM AVENUE HARTFORD, CT 06105 (203) 278-0280

ColecoVison Software Bulletin

Bulletin No. 0014
April 12, 1983

TO: Distribution

FROM: ARD Software Engineering DKH RFJ

RE: OS_SYMBOLS Rev.4

Attached please find a listing of OS_SYMBOLS Rev. 4. This listing holds all ColecoVision OS reserved data entry points released to date.

Attachment

Distribution:

C. Baldyga
R. Dionne
L. Gray
C. Hager
R. Harris
L. Henderson
R. Jepson
D. Jonker
K. Lagace
J. Michaels
J. Milano
M. Minto
A. Nguyen
L. Olbrych
R. Rizzo
B. Rochette
D. Schulze
D. Stern
D. Taylor
D. Thompson
B. Zawislak

CC:

E. Bromley
C. M. Caras
T. A. Helmer
D. Hwang
R. Schenck
File

CONFIDENT

LOCATION OBJECT CODE LINE   SOURCE LINE

```
"Z80"
 NAME "Rev 4 - RFJ"

DESCRIPTION MACRO
        .GOTO ENDESCRIPTION

 Author:          Zac Smith
 Userid:          OS
 Starting date:   13may1982
 Header Rev: 1
 ****************************************************************************
 *                                                                          *
 *                            OS_SYMBOLS                                    *
 *                ColecoVision Operating System                             *
 *                     Software Engineering                                 *
 *               Advanced Research and Development                         *
 *                     Coleco Industries                                    *
 *                                                                          *
 *                    *** Confidential ***                                  *
 *                                                                          *
 ****************************************************************************

     List of access points to the Colecovision Operating system ROM.
 Only these points listed in this file have been approved as absolute
 locations of which the cartridge developer can access the OS rom.
     Additionally, access to any memory locations indirectly, or by
 offset to locations defined herein is denied accept where defined by
 the ColecoVision Programmer's Manual ( current rev 15 ).

 List of OS symbols in alphabetical order with defining and referencing
     modules (if any).

     Rev History (one line note indicating the change)

     Rev.    Date         Name        Change
      4      13apr1359    Rob         Remove Zaxxon related documentation
                                       in preparation for re-release of
                                       this file for general distribution
             11apr1626    Rob         Added PUTFRAME (no underline) to
                                       match label in OS listing.  Kept
                                       PUT_FRAME due to Software Bulletin
                                       released.
             11apr 900    Rob         Updated Header to expand the
                                       description of this file.
                                      GLBed locations added in rev 3
      3      05apr1444    Rob         Added locations
                                       PX_TO_PTRN_POS   PUT_FRAME
                                       GET_BKGRND       CALC_OFFSET
      2      13sept1114p  Rob         Added documentation specific to
                                      Zaxxon Development.
      1      2sept1153p   Ken Lucace  Added 9 SOUND OS equates
      0      13may        Zac Smith   Initial Jump table equates
                                       DATE    : 5/13/82
                                       FOR REV : 5 (OS 5:OS)
ENDESCRIPTION:
        MEND
```

CONFIDENTIAL

CONFIDENTIAL

```
CATION OBJECT CODE LINE     SOURCE LINE

                  ;Symbol                Absolute   Partial Xref of routines used
                  ;Name                  Address    by other OS routines
                  ;-------------------------------------------------------------
                                                    ;Start of defined reference points
(1FF7)             ACTIVATE         EQU 01FF7H
(1F64)             ACTIVATEP        EQU 01F64H
(01B1)             ADD816           EQU 001B1H
(0069)             AMERICA          EQU 00069H
(006A)             ASCII_TABLE      EQU 0006AH
(012F)             ATN_SWEEP        EQU 0012FH
(00C0)             CALC_OFFSET      EQU 000C0H
(0000)             CARTRIDGE        EQU 08000H
(8008)             CONTROLLER_MAP   EQU 08008H      ;CONTROLLE:OS
(0190)             DECLSN           EQU 00190H
(019B)             DECMSN           EQU 0019BH
(1F79)             DECODEP          EQU 01F79H
(73C6)             DEFER_WRITES     EQU 073C6H      ;PUT_OBJEC:OS
(02EE)             EFXOVER          EQU 002EEH
(1F73)             ENLARGE          EQU 01F73H
(1D6C)             ENLRG            EQU 01D6CH
(1F82)             FILL_VRAM        EQU 01F82H      ;GAME_OPT:OS
(1FCA)             FREE_SIGNAL      EQU 01FCAH
(1F9D)             FREE_SIGNALP     EQU 01F9DH
(00FC)             FREQ_SWEEP       EQU 000FCH
(8024)             GAME_NAME        EQU 08024H      ;LOGO:OS
(1F7C)             GAME_OPT         EQU 01F7CH
(0898)             GET_BKGRND       EQU 00898H
(1FBB)             GET_VRAM         EQU 01FBBH      ;PUT_MOBIL:OS  PUT_SPR:OS
(1F8E)             GET_VRAMP        EQU 01F8EH
(1FC1)             INIT_SPR_ORDER   EQU 01FC1H
(1F94)             INIT_SPR_ORDERP  EQU 01F94H
(1FB8)             INIT_TABLE       EQU 01FB8H      ;GAME_OPT:OS  LOGO:OS`
(1F8B)             INIT_TABLEP      EQU 01F8BH
(1FC7)             INIT_TIMER       EQU 01FC7H
(1F9A)             INIT_TIMERP      EQU 01F9AH
(1FE5)             INIT_WRITER      EQU 01FE5H
(1FAF)             INIT_WRITERP     EQU 01FAFH
(801E)             IRQ_INT_VECT     EQU 0801EH
(01D5)             LEAVE_EFFECT     EQU 001D5H
(1F7F)             LOAD_ASCII       EQU 01F7FH      ;GAME_OPT:OS
(8002)             LOCAL_SPR_TBL    EQU 08002H      ;TABLE_MA:OS
(1F85)             MODE_1           EQU 01F85H      ;GAME_OPT:OS
(01A6)             MSNTOLSN         EQU 001A6H
(73C7)             MUX_SPRITES      EQU 073C7H      ;TABLE_MA:OS
(8021)             NMI_INT_VECT     EQU 08021H
(006C)             NUMBER_TABLE     EQU 0006CH
(1FF1)             PLAY_IT          EQU 01FF1H
(1FB5)             PLAY_ITP         EQU 01FB5H
(1F61)             PLAY_SONGS       EQU 01F61H
(1FEB)             POLLER           EQU 01FEBH
(0008)             PUTFRAME         EQU 00008H
(1FFA)             PUTOBJ           EQU 01FFAH      ;PUT_CMPLX:OS
(1F67)             PUTOBJP          EQU 01F67H
(080B)             PUT_FRAME        EQU 0080BH
(  BE)             PUT_VRAM         EQU 01FBEH      ;GAME_OPT:O   LOGO:OS  PUT_MOBIL:OS  PUT_SPR:OS
(  91)             PUT_VRAMP        EQU 01F91H
```

```
LOCATION OB..CT CODE LINE     SOURCE LINE

(07E8)     PX_TO_PTRN_POS    EQU 007E8H
(1FFD)     RAND_GEN          EQU 01FFDH
(73C8)     RAND_NUM          EQU 073C8H
(1FDC)     READ_REGISTER     EQU 01FDCH      ;LOCO:OS
(1FE2)     READ_VRAM         EQU 01FE2H      ;PUT_MOBIL:OS
(1FAC)     READ_VRAMP        EQU 01FACH
(1F6D)     REFLECT_HORIZON   EQU 01F6DH
(1F6A)     REFLECT_VERTICA   EQU 01F6AH
(1FCD)     REQUEST_SIGNAL    EQU 01FCDH
(1FA0)     REQUEST_SIGNALP   EQU 01FA0H
(1F70)     ROTATE_90         EQU 01F70H
(800F)     RST_10H_RAM       EQU 0800FH
(8012)     RST_18H_RAM       EQU 08012H
(8015)     RST_20H_RAM       EQU 08015H
(8018)     RST_28H_RAM       EQU 08018H
(801B)     RST_30H_RAM       EQU 0801BH
(800C)     RST_8H_RAM        EQU 0800CH
(1FEE)     SOUND_INIT        EQU 01FEEH
(1FB2)     SOUND_INITP       EQU 01FB2H
(1FF4)     SOUND_MAN         EQU 01FF4H
(8004)     SPRITE_ORDER      EQU 08004H      ;TABLE_MA:OS
(73B9)     STACK             EQU 073B9H
(800A)     START_GAME        EQU 0800AH      ;LOGO:OS
(1FD0)     TEST_SIGNAL       EQU 01FD0H
(1FA3)     TEST_SIGNALP      EQU 01FA3H
(1FD3)     TIME_MGR          EQU 01FD3H
(1FD6)     TURN_OFF_SOUND    EQU 01FD6H
(1F88)     UPDATE_SPINNER    EQU 01F88H
(73C3)     VDP_MODE_WORD     EQU 073C3H      ;GRAPHICS:OS  VD_DRIVER:OS  TABLE_MA:OS  PUT_MOBIL:OS  ACT2:OS
(73C5)     VDP_STATUS_BYTE   EQU 073C5H
(8006)     WORK_BUFFER       EQU 08006H      ;GRAPHICS:OS  PUT_MOBIL:OS  PUT_SPR:OS  PUTSEMI2:OS  ACT2:OS
(1FE8)     WRITER            EQU 01FE8H
(1FD9)     WRITE_REGISTER    EQU 01FD9H      ;GAME_OPT:OS  LOCO:OS
(1FA6)     WRITE_REGISTERP   EQU 01FA6H
(1FDF)     WRITE_VRAM        EQU 01FDFH      ;GAME_OPT:OS  LOCO:OS  PUT_MOBIL:OS
(1FA9)     WRITE_VRAMP       EQU 01FA9H
(1FC4)     WR_SPR_NM_TBL     EQU 01FC4H
(1F97)     WR_SPR_NM_TBLP    EQU 01F97H
                                             ;End of defined reference points
```

CONFIDENTIAL

LOCATION OBJECT CODE LINE   SOURCE LINE

```
                                ;The following defines each access point
                                ; as Global.
 GLB ACTIVATE
 GLB ACTIVATEP
 GLB ADD816
 GLB AMERICA
 GLB ASCII_TABLE
 GLB ATN_SWEEP
 GLB CALC_OFFSET
 GLB CARTRIDGE
 GLB CONTROLLER_MAP
 GLB DECLSN
 GLB DECMSN
 GLB DECODER
 GLB DEFER_WRITES
 GLB EFXOVER
 GLB ENLARGE
 GLB ENLRG
 GLB FILL_VRAM
 GLB FREE_SIGNAL
 GLB FREE_SIGNALP
 GLB FREQ_SWEEP
 GLB GAME_NAME
 GLB GAME_OPT
 GLB GET_BKGRND
 GLB GET_VRAM
 GLB GET_VRAMP
 GLB INIT_SPR_ORDER
 GLB INIT_SPR_ORDERP
 GLB INIT_TABLE
 GLB INIT_TABLEP
 GLB INIT_TIMER
 GLB INIT_TIMERP
 GLB INIT_WRITER
 GLB INIT_WRITERP
 GLB IRQ_INT_VECT
 GLB LEAVE_EFFECT
 GLB LOAD_ASCII
 GLB LOCAL_SPR_TBL
 GLB MSNTOLSN
 GLB MODE_1
 GLB MUX_SPRITES
 GLB NMI_INT_VECT
 GLB NUMBER_TABLE
 GLB PLAY_IT
 GLB PLAY_ITP
 GLB PLAY_SONGS
 GLB POLLER
 GLB PUTFRAME
 GLB PUTOBJ
 GLB PUTOBJP
 GLB PUT_FRAME
 GLB PUT_VRAM
 GLB PUT_VRAMP
 GLB PX_TO_PTRN_POS
 GLB RAND_GEN
 GLB RND_NUM
```

CONFIDENTIAL

```
LOCATION OBJECT CODE LINE     SOURCE LINE

              GLB READ_REGISTER
              GLB READ_VRAM
              GLB READ_VRAMP
              GLB REFLECT_HORIZON
              GLB REFLECT_VERTICA
              GLB REQUEST_SIGNAL
              GLB REQUEST_SIGNALP
              GLB ROTATE_90
              GLB RST_10H_RAM
              GLB RST_18H_RAM
              GLB RST_20H_RAM
              GLB RST_28H_RAM
              GLB RST_30H_RAM
              GLB RST_8H_RAM
              GLB SOUND_INIT
              GLB SOUND_INITP
              GLB SOUND_MAN
              GLB SPRITE_ORDER
              GLB STACK
              GLB START_GAME
              GLB TEST_SIGNAL
              GLB TEST_SIGNALP
              GLB TIME_MGR
              GLB TURN_OFF_SOUND
              GLB UPDATE_SPINNER
              GLB VDP_MODE_WORD
              GLB VDP_STATUS_BYTE
              GLB WORK_BUFFER
              GLB WRITER
              GLB WRITE_REGISTER
              GLB WRITE_REGISTERP
              GLB WRITE_VRAM
              GLB WRITE_VRAMP
              GLB WR_SPR_NM_TBL
              GLB WR_SPR_NM_TBLP

Errors= 0
```

LOCATION OBJECT CODE LINE   SOURCE LINE

```
^Z80^
 NAME ^Rev 4 - RFJ^

DESCRIPTION MACRO
        .GOTO ENDESCRIPTION

 Author:         Zac Smith
 Userid:         OS
 Starting date:  13may1982
 Header Rev: 1
 # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #
 #                                                                                                  #
 #                                          O S _ S Y M B O L S                                     #
 #                    C o l e c o V i s i o n   O p e r a t i n g   S y s t e m                     #
 #                              S o f t w a r e   E n g i n e e r i n g                             #
 #                  A d v a n c e d   R e s e a r c h   a n d   D e v e l o p m e n t               #
 #                                C o l e c o   I n d u s t r i e s                                 #
 #                                                                                                  #
 #                            * * *     C o n f i d e n t i a l    * * *                            #
 #                                                                                                  #
 # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # # #

     List of access points to the Colecovision Operating system ROM.
 Only these points listed in this file have been approved as absolute
 locations of which the cartridge developer can access the OS rom.
     Additionally, access to any memory locations indirectly, or by
 offset to locations defined herein is denied accept where defined by
 the ColecoVision Programmer's Manual ( current rev 05 ).

 List of OS symbols in alphabetical order with defining and referencing
     modules (if any).

      Rev History (one line note indicating the change)

      Rev.   Date       Name      Change
       4     13apr1359  Rob       Remove Zaxxon related documentation
                                   in preparation for re-release of
                                   this file for general distribution
             11apr1626  Rob       Added PUTFRAME (no underline) to
                                   match label in OS listing.  Kept
                                   PUT_FRAME due to Software Bulletin
                                   released.
             11apr 900  Rob       Updated Header to expand the
                                   description of this file.
                                  GLBed locations added in rev 3
       3     05apr1444  Rob       Added locations
                                   PX_TO_PTRN_POS   PUT_FRAME
                                   GET_BKGRND       CALC_OFFSET
       2     13sept1114p Rob      Added documentation specific to
                                   Zaxxon Development.
       1     2sept1153p Ken Lagace Added 9 SOUND OS equates
       0     13may      Zac Smith  Initial Jump table equates
                                   DATE    : 5/13/82
                                   FOR REV : 5 (OS 5:05)
ENDESCRIPTION:
        MEND
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                  59
                  60 |Symbol              Absolute  Partial Xref of routines used
                  61 |Name                Address   by other OS routines
                  62 |-----------------------------------------------------------
                  63                                ;Start of defined reference points
(1FF7)            64 ACTIVATE          EQU 01FF7H
(1F64)            65 ACTIVATEP         EQU 01F64H
(01B1)            66 ADD816            EQU 001B1H
(0069)            67 AMERICA           EQU 00069H
(006A)            68 ASCII_TABLE       EQU 0006AH
(012F)            69 ATN_SWEEP         EQU 0012FH
(08C0)            70 CALC_OFFSET       EQU 008C0H
(8000)            71 CARTRIDGE         EQU 08000H
(8008)            72 CONTROLLER_MAP    EQU 08008H     ;CONTROLLE:OS
(0190)            73 DECLSN            EQU 00190H
(019B)            74 DECMSN            EQU 0019BH
(1F79)            75 DECODER           EQU 01F79H
(73C6)            76 DEFER_WRITES      EQU 073C6H     ;PUT_OBJEC:OS
(02EE)            77 EFXOVER           EQU 002EEH
(1F73)            78 ENLARGE           EQU 01F73H
(1D6C)            79 ENLRG             EQU 01D6CH
(1F82)            80 FILL_VRAM         EQU 01F82H     ;GAME_OPT:OS
(1FCA)            81 FREE_SIGNAL       EQU 01FCAH
(1F9D)            82 FREE_SIGNALP      EQU 01F9DH
(00FC)            83 FREQ_SWEEP        EQU 000FCH
(8024)            84 GAME_NAME         EQU 08024H     ;LOGO:OS
(1F7C)            85 GAME_OPT          EQU 01F7CH
(0898)            86 GET_BKGRND        EQU 00898H
(1FBB)            87 GET_VRAM          EQU 01FBBH     ;PUT_MOBIL:OS  PUT_SPR:OS
(1F8E)            88 GET_VRAMP         EQU 01F8EH
(1FC1)            89 INIT_SPR_ORDER    EQU 01FC1H
(1F94)            90 INIT_SPR_ORDERP   EQU 01F94H
(1FB8)            91 INIT_TABLE        EQU 01FB8H     ;GAME_OPT:OS  LOGO:OS
(1F8B)            92 INIT_TABLEP       EQU 01F8BH
(1FC7)            93 INIT_TIMER        EQU 01FC7H
(1F9A)            94 INIT_TIMERP       EQU 01F9AH
(1FE5)            95 INIT_WRITER       EQU 01FE5H
(1FAF)            96 INIT_WRITERP      EQU 01FAFH
(801E)            97 IRQ_INT_VECT      EQU 0801EH
(01D5)            98 LEAVE_EFFECT      EQU 001D5H
(1F7F)            99 LOAD_ASCII        EQU 01F7FH     ;GAME_OPT:OS
(8002)           100 LOCAL_SPR_TBL     EQU 08002H     ;TABLE_MA:OS
(1F85)           101 MODE_1            EQU 01F85H     ;GAME_OPT:OS
(01A6)           102 MSNTOLSN          EQU 001A6H
(73C7)           103 MUX_SPRITES       EQU 073C7H     ;TABLE_MA:OS
(8021)           104 NMI_INT_VECT      EQU 08021H
(006C)           105 NUMBER_TABLE      EQU 0006CH
(1FF1)           106 PLAY_IT           EQU 01FF1H
(1FB5)           107 PLAY_ITP          EQU 01FB5H
(1F61)           108 PLAY_SONGS        EQU 01F61H
(1FEB)           109 POLLER            EQU 01FEBH
(080B)           110 PUTFRAME          EQU 0080BH
(1FFA)           111 PUTOBJ            EQU 01FFAH     ;PUT_CMPLX:OS
(1F67)           112 PUTOBJP           EQU 01F67H
(080B)           113 PUT_FRAME         EQU 0080BH
(1FBE)           114 PUT_VRAM          EQU 01FBEH     ;GAME_OPT:OS  LOGO:OS  PUT_MOBIL:OS  PUT_SPR:OS
(1F91)           115 PUT_VRAMP         EQU 01F91H
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

          (07E8)    116 PX_TO_PTRN_POS  EQU 007E8H
          (1FFD)    117 RAND_GEN        EQU 01FFDH
          (73C8)    118 RAND_NUM        EQU 073C8H
          (1FDC)    119 READ_REGISTER   EQU 01FDCH   ;LOGO:OS
          (1FE2)    120 READ_VRAM       EQU 01FE2H   ;PUT_MOBIL:OS
          (1FAC)    121 READ_VRAMP      EQU 01FACH
          (1F6D)    122 REFLECT_HORIZON EQU 01F6DH
          (1F6A)    123 REFLECT_VERTICA EQU 01F6AH
          (1FCD)    124 REQUEST_SIGNAL  EQU 01FCDH
          (1FA0)    125 REQUEST_SIGNALP EQU 01FA0H
          (1F70)    126 ROTATE_90       EQU 01F70H
          (800F)    127 RST_10H_RAM     EQU 0800FH
          (8012)    128 RST_18H_RAM     EQU 08012H
          (8015)    129 RST_20H_RAM     EQU 08015H
          (8018)    130 RST_28H_RAM     EQU 08018H
          (801B)    131 RST_30H_RAM     EQU 0801BH
          (800C)    132 RST_8H_RAM      EQU 0800CH
          (1FEE)    133 SOUND_INIT      EQU 01FEEH
          (1FB2)    134 SOUND_INITP     EQU 01FB2H
          (1FF4)    135 SOUND_MAN       EQU 01FF4H
          (8004)    136 SPRITE_ORDER    EQU 08004H   ;TABLE_MA:OS
          (73B9)    137 STACK           EQU 073B9H
          (800A)    138 START_GAME      EQU 0800AH   ;LOGO_OS
          (1FD0)    139 TEST_SIGNAL     EQU 01FD0H
          (1FA3)    140 TEST_SIGNALP    EQU 01FA3H
          (1FD3)    141 TIME_MGR        EQU 01FD3H
          (1FD6)    142 TURN_OFF_SOUND  EQU 01FD6H
          (1F88)    143 UPDATE_SPINNER  EQU 01F88H
          (73C3)    144 VDP_MODE_WORD   EQU 073C3H   ;GRAPHICS:OS  VD_DRIVER:OS  TABLE_MA:OS  PUT_MOBIL:OS  ACT2:OS
          (73C5)    145 VDP_STATUS_BYTE EQU 073C5H
          (8006)    146 WORK_BUFFER     EQU 08006H   ;GRAPHICS:OS  PUT_MOBIL:OS  PUT_SPR:OS  PUTSEMI2:OS  ACT2:OS
          (1FE8)    147 WRITER          EQU 01FE8H
          (1FD9)    148 WRITE_REGISTER  EQU 01FD9H   ;GAME_OPT:OS  LOGO:OS
          (1FA6)    149 WRITE_REGISTERP EQU 01FA6H
          (1FDF)    150 WRITE_VRAM      EQU 01FDFH   ;GAME_OPT:OS  LOGO:OS  PUT_MOBIL:OS
          (1FA9)    151 WRITE_VRAMP     EQU 01FA9H
          (1FC4)    152 WR_SPR_NM_TBL   EQU 01FC4H
          (1F97)    153 WR_SPR_NM_TBLP  EQU 01F97H
                    154                              ;End of defined reference points
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                              ;The following defines each access point
                              ; as Global.
GLB ACTIVATE
GLB ACTIVATEP
GLB ADD816
GLB AMERICA
GLB ASCII_TABLE
GLB ATN_SWEEP
GLB CALC_OFFSET
GLB CARTRIDGE
GLB CONTROLLER_MAP
GLB DECLSN
GLB DECMSN
GLB DECODER
GLB DEFER_WRITES
GLB EFXOVER
GLB ENLARGE
GLB ENLRG
GLB FILL_VRAM
GLB FREE_SIGNAL
GLB FREE_SIGNALP
GLB FREQ_SWEEP
GLB GAME_NAME
GLB GAME_OPT
GLB GET_BCKGRND
GLB GET_VRAM
GLB GET_VRAMP
GLB INIT_SPR_ORDER
GLB INIT_SPR_ORDERP
GLB INIT_TABLE
GLB INIT_TABLEP
GLB INIT_TIMER
GLB INIT_TIMERP
GLB INIT_WRITER
GLB INIT_WRITERP
GLB IRQ_INT_VECT
GLB LEAVE_EFFECT
GLB LOAD_ASCII
GLB LOCAL_SPR_TBL
GLB MSNTOLSN
GLB MODE_1
GLB MUX_SPRITES
GLB NMI_INT_VECT
GLB NUMBER_TABLE
GLB PLAY_IT
GLB PLAY_ITP
GLB PLAY_SONGS
GLB POLLER
GLB PUTFRAME
GLB PUTOBJ
GLB PUTOBJP
GLB PUT_FRAME
GLB PUT_VRAM
GLB PUT_VRAMP
GLB PX_TO_PTRN_POS
GLB RAND_GEN
GLB RAND_NUM
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

         GLB READ_REGISTER
         GLB READ_VRAM
         GLB READ_VRAMP
         GLB REFLECT_HORIZON
         GLB REFLECT_VERTICA
         GLB REQUEST_SIGNAL
         GLB REQUEST_SIGNALP
         GLB ROTATE_90
         GLB RST_10H_RAM
         GLB RST_18H_RAM
         GLB RST_20H_RAM
         GLB RST_28H_RAM
         GLB RST_30H_RAM
         GLB RST_8H_RAM
         GLB SOUND_INIT
         GLB SOUND_INITP
         GLB SOUND_MAN
         GLB SPRITE_ORDER
         GLB STACK
         GLB START_GAME
         GLB TEST_SIGNAL
         GLB TEST_SIGNALP
         GLB TIME_MGR
         GLB TURN_OFF_SOUND
         GLB UPDATE_SPINNER
         GLB VDP_MODE_WORD
         GLB VDP_STATUS_BYTE
         GLB WORK_BUFFER
         GLB WRITER
         GLB WRITE_REGISTER
         GLB WRITE_REGISTERP
         GLB WRITE_VRAM
         GLB WRITE_VRAMP
         GLB WR_SPR_NM_TBL
         GLB WR_SPR_NM_TBLP

Errors=     0
```

APPENDIX E
JUMP TABLE

PLAY_SONGS	1F61
ACTIVATEP	1F64
PUTOBJP	1F67
REFLECT_VERTICAL	1F6A
REFLECT_HORIZONTAL	1F6D
ROTATE_90	1F70
ENLARGE	1F73
CONTROLLER_SCAN	1F76
DECODER	1F79
GAME_OPT	1F7C
LOAD_ASCII	1F7F
FILL_VRAM	1F82
MODE_1	1F85
UPDATE_SPINNER	1F88
INIT_TABLEP	1F8B
GET_VRAMP	1F8E
PUT_VRAMP	1F91
INIT_SPR_ORDERP	1F94
WR_SPR_NM_TBLP	1F97
INIT_TIMERP	1F9A
FREE_SIGNALP	1F9D
REQUEST_SIGNALP	1FA0
TEST_SIGNALP	1FA3
WRITE_REGISTERP	1FA6
WRITE_VRAMP	1FA9
READ_VRAMP	1FAC
INIT_WRITERP	1FAF
SOUND_INITP	1FB2
PLAY_ITP	1FB5
INIT_TABLE	1FB8
GET_VRAM	1FBB
PUT_VRAM	1FBE
INIT_SPR_ORDER	1FC1
WR_SPR_NM_TBL	1FC4
INIT_TIMER	1FC7
FREE_SIGNAL	1FCA
REQUEST_SIGNAL	1FCD
TEST_SIGNAL	1FD0
TIME_MGR	1FD3
TURN_OFF_SOUND	1FD6
WRITE_REGISTER	1FD9
READ_REGISTER	1FDC
WRITE_VRAM	1FDF
READ_VRAM	1FE2
INIT_WRITER	1FE5
WRITER	1FE8
POLLER	1FEB
SOUND_INIT	1FEE
PLAY_IT	1FF1
SOUND_MAN	1FF4
ACTIVATE	1FF7
PUTOBJ	1FFA
RAND_GEN	1FFD

PLAY_SONGS	1F61
ACTIVATEP	1F64
PUTOBJ	1F67
REFLECT_VERTICAL	1F6A
REFLECT_HORIZONTAL	1F6D
ROTATE_90	1F70
ENLARGE	1F73
CONTROLLER_SCAN	1F76
DECODER	1F79
GAME_OPT	1F7C
LOAD_ASCII	1F7F
FILL_VRAM	1F82
MODE_1	1F85
UPDATE_SPINNER	1F88
INIT_TABLEP	1F8B
GET_VRAMP	1F8E
PUT_VRAMP	1F91
INIT_SPR_ORDERP	1F94
WR_SPR_NM_TBLP	1F97
INIT_TIMERP	1F9A
FREE_SIGNALP	1F9D
REQUEST_SIGNALP	1FA0
TEST_SIGNALP	1FA3
WRITE_REGISTERP	1FA6
WRITE_VRAMP	1FA9
READ_VRAMP	1FAC
INIT_WRITERP	1FAF
SOUND_INITP	1FB2
PLAY_ITP	1FB5
INIT_TABLE	1FB8
GET_VRAM	1FBB
PUT_VRAM	1FBE
INIT_SPR_ORDER	1FC1
WR_SPR_NM_TBL	1FC4
INIT_TIMER	1FC7
FREE_SIGNAL	1FCA
REQUEST_SIGNAL	1FCD
TEST_SIGNAL	1FD0
TIME_MGR	1FD3
TURN_OFF_SOUND	1FD6
WRITE_REGISTER	1FD9
READ_REGISTER	1FDC
WRITE_VRAM	1FDF
READ_VRAM	1FE2
INIT_WRITER	1FE5
WRITER	1FE8
POLLER	1FEB
SOUND_INIT	1FEE
PLAY_IT	1FF1
SOUND_MAN	1FF4
ACTIVATE	1FF7
PUT_OBJ	1FFA
RAND_GEN	1FFD

APPENDIX F

OS SYMBOLS

```
ACTIVATE         EQU 01FF7H    ;    OS:OS
ACTIVATEP        EQU 01F64H    ;    OS:OS
ADD816           EQU 001B1H    ;    OS:OS
AMERICA          EQU 00069H    ;    OS:OS
ASCII_TABLE      EQU 0006AH    ;    OS:OS
ATN_SWEEP        EQU 0012FH    ;    OS:OS
CARTRIDGE        EQU 08000H    ;    OS:OS
CONTROLLER_MAP   EQU 08008H    ;    OS:OS
CTRL_PORT_PTR    EQU 01D43H
DATA_PORT_PTR    EQU 01D47H
DECLSN           EQU 00190H    ;    OS:OS
DECMSN           EQU 0019BH    ;    OS:OS
DECODER          EQU 01F79H    ;    OS:OS
DEFER_WRITES     EQU 073C6H    ;    OS:OS
EFXOVER          EQU 002EEH    ;    OS:OS
ENLARGE          EQU 01F73H    ;    OS:OS
ENLRG            EQU 01D6CH    ;    OS:OS
FILL_VRAM        EQU 01F82H    ;    OS:OS
FREE_SIGNAL      EQU 01FCAH    ;    OS:OS
FREE_SIGNALP     EQU 01F9DH    ;    OS:OS
FREQ_SWEEP       EQU 000FCH    ;    OS:OS
GAME_NAME        EQU 08024H    ;    OS:OS
GAME_OPT         EQU 01F7CH    ;    OS:OS
GET_VRAM         EQU 01FBBH    ;    OS:OS
GET_VRAMP        EQU 01F8EH    ;    OS:OS
INIT_SPR_ORDER   EQU 01FC1H    ;    OS:OS
INIT_SPR_ORDERP  EQU 01F94H    ;    OS:OS
INIT_TABLE       EQU 01FB8H    ;    OS:OS
INIT_TABLEP      EQU 01F8BH    ;    OS:OS
INIT_TIMER       EQU 01FC7H    ;    OS:OS
INIT_TIMERP      EQU 01F9AH    ;    OS:OS
INIT_WRITER      EQU 01FE5H    ;    OS:OS
INIT_WRITERP     EQU 01FAFH    ;    OS:OS
IRQ_INT_VECT     EQU 0801EH    ;    OS:OS
LEAVE_EFFECT     EQU 001D5H    ;    OS:OS
LOAD_ASCII       EQU 01F7FH    ;    OS:OS
LOCAL_SPR_TBL    EQU 08002H    ;    OS:OS
MODE_1           EQU 01F85H    ;    OS:OS
MSNTOLSN         EQU 001A6H    ;    OS:OS
MUX_SPRITES      EQU 073C7H    ;    OS:OS
NMI_INT_VECT     EQU 08021H    ;    OS:OS
NUMBER_TABLE     EQU 0006CH    ;    OS:OS
PLAY_IT          EQU 01FF1H    ;    OS:OS
PLAY_ITP         EQU 01FB5H    ;    OS:OS
PLAY_SONGS       EQU 01F61H    ;    OS:OS
```

```
POLLER            EQU 01FEBH    ;    OS:OS
PUTOBJ            EQU 01FFAH    ;    OS:OS
PUTOBJP           EQU 01F67H    ;    OS:OS
PUT_VRAM          EQU 01FBEH    ;    OS:OS
PUT_VRAMP         EQU 01F91H    ;    OS:OS
RAND_GEN          EQU 01FFDH    ;    OS:OS
RAND_NUM          EQU 073C8H    ;    OS:OS
READ_REGISTER     EQU 01FDCH    ;    OS:OS
READ_VRAM         EQU 01FE2H    ;    OS:OS
READ_VRAMP        EQU 01FACH    ;    OS:OS
REFLECT_HORIZON   EQU 01F6DH    ;    OS:OS
REFLECT_VERTICA   EQU 01F6AH    ;    OS:OS
REQUEST_SIGNAL    EQU 01FCDH    ;    OS:OS
REQUEST_SIGNALP   EQU 01FA0H    ;    OS:OS
ROTATE_90         EQU 01F70H    ;    OS:OS
RST_10H_RAM       EQU 0800FH    ;    OS:OS
RST_18H_RAM       EQU 08012H    ;    OS:OS
RST_20H_RAM       EQU 08015H    ;    OS:OS
RST_28H_RAM       EQU 08018H    ;    OS:OS
RST_30H_RAM       EQU 0801BH    ;    OS:OS
RST_8H_RAM        EQU 0800CH    ;    OS:OS
SOUND_INIT        EQU 01FEEH    ;    OS:OS
SOUND_INITP       EQU 01FB2H    ;    OS:OS
SOUND_MAN         EQU 01FF4H    ;    OS:OS
SPRITE_ORDER      EQU 08004H    ;    OS:OS
STACK             EQU 073B9H    ;    OS:OS
START_GAME        EQU 0800AH    ;    OS:OS
TEST_SIGNAL       EQU 01FD0H    ;    OS:OS
TEST_SIGNALP      EQU 01FA3H    ;    OS:OS
TIME_MGR          EQU 01FD3H    ;    OS:OS
TURN_OFF_SOUND    EQU 01FD6H    ;    OS:OS
UPDATE_SPINNER    EQU 01F88H    ;    OS:OS
VDP_MODE_WORD     EQU 073C3H    ;    OS:OS
VDP_STATUS_BYTE   EQU 073C5H    ;    OS:OS
WORK_BUFFER       EQU 08006H    ;    OS:OS
WRITER            EQU 01FEBH    ;    OS:OS
WRITE_REGISTER    EQU 01FD9H    ;    OS:OS
WRITE_REGISTERP   EQU 01FA6H    ;    OS:OS
WRITE_VRAM        EQU 01FDFH    ;    OS:OS
WRITE_VRAMP       EQU 01FA9H    ;    OS:OS
WR_SPR_NM_TBL     EQU 01FC4H    ;    OS:OS
WR_SPR_NM_TBLP    EQU 01F97H    ;    OS:OS
```

```
GLB ACTIVATE          ;    OS:OS
GLB ACTIVATEP         ;    OS:OS
GLB ADD816            ;    OS:OS
GLB AMERICA           ;    OS:OS
GLB ASCII_TABLE       ;    OS:OS
GLB ATN_SWEEP         ;    OS:OS
GLB CARTRIDGE         ;    OS:OS
GLB CONTROLLER_MAP    ;    OS:OS
GLB CTRL_PORT_PTR
GLB DATA_PORT_PTR
GLB DECLSN            ;    OS:OS
GLB DECMSN            ;    OS:OS
GLB DECODER           ;    OS:OS
GLB DEFER_WRITES      ;    OS:OS
GLB EFXOVER           ;    OS:OS
GLB ENLARGE           ;    OS:OS
GLB ENLRG             ;    OS:OS
GLB FILL_VRAM         ;    OS:OS
GLB FREE_SIGNAL       ;    OS:OS
GLB FREE_SIGNALP      ;    OS:OS
GLB FREQ_SWEEP        ;    OS:OS
GLB GAME_NAME         ;    OS:OS
GLB GAME_OPT          ;    OS:OS
GLB GET_VRAM          ;    OS:OS
GLB GET_VRAMP         ;    OS:OS
GLB INIT_SPR_ORDER    ;    OS:OS
GLB INIT_SPR_ORDERP   ;    OS:OS
GLB INIT_TABLE        ;    OS:OS
GLB INIT_TABLEP       ;    OS:OS
GLB INIT_TIMER        ;    OS:OS
GLB INIT_TIMERP       ;    OS:OS
GLB INIT_WRITER       ;    OS:OS
GLB INIT_WRITERP      ;    OS:OS
GLB IRQ_INT_VECT      ;    OS:OS
GLB LEAVE_EFFECT      ;    OS:OS
GLB LOAD_ASCII        ;    OS:OS
GLB LOCAL_SPR_TBL     ;    OS:OS
GLB MSNTOLSN          ;    OS:OS
GLB MODE_1            ;    OS:OS
GLB MUX_SPRITES       ;    OS:OS
GLB NMI_INT_VECT      ;    OS:OS
GLB NUMBER_TABLE      ;    OS:OS
GLB PLAY_IT           ;    OS:OS
GLB PLAY_ITP          ;    OS:OS
GLB PLAY_SONGS        ;    OS:OS
GLB POLLER            ;    OS:OS
GLB PUTOBJ            ;    OS:OS
GLB PUTOBJP           ;    OS:OS
```

```
GLB PUT_VRAM            ;    OS:OS
GLB PUT_VRAMP           ;    OS:OS
GLB RAND_GEN            ;    OS:OS
GLB RAND_NUM            ;    OS:OS
GLB READ_REGISTER       ;    OS:OS
GLB READ_VRAM           ;    OS:OS
GLB READ_VRAMP          ;    OS:OS
GLB REFLECT_HORIZON     ;    OS:OS
GLB REFLECT_VERTICA     ;    OS:OS
GLB REQUEST_SIGNAL      ;    OS:OS
GLB REQUEST_SIGNALP     ;    OS:OS
GLB ROTATE_90           ;    OS:OS
GLB RST_10H_RAM         ;    OS:OS
GLB RST_18H_RAM         ;    OS:OS
GLB RST_20H_RAM         ;    OS:OS
GLB RST_28H_RAM         ;    OS:OS
GLB RST_30H_RAM         ;    OS:OS
GLB RST_8H_RAM          ;    OS:OS
GLB SOUND_INIT          ;    OS:OS
GLB SOUND_INITP         ;    OS:OS
GLB SOUND_MAN           ;    OS:OS
GLB SPRITE_ORDER        ;    OS:OS
GLB STACK               ;    OS:OS
GLB START_GAME          ;    OS:OS
GLB TEST_SIGNAL         ;    OS:OS
GLB TEST_SIGNALP        ;    OS:OS
GLB TIME_MGR            ;    OS:OS
GLB TURN_OFF_SOUND      ;    OS:OS
GLB UPDATE_SPINNER      ;    OS:OS
GLB VDP_MODE_WORD       ;    OS:OS
GLB VDP_STATUS_BYTE     ;    OS:OS
GLB WORK_BUFFER         ;    OS:OS
GLB WRITER              ;    OS:OS
GLB WRITE_REGISTER      ;    OS:OS
GLB WRITE_REGISTERP     ;    OS:OS
GLB WRITE_VRAM          ;    OS:OS
GLB WRITE_VRAMP         ;    OS:OS
GLB WR_SPR_NM_TBL       ;    OS:OS
GLB WR_SPR_NM_TBLP      ;    OS:OS
```

APPENDIX G

TIMING SOFTWARE DATA STRUCTURE

Table Name:

TIMER_TABLE

Description:

A variable length table located in CRAM which consists of an array of three byte entries. Each entry represents a time request.

Access Method:

Pointed to by TIME_TABLE_BASE.

Format:

Each entry appears as:

7	6	5	4	3	2	1	0
D	R	F	E	L	U	U	U
a							
a							

Where:

D: Done
R: Repeat
F: Free
E: Last_Timer_In_Table
L: Long
U: Unused
a: Counter Byte or pointer to a four byte block for long-repeating timers

Appendix G (continued)

Notes:

Done Bit:	This bit is set when the counter has finished.
Repeat Bit:	This bit is set to allow TIME_MGR to restart the counter at its original value.
Free Bit:	This bit is set to signify that the timer is not in use.
Last_Timer_In_Table Bit:	This bit indicates the last initialized timer in the table.
Long Bit:	This bit defines the timer type. 0 - Short timer 1 - Long timer

APPENDIX H
OS SUBROUTINE LIBRARY

NAME	INPUT REGISTERS										DESCRIPTION	SECTION
ACTIVATE	x	x						x			[illegible] to VRAM	3.3.2.4
ADD816	x	x					x				Adds 8-bit signed (A) to 16-bit signed (HL)→(HL)	9.1
CONT_SCAN											Moves controller port data to RAM	6.4
DECLSN	x	x									Decrements least significant nibble pointed to by (HL)	9.2
DECMSN	x	x									Decrements most significant nibble pointed to by (HL)	[illegible]
DECODER	x	x									Calls CONT_SCAN	6.3
ENLARGE	x	x	x	x	x	x	x				[illegible] the size of the original object	3.2.2.4
FILL_VRAM	x	x	x	x			x				Writes a specific value to VRAM	3.1.5
FREE_SIGNAL							x				Releases a timer [illegible]	5.6
GET_VRAM	x	x	x	x			x			x	Moves VRAM table entry to RAM	3.2.1.2
INIT_SPR_ORDER							x				[illegible] data area [illegible]	3.2.3.1
INIT_TABLE	x	x					x				Initializes the VDP base address for given table	3.2.2.1
INIT_TIMER	x	x	x	x							Initializes timer data areas	5.4
INIT_WRITER	x	x					x				Initializes queue size, head and tail addresses [illegible]	4.1
LOAD_ASCII											Writes ASCII generator set to pattern generator table	9.6
MODE_1											Sets VDP to graphics mode 1 and sprite size [illegible]	3.1.6
MSNTOLSN	x	x									(HL)→byte.MSN to (HL)→byte.LSN	9.4
PLAY_IT					x						Called to start a sound	7.3
PLAY_SONGS											Moves frequency and attenuation data to sound chips	7.5
POLLER											Reads, decodes and debounces all active portions of both controllers	6.2
PUTOBJ					x				x		Changes an object's frame or location on the display	3.3.4
PUT_VRAM	x	x	x	x			x			x	Moves data from RAM to VRAM table	3.2.1.3
RAND_GEN											16-bit random number generator	9.5
READ_REGISTER											Reads and returns the contents of the VDP register	3.1.3
READ_VRAM	x	x	x	x	x	x					Reads from VRAM writes to buffer in VRAM	3.1.1
REFLECT_HORIZONTAL	x	x	x	x	x	x					Reflection of generators about the horizontal axis	3.2.2.2
REFLECT_VERTICAL	x	x	x	x	x	x	x				Reflection of generator about the vertical axis	3.2.2.1
REQUEST_SIGNAL	x	x					x				Sets up a timer for the caller	5.5
ROTATE_90	x	x	x	x	x	x	x				90-degree clockwise rotation of generator	3.2.2.3
SOUND_INIT					x						Initializes various sound data areas	7.2
SOUND_MAN											Called every VDP interrupt, manages sound data base	7.4
TEST_SIGNAL							x				Tests for a time-out of a timer	5.7
TIME_MGR											Maintains all OS software timers	5.3
UPDATE_SPINNER											Services controller spinner switch interrupts	6.5
WRITER											Performs deferred PUTOBJ operations	4.2
WRITE_REGISTER					x	x					Writes a value to a selected VDP register	3.1.4
WRITE_VRAM	x	x	x	x	x	x					Moves data from RAM to VRAM	3.1.2
WR_SPR_NM_TBL							x				Moves local sprite data to VRAM sprite attribute table	3.2.3.2

APPENDIX H
OS SUBROUTINE LIBRARY

NAME	INPUT REGISTERS										DESCRIPTION	SECTION
	H	L	D	E	B	C	A	F	IX	IY		
ACTIVATE	X	X						X			Moves pattern and color generator to VRAM	3.3.3
ADD816	X	X					X				Adds 8-bit signed (A) to 16-bit unsigned (HL) → (HL)	9.1
CONT_SCAN											Saves controller port data to CRAM	6.4
DECLSN	X	X									Decrements least significant nibble pointed to by (HL)	9.2
DECMSN	X	X									Decrements most significant nibble pointed to by (HL)	9.3
DECODER	X	X									Calls COUNT_SCAN	6.3
ENLARGE	X	X	X	X	X	X	X				Double the size of the original object	3.2.2.4
FILL_VRAM	X	X	X	X			X				Writes a value DE times to VRAM	3.1.5
FREE_SIGNAL							X				Releases a timer to the free list based on SIGNAL_NUM	5.8
GET_VRAM	X	X	X	X			X			X	Copies VRAM table entry to CRAM	3.2.1.2
INIT_SPR_ORDER							X				Initializes SPRITE_ORDER data with zeros	3.2.3.1
INIT_TABLE	X	X					X				Initializes the VDP table address for given table	3.2.1.1
INIT_TIMER	X	X	X	X							Initializes timer data areas	5.4
INIT_WRITER	X	X					X				Initializes queue size, head and tail addresses to the beginning of the buffer, and head and tail to zero	4.1
LOAD_ASCII											Writes ASCII generator set to pattern generator table	9.6
MODE_1											Sets VDP to graphics mode 1 and sprite size 0	3.1.6
MSNTOLSN	X	X									(HL) → byte,MSN to (HL) → byte,LSN	9.4
PLAY_IT					X						Called to start a sound	7.3
PLAY_SONGS											Saves frequency and attenuation data to sound chip	7.5
POLLER											Reads, decodes and debounces all active portions of the controllers	6.2
PUTOBJ					X				X		Changes an object's frame or location on the display	3.3.4
PUT_VRAM	X	X	X	X			X			X	Saves data from CRAM to VRAM table	3.2.1.3
RAND_GEN											16-bit pseudo random number generator	9.5
READ_REGISTER											Reads and returns the contents of the VDP register	3.1.3
READ_VRAM	X	X	X	X	X	X					Reads from VRAM writes to buffer in CRAM	3.1.1
REFLECT_HORIZONTAL	X	X	X	X	X	X	X				Reflection of generators arround horizontal axis	3.2.2.2
REFLECT_VERTICAL	X	X	X	X	X	X	X				Reflection of generators arround vertical axis	3.2.2.1
REQUEST_SIGNAL	X	X					X				Sets up a timer for the caller	5.6
ROTATE_90	X	X	X	X	X	X	X				90-degree clockwise rotation of generators	3.2.2.3
SOUND_INIT					X						Initializes various sound data areas	7.2
SOUND_MAN											Called every VDP interrupt, manages sound data areas	7.4
TEST_SIGNAL							X				Tests for a time-out of a timer	5.7
TIME_MGR											Maintains all OS software timers	5.5
UPDATE_SPINNER											Processes controller spinner switch interrupts	6.5
WRITER											Performs deferred PUTOBJ operations	4.2
WRITE_REGISTER					X	X					Writes a value to a selected VDP register	3.1.4
WRITE_VRAM	X	X	X	X	X	X					Saves data from CRAM to VRAM	3.1.2
WR_SPR_NB_TBL							X				Saves local sprite data to VRAM sprite attribute table	3.2.3.2

SECTION VIII

BOOT-UP SOFTWARE

8.1 Power-Up Procedure

```
begin (*run from 0*)
     set up stack_pointer
     (*power up*)
          if cartridge type = test
               execute the code at starting address
               found in location 800AH
          else
               disable sound chip
               init random number generator
               init controller buffer areas
               defer writes = false
               mux sprites = false
               (*display_logo*)
                    fill VRAM with 0's
                    set up VDP to mode 1
                    load ASCII generators
```

```
                      load logo generators
                      load logo names
                      load logo colors
                       enable display
                      if cartridge = game
                          display logo and game name
                          wait 12 seconds
                          disable display
                          execute the code at starting
                          address found in location 800AH
                       else (*cartridge not present*)
                          display log and "insert cartridge"
                          message
                          wait 60 seconds
                          disable display
                          soft halt
                      endif (*cartridge = game*)
                   endif (*cartridge type = test*)
          end (*run from 0*)
```

8.2 Title Screen

During the power-up process, the boot-up software will look for an ASCII string of characters at Cartridge ROM location GAME_NAME for display on the logo screen.

The following information should be in the string:

1. Cartridge title with trademark (T=1EH, M=1FH).
2. Original licensor of the game.
3. The year the cartridge is released.

Example:

```
DEFB   "DONKEY KONG JUNIOR",1EH,1FH
DEFB   /PRESENTS NINTENDO'S/1983"
```

Each string is delimited by a slash (/). The first two strings are limited to 28 characters and the last string is four characters.

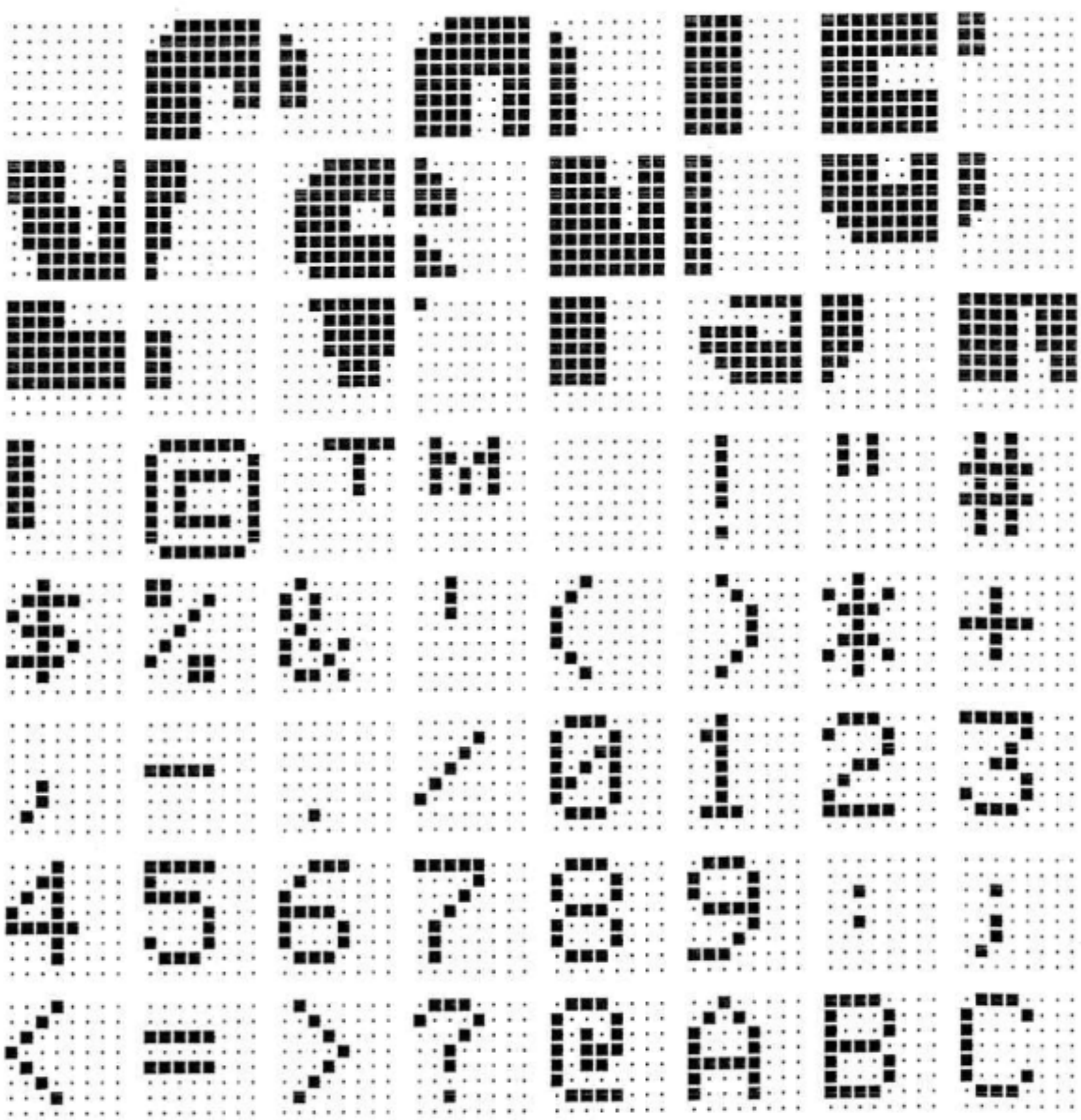
8.3 Cartridge Present Identifier:

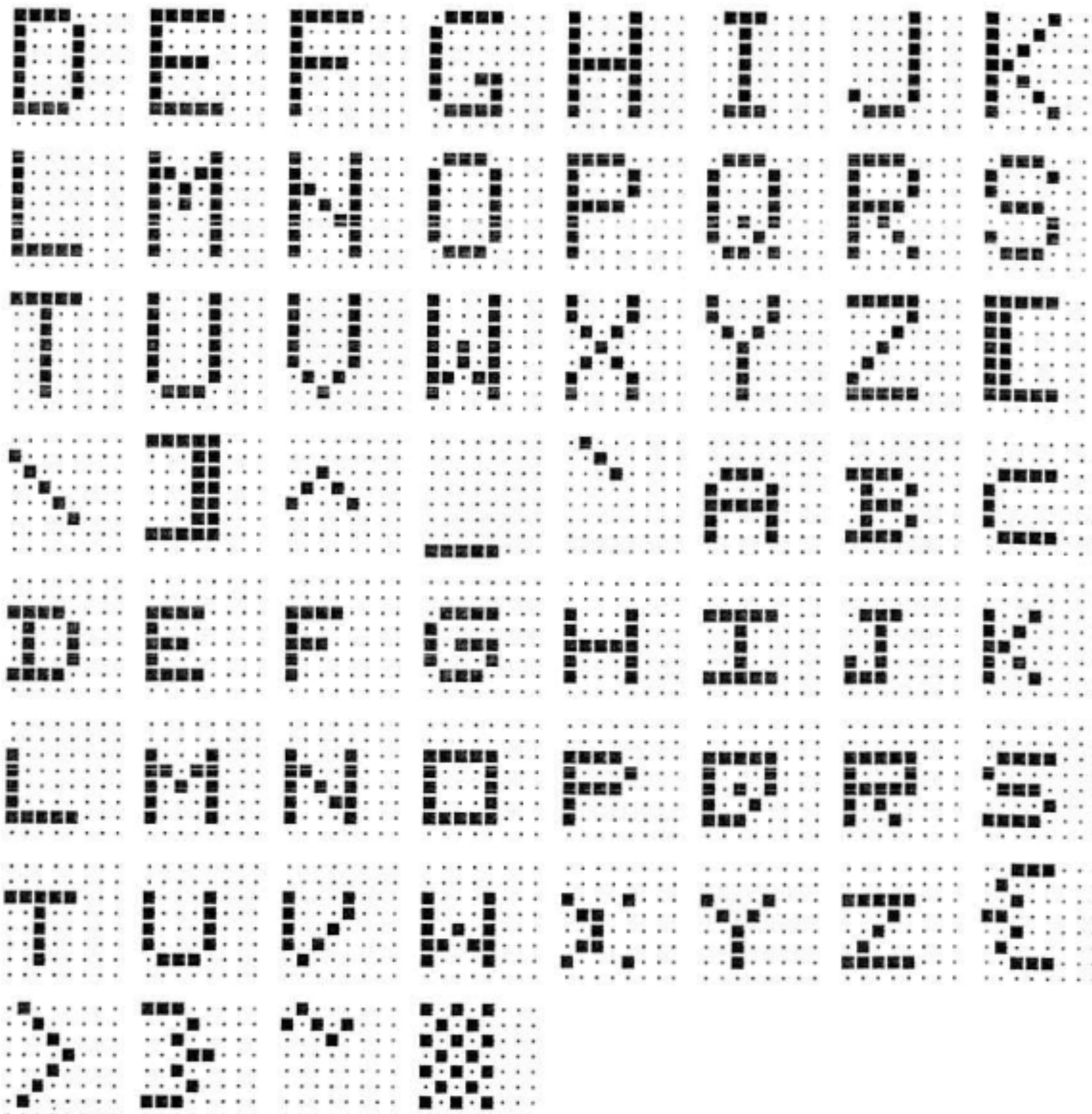
All cartridges must store 0AAH at location 8000H for the OS to recognize them as cartridges that require logo display.

The OS will initialize portions of the hardware, select data areas, display the logo screen and then pass control to the cartridge program.

APPENDIX I: CHARACTER ROM PATTERN GENERATORS.

This section of the manual was missing in the copy I worked from. I have restored its probable appearance by dumping the pattern generators directly from the OS-7 ROM on my R80 ADAM computer. The binary data was transferred to my Tandy 2800HD laptop computer, and a MicroSoft BASIC program used to create EGA pictures of the character patterns, which were then printed out on an HP LaserJet III laser printer.





SECTION V

OS 7 ABSOLUTE LISTINGS

APPENDIX E
JUMP TABLE

PLAY_SONGS	1F61
ACTIVATEP	1F64
PUTOBJP	1F67
REFLECT_VERTICAL	1F6A
REFLECT_HORIZONTAL	1F6D
ROTATE_90	1F70
ENLARGE	1F73
CONTROLLER_SCAN	1F76
DECODER	1F79
GAME_OPT	1F7C
LOAD_ASCII	1F7F
FILL_VRAM	1F82
MODE_1	1F85
UPDATE_SPINNER	1F88
INIT_TABLEP	1F8B
GET_VRAMP	1F8E
PUT_VRAMP	1F91
INIT_SPR_ORDERP	1F94
WR_SPR_NM_TBLP	1F97
INIT_TIMERP	1F9A
FREE_SIGNALP	1F9D
REQUEST_SIGNALP	1FA0
TEST_SIGNALP	1FA3
WRITE_REGISTERP	1FA6
WRITE_VRAMP	1FA9
READ_VRAMP	1FAC
INIT_WRITERP	1FAF
SOUND_INITP	1FB2
PLAY_ITP	1FB5
INIT_TABLE	1FB8
GET_VRAM	1FBB
PUT_VRAM	1FBE
INIT_SPR_ORDER	1FC1
WR_SPR_NM_TBL	1FC4
INIT_TIMER	1FC7
FREE_SIGNAL	1FCA
REQUEST_SIGNAL	1FCD
TEST_SIGNAL	1FD0
TIME_MGR	1FD3
TURN_OFF_SOUND	1FD6
WRITE_REGISTER	1FD9
READ_REGISTER	1FDC
WRITE_VRAM	1FDF
READ_VRAM	1FE2
INIT_WRITER	1FE5
WRITER	1FE8
POLLER	1FEB
SOUND_INIT	1FEE
PLAY_IT	1FF1
SOUND_MAN	1FF4
ACTIVATE	1FF7
PUTOBJ	1FFA
RAND_GEN	1FFD

APPENDIX F

OS SYMBOLS

ACTIVATE	EQU	01FF7H	;	OS:OS
ACTIVATEP	EQU	01F64H	;	OS:OS
ADD816	EQU	001B1H	;	OS:OS
AMERICA	EQU	00069H	;	OS:OS
ASCII_TABLE	EQU	0006AH	;	OS:OS
ATN_SWEEP	EQU	0012FH	;	OS:OS
CARTRIDGE	EQU	08000H	;	OS:OS
CONTROLLER_MAP	EQU	08008H	;	OS:OS
CTRL_PORT_PTR	EQU	01D43H		
DATA_PORT_PTR	EQU	01D47H		
DECLSN	EQU	00190H	;	OS:OS
DECMSN	EQU	0019BH	;	OS:OS
DECODER	EQU	01F79H	;	OS:OS
DEFER_WRITES	EQU	073C6H	;	OS:OS
EFXOVER	EQU	002EEH	;	OS:OS
ENLARGE	EQU	01F73H	;	OS:OS
ENLRG	EQU	01D6CH	;	OS:OS
FILL_VRAM	EQU	01F82H	;	OS:OS
FREE_SIGNAL	EQU	01FCAH	;	OS:OS
FREE_SIGNALP	EQU	01F9DH	;	OS:OS
FREQ_SWEEP	EQU	000FCH	;	OS:OS
GAME_NAME	EQU	08024H	;	OS:OS
GAME_OPT	EQU	01F7CH	;	OS:OS
GET_VRAM	EQU	01FBBH	;	OS:OS
GET_VRAMP	EQU	01F8EH	;	OS:OS
INIT_SPR_ORDER	EQU	01FC1H	;	OS:OS
INIT_SPR_ORDERP	EQU	01F94H	;	OS:OS
INIT_TABLE	EQU	01FB8H	;	OS:OS
INIT_TABLEP	EQU	01F8BH	;	OS:OS
INIT_TIMER	EQU	01FC7H	;	OS:OS
INIT_TIMERP	EQU	01F9AH	;	OS:OS
INIT_WRITER	EQU	01FE5H	;	OS:OS
INIT_WRITERP	EQU	01FAFH	;	OS:OS
IRQ_INT_VECT	EQU	0801EH	;	OS:OS
LEAVE_EFFECT	EQU	001D5H	;	OS:OS
LOAD_ASCII	EQU	01F7FH	;	OS:OS
LOCAL_SPR_TBL	EQU	08002H	;	OS:OS
MODE_1	EQU	01F85H	;	OS:OS
MSNTOLSN	EQU	001A6H	;	OS:OS
MUX_SPRITES	EQU	073C7H	;	OS:OS
NMI_INT_VECT	EQU	08021H	;	OS:OS
NUMBER_TABLE	EQU	0006CH	;	OS:OS
PLAY_IT	EQU	01FF1H	;	OS:OS
PLAY_ITP	EQU	01FB5H	;	OS:OS
PLAY_SONGS	EQU	01F61H	;	OS:OS

```
POLLER              EQU 01FEBH    ;    OS:OS
PUTOBJ              EQU 01FFAH    ;    OS:OS
PUTOBJP             EQU 01F67H    ;    OS:OS
PUT_VRAM            EQU 01FBEH    ;    OS:OS
PUT_VRAMP           EQU 01F91H    ;    OS:OS
RAND_GEN            EQU 01FFDH    ;    OS:OS
RAND_NUM            EQU 073C8H    ;    OS:OS
READ_REGISTER       EQU 01FDCH    ;    OS:OS
READ_VRAM           EQU 01FE2H    ;    OS:OS
READ_VRAMP          EQU 01FACH    ;    OS:OS
REFLECT_HORIZON     EQU 01F6DH    ;    OS:OS
REFLECT_VERTICA     EQU 01F6AH    ;    OS:OS
REQUEST_SIGNAL      EQU 01FCDH    ;    OS:OS
REQUEST_SIGNALP     EQU 01FA0H    ;    OS:OS
ROTATE_90           EQU 01F70H    ;    OS:OS
RST_10H_RAM         EQU 0800FH    ;    OS:OS
RST_18H_RAM         EQU 08012H    ;    OS:OS
RST_20H_RAM         EQU 08015H    ;    OS:OS
RST_28H_RAM         EQU 08018H    ;    OS:OS
RST_30H_RAM         EQU 0801BH    ;    OS:OS
RST_8H_RAM          EQU 0800CH    ;    OS:OS
SOUND_INIT          EQU 01FEEH    ;    OS:OS
SOUND_INITP         EQU 01FB2H    ;    OS:OS
SOUND_MAN           EQU 01FF4H    ;    OS:OS
SPRITE_ORDER        EQU 08004H    ;    OS:OS
STACK               EQU 073B9H    ;    OS:OS
START_GAME          EQU 0800AH    ;    OS:OS
TEST_SIGNAL         EQU 01FD0H    ;    OS:OS
TEST_SIGNALP        EQU 01FA3H    ;    OS:OS
TIME_MGR            EQU 01FD3H    ;    OS:OS
TURN_OFF_SOUND      EQU 01FD6H    ;    OS:OS
UPDATE_SPINNER      EQU 01F88H    ;    OS:OS
VDP_MODE_WORD       EQU 073C3H    ;    OS:OS
VDP_STATUS_BYTE     EQU 073C5H    ;    OS:OS
WORK_BUFFER         EQU 08006H    ;    OS:OS
WRITER              EQU 01FEBH    ;    OS:OS
WRITE_REGISTER      EQU 01FD9H    ;    OS:OS
WRITE_REGISTERP     EQU 01FA6H    ;    OS:OS
WRITE_VRAM          EQU 01FDFH    ;    OS:OS
WRITE_VRAMP         EQU 01FA9H    ;    OS:OS
WR_SPR_NM_TBL       EQU 01FC4H    ;    OS:OS
WR_SPR_NM_TBLP      EQU 01F97H    ;    OS:OS
```

```
GLB ACTIVATE            ;    OS:OS
GLB ACTIVATEP           ;    OS:OS
GLB ADD816              ;    OS:OS
GLB AMERICA             ;    OS:OS
GLB ASCII_TABLE         ;    OS:OS
GLB ATN_SWEEP           ;    OS:OS
GLB CARTRIDGE           ;    OS:OS
GLB CONTROLLER_MAP      ;    OS:OS
GLB CTRL_PORT_PTR
GLB DATA_PORT_PTR
GLB DECLSN              ;    OS:OS
GLB DECMSN              ;    OS:OS
GLB DECODER             ;    OS:OS
GLB DEFER_WRITES        ;    OS:OS
GLB EFXOVER             ;    OS:OS
GLB ENLARGE             ;    OS:OS
GLB ENLRG               ;    OS:OS
GLB FILL_VRAM           ;    OS:OS
GLB FREE_SIGNAL         ;    OS:OS
GLB FREE_SIGNALP        ;    OS:OS
GLB FREQ_SWEEP          ;    OS:OS
GLB GAME_NAME           ;    OS:OS
GLB GAME_OPT            ;    OS:OS
GLB GET_VRAM            ;    OS:OS
GLB GET_VRAMP           ;    OS:OS
GLB INIT_SPR_ORDER      ;    OS:OS
GLB INIT_SPR_ORDERP     ;    OS:OS
GLB INIT_TABLE          ;    OS:OS
GLB INIT_TABLEP         ;    OS:OS
GLB INIT_TIMER          ;    OS:OS
GLB INIT_TIMERP         ;    OS:OS
GLB INIT_WRITER         ;    OS:OS
GLB INIT_WRITERP        ;    OS:OS
GLB IRQ_INT_VECT        ;    OS:OS
GLB LEAVE_EFFECT        ;    OS:OS
GLB LOAD_ASCII          ;    OS:OS
GLB LOCAL_SPR_TBL       ;    OS:OS
GLB MSNTOLSN            ;    OS:OS
GLB MODE_1              ;    OS:OS
GLB MUX_SPRITES         ;    OS:OS
GLB NMI_INT_VECT        ;    OS:OS
GLB NUMBER_TABLE        ;    OS:OS
GLB PLAY_IT             ;    OS:OS
GLB PLAY_ITP            ;    OS:OS
GLB PLAY_SONGS          ;    OS:OS
GLB POLLER              ;    OS:OS
GLB PUTOBJ              ;    OS:OS
GLB PUTOBJP             ;    OS:OS
```

```
GLB PUT_VRAM            ;     OS:OS
GLB PUT_VRAMP           ;     OS:OS
GLB RAND_GEN            ;     OS:OS
GLB RAND_NUM            ;     OS:OS
GLB READ_REGISTER       ;     OS:OS
GLB READ_VRAM           ;     OS:OS
GLB READ_VRAMP          ;     OS:OS
GLB REFLECT_HORIZON     ;     OS:OS
GLB REFLECT_VERTICA     ;     OS:OS
GLB REQUEST_SIGNAL      ;     OS:OS
GLB REQUEST_SIGNALP     ;     OS:OS
GLB ROTATE_90           ;     OS:OS
GLB RST_10H_RAM         ;     OS:OS
GLB RST_18H_RAM         ;     OS:OS
GLB RST_20H_RAM         ;     OS:OS
GLB RST_28H_RAM         ;     OS:OS
GLB RST_30H_RAM         ;     OS:OS
GLB RST_8H_RAM          ;     OS:OS
GLB SOUND_INIT          ;     OS:OS
GLB SOUND_INITP         ;     OS:OS
GLB SOUND_MAN           ;     OS:OS
GLB SPRITE_ORDER        ;     OS:OS
GLB STACK               ;     OS:OS
GLB START_GAME          ;     OS:OS
GLB TEST_SIGNAL         ;     OS:OS
GLB TEST_SIGNALP        ;     OS:OS
GLB TIME_MGR            ;     OS:OS
GLB TURN_OFF_SOUND      ;     OS:OS
GLB UPDATE_SPINNER      ;     OS:OS
GLB VDP_MODE_WORD       ;     OS:OS
GLB VDP_STATUS_BYTE     ;     OS:OS
GLB WORK_BUFFER         ;     OS:OS
GLB WRITER              ;     OS:OS
GLB WRITE_REGISTER      ;     OS:OS
GLB WRITE_REGISTERP     ;     OS:OS
GLB WRITE_VRAM          ;     OS:OS
GLB WRITE_VRAMP         ;     OS:OS
GLB WR_SPR_NM_TBL       ;     OS:OS
GLB WR_SPR_NM_TBLP      ;     OS:OS
```

LOCATION OBJECT CODE LINE   SOURCE LINE

```
"Z80"
 NAME ^OS_7PRIME^

DESCRIPTION MACRO
        .GOTO ENDESCRIPTION

 Author:         Coleco Industries Inc.
                 Advanced Research & Development - Software Engineering
 Userid:         OS
 Starting date:  A long long time ago in a galaxy far far away . . .

 Prom release Date: 24 Nov 1982. for internal use only
 Prom release Rev: 7B

 Prom release Date: December 28, 1982
 Prom release Rev: 7PRIME

 Header Rev: 2

 ***********************************************************************
 *                                                                     *
 *                    ColecoVision Operating System                    *
 *                   Absolute Listing ( Rev 7PRIME )                   *
 *                     (c) Coleco Industries 1982                      *
 *                                                                     *
 *                          *** Confidential ***                       *
 *                                                                     *
 ***********************************************************************


       This listing has the actual addresses of the start of OS routines

       Rev History (one line note indicating the change)

       Rev.   Date                  Change
        4     14feb1983           Filler locations changed to 0FFH to
                                  reflect OS_7PRIME. Prom release date
                                  changed to December 28, 1982 from May
                                  1982.  Name change to OS_7PRIME to
                                  reflect majority of versions in the
                                  field at this date.

        3     24nov1982           Timing change to shorten LOGO delay
                                  Title changes to JMPTABLES and OSSR_EQU
        2     6oct1982            Minor comment modifications
        1     23sept1982          OS_7 as one absolute file
        0     May 1982            OS_7 listing by module

ENDESCRIPTION:
        MEND
 PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      ;        Operating system sound routine EQUATES
                      ;        FILE NAME: OSSR.EQU
                      ;        *** Equates ***
                      ;        Dedicated Cartridge RAM locations
        <7020>        DEDAREA EQU 7020H        ;the start of the RAM area dedicated to sound routines
        <7020>        PTR_TO_LST_OF_SND_ADDRS EQU DEDAREA+0
        <7022>        PTR_TO_S_ON_0 EQU DEDAREA+2
        <7024>        PTR_TO_S_ON_1 EQU DEDAREA+4
        <7026>        PTR_TO_S_ON_2 EQU DEDAREA+6
        <7028>        PTR_TO_S_ON_3 EQU DEDAREA+8
        <702A>        SAVE_CTRL EQU DEDAREA+10
                      ;        Attenuation level codes
        <000F>        OFF      EQU 0FH          ;OFF [NO SOUND]
                      ;        Sound output port
        <00FF>        SOUND_PORT EQU 0FFH    ;data to sound chip thru this port
                      ;        Special byte 0 codes
        <00FF>        INACTIVE EQU 0FFH
        <003E>        SEFFECT EQU 62
        <0000>        ENDSDATA EQU 0
                      ;        Offsets within an SxDATA song data area
        <0000>        CH       EQU 0
        <0000>        SONGNO   EQU 0
        <0001>        NEXTNOTEPTR EQU 1
        <0003>        FREQ     EQU 3
        <0004>        ATN      EQU 4
        <0004>        CTRL     EQU 4
        <0005>        NLEN     EQU 5
        <0006>        FPS      EQU 6
        <0006>        FPSV     EQU 6
        <0007>        FSTEP    EQU 7
        <0008>        ALEN     EQU 8
        <0008>        ASTEP    EQU 8
        <0009>        APS      EQU 9
        <0009>        APSV     EQU 9
                      ;        song end codes
        <0010>        CH0END   EQU 00010000B
        <0050>        CH1END   EQU 01010000B
        <0090>        CH2END   EQU 10010000B
        <0000>        CH3END   EQU 11010000B
        <0018>        CH0REP   EQU 00011000B
        <0058>        CH1REP   EQU 01011000B
        <0098>        CH2REP   EQU 10011000B
        <0008>        CH3REP   EQU 11011000B
                      ;        channel numbers, B7 -B6
        <0000>        CH0      EQU 0
        <0040>        CH1      EQU 01000000B
        <0080>        CH2      EQU 10000000B
        <00C0>        CH3      EQU 11000000B
                      ;        [page]
                       PROG
```

OCATION OBJECT CODE LINE   SOURCE LINE

```

;                       Modified February 14, 1983.  Filler areas were changed
;                       to 0FFH to reflect OS_7PRIME.  Also minor comment changes
;                       were made to clarify 055AAH for test cartridge condition.

************************** EXTERNAL SYMBOLS **************************

* EXTERNAL ROUTINES LINKED INTO OS

                ;EXT REG_WRITE                     ;VIDEO DRIVERS
                ;EXT REG_READ
                ;EXT VRAM_WRITE
                ;EXT VRAM_READ
                ;EXT INIT_QUEUE
                ;EXT WRITER_
                ;EXT REG_WRITEQ                    ;PASCAL CALLS
                ;EXT VRAM_WRITEQ
                ;EXT VRAM_READQ
                ;EXT INIT_QUEUEQ

                ;EXT POLLER_                       ; CONTROLLER ROUTINE
                ;EXT UPDATE_SPINNER_
                ;EXT CONT_SCAN
                ;EXT DECODER_

                ;EXT INIT_SOUND                    ; SOUND ROUTINES
                ;EXT ALL_OFF
                ;EXT JUKE_BOX
                ;EXT SND_MANAGER
                ;EXT PLAY_SONGS_
                ;EXT INIT_SOUNDQ                   ; PASCAL CALLS
                ;EXT JUKE_BOXQ

                ;EXT INIT_TIMER_                   ; TIME  MGMT ROUTINES
                ;EXT FREE_SIGNAL_
                ;EXT REQUEST_SIGNAL_
                ;EXT TEST_SIGNAL_
                ;EXT TIME_MGR_
                ;EXT INIT_TIMERQ                   ; PASCAL CALLS
                ;EXT FREE_SIGNALQ
                ;EXT REQUEST_SIGNALQ
                ;EXT TEST_SIGNALQ

                ;EXT INIT_TABLE_                   ;TABLE MA
                ;EXT GET_VRAM_
                ;EXT PUT_VRAM_
                ;EXT INIT_SPR_ORDER_
                ;EXT WR_SPR_NM_TBL_
                ;EXT INIT_TABLEQ                   ;PASCAL CALLS
                ;EXT GET_VRAMQ
                ;EXT PUT_VRAMQ
                ;EXT INIT_SPR_ORDERQ
                ;EXT WR_SPR_NM_TBLQ

                ;EXT ACTIVATE_                     ; GRAPHICS ROUTINES
                ;EXT PUTOBJ_
                ;EXT RFLCT_VERT
```

OCATION OBJECT CODE LINE SOURCE LINE

```
                ;EXT RFLCT_HOR
                ;EXT ROT_90
                ;EXT ENLRG
                ;EXT PUTOBJQ                    ; PASCAL CALLS
                ;EXT ACTIVATEQ

                ;EXT GAME_OPT_                  ;DISPLAYS THE GAME OPTION SCREEN
                ;EXT LOAD_ASCII_                ;LOADS ASCII CHARACTER GENERATORS
                ;EXT FILL_VRAM_                 ;FILLS DESIGNATED AREA OF VRAM WITH VALUE
                ;EXT MODE_1_                    ;SETS UP A DEFAULT GRAPHICS MODE 1


* "HIDDEN EXTERNALS"

                ;EXT            DISPLAY_LOGO
                ;EXT            CONTROLLER_INIT
                ;EXT            ASCII_TBL
                ;EXT            NUMBER_TBL


******************************** EXPORTS ********************************

* ENTRY POINTS TO OS ROUTINES

                GLB INIT_TABLE                  ;TABLE MA
                GLB GET_VRAM
                GLB PUT_VRAM
                GLB INIT_SPR_ORDER
                GLB WR_SPR_NM_TBL
                GLB INIT_TABLEP                 ;PASCAL CALLS
                GLB GET_VRAMP
                GLB PUT_VRAMP
                GLB INIT_SPR_ORDERP
                GLB WR_SPR_NM_TBLP

                GLB WRITE_REGISTER              ; VIDEO DRIVERS
                GLB READ_REGISTER
                GLB WRITE_VRAM
                GLB READ_VRAM
                GLB INIT_WRITER
                GLB WRITER
                GLB WRITE_REGISTERP             ; PASCAL CALLS
                GLB WRITE_VRAMP
                GLB READ_VRAMP
                GLB INIT_WRITERP

                GLB POLLER                      ; CONTROLLER ROUTINES
                GLB UPDATE_SPINNER
                GLB CONTROLLER_SCAN
                GLB DECODER

                GLB SOUND_INIT                  ; SOUND ROUTINES
                GLB TURN_OFF_SOUND
                GLB PLAY_IT
                GLB SOUND_MAN
                GLB PLAY_SONGS
                GLB SOUND_INITP                 ; PASCAL CALLS
```

LOCATION OBJECT CODE LINE SOURCE LINE

```
        GLB PLAY_ITP

        GLB INIT_TIMER              ; TIME MGMT ROUTINES
        GLB FREE_SIGNAL
        GLB REQUEST_SIGNAL
        GLB TEST_SIGNAL
        GLB TIME_MGR
        GLB INIT_TIMERP             ; PASCAL CALLS
        GLB FREE_SIGNALP
        GLB REQUEST_SIGNALP
        GLB TEST_SIGNALP

        GLB STACK                   ; MISC GLOBALS
        GLB VDP_STATUS_BYTE
        GLB VDP_MODE_WORD
        GLB AMERICA
        GLB MUX_SPRITES
        GLB DEFER_WRITES
        GLB RAND_GEN                ;Can be called from Pascal
                                    ; or assembly language
        GLB PUTOBJ                  ; GRAPHICS ROUTINES
        GLB ACTIVATE
        GLB REFLECT_VERTICAL
        GLB REFLECT_HORIZONTAL
        GLB ROTATE_90
        GLB ENLARGE
        GLB PUTOBJP                 ; PASCAL CALLS
        GLB ACTIVATEP

        GLB GAME_OPT                ;GAME OPTIONS DISPLAY
        GLB LOAD_ASCII              ;LOADS ASCII CHARACTER GENERATORS
        GLB FILL_VRAM               ;FILLS DESIGNATED AREA OF VRAM WITH VALUE
        GLB MODE_1                  ;SETS UP A DEFAULT GRAPHICS MODE 1
        GLB ASCII_TABLE             ;POINTER TO TABLE OF ASCII GENERATORS
        GLB NUMBER_TABLE            ;POINTER TO TABLE OF 0-9 PATTERN GENERATORS

```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                 ********************** CARTRIDGE ROM DATA AREA **********************

                                   GLB            CARTRIDGE
<8000>           CARTRIDGE         EQU            8000H
                 * THIS IS THE MEMORY LOCATION TESTED TO SEE IF A CARTRIDGE IS PLUGGED
                 * IN. IF IT CONTAINS THE PATTERN AA55H THE OS ASSUMES THAT A GAME
                 * CARTRIDGE IS PRESENT. IF IT CONTAINS THE THE PATTERN 55AAH, THE OS
                 * ASSUMES THAT A TEST CARTRIDGE IS PRESENT.

                                   GLB            LOCAL_SPR_TBL
<8002>           LOCAL_SPR_TBL     EQU            8002H
                 * THIS IS A POINTER TO THE CPU RAM COPY OF THE SPRITE NAME TABLE. THE
                 * TABLE COPY IS USED WHENEVER ONE LEVEL OF INDIRECTION IS DESIRED IN
                 * ADDRESSING THE VRAM TABLE. FOR EXAMPLE WHEN USING THE OS SPRITE
                 * MULTIPLEXING SOFTWARE.

                                   GLB            SPRITE_ORDER
<8004>           SPRITE_ORDER      EQU            8004H
                 * THIS IS A POINTER TO THE CPU RAM SPRITE ORDER TABLE. THIS TABLE IS
                 * USED TO ORDER THE LOCAL SPRITE NAME TABLE.

                                   GLB            WORK_BUFFER
<8006>           WORK_BUFFER       EQU            8006H
                 * THIS IS A POINTER TO A FREE BUFFER SPACE IN RAM. THE OBJECT ORIENTED
                 * GRAPHICS ROUTINES USED THIS BUFFER FOR TEMPORARY STORAGE.

                                   GLB            CONTROLLER_MAP
<8008>           CONTROLLER_MAP    EQU            8008H
                 * THIS IS A POINTER TO THE CONTROLLER MEMORY MAP THAT IS MAINTAINED BY
                 * THE HIGH-LEVEL CONTROLLER SCANNING AND DEBOUNCE SOFTWARE.

                                   GLB            START_GAME
<800A>           START_GAME        EQU            800AH
                 * THIS IS A POINTER TO THE START OF THE GAME.

                 ******************** RESTART AND INTERRUPT VECTORS *******************
                 * THESE ARE ADDRESSES IN CARTRIDGE ROM OF VECTORS WHICH MUST BE PLACED
                 * THERE BY THE CARTRIDGE PROGRAMMER. WHEN AN INTERRUPT OR RESTART
                 * OCCURS, THE OS VECTORS IT THROUGH THIS AREA. THE CARTRIDGE PROGRAMMER
                 * SHOULD PLACE A JUMP TO HIS OWN INTERRUPT HANDLER IN THE APPROPRIATE
                 * LOCATION.

                                   GLB            RST_8H_RAM
<800C>           RST_8H_RAM        EQU            800CH
                 * THIS IS THE RESTART 8 SOFT VECTOR.

                                   GLB            RST_10H_RAM
<800F>           RST_10H_RAM       EQU            800FH
                 * THIS IS THE RESTART 10 SOFT VECTOR.

                                   GLB            RST_18H_RAM
<8012>           RST_18H_RAM       EQU            8012H
                 * THIS IS THE RESTART 18 SOFT VECTOR.

                                   GLB            RST_20H_RAM
<8015>           RST_20H_RAM       EQU            8015H
                 * THIS IS THE RESTART 20 SOFT VECTOR.
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE


                            GLB             RST_28H_RAM
<8018>        RST_28H_RAM   EQU             8018H
              * THIS IS THE RESTART 28 SOFT VECTOR.

                            GLB             RST_30H_RAM
<801B>        RST_30H_RAM   EQU             801BH
              * THIS IS THE RESTART 30 SOFT VECTOR.

                            GLB             IRQ_INT_VECT
<801E>        IRQ_INT_VECT  EQU             801EH
              * THIS IS THE MASKABLE INTERRUPT SOFT VECTOR

                            GLB             NMI_INT_VECT
<8021>        NMI_INT_VECT  EQU             8021H
              * THIS IS THE NMI SOFT VECTOR.

                            GLB             GAME_NAME
<8024>        GAME_NAME     EQU             8024H
              * FROM HERE TO START_GAME THERE SHOULD BE A STRING OF ASCII CHARACTERS
              * NAMES THAT HAS THE FOLLOWING FORM:
              *
              *    NAME_OF_THIS_GAME/MAKER_OF_THIS_GAME/COPYWRITE_YEAR.
              *
              *   FOR EXAMPLE:
              *
              *      "DONKEY KONG/NINTENDO/1982"

              * IMPORTANT NOTE *****

              *               **** IT IS THE RESPONSIBILITY OF THE ****
              *               **** CARTRIDGE PROGRAMMER TO PLACE   ****
              *               **** THESE CODES IN CARTRIDGE ROM    ****

```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                 ***********************************************************************
                 *                                                                     *
                 *                     OPERATING SYSTEM ROM CODE                       *
                 *                                                                     *
                 ***********************************************************************

                 ****************************** PAGE ZERO ******************************
                 * PAGE ZERO CONTAINS THE RESTART VECTORS, INTERRUPT VECTORS, AND
                 * THE INTERRUPT VECTORING SOFTWARE, AS WELL AS THE DEFAULT HANDLERS
                 * FOR INTERRUPTS AND RESTARTS.

                 * BOOT-UP ROUTINE

                 * THE BOOT-UP ROUTINE HANDLES POWER ON RESETS AND RESTARTS TO 0. IT
                 * INITIALIZES THE STACK AND JUMPS TO  THE POWER_UP ROUTINE.

                 * BEGIN BOOT-UP
                 BOOT_UP         PROG

                 *   KICK STACK
0000 317389                      LD              SP,STACK

                 *   JUMP TO POWER_UP

0003 C3006E                      JP POWER_UP
0006             END_BOOTUP
                 * END BOOT-UP
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     379
                     380 * RESTART VECTORS
                     381
                     382 * THE FOLLOWING ARE THE 8 PROGRAMMABLE RESTARTS. FOR EACH OF THE
                     383 * RESTART LOCATIONS BELOW THERE IS A VECTOR IN CARTRIDGE ROM.
                     384 * TO USE A RESTART, THE PROGRAMMER MUST PLACE THE ADDRESS OF THE
                     385 * ROUTINE WHICH HE/SHE WISHES TO ACCESS THROUGH THE RESTART AT THE
                     386 * CORRSEPONDING VECTOR. THEREAFTER EVERY TIME THAT RESTART IS
                     387 * EXECUTED, THE CARTRIDGE PROGRAMMER'S ROUTINE WILL BE CALLED.
                     388
0006 FFFF            389                HEX             FF,FF                  ;Filler
0008 C3800C          390 RST_8H         JP RST_8H_RAM
                     391
000B FFFFFFFFFF      392                HEX             FF,FF,FF,FF,FF         ;Filler
0010 C3800F          393 RST_10H        JP RST_10H_RAM
                     394
0013 FFFFFFFFFF      395                HEX             FF,FF,FF,FF,FF         ;Filler
0018 C38012          396 RST_18H        JP RST_18H_RAM
                     397
001B FFFFFFFFFF      398                HEX             FF,FF,FF,FF,FF         ;Filler
0020 C38015          399 RST_20H        JP RST_20H_RAM
                     400
0023 FFFFFFFFFF      401                HEX             FF,FF,FF,FF,FF         ;Filler
0028 C38018          402 RST_28H        JP RST_28H_RAM
                     403
002B FFFFFFFFFF      404                HEX             FF,FF,FF,FF,FF         ;Filler
0030 C3801B          405 RST_30H        JP RST_30H_RAM
                     406
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                * MASKABLE INTERRUPT VECTORING SOFTWARE

                * A MASKABLE INTERRUPT OCCURRING IN THE SYSTEM IS EQUIVALENT TO A
                * RESTART TO 38H. THUS, THE MASKABLE INTERRUPT IS VECTORED IN EXACTLY
                * THE SAME WAY AS THE VARIOUS RESTARTS GIVEN ABOVE. IN ORDER TO USE
                * THE INTERRUPT, THE CARTRIDGE MUST PLACE THE ADDRESS OF HIS/HER
                * INTERRUPT HANDLER IN THE IRQ_INT_VECT LOCATION IN CARTRIDGE ROM.

                * THE CARTRIDGE PROGRAMMER IS RESPONSIBLE FOR SAVING ANY REGISTERS
                * HIS/HER OWN INTERRUPT HANDLERS MAY USE, AND FOR RE-ENABLING
                * INTERRUPTS IF HE/SHE NEEDS THEM TO BE RE-ENABLED.

                * MASKABLE INTERRUPT
0033 FFFFFFFFFF                 HEX     FF,FF,FF,FF,FF      ;Filler
0038            IRQ_INTERRUPT           ;38H
0038 C3801E                     JP      (IRQ_INT_VECT)

                ********************** RANDOM NUMBER GENERATOR **********************

                * (PLACED HERE FOR PURPOSES OF CODE COMPACTION)

                * Random number generator (psuedo) for a 16 bit value
                * This routine 'exclusive or's the 15th and 8th bit
                * together.  It then rotates the entire quantity to the
                * left and inserts the 'exclusive or'ed bit into the rightmost
                * bit.  Upon leaving it stores the random # in a specified
                * memory location.

                * The random number can be accessed from the global location
                * RAND_NUM or the HL pair or the Accumulator.
0038            RAND_GEN_:
0038 2A73C8             LD      HL,[RAND_NUM]
003C CB7C               BIT     7,H
0040 2806               JR      Z,NOT_ON
                                                ;15th bit is on
0042 CB44               BIT     0,H
0044 2806               JR      Z,SET           ;For 10 quantity then set
0046 1807               JR      RESET           ;For 11 reset
0048            NOT_ON:
0048 CB44               BIT     0,H
004A 2803               JR      Z,RESET         ;For 00 reset
                                                ;For 01 fall through to set
004C            SET:
004C 37                 SCF
004D 1801               JR      CARRY_READY
004F            RESET:
004F B7                 OR      A
0050            CARRY_READY:
0050 CB15               RL      L
0052 CB14               RL      H
0054 2273C8             LD      [RAND_NUM],HL
0057 7D                 LD      A,L
0058 C9                 RET
0059            AFTER_RANDOM
```

LOCATION OBJECT CODE LINE SOURCE LINE

```
***********************************************************************

```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                    * THE NMI VECTORING SOFTWARE AND DEFAULT HANDLER

                    * WHEN AN NMI IS RAISED BY THE VDP IN THE COLECOVISION SYSTEM, IT
                    * CAUSES THE CPU TO RESTART TO 66H. THE VECTORING SOFTWARE FOR THE
                    * NMI IS IDENTICAL TO THAT FOR THE MASKABLE INTERRUPT EXCEPT THAT
                    * IT GETS ITS VECTOR FROM NMI_INT_VECT INSTEAD OF IRQ_INT_VECT.

                    * AGAIN THE CARTRIDGE PROGRAMMER IS RESPONSIBLE, IN HIS/HER OWN
                    * INTERRUPT HANDLERS FOR SAVING AND RESTORING THE PROCESSOR STATE
                    * WHEN NECESSARY, AND FOR CLEARING THE VDP CONDITION BY READING THE
                    * VDP STATUS REGISTER.

0059 FFFFFFFFFF                   HEX        FF,FF,FF,FF,FF     ;Filler
005E FFFFFFFFFF                   HEX        FF,FF,FF,FF,FF     ;Filler
0063 FFFFFF                       HEX        FF,FF,FF           ;Filler
                    * NON-MASKABLE INTERRUPT
0066                NMI_INTERRUPT
0066 C38021                       JP         (NMI_INT_VECT)
```

```
LOCATION OBJECT CODE      SOURCE LINE


                     ************************* OS ROM DATA AREA *************************

 0069 3C             AMERICA         DEFB          60
                     * THIS BYTE SHOULD BE USED WHENEVER THE CARTRIDGE PROGRAMMER WANTS TO
                     * SET UP REAL-TIME COUNTERS. IT HAS A VALUE OF 60 FOR COLECOVISIONS
                     * MARKETED IN THE USA AND 50 FOR EUROPEAN UNITS. USE OF THIS BYTE
                     * ENSURES CARTRIDGE COMPATIBILITY AT LEAST WHERE REAL-TIME COUNTING
                     * IS CONCERNED.

 006A 16AB           ASCII_TABLE     DEFW ASCII_TBL
                     * THIS IS THE ADDRESS OF THE ROM PATTERN GENERATORS FOR UPPERCASE
                     * ASCII WHICH ARE CONTAINED WITHIN THE OPERATING SYSTEM.

 006C 1623           NUMBER_TABLE    DEFW NUMBER_TBL
                     * THIS IS THE ADDRESS OF THE ROM PATTERN GENERATORS FOR THE NUMBERS
                     * 0_9 WHICH ARE CONTAINED WITHIN THE OPERATING SYSTEM.

```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      ************************ POWER ON BOOT SOFTWARE *******************
                      ;
                      ;BOOT_UP        SINCE THE VIDEO GAME SYSTEM MAY BE STARTED UP WITH A
                      ;               GAME CARTRIDGE, KEYBOARD MODULE, OR BOTH (OR NOTHING)
                      ;               INSTALLED AT BOOT UP, THE SOFTWARE MUST PERFORM THE
                      ;               FOLLOWING:
                      ;
                      ;                    A.  INITIALIZE THE INTERRUPT VECTORS.
                      ;
                      ;                    B.  INITIALIZE RESTART VECTORS
                      ;
                      ;                    C.  TURN OFF THE SOUND CHIP.
                      ;
                      ;                    D.  DETERMINE IF A CARTRIDGE IS PLUGGED IN.
                      ;                        IF SO, BRANCH TO THE CARTRIDGE PROGRAM
                      ;                        ELSE, WAIT FOR CARTRIDGE.

        <0000>        FALSE           EQU             0
        <0001>        TRUE            EQU             1
                      * VALUES FOR BOOLEAN  FLAGS

                      * BEGIN POWER_UP
        <006E>        POWER_UP        EQU             $

                      *    IF CARTRIDGE = 55AAH THEN  EXIT TO START_GAME (TEST)
0006E 2A8000                          LD              HL,[CARTRIDGE]
0071 7D                               LD              A,L
0072 FE55                             CP              55H
0074 C20081                           JP              NZ,NO_TEST_
0077 7C                               LD              A,H
0078 FEAA                             CP              0AAH
007A C20081                           JP              NZ,NO_TEST_
007D 2A800A                           LD              HL,[START_GAME]
0080 E9                               JP              [HL]

                      *    ELSE
0081                  NO_TEST_

                      *    TURN OFF SOUND CHIP
0081 CD1FD6                           CALL TURN_OFF_SOUND

                      *    INITIALIZE RANDOM NUMBER GENERATOR
0084 210033                           LD              HL,33H
0087 2273C8                           LD              [RAND_NUM],HL

                      *    CLEAR CONTROLLER BUFFER AREAS
008A CD1105                           CALL            CONTROLLER_INIT

                      *    DEFER_WRITES := FALSE
008D 3E00                             LD              A,FALSE
008F 3273C6                           LD              [DEFER_WRITES],A

                      *    MUX_SPRITES := FALSE
0092 3273C7                           LD              [MUX_SPRITES],A
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                          *   EXIT TO DISPLAY LOGO AND TEST FOR CARTRIDGE
   0095 C31319                         JP               DISPLAY_LOGO

                          * END BOOT-UP
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     570 ************************** SYSTEM RAM AREA ****************************
                     571                  DATA
  0000               572                  DEFS          73BAH ;Added to offset to first location
          <73BA>     573 SYSTEM_RAM_AREA EQU            $
                     574 * THIS IS THE RAM AREA DEDICATED TO THE BASIC OS NEEDS. IT INCLUDES THE
                     575 * STACK, VARIOUS STATUS VARIABLES, AND ALL THE VARIABLES USED BY OS
                     576 * ROUTINES.
                     577
          <73B9>     578 STACK            EQU           SYSTEM_RAM_AREA-1
                     579 * THIS IS THE TOP OF THE STACK
                     580
                     581 ;                COMN
                     582 ;                DEFS          9
                     583                  DATA
  73BA               584 PARAM_AREA:
  73BA               585 INIT_SOUND_DATA:
  73BA               586 PRM_AREA:
  73BA               587 INIT_TIME_DATA:
  73BA               588 TEMP1:
  73BA               589             DEFS 1
  73BB               590             DEFS 1
  73BC               591 TEMP2:
  73BC               592             DEFS 1
  73BD               593             DEFS 1
  73BE               594 SIGNAL_NUM:
  73BE               595             DEFS 1
  73BF               596 REPEAT_SIG_CODE:
  73BF               597             DEFS 1
  73C0               598 TIMER_LENGTH:
  73C0               599             DEFS 1
  73C1               600             DEFS 1
  73C2               601 TEST_SIG_NUM:
  73C2               602             DEFS 1
                     603
                     604
                     605 * THIS IS THE COMMON PARAMETER PASSING AREA AND THE HOLE IN THE DATA
                     606 * AREA THAT IS PROVIDED TO MAKE ROOM FOR IT.
                     607
  73C3               608 VDP_MODE_WORD   DEFS          2
                     609 * THE VDP MODE WORD CONTAINS A COPY OF THE DATA IN THE FIRST TWO VDP
                     610 * REGISTERS. BY EXAMINING THIS DATA, THE OS AND CARTRIDGE PROGRAMS
                     611 * CAN MAKE MODE-DEPENDENT DECISIONS ABOUT THE SPRITE SIZE OR VRAM
                     612 * TABLE ARRANGEMENT. THIS WORD IS MAINTAINED BY THE WRITE_REGISTER
                     613 * ROUTINE WHENEVER THE CONTENTS OF REGISTERS 0 OR 1 ARE CHANGED.
                     614
                     615 *   IMPORTANT NOTE *****
                     616
                     617 *                **** IT IS THE RESPONSIBILITY OF THE ****
                     618 *                **** CARTRIDGE PROGRAMMER TO MAKE    ****
                     619 *                **** SURE THAT NON-STANDARD USE OF   ****
                     620 *                **** THE VDP REGISTERS DOES NOT MAKE ****
                     621 *                **** THE DATA IN THIS WORD INVALID   ****
                     622
  73C5               623 VDP_STATUS_BYTE DEFS          1
                     624 * THE DEFAULT HANDLER FOR THE NMI, WHICH MUST READ THE VDP STATUS
                     625 * REGISTER TO CLEAR THE INTERRUPT CONDITION, PLACES ITS CONTENTS
                     626 * HERE. THIS BYTE IS THE MOST ACCURATE REPRESENTATION OF THE ACTUAL
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      627 * VDP STATUS THAT IS AVAILABLE TO THE CARTRIDGE PROGRAMMER PROVIDED
                      628 * THAT THE VDP INTERRUPT IS ENABLED ON-CHIP
                      629
  73C6                630 DEFER_WRITES    DEFS          1
                      631 * DEFER_WRITES IS A BOOLEAN FLAG WHICH IS SET TO FALSE AT POWER UP
                      632 * TIME, SHOULD BE SET TO TRUE ONLY IF THE CARTRIDGE PROGRAMMER WISHES
                      633 * TO DEFER WRITES TO VRAM. IF THIS FLAG  IS TRUE THEN THE WRITER
                      634 * ROUTINE MUST BE CALLED REGULARLY TO PERFORM DEFERRED WRITES.
                      635
  73C7                636 MUX_SPRITES     DEFS          1
                      637 * THIS BOOLEAN FLAG WITH DEFAULT FALSE VALUE SHOULD BE SET TO TRUE IF
                      638 * THE CARTRIDGE PROGRAMMER WISHES ONE LEVEL OF INDIRECTION TO BE
                      639 * INSERTED INTO SPRITE PROCESSING BY HAVING ALL SPRITES WRITTEN TO
                      640 * A LOCAL SPRITE_NAME_TABLE BEFORE BEING WRITTEN TO VRAM. THIS  AIDS
                      641 * SPRITE MULTIPLEXING SOLUTIONS TO THE FIFTH SPRITE PROBLEM.
                      642
                      643             GLB           RAND_NUM
  73C8                644 RAND_NUM        DEFS  2
                      645 * THIS IS THE SHIFT REGISTER USED BY THE RANDOM NUMBER GENERATOR.
                      646 * IT IS INITIALIZED AT POWER-UP.
                      647         GLB PARAM_
                      648         PROG
 0098                 649 PARAM_
 0098 E1E3E50A6F      650         HEX E1,E3,E5,0A,6F,03,0A,03,67,E3,D5,5E,23,56,23,E5
 009D 030A0367E3
 00A2 D55E235623
 00A7 E5
 00A8 7BB2C2B700      651         HEX 7B,B2,C2,B7,00,E1,5E,23,56,23,E5,EB,5E,23,56,03
 00AD E15E235623
 00B2 E5EB5E2356
 00B7 03
 00B8 0A07D2DA00      652         HEX 0A,07,D2,DA,00,03,E1,E3,73,23,72,23,D1,E3,2B,AF
 00BD 03E1E37323
 00C2 7223D1E32B
 00C7 AF
 00C8 BCC2D000BD      653         HEX BC,C2,D0,00,BD,CA,D6,00,E3,E5,EB,C3,A3,00,E1,EB
 00CD CAD600E3E5
 00D2 EBC3A300E1
 00D7 EB
 00D8 E3E9E1E3E5      654         HEX E3,E9,E1,E3,E5,0F,67,0B,0A,6F,E3,03,03,1A,77,23
 00DD 0F670B0A6F
 00E2 E303031A77
 00E7 23
 00E8 13E32BAFBD      655         HEX 13,E3,2B,AF,BD,C2,F4,00,BC,CA,F8,00,E3,C3,E5,00
 00ED C2F400BCCA
 00F2 F800E3C3E5
 00F7 00
 00F8 E1C3C400        656         HEX E1,C3,C4,00
                      657  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     659 ;         .IDENT FREQSWE          ;includes FREQ_SWEEP
                     660 ;***************************
                     661 ;*        FREQ_SWEEP       *
                     662 ;***************************
                     663 ;.COMMENT )
                     664 ;See Users' Manual for description
                     665 ;RETs Z SET: if note over
                     666 ;RETs Z RESET: if sweep in progress or note not over
                     667 ;)
                     668         GLB FREQ_SWEEP
                     669        ;EXT DECLSN,MSNTOLSN,DECMSN,ADD816
                     670        ;INCLUDE OSSR_EQU:OS:0  ;equates
 00FC                671 FREQ_SWEEP
                     672 ;       * if freq not swept, dec NLEN and RET [setting Z flag]
 00FC DD7E07         673        LD A,[IX+FSTEP]          ;check  for no sweep code
 00FF FE00           674        CP 0                     ;SET Z flag if FSTEP = 0
                     675 ;      IF [PSW,IS,ZERO]         ;note not to be swept
 0101 2009           676        JR NZ,L20
 0103 DD7E05         677          LD A,[IX+NLEN]           ;dec NLEN and
 0106 3D             678          DEC A                    ;SET Z flag if NLEN = 0
 0107 C8             679          RET Z                    ;leave if note over with Z flag SET
 0108 DD7705         680            LD [IX+NLEN],A           ;store decremented NLEN
 010B C9             681            RET                      ;RET with Z flag RESET [note not over]
                     682 ;      ENDIF
                     683 ;      * sweep going, so dec FPSV
 010C DDE5           684 L20    PUSH IX                  ;point HL to FPSV
 010E E1             685        POP HL
 010F 1E06           686        LD E,FPSV
 0111 1600           687        LD D,0
 0113 19             688        ADD HL,DE
 0114 CD0190         689        CALL DECLSN              ;dec FPSV
                     690 ;      IF [PSW,IS,ZERO]         ;FPSV has timed out
 0117 2015           691        JR NZ,L21
                     692 ;        * dec NLEN and leave if sweep is over
 0119 CD01A6         693          CALL MSNTOLSN            ;reload FPSV from FPS
 011C 2B             694          DEC HL                   ;point to NLEN [# steps in the sweep]
 011D 7E             695          LD A,[HL]                ;dec NLEN and
 011E 3D             696          DEC A                    ;SET Z flag if NLEN = 0
 011F C8             697          RET Z                    ;leave if sweep over with Z flag SET
                     698 ;          * sweep not over, so add FSTEP to FREQ
 0120 77             699            LD [HL],A                ;store decremented NLEN
 0121 2B             700            DEC HL                   ;point HL
 0122 2B             701            DEC HL                   ;to FREQ
 0123 DD7E07         702            LD A,[IX+FSTEP]          ;A = FSTEP [two's complement step size]
 0126 CD0181         703            CALL ADD816              ;FREQ = FREQ + FSTEP
 0129 23             704            INC HL                   ;point HL to hi FREQ
 012A CB96           705            RES 2,[HL]               ;RESET B2 in hi FREQ in case add caused > 10 bit FREQ
 012C F6FF           706            OR OFFH                  ;RESET Z flag, sweep not over yet
                     707 ;      ENDIF
 012E C9             708 L21    RET
                     709 ;      END ;FREQSWE
                     710  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                    ;         .IDENT ATNSWEE          ;Includes ATN_SWEEP
                    ;****************************
                    ;*        ATN_SWEEP         *
                    ;****************************
                    ;.COMMENT )
                    ;See User's Manual for description
                    ;RETs Z SET: if byte 8 is 0 [means sweep is over, or note was never swept]
                    ;RETs Z RESET: if sweep in progress
                    ;)
                             GLB ATN_SWEEP
                            ;EXT DECLSN,DECMSN,MSNTOLSN
                            ;INCLUDE OSSR_EQU:OS:0  ;equates
012F                ATN_SWEEP
                    ;        * RET with Z SET if byte 8 = 00
012F DD7E08                  LD A,[IX+8]          ;check byte 8 for no sweep code
0132 FE00                    CP 0                 ;Z is set if byte 8 = 0
0134 C8                      RET Z                ;leave if Z set, sweep not going
                    ;          * sweep going, so dec APSV
0135 DDE5                      PUSH IX
0137 E1                        POP HL               ;point HL to APSV
0138 1600                      LD D,0
013A 1E09                      LD E,APSV
013C 19                        ADD HL,DE
013D CD0190                    CALL DECLSN
                    ;          IF [PSW,IS,ZERO]       ;dec APSV [LSN of byte 9]
0140 2021                      JR NZ,L22              ;APSV has timed out
                    ;            * dec ALEN to see if sweep over
0142 CD01A6                      CALL MSNTOLSN          ;reload APSV from APS
0145 2B                          DEC HL                 ;point to ALEN [# of steps in the sweep]
0146 CD0190                      CALL DECLSN            ;dec ALEN [LSN byte 8]
                    ;            IF [PSW,IS,NZERO]      ;sweep not over yet
0149 2816                        JR Z,L23
                    ;              * add ASTEP to ATN
014B 7E                            LD A,[HL]              ;MSN A = ASTEP
014C E6F0                          AND 0F0H               ;mask LSN
014E 5F                            LD E,A                 ;E = ASTEP | 0
014F 2B                            DEC HL                 ;point HL to ATN
0150 2B                            DEC HL
0151 2B                            DEC HL
0152 2B                            DEC HL
0153 7E                            LD A,[HL]              ;MSN A = ATN
0154 E6F0                          AND 0F0H               ;A = ATN | 0
0156 83                            ADD A,E                ;MSN A = [ASTEP + ATN] | 0
0157 5F                            LD E,A                 ;E = [ASTEP + ATN] | 0
0158 7E                            LD A,[HL]              ;A = ATN | freq or CTRL
0159 E60F                          AND 0FH                ;mask old ATN  A = 0 | freq or CTRL
015B B3                            OR E                   ;OR in new ATN
015C 77                            LD [HL],A              ;store updated value back into song data area
015D F6FF                          OR 0FFH                ;RESET Z flag, sweep not over yet
015F 1802                          JR L22
                    ;            ELSE                   ;Z flag is SET: sweep over
0161 3600           L23            LD [HL],0              ;set byte 8 to 0 to indicate end sweep
                    ;            ENDIF
                    ;          ENDIF
0163 C9             L22      RET
                    ;        END ;ATNSWEE
                     PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      ;        .IDENT UTIL           ;Includes UPATNCTRL,UPFREQ,
                                                     ;DECLSN,DECMSN,MSNTOLSN,ADD816,PT_IX_TO_SxDATA,
                                                     ;LEAVE_EFFECT,AREA_SONG_IS
                      ;***************************
                      ;*       UPATNCTRL         *
                      ;***************************
                      ;.COMMENT )
                      ;Perform single byte update of the snd chip noise control register or any
                      ;attenuation register. IX is passed pointing to byte 0 of a song data area, MSN
                      ;register C = formatted channel attenuation code.
                      ;)
                               GLB UPATNCTRL
                              ;INCLUDE OSSR_EQU:OS:0  ;equates
0164                  UPATNCTRL
0164 DD7E04                    LD A,[IX+4]           ;MSN A = ATN, LSN may be CTRL data
0167 CB61                      BIT 4,C               ;test for ATN
                      ;        IF [PSW,IS,NZERO]     ;ATN is to be sent, move it to the LSN
0169 2804                      JR Z,L24
016B 0F                          RRCA                  ;swap nibbles
016C 0F                          RRCA
016D 0F                          RRCA
016E 0F                          RRCA                  ;LSN A = ATN
                      ;        ENDIF
016F E60F             L24      AND 0FH               ;mask MSN
0171 B1                        OR C                  ;A = formatted register# | ATN or CTRL
0172 D3FF                      OUT [SOUND_PORT],A    ;output ATN or CTRL data
0174 C9                        RET
                      ;***************************
                      ;*       UPFREQ            *
                      ;***************************
                      ;.COMMENT )
                      ;Perform double byte update of a sound chip frequency register.  IX is passed
                      ;pointing to byte0 of a song data area, MSN register D = formatted channel
                      ;frequency code.
                      ;)
                               GLB UPFREQ
0175                  UPFREQ
0175 DD7E03                    LD A,[IX+FREQ]        ;A = F2 F3 F4 F5 F6 F7 F8 F9
0178 E60F                      AND 0FH               ;A =  0  0  0  0 F6 F7 F8 F9
017A B2                        OR D                  ;A =  formatted reg# | F6 F7 F8 F9
017B D3FF                      OUT [SOUND_PORT],A    ;output first freq byte
017D DD7E03                    LD A,[IX+FREQ]        ;A = F2 F3 F4 F5 F6 F7 F8 F9 again
0180 E6F0                      AND 0F0H              ;A = F2 F3 F4 F5  0  0  0  0
0182 57                        LD D,A                ;save in D
0183 DD7E04                    LD A,[IX+FREQ+1]      ;LSN A = 0  0 F0 F1
0186 E60F                      AND 0FH               ;A =  0  0  0  0  0  0 F0 F1
0188 B2                        OR D                  ;A = F2 F3 F4 F5  0  0 F0 F1
0189 0F                        RRCA                  ;swap nibbles
018A 0F                        RRCA
018B 0F                        RRCA
018C 0F                        RRCA                  ;A =  0  0 F0 F1 F2 F3 F4 F5
018D D3FF                      OUT [SOUND_PORT],A    ;output 2nd [most significant] freq byte
018F C9                        RET
                      ;***************************
                      ;*       DECLSN            *
                      ;***************************
                      ;.COMMENT )
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                  ;Without affecting the MSN, decrement the LSN of the byte pointed to by HL.  HL
                  ;remains the same.
                  ;RET with Z flag set if dec LSN results in 0, reset otherwise.
                  ;RET with C flag set if dec LSN results in -1, reset otherwise.
                  ;)
                          GLB DECLSN
0190 3E00         DECLSN  LD A,0
0192 ED67                 RRD               ;A = 0 | LSN [HL]
0194 D601                 SUB 1             ;Z flag set if dec to 0, C flag if dec to -1
0196 F5                   PUSH AF           ;save Z and C flags
0197 ED6F                 RLD               ;[HL] = old MSN | new LSN
0199 F1                   POP AF            ;restore Z and C flags, A = 0 | new LSN
019A C9                   RET
                  ;***************************
                  ;*        DECMSN          *
                  ;***************************
                  ;.COMMENT )
                  ;Without affecting the LSN, decrement the MSN of the byte pointed to by HL.  HL
                  ;remains the same.
                  ;RET with Z flag set if dec MSN results in 0, reset otherwise.
                  ;RET with C flag set if dec MSN results in -1, reset otherwise.
                  ;)
                          GLB DECMSN
019B 3E00         DECMSN  LD A,0
019D ED6F                 RLD               ;A = 0 | MSN [HL]
019F D601                 SUB 1             ;Z flag set if dec to 0, C flag set if dec -1
01A1 F5                   PUSH AF           ;save Z and C flags
01A2 ED67                 RRD               ;[HL] = new MSN | old LSN
01A4 F1                   POP AF            ;restore Z and C flags, A = 0 | new MSN
01A5 C9                   RET
                  ;***************************
                  ;*        MSNTOLSN        *
                  ;***************************
                  ;.COMMENT )
                  ;Copy MSN of the byte pointed to by HL to the LSN of that byte.  HL remains
                  ;the same.
                  ;)
                          GLB MSNTOLSN
01A6              MSNTOLSN
01A6 7E                   LD A,[HL]         ;A = MSN | LSN to be changed
01A7 E6F0                 AND 0F0H           ;A = MSN | 0
01A9 47                   LD B,A             ;save in B
01AA 0F                   RRCA               ;swap nibbles
01AB 0F                   RRCA
01AC 0F                   RRCA
01AD 0F                   RRCA              ;A = 0 | MSN
01AE B0                   OR B              ;A = MSN | MSN
01AF 77                   LD [HL],A         ;[HL] = MSN | MSN
01B0 C9                   RET
                  ;***************************
                  ;*        ADD816          *
                  ;***************************
                  ;.COMMENT )
                  ;Adds 8 bit two's complement signed value passed in A to the 16 bit location
                  ;pointed to by HL.
                  ;)
                          GLB ADD816
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

0181 0600         ADD816  LD B,0              ;set B for positive value in A
0183 CB7F                 BIT 7,A             ;if A is positive
0185 2802                 JR Z,POS            ;skip
0187 06FF                   LD B,0FFH           ;A is neg: extend sign bit thru B
0189 86           POS     ADD A,[HL]          ;do 8 bit add [and set Carry]
018A 77                   LD [HL],A           ;store result into LSB 16 bit number
018B 23                   INC HL              ;put MSB
018C 7E                   LD A,[HL]           ;into A
018D 88                   ADC A,B             ;A = MSB + Carry + B [B is 0 or FF]
018E 77                   LD [HL],A           ;store result into MSB
018F 2B                   DEC HL              ;re-point HL to LSB 16 bit number
01C0 C9                   RET
                  ;***************************
                  ;*     PT_IX_TO_SxDATA     *
                  ;***************************
                  ;.COMMENT )
                  ;SONGNO passed in B.
                  ;Point IX to byte 0 in SONGNO's song data area.
                  ;RET with both DE and IX pointing to SxDATA,
                  ;HL pointing to MSB SxDATA entry in LST_OF_SND_ADDRS.
                  ;)
                          GLB PT_IX_TO_SxDATA
01C1              PT_IX_TO_SxDATA
                  ;         * IX & DE := addr of byte 0 in SONGNO's song  data area,
                  ;           HL pointing to MSB SxDATA entry in LST_OF_SND_ADDRS.
01C1 2A7020               LD HL,[PTR_TO_LST_OF_SND_ADDRS] ;point HL to start LST_OF_SND_ADDRS
01C4 2B                   DEC HL                          ;init HL for addition
01C5 2B                   DEC HL
01C6 48                   LD C,B              ;form 4 * SONGNO in C
01C7 0600                 LD B,0
01C9 CB01                 RLC C
01CB CB01                 RLC C
01CD 09                   ADD HL,BC           ;HL pts to SxDATA's entry in LST_OF_SND_ADDRS
01CE 5E                   LD E,[HL]           ;move addr SxDATA to IX thru DE
01CF 23                   INC HL
01D0 56                   LD D,[HL]
01D1 D5                   PUSH DE
01D2 DDE1                 POP IX
01D4 C9                   RET
                  ;
                  ;                 LEAVE EFFECT
                  ;4/19/82
                  ;
                  ;***************************
                  ;*      LEAVE_EFFECT       *
                  ;***************************
                  ;.COMMENT )
                  ;LEAVE_EFFECT, called by a special sound effect routine when it's finished,
                  ;restores the SONGNO of the song to which the effect note belongs to B5 - B0 of
                  ;byte 0 in the effect's data area, and loads bytes 1 and 2 with the address of
                  ;the next note in the song.  The address of the 1 byte SONGNO (saved by the
                  ;effect when first called) is passed in DE.  The 2 byte address of the next note
                  ;in the song, also saved by the effect, is passed in HL.  IX is assumed to be
                  ;pointing to byte 0 of the data area to which the song number is to be restored.
                  ;Bits 7 and 6 of the saved SONGNO are ignored, and therefore may be used by the
                  ;effect to store flag information during the course of the note.
                  ;)
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                         GLB LEAVE_EFFECT
01D5                LEAVE_EFFECT
01D5 DD7501              LD [IX+1],L            ;LSB NEXT_NOTE_PTR := LSB addr next note in song
01D8 DD7402              LD [IX+2],H            ;MSB NEXT_NOTE_PTR := MSB addr next note in song
01DB 1A                  LD A,[DE]              ;A := x x SONGNO (i.e., the saved, original SONGNO)
01DC E63F                AND 03FH               ;A := 0 0 SONGNO
01DE 47                  LD B,A                 ;B := 0 0 SONGNO (B now = original SONGNO)
01DF DD7E00              LD A,[IX+0]            ;A := CH# | 62 (all effect notes have SONGNO = 62)
01E2 E6C0                AND 0C0H               ;A := CH# 0 0 0 0 0 0
01E4 B0                  OR B                   ;A := CH# | SONGNO
01E5 DD7700              LD [IX+0],A            ;restore the song number
01E8 C9                  RET

                    ;***************************
                    ;*       AREA_SONG_IS      *
                    ;***************************
                    ;.COMMENT )
                    ;The address of byte 0 of a song data area is passed in IX.  The song number of
                    ;the song using that area is returned in A [0FFH if inactive].  If a special
                    ;effect was using that area, 62 is returned in A and HL is returned with the
                    ;address of the special sound effect routine.
                    ;)
                         GLB AREA_SONG_IS
01E9                AREA_SONG_IS
01E9 DD7E00              LD A,[IX+0]            ;A := CH# | SONGNO or 62, or A := FF
01EC FEFF                CP 0FFH
01EE C8                  RET Z                  ;leave with A = FF if area inactive
01EF E63F                  AND 00111111B          ;mask CH#
01F1 FE3E                  CP 62
01F3 C0                    RET NZ                 ;leave with A = SONGNO if not a special effect [62]
                    ;          special effect, so set HL to addr effect, stored in bytes 1&2
01F4 DDE5                    PUSH IX                ;point HL to byte 1
01F6 E1                      POP HL
01F7 23                      INC HL
01F8 5E                      LD E,[HL]              ;save LSB effect addr in E
01F9 23                      INC HL                 ;HL to byte 2
01FA 56                      LD D,[HL]              ;save MSB effect addr in D
01FB EB                      EX DE,HL               ;HL := addr special effect
01FC C9                      RET
                    ;        END ;UTIL
                     PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                   ;         .IDENT INITSOU          ;includes INIT_SOUND,ALL_OFF
                   ;***************************
                   ;*          INIT_SOUND       *
                   ;***************************
                   ;.COMMENT )
                   ;See Users' Manual for description; includes ENTRY POINT ALL_OFF
                   ;addr LST_OF_SND_ADDRS passed in HL
                   ;n = # of song data areas to init, passed in B
                   ;)
                            GLB INIT_SOUND,ALL_OFF,DUMAREA
                           ;INCLUDE OSSR_EQU:OS  ;equates
                   ;        *** Sound chip register code EQUATES
                   ;        Tone generator frequency and attenuation formatted register codes
          <0080>   SR1FRQ  EQU 10000000B          ;BIT7 = 1, BIT6-4 = TONE GEN 1 FREQ CODE
          <0090>   SR1ATN  EQU 10010000B          ;BIT7 = 1, BIT6-4 = TONE GEN 1 ATTN CODE
          <00A0>   SR2FRQ  EQU 10100000B          ;BIT7 = 1, BIT6-4 = TONE GEN 2 FREQ CODE
          <00B0>   SR2ATN  EQU 10110000B          ;BIT7 = 1, BIT6-4 = TONE GEN 2 ATTN CODE
          <00C0>   SR3FRQ  EQU 11000000B          ;BIT7 = 1, BIT6-4 = TONE GEN 3 FREQ CODE
          <00D0>   SR3ATN  EQU 11010000B          ;BIT7 = 1, BIT6-4 = TONE GEN 1 ATTN CODE
                   ;        Noise generator control and attenuation formatted register codes
          <00E0>   SRNCTL  EQU 11100000B          ;BIT7 = 1, BIT6-4 = NOISE GEN CONTROL CODE
          <00F0>   SRNATN  EQU 11110000B          ;BIT7 = 1, BIT6-4 = NOISE GEN ATTN CODE

                   * PROCEDURE INIT_SOUNDQ (AREA_COUNT:BYTE;LIST_OF_ADDR:INTEGER)

                   * THIS IS THE PASCAL ENTRY POINT TO INIT_SOUND

                             ;EXT  PARAM
                              GLB  INIT_SOUNDQ
01FD 00020001      INIT_SOUND_PAR:      DEFW  2,1,2
0201 0002

                   ;          COMN
                   ;INIT_SOUND_DATA:     DEFS  3     ;Moved to OS

                              PROG
0203               INIT_SOUNDQ:
0203 0101FD                   LD   BC,INIT_SOUND_PAR
0206 11738A                   LD   DE,INIT_SOUND_DATA
0209 CD0098                   CALL PARAM
020C 3A738A                   LD   A,[INIT_SOUND_DATA]
020F 47                       LD   B,A
0210 2A738B                   LD   HL,[INIT_SOUND_DATA+1]

0213               INIT_SOUND
                   ;         * initialize PTR_TO_LST_OF_SND_ADDRS with value passed in HL
0213 227020                  LD [PTR_TO_LST_OF_SND_ADDRS],HL
                   ;         * store inactive code at byte 0 of each of the n data areas [B = n]
0216 23                      INC HL                  ;pt HL to song 1 data area entry in LST_OF_SND_ADDRS
0217 23                      INC HL
0218 5E                      LD E,[HL]               ;pt DE to byte 0 in first song data area
0219 23                      INC HL
021A 56                      LD D,[HL]
021B EB                      EX DE,HL                ;pt HL to byte 0 in first song data area
021C 1E0A                    LD E,10                 ;set DE for 10 byte increment
021E 1600                    LD D,0
0220 36FF          B1:       LD [HL],0FFH            ;deactivate area
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

 0222 19                          ADD HL,DE              ;pt HL to byte 0 next area (10 bytes away)
 0223 10FB                        DJNZ B1                ;do this for the n (passed in B) data areas
                             ;    * store end of data area code (0) at first byte after last song data area
 0225 3600                        LD [HL],0              ;store end of data area code in byte 0 data area n + 1
                             ;    * set the 4 channel data area pointers to a dummy, inactive data area
 0227 21024C                      LD HL,DUMAREA          ;point HL to inactive byte below [after the RET]
 022A 227022                      LD [PTR_TO_S_ON_0],HL  ;store addr DUMAREA at PTR_TO_S_ON_0
 022D 227024                      LD [PTR_TO_S_ON_1],HL  ;store addr DUMAREA at PTR_TO_S_ON_1
 0230 227026                      LD [PTR_TO_S_ON_2],HL  ;store addr DUMAREA at PTR_TO_S_ON_2
 0233 227028                      LD [PTR_TO_S_ON_3],HL  ;store addr DUMAREA at PTR_TO_S_ON_3
                             ;    * initialize SAVE_CTRL
 0236 3EFF                        LD A,0FFH              ;note: this is only time MSN SAVE_CTRL will be non zero,
 0238 32702A                      LD [SAVE_CTRL],A       ;thus ensuring PLAY_SONGS will output 1st real CTRL data
 023B                        ALL_OFF
                             ;    * turn off all 4 sound generators
 023B 3E9F                        LD A,OFF+SR1ATN        ;form off code for tone generator 1
 023D D3FF                        OUT [SOUND_PORT],A     ;send it out
 023F 3EBF                        LD A,OFF+SR2ATN        ;form off code for tone generator 2
 0241 D3FF                        OUT [SOUND_PORT],A     ;send it out
 0243 3EDF                        LD A,OFF+SR3ATN        ;form off code for tone generator 3
 0245 D3FF                        OUT [SOUND_PORT],A     ;send it out
 0247 3EFF                        LD A,OFF+SRNATN        ;form off code for noise generator, N
 0249 D3FF                        OUT [SOUND_PORT],A     ;send it out
 024B C9                          RET
 024C FF                     DUMAREA DEFB INACTIVE
                             ;    END ;INITSOU
                              PROG
```

```
LOCATION OBJECT CODE      SOURCE LINE

                          ;         .IDENT JUKEBOX          ;includes JUKE_BOX
                          ;****************************
                          ;*          JUKE_BOX          *
                          ;****************************
                          ;.COMMENT )
                          ;see Users' Manual for description
                          ;SONGNO passed in B
                          ;)
                                  GLB JUKE_BOX
                                  ;EXT PT_IX_TO_SxDATA,LOAD_NEXT_NOTE,UP_CH_DATA_PTRS
                                  ;INCLUDE OSSR_EQU:OS   ;equates

                          * PROCEDURE JUKE_BOXQ (SONG_NUM:BYTE)

                          * THIS IS THE PASCAL ENTRY POINT FOR JUKE_BOX

0240 00010001             JUKE_BOX_PAR     DEFW            1,1
                          * THE PARAMETER DESCRIPTOR FOR JUKE_BOXQ

                                           ;EXT            PARAM_
                          * THE PARAMETER PASSING ROUTINE

                          ;                COMN
                          ;PARAM_AREA       DEFS            1
                          * THIS IS WHERE THE SONG NUMBER IS ACTUALLY PASSED

                                           PROG
                                           GLB             JUKE_BOXQ
0251                      JUKE_BOXQ
0251 01024D                        LD   BC,JUKE_BOX_PAR
0254 11738A                        LD   DE,PARAM_AREA
0257 CD0098                        CALL PARAM_
025A 3A738A                        LD   A,[PARAM_AREA]
025D 47                            LD   B,A

025E                      JUKE_BOX
                          ;        * RET if song already in progress
025E C5                            PUSH BC                ;save SONGNO on stack
025F CD01C1                        CALL PT_IX_TO_SxDATA   ;point IX to SONGNO's song data area
0262 DD7E00                        LD A,[IX+0]            ;A := CH# [if any] | SONGNO [if any]
0265 E63F                          AND 3FH                ;A := 0 0 SONGNO
0267 C1                            POP BC                 ;B := SONGNO
0268 B8                            CP B                   ;test if already in progress
0269 C8                            RET Z                  ;if so, leave
                          ;         * load first note and set NEXT_NOTE_PTR [thru LOAD_NEXT_NOTE]
026A DD7000                         LD [IX+0],B             ;store SONGNO in byte 0
026D 2B                             DEC HL                  ;-HL left by PT_IX_TO_SxDATA pointing to MSB SxDATA
026E 2B                             DEC HL                  ;-entry in LST_OF_SND_ADDRS; point HL to note list
026F 56                             LD D,[HL]               ;-starting addr entry in LST_OF_SND_ADDRS and save this
0270 2B                             DEC HL                  ;-addr in DE
0271 5E                             LD E,[HL]               ;DE now has the initial value for NEXT_NOTE_PTR
0272 DD7301                         LD [IX+1],E             ;set NEXT_NOTE_PTR for first note in song
0275 DD7202                         LD [IX+2],D
0278 CD035F                         CALL LOAD_NEXT_NOTE     ;load note, byte 0 := CH#|SONGNO, set new NEXT_NOTE_PTR
027B CD0295                         CALL UP_CH_DATA_PTRS    ;new song, so update channel data ptrs
027E C9                             RET
                          ;         END        BOX
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     1124 ;         [page]
                     1125  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                   ;        .IDENT SNDMAN            ;includes SND_MANAGER,PROCESS_DATA_AREA,
                                                     ;UP_CH_DATA_PTRS
                   ;****************************
                   ;*       SND_MANAGER        *
                   ;****************************
                   ;.COMMENT )
                   ;See Users' Manual for description
                   ;)
                           GLB SND_MANAGER
                          ;EXT PT_IX_TO_SxDATA,AREA_SONG_IS
                          ;INCLUDE OSSR_EQU:OS  ;equates
027F               SND_MANAGER
                   ;        * IX := addr of song #1 data area [S1DATA]
027F 0601                  LD B,1                ;pt IX to byte 0 song data area for song # 1
0281 CD01C1                CALL PT_IX_TO_SxDATA
                   ;       LOOP until end of song data areas
0284 3E00          L1        LD A,ENDSDATA          ;check for end of song data areas
0286 DDBE00                  CP [IX+0]              ;set Z flag if inactive
0289 C8                      RET Z                  ;leave [Z set], if all data areas have been processed
                   ;           * process active song data areas
028A CD02D6                  CALL PROCESS_DATA_AREA  ;update counters or call effect; get next note
                   ;           * point IX to byte 0 next song data area
028D 1E0A                    LD E,10
028F 1600                    LD D,0
0291 DD19                    ADD IX,DE
0293 18EF                  JR L1 ;REPEAT LOOP
                   ;****************************
                   ;*     UP_CH_DATA_PTRS      *
                   ;****************************
                   ;.COMMENT )
                   ;For each active data area, starting with S1DATA and proceeding in order, load
                   ;the associated channel data area pointer [PTR_TO_S_ON_x] with the address of
                   ;byte 0.  This routine is called by JUKE_BOX, when a song starts and
                   ;PROCESS_DATA_AREA when the channel using a data area has changed as a result of
                   ;calling LOAD_NEXT_NOTE [this happens when a song finishes and when it switches
                   ;back and forth between noise and tone notes].
                   ;)
                           GLB UP_CH_DATA_PTRS
                          ;EXT DUMAREA
0295               UP_CH_DATA_PTRS
0295 DDE5                  PUSH IX                  ;save curent IX
0297 21024C                LD HL,DUMAREA            ;set all 4 ch data ptrs to dummy inactive area
029A 227022                LD [PTR_TO_S_ON_0],HL
029D 227024                LD [PTR_TO_S_ON_1],HL
02A0 227026                LD [PTR_TO_S_ON_2],HL
02A3 227028                LD [PTR_TO_S_ON_3],HL
02A6 0601                  LD B,1                   ;set IX to byte 0 S1DATA
02A8 CD01C1                CALL PT_IX_TO_SxDATA     ;RETs with IX addr byte 0 song 1
                   ;       LOOP until end of song data areas
02AB DD7E00        L2        LD A,[IX+0]            ;test for end of song data araes
02AE FE00                    CP ENDSDATA
02B0 2821                    JR Z,DONE_SNDMAN           ;leave loop if all data areas checked
                   ;           * if area active, set appropriate channel data area pointer
02B2 FEFF                      CP INACTIVE            ;check for inactive data area: don't up date ptr if so
                   ;             IF [PSW,IS,NZERO]    ;area is active: update channel data ptrs
02B4 2815                      JR Z,L9
02[illegible] [illegible]7E00            LD A,[illegible]]                [illegible]
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  02B9 E6C0       1184              AND 0C0H                 ;B7 - B6  in A  = CH#
  02BB 07         1185              RLCA                     ;form CH#  * 2  in A, i.e., the offset from
  02BC 07         1186              RLCA                     ;PTR_TO_S_ON_0  of the  channel data area pointer
  02BD 07         1187              RLCA                     ;that points  to channel  CH#
  02BE 5F         1188              LD E,A                   ;add offset to addr of PTR_TO_ S_ON_0
  02BF 1600       1189              LD D,0
  02C1 217022     1190              LD HL,PTR_TO_S_ON_0
  02C4 19         1191              ADD HL,DE                ;HL points to  proper channel  data area pointer
  02C5 DDE5       1192              PUSH IX                  ;store this song data area's byte  0  addr there
  02C7 D1         1193              POP DE
  02C8 73         1194              LD [HL],E
  02C9 23         1195              INC HL
  02CA 72         1196              LD [HL],D
                  1197 ;          ENDIF
                  1198 ;          * point IX to byte 0 next song data area
  02CB 1E0A       1199 L9          LD E,10
  02CD 1600       1200             LD D,0
  02CF DD19       1201             ADD IX,DE
  02D1 18D8       1202           JR L2 ;REPEAT LOOP
  02D3            1203 DONE_SNDMAN
  02D3 DDE1       1204           POP IX                 ;restore IX
  02D5 C9         1205           RET
                  1206 ;***************************
                  1207 ;*   PROCESS_DATA_AREA     *
                  1208 ;***************************
                  1209 ;.COMMENT )
                  1210 ;See Users' Manual for description
                  1211 ;Terminology: SFX = address of sound effect routine
                  1212 ;)
                  1213          GLB PROCESS_DATA_AREA,EFXOVER
                  1214         ;EXT LOAD_NEXT_NOTE,ATN_SWEEP,FREQ_SWEEP
  02D6            1215 PROCESS_DATA_AREA
  02D6 CD01E9     1216          CALL AREA_SONG_IS        ;return area's SONGNO in A [and addr SFX in HL]
  02D9 FEFF       1217          CP INACTIVE              ;test for inactive code
  02DB C8         1218          RET Z                    ;RET, no processing if area inactive
                  1219 ;          * if special effect, call it to process the data area
  02DC FE3E       1220           CP 62                   ;test for special sound effect
                  1221 ;          IF [PSW,IS,ZERO]           ;data area used by sound effect
  02DE 2006       1222           JR NZ,L10
  02E0 1E07       1223             LD E,7                  ;pt HLto SFX+7, starting adr of the effect's cod e
  02E2 1600       1224             LD D,0
  02E4 19         1225             ADD HL,DE
  02E5 E9         1226             JP [HL]                 ;do 1 pass thru effect, RET from effect
                  1227 ;         ENDIF
                  1228 ;          * else process a non-effect note
  02E6 CD012F     1229 L10         CALL ATN_SWEEP          ;process atn sweep data, if any
  02E9 CD00FC     1230             CALL FREQ_SWEEP         ;proc frq sweep data, if any, & note dura timer s
                  1231 ;           IF [PSW,IS,ZERO]        ;note is over
  02EC 2011       1232             JR NZ,L12
  02EE DD7E00     1233 EFXOVER       LD A,[IX+0]          ;A := CH# | SONGNO this note
  02F1 F5         1234               PUSH AF              ;save on stack
  02F2 CD035F     1235               CALL LOAD_NEXT_NOTE  ;load data for next note
  02F5 C1         1236               POP BC               ;B := CH# | SONGNO previous note
  02F6 DD7E00     1237               LD A,[IX+0]          ;A := CH# | SONGNO new note [may be inactive]
  02F9 B8         1238               CP B                 ;check against new note's CH# | SONGNO
                  1239 ;             IF [PSW,IS,NZERO]    ;change to/from tone/efx/noise
  02FA 2803       1240               JR Z,L12
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  02FC CD0295        1241                 CALL UP_CN_DATA_PTRS    ;to maintain data area priority system
                     1242 ;             ENDIF
                     1243 ;           ENDIF
  02FF C9            1244 L12          RET
                     1245 ;        END ;SNDMAN
                     1246  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                         ;         .IDENT PLAYSON          ;Includes TONE_OUT
                         ;****************************
                         ;*         PLAY_SONGS_         *
                         ;****************************
                         ;.COMMENT )
                         ;see Users' Manual for description
                         ;)
                                  GLB PLAY_SONGS_,TONE_OUT
                                  ;EXT UPATNCTRL,UPFREQ
                                  ;INCLUDE OSSR_EQU:OS  ;equates
                         ;        *** Sound chip register code EQUATES
                         ;        Tone generator frequency and attenuation formatted register codes
                         ;SR1FRQ  EQU 10000000B             ;BIT7 = 1, BIT6-4 = TONE GEN 1 FREQ CODE
                         ;SR1ATN  EQU 10010000B             ;BIT7 = 1, BIT6-4 = TONE GEN 1 ATTN CODE
                         ;SR2FRQ  EQU 10100000B             ;BIT7 = 1, BIT6-4 = TONE GEN 2 FREQ CODE
                         ;SR2ATN  EQU 10110000B             ;BIT7 = 1, BIT6-4 = TONE GEN 2 ATTN CODE
                         ;SR3FRQ  EQU 11000000B             ;BIT7 = 1, BIT6-4 = TONE GEN 3 FREQ CODE
                         ;SR3ATN  EQU 11010000B             ;BIT7 = 1, BIT6-4 = TONE GEN 1 ATTN CODE
                         ;        Noise generator control and attenuation formatted register codes
                         ;SRNCTL  EQU 11100000B             ;BIT7 = 1, BIT6-4 = NOISE GEN CONTROL CODE
                         ;SRNATN  EQU 11110000B             ;BIT7 = 1, BIT6-4 = NOISE GEN ATTN CODE
                         ;        Noise generator formatted control codes
           <0004>        WHITE    EQU 00000100B             ;BIT2 = 1, WHITE NOISE CODE
           <0000>        PERIOD   EQU 00000000B             ;BIT2 = 0, PERIODIC NOISE CODE
           <0000>        NSRHI    EQU 00000000B             ;BIT0-1 SET FOR HIGHEST NOISE SHIFT RATE [N/512]
           <0001>        NSRMED   EQU 00000001B             ;BIT0-1 SET FOR MEDIUM NOISE SHIFT RATE [N/1024]
           <0002>        NSRLOW   EQU 00000010B             ;BIT0-1 SET FOR LOWEST NOISE SHIFT RATE [N/2048]
           <0003>        NSRTG3   EQU 00000011B             ;BIT0-1 SET FOR SHIFT FROM TONE GEN 3 OUTPUT
0300                     PLAY_SONGS_
                         ;        * output CH1 attenuation and frequency
0300 3E9F                         LD A,OFF+SR1ATN           ;format CH1 OFF byte into A
0302 0E90                         LD C,SR1ATN               ;format MSN C for CH1 attenuation
0304 1680                         LD D,SR1FRQ               ;format MSN D for CH1 frequency
0306 DD2A7024                     LD IX,[PTR_TO_S_ON_1]     ;point IX to byte 0 data area of song for CH1
030A CD034E                       CALL TONE_OUT
                         ;        * output CH2 attenuation and frequency
030D 3EBF                         LD A,OFF+SR2ATN           ;format CH2 OFF byte into A
030F 0EB0                         LD C,SR2ATN               ;format MSN C for CH2 attenuation
0311 16A0                         LD D,SR2FRQ               ;format MSN D for CH2 frequency
0313 DD2A7026                     LD IX,[PTR_TO_S_ON_2]     ;point IX to byte 0 data area of song for CH2
0317 CD034E                       CALL TONE_OUT
                         ;        * output CH3 attenuation and frequency
031A 3EDF                         LD A,OFF+SR3ATN           ;format CH3 OFF byte into A
031C 0ED0                         LD C,SR3ATN               ;format MSN C for CH3 attenuation
031E 16C0                         LD D,SR3FRQ               ;format MSN D for CH3 frequency
0320 DD2A7028                     LD IX,[PTR_TO_S_ON_3]     ;point IX to byte 0 data area of song for CH3
0324 CD034E                       CALL TONE_OUT
                         ;        * output CH0 [noise] ATN [and CTRL, if different from last time]
0327 3EFF                         LD A,OFF+SRNATN           ;format CH0 OFF byte into A
0329 0EF0                         LD C,SRNATN               ;format MSN C for CH0 attenuation
032B DD2A7022                     LD IX,[PTR_TO_S_ON_0]     ;point IX to byte 0 data area of song for CH0
032F DD5E00                       LD E,[IX+0]               ;look for inactive code, 0FFH
0332 1C                           INC E                     ;this sets Z flag if E = 0FFH
                         ;        IF [PSW,IS,ZERO]          ; song data area is inactive
0333 2004                         JR NZ,L5
0335 D3FF                           OUT [SOUND_PORT],A        ;turn off CH0
0337 1814                           JR L6
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

   03A1 C30461                    JP MODB0               ;to load byte 0
                       ;       ENDIF
                       ;       - test for special sound effect
   03A4 E63C             L14   AND 00111100B          ;mask irrelevant bits
   03A6 FE04                   CP 00000100B           ;test for B5 - B2 = 0001
                       ;       IF [PSW,IS,ZERO]       ;note is a special effect
   03A8 2028                   JR NZ,L15
                                ;--CASE-- special effect
   03AA FDE1             EFFECT  POP IY               ;IY := SONGNO
   03AC FDE5                     PUSH IY              ;put SONGNO back on stack
   03AE C5                       PUSH BC              ;save header on stack; NEXT_NOTE_PTR := SFX, DE := SFX
   03AF 23                       INC HL               ;-pt HL to next byte [LSB addr SFX]
   03B0 5E                       LD E,[HL]            ;-E := LSB SFX
   03B1 DD7301                   LD [IX+1],E          ;-put LSB of SFX in byte 1 of SxDATA [NEXT_NOTE_PTR]
   03B4 23                       INC HL               ;-pt HL to MSB SFX
   03B5 56                       LD D,[HL]            ;-D := MSB SFX
   03B6 DD7202                   LD [IX+2],D          ;-put MSB SFX in byte 2 of SxDATA
   03B9 23                       INC HL               ;point HL to next note [after this new note]
   03BA FDE5                     PUSH IY              ;A := SONGNO
   03BC F1                       POP AF
   03BD D5                       PUSH DE              ;IY := SFX
   03BE FDE1                     POP IY
   03C0 1103C6                   LD DE,PASS1          ;create "CALL [IY]" with RET to PASS1 by storing
   03C3 D5                       PUSH DE              ;PASS1 on the stack
   03C4 FDE9                     JP [IY]              ;1st 7 bytes SFX will save addr next note & SONGNO
   03C6 1600             PASS1   LD D,0               ;in same fashion, create a "CALL (IY+7)"
   03C8 1E07                     LD E,7               ;to allow SFX to load initial values
   03CA FD19                     ADD IY,DE
   03CC 110461                   LD DE,MODB0          ;RET to MODB0
   03CF D5                       PUSH DE
   03D0 FDE9                     JP [IY]
                       ;       ENDIF
                       ;       - if here, note is type 0 - 3
   03D2 C5               L15   PUSH BC                ;save header on stack
   03D3 78                     LD A,B                 ;A := fresh copy header
   03D4 E603                   AND 00000011B          ;mask all but type number
   03D6 FE00                   CP 0                   ;test for type 0
                       ;       IF [PSW,IS,ZERO]       ;note is type 0: fixed freq and atn
   03D8 2020                   JR NZ,L16
                       ;         --CASE-- note type 0
                       ;         * set up NEXT_NOTE_PTR
   03DA 23               TYPE0   INC HL               ;next note [after this new note] is 4 bytes away,
   03DB 23                       INC HL               ;point HL to it
   03DC 23                       INC HL
   03DD 23                       INC HL
   03DE DD7501                   LD [IX+1],L          ;put addr in NEXT_NOTE_PTR
   03E1 DD7402                   LD [IX+2],H
                       ;         * move new note data and fill in bytes where necessary
   03E4 2B                       DEC HL               ;point HL back to 1st ROM data to move, NLEN
   03E5 110005                   LD DE,05             ;point DE to destination: bytes 5, 4, and 3
   03E8 CD0478                   CALL DE_TO_DEST
   03EB 010003                   LD BC,3              ;move 3 bytes
   03EE EDB8                     LDDR
   03F0 DD360700                 LD [IX+FSTEP],0      ;set for no freq sweep
   ...  DD360...                 ... [IX+...],0       ;... no ... sweep
   ...  1867                     ... MODB...
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

 03FA FE01          L16     CP 1                      ;test for type 1
                    ;       IF [PSW,IS,ZERO]          ;note is type 1: swept freq, fixed attenuation
 03FC 201B                  JR NZ,L17
                    ;         --CASE-- note type 1
                    ;         * set up NEXT_NOTE_PTR
 03FE 1E06          TYPE1     LD E,6                  ;note after this note is 6 bytes away,
 0400 1600                    LD D,0                  ;pt HL to it
 0402 19                      ADD HL,DE
 0403 DD7501                  LD [IX+1],L             ;store in NEXT_NOTE_PTR
 0406 DD7402                  LD [IX+2],H
                    ;         * move new note data and fill in bytes where necessary
 0409 2B                      DEC HL                  ;point HL back to 1st ROM data to move, FSTEP
 040A 1C                      INC E                   ;E := 7; point DE to destination: bytes 7 - 3
 040B CD0478                  CALL DE_TO_DEST
 040E 010005                  LD BC,5                 ;move 5 bytes
 0411 EDB8                    LDDR
 0413 DD360800                LD [IX+ASTEP],0         ;set for no atn sweep
 0417 1848                    JR MODB0
                    ;       ENDIF
 0419 FE02          L17     CP 2                      ;test for type 2
                    ;       IF [PSW,IS,ZERO]          ;note is type 2: fixed freq, swept attenuation
 041B 2028                  JR NZ,TYPE3
                    ;         --CASE-- note type 2
                    ;         * set up NEXT_NOTE_PTR
 041D 1E06          TYPE2     LD E,6                  ;pt HL to note after this note; since it's 6 bytes away,
 041F 1600                    LD D,0                  ;pt HL to it by adding 6
 0421 19                      ADD HL,DE
 0422 F1                      POP AF                  ;A := header this note [CH# | SONGNO]
 0423 F5                      PUSH AF                 ;put back on stack
 0424 E6C0                    AND 11000000B           ;mask SONGNO, leaving CH#
                    ;         IF [PSW,IS,ZERO]        ;this is a noise note, which is only 5 ROM bytes long
 0426 2001                    JR NZ,L18
 0428 2B                        DEC HL                  ;so move HL back 1 byte
                    ;         ENDIF
 0429 DD7501        L18       LD [IX+1],L             ;put addr in NEXT_NOTE_PTR
 042C DD7402                  LD [IX+2],H
                    ;         * move new note data and fill in bytes where necessary
 042F 2B                      DEC HL                  ;point HL back to 1st ROM data to move, APS
 0430 1E09                    LD E,9                  ;point DE to destination: bytes 9,8,5 - 3
 0432 CD0478                  CALL DE_TO_DEST
 0435 010002                  LD BC,2                 ;move 2 bytes
 0438 EDB8                    LDDR                    ;when done, DE points to FSTEP, HL to ROM NLEN
 043A 3E00                    LD A,0
 043C 12                      LD [DE],A               ;FSTEP := 0 for no freq sweep
 043D 1B                      DEC DE                  ;pt DE to RAM NLEN
 043E 1B                      DEC DE
 043F 0E03                    LD C,3                  ;move last 3 ROM bytes; if this is a noise note, garbage
 0441 EDB8                    LDDR                    ;will be loaded into byte 3, buts that's OK
 0443 181C                    JR MODB0
                    ;       ENDIF
                    ;       if here, note is type 3: swept freq, swept attenuation
                    ;       --CASE-- note type 3
                    ;       * set up NEXT_NOTE_PTR
 0445 1E08          TYPE3   LD E,8                    ;note after this note is 8 bytes away,
 0447 1600                  LD D,0                    ;pt HL to it
 0449 19                    ADD HL,DE
 044A DD7501                LD [IX+1],L               ;put addr in NEXT_NOTE_PTR
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0440 DD7402                  LD [IX+2],H
                        ;      * move new note data and fill in bytes where necessary
  0450 2B                      DEC HL                 ;point HL back to 1st ROM data to move, APS
  0451 DDE5                    PUSH IX                ;point DE to destination: bytes 9 - 3
  0453 FDE1                    POP IY                 ;IY := addr byte 0 [and DE = 6]
  0455 1E09                    LD E,9                 ;DE := 9
  0457 FD19                    ADD IY,DE              ;IY := addr byte 9 [APS]
  0459 FDE5                    PUSH IY
  045B D1                      POP DE                 ;DE := addr APS
  045C 010007                  LD BC,7                ;move 7 bytes
  045F EDB8                    LDDR
                        ;      * modify byte 0 basis header new note
  0461 DDE5             MODB0  PUSH IX                ;pt HL to byte 0
  0463 E1                      POP HL
  0464 F1                      POP AF                 ;A := header new note
  0465 C1                      POP BC                 ;B := SONGNO
  0466 FEFF                    CP INACTIVE            ;test for inactive [song over, as detected above]
  0468 C8                        RET Z
  0469 57                      LD D,A                 ;save header in D
  046A E63F                    AND 3FH                ;Rid channel bits
  046C FE04                    CP 04                  ;Special effect
  046E 2002                    JR NZ,L20_LOAD_NEX
  0470 063E                      LD B,62
  0472                  L20_LOAD_NEX:
  0472 7A                      LD A,D                 ;restore A to header
  0473 E6C0                    AND 0C0H               ;A := CH# 0 0 0 0 0 0
  0475 B0                      OR B                   ;A := new CH# | SONGNO
  0476 77                      LD [HL],A              ;store back in byte 0
                        ;      ENDIF
  0477 C9               L19    RET
  0478                  DE_TO_DEST                    ;DE passed = offset from byte 0, RETed w addr byte offset
  0478 DDE5                    PUSH IX
  047A FDE1                    POP IY                 ;IY := addr byte 0 [and DE = offset]
  047C FD19                    ADD IY,DE              ;IY := addr byte 0 + offset
  047E FDE5                    PUSH IY
  0480 D1                      POP DE                 ;DE := addr of destination byte in SxDATA
  0481 C9                      RET
                        ;      END ;LOADNEX
                         PROG
```

LOCATION OBJECT CODE LINE SOURCE LINE

```
;        .IDENT  ACTIVATE
;        .ZOP
;        .EPOP
;        .COMMENT )
;********************* ACTIVATE *************************************************
;
;                                                                   4/22/82
;                                                                   13:50:00
;
;THE FOLLOWING CHANGES/REVISIONS WERE MADE:
;
;        1. ELIMINATE CODE PLACING OLD.SCREEN ADDRESS IN STATUS AREA
;        2. INIT X.PAT.POS IN OLD.SCREEN WHEN IN VRAM AS WELL AS WHEN IN CRAM
;        3. USE VDP.MODE.WORD TO TEST GRAPHICS MODE
;        4. ADD CODE TO EXPAND ONE COLOR GENERATOR BYTE TO 8
;        5. ADDED C_BUFF DEFS 8 FOR COLOR EXPANDING CODE
; 5/02   6. FIX COLOR GEN MOVE IN MODE I
;        7. USE CONTROLER_MAP FOR BUFFER AREA
;
*  ACTIVATE is used to initialize the RAM status area for the passed
*object and move its pattern and color generators to the PATTERN and
*COLOR GENERATOR tables in VRAM_  The second function is enabled or
*disabled by setting or resetting the carry flag in the PSW_  this is
*necessary to prevent sending the same graphics data to VRAM more than
*once when creating identical objects_  The calling sequence for act-
*ivating an object is as follows:
*
*               LD          HL,OBJ_n        ;->OBJ TO ACTIVATE
*               SCF                         ;SIGNAL MV TO VRAM
*               CALL        ACTIVATE
*
*
*OR
*
*               LD          HL,OBJ_n        ;->OBJ TO ACTIVATE
*               OR          A               ;DON'T MV TO VRAM
*               CALL        ACTIVATE
*
*)
               ;EXT         PUT_VRAM_,VRAM_WRITE,VDP_MODE_WORD
               ;EXT         WORK_BUFFER
;
               GLB          ACTIVATE_
;
;REGISTER USAGE: FOLLOWING WILL BE CHANGED BY ACTIVATE, ADDITIONAL
;MAY BE CHANGED BY CALLED SUBR
;              AF,HL,DE,BC,IY
;


; PROCEDURE ACTIVATEQ[VAR OBJ:OBJECT;MOVE:BOOLEAN];

; ACTIVATEQ IS THE PASCAL ENTRY POINT TO ACTIVATE

               ;EXT     PARAM_
; THE PASCAL PARAMETER PASSING PROCEDURE
;              COMN
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      ;PRM_AREA:     DEFS          3          ;Moved to OS
                      ; THIS IS THE COMMON PARAMETER PASSING AREA

                                     PROG
 0482 0002FFFE        ACTIVATE_P:    DEFW          2,-2,1
 0486 0001

                                     GLB           ACTIVATEQ
       <0488>         ACTIVATEQ      EQU           $
 0488 010482                         LD            BC,ACTIVATE_P
 048B 11738A                         LD            DE,PRM_AREA
 048E CD0098                         CALL          PARAM_
 0491 2A738A                         LD            HL,[PRM_AREA]
 0494 5E                             LD            E,[HL]
 0495 23                             INC           HL
 0496 56                             LD            D,[HL]
 0497 EB                             EX            DE,HL
 0498 3A738C                         LD            A,[PRM_AREA+2]
 049B FE00                           CP            0
 049D 2803                           JR            Z,NTZZZ_
 049F 37                             SCF
 04A0 1801                           JR            TZZZ_
 04A2 B7              NTZZZ_:         OR            A
 04A3                 TZZZ_:

       <04A3>         ACTIVATE_      EQU           $
                      ;SUP POINTERS ETC_ COMMON TO ALL SUBCASES
                      ;              HL->OBJ DEF CROM
                      ;              C FLG=SUP VRAM FLG
 04A3 5E                             LD            E,[HL]            ;->OBJ GEN CROM
 04A4 23                             INC           HL
 04A5 56                             LD            D,[HL]
 04A6 23                             INC           HL
 04A7 4E                             LD            C,[HL]            ;->OBJ CRAM
 04A8 23                             INC           HL
 04A9 46                             LD            B,[HL]
 04AA 23                             INC           HL
 04AB 3E00                           LD            A,0               ;ZERO FRAME
 04AD 02                             LD            [BC],A
 04AE 1A                             LD            A,[DE]            ;GET OBJ_TYPE
 04AF F5                             PUSH          AF                ;SV OBJ_TYPE & FLG
 04B0 E60F                           AND           0FH               ;GET OBJ_TYPE NUM
 04B2 CA04E7                         JP            Z,ACT_SEMI                  ;TYPE=0
 04B5 3D                             DEC           A
 04B6 CA05F1                         JP            Z,ACT_MOBILE                ;TYPE=1
 04B9 3D                             DEC           A
 04BA CA0600                         JP            Z,ACT_0SPRT                 ;TYPE=2
 04BD 3D                             DEC           A
 04BE CA0600                         JP            Z,ACT_1SPRT                 ;TYPE=3
 04C1 3D                             DEC           A
 04C2 2802                           JR            Z,ACT_CMPLX                 ;TYPE=4
 04C4 F1                             POP           AF                ;SUBCASE ELSE
 04C5 C9                             RET
                      ;ON ENTRY TO SUBCASES:
                      ;              STACK=OBJ TYPE & SUP VRAM FLG
                      ;              ->OB_
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                    ;                DE->OBJ GRAPHICS+0
                    ;                BC->OBJ STATUS+0
                    ;                A=0
                    ;
         <04C6>     ACT_CMPLX  EQU          $
                    ; SUBCASE Complex
04C6 1A                          LD      A,[DE]          ;GET COMP_CNT
04C7 1F                          RRA
04C8 1F                          RRA
04C9 1F                          RRA
04CA 1F                          RRA
04CB E60F                        AND     0FH
04CD 47                          LD      B,A             ;SET CNTR
04CE 5E                          LD      E,[HL]          ;DE->COMP PTRS LIST
04CF 23                          INC     HL
04D0 56                          LD      D,[HL]
04D1 23                          INC     HL
04D2 B7                          OR      A               ;? EMPTY
04D3 2810                        JR      Z,CMPLX9
         <04D5>     CMPLX4      EQU     $
04D5 F1                          POP     AF              ;SUP CALL, COMP OBJ
04D6 F5                          PUSH    AF
04D7 E5                          PUSH    HL
04D8 C5                          PUSH    BC
04D9 EB                          EX      DE,HL
04DA CD04A3                      CALL    ACTIVATE_
04DD C1                          POP     BC              ;RESTORE PNTRS
04DE E1                          POP     HL
04DF 5E                          LD      E,[HL]
04E0 23                          INC     HL
04E1 56                          LD      D,[HL]
04E2 23                          INC     HL
04E3 10F0                        DJNZ    CMPLX4          ;? MORE, RELOOP
04E5 F1             CMPLX9:      POP     AF              ;CLEAR STACK FOR RTN
04E6 C9                          RET     ;TECHNICALLY SHOULD JMP TO RTN
                    ;
         <04E7>     ACT_SEMI   EQU          $
                    ; SUBCASE Semi_Moblile
04E7 CD0572                      CALL    INIT_XP_OS      ;X_PAT_POS := 80H
04EA 1A                          LD      A,[DE]          ;A := FIRST_GEN_NAME
04EB 6F                          LD      L,A
04EC 13                          INC     DE
04ED 1A                          LD      A,[DE]          ;A := NUMGEN
04EE 85                          ADD     A,L
04EF FD7705                      LD      [IY+5],A        ;NEXT_GEN := FIRST_GEN_NAME + NUMGEN
04F2 2600                        LD      H,0             ;HL=FIRST_GEN_NAME
                    ;AT THIS POINT:
                    ;                STACK=OBJ_TYPE & SUP VRAM FLG
                    ;                HL=FIRST_GEN_NAME
                    ;                DE->NUMGEN
                    ;                BC:FREE
                    ; SUP FOR VRAM INIT
04F4 F1                          POP     AF              ;IF SUP VRAM FLG ON
04F5 3038                        JR      NC,SEMI_EXIT
04F7 F5                          PUSH    AF
04F8 3A73C3                      LD      A,[VDP_MODE_WORD] ;SEE WHICH GRAPHICS MODE
04FB CB4F                        BIT     1,A             ;IF GR II MODE
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

 04FD 2831                          JR      Z,SEMI_GRI      ;·, GO GRI
 04FF EB                            EX      DE,HL           ;DE=FIRST_GEN_NAME
 0500 44                            LD      B,H             ;SV -> NUMGEN
 0501 4D                            LD      C,L
 0502 6E                            LD      L,[HL]          ;CALC SOURCE OFFSET
 0503 2600                          LD      H,0
 0505 E5                            PUSH    HL
 0506 29                            ADD     HL,HL
 0507 29                            ADD     HL,HL
 0508 29                            ADD     HL,HL
 0509 E5                            PUSH    HL
 050A 03                            INC     BC              ;HL->SOURCE BUFFER
 050B 0A                            LD      A,[BC]
 050C 6F                            LD      L,A
 050D 03                            INC     BC
 050E 0A                            LD      A,[BC]
 050F 67                            LD      H,A
 0510 C1                            POP     BC
 0511 FDE1                          POP     IY
 0513 F1                            POP     AF
                 ;AT THIS POINT:
                 ;                  HL->SOURCE BUFFER, PTRN GNRTRS
                 ;                  DE=INDEX TO START OF VRAM ENTRIES
                 ;                  IY=NUMBER OF ITEMS TO READ FROM VRAM
                 ;                  BC=OFFSET TO COLOR SOURCE BUFFER @
                 ;                  AF=OBJ_TYPE [& SUP VRAM FLG, UNNEEDED]
                 ; FILL AS NEEDED TOP, MID, AND BOT PTRN_GNRTRS & DITTO FOR COLOR_GNRTRS
 0514 CB7F                          BIT     7,A             ;IF BIT 7 OBJ_TYPE ON (TOP)
 0516 2803                          JR      Z,SEMI_MID      ;·, GO HNDL MID
 0518 CD0594                        CALL    SUP_GEN_CLR
      <051B>     SEMI_MID           EQU     $
 051B CD05E8                        CALL    SUP_UPDATE
 051E CB77                          BIT     6,A             ;IF BIT 6 OBJ_TYPE ON (MID)
 0520 2803                          JR      Z,SEMI_BOT
 0522 CD0594                        CALL    SUP_GEN_CLR
      <0525>     SEMI_BOT           EQU     $
 0525 CD05E8                        CALL    SUP_UPDATE
 0528 CB6F                          BIT     5,A             ;IF BIT 5 OBJ_TYPE ON (BOT)
 052A 2803                          JR      Z,SEMI_EXIT
 052C CD0594                        CALL    SUP_GEN_CLR
      <052F>     SEMI_EXIT          EQU     $
 052F C9                            RET
                 ;
                 ; Handle GRAPHICS MODE I
      <0530>     SEMI_GRI           EQU     $
 0530 EB                            EX      DE,HL           ;HL->NUMGEN
 0531 4E                            LD      C,[HL]          ;IY=NUMGEN
 0532 0600                          LD      B,0
 0534 C5                            PUSH    BC
 0535 FDE1                          POP     IY
 0537 23                            INC     HL              ;HL->PTRN_GNRTRS
 0538 7E                            LD      A,[HL]
 0539 23                            INC     HL
 053A 66                            LD      H,[HL]
 053B 6F                            LD      L,A
 053C E5                            PUSH    HL              ;SAVE FOR RESTORE
 053D C5                            PUSH    BC
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  053E D5          1772                PUSH        DE
  053F FDE5        1773                PUSH        IY
  0541 3E03        1774                LD          A,3          ;SIGNAL PTRN GEN FILL
  0543 CD1C27      1775                CALL        PUT_VRAM_
  0546 C1          1776                POP BC  ;BC := NUMGEN
  0547 E1          1777                POP HL  ;HL := FIRST_GEN_NAME
  0548 50          1778                LD E,L
  0549 54          1779                LD D,H  ;DE := FIRST_GEN_NAME
  054A 09          1780                ADD HL,BC ;HL := FIRST_GEN_NAME + NUMGEN
  054B 2B          1781                DEC HL
  054C CB3C        1782                SRL H
  054E CB1D        1783                RR L
  0550 CB3C        1784                SRL H
  0552 CB1D        1785                RR L
  0554 CB3C        1786                SRL H
  0556 CB1D        1787                RR L    ;HL := (FIRST_GEN_NAME + NUMGEN - 1)/8
  0558 CB2B        1788                SRA E
  055A CB2B        1789                SRA E
  055C CB2B        1790                SRA E   ;DE := FIRST_GEN_NAME/8
  055E B7          1791                OR A    ;CLEAR CARRY
  055F ED52        1792                SBC HL,DE
  0561 23          1793                INC HL  ;HL := (F_G_N + NMGN - 1)/8 - F_G_N/8 + 1 = NUMBER COLR GENS
  0562 E5          1794                PUSH HL
  0563 FDE1        1795                POP IY
  0565 E1          1796                POP         HL           ;RESTORE REG
  0566 29          1797                ADD         HL,HL        ;STEP OVER PTRN_GNRTRS
  0567 29          1798                ADD         HL,HL
  0568 29          1799                ADD         HL,HL
  0569 C1          1800                POP         BC
  056A 09          1801                ADD         HL,BC        ;HL->COLOR GNRTR SOURCE
  056B 3E04        1802                LD          A,4          ;SIGNAL PTRN COLOR TBL
  056D CD1C27      1803                CALL        PUT_VRAM_
  0570 F1          1804                POP         AF           ;FIX STACK
  0571 C9          1805                RET
                   1806 ; Internal routine to initialize X_Pat_Pos in Old_Screen
  0572 C5          1807 INIT_XP_OS:    PUSH        BC
  0573 FDE1        1808                POP         IY           ;IY -> STATUS
  0575 D5          1809                PUSH        DE           ;SAVE -> GRAPHICS
  0576 5E          1810                LD          E,(HL)       ;DE := OLD_SCREEN ADDRESS
  0577 23          1811                INC         HL
  0578 56          1812                LD          D,(HL)
  0579 CB7A        1813                BIT         7,D          ;? OLD SCRN IN CROM
  057B 2014        1814                JR          NZ,SM_BY_OLD
  057D 7A          1815                LD          A,D          ;OLD_SCREEN IN VRAM?
  057E FE70        1816                CP          70H
  0580 3806        1817                JR          C,OS_IN_VRAM
  0582 3E80        1818                LD          A,80H        ;INIT X_PAT_POS = 80H
  0584 12          1819                LD          (DE),A
  0585 180A        1820                JR          SM_BY_OLD
  0587 80          1821 INIT_80:       DEFB        80H
  0588 210587      1822 OS_IN_VRAM:    LD          HL,INIT_80
  058B 010001      1823                LD          BC,1         ;ONE BYTE TO MOVE TO VRAM
  058E CD1D01      1824                CALL        VRAM_WRITE
        <0591>     1825 SM_BY_OLD     EQU          $
  0591 D1          1826                POP         DE           ;DE -> GRAPHICS
  0592 13          1827                INC         DE           ;DE -> FIRST_GEN_NAME
  0593 C9          1828                RET
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     1829 ;
                     1830 ;Internal rout to setup Ptrn Gen VRAM & Color Gen VRAM
           <0594>    1831 SUP_GEN_CLR    EQU           $
 0594 F5             1832                PUSH          AF             ;SAVE FOR RESTORE
 0595 C5             1833                PUSH          BC
 0596 FDE5           1834                PUSH          IY
 0598 D5             1835                PUSH          DE
 0599 E5             1836                PUSH          HL
 059A 3E03           1837                LD            A,3            ;SIGNAL PTRN GEN FILL
 059C CD1C27         1838                CALL          PUT_VRAM_
 059F E1             1839                POP           HL             ;RESTORE
 05A0 D1             1840                POP           DE
 05A1 FDE1           1841                POP           IY
 05A3 C1             1842                POP           BC
 05A4 F1             1843                POP           AF
 05A5 F5             1844                PUSH          AF
 05A6 C5             1845                PUSH          BC             ;SAVE FOR RESTORE
 05A7 FDE5           1846                PUSH          IY
 05A9 D5             1847                PUSH          DE
 05AA E5             1848                PUSH          HL
 05AB CB67           1849                BIT           4,A            ;HOW MANY COLOR GEN BYTES?
 05AD 2000           1850                JR            NZ,ONE_BYTE
 05AF 09             1851                ADD           HL,BC          ;HL->COLOR GEN SOURCE
 05B0 3E04           1852                LD            A,4            ;SIGNAL PTRN COLOR FILL
 05B2 CD1C27         1853                CALL          PUT_VRAM_
 05B5 E1             1854 O_B_RET:       POP           HL
 05B6 D1             1855                POP           DE
 05B7 FDE1           1856                POP           IY
 05B9 C1             1857                POP           BC
 05BA F1             1858                POP           AF
 05BB C9             1859                RET
                     1860 ; For each item to send, duplicate the color byte 8 times [in C_BUFF]
                     1861 ; then send this generator to VRAM color table indexed by DE
 05BC                1862 ONE_BYTE:
 05BC 09             1863          ADD HL,BC                 ;HL -> COLOR BYTE
 05BD 4D             1864          LD C,L
 05BE 44             1865          LD B,H                    ;BC -> COLOR BYTE
 05BF FDE5           1866          PUSH IY
 05C1 E1             1867          POP HL                    ;HL = ITEM COUNT
 05C2                1868 NEXT_COLOR:
 05C2 E5             1869          PUSH HL                   ;SAVE COUNTER
 05C3 0A             1870          LD A,[BC]                 ;GET COLOR BYTE
 05C4 C5             1871          PUSH BC                   ;SAVE POINTER TO COLOR
 05C5 010008         1872          LD BC,8                   ;CREATE 8 DUPLICATES
 05C8 2A8006         1873          LD HL,[WORK_BUFFER]
 05CB 09             1874          ADD HL,BC                 ;PLACE THEM HERE, STARTING AT END OF BUFFER
 05CC 0608           1875          LD B,8
 05CE 2B             1876 DUPLI:   DEC HL
 05CF 77             1877          LD [HL],A
 05D0 10FC           1878          DJNZ DUPLI
 05D2 D5             1879          PUSH DE                   ;SAVE INDEX INTO TABLES
 05D3 FD210001       1880          LD IY,1                   ;1 ITEM TO SEND
 05D7 3E04           1881          LD A,4                    ;COLOR TABLE CODE
 05D9 CD1C27         1882          CALL PUT_VRAM_
 05DC D1             1883          POP DE                    ;GET INDEX BACK
 05DD C1             1884          POP BC                    ;POINTER TO COLOR BYTE
 05DE 13             1885          INC DE                    ;INCREMENT INDEX
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  05DF 03                       INC BC                  ;INCREMENT COLOR POINTER
  05E0 E1                       POP HL                  ;GET ITEM COUNTER
  05E1 2B                       DEC HL
  05E2 7C                       LD A,H
  05E3 B5                       OR L
  05E4 200C                     JR NZ,NEXT_COLOR
  05E6 18CD                     JR O_B_RET
                          ;Internal rout to update to next VRAM index screen area
       <05E8>             SUP_UPDATE     EQU        $
  05E8 C5                                PUSH       BC
  05E9 010100                            LD         BC,100H
  05EC EB                                EX         DE,HL
  05ED 09                                ADD        HL,BC
  05EE EB                                EX         DE,HL
  05EF C1                                POP        BC
  05F0 C9                                RET
                          ;
       <05F1>             ACT_MOBILE     EQU        $
                          ; SUBCASE Mobile
  05F1 CD0572                            CALL       INIT_XP_OS      ;X_PAT_POS := 80H
                          ; INSERT NEW_GENERATOR ADDRESS IN OBJECT CRAM
  05F4 13                                INC        DE
  05F5 1A                                LD         A,[DE]
  05F6 FD7705                            LD         [IY+5],A
  05F9 13                                INC        DE
  05FA 1A                                LD         A,[DE]
  05FB FD7706                            LD         [IY+6],A        ;INIT NEW_GEN IN STATUS
  05FE F1                                POP        AF
  05FF C9                                RET
       <0600>             ACT_OSPRT  EQU          $
                          ; SUBCASE Sprite size 0
       <0600>             ACT_1SPRT  EQU          $
                          ; SUBCASE Sprite size 1
  0600 03                                INC        BC              ;->NEXT_GEN IN CRAM
  0601 03                                INC        BC
  0602 03                                INC        BC
  0603 03                                INC        BC
  0604 03                                INC        BC
  0605 EB                                EX         DE,HL           ;HL->FIRST_GEN_NAME
  0606 23                                INC        HL
  0607 7E                                LD         A,[HL]
  0608 5F                                LD         E,A             ;SV INDEX TO VRAM
  0609 1600                              LD         D,0
  060B D5                                PUSH       DE
  060C 23                                INC        HL              ;DE=PTRN_PTR
  060D 5E                                LD         E,[HL]
  060E 23                                INC        HL
  060F 56                                LD         D,[HL]
  0610 23                                INC        HL              ;CALC & SET NEXT_GEN CRAM
  0611 86                                ADD        A,[HL]
  0612 02                                LD         [BC],A
  0613 4E                                LD         C,[HL]
  0614 0600                              LD         B,0
  0616 C5                                PUSH       BC
  0617 FDE1                              POP        IY
  0619 EB                                EX         DE,HL           ;HL->SOURCE PTRN GEN
  061A D1                                POP        DE              ;DE=INDEX TO PTRN GEN VRAM
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  061B F1          1943             POP     AF
  061C D0          1944             RET     NC
  061D 3E01        1945             LD      A,1         ;SIGNAL SPRITE PRTN GEN FILL
  061F CD1C27      1946             CALL    PUT_VRAM_
  0622 C9          1947             RET
                   1948
                   1949   PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     1951
                     1952 ********************  PUTOBJ  ********************************************
                     1953 ;DESCRIPTION:     PUTOBJ VECTORS TO ONE OF 5 SPECIFIC ROUTINES FOR PLACING THE
                     1954 ;                 DIFFERENT OBJECT TYPES ON THE DISPLAY
                     1955 ;INPUT:            IX = ADDRESS OF OBJECT TO BE PROCESSED
                     1956 ;                  B = PARAMETER TO BE PASSED SPECIFIC PUT ROUTINES
                     1957
                     1958 * IN ADDITION, THIS MODULE CONTAINS ROUTINES WHICH ALLOW VRAM OPERATIONS
                     1959 * TO BE DEFERRED, TYPICALLY UNTIL AN INTERRUPT OCCURS, AND PERFORMED
                     1960 * IN A BLOCK BY A CENTRAL WRITER ROUTINE.
                     1961 ****************************************************************************
                     1962
                     1963                  DATA
  73CA               1964 QUEUE_SIZE       DEFS            1
                     1965 * THIS IS THE SIZE OF THE DEFERRED WRITE QUEUE. IT IS SET BY THE
                     1966 * CARTRIDGE PROGRAMMER. IT HAS RANGE 0 - 255.
                     1967
  73CB               1968 QUEUE_HEAD       DEFS            1
  73CC               1969 QUEUE_TAIL       DEFS            1
                     1970 * THESE ARE THE INDICES OF THE HEAD AND TAIL OF THE WRITE QUEUE.
                     1971
  73CD               1972 HEAD_ADDRESS     DEFS            2
  73CF               1973 TAIL_ADDRESS     DEFS            2
                     1974 * THESE ARE THE ADDRESSES OF THE QUEUE HEAD AND TAIL
                     1975
                     1976 ;TRUE             EQU             1
                     1977 ;FALSE            EQU             0
                     1978 * VALUES FOR BOOLEAN DEFERAL_FLAG
                     1979
  73D1               1980 BUFFER           DEFS            2
                     1981 * THIS IS A POINTER TO THE BEGINNING OF THE DEFERRED WRITE QUEUE. THE
                     1982 * CARTRIDGE PROGRAMMER IS RESPONSIBLE FOR PROVIDING A RAM AREA TO HOLD
                     1983 * THE QUEUE, AND PASSING ITS LOCATION AND SIZE TO INIT_QUEUE.
                     1984
                     1985 ;                 COMN
                     1986 ;PARAM_AREA       DEFS            3
                     1987 * PARAM_AREA IS THE COMMON PARAMETER PASSING AREA FOR PASCAL ENTRY PTS
                     1988
                     1989
                     1990                  PROG
          <0623>     1991 SET_UP_WRITE     EQU             $
                     1992
                     1993 * SET_UP_WRITE SETS UP A DEFERRED VRAM OPERATION.
                     1994
                     1995 *   PUT DATA AT QUEUE_HEAD
0623 DDE5            1996                  PUSH            IX
0625 2A73CD          1997                  LD              HL,[HEAD_ADDRESS]
0628 D1              1998                  POP             DE
0629 73              1999                  LD              [HL],E          ; PUT DATA POINTER
062A 23              2000                  INC             HL
062B 72              2001                  LD              [HL],D
062C 23              2002                  INC             HL
062D 70              2003                  LD              [HL],B          ;STORE PUTOBJ PARAMETER
062E 23              2004                  INC             HL
062F EB              2005                  EX              DE,HL           ; HEAD ADDRESS IN DE
                     2006
                     2007 *   INCREMENT QUEUE_HEAD
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0630 3A73CB                        LD          A,[QUEUE_HEAD]
  0633 3C                            INC         A                  ; NEW HEAD IN A

                           *   IF QUEUE_HEAD = QUEUE_SIZE THEN
  0634 2173CA                        LD          HL,QUEUE_SIZE
  0637 BE                            CP          [HL]
  0638 2000                          JR          NZ,NOT_TOO_BIG

                           *      QUEUE_HEAD := 0
  063A 3E00                          LD          A,0
  063C 3273CB                        LD          [QUEUE_HEAD],A

                           *      HEAD_ADDRESS := BUFFER
  063F 2A7301                        LD          HL,[BUFFER]
  0642 2273CD                        LD          [HEAD_ADDRESS],HL

  0645 1807                          JR          SET_UP_ENDIF
                           *   ELSE
        <0647>             NOT_TOO_BIG       EQU         $

                           *      STORE NEW QUEUE_HEAD
  0647 3273CB                        LD          [QUEUE_HEAD],A

                           *      STORE HEAD_ADDRESS
  064A ED5373CD                      LD          [HEAD_ADDRESS],DE

                           *   END IF
  064E                     SET_UP_ENDIF

                           * END SET_UP_WRITE
  064E C9                            RET

                           * PROCEDURE  INIT_QUEUE (SIZE:BYTE;VAR A_QUEUE:QUEUE)

                           * SIZE PASSED IN A, LOCATION PASSED IN HL
                           * DESTROYS: A

  064F 00020001            INIT_QUEUE_P      DEFW        2,1,-2
  0653 FFFE
                           * THIS IS THE PARAMETER DESCRIPTOR FOR INIT_QUEUEQ

                           * BEGIN INIT_QUEUE
                                             GLB         INIT_QUEUEQ
        <0655>             INIT_QUEUEQ       EQU         $
  0655 01064F                        LD          BC,INIT_QUEUE_P
  0658 11738A                        LD          DE,PARAM_AREA
  065B CD0098                        CALL        PARAM
  065E 3A738A                        LD          A,[PARAM_AREA]
  0661 2A738B                        LD          HL,[PARAM_AREA+1]

                                             GLB         INIT_QUEUE
        <0664>             INIT_QUEUE        EQU         $

                           *   QUEUE_SIZE := SIZE
  0664 3273CA                        LD          [QUEUE_SIZE],A

                           *   QUEUE_HEAD := QUEUE_TAIL := 0
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

 0667 3E00                          LD          A,0
 0669 3273CB                        LD          [QUEUE_HEAD],A
 066C 3273CC                        LD          [QUEUE_TAIL],A

                          *   BUFFER := TAIL_ADDRESS := HEAD_ADDRESS := LOCATION
 066F 227301                        LD          [BUFFER],HL
 0672 2273CD                        LD          [HEAD_ADDRESS],HL
 0675 2273CF                        LD          [TAIL_ADDRESS],HL

                          * END INIT_QUEUE
 0678 C9                            RET

                          * PROCEDURE  WRITER_

                          * TAKES NO PARAMETERS
                          * DESTROYS: ALL

                          * BEGIN WRITER_
                                    GLB         WRITER_
        <0679>              WRITER_ EQU         $

                          *   SAVE DEFERAL FLAG
 0679 3A73C6                        LD          A,[DEFER_WRITES]
 067C F5                            PUSH        AF

                          *   DEFER_WRITES := FALSE
 067D 3E00                          LD          A,FALSE
 067F 3273C6                        LD          [DEFER_WRITES],A

                          *   WHILE QUEUE_TAIL <> QUEUE_HEAD DO
        <0682>              WRTR_WHILE EQU      $
 0682 3A73CC                        LD          A,[QUEUE_TAIL]
 0685 2173CB                        LD          HL,QUEUE_HEAD
 0688 BE                            CP          [HL]
 0689 2831                          JR          Z,WRTR_END_WHILE

                          *     WRITE DATA AT QUEUE_TAIL TO VRAM
 068B 2A73CF                        LD          HL,[TAIL_ADDRESS]
 068E 5E                            LD          E,[HL]          ; GET OBJECT POINTER
 068F 23                            INC         HL
 0690 56                            LD          D,[HL]
 0691 23                            INC         HL
 0692 46                            LD          B,[HL]          ; GET PARAMETER
 0693 23                            INC         HL

                          *     PROCESS OBJECT IN QUEUE
 0694 D5                            PUSH        DE
 0695 DDE1                          POP         IX
 0697 E5                            PUSH        HL              ;SAVE QUEUE TAIL ADDRESS
 0698 CD06E3                        CALL        DO_PUTOBJ

                          *     INCREMENT QUEUE_TAIL
 069B 3A73CC                        LD          A,[QUEUE_TAIL]
 069E 3C                            INC         A

                          *     IF QUEUE_TAIL = QUEUE_SIZE THEN
 069F 2173CA                        LD          HL,QUEUE_SIZE
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  06A2 BE           2121                CP          [HL]
  06A3 200E         2122                JR          NZ,WRTR_ELSE
                    2123
                    2124 *        QUEUE_TAIL := 0
  06A5 3E00         2125                LD          A,0
  06A7 3273CC       2126                LD          [QUEUE_TAIL],A
                    2127
                    2128 *        TAIL_ADDRESS := BUFFER
  06AA 2A7301       2129                LD          HL,[BUFFER]
  06AD 2273CF       2130                LD          [TAIL_ADDRESS],HL
  06B0 E1           2131                POP         HL              ;RESTORE STACK POINTER
                    2132
  06B1 1807         2133                JR          WRTR_END_IF
                    2134 *      ELSE
        <06B3>      2135 WRTR_ELSE      EQU         $
                    2136
                    2137 *        STORE NEW QUEUE_TAIL
  06B3 3273CC       2138                LD          [QUEUE_TAIL],A
                    2139
                    2140 *        TAIL_ADDRESS := TAIL_ADDRESS + 3
  06B6 E1           2141                POP         HL
  06B7 2273CF       2142                LD          [TAIL_ADDRESS],HL
                    2143
                    2144 *      END_IF
        <06BA>      2145 WRTR_END_IF    EQU         $
                    2146
  06BA 18C6         2147                JR          WRTR_WHILE
                    2148 *    END_WHILE
        <06BC>      2149 WRTR_END_WHILE EQU         $
                    2150
                    2151 * RESTORE DEFERAL FLAG
  06BC F1           2152                POP         AF
  06BD 3273C6       2153                LD          [DEFER_WRITES],A
                    2154
                    2155 * END WRITER_
  06C0 C9           2156                RET
                    2157
                    2158                GLB         PUTOBJ_
                    2159
                    2160
                    2161         ;EXT PUTSEMI,PUT_MOBILE,PUTOSPRITE,PUT1SPRITE,PUTCOMPLEX
                    2162
                    2163         ;EXT  DEFER_WRITES
                    2164         ;EXT  PARAM_
                    2165          GLB  PUTOBJQ
  06C1 00020002     2166 PUTOBJ_PAR:     DEFW  2,2,1
  06C5 0001
                    2167
                    2168 * PROCEDURE PUT_OBJP (VAR DATA:BUFFER;PARAM:BYTE);
                    2169
                    2170 * THIS IS THE PASCAL ENTRY POINT TO THE PUTOBJ ROUTINE
                    2171
                    2172          PROG
  06C7              2173 PUTOBJQ:
  06C7 0106C1       2174         LD    BC,PUTOBJ_PAR
  06CA 11738A       2175         LD    DE,PARAM_AREA
  06CD     98       2176         CAL       AM
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

   06D0 DD2A73BA     2177          LD    IX,[PARAM_AREA]
   06D4 3A738C       2178          LD    A,[PARAM_AREA+2]
   06D7 47           2179          LD    B,A
                     2180
           <0001>    2181 DEFER             EQU              1
   06D8              2182 PUTOBJ_
   06D8 3A73C6       2183          LD        A,[DEFER_WRITES]  ;CHECK IF DEFERRED WRITE IS DESIRED
   06DB FE01         2184          CP        DEFER
   06DD 2004         2185          JR        NZ,DO_PUTOBJ     ;IF NOT, PROCESS OBJECT
   06DF CD0623       2186          CALL      SET_UP_WRITE     ;IF SO, SET UP FOR DEFERRED WRITE
   06E2 C9           2187          RET
   06E3 DD6601       2188 DO_PUTOBJ LD H,[IX+1]                ;GET ADDRESS OF GRAPHICS FOR OBJ_n
   06E6 DD6E00       2189          LD L,[IX+0]
   06E9 7E           2190          LD A,[HL]                 ;A := OBJ_TYPE
   06EA 4F           2191          LD C,A                    ;SAVE COPY
   06EB E60F         2192          AND 0FH                   ;MASK FOR OBJ_TYPE NUMBER
   06ED CA06FF       2193          JP Z,PUTSEMI              ;0 = SEMI_MOBILE
   06F0 3D           2194          DEC A
   06F1 CA0A87       2195          JP Z,PUT_MOBILE            ;1 = MOBILE
   06F4 3D           2196          DEC A
   06F5 CA08DF       2197          JP Z,PUT0SPRITE           ;2 = SPRITE0
   06F8 3D           2198          DEC A
   06F9 CA0955       2199          JP Z,PUT1SPRITE           ;3 = SPRITE1
   06FC C30EA2       2200          JP PUTCOMPLEX             ;>3 = COMPLEX
                     2201 ;
                     2202 ;        END      ;pname
                     2203  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     ******************** PUT_SEMI ******************************************
                     ;DESCRIPTION:     PUTS SEMI_MOBILE OBJECTS ON SCREEN
                     ;
                     ;INPUT:           IX = ADDRESS OF OBJECT TO BE PROCESSED
                     ;                 HL = ADDRESS OF OBJECT'S GRAPHICS TABLES IN ROM
                     ***************************************************************************

                                 GLB            PUTSEMI


06FF DD5603          PUTSEMI:  LD D,[IX+3]           ;GET ADDRESS OF STATUS
0702 DD5E02                    LD E,[IX+2]
0705 D5                        PUSH DE               ;                    AND PUT INTO IY
0706 FDE1                      POP IY
0708 FD5602                    LD D,[IY+2]           ;GET X_LOCATION
070B FD5E01                    LD E,[IY+1]
070E CD07E8                    CALL    PX_TO_PTRN_POS

0711 4B                        LD C,E                ;C := PATTERN PLANE COL.
0712 FD5604                    LD D,[IY+4]           ;GET Y_LOCATION
0715 FD5E03                    LD E,[IY+3]
0718 CD07E8                    CALL    PX_TO_PTRN_POS

071B 43                        LD B,E                ;B := PATTERN PLANE ROW
071C FD5E00                    LD E,[IY+0]           ;GET FRAME NUMBER
                     ;
                     ; HL = GRAPHICS_n, IX = OBJ_n, IY = STATUS_n, C = COL., B = ROW, E = FRAME
                     ;
071F 1600                      LD D,0                ;DE HAS FRAME NUMBER
0721 19                        ADD HL,DE
0722 19                        ADD HL,DE             ;2*FRAME NUMBER + ADDR OF GRAPHICS_n
0723 1E05                      LD E,5                ;FRAME POINTER OFFSET
0725 19                        ADD HL,DE             ;HL NOW POINTS TO LOCATION HOLDING ADDRESS
                                                     ;OF FRAME
0726 5E                        LD E,[HL]             ;GET ADDRESS INTO DE
0727 23                        INC HL
0728 56                        LD D,[HL]
0729 EB                        EX DE,HL              ;HL := ADDRESS OF FRAME
072A C5                        PUSH BC
072B D1                        POP DE                ;DE := Y_PAT_POS & X_PAT_POS
072C 4E                        LD C,[HL]             ;C := X_EXTENT
072D 23                        INC HL
072E 46                        LD B,[HL]             ;B := Y_EXTENT
072F 23                        INC HL                ;HL POINTS TO FIRST NAME IN LIST
                     ;
                     ; TEST TO SEE IF OLD_SCREEN IS TO BE SAVED
                     ;
0730 DD7E05                    LD A,[IX+5]           ;GET HIGH BYTE OF OLD_SCREEN ADDRESS
0733 CB7F                      BIT 7,A               ;TEST BIT 15 OF OLD_SCREEN ADDRESS
0735 2804                      JR Z,S_OLD_SCRN
                     ;
0737 CD080B                    CALL PUTFRAME
073A C9                        RET

073B                 S_OLD_SCRN
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

0738 C5                         PUSH BC          ;SAVE REGS
073C D5                         PUSH DE
073D E5                         PUSH HL

073E FE70                               CP               70H
0740 2802                               JR               Z,EQUAL_TO
0742 3807                               JR               C,ELSE_1

                          ;         IF [.A,GE,70H]         ;THEN OLD_SCREEN IN CPU RAM
0744                      EQUAL_TO
0744 67                             LD H,A
0745 DD6E04                         LD L,[IX+4]              ;HL := OLD_SCREEN ADDRESS
0748 7E                             LD A,[HL]

0749 1835                               JR               END_IF_1
074B                      ELSE_1

                          ;         ELSE                   ;OLD_SCREEN IN VRAM
074B 2A8006                         LD HL,[WORK_BUFFER]      ;GET ADDRESS OF FREE BUFFER SPACE
074E DD5605                         LD D,[IX+5]              ;DE := OLD_SCREEN ADDRESS
0751 DD5E04                         LD E,[IX+4]
0754 E5                             PUSH HL                  ;SAVE 2 COPIES FREE BUFFER ADDR
0755 D5                             PUSH DE                  ;SAVE OLD_SCREEN ADDR
0756 E5                             PUSH HL
0757 010004                         LD BC,4                  ;READ 4 BYTES [X,Y_PAT_POSs, X,Y_EXTETNs]
075A CD1D3E                         CALL VRAM_READ
075D E1                             POP HL                   ;HL := FREE BUFFER ADDR
075E 7E                             LD  A,[HL]
075F FE80                           CP  80H
0761 2003                           JR  NZ,GET_OLD
0763 D1                             POP  DE
0764 1819                           JR  SKIP_OLD
0766 23                   GET_OLD   INC HL
0767 23                             INC HL
0768 46                             LD B,[HL]                ;B := X_EXTENT OF OLD_SCREEN
0769 23                             INC HL
076A 5E                             LD E,[HL]                ;E := Y_EXTENT
076B 1600                           LD D,0
076D 23                             INC HL
076E EB                             EX DE,HL                 ;MULTIPLY X_EXTENT*Y_EXTENT IN HL

076F 1801                               JR               M_XY+1
0771 29                   M_XY:         ADD              HL,HL
0772 10FD                               DJNZ             M_XY

                          ;         DJNZ M.XY
0774 E5                             PUSH HL
0775 C1                             POP BC                   ;BC := NUMBER OF BYTES TO READ
0776 EB                             EX DE,HL                 ;HL := FREE BUFF ADDR + 4
0777 D1                             POP DE                   ;DE := OLD_SCREEN ADDR.
0778 13                             INC DE
0779 13                             INC DE
077A 13                             INC DE
077B 13                             INC DE
077C CD1D3E                         CALL VRAM_READ            ;READ SAVED NAMES FOR BACKGROUND
077F E1                   SKIP_OLD  POP HL                   ;HL := FREE BUFF ADDR.
                          ;         ENDIF
```

```
LOCATION OBJECT CODE      SOURCE LINE

0780                 END_IF_1

0780 7E                       LD A,[HL]              ;A := X_PAT_POS


0781 FE80                           CP               80H
0783 280F                           JR               Z,END_IF_2

                     ;        IF [.A,NE,80H]         ;THEN THERE IS AN OLD_SCREEN
0785 5E                         LD E,[HL]              ;E := X_PAT_POS
0786 23                         INC HL
0787 56                         LD D,[HL]              ;D := Y_PAT_POS
0788 23                         INC HL
0789 4E                         LD C,[HL]              ;C := X_EXTENT
078A 23                         INC HL
078B 46                         LD B,[HL]              ;B := Y_EXTENT
078C 23                         INC HL                 ;HL POINTS TO FIRST NAME IN LIST

078D DDE5                           PUSH             IX              ;SAVE OBJECT POINTER

078F CD0808                     CALL PUTFRAME          ;RESTORE OLD_SCREEN TO DISPLAY

0792 DDE1                           POP              IX              ;RESTORE OBJECT POINTER

0794                 END_IF_2

                     ;        ENDIF
0794 E1              SV1:     POP HL                 ;HL := ADDR OF FIRST NAME IN FRAME
0795 D1                       POP DE                 ;DE := Y,X_PAT_POSs
0796 C1                       POP BC                 ;BC := Y,X_EXTENTs
0797 C5                       PUSH BC
0798 D5                       PUSH DE
0799 E5                       PUSH HL
079A DD6605                   LD H,[IX+5]            ;HL := OLD_SCREEN ADDRESS
079D DD6E04                   LD L,[IX+4]

07A0 3E70                           LD               A,70H
07A2 BC                             CP               H
07A3 3803                           JR               C,END_IF_3

                     ;        IF [.H,LT,70H]         ;THE OLD_SCREEN NOW IN FREE BUFFER
07A5 2A8006                     LD HL,[WORK_BUFFER]    ;THEREFORE, MOVE BACKGROUND TO BUFFER

07A8                 END_IF_3

                     ;        ENDIF
07A8 73                       LD [HL],E              ;OLD_SCREEN +   0 := X_PAT_POS
07A9 23                       INC HL
07AA 72                       LD [HL],D              ;      "        1 := Y_PAT_POS
07AB 23                       INC HL
07AC 71                       LD [HL],C              ;      "        2 := X_EXTENT
07AD 23                       INC HL
07AE 70                       LD [HL],B              ;      "        3 := Y_EXTENT
07AF 23                       INC HL                 ;HL := ADDRESS TO STORE NAMES

07?                                 POS              OBJ     INIT
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     2376
   07B2 CD0898       2377         CALL GET_BKGRND
                     2378
   07B5 DDE1         2379                 POP             IX              ;RESTORE OBJECT POINTER
                     2380
   07B7 E1           2381         POP HL                  ;WHERE NAMES ARE IN CPU RAM
   07B8 D1           2382         POP DE                  ;WHERE TO MOVE THEM TO IN VRAM [NAME TABLE]
   07B9 C1           2383         POP BC                  ;HOW MANY TO MOVE
                     2384
   07BA DDE5         2385                 PUSH            IX              ;SAVE OBJECT POINTER
                     2386
   07BC CD0808       2387         CALL PUTFRAME
                     2388
   07BF DDE1         2389                 POP             IX              ;RESTORE OBJECT POINTER
                     2390
   07C1 DD5605       2391         LD D,[IX+5]             ;SEE IF SAVED BACKGROUND TO BE MOVED TO VRAM
                     2392
   07C4 3E70         2393 SV2:            LD              A,70H
   07C6 BA           2394                 CP              D
   07C7 281E         2395                 JR              Z,END_IF_4
   07C9 381C         2396                 JR              C,END_IF_4
                     2397
                     2398 ;SV2::   IF [.D,LT,70H]
   07CB DD5E04       2399           LD E,[IX+4]           ;DE := OLD_SCREEN ADDR
   07CE D9           2400           EXX                   ;USE 'REG FOR CALCULATION
   07CF 2A8006       2401           LD HL,[WORK_BUFFER]   ;WHERE NEXT OLD_SCREEN DATA IS
   07D2 E5           2402           PUSH HL
   07D3 23           2403           INC HL
   07D4 23           2404           INC HL
   07D5 5E           2405           LD E,[HL]             ;E := X_EXTENT
   07D6 1600         2406           LD D,0
   07D8 23           2407           INC HL
   07D9 46           2408           LD B,[HL]             ;B := Y_EXTENT
   07DA EB           2409           EX DE,HL              ;HL := X_EXTENT
   07DB 1801         2410           JR M_XY2+1
   07DD 29           2411 M_XY2:    ADD HL,HL             ;HL := X_EXTENT*Y_EXTENT
   07DE 10FD         2412           DJNZ M_XY2
   07E0 E5           2413           PUSH HL
   07E1 D9           2414           EXX
   07E2 C1           2415           POP BC                ;BC := NUMBER OF BYTES TO WRITE
   07E3 E1           2416           POP HL                ;HL := FREE BUFFER ADDRESS
   07E4 CD1D01       2417           CALL VRAM_WRITE
                     2418
   07E7              2419 END_IF_4
                     2420
                     2421 ;         ENDIF
   07E7 C9           2422           RET
                     2423
                     2424 ******************* PX.TO.PTRN.POS ****************************************
                     2425 ;DESCRIPTION:     DIVIDES REG DE BY 8, IF SIGNED RESULT > 127 THEN E := MAX SIGNED
                     2426 ;                 POSITIVE NUMBER. IF RESULT < -128, THEN E := MIN NEGATIVE NUM
                     2427 ;INPUT:           DE = 16 BIT SIGNED NUMBER
                     2428 ;OUTPUT:          DE/8 < -128       E = -128
                     2429 ;          -128 <= DE/8 <=+127      E = DE/8
                     2430 ;          +127 <  DE/8             E = +127
                     2431 ***************************************************************************
                     2432 ;
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     2433
                     2434                 GLB              PX_TO_PTRN_POS
  07E8               2435 PX_TO_PTRN_POS
                     2436
  07E8 E5            2437         PUSH HL                  ;HL USED TO TEST MAGNITUDE
  07E9 CB2A          2438         SRA D                    ;16 BIT SHIFT LEFT
  07EB CB1B          2439         RR E
  07ED CB2A          2440         SRA D
  07EF CB1B          2441         RR E
  07F1 CB2A          2442         SRA D                    ;                X3
  07F3 CB1B          2443         RR E
  07F5 CB7A          2444         BIT 7,D                  ;IS RESULT NEGATIVE
  07F7 2009          2445         JR NZ,NEGTV
  07F9 21FF80        2446         LD HL,OFF80H             ;SEE IF RESULT < 127
  07FC 19            2447         ADD HL,DE
  07FD E1            2448         POP HL
  07FE D0            2449         RET NC
  07FF 1E7F          2450         LD E,7FH                 ;IF > 128, THEN E := MAX SIGNED + NUM.
  0801 C9            2451         RET
                     2452 ;XXXXX
  0802 210080        2453 NEGTV:  LD HL,080H               ;IS RESULT > -128
                     2454 ;XXXXX
  0805 19            2455         ADD HL,DE
  0806 E1            2456         POP HL
  0807 D8            2457         RET C
  0808 1E80          2458         LD E,80H                 ;IF < -128, THE E := MIN SIGNED - NUM.
  080A C9            2459         RET
                     2460 ;
                     2461 ************************** PUT FRAME ****************************************
                     2462 ;DESCRIPTION:    THE NAMES WHICH CONSTITUTE A FRAME ARE MOVED TO THE NAME TABLE
                     2463 ;               IN VRAM.  THE UPPER LEFT HAND CORNER OF THE FRAME IS POSITIONED
                     2464 ;               AT X_PAT_POS, Y_PAT_POS.
                     2465 ;INPUT:          HL = ADDRESS OF LIST OF NAMES [IN CPU RAM]
                     2466 ;                E = X_PAT_POS
                     2467 ;                D = Y_PAT_POS
                     2468 ;                C = X_EXTENT
                     2469 ;                B = Y_EXTENT
                     2470 *****************************************************************************
                     2471 ;
                     2472                 GLB              PUTFRAME
  080B C5            2473 PUTFRAME:  PUSH BC               ;COPY PARAMETERS INTO PRIMED REGISTERS
  080C D5            2474         PUSH DE
  080D E5            2475         PUSH HL                  ;AND FRAME ADDRESS INTO DE'
  080E D9            2476         EXX
  080F E1            2477         POP HL
  0810 D1            2478         POP DE
  0811 C1            2479         POP BC
  0812 CD08C0        2480         CALL CALC_OFFSET
  0815 D9            2481         EXX
                     2482 ;XXXXX
                     2483 ; TEST FOR THE FOLLOWING CONDITION: (X_PAT_POS SLE 32) AND (X_PAT_POS + X_EXTENT
                     2484 ;                                     SGT 0)
  0816 7B            2485 PF1:    LD A,E                   ;IS X_PAT_POS < 0?
  0817 CB7F          2486         BIT 7,A
  0819 2003          2487         JR NZ,XP_NEG
  081B FE20          2488         CP 32                    ;IS X_PAT_POS < 32?
  81D F                 ,         RET                          01,
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  081E 81            XP_NEG: ADD A,C              ;A := X_PAT_POS + X_EXTENT
  081F CB7F                  BIT 7,A              ;IS A NEG?
  0821 C0                    RET NZ               ;YES, RETURN
  0822 B7                    OR A                 ;A = 0?
  0823 C8                    RET Z                ;RETURN IF 0
                     ;XXXXX
                     ;X.IN.BOUNDS::

  0824               X_IN_BOUNDS

                     ;          IF [.E,IS,MINUS]        ;IF X_PAT_POS < 0 , FRAME BLEEDING ON FROM LEFT

  0824 CB7B                         BIT           7,E
  0826 2820                         JR            Z,ELSE_8

  0828 79                     LD A,C                ;CALCULATE AMOUNT OF FRAME ON SCREEN
  0829 83                     ADD A,E               ;A := X_EXTENT + X_PAT_POS
  082A D5                     PUSH DE
                     ;XXXXX
  082B FE21                   CP 33                 ;IF NUMBER OF NAMES > 32
  082D 3802                   JR C,LT33
  082F 3E20                   LD A,32               ;THEN NUMBER OF NAMES := 32
  0831 5F            LT33:    LD E,A                ;NUMBER OF NAMES ON SCREEN EACH ROW
                     ;XXXXX
  0832 1600                   LD D,0
  0834 D5                     PUSH DE               ;GET COUNT INTO IY
  0835 FDE1                   POP IY
  0837 D1                     POP DE                ;RESTORE DE
  0838 7B                     LD A,E                ;A := X_PAT_POS
  0839 D9                     EXX                   ;NOW ADJUST STARTING POINTS IN FRAME LIST AND
                                                    ;NAME TABLE
  083A C5                     PUSH BC               ;SAVE X AND Y EXTENT
  083B ED44                   NEG                   ;2'S COMPLIMENT OF X_PAT_POS
  083D 4F                     LD C,A
  083E 0600                   LD B,0
  0840 09                     ADD HL,BC             ;ADD DISPLACEMENT TO FRAME POINTER
  0841 EB                     EX DE,HL
  0842 09                     ADD HL,BC             ;ADD DISPLACEMENT TO NAME TABLE POINTER
  0843 EB                     EX DE,HL
  0844 C1                     POP BC
  0845 D9                     EXX

  0846 181C                         JR            END_IF_8

                     ;          ELSE

  0848               ELSE_8

  0848 7B            PF2:     LD A,E                ;IS X_PAT_POS + X_EXTENT > 31
  0849 81                     ADD A,C
                     ;        IF [.A,GT,31]

  084A FE1F                         CP            31
  084C 280F                         JR            Z,ELSE_9
  084E 3800                         JR            C,ELSE_9

  0850 3E20                    LD A,32              ;SUBTRACT X_PAT_POS FROM 31
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0852 93                             SUB E
  0853 D5                             PUSH DE
  0854 5F                             LD E,A                ;GET THIS NUMBER INTO IY
  0855 1600                           LD D,0
  0857 D5                             PUSH DE
  0858 FDE1                           POP IY
  085A D1                             POP DE
  085B 1807                               JR            END_IF_9
                         ;        ELSE                  ;BOTH ENDS OF FRAME WITHIN PATTERN PLANE
  085D                   ELSE_9
  085D C5                PF3:         PUSH BC
  085E 0600                           LD B,0
  0860 C5                             PUSH BC
  0861 FDE1                           POP IY
  0863 C1                             POP BC
                         ;        ENDIF
  0864                   END_IF_9
                         ;       ENDIF
  0864                   END_IF_8
  0864 1E00                      LD E,0
                         ;       REPEAT                 ;Y_EXTENT-1 TIMES
  0866                   RPT_1
  0866 7A                PF4:    LD A,D                 ;GET Y_PAT_POS
  0867 83                        ADD A,E                ;ADD Y
                         ;       IF [.A,IS,PLUS]
  0868 CB7F                           BIT          7,A
  086A 2019                           JR           NZ,END_IF_10
                         ;         IF [.A,LE,23]          ;IS 0<=Y_PAT_POS + Y <=23
  086C FE18                           CP           24
  086E 3015                           JR           NC,END_IF_10
  0870 C5                             PUSH BC
  0871 D5                             PUSH DE
  0872 D9                             EXX
  0873 C5                             PUSH BC
  0874 D5                             PUSH DE
  0875 E5                             PUSH HL
  0876 FDE5                           PUSH IY
  0878 3E02                           LD A,2               ;CODE FOR PATTERN NAME TABLE ADDED 4/20
  087A CD1C27                         CALL PUT_VRAM_
  087D FDE1                           POP IY
  087F E1                             POP HL
  0880 D1                             POP DE
  0881 C1                             POP BC
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

  0882 D9                              EXX
  0883 D1                              POP DE
  0884 C1                              POP BC
                          ;          ENDIF
                          ;        ENDIF

  0885                    END_IF_10

  0885 D9                          EXX
  0886 C5                          PUSH BC
  0887 0600                        LD B,0                 ;INCREMENT POINTER INTO FRAME BY X_EXTENT
  0889 09                          ADD HL,BC
  088A EB                          EX DE,HL
  088B 010020                      LD BC,32               ;INCREMENT OFFSET BY 32
  088E 09                          ADD HL,BC
  088F EB                          EX DE,HL
  0890 C1                          POP BC
  0891 D9                          EXX
  0892 1C                          INC E
                          ;        UNTIL [.E,EQ,.B]       ;UNTIL Y=Y_EXTENT

  0893 7B                              LD             A,E
  0894 B8                              CP             B
  0895 20CF                            JR             NZ,RPT_1

  0897 C9                          RET
                          ;
                          ;        .COMMENT )
                          *************** GET.BKGRND ****************************************************
                          ;DESCRIPTION:     THIS ROUTINE GETS THE NAMES FROM THE NAME TABLE WHICH CONSTITUTE
                          ;                 THE BACKGROUND ON WHICH AN OBJECT IS TO BE MOVED
                          ;INPUT:            HL = LOCATION IN CPU RAM TO WHICH THE NAMES ARE MOVED
                          ;                  D = Y_PAT_POS [TOP ROW OF PATTERN]
                          ;                  E = X_PAT_POS [LEFT HAND COLUMN]
                          ;                  B = Y_EXTENT OF PATTERN
                          ;                  C = X_EXTENT OF PATTERN
                          ******************************************************************************
                          ;
                                               GLB            GET_BKGRND
  0898                    GET_BKGRND:
  0898 CD08C0                      CALL CALC_OFFSET       ;OFFSET INTO NAME TABLE OF POSITION OF UPPER LEFT
  089B C5                          PUSH BC                ;                               HAND PATTERN
  089C 0600                        LD B,0                 ;GET X_EXTENT INTO IY
  089E C5                          PUSH BC                ;   NUMBER OF NAMES PER ROW
  089F FDE1                        POP IY
  08A1 C1                          POP BC
                          ;        REPEAT                 ;Y_EXTENT-1 TIMES

  08A2                    RPT_2

  08A2 C5                           PUSH BC
  08A3 D5                           PUSH DE
  08A4 E5                           PUSH HL
  08A5 FDE5                         PUSH IY
  08A7 3E02                         LD   A,2              ;TABLE CODE FOR PATTERN NAME TABLE
  08A9 CD1BA3                       CALL GET_VRAM_
  08AC FDE1                         POP IY
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                              * THIS MODULE CONTAINS CODE FOR THE PUT1SPRITE AND PUT0SPRITE
                              * ROUTINES. THESE ROUTINES TURN OUT TO BE ESSENTIALLY THE SAME CODE
                              * WITH TWO SLIGHTLY DIFFERENT ENTRY POINTS


                              * IT IS CALLED WITH THE ADDRESS OF THE SPRITE OBJECT IN THE IX REGISTER.

                              * THE FORMAT FOR SPRITE OBJECTS IS

                              * SPRITE_OBJECT = RECORD
                              *   GRAPHICS:^SPRITE_GRAPHICS
                              *   STATUS:^SPRITE_STATUS
                              *   SPRITE_INDEX:BYTE               (SPRITE_NAME_TABLE INDEX OF THIS SPRITE)
                              * END SPRITE_OBJECT

                              * SPRITE_GRAPHICS = RECORD
                              *   OBJECT_TYPE:BYTE
                              *   FIRST_GEN_NAME:BYTE             (NAME OF FIRST SPRITE GENERATOR)
                              *   PTRN_POINTER:^PATTERN_GENERATOR (POINTER TO ROM'ED GENERATORS)
                              *   NUMGEN:BYTE                     (NUMBER OF ROM'ED GENERATORS)
                              *   FRAME_TABLE_PTR:^ARRAY[0..nn] OF FRAME (TABLE OF ANIMATION FRAMES)
                              * END SPRITE_ROM_GRAPHICS

                              * SPRITE_STATUS  = RECORD
                              *   FRAME:BYTE              (CURRENT ANIMATION FRAME)
                              *   X_LOCATION:INTEGER
                              *   Y_LOCATION:INTEGER
                              *   NEXT_GEN:BYTE           (INDEX OF FREE SPACE IN GENERATOR TABLE)
                              * END SPRITE_STATUS

                              * FRAME = RECORD
                              *   COLOR:BYTE              (SPRITE'S COLOR FOR THIS FRAME)
                              *   SHAPE:BYTE              (THIS FRAME'S OFFSET FROM NAME FROM FIRST_GEN_NAME)
                              * END FRAME

                              * SPRITE = RECORD
                              *   Y:BYTE
                              *   X:BYTE
                              *   NAME:BYTE
                              *   COLOR_AND_TAG:BYTE
                              * END SPRITE

                              ***********************************************************************

                              **************************** DICTIONARY *****************************

                              ;                   EXT           WORK_BUFFER
                              * WORK_BUFFER IS A POINTER IN CARTRIDGE ROM, LOCATED AT 8006H, TO THE
                              * FREE BUFFER AREA TO BE USED BY THE GRAPHICS ROUTINES.

<000F>                        SPRITE_PTR      EQU           IY
                              * SPRITE_PTR IS A POINTER TO THE NEW SPRITE NAME TABLE ENTRY BEING
                              * BUILT BY THIS ROUTINE.

<0                            SPR             EQU           IX
                              * THIS SPRITE IS A POINTER TO THE SPRITE OBJECT BEING PUT
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

           
   <0000>       GRAPHICS        EQU             0
   <0002>       STATUS          EQU             2
   <0004>       SPRITE_INDEX    EQU             4
                * FIELD OFFSETS FOR SPRITE_OBJECT RECORDS

   <0000>       OBJECT_TYPE     EQU             0
   <0001>       FIRST_GEN_NAME  EQU             1
   <0002>       PTRN_POINTER    EQU             2
   <0004>       NUMGEN          EQU             4
   <0005>       FRAME_TABLE_PTR EQU             5
                * FIELD OFFSETS FOR SPRITE GRAPHICS RECORDS

   <0000>       FRAME           EQU             0
   <0001>       X_LOCATION      EQU             1
   <0003>       Y_LOCATION      EQU             3
   <0005>       NEXT_GEN        EQU             5
                * FIELD OFFSETS FOR SPRITE_STATUS RECORDS

   <0000>       COLOR           EQU             0
   <0001>       SHAPE           EQU             1
                * FIELD OFFSETS FOR FRAME RECORDS

   <0000>       Y               EQU             0
   <0001>       X               EQU             1
   <0002>       NAME            EQU             2
   <0003>       COLOR_AND_TAG   EQU             3
                * FIELD OFFSETS FOR SPRITE RECORDS
                *********************** EXTERNAL PROCEDURES ***********************

                ; EXT PUT_VRAM,GET_VRAM
                * EXTERNAL PROCEDURE PUT_VRAMP (TABLE_CODE:BYTE; START_INDEX,SLICE:BYTE;
                *                               VAR DATA:BUFFER;ITEM_COUNT:INTEGER);

                * EXTERNAL PROCEDURE GET_VRAMP (TABLE_CODE:BYTE; START_INDEX,SLICE:BYTE;
                *                               VAR DATA:BUFFER;ITEM_COUNT:INTEGER);

                * PUT_VRAM SENDS A BLOCK OF DATA TO THE TABLE SPECIFIED BY TABLE_CODE.
                * THE SLICE, START_INDEX, AND ITEM_COUNT ARE TABLE DEPENDANT. GET_VRAM
                * DOES THE INVERSE OPERATION.

                *    - TABLE_CODE IS PASSED IN A
                *    - START_INDEX,SLICE IN DE
                *    - DATA BUFFER ADDRESS IN HL
                *    - BYTE COUNT PASSED IN IY

                ************************ PROCEDURE BODY ************************

                                PROG
                 GLB PUT0SPRITE,PUT1SPRITE

                * BEGIN PUT0SPRITE
   <080F>       PUT0SPRITE      EQU             $

                *    SPRITE_PTR := WORK_BUFFER
080F FD2A8006                   LD              SPRITE_PTR,[WORK_BUFFER]

```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     2861 *   WITH THIS_SPRITE^,SPRITE_PTR^ DO
                     2862
                     2863 *     IF (STATUS^.X_LOCATION > -8) AND (STATUS^.X_LOCATION < 256) AND
                     2864 *        (STATUS^.Y_LOCATION > -8) AND (STATUS^.Y_LOCATION < 192) THEN
  08E3 DD6E02        2865              LD          L,[THIS_SPRITE+STATUS]
  08E6 DD6603        2866              LD          H,[THIS_SPRITE+STATUS+1]
  08E9 110001        2867              LD          DE,X_LOCATION
  08EC 19            2868              ADD         HL,DE          ; [HL] = X_LOCATION
  08ED 4E            2869              LD          C,[HL]
  08EE 23            2870              INC         HL
  08EF 46            2871              LD          B,[HL]         ; BC = X_LOCATION
  08F0 78            2872              LD          A,B            ; COMPARE BC WITH -8
  08F1 FE00          2873              CP          0
  08F3 2808          2874              JR          Z,OK__1
  08F5 FEFF          2875              CP          -1
  08F7 C20A54        2876              JP          NZ,DONT_PUT
  08FA 79            2877              LD          A,C
  08FB FEF9          2878              CP          -7
  08FD FA0A54        2879              JP          M,DONT_PUT
  0900               2880 OK__1
  0900 23            2881              INC         HL             ; [HL] = Y_LOCATION
  0901 4E            2882              LD          C,[HL]
  0902 23            2883              INC         HL
  0903 46            2884              LD          B,[HL]         ; BC = Y_LOCATION
  0904 78            2885              LD          A,B            ; COMPARE BC WITH -8
  0905 FE00          2886              CP          0
  0907 2808          2887              JR          Z,OK__2
  0909 FEFF          2888              CP          -1
  090B C20A54        2889              JP          NZ,DONT_PUT
  090E 79            2890              LD          A,C
  090F FEF9          2891              CP          -7
  0911 FA0A54        2892              JP          M,DONT_PUT
  0914               2893 OK__2
                     2894
                     2895 *      IF STATUS^.X_LOCATION < 0 THEN
  0914 2B            2896              DEC         HL
  0915 2B            2897              DEC         HL             ; [HL] = HI(X_LOCATION)
  0916 7E            2898              LD          A,[HL]         ; COMPARE WITH 0
  0917 FE00          2899              CP          0
  0919 CA09CA        2900              JP          Z,CONTINUE
                     2901
                     2902 *        X := BYTE(STATUS^.X_LOCATION) + 8
  091C 2B            2903              DEC         HL             ;[HL] = ^X_LOCATION
  091D 4E            2904              LD          C,[HL]
  091E 23            2905              INC         HL
  091F 46            2906              LD          B,[HL]
  0920 210008        2907              LD          HL,8
  0923 09            2908              ADD         HL,BC
  0924 7D            2909              LD          A,L
  0925 FD7701        2910              LD          [SPRITE_PTR+X],A
                     2911
                     2912 *        COLOR_AND_TAG := GRAPHICS^.FRAME_TABLE[STATUS^.FRAME].COLOR OR 80H
  0928 DD6E00        2913              LD          L,[THIS_SPRITE+GRAPHICS]
  092B DD6601        2914              LD          H,[THIS_SPRITE+GRAPHICS+1]
  092E 110005        2915              LD          DE,FRAME_TABLE_PTR
  0931 19            2916              ADD         HL,DE          ;[HL] = FRAME_TABLE_PTR
  0932 ..            2917              .X          ..
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0933 1A           2918                LD          A,[DE]
  0934 6F           2919                LD          L,A
  0935 13           2920                INC         DE
  0936 1A           2921                LD          A,[DE]
  0937 67           2922                LD          H,A              ;[HL] = FRAME_TABLE_PTR^
  0938 E5           2923                PUSH        HL
  0939 DD6E02       2924                LD          L,[THIS_SPRITE+STATUS]
  093C DD6603       2925                LD          H,[THIS_SPRITE+STATUS+1]
  093F 110000       2926                LD          DE,FRAME
  0942 19           2927                ADD         HL,DE            ;[HL] = FRAME
  0943 7E           2928                LD          A,[HL]           ;CALCULATE OFFSET OF
  0944 CB27         2929                SLA         A                ;COLOR ENTRY
  0946 010000       2930                LD          BC,0
  0949 4F           2931                LD          C,A
  094A E1           2932                POP         HL
  094B 09           2933                ADD         HL,BC            ;[HL] = COLOR
  094C 7E           2934                LD          A,[HL]           ;OR IN 80H
  094D F680         2935                OR          80H
  094F FD7703       2936                LD          [SPRITE_PTR+COLOR_AND_TAG],A
                    2937
  0952 C30A00       2938                JP          PUT_Y_AND_NAME
                    2939 *       ELSE
                    2940 ********* CONTINUE BELOW
                    2941
                    2942
                    2943 * BEGIN PUT1SPRITE
          <0955>    2944 PUT1SPRITE     EQU         $
                    2945
                    2946 *    SPRITE_PTR := WORK_BUFFER
  0955 FD2A8006     2947                LD          SPRITE_PTR,[WORK_BUFFER]
                    2948
                    2949 *    WITH THIS_SPRITE^,SPRITE_PTR^ DO
                    2950
                    2951 *       IF (STATUS^.X_LOCATION > -32) AND (STATUS^.X_LOCATION < 256) AND
                    2952 *          (STATUS^.Y_LOCATION > -32) AND (STATUS^.Y_LOCATION < 192) THEN
  0959 DD6E02       2953                LD          L,[THIS_SPRITE+STATUS]
  095C DD6603       2954                LD          H,[THIS_SPRITE+STATUS+1]
  095F 110001       2955                LD          DE,X_LOCATION
  0962 19           2956                ADD         HL,DE            ; [HL] = X_LOCATION
  0963 4E           2957                LD          C,[HL]
  0964 23           2958                INC         HL
  0965 46           2959                LD          B,[HL]           ; BC = X_LOCATION
  0966 78           2960                LD          A,B              ; COMPARE BC WITH -32
  0967 FE00         2961                CP          0
  0969 2808         2962                JR          Z,OK__3
  096B FEFF         2963                CP          -1
  096D C20A54       2964                JP          NZ,DONT_PUT
  0970 79           2965                LD          A,C
  0971 FEE1         2966                CP          -31
  0973 FA0A54       2967                JP          M,DONT_PUT
  0976              2968 OK__3
  0976 23           2969                INC         HL               ; [HL] = Y_LOCATION
  0977 4E           2970                LD          C,[HL]
  0978 23           2971                INC         HL
  0979 46           2972                LD          B,[HL]           ; BC = Y_LOCATION
  097A 78           2973                LD          A,B              ; COMPARE BC WITH -32
  097B FE00         2974                CP          0
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  097D 2808                      JR         Z,OK__4
  097F FEFF                      CP         -1
  0981 C20A54                    JP         NZ,DONT_PUT
  0984 79                        LD         A,C
  0985 FEE1                      CP         -31
  0987 FA0A54                    JP         M,DONT_PUT
  098A                      OK__4
                            *        IF STATUS^.X_LOCATION < 0 THEN
  098A 2B                        DEC        HL
  098B 2B                        DEC        HL             ; [HL] = HI(X_LOCATION)
  098C 7E                        LD         A,[HL]         ; COMPARE WITH 0
  098D FE00                      CP         0
  098F CA09CA                    JP         Z,CONTINUE
                            *        X := BYTE(STATUS^.X_LOCATION) + 32
  0992 2B                        DEC        HL             ;[HL] = ^X_LOCATION
  0993 4E                        LD         C,[HL]
  0994 23                        INC        HL
  0995 46                        LD         B,[HL]
  0996 210020                    LD         HL,32
  0999 09                        ADD        HL,BC
  099A 7D                        LD         A,L
  099B FD7701                    LD         [SPRITE_PTR+X],A
                            *        COLOR_AND_TAG := GRAPHICS^.FRAME_TABLE[STATUS^.FRAME].COLOR OR 80H
  099E DD6E00                    LD         L,[THIS_SPRITE+GRAPHICS]
  09A1 DD6601                    LD         H,[THIS_SPRITE+GRAPHICS+1]
  09A4 110005                    LD         DE,FRAME_TABLE_PTR
  09A7 19                        ADD        HL,DE          ;[HL] = FRAME_TABLE_PTR
  09A8 EB                        EX         DE,HL
  09A9 1A                        LD         A,[DE]
  09AA 6F                        LD         L,A
  09AB 13                        INC        DE
  09AC 1A                        LD         A,[DE]
  09AD 67                        LD         H,A            ;[HL] = FRAME_TABLE_PTR^
  09AE E5                        PUSH       HL
  09AF DD6E02                    LD         L,[THIS_SPRITE+STATUS]
  09B2 DD6603                    LD         H,[THIS_SPRITE+STATUS+1]
  09B5 110000                    LD         DE,FRAME
  09B8 19                        ADD        HL,DE          ;[HL] = FRAME
  09B9 7E                        LD         A,[HL]         ;CALCULATE OFFSET OF
  09BA CB27                      SLA        A              ;COLOR ENTRY
  09BC 010000                    LD         BC,0
  09BF 4F                        LD         C,A
  09C0 E1                        POP        HL
  09C1 09                        ADD        HL,BC          ;[HL] = COLOR
  09C2 7E                        LD         A,[HL]         ;OR IN 80H
  09C3 F680                      OR         80H
  09C5 FD7703                    LD         [SPRITE_PTR+COLOR_AND_TAG],A
  09C8 1836                      JR         PUT_Y_AND_NAME
                            *        ELSE
                            ********** CONTINUE FROM HERE
  09CA                      CONTINUE
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                          *          X := BYTE(STATUS^.X_LOCATION)
  09CA DD6E02                              LD        L,[THIS_SPRITE+STATUS]
  09CD DD6603                              LD        H,[THIS_SPRITE+STATUS+1]
  09D0 110001                              LD        DE,X_LOCATION
  09D3 19                                  ADD       HL,DE          ; [HL] = X_LOCATION
  09D4 7E                                  LD        A,[HL]
  09D5 FD7701                              LD        [SPRITE_PTR+X],A

                          *          COLOR_AND_TAG := GRAPHICS^.FRAME_TABLE[STATUS^.FRAME].COLOR
  09D8 DD6E00                              LD        L,[THIS_SPRITE+GRAPHICS]
  09DB DD6601                              LD        H,[THIS_SPRITE+GRAPHICS+1]
  09DE 110005                              LD        DE,FRAME_TABLE_PTR
  09E1 19                                  ADD       HL,DE          ;[HL] = FRAME_TABLE_PTR
  09E2 EB                                  EX        DE,HL
  09E3 1A                                  LD        A,[DE]
  09E4 6F                                  LD        L,A
  09E5 13                                  INC       DE
  09E6 1A                                  LD        A,[DE]
  09E7 67                                  LD        H,A            ;[HL] = FRAME_TABLE_PTR^
  09E8 E5                                  PUSH      HL
  09E9 DD6E02                              LD        L,[THIS_SPRITE+STATUS]
  09EC DD6603                              LD        H,[THIS_SPRITE+STATUS+1]
  09EF 110000                              LD        DE,FRAME
  09F2 19                                  ADD       HL,DE          ;[HL] = FRAME
  09F3 7E                                  LD        A,[HL]         ;CALCULATE OFFSET OF
  09F4 CB27                                SLA       A              ;COLOR ENTRY
  09F6 010000                              LD        BC,0
  09F9 4F                                  LD        C,A
  09FA E1                                  POP       HL
  09FB 09                                  ADD       HL,BC          ;[HL] = COLOR
  09FC 7E                                  LD        A,[HL]
  09FD FD7703                              LD        [SPRITE_PTR+COLOR_AND_TAG],A

                          *        END_IF
  0A00                         PUT_Y_AND_NAME

                          *        Y := BYTE(STATUS^.Y_LOCATION)
  0A00 DD6E02                              LD        L,[THIS_SPRITE+STATUS]
  0A03 DD6603                              LD        H,[THIS_SPRITE+STATUS+1]
  0A06 110003                              LD        DE,Y_LOCATION
  0A09 19                                  ADD       HL,DE          ; [HL] = Y_LOCATION
  0A0A 7E                                  LD        A,[HL]
  0A0B FD7700                              LD        [SPRITE_PTR+Y],A

                          *        NAME := GRAPHICS^.FRAME_TABLE[STATUS^.FRAME].SHAPE
                          *                + GRAPHICS^.FIRST_GEN_NAME
  0A0E DD6E00                              LD        L,[THIS_SPRITE+GRAPHICS]
  0A11 DD6601                              LD        H,[THIS_SPRITE+GRAPHICS+1]
  0A14 110005                              LD        DE,FRAME_TABLE_PTR
  0A17 19                                  ADD       HL,DE          ;[HL] = FRAME_TABLE_PTR
  0A18 EB                                  EX        DE,HL
  0A19 1A                                  LD        A,[DE]
  0A1A 6F                                  LD        L,A
  0A1B 13                                  INC       DE
  0A1C 1A                                  LD        A,[DE]
  0A1D 67                                  LD        H,A            ;[HL] = FRAME_TABLE_PTR^
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0A1E E5           3089                PUSH        HL
  0A1F DD6E02       3090                LD          L,[THIS_SPRITE+STATUS]
  0A22 DD6603       3091                LD          H,[THIS_SPRITE+STATUS+1]
  0A25 110000       3092                LD          DE,FRAME
  0A28 19           3093                ADD         HL,DE           ;[HL] = FRAME
  0A29 7E           3094                LD          A,[HL]          ;CALCULATE OFFSET OF
  0A2A CB27         3095                SLA         A               ;SHAPE ENTRY
  0A2C 010000       3096                LD          BC,0
  0A2F 4F           3097                LD          C,A
  0A30 E1           3098                POP         HL
  0A31 09           3099                ADD         HL,BC
  0A32 23           3100                INC         HL              ;[HL] = SHAPE
  0A33 7E           3101                LD          A,[HL]
  0A34 DD6E00       3102                LD          L,[THIS_SPRITE+GRAPHICS]
  0A37 DD6601       3103                LD          H,[THIS_SPRITE+GRAPHICS+1]
  0A3A 110001       3104                LD          DE,FIRST_GEN_NAME
  0A3D 19           3105                ADD         HL,DE           ;[HL] = FIRST_GEN_NAME
  0A3E 86           3106                ADD         A,[HL]
  0A3F FD7702       3107                LD          [SPRITE_PTR+NAME],A
                    3108
                    3109 *        PUT_VRAM  (0,THIS_SPRITE^.SPRITE_INDEX,SPRITE_PTR,1)
  0A42 AF           3110                XOR         A
  0A43 1600         3111                LD          D,0
  0A45 DD5E04       3112                LD          E,[THIS_SPRITE+SPRITE_INDEX]
  0A48 FDE5         3113                PUSH        SPRITE_PTR
  0A4A E1           3114                POP         HL
  0A4B FD210001     3115                LD          IY,1            ;COUNT OF ONE ITEM
  0A4F CD1FBE       3116                CALL        PUT_VRAM
                    3117
  0A52 1832         3118                JR          EXIT_PUT_SPR
                    3119 *     ELSE
  0A54              3120 DONT_PUT  ; PUT SPRITE OFF THE SCREEN BY SETTING ITS X AND EARLY CLOCK
                    3121
                    3122 *        GET_VRAM  (0,THIS_SPRITE^.SPRITE_INDEX,SPRITE_PTR,1)
  0A54 FDE5         3123                PUSH        SPRITE_PTR      ; SAVE INDEX REGS.
  0A56 DDE5         3124                PUSH        THIS_SPRITE
  0A58 FDE5         3125                PUSH        SPRITE_PTR
  0A5A FDE5         3126                PUSH        SPRITE_PTR
  0A5C AF           3127                XOR         A
  0A5D 1600         3128                LD          D,0
  0A5F DD5E04       3129                LD          E,[THIS_SPRITE+SPRITE_INDEX]
  0A62 E1           3130                POP         HL
  0A63 FD210001     3131                LD          IY,1            ;COUNT OF ONE ITEM
  0A67 CD1FBB       3132                CALL        GET_VRAM
                    3133
                    3134 *        SPRITE_PTR.X := 0
  0A6A 3E00         3135                LD          A,0
  0A6C FDE1         3136                POP         SPRITE_PTR
  0A6E FD7701       3137                LD          [SPRITE_PTR+X],A
                    3138
                    3139 *        SPRITE_PTR.COLOR_AND_TAG := 80H
  0A71 3E80         3140                LD          A,80H
  0A73 FD7703       3141                LD          [SPRITE_PTR+COLOR_AND_TAG],A
                    3142
                    3143 *        PUT_VRAM  (0,THIS_SPRITE^.SPRITE_INDEX,SPRITE_PTR,1)
  0A76 AF           3144                XOR         A
  0A77 1600         3145                LD
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

   0A79 DDE1        3146                 POP          THIS_SPRITE
   0A7B DD5E04      3147                 LD           E,[THIS_SPRITE+SPRITE_INDEX]
   0A7E E1          3148                 POP          HL
   0A7F FD210001    3149                 LD           IY,1          ;COUNT OF ONE ITEM
   0A83 CD1FBE      3150                 CALL         PUT_VRAM
                    3151
                    3152 *      END_IF
                    3153
                    3154 * END PUT0SPRITE,PUT1SPRITE
   0A86             3155 EXIT_PUT_SPR
   0A86 C9          3156                 RET
                    3157  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                    3159 ;
                    3160 ;*********** MODIFIED VERSION TO RUN ON HP ASSEMBLER **************************
                    3161 ;
                    3162 ;                                                          4/16/82
                    3163 ;                                                          13:50:00
                    3164 ;****************** PUT_MOBILE *********************************************
                    3165 ;
                    3166 ;DESCRIPTION:    THIS PROCEDURE PLACES A MOBILE OBJECT ON THE PATTERN PLANE
                    3167 ;                AT THE X,Y PIXEL LOCATION SPECIFIED IN THAT OBJECT'S RAM STATUS
                    3168 ;                AREA_
                    3169 ;
                    3170 ;                A BUFFER AREA OF 204 BYTES (GRAPHICS MODE II) OR 141 BYTES
                    3171 ;                (GRAPHICS MODE I) IS REQUIRED FOR FORMING THE NEW GENERATORS
                    3172 ;                REPRESENTING THE OBJECT ON IT'S BACKGROUND_  THE PROCEDURE
                    3173 ;                USES RAM STARTING AT (F_BUF_SPACE) FOR THIS BUFFER
                    3174 ;
                    3175 ;INPUT:          IX = ADDRESS OF OBJECT TO BE PROCESSED
                    3176 ;                HL = ADDRESS OF OBJECT'S GRAPHICS TABLES IN ROM
                    3177 ;                 B = SELECTOR FOR METHOD OF COMBINING OBJECT GENERATORS
                    3178 ;                     WITH BACKGROUND GENERATORS
                    3179 ;
                    3180 ;                     1 = OBJECT PATTERN GENS ORed WITH BACKGROUND PATTERN GENS
                    3181 ;                         COLOR1 OF BACKGROUND CHANGED TO MOBILE OBJECT'S COLOR
                    3182 ;                         IF CORRESPONDING PATTERN BYTE NOT ZERO
                    3183 ;
                    3184 ;                     2 = REPLACE BACKGROUND PATTERN GENS WITH OBJECT PATTERN GENS
                    3185 ;                         TREAT COLOR SAME AS #1
                    3186 ;
                    3187 ;                     3 = SAME AS #1 EXCEPT COLOR0 CHANGED TO TRANSPARENT
                    3188 ;
                    3189 ;                     4 = SAME AS #2 EXCEPT COLOR0 CHANGED TO TRANSPARENT
                    3190 ;
                    3191 ;*****************************************************************************
                    3192 ;
                    3193 ;)
                    3194 ;          EXT     READ_VRAM,WRITE_VRAM,WORK_BUFFER,GET_VRAM,PUT_VRAM
                    3195 ;          EXT     PX_TO_PTRN_POS,GET_BKGRND,VDP_MODE_WORD,PUTFRAME
                    3196           GLB     PUT_MOBILE
                    3197 ; THE FOLLOWING ARE OFFSETS FROM THE START OF THE FREE BUFFER AREA
                    3198 ; THESE LOCATIONS USED TO STORE VARIABLES AND PATTERN AND COLOR DATA
         <0000>     3199 YDISP    EQU     0         ;Y DISPLACEMENT
         <0001>     3200 XDISP    EQU     1         ;X DISPLACEMENT
         <0002>     3201 COLR     EQU     2         ;COLOR
         <0003>     3202 FLAGS    EQU     3         ;BITS 0,1 = SELECTOR #, BIT X = GRAPHICS MODE [I/II]
         <0004>     3203 FRM      EQU     4         ;FRM   TO BE DISPLAYED
         <0005>     3204 F_GEN    EQU     5         ;NAME OF FIRST GENERATOR IN OBJECT'S GEN TABLE
         <0007>     3205 YP_OS    EQU     7         ;Y_PAT_POS OF OLD_SCREEN
         <0006>     3206 XP_OS    EQU     6         ;X_PAT_POS OF OLD_SCREEN
         <0012>     3207 YP_BK    EQU     18        ;Y_PAT_POS OF BACKGROUND
         <0011>     3208 XP_BK    EQU     17        ;X_PAT_POS OF BACKGROUND
         <001C>     3209 BK_PTN   EQU     28        ;START OF BACKGROUND PATTERN GENERATORS
         <0064>     3210 OBJ_PTN  EQU     100       ;START OF OBJECT'S PATTERN GENERATORS
         <0084>     3211 BK_CLR   EQU     132       ;START OF BACKGROUND COLOR GENERATORS
0A87                3212 PUT_MOBILE
                    3213 ; GET X AND Y LOCATIONS, CONVERT TO X AND Y PATTERN POSITIONS AND X AND Y
                    3214 ; DISPLACEMENTS [AMOUNT BY WHICH OBJECT SHIFTED OFF PATTERN POSITION BOUNDARY]
0A87     3006         15          LD      ORK_I     1       T ST/       FRE      ER #
```

```
LOCATION OBJECT CODE LINE      SOURCE LINE

  0A88 3A73C3                LD A,[VDP_MODE_WORD]      ;FIND OUT WHICH GRAPHICS MODE WE ARE IN
  0A8B CB4F                  BIT 1,A
                 ;           IF [PSW,IS,ZERO]          ;THEN MODE I
  0A90 2004                  JR NZ,ELSE1
  0A92 CB88                    RES 7,B
  0A94 1802                  JR END1
                 ;           ELSE                      ;MODE II
  0A96 CBF8      ELSE1         SET 7,B
                 ;           ENDIF
  0A98 FD7003    END1        LD [IY+FLAGS],B           ;SAVE SELECTOR
  0A9B E5                    PUSH HL                   ;SAVE GRAPHICS ADDRESS
  0A9C DD6603                LD H,[IX+3]               ;HL := ADDR_ OF STATUS
  0A9F DD6E02                LD L,[IX+2]
  0AA2 7E                    LD A,[HL]                 ;GET FRAME #
  0AA3 FD7704                LD [IY+FRM],A            ;AND SAVE
  0AA6 EE80                  XOR 80H                   ;COMPLEMENT TABLE_IN_USE FLAG
  0AA8 77                    LD [HL],A                 ;SAVE BACK IN STATUS AREA
  0AA9 23                    INC HL                    ;POINT TO X_LOCATION
  0AAA 5E                    LD E,[HL]                 ;E := LOW X_LOCATION
  0AAB 7B                    LD A,E
  0AAC E607                  AND 7                     ;A := #PIXELS TO RIGHT OF PATTERN BOUNDARY
  0AAE ED44                  NEG
  0AB0 C608                  ADD A,8                   ;AMOUNT TO SHIFT PATTERN LEFT FROM NEXT PAT BOUNDARY
  0AB2 FD7701                LD [IY+XDISP],A           ;SAVE
  0AB5 23                    INC HL
  0AB6 56                    LD D,[HL]                 ;DE := X_LOCATION
  0AB7 CD07E8                CALL PX_TO_PTRN_POS       ;CALCULATE X_PAT_POS OF BACKGROUND
  0ABA FD7311                LD [IY+XP_BK],E           ;AND SAVE
  0ABD 23                    INC HL                    ;POINT TO Y_LOCATION
  0ABE 5E                    LD E,[HL]                 ;E := LOW Y_LOCATION
  0ABF 7B                    LD A,E
  0AC0 E607                  AND 7                     ;A := #PIXELS TO RIGHT OF PATTERN BOUNDARY
  0AC2 FD7700                LD [IY+YDISP],A           ;SAVE
  0AC5 23                    INC HL
  0AC6 56                    LD D,[HL]                 ;DE := Y_LOCATION
  0AC7 CD07E8                CALL PX_TO_PTRN_POS       ;CALCULATE Y_PAT_POS
  0ACA FD7312                LD [IY+YP_BK],E

                 ; NOW GET THE NINE NAMES THAT CONSTITUTE THE BACKGROUND ON WHICH THE MOBILE OBJECT
                 ; WILL BE SUPERIMPOSED
  0ACD 2A8006    PM1         LD HL,[WORK_BUFFER]
  0AD0 110013                LD DE,YP_BK+1             ;POINT TO SPACE FOR BACKGROUND NAMES
  0AD3 19                    ADD HL,DE
  0AD4 FD5612                LD D,[IY+YP_BK]           ;D := Y_PAT_POS
  0AD7 FD5E11                LD E,[IY+XP_BK]           ;E := X_PAT_POS
  0ADA 010303                LD BC,303H                ;B := Y_EXTENT, C := X_EXTENT
  0ADD CD0898                CALL GET_BKGRND           ;GET BACKGROUND NAMES
                 ; READ OLD_SCREEN INTO BUFFER AND GET COLOR AND FIRST_GEN_NAME
  0AE0 DD5605    PM2         LD D,[IX+5]               ;DE := OLD_SCREEN ADDRESS
  0AE3 DD5E04                LD E,[IX+4]
  0AE6 DD7E06                LD A,[IX+6]               ;GET FIRST GEN NAME
  0AE9 DDE1                  POP IX                    ;IX := ADDRESS OF GRAPHICS
  0AEB FD2A8006              LD IY,[WORK_BUFFER]
  0AEF FD7705                LD [IY+F_GEN],A           ;SAVE IN BUFFER
  0AF2 D5                    PUSH DE                   ;SAVE OLD_SCREEN ADDRESS
  0AF3 2A8006                LD HL,[WORK_BUFFER]       ;HL := ADDR OF START OF BUFFER
  0AF6 010006                LD BC,XP_OS               ;SPACE TO MOVE OLD_SCREEN TO
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0AF9 09           3273          ADD HL,BC
  0AFA 010008       3274          LD BC,11              ;GET 9 NAMES FROM VRAM
                    3275 ;        IF [_D,LT,70H]        ;THEN OLD_SCREEN IS IN VRAM
  0AFD 7A           3276          LD A,D
  0AFE FE70         3277          CP 70H
  0B00 3005         3278          JR NC,ELSE2
  0B02 CD1FE2       3279            CALL READ_VRAM
  0B05 1803         3280            JR END2
                    3281 ;        ELSE                  ;OLD_SCREEN IN CPU RAM
  0B07 EB           3282 ELSE2      EX DE,HL
  0B08 EDB0         3283            LDIR
  0B0A              3284 END2  ;ENDIF
                    3285
                    3286 ; AT THIS POINT, IX = GRAPHICS, [SP] = OLD_SCREEN
                    3287 ; BACKGROUND PATTERN POSITION AND NAMES STARTING AT YP_BK
                    3288 ; OLD_SCREEN PATTERN POSITION AND NAMES STARTING AT YP_OS
                    3289 ; FIND ALL NAMES IN BACKGROUND WHICH BELONG TO THIS OBJECT'S PATTERN GENERATORS
                    3290 ; AND REPLACE WITH NAME FROM OLD_SCREEN WHICH CORRESPONDS TO THAT PATTERN POSITION
  0B0A 2A8006       3291 PM3      LD HL,[WORK_BUFFER]   ;HL := BUFFER BASE
  0B0D 110013       3292          LD DE,YP_BK+1
  0B10 19           3293          ADD HL,DE             ;POINT TO FIRST OF BACKGROUND NAMES
  0B11 D9           3294          EXX
  0B12 ED5B8006     3295          LD DE,[WORK_BUFFER]   ;DE' := BUFFER BASE
  0B16 210008       3296          LD HL,YP_OS+1
  0B19 19           3297          ADD HL,DE
  0B1A EB           3298          EX DE,HL              ;POINTS TO FIRST OF OLD_SCREEN NAMES
  0B1B D9           3299          EXX
  0B1C FD2A8006     3300          LD IY,[WORK_BUFFER]
  0B20 FD4E05       3301          LD C,[IY+F_GEN]       ;C := FIRST_GEN_NAME
                    3302 ;        DO B,9
  0B23 0609         3303          LD B,9
  0B25 7E           3304 DLP1       LD A,[HL]             ;GET A NAME
  0B26 91           3305            SUB C                 ;SUBTRACT FIRST_GEN_NAME
                    3306 ;          IF [_A,LT,18]         ;THEN NAME FALLS IN RANGE OF NAMES FOR OBJECT
  0B27 FE12         3307            CP 18
  0B29 300E         3308            JR NC,END3
                    3309 ;            IF [_A,GE,9]          ;THEN SUB 9 TO FIND CORRECT
  0B2B FE09         3310              CP 9
  0B2D 3802         3311              JR C,END4
  0B2F D609         3312                SUB 9               ;   POSITION IN OLD_SCREEN
  0B31              3313 END4         ;ENDIF
  0B31 D9           3314              EXX
  0B32 6F           3315              LD L,A              ;FORM A POINTER INTO OLD_SCREEN NAMES
  0B33 2600         3316              LD H,0
  0B35 19           3317              ADD HL,DE
  0B36 7E           3318              LD A,[HL]           ;GET OLD_SCREEN NAME
  0B37 D9           3319              EXX
  0B38 77           3320              LD [HL],A           ;REPLACE BACKGROUND NAME WITH OLD_SCREEN NAME
  0B39              3321 END3       ;ENDIF
  0B39 23           3322          INC HL                ;POINT TO NEXT NAME IN BACKGROUND
                    3323 ;        ENDDO
  0B3A 10E9         3324          DJNZ DLP1
                    3325
                    3326 ; NOW NEW VERSION OF BACKGROUND NAMES WILL NOT CONTAIN ANY NAMES OF THIS OBJECT
                    3327 ; REPLACE PREVIOUS VERSION OF OLD_SCREEN WITH THIS NEW BACKGROUND
  0B3C D1           3328 PM4      POP DE                ;DE := OLD SCREEN ADDRESS
  0B3D 2A8006       3329          LD HL,[WORK BUFFER]   ;HL := BUFFER BASE
```

```
LOCATION OBJECT CODE      SOURCE LINE

0840 010011               LD BC,XP_BK
0843 09                   ADD HL,BC            ;HL POINTS TO BACKGROUND POSITION AND NAMES
0844 010008               LD BC,11             ;NUMBER OF BYTES TO MOVE
                    ;     IF [_D,LT,70H]       ;THEN MOVE DATA TO VRAM
0847 7A                   LD A,D
0848 FE70                 CP 70H
084A 3005                 JR NC,ELSE5
084C CD1FDF                 CALL WRITE_VRAM
084F 1802                 JR END5
0851                ELSE5 ;ELSE                ;MOVE TO CPU RAM
0851 EDB0                   LDIR
0853                END5  ;ENDIF
                    ; GET THE PATTERN AND COLOR GENERATORS SPECIFIED BY THE NINE BACKGROUND NAMES
                    ; IX = GRAPHICS

0853 DDE5                         PUSH          IX                ;SAVE GRAPHICS POINTER

0855 ED5B8006       PM5   LD DE,[WORK_BUFFER]  ;DE := BUFFER BASE
0859 210013               LD HL,YP_BK+1        ;DISPLACEMENT TO BEGINNING OF BKGND NAMES
085C 19                   ADD HL,DE
085D EB                   EX DE,HL             ;DE POINT TO FIRST BACKGROUND NAME
085E 010014               LD BC,BK_PTN-8       ;HL POINTS TO BACKGROUND PAT GEN STORAGE AREA -8
0861 09                   ADD HL,BC            ;  WILL BE INCREMENTED BEFORE FIRST USE
                    ;     DO B,9
0862 0609                 LD B,9
0864 1A             DLP2    LD A,[DE]            ;GET NAME
0865 13                     INC DE               ;POINT TO NEXT NAME
0866 D5                     PUSH DE              ;SAVE NAME POINTER
0867 110008                 LD DE,8              ;INCREMENT BUFFER POINTER
086A 19                     ADD HL,DE
086B E5                     PUSH HL              ;SAVE BUFFER POINTER
086C 5F                     LD E,A               ;INDEX INTO PATTERN TABLE
086D 1600                   LD D,0
086F 4F                     LD C,A               ;SAVE PATTERN NAME IN C
0870 C5                     PUSH BC              ;SAVE COUNTER
0871 3E09                   LD A,9               ;ADD (8-9)/3 TO YP_BK TO FIND
0873 90                     SUB B                ;     OUT WHICH THIRD OF TABLES GENs ARE IN
0874 0600                   LD B,0
0876 D603           PM52    SUB 3
0878 3803                   JR C,PM51
087A 04                     INC B
087B 18F9                   JR PM52
087D 78             PM51    LD A,B
087E FD2A8006               LD IY,[WORK_BUFFER]
0882 FD8612                 ADD A,[IY+YP_BK]
0885 FDCB037E               BIT 7,[IY+FLAGS]     ;TEST GRAPHICS MODE
0889 FD210001               LD IY,1              ;NUMBER OF ELEMENTS TO READ
088D                PM6     ;IF [PSW,IS,ZERO]    ;THEN MODE I
088D 2029                   JR NZ,ELSE6
088F 3E03                     LD A,3             ;CODE FOR PATTERN GENERATOR TABLE
0891 CD1FBB                   CALL GET_VRAM
0894 C1                       POP BC             ;GET COUNT AND NAME
0895 2A8006                   LD HL,[WORK_BUFFER]
0898 C5                       PUSH BC
0899 110084                   LD DE,BK_CLR       ;DISPLACEMENT TO COLOR GEN AREA
089C 19                       ADD HL,DE          ;POINT TO IT
089D 59                       LD E,C             ;PATTERN NAME
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

   089E CB38        3387              SRL E                  ;DIVIDE NAME BY 8
   08A0 CB38        3388              SRL E
   08A2 CB38        3389              SRL E
   08A4 1600        3390              LD D,0
   08A6 3E09        3391              LD A,9                 ;CALC POSITION IN BUFFER TO MOVE GEN TO
   08A8 90          3392              SUB B
   08A9 4F          3393              LD C,A
   08AA 0600        3394              LD B,0
   08AC 09          3395              ADD HL,BC
   08AD FD210001    3396              LD IY,1
   08B1 3E04        3397              LD A,4                 ;COLOR GENERATOR TABLE CODE
   08B3 CD1FBB      3398              CALL GET_VRAM
   08B6 1821        3399              JR END6
   08B8             3400 ELSE6      ;ELSE                    ;MUST BE MODE II
   08B8 CB2F        3401              SRA A                  ;DIVIDE Y_POS BY 8
   08BA CB2F        3402              SRA A
   08BC CB2F        3403              SRA A                  ;A := THIRD OF TABLE, 0=1ST, 1=2ND, 2=3RD
   08BE             3404 PM7        ;IF [_A,LT,3]            ;IF A > 2, THEN Y_POS > 23 AND THEREFORE OFF SCREEN
   08BE FE03        3405              CP 3
   08C0 3017        3406              JR NC,END7
   08C2 57          3407                LD D,A               ;DE := 256*A + NAME
   08C3 D5          3408                PUSH DE
   08C4 E5          3409                PUSH HL
   08C5 3E03        3410                LD A,3               ;PATTERN GENERATOR TABLE CODE
   08C7 CD1FBB      3411                CALL GET_VRAM
   08CA E1          3412                POP HL               ;HL := PATTERN BUFFER ADDRESS
   08CB 110068      3413                LD DE,BK_CLR-BK_PTN ;DISPLACEMENT BETWEEN PATTERN AND COLOR BUFFERS
   08CE 19          3414                ADD HL,DE            ;HL := POINTER TO COLOR BUFFER
   08CF D1          3415                POP DE
   08D0 FD210001    3416                LD IY,1
   08D4 3E04        3417                LD A,4               ;CODE FOR COLOR TABLE
   08D6 CD1FBB      3418                CALL GET_VRAM
   08D9             3419 END7         ;ENDIF
   08D9             3420 END6       ;ENDIF
   08D9 C1          3421            POP BC                   ;RESTORE REGISTERS
   08DA E1          3422            POP HL
   08DB D1          3423            POP DE
                    3424 ;        ENDDO
   08DC 1086        3425          DJNZ DLP2
                    3426
                    3427 ; NOW THE PATTERN AND COLOR GENERATORS ARE IN THEIR RESPECTIVE BUFFERS
                    3428 ; SO GET THE FOUR GENERATORS FOR THIS FRAME OF THE OBJECT
                    3429
   08DE DDE1        3430                POP          IX               ;RESTORE GRAPHICS POINTER
                    3431
   08E0 D9          3432 PM8      EXX
   08E1 DD5603      3433          LD D,[IX+3]
   08E4 DD5E02      3434          LD E,[IX+2]          ;DE' := NEW_GEN
   08E7 DD4605      3435          LD B,[IX+5]
   08EA DD4E04      3436          LD C,[IX+4]          ;BC' := PTRN_POINTER
   08ED D9          3437          EXX
   08EE DDE5        3438          PUSH IX
   08F0 E1          3439          POP HL               ;HL := ADDRESS OF GRAPHICS
   08F1 FD2A8006    3440          LD IY,[WORK_BUFFER]
   08F5 FD7E04      3441          LD A,[IY+FRM]       ;A := FRM #
   08F8 87          3442          ADD A,A             ;X2
   08F9 4F          3443          LD C,A
```

```
LOCATION OBJECT CODE        SOURCE LINE

  08FA 0600                 LD B,0
  08FC 110006               LD DE,6               ;FRAME POINTERS START AT THIS OFFEST
  08FF 19                   ADD HL,DE
  0C00 09                   ADD HL,BC             ;HL := POINTS TO FRAME_POINTER FOR FRAME
  0C01 5E                   LD E,[HL]
  0C02 23                   INC HL
  0C03 56                   LD D,[HL]             ;DE := ADDRESS OF FRAME NAMES
  0C04 2A8006               LD HL,[WORK_BUFFER]   ;HL := BUFFER BASE ADDRESS
  0C07 01007C               LD BC,OBJ_PTN+24      ;USE LOCATION FOR LAST GEN TO STORE NAMES
  0C0A 09                   ADD HL,BC
  0C0B E5                   PUSH HL               ;SAVE FOR LATER USE
  0C0C C5                   PUSH BC               ;SAVE OFFSET
  0C0D 010005               LD BC,5
                    ;       IF [_D,LT,70H]        ;THEN NAMES ARE IN VRAM
  0C10 7A                   LD A,D
  0C11 FE70                 CP 70H
  0C13 3005                 JR NC,ELSE8
  0C15 CD1FE2                 CALL READ_VRAM        ;GET THE 4 NAMES
  0C18 1803                   JR END8
  0C1A              ELSE8   ;ELSE                 ;NAMES IN CPU MEMORY SPACE
  0C1A EB                     EX DE,HL
  0C1B EDB0                   LDIR
  0C1D              END8    ;ENDIF
  0C1D FD2A8006             LD IY,[WORK_BUFFER]   ;GET COLOR BYTE
  0C21 C1                   POP BC                ;OFFSET TO FIRST NAME
  0C22 FD09                 ADD IY,BC
  0C24 FD7E04               LD A,[IY+4]           ;A := COLOR BYTE
  0C27 FD2A8006             LD IY,[WORK_BUFFER]
  0C2B FD7702               LD [IY+COLR],A        ;COLR := COLOR BYTE
  0C2E D1           PM9     POP DE                ;DE := ADDRESS OF FIRST NAME IN BUFFER
  0C2F 2A8006               LD HL,[WORK_BUFFER]   ;HL := BUFFER BASE ADDRESS
  0C32 010064               LD BC,OBJ_PTN         ;START OF OBJECT'S PATTERN BUFFER
  0C35 09                   ADD HL,BC
                    ;       DO B,4                ;GET 4 PATTERNS CORRESPONDING TO THE FOUR NAMES
  0C36 0604                 LD B,4
  0C38 1A           DLP4    LD A,[DE]               ;GET NAME
  0C39 DDBE01                 CP [IX+1]             ;COMPARE TO NUMGEN
  0C3C D5                     PUSH DE               ;SAVE POINTER TO NAMES
                    ;         IF [PSW,IS,CARRY]     ;THEN NAME < NUMGEN, THEREFORE
  0C3D 3016                   JR NC,ELSE9
  0C3F D9                       EXX                 ;     PATTERN PART OF GRAPHICS TABLES
  0C40 87                       ADD A,A
  0C41 87                       ADD A,A
  0C42 87                       ADD A,A             ;A := 8*NAME
  0C43 6F                       LD L,A
  0C44 2600                     LD H,0
  0C46 09                       ADD HL,BC           ;HL := POINTER TO PATTERN
  0C47 E5                       PUSH HL
  0C48 D9                       EXX
  0C49 D1                       POP DE              ;DE :=  "          "
  0C4A EB                       EX DE,HL
  0C4B C5                       PUSH BC
  0C4C 010008                   LD BC,8             ;NUMBER OF BYTES TO MOVE
  0C4F EDB0                     LDIR
  0C51 C1                       POP BC
  0C52 EB                       EX DE,HL
  0C53 1830                     JR END9
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0DB8 CB3F                              SRL A                 ;THIS NUMBER / 8 INDICATES WHICH 1/3 OF
  0DBA CB3F                              SRL A                 ;     TABLES TO USE
  0DBC CB3F                              SRL A
  0DBE 57                                LD D,A
  0DBF 59                                LD E,C                ;DE := INDEX INTO PATTERN AND COLOR TABLES
  0DC0 D5                                PUSH DE               ;SAVE INDEX
  0DC1 01001C                            LD BC,BK_PTN          ;FORM POINTER TO GENERATORS IN HL
  0DC4 09                                ADD HL,BC
  0DC5 ED4B8006                          LD BC,[WORK_BUFFER] ;GET BUFFER BASE ADDR
  0DC9 09                                ADD HL,BC
  0DCA E5                                PUSH HL               ;SAVE THIS POINTER
  0DCB FD210003                          LD IY,3               ;NUMBER OF ELEMENTS TO MOVE
  0DCF 3E03                              LD A,3                ;PATTERN GENERATOR TABLE CODE
  0DD1 CD1FBE                            CALL PUT_VRAM
  0DD4 E1                                POP HL                ;GET POINTER BACK
  0DD5 110068                            LD DE,BK_CLR-BK_PTN ;OFFSET BETWEEN BUFFERS
  0DD8 19                                ADD HL,DE             ;HL POINTS TO START OF NEXT 3 COLOR GENERATORS
  0DD9 D1                                POP DE                ;GET INDEX INTO GEN TABLES
  0DDA FD210003                          LD IY,3
  0DDE 3E04                              LD A,4                ;CODE FOR COLOR GENERATOR TABLE
  0DE0 CD1FBE                            CALL PUT_VRAM
  0DE3                      END15      ;ENDIF
  0DE3 C1                               POP BC              ;RESTORE COUNTER AND INDEX
  0DE4 04                               INC B
  0DE5 78                             LD A,B
  0DE6 FE03                           CP 3
  0DE8 20AE                           JR NZ,RPT2
                            ;         UNTIL [_B,EQ,3]     ;REPEAT 3 TIMES
  0DEA                      END04   ;ENDIF

                            ; RESTORE OLD_SCREEN IF IT'S Y_PAT_POS AND X_PAT_POS DIFFERS FROM THE
                            ; Y_PAT_POS AND X_PAT_POS FOR THE OBJECT
  0DEA FD2A8006             PM14    LD IY,[WORK_BUFFER]
  0DEE FD4606                       LD B,[IY+XP_OS]            ;TEST FOR VALID OLD_SCREEN DATA
                            ;       IF [_B,NE,80H]             ;THEN THERE IS VALID DATA
  0DF1 78                           LD A,B
  0DF2 FE80                         CP 80H
  0DF4 2821                         JR Z,END16
  0DF6 FD4E07                         LD C,[IY+YP_OS]            ;TEST IF OS POSITION SAME AS CURRENT POSITION
  0DF9 FD6611                         LD H,[IY+XP_BK]
  0DFC FD6E12                         LD L,[IY+YP_BK]
  0DFF B7                             OR A                     ;CLEAR THE CARRY
  0E00 ED42                           SBC HL,BC                ;IS THERE ANY DIFFERENCE?
                            ;         IF [PSW,IS,NZERO]        ;THEN POSTION HAS CHANGED
  0E02 2813                           JR Z,END17
  0E04 2A8006                           LD HL,[WORK_BUFFER]      ;GET BUFFER BASE
  0E07 110008                           LD DE,YP_OS+1
  0E0A 19                               ADD HL,DE                ;POINT TO OLD_SCREEN NAMES
  0E0B FD5E06                           LD E,[IY+XP_OS]
  0E0E FD5607                           LD D,[IY+YP_OS]          ;DE := X AND Y PAT_POS
  0E11 010303                           LD BC,0303H              ;BC := X AND Y EXTENT
  0E14 CD0808                           CALL PUTFRAME
  0E17                      END17     ;ENDIF
  0E17                      END16   ;ENDIF
                            ; PLACE OBJECT ON SCREEN
  0E17 FD2A8006                     LD IY,[WORK_BUFFER]
  0E1B 2A8006                       LD HL,[WORK_BUFFER]       ;HL := BUFFER BASE ADDR
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0E1E 110013     3786          LD DE,YP_BK+1          ;POINT TO NAMES FOR OBJECT
  0E21 19         3787          ADD HL,DE
  0E22 FD5E11     3788          LD E,[IY+XP_BK]
  0E25 FD5612     3789          LD D,[IY+YP_BK]        ;DE := X AND Y PAT_POS
  0E28 010303     3790          LD BC,0303H            ;BC := X AND Y EXTENT
  0E2B CD0808     3791          CALL PUTFRAME
                  3792 ***************** END OF PUT_MOBILE **********************************************
  0E2E C9         3793          RET
                  3794
                  3795 ; REGS A, H AND L CONTAIN 24 BIT PATTERN TO BE COMBINED WITH BACKGROUND GENERATORS
                  3796 ; IX POINTS TO THE FIRST OF THREE GENERATOR BYTES TO BE COMBINED WITH A, H AND L
  0E2F            3797 COM_PAT_COL
  0E2F FDCB0346   3798          BIT 0,[IY+FLAGS]       ;'OR' GENs OR REPLACE
                  3799 ;        IF [PSW,IS,ZERO]       ;THEN 'OR'
  0E33 2016       3800          JR NZ,ELSE18
  0E35 DDB600     3801            OR [IX+0]             ;'OR' LEFT BYTE WITH BACKGROUND
  0E38 DD7700     3802            LD [IX+0],A           ;AND SUBSTITUTE FOR THAT GENERATOR BYTE
  0E3B 7C         3803            LD A,H                ;NOW DO MIDDLE BYTE
  0E3C DDB608     3804            OR [IX+8]
  0E3F DD7708     3805            LD [IX+8],A
  0E42 7D         3806            LD A,L                ;RIGHT HAND BYTE
  0E43 DDB610     3807            OR [IX+16]
  0E46 DD7710     3808            LD [IX+16],A
  0E49 1814       3809            JR END18
  0E4B            3810 ELSE18   ;ELSE                  ;REPLACE BACKGROUND WITH NON-ZERO BYTES
  0E4B B7         3811            OR A                  ;IS BYTE NON-ZERO
                  3812 ;          IF [PSW,IS,NZERO]
  0E4C 2803       3813            JR Z,END19
  0E4E DD7700     3814              LD [IX+0],A         ;YES, THEN REPLACE BACKGROUND WITH OBJECT
  0E51            3815 END19      ;ENDIF
  0E51 7C         3816            LD A,H                ;SAME FOR MIDDLE BYTE
  0E52 B7         3817            OR A
                  3818 ;          IF [PSW,IS,NZERO]
  0E53 2803       3819            JR Z,END20
  0E55 DD7708     3820              LD [IX+8],A
  0E58            3821 END20      ;ENDIF
  0E58 7D         3822            LD A,L                ;SAME FOR RIGHT HAND BYTE
  0E59 B7         3823            OR A
                  3824 ;          IF [PSW,IS,NZERO]
  0E5A 2803       3825            JR Z,END21
  0E5C DD7710     3826              LD [IX+16],A
  0E5F            3827 END21      ;ENDIF
  0E5F            3828 END18    ;ENDIF
  0E5F FDCB037E   3829 PM15     BIT 7,[IY+FLAGS]       ;FIND OUT WHICH GRAPHICS MODE USED
                  3830 ;        IF [PSW,IS,NZERO]      ;THEN MODE 2 [MODE 1 COLORS DONE AFTER COMBINELOOP]
  0E63 283C       3831          JR Z,END22
  0E65 DDE5       3832            PUSH IX               ;SAVE BACKGROUND POINTER
  0E67 010068     3833            LD BC,BK_CLR-BK_PTN   ;CHANGE IX TO POINT TO COLOR GENERATORS
  0E6A DD09       3834            ADD IX,BC
  0E6C FD4602     3835            LD B,[IY+COLR]        ;GET OBJECT COLOR
  0E6F FDCB034E   3836            BIT 1,[IY+FLAGS]      ;COLOR0 = BACKGROUND OR TRANSPARENT ?
                  3837 ;          IF [PSW,IS,ZERO]     ;THEN USE BACKGROUND COLOR0
  0E73 2004       3838            JR NZ,ELSE23
  0E75 0E0F       3839              LD C,0FH            ;MASK FOR COLOR0 OF BACKGROUND
  0E77 1802       3840              JR END23
  0E79            3841 ELSE23     ;ELSE
  0E79 0E00       3842              LD C,0              ;MASK REPLACE COLOR0 WITH TRANSPARENT
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0E7B                  END23     ;ENDIF
  0E7B 7B                         LD A,E                 ;GET FIRST OBJECT'S PATTERN BYTE
  0E7C B7                         OR A                   ;ARE THERE ANY '1' BITS ?
                        ;         IF [PSW,IS,NZERO]
  0E7D 2808                       JR Z,END24
  0E7F DD7E00                       LD A,[IX+0]            ;GET BACKGROUND COLOR GEN
  0E82 A1                           AND C                  ;MASK OUT COLOR1
  0E83 B0                           OR B                   ;ADD OBJECT COLOR1
  0E84 DD7700                       LD [IX+0],A            ;UPDATE COLOR GENERATOR
  0E87                  END24     ;ENDIF
  0E87 7C                         LD A,H                 ;SAME FOR MIDDLE BYTE
  0E88 B7                         OR A
                        ;         IF [PSW,IS,NZERO]
  0E89 2808                       JR Z,END25
  0E8B DD7E08                       LD A,[IX+8]
  0E8E A1                           AND C
  0E8F B0                           OR B
  0E90 DD7708                       LD [IX+8],A
  0E93                  END25     ;ENDIF
  0E93 7D                         LD A,L                 ;RIGHT HAND BYTE
  0E94 B7                         OR A
                        ;         IF [PSW,IS,NZERO]
  0E95 2808                       JR Z,END26
  0E97 DD7E10                       LD A,[IX+16]
  0E9A A1                           AND C
  0E9B B0                           OR B
  0E9C DD7710                       LD [IX+16],A
  0E9F                  END26     ;ENDIF
  0E9F DDE1                       POP IX                 ;RESTORE BACKGROUND POINTER
  0EA1                  END22    ;ENDIF
  0EA1 C9                        RET
                        ;        END            ;PUT_MOBILE
                         PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      ;       .IDENT PUTCOMP
                      ;       .ZOP
                      ;       .EPOP
                      ;       .IF1 ,.INSERT B:SPZ80.ASM
                      ;       .COMMENT )
                      ;                                                                   4/15/82
                      ;                                                                   10:40:00
                      ;****************** PUT_COMPLEX *************************************************
                      ;
                      ;DESCRIPTION:    THE POSITION AND FRAME NUMBER OF EACH OF A COMPLEX OBJECT'S
                      ;                COMPONENT OBJECTS IS UPDATED.  THEN PUT_OBJECT IS CALLED FOR
                      ;                EACH OF THE COMPONENT OBJECTS.
                      ;
                      ;INPUT:          IX = ADDRESS OF OBJECT TO BE PROCESSED
                      ;                HL = ADDRESS OF OBJECT'S GRAPHICS TABLES IN ROM
                      ;                 B = SELECTOR FOR METHODE OF COMBINING OBJECT GENERATORS
                      ;                     WITH BACKGROUND GENERATORS
                      ;
                      ;                     1 = OBJECT PATTERN GENS ORed WITH BACKGROUND PATTERN GENS
                      ;                         COLOR1 OF BACKGROUND CHANGED TO MOBILE OBJECT'S COLOR
                      ;                         IF CORRESPONDING PATTERN BYTE NOT ZERO
                      ;
                      ;                     2 = REPLACE BACKGROUND PATTERN GENS WITH OBJECT PATTERN GENS
                      ;                         TREAT COLOR SAME AS #1
                      ;
                      ;                     3 = SAME AS #1 EXCEPT COLOR0 CHANGED TO TRANSPARENT
                      ;
                      ;                     4 = SAME AS #2 EXCEPT COLOR0 CHANGED TO TRANSPARENT
                      ;
                      ;                 C = OBJECT TYPE, AND NUMBER OF COMPONENTS
                      ;
                      ;******************************************************************************
                      ;)
                      ;       EXT     PUTOBJ
                              GLB     PUTCOMPLEX
                      ;
 0EA2                 PUTCOMPLEX
                      ; UPDATE THE FRAME NUMBER AND THE X AND Y LOCATION IN EACH OF THE COMPONTEN
                      ; OBJECT'S STATUS AREAS
 0EA2 C5                      PUSH BC                 ;SAVE SELECTOR AND COMPONENT COUNT [COMP_CNT]
 0EA3 D9                      EXX                     ;USE PRIMED REGS FOR X_LOC AND Y_LOC
 0EA4 DD6603                  LD H,[IX+3]             ;HIGH BYTE OF STATUS
 0EA7 DD6E02                  LD L,[IX+2]             ;LOW BYTE OF STATUS
 0EAA 7E                      LD A,[HL]               ;A := FRAME
 0EAB 23                      INC HL
 0EAC 4E                      LD C,[HL]               ;BC' := X_LOCATION
 0EAD 23                      INC HL
 0EAE 46                      LD B,[HL]
 0EAF 23                      INC HL
 0EB0 5E                      LD E,[HL]               ;DE' := Y_LOCATION
 0EB1 23                      INC HL
 0EB2 56                      LD D,[HL]
 0EB3 D9                      EXX
 0EB4 87                      ADD A,A                 ;FRAME := 4*FRAME
 0EB5 87                      ADD A,A
 0EB6 5F                      LD E,A                  ;FORM POINTER TO FRAME AND OFFSET POINTERS
 0EB7 1600                    LD D,0
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0EB9 23                     INC HL                 ;POINT TO FIRST OF FRA_OFFSET_PNTR PAIRS
  0EBA 19                     ADD HL,DE              ;POINT TO FRAME POINTER
  0EBB 4E                     LD C,[HL]
  0EBC 23                     INC HL
  0EBD 46                     LD B,[HL]              ;BC := FRAME POINTER [POINTER TO LIST OF FRAME #'S]
  0EBE 23                     INC HL
  0EBF 5E                     LD E,[HL]
  0EC0 23                     INC HL
  0EC1 56                     LD D,[HL]              ;DE := OFFSET POINTER [PNTR TO LIST OF OFFSETS]
  0EC2 60                     LD H,B
  0EC3 69                     LD L,C                 ;HL := FRAME POINTER
                         ;
                         ; DE' = Y_LOC, BC' = X_LOC, HL = PNTR TO FRAME LIST, DE = PNTR TO OFFSET LIST
                         ; IX = ADDR OF OBJ, [SP] = COMP_CNT & SELECTOR
                         ; FOR N=0 TO COMP_CNT-1: COMP_OBJ[N] FRAME := FRAME#[N] FROM FRAME LIST
                         ;                        COMP_OBJ[N] X_LOCATION := CMPLX_OBJ X_LOCATION + X_OFFSET[N]
                         ;                        COMP_OBJ[N] Y_LOCATION := CMPLX_OBJ Y_LOCATION + Y_OFFSET[N]
  0EC4 C1                     POP BC
  0EC5 79                     LD A,C                 ;GET COMPONENT COUNT INTO B
  0EC6 48                     LD C,B                 ;SAVE SELECTOR IN C
  0EC7 CB3F                   SRL A                  ;GET COUNT INTO LOW NIBBLE
  0EC9 CB3F                   SRL A
  0ECB CB3F                   SRL A
  0ECD CB3F                   SRL A
  0ECF 47                     LD  B,A                ;B := component count
  0ED0 C5                     PUSH BC                ;SAVE COUNTER AND SELECTOR ON STACK
  0ED1 DDE5                   PUSH IX                ;SAVE ADDR OF OBJ
  0ED3 E5                LP1  PUSH HL                ;SAVE PNTR TO FRAME LIST
  0ED4 D5                     PUSH DE                ;SAVE PNTR TO OFFSET LIST
  0ED5 DD6E04                 LD L,[IX+4]
  0ED8 DD6605                 LD H,[IX+5]            ;HL := ADDR OF COMPONENT OBJ
  0EDB DD23                   INC IX                 ;Point to next object pointer
  0EDD DD23                   INC IX
  0EDF 23                     INC HL
  0EE0 23                     INC HL                 ;HL POINTS TO STATUS POINTER
  0EE1 5E                     LD E,[HL]
  0EE2 23                     INC HL
  0EE3 56                     LD D,[HL]              ;DE := ADDR OF STATUS FOR COMPONENT OBJECT
  0EE4 D5                     PUSH DE
  0EE5 FDE1                   POP IY                 ;IY := ADDR OF STATUS FOR COMPONENT OBJECT
  0EE7 D1                     POP DE                 ;DE := PNTR TO OFFSET LIST
  0EE8 E1                     POP HL                 ;HL := PNTR TO FRAME LIST
  0EE9 7E                     LD A,[HL]              ;GET FRAME NUMBER
  0EEA FDCB007E               BIT 7,[IY+0]           ;Preserve bit 7 of frame
  0EEE 2802                   JR  Z,TBLO             ;used by mobile objects to indicate which
                                                     ;VRAM tables in use.
  0EF0 CBFF                   SET 7,A
  0EF2                   TBLO:
  0EF2 FD7700                 LD [IY+0],A            ;MOVE TO COMPONENTS STATUS AREA
  0EF5 23                     INC HL                 ;POINT TO NEXT FRAME NUMBER
  0EF6 1A                     LD A,[DE]              ;GET X_OFFSET
  0EF7 D9                     EXX
  0EF8 6F                     LD L,A
  0EF9 2600                   LD H,0                 ;HL' := X_OFFSET
  0EFB 09                     ADD HL,BC              ;HL' := X_OFFSET + X_LOCATION
  0EFC FD7501                 LD [IY+1],L
  0EFF FD7402                 LD [IY+2],H            ;COMPONENT'S X LOCATION := X_OFFSET + X_LOCATION
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

   0F02 D9          3991          EXX
   0F03 13          3992          INC DE                  ;POINT TO Y_OFFSET
   0F04 1A          3993          LD A,[DE]
   0F05 D9          3994          EXX
   0F06 6F          3995          LD L,A
   0F07 2600        3996          LD H,0                  ;HL := Y_OFFSET
   0F09 19          3997          ADD HL,DE               ;HL := Y_OFFSET + Y_LOCATION
   0F0A FD7503      3998          LD [IY+3],L
   0F0D FD7404      3999          LD [IY+4],H             ;COMPONENT'S Y_LOCATION := Y_OFFSET + Y_LOCATION
   0F10 D9          4000          EXX
   0F11 13          4001          INC DE                  ;POINT TO NEXT OFFSET PAIR
   0F12 10BF        4002          DJNZ LP1
                    4003 ;
                    4004 ; CALL PUT_OBJECT FOR EACH OF THE COMPONENT OBJECTS, PASS SELECTOR IN B
   0F14 FDE1        4005          POP IY                  ;GET OBJECT ADDRESS BACK
   0F16 010004      4006          LD BC,4
   0F19 FD09        4007          ADD IY,BC               ;IY POINTS TO POINTER TO FIRST COMPONENT OBJECT
   0F1B D1          4008          POP DE                  ;DE := COUNTER AND SELECTOR
   0F1C FD6E00      4009 LP2      LD L,[IY+0]
   0F1F FD6601      4010          LD H,[IY+1]             ;HL := ADDRESS OF COMPONENT OBJECT
   0F22 FD23        4011          INC IY
   0F24 FD23        4012          INC IY                  ;IY POINTS TO NEXT COMPONENT OBJECT POINTER
   0F26 E5          4013          PUSH HL
   0F27 DDE1        4014          POP IX                  ;IX := ADDRESS OF COMPONENT OBJECT
   0F29 FDE5        4015          PUSH IY                 ;SAVE POINTER
   0F2B D5          4016          PUSH DE                 ;SAVE COUNTER AND SELECTOR
   0F2C 43          4017          LD B,E                  ;B := SELECTOR
   0F2D CD1FFA      4018          CALL PUTOBJ
   0F30 D1          4019          POP DE                  ;GET COUNTER AND SELECTOR
   0F31 FDE1        4020          POP IY                  ;GET ADDRESS OF NEXT COMPONENT OBJECT POINTER
   0F33 15          4021          DEC D
   0F34 20E6        4022          JR NZ,LP2
   0F36 C9          4023          RET
                    4024   PROG
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

                         ;               Ken   Lagace and Rob Jepson 3/82

                                            GLB              TIMER_TABLE_BASE       ;IS THIS NECESSARY ??
                                            GLB              NEXT_TIMER_DATA_BYTE   ; OR THIS???

                                           ;EXT              PARAM_       ;Parameter passing routine needed
                                                                          ;for setting up pascal interfaces

                                                         ;Global routine labels are the routines to call
                                                         ;from pascal
                                            GLB              INIT_TIMER_
                                            GLB              INIT_TIMERQ
                                            GLB              FREE_SIGNAL_
                                            GLB              FREE_SIGNALQ
                                            GLB              REQUEST_SIGNAL_
                                            GLB              REQUEST_SIGNALQ
                                            GLB              TEST_SIGNAL_
                                            GLB              TEST_SIGNALQ
                                            GLB              TIME_MGR_
                                            GLB              TIME_MGRQ
         <0007>          DONE               EQU              7
         <0006>          REPEAT             EQU              6
         <0005>          FREE               EQU              5
         <0004>          EOT                EQU              4
         <0003>          LONG               EQU              3
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         ;                                                                     ;
                         ;                ------------ MODE BIT -------------                  ;
                         ;      TO CHECK BIT =====  BIT ?  JR Z = JUMP IF BIT IS 0 !           ;
                         ;                                                                     ;
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         ;       ;        ;       ;       ;;       ;       ;       ;       ;
                         ; DONE  ; REPEAT ; FREE  ; E O T ;; LONG  ;       ;       ;       ;
                         ;   7   ;   6    ;   5   ;   4   ;;  3    ;   2   ;   1   ;   0   ;
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

                                           PROG

 0F37                    TIME_MGRQ
 0F37                    TIME_MGR_
 0F37 2A7303                               LD              HL,[TIMER_TABLE_BASE];Current timer addr.
 0F3A                    NEXT_TIMERO
 0F3A CB6E                                 BIT             FREE,[HL]       ;Free?
 0F3C CC0F49                               CALL            Z,DCR_TIMER     ;If not, decr.
 0F3F CB66                                 BIT             EOT,[HL]        ;End of table?
 0F41 2005                                 JR              NZ,SCRAM        ; If it is, we're done.
 0F43 23                                   INC             HL              ;Otherwise get next timer
 0F44 23                                   INC             HL              ; and start over.
 0F45 23                                   INC             HL
 0F46 18F2                                 JR              NEXT_TIMERO
 0F48                    SCRAM
 0F48 C9                                   RET
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

  0F49                   DCR_TIMER
  0F49 E5                          PUSH   HL                ;Save current timer.
  0F4A CB5E                        BIT    LONG,[HL]         ;Long?
  0F4C 282B                        JR     Z,DCR_S_MODE_TBL;Short,non-repeating
  0F4E CB76                        BIT    REPEAT,[HL]       ;Repeat?
  0F50 200D                        JR     NZ,DCR_L_RPT_TBL;Long, repeating
  0F52                   DCR_L_MODE_TBL                     ;Long non-repeating
  0F52 23                          INC    HL
  0F53 5E                          LD     E,[HL]            ;Move counter to DE
  0F54 23                          INC    HL
  0F55 56                          LD     D,[HL]
  0F56 1B                          DEC    DE                ;Decrement.
  0F57 7B                          LD     A,E
  0F58 B2                          OR     D                 ;Check if 0.
  0F59 2034                        JR     NZ,SAVE_2_BYTES ;If not, save'm.
  0F5B E1                          POP    HL                ;Otherwise, get mode byte
  0F5C E5                          PUSH   HL                ; and set it's done bit.
  0F5D 182C                        JR     SET_DONE_BIT
  0F5F                   DCR_L_RPT_TBL
  0F5F 23                          INC    HL                ;Long-repeating timer.
  0F60 5E                          LD     E,[HL]            ;Load addr. into DE.
  0F61 23                          INC    HL
  0F62 56                          LD     D,[HL]
  0F63 EB                          EX     DE,HL             ;Exchange and
  0F64 5E                          LD     E,[HL]            ; load counter into DE.
  0F65 23                          INC    HL
  0F66 56                          LD     D,[HL]
  0F67 1B                          DEC    DE                ;Decrement.
  0F68 7B                          LD     A,E
  0F69 B2                          OR     D                 ;Check for 0.
  0F6A 2023                        JR     NZ,SAVE_2_BYTES ;Save if not.
  0F6C 23                          INC    HL                ;Otherwise, reload
  0F6D 5E                          LD     E,[HL]            ; original counter #.
  0F6E 23                          INC    HL
  0F6F 56                          LD     D,[HL]            ;Jockey all over to
  0F70 2B                          DEC    HL                ; perform said task!
  0F71 2B                          DEC    HL
  0F72 72                          LD     [HL],D
  0F73 2B                          DEC    HL
  0F74 73                          LD     [HL],E
  0F75 E1                          POP    HL
  0F76 E5                          PUSH   HL
  0F77 1812                        JR     SET_DONE_BIT      ;Then set done bit.
  0F79                   DCR_S_MODE_TBL
  0F79 23                          INC    HL
  0F7A 35                          DEC    [HL]
  0F7B 2010                        JR     NZ,TIMER_EXIT
  0F7D E1                          POP    HL
  0F7E E5                          PUSH   HL
  0F7F CB76                        BIT    REPEAT,[HL]       ;Repeat?
  0F81 2808                        JR     Z,SET_DONE_BIT    ;If not, leave.
  0F83 23                          INC    HL                ;Otherwise, jockey
  0F84 23                          INC    HL                ; around again and
  0F85 7E                          LD     A,[HL]            ; reload original #.
```

```
CATION OBJECT CODE  SOURCE LINE

 0F86 2B                          DEC     HL
 0F87 77                          LD      (HL),A
 0F88 2B                          DEC     HL
 0F89 E1                          POP     HL
 0F8A E5                          PUSH    HL
 0F8B                SET_DONE_BIT
 0F8B CBFE                        SET     7,(HL)
 0F8D                TIMER_EXIT
 0F8D E1                          POP     HL
 0F8E C9                          RET



 0F8F                SAVE_2_BYTES
 0F8F 72                          LD      (HL),D
 0F90 2B                          DEC     HL
 0F91 73                          LD      (HL),E
 0F92 18F9                        JR      TIMER_EXIT

                     ;Procedure Init Timer
                     ;HL has address of Timer Table
                     ;DE has address of Timer Data Table

                     ;          COMN
                     ;INIT_TIME_DATA:
                     ;TEMP1:
                     ;          DEFS 2
                     ;TEMP2:
                     ;          DEFS 2

                                PROG
 0F94                INIT_TIME_PAR:
 0F94 00020002                  DEFW 2,2,2
 0F98 0002

 0F9A                INIT_TIMERQ:
 0F9A 010F94                    LD   BC,INIT_TIME_PAR
 0F9D 11738A                    LD   DE,INIT_TIME_DATA
 0FA0 CD0098                    CALL PARAM
 0FA3 2A738A                    LD   HL,(TEMP1)
 0FA6 ED5B738C                  LD   DE,(TEMP2)
 0FAA                INIT_TIMER_:
 0FAA 227303                    LD   (TIMER_TABLE_BASE),HL       ;Store given base address for timer table
 0FAD 3630                      LD   (HL),30H                    ;Set first byte in timer table to free and last timer
 0FAF EB                        EX   DE,HL
 0FB0 227305                    LD   (NEXT_TIMER_DATA_BYTE),HL   ;Store given base address for data block
 0FB3 C9                        RET

                     ;Procedure Free Signal
                     ;Acc has signal number to be freed
                     ;No output is generated
                     ;          COMN
                     ;SIGNAL_NUM:
                     ;          DEFS 1


                                PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     4196
  0FB4               4197 FREE_SIG_PAR:
  0FB4 00010001      4198          DEFW 1,1
                     4199
  0FB8               4200 FREE_SIGNALQ:
  0FB8 010FB4        4201          LD   BC,FREE_SIG_PAR
  0FBB 1173BE        4202          LD   DE,SIGNAL_NUM
  0FBE CD0098        4203          CALL PARAM_
  0FC1 3A73BE        4204          LD   A,[SIGNAL_NUM]
                     4205
  0FC4               4206 FREE_SIGNAL_:
  0FC4 4F            4207          LD   C,A                        ;Put free Code into C register
  0FC5 2A73D3        4208          LD   HL,[TIMER_TABLE_BASE]      ;Get Timer Base Address
  0FC8 47            4209          LD   B,A                        ;Timer offset already available
  0FC9 110003        4210          LD   DE,3                       ;Setup offset to subsequent timers
  0FCC B7            4211          OR   A                          ;See if the first timer is requested
  0FCD 2808          4212          JR   Z,FREE_MATCH               ;If so we have a match
                     4213
  0FCF               4214 FREE1:                                   ;Loop to find requested signal
  0FCF CB66          4215          BIT  EOT,[HL]                   ;Check if end of table
  0FD1 206A          4216          JR   NZ,FREE_EXIT               ;If so go no further
  0FD3 19            4217          ADD  HL,DE                      ;If not then offset to next timer
  0FD4 0D            4218          DEC  C                          ;Decrement signal requested
  0FD5 20F8          4219          JR   NZ,FREE1                   ;If requested signal not zero then go try the next
                     4220
  0FD7               4221 FREE_MATCH:                              ;Here when the requested signal matches current signal
  0FD7 CB6E          4222          BIT  FREE,[HL]                  ;Check if already free
  0FD9 2062          4223          JR   NZ,FREE_SET               ;If so then just reset LONG
  0FDB CBEE          4224          SET  FREE,[HL]                  ;Set current timer to free
  0FDD CB76          4225          BIT  REPEAT,[HL]                ;Check for repeating timer
  0FDF 285C          4226          JR   Z,FREE_SET                 ;If not then go free it
  0FE1 CB5E          4227          BIT  LONG,[HL]                  ;Check also for long timer
  0FE3 2858          4228          JR   Z,FREE_SET                 ;If not long then go free it
                     4229 ;        CALL FREE_COUNTER_              ;Release timer data before freeing timer
                     4230
                     4231 ;
                     4232 ;             FREE   (DELETE) COUNTER
                     4233 ;                    KENL 3/82
                     4234 ;
                     4235 ;       NEEDS   "TO-DELETE":COUNTER ADDR.IN DE
                     4236
  0FE5               4237 FREE_COUNTER_                  ;HL should contain timer to delete addr.
  0FE5 23            4238             INC        HL        ;Get next after mode byte
  0FE6 5E            4239             LD         E,[HL]         ; into DE.
  0FE7 23            4240             INC        HL
  0FE8 56            4241             LD         D,[HL]
  0FE9 D5            4242             PUSH       DE             ;Save them for later.
  0FEA 2A73D3        4243             LD         HL,[TIMER_TABLE_BASE]
  0FED E5            4244             PUSH       HL             ;Save beginning of table.
  0FEE               4245 NEXT
  0FEE CB66          4246             BIT        EOT,[HL]       ;End of table?
  0FF0 202E          4247             JR         NZ,MOVE_IT
  0FF2 CB6E          4248             BIT        FREE,[HL]      ;Free?
  0FF4 2023          4249             JR         NZ,GET_NEXT    ;If so we don't want it.
  0FF6 7E            4250             LD         A,[HL]
  0FF7 E648          4251             AND        48H            ;Repeating and long?
  0FF9 FE48          4252             CP         48H
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  0FFB 201C                      JR      NZ,GET_NEXT      ;If NOT we don't want it.
  0FFD 23                        INC     HL
  0FFE 23                        INC     HL
  0FFF 7E                        LD      A,[HL]
  1000 BA                        CP      D
  1001 3816                      JR      C,GET_NEXT       ;If so we don't want it.
  1003 2008                      JR      NZ,SUBTRACT_4    ;However, if larger, change it.
  1005 2B                        DEC     HL
  1006 7E                        LD      A,[HL]
  1007 BB                        CP      E                ;Smaller?
  1008 380F                      JR      C,GET_NEXT       ;If so we don't want it.
  100A 2831                      JR      Z,EXIT           ;Error if equal!
  100C 23                        INC     HL               ;Set up HL for SUBTRACT_4
  100D                SUBTRACT_4
  100D 56                        LD      D,[HL]
  100E 2B                        DEC     HL
  100F 5E                        LD      E,[HL]
  1010 1B                        DEC     DE               ;Reduce this addr. by 4.
  1011 1B                        DEC     DE
  1012 1B                        DEC     DE
  1013 1B                        DEC     DE
  1014 73                        LD      [HL],E           ;Replace reduced addr.
  1015 23                        INC     HL               ; back where we got it.
  1016 72                        LD      [HL],D
  1017 1800                      JR      GET_NEXT
  1019                GET_NEXT
  1019 E1                        POP     HL               ;Now we can get next timer.
  101A 23                        INC     HL
  101B 23                        INC     HL
  101C 23                        INC     HL
  101D E5                        PUSH    HL
  101E 18CE                      JR      NEXT
  1020                MOVE_IT
  1020 0600                      LD      B,0
  1022 B7                        OR      A                ;CLEAR CARRY
  1023 E1                        POP     HL
  1024 D1                        POP     DE               ;Get addr. of timer to delete.
  1025 E5                        PUSH    HL
  1026 2A7305                    LD      HL,[NEXT_TIMER_DATA_BYTE] ;Find # of bytes
  1029 ED52                      SBC     HL,DE            ;to move by subtraction.
  102B 4D                        LD      C,L              ;Save in counter reg.
  102C 6B                        LD      L,E              ;Copy into HL.
  102D 62                        LD      H,D
  102E 23                        INC     HL               ;Find source addr.
  102F 23                        INC     HL
  1030 23                        INC     HL
  1031 23                        INC     HL
  1032 EDB0                      LDIR                     ;Move it!
  1034 010008                    LD      BC,8             ;Adjust Next available byte by -4 from LDIR dest.
  1037 ED42                      SBC     HL,BC            ;(or -8 from source of LDIR! [saves instrs.]).
  1039 227305                    LD      [NEXT_TIMER_DATA_BYTE],HL
  103C E1                        POP     HL
  103D                EXIT
                      ;          RET

  13D
```

```
LOCATION OBJECT CODE LINE      SOURCE LINE

                         ;              RES  LONG,(HL)
1030                     FREE_EXIT:                                  ;Reset repeat bit just in case
1030 C9                                 RET                          ;Return


                         ;Procedure Request Signal
                         ;HL pair has length of timer
                         ;Acc has zero for repeating timer any other value for a non_repeating type
                         ;Signal number is returned in the Accumulator
                         ;              COMM
                         ;REPEAT_SIG_CODE:
                         ;              DEFS 1
                         ;TIMER_LENGTH:
                         ;              DEFS 2

                                        PROG

103E                     REQUEST_SIG_PARAM:
103E 00020001                           DEFW 2,1,2
1042 0002

1044                     REQUEST_SIGNALQ:
1044 01103E                             LD   BC,REQUEST_SIG_PARAM
1047 11738F                             LD   DE,REPEAT_SIG_CODE
104A CD0098                             CALL PARAM
104D 2A73C0                             LD   HL,[TIMER_LENGTH]
1050 3A738F                             LD   A,[REPEAT_SIG_CODE]

1053                     REQUEST_SIGNAL_:
1053 4F                                 LD   C,A
1054 EB                                 EX   DE,HL                   ;Put Repeat Code into C register
1055 2A73D3                             LD   HL,[TIMER_TABLE_BASE]   ;Put length of timer into DE
1058 AF                                 XOR  A                       ;Get Timer Base Address
1059 47                                 LD   B,A                     ;Init offset to First Table value
105A                     TIMER1:
105A CB6E                               BIT  FREE,[HL]
105C 283E                               JR   Z,NEXT_TIMER1           ;See if current timer free
105E E5                                 PUSH HL                      ;If not go get the next timer
                         ;              PUSH AF
105F 7E                                 LD   A,[HL]
1060 E610                               AND  10H
1062 F620                               OR   20H
1064 77                                 LD   [HL],A
1065 AF                                 XOR  A
                         ;              POP  AF
1066 B2                                 OR   D
1067 2008                               JR   NZ,LONG_TIMER
                         ;              RES  FREE,[HL]
                         ;              OR   D                       ;Reset free bit
                         ;              JR   NZ,LONG_TIMER           ;Check for zero
                         ;              RES  REPEAT,[HL]             ;If non_zero then its a long timer
                         ;              RES  LONG,[HL]               ;Set for a NON_Repeating timer
                         ;              LD   A,C
1069 B1                                 OR   C                       ;Check for a short repeating timer
106A 2802                               JR   Z,NOT_A_REPEAT_TIMER
106C CBF6                               SET  REPEAT,[HL]             ;Don't reset repeat bit in mode byte if non_repeating
106E                     NOT_A_REPEAT_TIMER:                         ;Set repeat bit
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  106E 23         4366              INC  HL                          ;Go to next table location
  106F 73         4367              LD   [HL],E                      ;Store timer length
  1070 23         4368              INC  HL
  1071 73         4369              LD   [HL],E                      ;Store timer length again in case of repeat
  1072 1842       4370              JR   INIT_TIMER__EXIT            ;All done so let's exit
  1074            4371 LONG_TIMER:
  1074 CBDE       4372              SET  LONG,[HL]                   ;Set long timer bit
  1076 79         4373              LD   A,C                         ;Check for a long repeat timer
  1077 B7         4374              OR   A
  1078 281B       4375              JR   Z,NOT_A_LONG_REPEAT         ;If zero then go to section for non_repeating timer
  107A D5         4376              PUSH DE                          ;Store timer length temporarily
  107B EB         4377              EX   DE,HL                       ;Swap regisers
  107C 2A7305     4378              LD   HL,[NEXT_TIMER_DATA_BYTE]   ;To get free space in long timer table
  107F EB         4379              EX   DE,HL                       ;Then swap back
  1080 CBF6       4380              SET  REPEAT,[HL]                 ;Set mode byte to repeating
  1082 23         4381              INC  HL
  1083 73         4382              LD   [HL],E                      ;Store low byte of timer address into the value word
  1084 23         4383              INC  HL
  1085 72         4384              LD   [HL],D                      ;Store high byte of timer address
  1086 EB         4385              EX   DE,HL                       ;Move address of data area into HL
  1087 D1         4386              POP  DE                          ;Get back the length of timer
  1088 73         4387              LD   [HL],E                      ;Store that in the data table
  1089 23         4388              INC  HL
  108A 72         4389              LD   [HL],D
  108B 23         4390              INC  HL
  108C 73         4391              LD   [HL],E                      ;Store it again
  108D 23         4392              INC  HL
  108E 72         4393              LD   [HL],D
  108F 23         4394              INC  HL
  1090 227305     4395              LD   [NEXT_TIMER_DATA_BYTE],HL   ;Store the next available data area for future use
  1093 1821       4396              JR   INIT_TIMER__EXIT
  1095            4397 NOT_A_LONG_REPEAT:
  1095 23         4398              INC  HL
  1096            4399 TIMER2:
  1096 73         4400              LD   [HL],E                      ;Store it again
  1097 23         4401              INC  HL
  1098 72         4402              LD   [HL],D
  1099 23         4403              INC  HL
  109A 181A       4404              JR   INIT_TIMER__EXIT
                  4405
  109C            4406 NEXT_TIMER1:
  109C CB66       4407              BIT  EOT,[HL]
  109E 2006       4408              JR   NZ,MAKE_NEW_TIMER
  10A0 23         4409              INC  HL                          ;Go to next mode byte
  10A1 23         4410              INC  HL                          ;
  10A2 23         4411              INC  HL                          ;
  10A3 04         4412              INC  B                           ;Count to next offset
  10A4 1884       4413              JR   TIMER1                      ;Go back up to init. tomer
                  4414
  10A6            4415 MAKE_NEW_TIMER:                ;Maximum of 255 signals allowed
  10A6 D5         4416              PUSH DE                          ;Save DE for a work register
  10A7 E5         4417              PUSH HL                          ;Save current timer address
  10A8 23         4418              INC  HL                          ;Go to next available memory location in the Timer Table
  10A9 23         4419              INC  HL
  10AA 23         4420              INC  HL
  10AB 04         4421              INC  B                           ;Increment the Signal count
  10AC 3630       4422              LD   [HL],30H                    ;Set to free and last timer
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  10AE EB            4423              EX    DE,HL                          ;Save momentarily
  10AF E1            4424              POP   HL                             ;Get back original timer
  10B0 CBA6          4425              RES   EOT,(HL)                       ;Reset previous last timer
  10B2 EB            4426              EX    DE,HL                          ;Get back current last timer
  10B3 D1            4427              POP   DE                             ;Restore DE register
  10B4 18A4          4428              JR    TIMER1                         ;Go back up and initialize counter for use
                     4429
  10B6               4430 INIT_TIMER__EXIT:
  10B6 E1            4431              POP   HL
  10B7 CBAE          4432              RES   FREE,(HL)
  10B9 78            4433              LD    A,B            ;Put the offset into the Accumulator for the user of routine
  10BA C9            4434              RET
                     4435
                     4436
                     4437 ;Procedure Test Signal
                     4438 ;Acc has the Signal number to be tested
                     4439 ;A value of True(1) or False(0) is returned in the Accumulator for the
                     4440 ;                  Signal given.
                     4441 ;            CONN
                     4442 ;TEST_SIG_NUM:
                     4443 ;            DEFS 1
                     4444
                     4445              PROG
  10BB               4446 TEST_SIG_PARAM:
  10BB 00010001      4447              DEFW 1,1
                     4448
  10BF               4449 TEST_SIGNALQ
  10BF 0110BB        4450              LD    BC,TEST_SIG_PARAM
  10C2 1173C2        4451              LD    DE,TEST_SIG_NUM
  10C5 CD009B        4452              CALL  PARAM
  10C8 3A73C2        4453              LD    A,(TEST_SIG_NUM)
                     4454
  10CB               4455 TEST_SIGNAL_
  10CB 4F            4456              LD    C,A                            ;Put Signal Code into C register
  10CC 2A7303        4457              LD    HL,(TIMER_TABLE_BASE)          ;Get Timer Base Address
  10CF 47            4458              LD    B,A                            ;Save Signal
  10D0 110003        4459              LD    DE,3                           ;Set up offset for next timer
  10D3 B7            4460              OR    A                              ;See if first timer is a match
  10D4 2808          4461              JR    Z,SIGNAL_MATCH                 ;If so go check it
                     4462
  10D6               4463 TEST1:                                              ;Loop to match timer table to desired timer
  10D6 CB66          4464              BIT   EOT,(HL)                       ;Check for end of table
  10D8 200C          4465              JR    NZ,SIGNAL_FALSE                ;If so then return a not done
  10DA 19            4466              ADD   HL,DE                          ;Now index to next timer
  10DB 0D            4467              DEC   C                              ;Decrement to the timer desired
  10DC 20F8          4468              JR    NZ,TEST1                       ;If not a timer desired timer then go back
                     4469
  10DE               4470 SIGNAL_MATCH:                                       ;Here with a timer match
  10DE CB6E          4471              BIT   FREE,(HL)
  10E0 2004          4472              JR    NZ,SIGNAL_FALSE
  10E2 CB7E          4473              BIT   DONE,(HL)                      ;Check for timer done
  10E4 2003          4474              JR    NZ,SIGNAL_TRUE                 ;If so then go return a True
  10E6               4475 SIGNAL_FALSE:                                       ;Here to return a false for either
                     4476                                                   ;a not done or non_existent timer
  10E6 AF            4477              XOR   A                              ;Put a false in Acc
  10E7 180A          4478              JR    TEST_EXIT                      ;Go to the exit
                     4479
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  10E9              4480 SIGNAL_TRUE:                              ;Here when timer is finished
  10E9 CB76         4481             BIT  REPEAT,[HL]              ;Check for repeating timer
  10EB 2002         4482             JR   NZ,SIGNAL_TRUE1          ;If so then just return True
  10ED CBEE         4483             SET  FREE,[HL]                ;Free current timer since not repeating
  10EF              4484 SIGNAL_TRUE1:
                    4485                                           ;***** Start add 4/30/82****
  10EF CB8E         4486             RES  DONE,[HL]                ;Reset current timer to not done
                    4487                                           ;***** End add 4/30/82****
  10F1 3E01         4488             LD   A,1                      ;Put a True in the Acc
  10F3              4489 TEST_EXIT:                                ;Return
  10F3 B7           4490             OR   A
  10F4 C9           4491             RET
                    4492
                    4493             DATA
                    4494
  7303              4495 TIMER_TABLE_BASE:     DEFS 2
  7305              4496 NEXT_TIMER_DATA_BYTE: DEFS 2
                    4497
                    4498  PROG
```

```
LOCATION OBJECT CODE LINE    SOURCE LINE

                    ;CONTROLLER SOFTWARE
         <0000>     FIRE             EQU        0           ;BITS IN STATUS WORD TO CHECK
         <0001>     JOY              EQU        1           ;WHETHER DEVICE IS ACTIVE
         <0002>     SPIN             EQU        2
         <0003>     ARM              EQU        3
         <0004>     KBD              EQU        4
         <0002>     PLYR_0           EQU        2           ;OFFSET TO CONTROLLER MEMORY
         <0007>     PLYR_1           EQU        7
         <0007>     SEG_0            EQU        07H         ;BIT MASK IN PLAYER STATUS
         <0018>     SEG_1            EQU        18H         ;FOR SEGMENT 0 OR 1 DEVICES
         <0000>     FIRE_OLD         EQU        0           ;OFFSETS TO DEBOUNCE STATUS
         <0001>     FIRE_STATE       EQU        1           ;BUFFER: (OLD,STATE)
         <0002>     JOY_OLD          EQU        2
         <0003>     JOY_STATE        EQU        3
         <0004>     SPIN_OLD         EQU        4
         <0005>     SPIN_STATE       EQU        5
         <0006>     ARM_OLD          EQU        6
         <0007>     ARM_STATE        EQU        7
         <0008>     KBD_OLD          EQU        8
         <0009>     KBD_STATE        EQU        9
         <000F>     KBD_MASK         EQU        0FH      ;MASK FOR INPUT DATA BYTE
         <0040>     FIRE_MASK        EQU        40H
         <0040>     ARM_MASK         EQU        40H
         <000F>     JOY_MASK         EQU        0FH
         <0080>     SPNR_MASK        EQU        10110000B
         <0005>     NUM_DEV          EQU        5        ;NUMBER OF POSSIBLE DEVICES
                    ;STACK            EQU        73FFH
                    ;MODE_0_PORT      EQU        0BEH        ;D8
                    ;MODE_1_PORT      EQU        0BFH        ;D9
         <000F>     KBD_NULL         EQU        0FH
         <00FF>     CTRL_1_PORT      EQU        0FFH
         <00FC>     CTRL_0_PORT      EQU        0FCH
         <00C0>     STRB_RST_PORT    EQU        0C0H              ;STROBE RESET PORT
         <0080>     STRB_SET_PORT    EQU        080H              ;STROBE SET PORT
         <0000>     CONTROLLER_0     EQU        0
         <0001>     CONTROLLER_1     EQU        1
         <0000>     STROBE_RESET     EQU        0
         <0001>     STROBE_SET       EQU        1
                    ************************* MACRO **********************************


                    DELAY_10    MACRO
                                CALL DELAY
                                MEND

                    ********************************DATA******************************
                    *
                    *  DECODER TABLE FOR THE KEYBOARD
                    *
10F5                DEC_KBD_TBL
10F5 0F                         DEFB KBD_NULL       ; NULL ENTRY
10F6 06                         DEFB 6              ; '6'
10F7 01                         DEFB 1              ; '1'
10F8 03                         DEFB 3              ; '3'
10F9 09                         DEFB 9              ; '9'
10FA 00                         DEFB 0              ; '0'
10FB 0A                         DEFB 10             ; '*'
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

10FC 0F                           DEFB KBD_NULL        ; NULL ENTRY
10FD 02                           DEFB 2               ; '2'
10FE 08                           DEFB 11              ; RESET
10FF 07                           DEFB 7               ; '7'
1100 0F                           DEFB KBD_NULL        ; NULL ENTRY
1101 05                           DEFB 5               ; '5'
1102 04                           DEFB 4               ; '4'
1103 08                           DEFB 8               ; '8'
1104 0F                           DEFB KBD_NULL        ; NULL ENTRY

                    ******************************SUBROUTINES**********************

                                       GLB             CONTROLLER_INIT
1105                 CONTROLLER_INIT                   ;INITIALIZE CONTROLLER TO STROBE RESET
1105 D3C0                          OUT  [STRB_RST_PORT],A
1107 AF                            XOR  A
1108 DD2A8008                      LD   IX,[CONTROLLER_MAP]
110C DD23                          INC  IX
110E DD23                          INC  IX
1110 FD2173D7                      LD   IY,DBNCE_BUFF
1114 060A                          LD   B,NUM_DEV*2
                     *  CLEAR CONTROLLER MEMORY AND DEBOUNCE STATUS BUFFER
1116                 CINIT1
1116 DD7700                        LD   [IX+0],A
1119 DD23                          INC  IX
111B FD7700                        LD   [IY+0],A
111E FD23                          INC  IY
1120 FD7700                        LD   [IY+0],A
1123 FD23                          INC  IY
1125 05                            DEC  B
1126 20EE                          JR   NZ,CINIT1
                     * CLEAR REMAINING VARIABLES
1128 3273EB                        LD   [SPIN_SW0_CT],A
112B 3273EC                        LD   [SPIN_SW1_CT],A
112E 3273EE                        LD   [S0_C0],A
1131 3273EF                        LD   [S0_C1],A
1134 3273F0                        LD   [S1_C0],A
1137 3273F1                        LD   [S1_C1],A
113A C9                            RET



113B 00              DELAY         NOP                        ;DELAY AFTER STROBE, BEFORE READ
113C C9                            RET
                     *
                     *  CONTROLLER READ ROUTINE
                     *                  INPUT:
                     *                      H - CONTROLLER NUMBER
                     *                  OUTPUT:
                     *                      A - RAW DATA
                     *

113D                 CONT_READ
113D 7C                            LD   A,H
113E FE00                          CP   CONTROLLER_0              ;IF CONTROLLER<>0
1140 2004                          JR   NZ,CONT_READ1             ;THEN READ PLAYER 1
1142 [illegible]                   IN   A,[CTRL_0_PORT]           ;ELSE READ PLAYER 0
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

    1144 1802         4614              JR    CONT_READX
    1146              4615 CONT_READ1
    1146 DBFF         4616              IN    A,[CTRL_1_PORT]
    1148              4617 CONT_READX
    1148 2F           4618              CPL                            ;COMPLEMENT DATA SO 0=NO DATA
    1149 C9           4619              RET                                                 ;1= DATA
                      4620
                      4621
                      4622 *
                      4623 *  CONTROLLER SCANNER ROUTINE
                      4624 *
                      4625              GLB              CONT_SCAN
    114A              4626 CONT_SCAN
    114A DBFC         4627              IN    A,[CTRL_0_PORT]         ;READ SEGMENT 0, BOTH PLAYERS
    114C 2F           4628              CPL
    114D 3273EE       4629              LD    [S0_C0],A
    1150 DBFF         4630              IN    A,[CTRL_1_PORT]
    1152 2F           4631              CPL
    1153 3273EF       4632              LD    [S0_C1],A
    1156 D380         4633              OUT   [STRB_SET_PORT],A       ;STROBE SEGMENT 1
    1158              4634              DELAY_10                      ;WAIT 10 MICROSECS
    1158 CD1138          +              CALL  DELAY
    115B DBFC         4635              IN    A,[CTRL_0_PORT]         ;READ SEGMENT 1, BOTH PLAYERS
    115D 2F           4636              CPL
    115E 3273F0       4637              LD    [S1_C0],A
    1161 DBFF         4638              IN    A,[CTRL_1_PORT]
    1163 2F           4639              CPL
    1164 3273F1       4640              LD    [S1_C1],A
    1167 D3C0         4641              OUT   [STRB_RST_PORT],A       ;RESET TO SEGMENT 0
    1169 C9           4642              RET
                      4643
                      4644
                      4645 *
                      4646 *            UPDATE SPINNER SWITCH ROUTINE                   *
                      4647 *
                      4648 *
                      4649
                      4650
                      4651
                      4652              GLB              UPDATE_SPINNER_
    116A              4653 UPDATE_SPINNER_
    116A DBFC         4654              IN    A,[CTRL_0_PORT]         ;GET DATA
    116C 2173EB       4655              LD    HL,SPIN_SW0_CT          ;ADDRESS OF SPINNER 0 COUNT
    116F CB67         4656              BIT   4,A                     ;IF INT BIT SET
    1171 2008         4657              JR    NZ,UPDATE_S1            ;THEN SPINNER 1
                      4658 *  CHECK DIRECTION                      ; ELSE SPINNER 0
    1173 CB6F         4659              BIT   5,A                     ; IF BIT 5 IS  SET
    1175 2003         4660              JR    NZ,UPDATE_R0            ; THEN GOING RIGHT
                      4661                                            ; ELSE LEFT
    1177 35           4662              DEC   [HL]                    ; DECREMENT SPINNER COUNTER
    1178 1801         4663              JR    UPDATE_S1               ; GO CHECK SPINNER 1
                      4664 ***  RIGHT SPINNER SWITCH
    117A              4665 UPDATE_R0
    117A 34           4666              INC   [HL]                    ;RIGHT, INCREMENT COUNTER
                      4667 * CHECK SPINNER 1
    117B DBFF         4668 UPDATE_S1   IN    A,[CTRL_1_PORT]          ;LOOK AT SPINNER 1 DATA
    117D CB67         4669              BIT   4,A                     ;IF INT BIT  SET
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  117F 2009        4670            JR   NZ,UPDATE_SPINX          ;THEN NOT SPINNER 1
  1181 23          4671            INC  HL                       ;ELSE SPINNER 1, BUMP HL
                   4672 *  CHECK DIRECTION
  1182 CB6F        4673            BIT  5,A                      ; IF BIT 5 IS SET
  1184 2003        4674            JR   NZ,UPDATE_R1             ; THEN GOING RIGHT
                   4675                                          ; ELSE LEFT
  1186 35          4676            DEC  [HL]                     ; DECREMENT SPINNER COUNTER
  1187 1801        4677            JR   UPDATE_SPINX
                   4678 ***  RIGHT SPINNER SWITCH
  1189             4679 UPDATE_R1
  1189 34          4680            INC  [HL]                     ;RIGHT, INCREMENT COUNTER
  118A             4681 UPDATE_SPINX
  118A C9          4682            RET
                   4683
                   4684
                   4685
                   4686 **************  DECODER ROUTINE  ************************
                   4687 *  THIS ROUTINE RETURNS DECODED RAW, UNDEBOUNCED DATA           *
                   4688 *  AND MAY OR MAY NOT BE REQUIRED BY O/S                       *
                   4689 *
                   4690 *          INPUT:
                   4691 *               H - CONTROLLER NUMBER
                   4692 *               L - SEGMENT NUMBER
                   4693 *          OUTPUT:
                   4694 *                              SEGMENT 0       SEGMENT 1
                   4695 *               H - BYTE 1     FIRE              ARM
                   4696 *               L - BYTE 2     JOYSTK            KBD
                   4697 *               E - BYTE 3     SPINNER
                   4698 *
                   4699 *
                   4700            GLB              DECODER_
  118B             4701 DECODER_
  118B 7D          4702            LD   A,L
  118C FE01        4703            CP   STROBE_SET               ;IF L=1 THEN DECODE SEGMENT 1
  118E 281A        4704            JR   Z,DEC_SEG1
                   4705 *
                   4706 *  SEGMENT 0 (FIRE BUTTON, JOYSTICK)
                   4707 *  RETURN H=FIRE BUTTON, L=JOYSTICK, E=SPINNER
                   4708 *
                   4709 * DO SPINNER FIRST
  1190 0173EB      4710            LD   BC,SPIN_SW0_CT
  1193 7C          4711            LD   A,H
  1194 FE00        4712            CP   CONTROLLER_0             ;IF PLAYER=0 THEN GO DECODE
  1196 2801        4713            JR   Z,DEC_PLYR
  1198 03          4714            INC  BC                       ;ELSE INCREMENT BC TO SPINNER 1
  1199 0A          4715 DEC_PLYR   LD   A,[BC]                   ;GET SPINNER SWITCH COUNT
  119A 5F          4716            LD   E,A                      ;RETURN IT IN E
  119B AF          4717            XOR  A
  119C 02          4718            LD   [BC],A                   ;CLEAR OUT SPINNER SWITCH COUNT
                   4719
  119D CD1130      4720            CALL CONT_READ                ;GET OTHER DEVICE DATA FOR PLAYER
  11A0 57          4721            LD   D,A                      ;SAVE IT
  11A1 E60F        4722            AND  JOY_MASK                 ;MASK OUT JOYSTICK DATA
  11A3 6F          4723            LD   L,A                      ;RETURN IT IN L
                   4724
  11A4 7A          4725            LD   A,D                      ;RESTORE DATA
  11A5 E640        4726            AND  FIRE_MASK                ;MASK OUT FIRE BUTTON DATA
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  11A7 67           4727              LD   H,A                  ;RETURN IT IN H
  11A8 1816         4728              JR   DECODERX
                    4729 *
                    4730 * SEGMENT 1 (ARM BUTTON, KEYBOARD)
                    4731 *  RETURN H=ARM BUTTON, L=KEYBOARD
                    4732 *
  11AA              4733 DEC_SEG1
  11AA D380         4734              OUT  [STRB_SET_PORT],A      ;STROBE SEGMENT 1
  11AC CD113D       4735              CALL CONT_READ              ;READ SEGMENT 1 PLAYER DATA
  11AF 57           4736              LD   D,A                    ;SAVE IT
  11B0 D3C0         4737              OUT  [STRB_RST_PORT],A      ;RESET BACK TO SEGMENT 0
  11B2 E60F         4738              AND  KBD_MASK               ;MASK OUT KBD DATA
  11B4 2110F5       4739              LD   HL,DEC_KBD_TBL         ;GET DECODER TABLE ADDRESS
  11B7 0600         4740              LD   B,0
  11B9 4F           4741              LD   C,A
  11BA 09           4742              ADD  HL,BC                  ;COMPUTE OFFSET
  11BB 6E           4743              LD   L,[HL]                 ;RETURN KBD DATA IN L
                    4744
  11BC 7A           4745              LD   A,D                    ;RESTORE DATA
  11BD E640         4746              AND  ARM_MASK               ;MASK OUT ARM BUTTON DATA
  11BF 67           4747              LD   H,A                    ;RETURN IT IN H
  11C0              4748 DECODERX
  11C0 C9           4749              RET
                    4750
                    4751
                    4752
                    4753
                    4754 *  POLLING ROUTINE FOR ALL DEVICES IN CONTROLLER *
                    4755 *
                    4756 *
                    4757
                    4758                   GLB            POLLER_
  11C1              4759 POLLER_
  11C1 CD114A       4760              CALL CONT_SCAN              ;GO SCAN ALL THE DATA FIRST
  11C4 FD2173D7     4761              LD   IY,DBNCE_BUFF          ;DEBOUNCE BUFFER POINTER
  11C8 DD2A8008     4762              LD   IX,[CONTROLLER_MAP]    ;CONTROLLER MEMORY POINTER
  11CC DDE5         4763              PUSH IX
  11CE DD7E00       4764              LD   A,[IX+0]          ;GET PLAYER 0 STATUS
  11D1 CB7F         4765              BIT  7,A               ;IF PLAYER 0 NOT ACTIVE
  11D3 281E         4766              JR   Z,CHK_PLYR_1      ;THEN CHECK PLAYER 1
                    4767 *  PLAYER 0 IS ACTIVE              ;ELSE
  11D5 47           4768              LD   B,A               ;SAVE STATUS
  11D6 110002       4769              LD   DE,PLYR_0         ;COMPUTE ADDRESS OF PLAYER_0
  11D9 DD19         4770              ADD  IX,DE             ;CONTROLLER_MEMORY
  11DB E607         4771              AND  SEG_0             ;IF SEGMENT 0 IS NOT ACTIVE
  11DD 2809         4772              JR   Z,CHK_SEG_01      ;THEN CHECK SEGMENT 1
                    4773 *  SEGMENT 0 ACTIVE                ;ELSE
  11DF 3A73EE       4774              LD   A,[S0_C0]
  11E2 2173EB       4775              LD   HL,SPIN_SW0_CT
  11E5 CD1220       4776              CALL DECODE_0          ;DECODE DATA FOR SEGMENT 0
  11E8              4777 CHK_SEG_01
  11E8 78           4778              LD   A,B               ;RESTORE PLAYER_0 STATUS
  11E9 E618         4779              AND  SEG_1             ;IF SEGMENT 1 IS NOT ACTIVE
  11EB 2806         4780              JR   Z,CHK_PLYR_1      ;THEN CHECK PLAYER 1
                    4781 *  SEGMENT 1 IS ACTIVE             ;ELSE
  11ED 3A73F0       4782              LD   A,[S1_C0]
  11F0 CD123F       4783              CALL DECODE_1          ;DECODE DATA FOR SEGMENT 1
```

```
CATION OBJECT CODE LINE     SOURCE LINE

  11F3              4784 CHK_PLYR_1
  11F3 DDE1         4785            POP  IX
  11F5 DD7E01       4786            LD   A,[IX+1]       ;GET PLAYER 1 STATUS
  11F8 CB7F         4787            BIT  7,A            ;IF PLAYER 1 IS NOT ACTIVE
  11FA 2823         4788            JR   Z,POLLER_X      ;THEN EXIT, ALL DONE
                    4789 *  PLAYER 1 IS ACTIVE
  11FC 47           4790            LD   B,A            ;SAVE PLAYER 1 STATUS
  11FD 11000A       4791            LD   DE,2*NUM_DEV   ;COMPUTE ADDRESS OF DEBOUNCE BUFFER
  1200 FD19         4792            ADD  IY,DE          ;FOR PLAYER 1
  1202 110007       4793            LD   DE,PLYR_1      ;COMPUTE ADDRESS OF CONTROLLER_MEMORY
  1205 DD19         4794            ADD  IX,DE          ;FOR PLAYER 1
  1207 E607         4795            AND  SEG_0          ;IF SEGMENT 0 IS NOT ACTIVE
  1209 2809         4796            JR   Z,CHK_SEG_11   ;THEN CHECK SEGMENT 1
                    4797 *  SEGMENT 0 IS ACTIVE         ;ELSE
  120B 3A73EF       4798            LD   A,[S0_C1]
  120E 2173EC       4799            LD   HL,SPIN_SW1_CT
  1211 CD1220       4800            CALL DECODE_0       ;DECODE DATA FOR SEGMENT 0
  1214              4801 CHK_SEG_11
  1214 78           4802            LD   A,B            ;RESTORE STATUS FOR PLAYER 1
  1215 E618         4803            AND  SEG_1          ;IF SEGMENT 1 IS NOT ACTIVE
  1217 2806         4804            JR   Z,POLLER_X      ;THEN EXIT, ALL DONE
                    4805 *  SEGMENT 1 IS ACTIVE         ;ELSE
  1219 3A73F1       4806            LD   A,[S1_C1]
  121C CD123F       4807            CALL DECODE_1       ;DECODE DATA FOR SEGMENT 1
  121F              4808 POLLER_X
  121F C9           4809            RET
                    4810
                    4811 *  DECODER ROUTINE FOR SEGMENT 0
                    4812 *
                    4813 *          INPUT:
                    4814 *                A - DATA
                    4815 *                B - DEVICE STATUS BYTE FOR CURRENT PLAYER
                    4816 *               HL - ADDRESS OF SPINNER DATA
                    4817 *               IX - POINTER TO CONTROLLER MEMORY
                    4818 *               IY - POINTER TO DEBOUNCE STATUS BUFFER
                    4819 *
  1220              4820 DECODE_0
  1220 4F           4821            LD   C,A            ;SAVE DATA
  1221 CB48         4822            BIT  JOY,B          ;IF JOYSTICK NOT ACTIVE
  1223 2804         4823            JR   Z,DEC_FIRE     ;THEN CHECK FIRE BUTTON
                    4824 *  JOYSTICK ACTIVE
  1225 CD12B9       4825            CALL JOY_DBNCE      ;DEBOUNCE JOYSTICK DATA
  1228 79           4826            LD   A,C
  1229              4827 DEC_FIRE
  1229 CB40         4828            BIT  FIRE,B         ;IF FIRE BUTTON NOT ACTIVE
  122B 2804         4829            JR   Z,DEC_SPNR     ;THEN CHECK SPINNER
                    4830 *  FIRE BUTTON ACTIVE          ;ELSE
  122D CD1289       4831            CALL FIRE_DBNCE     ;DEBOUNCE FIRE BUTTON
  1230 79           4832            LD   A,C
  1231              4833 DEC_SPNR
  1231 CB50         4834            BIT  SPIN,B         ;IF SPINNER NOT ACTIVE
  1233 2809         4835            JR   Z,DECODE_0X    ;THEN EXIT DECODER
                    4836 *  SPINNER ACTIVE
  1235 7E           4837            LD   A,[HL]         ;SAVE SPINNER COUNT
  1236 DD8602       4838            ADD  A,[IX+SPIN]
  1239 DD7702       4839            LD   [IX+SPIN],A    ;IN CONTROLLER MEMORY
  123C AF           4840            XOR  A
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  123D 77                                LD   [HL],A           ;CLEAR COUNTER
  123E                       DECODE_0X
  123E C9                                RET


                             *  DECODER ROUTINE FOR SEGMENT 1
                             *
                             *           INPUT:
                             *                A - DATA
                             *                B - DEVICE STATUS BYTE FOR CURRENT PLAYER
                             *               IX - POINTER TO CONTROLLER MEMORY
                             *               IY - POINTER TO DEBOUNCE STATUS BUFFER
                             *
  123F                       DECODE_1
  123F 4F                                LD   C,A              ;SAVE DATA
  1240 CB58                              BIT  ARM,B            ;IF ARM BUTTON NOT ACTIVE
  1242 2804                              JR   Z,DEC_KBD        ;THEN CHECK KEYBOARD
                             *  ARM BUTTON ACTIVE              ;ELSE
  1244 CD12E9                            CALL ARM_DBNCE        ;DEBOUNCE ARM BUTTON
  1247 79                                LD   A,C
  1248                       DEC_KBD
  1248 CB60                              BIT  KBD,B            ;IF KEYBOARD NOT ACTIVE
  124A 2803                              JR   Z,DECODE_1X      ;THEN EXIT DECODER
                             * KBD ACTIVE
  124C CD1250                            CALL KBD_DBNCE        ;DEBOUNCE KEYBOARD  -
  124F                       DECODE_1X
  124F C9                                RET



                             *  KEYBOARD DEBOUNCE ROUTINE  *
                             *
                             *           INPUT:
                             *                A - RAW DATA
                             *               IX - CONTROLLER MEMORY POINTER
                             *               IY - DEBOUNCE STATE BUFFER
                             *
  1250                       KBD_DBNCE
  1250 C5                                PUSH BC
  1251 D5                                PUSH DE
  1252 E5                                PUSH HL
  1253 E60F                              AND  KBD_MASK         ;MASK OUT VALID DATA
  1255 5F                                LD   E,A              ;SAVE IT
  1256 FD4608                            LD   B,[IY+KBD_OLD]   ;GET OLD DATA AND CURREN STATE
  1259 FD7E09                            LD   A,[IY+KBD_STATE]
  125C FE00                              CP   0                ;IF STATE <> 0
  125E 201A                              JR   NZ,KBD_ST1       ;THEN MUST BE STATE 1
                             *  STATE = 0                      ;ELSE
  1260 7B                                LD   A,E              ;GET CURRENT DATA
  1261 B8                                CP   B                ;IF OLD=NEW
  1262 2805                              JR   Z,KBD_REG        ;THEN SAW DATA TWICE IN SEQUENCE
  1264 FD7308                            LD   [IY+KBD_OLD],E   ;ELSE FIRST TIME, SAVE CURRENT DATA
  1267 181C                              JR   KBD_EXIT
                             *  SAW DATA TWICE IN SEQUENCE
  1269                       KBD_REG
  1269 3E01                              LD   A,1              ;SET STATE=1
  1268 FD7709                            LD   [IY+KBD_STATE],A
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

                    4898 *  DECODE KEYBOARD DATA
126E 2110F5         4899            LD   HL,DEC_KBD_TBL   ;DECODE TABLE ADDRESS
1271 1600           4900            LD   D,0              ;D/E RAW DATA
1273 19             4901            ADD  HL,DE            ;COMPUTE ADDRESS INTO TABLE
1274 7E             4902            LD   A,[HL]           ;DO TABLE LOOKUP
1275 DD7704         4903            LD   [IX+KBD],A       ;SAVE IN CONTROLLER MEMORY @KBD
1278 1808           4904            JR   KBD_EXIT
                    4905 *  STATE = 1
127A                4906 KBD_ST1
127A 7B             4907            LD   A,E              ;GET CURRENT DATA
127B B8             4908            CP   B                ;IF OLD=NEW
127C 2807           4909            JR   Z,KBD_EXIT       ;NO CHANGE IN STATE
127E FD7308         4910            LD   [IY+KBD_OLD],E   ;ELSE SAVE CURRENT DATA
1281 AF             4911            XOR  A                ;SET STATE=0
1282 FD7709         4912            LD   [IY+KBD_STATE],A
1285                4913 KBD_EXIT
1285 E1             4914            POP  HL
1286 D1             4915            POP  DE
1287 C1             4916            POP  BC
1288 C9             4917            RET
                    4918
                    4919 *  FIRE BUTTON DEBOUNCE ROUTINE  *
                    4920 *
                    4921 *          INPUT:
                    4922 *                A  - RAW DATA
                    4923 *                IX - CONTROLLER MEMORY POINTER
                    4924 *                IY - DEBOUNCE STATE BUFFER
                    4925 *
1289                4926 FIRE_DBNCE
1289 C5             4927            PUSH BC
128A D5             4928            PUSH DE
128B E640           4929            AND  FIRE_MASK        ;MASK OUT VALID DATA
128D 5F             4930            LD   E,A              ;SAVE IT
128E FD4600         4931            LD   B,[IY+FIRE_OLD]  ;GET OLD DATA AND CURRENT STATE
1291 FD7E01         4932            LD   A,[IY+FIRE_STATE]
1294 FE00           4933            CP   0                ;IF STATE <> 0
1296 2013           4934            JR   NZ,FIRE_ST1      ;THEN MUST BE STATE 1
                    4935 *  STATE = 0                     ;ELSE
1298 7B             4936            LD   A,E              ;GET CURRENT DATA
1299 B8             4937            CP   B                ;IF OLD=NEW
129A 2805           4938            JR   Z,FIRE_REG       ;THEN SAW DATA TWICE IN SEQUENCE
129C FD7300         4939            LD   [IY+FIRE_OLD],E  ;ELSE FIRST TIME, SAVE CURRENT DATA
129F 1815           4940            JR   FIRE_EXIT
                    4941 *  SAW DATA TWICE IN SEQUENCE
12A1                4942 FIRE_REG
12A1 3E01           4943            LD   A,1              ;SET STATE=1
12A3 FD7701         4944            LD   [IY+FIRE_STATE],A
12A6 DD7300         4945            LD   [IX+FIRE],E      ;SAVE IN CONTROLLER MEMORY @FIRE
12A9 1808           4946            JR   FIRE_EXIT
                    4947 *  STATE = 1
12AB                4948 FIRE_ST1
12AB 7B             4949            LD   A,E              ;GET CURRENT DATA
12AC B8             4950            CP   B                ;IF OLD=NEW
12AD 2807           4951            JR   Z,FIRE_EXIT      ;NO CHANGE IN STATE
12AF FD7300         4952            LD   [IY+FIRE_OLD],E  ;ELSE SAVE CURRENT DATA
12B2                 53            A                      1 ST.
12B3 FD7701         4954            LD   [IY+FIRE_STATE],A
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  12B6              4955 FIRE_EXIT
  12B6 D1           4956            POP  DE
  12B7 C1           4957            POP  BC
  12B8 C9           4958            RET
                    4959
                    4960 *  JOYSTICK DEBOUNCE ROUTINE  *
                    4961 *
                    4962 *          INPUT:
                    4963 *              A - RAW DATA
                    4964 *              IX - CONTROLLER MEMORY POINTER
                    4965 *              IY - DEBOUNCE STATE BUFFER
                    4966 *
  12B9              4967 JOY_DBNCE
  12B9 C5           4968            PUSH BC
  12BA D5           4969            PUSH DE
  12BB E60F         4970            AND  JOY_MASK          ;MASK OUT VALID DATA
  12BD 5F           4971            LD   E,A               ;SAVE IT
  12BE FD4602       4972            LD   B,[IY+JOY_OLD]    ;GET OLD DATA AND CURRENT STATE
  12C1 FD7E03       4973            LD   A,[IY+JOY_STATE]
  12C4 FE00         4974            CP   0                 ;IF STATE <> 0
  12C6 2013         4975            JR   NZ,JOY_ST1        ;THEN MUST BE STATE 1
                    4976 *  STATE = 0                      ;ELSE
  12C8 7B           4977            LD   A,E               ;GET CURRENT DATA
  12C9 B8           4978            CP   B                 ;IF OLD=NEW
  12CA 2805         4979            JR   Z,JOY_REG         ;THEN SAW DATA TWICE IN SEQUENCE
  12CC FD7302       4980            LD   [IY+JOY_OLD],E    ;ELSE FIRST TIME, SAVE CURRENT DATA
  12CF 1815         4981            JR   JOY_EXIT
                    4982 *  SAW DATA TWICE IN SEQUENCE
  12D1              4983 JOY_REG
  12D1 3E01         4984            LD   A,1               ;SET STATE=1
  12D3 FD7703       4985            LD   [IY+JOY_STATE],A
  12D6 DD7301       4986            LD   [IX+JOY],E        ;SAVE IN CONTROLLER MEMORY @JOY
  12D9 1808         4987            JR   JOY_EXIT
                    4988 *  STATE = 1
  12DB              4989 JOY_ST1
  12DB 7B           4990            LD   A,E               ;GET CURRENT DATA
  12DC B8           4991            CP   B                 ;IF OLD=NEW
  12DD 2807         4992            JR   Z,JOY_EXIT        ;NO CHANGE IN STATE
  12DF FD7302       4993            LD   [IY+JOY_OLD],E    ;ELSE SAVE CURRENT DATA
  12E2 AF           4994            XOR  A                 ;SET STATE=0
  12E3 FD7703       4995            LD   [IY+JOY_STATE],A
  12E6              4996 JOY_EXIT
  12E6 D1           4997            POP  DE
  12E7 C1           4998            POP  BC
  12E8 C9           4999            RET
                    5000
                    5001 *  ARM BUTTON DEBOUNCE ROUTINE  *
                    5002 *
                    5003 *          INPUT:
                    5004 *              A - RAW DATA
                    5005 *              IX - CONTROLLER MEMORY POINTER
                    5006 *              IY - DEBOUNCE STATE BUFFER
                    5007 *
  12E9              5008 ARM_DBNCE
  12E9 C5           5009            PUSH BC
  12EA D5           5010            PUSH DE
  12EB E640         5011            AND  ARM_MASK          ;MASK OUT VALID DATA
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

  12ED 5F          5012            LD   E,A              ;SAVE IT
  12EE FD4606      5013            LD   B,[IY+ARM_OLD]   ;GET OLD DATA AND CURRENT STATE
  12F1 FD7E07      5014            LD   A,[IY+ARM_STATE]
  12F4 FE00        5015            CP   0                ;IF STATE <> 0
  12F6 2013        5016            JR   NZ,ARM_ST1       ;THEN MUST BE STATE 1
                   5017 *  STATE = 0                     ;ELSE
  12F8 7B          5018            LD   A,E              ;GET CURRENT DATA
  12F9 B8          5019            CP   B                ;IF OLD=NEW
  12FA 2805        5020            JR   Z,ARM_REG        ;THEN SAW DATA TWICE IN SEQUENCE
  12FC FD7306      5021            LD   [IY+ARM_OLD],E   ;ELSE FIRST TIME, SAVE CURRENT DATA
  12FF 1815        5022            JR   ARM_EXIT
                   5023 *  SAW DATA TWICE IN SEQUENCE
  1301             5024 ARM_REG
  1301 3E01        5025            LD   A,1              ;SET STATE=1
  1303 FD7707      5026            LD   [IY+ARM_STATE],A
  1306 DD7303      5027            LD   [IX+ARM],E       ;SAVE IN CONTROLLER MEMORY @ARM
  1309 180B        5028            JR   ARM_EXIT
                   5029 *  STATE = 1
  130B             5030 ARM_ST1
  130B 7B          5031            LD   A,E              ;GET CURRENT DATA
  130C B8          5032            CP   B                ;IF OLD=NEW
  130D 2807        5033            JR   Z,ARM_EXIT       ;NO CHANGE IN STATE
  130F FD7306      5034            LD   [IY+ARM_OLD],E   ;ELSE SAVE CURRENT DATA
  1312 AF          5035            XOR  A                ;SET STATE=0
  1313 FD7707      5036            LD   [IY+ARM_STATE],A
  1316             5037 ARM_EXIT
  1316 D1          5038            POP  DE
  1317 C1          5039            POP  BC
  1318 C9          5040            RET
                   5041
                   5042
                   5043 ;               EXT             CONTROLLER_MAP
                   5044 * THIS IS AN EXTERNAL POINTER ( DEFINED IN THE CARTRIDGE) TO THE
                   5045 * CARTRIDGE PROGRAMMER'S CONTROLLER MAP AREA.
                   5046
                   5047 * THE CARTRIDGE PROGRAMMER IS RESPONSIBLE FOR MAINTAINING THIS AREA.
                   5048
                   5049                 DATA
  73D7             5050 DBNCE_BUFF      DEFS            NUM_DEV*4
  73EB             5051 SPIN_SW0_CT     DEFS            1
  73EC             5052 SPIN_SW1_CT     DEFS            1
  73ED             5053 STROBE_FLG      DEFS            1
  73EE             5054 S0_C0           DEFS            1
  73EF             5055 S0_C1           DEFS            1
  73F0             5056 S1_C0           DEFS            1
  73F1             5057 S1_C1           DEFS            1
                   5058  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                   ************************** EXTERNAL SYMBOLS ***************************

                   * EXTERNAL ROUTINES FROM OS

                                      ;EXT INIT_TABLE
                                      ;EXT PUT_VRAM
                                      ;EXT GAME_NAME
                                      ;EXT WRITE_REGISTER
                                      ;EXT READ_REGISTER
                                      ;EXT WRITE_VRAM
                                      ;EXT START_GAME

                   **************************** DEFINITIONS ****************************

<00BE>             MODE_0_PORT        EQU 0BEH
<00BF>             MODE_1_PORT        EQU 0BFH

                   ****************************** EXPORTS ******************************

                                      GLB ASCII_TBL                 ;POINTER TO UPPERCASE ASCII GENERATORS
                                      GLB NUMBER_TBL                ;POINTER TO 0-9 GENERATORS
                                      GLB DISPLAY_LOGO              ;DISPLAY COLECOVISION LOGO
                                      GLB LOAD_ASCII_               ;LOAD PATTERN_GEN_TABLE WITH FULL ASCII SET
                                      GLB FILL_VRAM_                ;FILL VRAM WITH A VALUE
                                      GLB MODE_1_                   ;SET UP MODE_1 GRAPHICS

                   ************************** DESCRIPTION ******************************
                   *
                   *  DISPLAY_LOGO  DISPLAYS THE COLECO LOGO SCREEN WITH COLECOVISION
                   *                ON A BLACK BACKGROUND.  THE GAME TITLE, MANUFACTURER,
                   *                AND COPYRIGHT YEAR ARE OBTAINED FROM THE CARTRIDGE
                   *                AND OVERLAYED ONTO THE LOGO SCREEN.  THE LOGO IS THEN
                   *                DISPLAYED FOR 10 SECONDS AFTER WHICH TIME A JUMP TO
                   *                THE GAME START ADDRESS IS EXECUTED.
                   *
                   *                IF NO CARTRIDGE IS PRESENT A DEFAULT MESSAGE IS
                   *                DISPLAYED, INSTRUCTING THE OPERATOR TO:
                   *
                   *                       "TURN GAME OFF"
                   *                       "BEFORE INSERTING CARTRIDGE"
                   *                       "OR EXPANSION MODULE."
                   *                       "[COPYRIGHT SYMBOL] 1982 COLECO"
                   *                THIS MESSAGE IS DISPLAYED FOR 60 SECONDS, THE SCREEN
                   *                IS THEN BLANKED AND FINALLY A SOFT HALT (JP $) IS
                   *                EXECUTED LOCKING UP THE PROGRAM UNTIL THE UNIT IS
                   *                RESET.
                   *
                   *                DISPLAY_LOGO EXITS WITH THE VDP IN MODE 1, THE SCREEN
                   *                BLANKED, AND THE ASCII CHARACTER SET IN VRAM.
                   *                THE MEMORY MAP IS AS FOLLOWS:
                   *
                   *                          VDP MEMORY MAP
                   *                3800H-3FFFH  SPRITE GENERATOR TABLE
                   *                2000H-37FFH  PATTERN COLOR TABLE
                   *                1B00H-1B7FH  SPRITE ATTRIBUTE TABLE
                   *                1800H-1AFFH  PATTERN NAME TABLE
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      5117 *                 0000H-17FFH  PATTERN GENERATOR TABLE
                      5118 *
                      5119 ********************************************************************
                      5120
                      5121 **************************** DISPLAY LOGO ****************************
                      5122
                      5123                   PROG
                      5124
                      5125 *  FILL VRAM WITH 0'S
 1319 210000          5126 DISPLAY_LOGO      LD HL,0
 131C 114000          5127                   LD DE,16384
 131F 3E00            5128                   LD A,0
 1321 CD18D4          5129                   CALL FILL_VRAM_
                      5130
                      5131 *  SET UP VDP WITH MODE_1
 1324 CD18E9          5132                   CALL MODE_1_
                      5133
                      5134 ****************** WRITE OUT PATTERN GEN TABLE ****************
                      5135
                      5136 *  WRITE OUT ASCII GENERATOR TABLES
 1327 CD1927          5137                   CALL LOAD_ASCII_
                      5138
                      5139 *  WRITE OUT GRAPHICS GENERATORS
 132A 2118A3          5140                   LD HL,OBJ_TABLE        ;POINT TO TABLE OF PTN GEN NUMBERS
 132D 110060          5141                   LD DE,60H              ;ITEM LOCATION IN VRAM PATTERN_GEN_TABLE
 1330 E5              5142 WRITE_LOOP        PUSH HL                ;SAVE LOCATION OF CURRENT CONSTRUCTION
 1331 D5              5143                   PUSH DE                ;SAVE VRAM ITEM #
                      5144 *  CALCULATE GENERATOR LOCATION
 1332 7E              5145                   LD A,(HL)
 1333 FEFF            5146                   CP OFFH                ;HAVE WE PROCESSED ALL GENERATORS?
 1335 281B            5147                   JR Z,DONE_LOGO         ; YES.  WE'RE ALL DONE
 1337 47              5148                   LD B,A                 ; NO. B=NUMBER FROM OBJ_TABLES
 1338 04              5149                   INC B
 1339 2114C3          5150                   LD HL,LOGO_GEN         ;POINT TO ROM GENERATOR TABLE
 133C 110008          5151                   LD DE,8                ;WE'RE GOING TO ADD 8 FOR EVERY
 133F 1015            5152 ADDR_ADJ          DJNZ ADD_8             ;GENERATOR INTO THE ROM GEN TBL
                      5153
                      5154 *  DONE ADJUSTING ROM GENERATOR ADDRESS
                      5155
 1341 D1              5156                   POP DE                 ;RESTORE ITEM # IN VRAM
 1342 D5              5157                   PUSH DE                ;SAVE IT
 1343 FD210001        5158                   LD IY,1                ;NUMBER OF GENERATORS
 1347 3E03            5159                   LD A,3                 ;PATTERN_GEN_TABLE CODE
 1349 CD1FBE          5160                   CALL PUT_VRAM          ;HL=ROM ADDRESS
 134C D1              5161                   POP DE                 ;RESTORE ITEM # IN VRAM
 134D E1              5162                   POP HL                 ;RESTORE CONSTRUCTION ADDRESS
 134E 13              5163                   INC DE                 ;SET UP FOR NEXT ITEM
 134F 23              5164                   INC HL
 1350 18DE            5165                   JR WRITE_LOOP          ;KEEP GOING UNTIL DONE
                      5166
 1352 D1              5167 DONE_LOGO         POP DE                 ;GOT TO POP FOR EVERY PUSH
 1353 E1              5168                   POP HL
 1354 1803            5169                   JR WRITE_NAMES         ;HOP AROUND ADD_8
                      5170
 1356 19              5171 ADD_8             ADD HL,DE              ;POINT TO NEXT GENERATOR
 1357 18E6            5172                   JR ADDR_ADJ
                      5173
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     5174 ****************** WRITE OUT PATTERN_NAME_TABLE ***************
                     5175
                     5176 *  WRITE OUT PATTERN NAME TABLE
  1359 21144D        5177 WRITE_NAMES      LD HL,LOGO_NAMES          ;WRITE OUT TOP HALF
  135C 110085        5178                  LD DE,133                 ;OF COLCOVISION
  135F FD210016      5179                  LD IY,22
  1363 3E02          5180                  LD A,2
  1365 CD1FBE        5181                  CALL PUT_VRAM
                     5182
  1368 211463        5183                  LD HL,LOGO_NAMES+22       ;WRITE OUT BOTTOM HALF
  136B 1100A5        5184                  LD DE,165                 ;OF COLECOVISION
  136E FD210016      5185                  LD IY,22
  1372 3E02          5186                  LD A,2
  1374 CD1FBE        5187                  CALL PUT_VRAM
                     5188
                     5189
  1377 2114C1        5190                  LD HL,TRADEMARK           ;WRITE OUT TM
  137A 11009B        5191                  LD DE,155
  137D FD210002      5192                  LD IY,2
  1381 3E02          5193                  LD A,2
  1383 CD1FBE        5194                  CALL PUT_VRAM
                     5195
                     5196 * SET UP DEFAULT COPYRIGHT MESSAGE
  1386 2114B4        5197                  LD HL,LOGO_NAMES+103      ;WRITE OUT c 1982 COLECO
  1389 1102AA        5198                  LD DE,682
  138C FD210000      5199                  LD IY,13
  1390 3E02          5200                  LD A,2
  1392 CD1FBE        5201                  CALL PUT_VRAM
                     5202
                     5203 ****************** WRITE OUT COLOR_NAME_TABLE ***************
                     5204
  1395 21143B        5205                  LD HL,LOGO_COLORS
  1398 110000        5206                  LD DE,0
  139B 3E04          5207                  LD A,4
  139D FD210012      5208                  LD IY,18
  13A1 CD1FBE        5209                  CALL PUT_VRAM
                     5210
                     5211 ************************** ENABLE DISPLAY *********************
                     5212
                     5213 * ENABLE DISPLAY
  13A4 0601          5214                  LD B,1
  13A6 0EC0          5215                  LD C,11000000B
  13A8 CD1FD9        5216                  CALL WRITE_REGISTER
                     5217
                     5218 * CARTRIDGE TEST
  13AB 218000        5219                  LD HL,8000H               ;IF A CARTRIDGE IS PRESENT
  13AE 7E            5220                  LD A,[HL]                 ;LOCATION 8000H WILL = 0AAH
  13AF FEAA          5221                  CP 0AAH                   ;AND 8001H WILL = 55H
  13B1 204C          5222                  JR NZ,NO_CARTRIDGE        ;    NOT PRESENT
  13B3 23            5223                  INC HL
  13B4 7E            5224                  LD A,[HL]
  13B5 FE55          5225                  CP 55H
  13B7 2046          5226                  JR NZ,NO_CARTRIDGE
                     5227                                            ;    NOT PRESENT
                     5228 * CARTRIDGE PRESENT
                     5229 * DISPLAY GAME TITLE
  13B9 218024        5230                  LD HL,GAME_NAME
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

 138C CD1946                          CALL PARSE          ;GET LENGTH OF STRING
 138F 118024                          LD DE,GAME_NAME     ;STARTING LOCATION OF 1ST STRING
 13C2 210201                          LD HL,513           ;LOCATION (0_767) TO START PRINTING
 13C5 CD1951                          CALL CENTER_PRT     ;PRINT IT

                   * DISPLAY COMPANY NAME
 13C8 218024                          LD HL,GAME_NAME     ;GET PAST 1ST STRING
 13CB CD1946                          CALL PARSE
 13CE 23                              INC HL
 13CF 54                              LD D,H              ;STARTING LOCATION OF 2ND STRING
 13D0 5D                              LD E,L              ;SAVED IN DE
 13D1 CD1946                          CALL PARSE          ;GET LENGTH OF STRING
 13D4 2101C1                          LD HL,449           ;LOCATION (0_767) TO START PRINTING
 13D7 CD1951                          CALL CENTER_PRT     ;PRINT IT

                   * CHANGE DATE
 13DA 218024                          LD HL,GAME_NAME     ;USE PARSE TO ADVANCE HL TO
 13DD CD1946                          CALL PARSE          ;COPYRIGHT YEAR
 13E0 23                              INC HL
 13E1 CD1946                          CALL PARSE
 13E4 23                              INC HL
 13E5 1102AC                          LD  DE,684          ;SCREEN LOCATION
 13E8 FD210004                        LD IY,4             ;# OF DIGITS
 13EC 3E02                            LD A,2
 13EE CD1FBE                          CALL PUT_VRAM

                   * DISPLAY 10 SECONDS
 13F1 CD1968                          CALL DELAY_10

                   * TURN OFF DISPLAY
 13F4 0601                            LD B,1
 13F6 0E80                            LD C,10000000B
 13F8 CD1FD9                          CALL WRITE_REGISTER

                   * EXIT LOGO
 13FB 2A800A                          LD HL,[START_GAME]         ;A CARTRIDGE WAS PRESENT
 13FE E9                              JP [HL]                    ;SO JUMP TO IT

                   *  WRITE OUT PATTERN_NAME_TABLE
 13FF 211479            NO_CARTRIDGE  LD HL,LOGO_NAMES+44        ;NO CARTRIDGE PRESENT
 1402 1101AA                          LD DE,426                  ;DISPLAY DEFAULT
 1405 FD21000D                        LD IY,13                   ;MESSAGE
 1409 3E02                            LD A,2
 140B CD1FBE                          CALL PUT_VRAM

 140E 211486                          LD HL,LOGO_NAMES+57        ;"TURN GAME OFF"
 1411 1101E4                          LD DE,484                  ;"TO INSERT CARTRIDGE"
 1414 FD21001A                        LD IY,26                   ;"OR EXPANSION MODULE"
 1418 3E02                            LD A,2
 141A CD1FBE                          CALL PUT_VRAM

 141D 2114A0                          LD HL,LOGO_NAMES+83
 1420 110227                          LD DE,551
 1423 FD210014                        LD IY,20
 1427 3E02                            LD A,2
 1429 CD1FBE                          CALL PUT_VRAM
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

 142C 218A00      5288              LD HL,8A00H                  ;DISPLAY 60 SECONDS
 142F CD1968      5289              CALL TIMER_1
                  5290 * TURN OFF DISPLAY
 1432 0601        5291              LD B,1
 1434 0E80        5292              LD C,10000000B               ;BLANK SCREEN
 1436 CD1FD9      5293              CALL WRITE_REGISTER
 1439 18FE        5294              JR $                         ;SOFT HALT
                  5295
                  5296 ************************************************************
                  5297 *                                                          *
                  5298 *                      DATA TABLES                         *
                  5299 *                                                          *
                  5300 ************************************************************
                  5301
                  5302 *********************** COLOR_NAME_TABLE *********************
                  5303
 143B 000000F0F0  5304 LOGO_COLORS    HEX 00,00,00,F0,F0,F0,F0,F0,F0,F0,F0,F0
 1440 F0F0F0F0F0
 1445 F0F0
 1447 D080908030  5305                HEX D0,80,90,80,30,40
 144C 40
                  5306
                  5307 ********************** PATTERN_NAME_TABLE ********************
                  5308
      <144D>      5309 LOGO_NAMES     EQU $
                  5310
 144D 6061686970  5311                HEX 60,61,68,69,70,71,78,79,80,81,88,89
 1452 7178798081
 1457 8889
 1459 64656C7475  5312                HEX 64,65,6C,74,75,7C,84,85,8C,8D
 145E 7C84858C8D
                  5313
 1463 62636A6B72  5314                HEX 62,63,6A,6B,72,73,7A,7B,82,83,8A
 1468 737A7B8283
 146D 8A
 146E 8B66676D76  5315                HEX 8B,66,67,6D,76,77,7D,86,87,8E,8F
 1473 777D86878E
 1478 8F
                  5316
 1479 5455524E20  5317                DEFB "TURN GAME OFF"
 147E 47414D4520
 1483 4F4646
 1486 4245464F52  5318                DEFB "BEFORE INSERTING CARTRIDGE"
 148B 4520494E53
 1490 455254494E
 1495 4720434152
 149A 5452494447
 149F 45
 14A0 4F52204558  5319                DEFB "OR EXPANSION MODULE."
 14A5 50414E5349
 14AA 4F4E204D4F
 14AF 44554C452E
 14B4 1D20313938  5320                DEFB 1DH," 1982 COLECO"
 14B9 3220434F4C
 14BE 45434F
                  5321
 14C1 1E1F        5322 TRADEMARK      HEX  1E,1F
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

                    5323
                    5324
                    5325
                    5326
                    5327 ******************** PATTERN GENERATOR TABLES ********************
                    5328
          <14C3>    5329 LOGO_GEN       EQU $
                    5330
                    5331 *GR0
  14C3 0000000000   5332                HEX             00,00,00,00,00,00,00,00
  14C8 000000
                    5333 *GR1
  14CB 3F7FFFFFF3   5334                HEX             3F,7F,FF,FF,F3,F3,F0,F0
  14D0 F3F0F0
                    5335 *GR2
  14D3 0080C0C0C0   5336                HEX             00,80,C0,C0,C0,C0,00,00
  14D8 C00000
                    5337 *GR3
  14DB 3F7FFFFFF3   5338                HEX             3F,7F,FF,FF,F3,F3,F3,F3
  14E0 F3F3F3
                    5339 *GR4
  14E3 0080C0C0C0   5340                HEX             00,80,C0,C0,C0,C0,C0,C0
  14E8 C0C0C0
                    5341 *GR5
  14EB F0F0F0F0F0   5342                HEX             F0,F0,F0,F0,F0,F0,F0,F0
  14F0 F0F0F0
                    5343 *GR6
  14F3 FFFFFFF0F0   5344                HEX             FF,FF,FF,F0,F0,FF,FF,FF
  14F8 FFFFFF
                    5345 *GR7
  14FB C0C0C00000   5346                HEX             C0,C0,C0,00,00,00,00,00
  1500 000000
                    5347 *GR8
  1503 F1F1F17B7B   5348                HEX             F1,F1,F1,7B,7B,7B,3F,3F
  1508 7B3F3F
                    5349 *GR9
  150B E0E0E0C0C0   5350                HEX             E0,E0,E0,C0,C0,C0,80,80
  1510 C08080
                    5351 *GR10
  1513 1F3F7F7978   5352                HEX             1F,3F,7F,79,78,7F,7F,3F
  1518 7F7F3F
                    5353 *GR11
  151B 80C0E0E000   5354                HEX             80,C0,E0,E0,00,80,C0,E0
  1520 80C0E0
                    5355 *GR12
  1523 F3F3FBFBFB   5356                HEX             F3,F3,FB,FB,FB,FF,FF,FF
  1528 FFFFFF
                    5357 *GR13
  152B C0C0C0C0C0   5358                HEX             C0,C0,C0,C0,C0,C0,C0,C0
  1530 C0C0C0
                    5359 *GR14
  1533 F3F3FFFF7F   5360                HEX             F3,F3,FF,FF,7F,3F,00,00
  1538 3F0000
                    5361 *GR15
  153B C0C0C0C080   5362                HEX             C0,C0,C0,C0,80,00,00,00
  1540 000000
                    ...3 *GR16
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  1543 F0F0FFFFFF  5364                   HEX        F0,F0,FF,FF,FF,FF,00,00
  1548 FF0000
                   5365 *GR17
  154B 0000C0C0C0  5366                   HEX        00,00,C0,C0,C0,C0,00,00
  1550 C00000
                   5367 *GR18
  1553 3F1F1F1F0E  5368                   HEX        3F,1F,1F,1F,0E,0E,00,00
  1558 0E0000
                   5369 *GR19
  155B 8000000000  5370                   HEX        80,00,00,00,00,00,00,00
  1560 000000
                   5371 *GR20
  1563 F0F0F0F0F0  5372                   HEX        F0,F0,F0,F0,F0,F0,00,00
  1568 F00000
                   5373 *GR21
  156B 1F01797F3F  5374                   HEX        1F,01,79,7F,3F,1F,00,00
  1570 1F0000
                   5375 *GR22
  1573 E0E0E0E0C0  5376                   HEX        E0,E0,E0,E0,C0,80,00,00
  1578 800000
                   5377 *GR23
  157B FFF7F7F7F3  5378                   HEX        FF,F7,F7,F7,F3,F3,00,00
  1580 F30000
                   5379 *GR24
  1583 C0C0C0C0C0  5380                   HEX        C0,C0,C0,C0,C0,C0,00,00
  1588 C00000
                   5381
        <158B>     5382 ASC_TABLE         EQU $
                   5383 * c=1D
  158B 7E81BDA1A1  5384                   HEX        7E,81,BD,A1,A1,BD,81,7E      ;cOPYRIGHT
  1590 BD817E
                   5385 * t=1E
  1593 1F04040400  5386                   HEX        1F,4,4,4,0,0,0,0             ;tRADE
  1598 000000
                   5387 * m=1F
  159B 446C545400  5388                   HEX        44,6C,54,54,0,0,0,0          ;mARK
  15A0 000000
                   5389 *   =20
  15A3 0000000000  5390 SPACE             DEFB       0,0,0,0,0,0,0,0
  15A8 000000
                   5391 * !=21
  15AB 2020202020  5392                   DEFB       20H,20H,20H,20H,20H,0,20H,00H
  15B0 002000
                   5393 * "=22
  15B3 5050500000  5394                   DEFB       50H,50H,50H,0,0,0,0,0
  15B8 000000
                   5395 * #=23
  15BB 5050F850F8  5396                   DEFB       50H,50H,0F8H,50H,0F8H,50H,50H,0
  15C0 505000
                   5397 * $=24
  15C3 2078A07028  5398                   DEFB       20H,78H,0A0H,70H,28H,0F0H,020H,0
  15C8 F02000
                   5399 * %=25
  15CB C0C8102040  5400                   DEFB       0C0H,0C8H,10H,20H,40H,98H,18H,0
  15D0 981800
                   5401 * &=26
  15D3 40A0A040A8  5402                   DEFB       40H,0A0H,0A0H,40H,0A8H,90H,68H,0
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

 1508 906800
                  5403 * '=27
 15DB 2020200000  5404             DEFB        20H,20H,20H,0,0,0,0,0
 15E0 000000
                  5405 * (=28
 15E3 2040808080  5406             DEFB        20H,40H,80H,80H,80H,40H,20H,0
 15E8 402000
                  5407 * )=29
 15EB 2010080808  5408             DEFB        20H,10H,08H,08H,08H,10H,20H,0
 15F0 102000
                  5409 * *=2A
 15F3 20A8702070  5410             DEFB        20H,0A8H,70H,20H,70H,0A8H,20H,0
 15F8 A82000
                  5411 * +=2B
 15FB 002020F820  5412             DEFB        0,20H,20H,0F8H,20H,20H,0,0
 1600 200000
                  5413 * ,=2C
 1603 0000000020  5414             DEFB        0,0,0,0,20H,20H,40H,0
 1608 204000
                  5415 * -=2D
 160B 000000F800  5416             DEFB        0,0,0,0F8H,0,0,0,0
 1610 000000
                  5417 * .=2E
 1613 0000000000  5418             DEFB        0,0,0,0,0,0,20H,0
 1618 002000
                  5419 * /=2F
 161B 0008102040  5420             DEFB        0,8,10H,20H,40H,80H,0,0
 1620 800000
                  5421
         <1623>   5422 NUMBER_TBL   EQU $
                  5423 * 0=30
 1623 708898A8C8  5424             DEFB        70H,88H,98H,0A8H,0C8H,088H,70H,0
 1628 887000
                  5425 * 1=31
 162B 2060202020  5426             DEFB        20H,60H,20H,20H,20H,20H,70H,0
 1630 207000
                  5427 * 2=32
 1633 7088083040  5428             DEFB        70H,88H,08,30H,40H,80H,0F8H,0
 1638 80F800
                  5429 * 3=33
 163B F808103008  5430             DEFB        0F8H,08,10H,30H,08,88H,70H,0
 1640 887000
                  5431 * 4=34
 1643 10305090F8  5432             DEFB        10H,30H,50H,90H,0F8H,10H,10H,0
 1648 101000
                  5433 * 5=35
 164B F880F00808  5434             DEFB        0F8H,80H,0F0H,08H,08H,88H,70H,0
 1650 887000
                  5435 * 6=36
 1653 384080F088  5436             DEFB        38H,40H,80H,0F0H,88H,88H,70H,0
 1658 887000
                  5437 * 7=37
 165B F808102040  5438             DEFB        0F8H,08H,10H,20H,40H,40H,40H,0
 1660 404000
                  5439 * 8=38
 1663 7088887088  5440             DEFB        70H,88H,88H,70H,88H,88H,70H,0
 1668 887000
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                      5441 * 9=39
  1668 7088887808     5442            DEFB       70H,88H,88H,78H,08H,10H,0E0H,0
  1670 10E000
                      5443 * :=3A
  1673 0000200020     5444            DEFB       0,0,20H,0,20H,0,0,0
  1678 000000
                      5445 * ;=3B
  167B 0000200020     5446            DEFB       0,0,20H,0,20H,20H,40H,0
  1680 204000
                      5447 * <=3C
  1683 1020408040     5448            DEFB       10H,20H,40H,80H,40H,20H,10H,0
  1688 201000
                      5449 * ==3D
  168B 0000F800F8     5450            DEFB       0,0,0F8H,0,0F8H,0,0,0
  1690 000000
                      5451 * >=3E
  1693 4020100810     5452            DEFB       40H,20H,10H,08H,10H,20H,40H,0
  1698 204000
                      5453 * ?=3F
  169B 7088102020     5454            DEFB       70H,88H,10H,20H,20H,0,20H,0
  16A0 002000
                      5455 * @=40
  16A3 7088A8B880     5456            DEFB       70H,88H,0A8H,0B8H,080H,080H,78H,0
  16A8 807800
                      5457
          <16AB>      5458 ASCII_TBL   EQU $
                      5459 * A=41
  16AB 20508888F8     5460            DEFB       20H,50H,88H,88H,0F8H,88H,88H,0
  16B0 888800
                      5461 * B=42
  16B3 F08888F088     5462            DEFB       0F0H,88H,88H,0F0H,88H,88H,0F0H,0
  16B8 88F000
                      5463 * C=43
  16BB 7088808080     5464            DEFB       70H,88H,80H,80H,80H,88H,70H,0
  16C0 887000
                      5465 * D=44
  16C3 F088888888     5466            DEFB       0F0H,88H,88H,88H,88H,88H,0F0H,0
  16C8 88F000
                      5467 * E=45
  16CB F88080F080     5468            DEFB       0F8H,80H,80H,0F0H,080H,80H,0F8H,0
  16D0 80F800
                      5469 * F=46
  16D3 F88080F080     5470            DEFB       0F8H,80H,80H,0F0H,80H,80H,80H,0
  16D8 808000
                      5471 * G=47
  16DB 7880808098     5472            DEFB       78H,80H,80H,80H,98H,88H,78H,0
  16E0 887800
                      5473 * H=48
  16E3 888888F888     5474            DEFB       88H,88H,88H,0F8H,88H,88H,88H,0
  16E8 888800
                      5475 * I=49
  16EB 7020202020     5476            DEFB       70H,20H,20H,20H,20H,20H,70H,0
  16F0 207000
                      5477 * J=4A
  16F3 0808080808     5478            DEFB       8,8,8,8,8,88H,70H,0
  16F8 887000
                      5479 * K=4B
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

  16FB 8890A0C0A0  5480                 DEFB        88H,90H,0A0H,0C0H,0A0H,90H,88H,0
  1700 908800
                   5481 * L=4C
  1703 8080808080  5482                 DEFB        80H,80H,80H,80H,80H,80H,0F8H,0
  1708 80F800
                   5483 * M=4D
  1708 88D8A8A888  5484                 DEFB        88H,0D8H,0A8H,0A8H,88H,88H,88H,0
  1710 888800
                   5485 * N=4E
  1713 8888C8A898  5486                 DEFB        88H,88H,0C8H,0A8H,98H,88H,88H,0
  1718 888800
                   5487 * O=4F
  1718 7088888888  5488                 DEFB        70H,88H,88H,88H,88H,88H,70H,0
  1720 887000
                   5489 * P=50
  1723 F08888F080  5490                 DEFB        0F0H,88H,88H,0F0H,80H,80H,80H,00
  1728 808000
                   5491 * Q=51
  1728 70888888A8  5492                 DEFB        70H,88H,88H,88H,0A8H,90H,68H,0
  1730 906800
                   5493 * R=52
  1733 F08888F0A0  5494                 DEFB        0F0H,88H,88H,0F0H,0A0H,90H,88H,0
  1738 908800
                   5495 * S=53
  1738 7088807008  5496                 DEFB        70H,88H,80H,70H,08H,88H,70H,0
  1740 887000
                   5497 * T=54
  1743 F820202020  5498                 DEFB        0F8H,20H,20H,20H,20H,20H,20H,0
  1748 202000
                   5499 * U=55
  1748 8888888888  5500                 DEFB        88H,88H,88H,88H,88H,88H,70H,0
  1750 887000
                   5501 * V=56
  1753 8888888888  5502                 DEFB        88H,88H,88H,88H,88H,50H,20H,0
  1758 502000
                   5503 * W=57
  1758 888888A8A8  5504                 DEFB        88H,88H,88H,0A8H,0A8H,0D8H,88H,0
  1760 D88800
                   5505 * X=58
  1763 8888502050  5506                 DEFB        88H,88H,50H,20H,50H,88H,88H,0
  1768 888800
                   5507 * Y=59
  1768 8888502020  5508                 DEFB        88H,88H,50H,20H,20H,20H,20H,0
  1770 202000
                   5509 * Z=5A
  1773 F808102040  5510                 DEFB        0F8H,08H,10H,20H,40H,80H,0F8H,0
  1778 80F800
                   5511 * [=5B
  1778 F8C0C0C0C0  5512                 DEFB        0F8H,0C0H,0C0H,0C0H,0C0H,0C0H,0F8H,0
  1780 C0F800
                   5513 * \=5C
  1783 0080402010  5514                 DEFB        0,80H,40H,20H,10H,08H,0,0
  1788 080000
                   5515 * ]=5D
  1788 F818181818  5516                 DEFB        0F8H,18H,18H,18H,18H,18H,0F8H,0
  1790 18F800
                   5517 * ^=5E
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE
    1793 0000205088  5518               DEFB       0,0,20H,50H,88H,0,0,0
    1798 000000
                     5519 *  _=5F
    1798 0000000000  5520               DEFB       0,0,0,0,0,0,0,0F8H
    17A0 0000F8
                     5521 *  '=60
    17A3 4020100000  5522               DEFB       40H,20H,10H,0,0,0,0,0
    17A8 000000
                     5523 *  a=61
    17AB 00007088F8  5524               DEFB       0,0,70H,88H,0F8H,88H,88H,0
    17B0 888800
                     5525 *  b=62
    17B3 0000F04870  5526               DEFB       0,0,0F0H,48H,70H,48H,0F0H,0
    17B8 48F000
                     5527 *  c=63
    17BB 0000788080  5528               DEFB       0,0,78H,80H,80H,80H,78H,0
    17C0 807800
                     5529 *  d=64
    17C3 0000F04848  5530               DEFB       0,0,0F0H,048H,048H,048H,0F0H,0
    17C8 48F000
                     5531 *  e=65
    17CB 0000F080E0  5532               DEFB       0,0,0F0H,080H,0E0H,80H,0F0H,0
    17D0 80F000
                     5533 *  f=66
    17D3 0000F080E0  5534               DEFB       0,0,0F0H,080H,0E0H,080H,80H,0
    17D8 808000
                     5535 *  g=67
    17DB 0000788088  5536               DEFB       0,0,78H,80H,088H,88H,70H,0
    17E0 887000
                     5537 *  h=68
    17E3 00008888F8  5538               DEFB       0,0,88H,88H,0F8H,88H,88H,0
    17E8 888800
                     5539 *  i=69
    17EB 0000F82020  5540               DEFB       0,0,0F8H,20H,20H,20H,0F8H,0
    17F0 20F800
                     5541 *  j=6A
    17F3 0000702020  5542               DEFB       0,0,70H,20H,20H,0A0H,0E0H,0
    17F8 A0E000
                     5543 *  k=6B
    17FB 000090A0C0  5544               DEFB       0,0,90H,0A0H,0C0H,0A0H,90H,0
    1800 A09000
                     5545 *  l=6C
    1803 0000808080  5546               DEFB       0,0,80H,80H,80H,80H,0F8H,0
    1808 80F800
                     5547 *  m=6D
    180B 000088D8A8  5548               DEFB       0,0,88H,0D8H,0A8H,88H,88H,0
    1810 888800
                     5549 *  n=6E
    1813 000088C8A8  5550               DEFB       0,0,88H,0C8H,0A8H,98H,88H,0
    1818 988800
                     5551 *  o=6F
    181B 0000F88888  5552               DEFB       0,0,0F8H,088H,88H,88H,0F8H,0
    1820 88F800
                     5553 *  p=70
    1823 0000F088F0  5554               DEFB       0,0,0F0H,88H,0F0H,80H,80H,0
    1828 808000
                     5555 *  q=71
```

```
LOCATION OBJECT CODE LINE      SOURCE LINE

    182B 0000F888A8  5556                DEFB          0,0,0F8H,88H,0A8H,90H,0E0H,0
    1830 90E000
                     5557 * r=72
    1833 0000F888F8  5558                DEFB          0,0,0F8H,88H,0F8H,0A0H,90H,0
    1838 A09000
                     5559 * s=73
    183B 0000788070  5560                DEFB          0,0,78H,80H,70H,08H,0F0H,0
    1840 08F000
                     5561 * t=74
    1843 0000F82020  5562                DEFB          0,0,0F8H,20H,20H,20H,20H,0
    1848 202000
                     5563 * u=75
    184B 0000888888  5564                DEFB          0,0,88H,88H,88H,88H,70H,00
    1850 887000
                     5565 * v=76
    1853 0000888890  5566                DEFB          0,0,88H,88H,90H,0A0H,40H,0
    1858 A04000
                     5567 * w=77
    185B 00008888A8  5568                DEFB          0,0,88H,88H,0A8H,0D8H,88H,00
    1860 D88800
                     5569 * x=78
    1863 0000886020  5570                DEFB          0,0,88H,60H,20H,60H,88H,0
    1868 608800
                     5571 * y=79
    186B 0000885020  5572                DEFB          0,0,88H,50H,20H,20H,20H,0
    1870 202000
                     5573 * z=7A
    1873 0000F81020  5574                DEFB          0,0,0F8H,10H,20H,40H,0F8H,0
    1878 40F800
                     5575 * {=7B
    187B 384020C020  5576                DEFB          38H,40H,20H,0C0H,20H,40H,38H,0
    1880 403800
                     5577 * >=7C
    1883 4020100810  5578                DEFB          40H,20H,10H,08H,10H,20H,40H,0
    1888 204000
                     5579 * }=7D
    188B E010201820  5580                DEFB          0E0H,10H,20H,18H,20H,10H,0E0H,0
    1890 10E000
                     5581 * ~=7E
    1893 40A8100000  5582                DEFB          40H,0A8H,10H,0,0,0,0,0
    1898 000000
                     5583 *  ■=7F
    189B A850A850A8  5584                DEFB          0A8H,50H,0A8H,50H,0A8H,50H,0A8H,0
    18A0 50A800
                     5585
          <18A3>     5586 OBJ_TABLE      EQU $
    18A3 01020E0F08  5587                DEFB 1,2,14,15,8,9,18,19
    18A8 091213
    18AB 03040E0F05  5588                DEFB 3,4,14,15,5,20,0,0
    18B0 140000
    18B3 050010110A  5589                DEFB 5,0,16,17,10,11,21,22
    18B8 0B1516
    18BB 0607101105  5590                DEFB 6,7,16,17,5,20,0,0
    18C0 140000
    18C3 01020E0F03  5591                DEFB 1,2,14,15,3,4,14,15
    18C8 040E0F
    18CB 03040E0F0C  5592                DEFB 3,4,14,15,12,13,23,24
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  18D0 001718
  18D3 FF                             HEX  FF                         ;END OF TABLE INDICATOR

                       ****************************************************************
                       *                                                              *
                       *                           SUBROUTINES                        *
                       *                                                              *
                       ****************************************************************

                       ************************ OS SUBROUTINES ************************

                       * FILL_VRAM_  WRITES TO VRAM ADDRESS POINTED TO BY HL THE VALUE IN A
                       *             DE TIMES.

                       *           VRAM STARTING ADDRESS IN HL
                       *           NO OF BYTES IN DE
                       *           VALUE TO BE WRITTEN IN A

  18D4 4F              FILL_VRAM_  LD   C,A
  18D5 7D                          LD   A,L
  18D6 D3BF                        OUT  [MODE_1_PORT],A
  18D8 7C                          LD   A,H
  18D9 F640                        OR   40H
  18DB D3BF                        OUT  [MODE_1_PORT],A
  18DD 79              FILL        LD   A,C
  18DE D3BE                        OUT  [MODE_0_PORT],A
  18E0 1B                          DEC  DE
  18E1 7A                          LD   A,D
  18E2 B3                          OR   E
  18E3 20F8                        JR   NZ,FILL
  18E5 CD1FDC                      CALL READ_REGISTER
  18E8 C9                          RET


                       * MODE_1_  SETS UP GRAPHICS MODE 1 WITH VRAM ADDRESSES AS IN THE TABLE
                       *          BELOW AND EXITS WITH THE VIDEO BLANKED AND A BLACK BACKGROUND.

                       *                        VDP MEMORY MAP
                       *               3800H-3FFFH  SPRITE GENERATOR TABLE
                       *               2000H-37FFH  PATTERN COLOR TABLE
                       *               1B00H-1B7FH  SPRITE ATTRIBUTE TABLE
                       *               1800H-1AFFH  PATTERN NAME TABLE
                       *               0000H-17FFH  PATTERN GENERATOR TABLE

  18E9 0600            MODE_1_         LD B,0
  18EB 0E00                            LD C,0
  18ED CD1FD9                          CALL WRITE_REGISTER

  18F0 0601                            LD B,1
  18F2 0E80                            LD C,10000000B
  18F4 CD1FD9                          CALL WRITE_REGISTER

                       * SET UP TABLE ADDRESSES IN VRAM

                       * PATTERN NAME TABLE
  18F7 3E02                            LD A,2
  18F9 211800                          LD HL,1800H
```

```
ATION OBJECT CODE LINE     SOURCE LINE

 18FC CD1FB8                        CALL INIT_TABLE

                 * PATTERN COLOR TABLE
 18FF 3E04                          LD A,4
 1901 212000                        LD HL,2000H
 1904 CD1FB8                        CALL INIT_TABLE

                 * PATTERN GENERATOR TABLE
 1907 3E03                          LD A,3
 1909 210000                        LD HL,0
 190C CD1FB8                        CALL INIT_TABLE

                 * SPRITE ATTRIBUTE TABLE
 190F 3E00                          LD A,0
 1911 211B00                        LD HL,1B00H
 1914 CD1FB8                        CALL INIT_TABLE

                 * SPRITE GENERATOR TABLE
 1917 3E01                          LD A,1
 1919 213800                        LD HL,3800H
 191C CD1FB8                        CALL INIT_TABLE

                 *  SET UP BLACK BACKGROUND
 191F 0607                          LD B,7
 1921 0E00                          LD C,0
 1923 CD1FD9                        CALL WRITE_REGISTER
 1926 C9                            RET

                 *  LOAD_ASCII_ WRITES OUT ASCII CHARACTER GENERATORS TO THE PATTERN
                 *              GENERATOR TABLE.  INIT_TABLE MUST BE USED TO SET UP
                 *              THE TABLE ADDRESS.

 1927 2115B8     LOAD_ASCII_        LD HL,ASC_TABLE          ;LOCATION OF GENERATORS
 192A 11001D                        LD DE,10H                ;OFFSET TO PLACE ASC_TABLE
 :/2D FD210060                      LD IY,96                 ;IN THE CORRECT LOCATION
 1931 3E03                          LD A,3
 1933 CD1FBE                        CALL PUT_VRAM

                 * WRITE OUT A BLANK PATTERN FOR ASCII_0
 1936 2115A3                        LD HL,SPACE
 1939 110000                        LD DE,0
 193C FD210001                      LD IY,1
 1940 3E03                          LD A,3
 1942 CD1FBE                        CALL PUT_VRAM
 1945 C9                            RET

                 ********************** LOCAL SUBROUTINES **********************

 1946 010000     PARSE              LD BC,0          ;FROM HL INCREMENT BC UNTIL
 1949 7E         P_LOOP             LD A,[HL]        ;[HL] = "/"
 194A FE2F                          CP "/"
 194C C8                            RET Z
 194D 23                            INC HL
 194E 03                            INC BC
 194F 18F8                          JR P_LOOP

 1951 C5          ENTER PRT         PUSH BC          ;BC=LENS
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

    1952 FDE1        5706                 POP IY               ;IY= #ITEMS TO BE TRANSFERED IN PUT_VRAM
    1954 3E20        5707                 LD A,32              ;DE= LOCATION OF START OF STRING
    1956 99          5708                 SBC A,C              ; A=32-C
    1957 1F          5709                 RRA                  ; DIV 2
    1958 0600        5710                 LD B,0
    195A 4F          5711                 LD C,A
    195B 09          5712                 ADD HL,BC
    195C 44          5713                 LD B,H
    195D 4D          5714                 LD C,L
    195E 62          5715                 LD H,D
    195F 6B          5716                 LD L,E
    1960 50          5717                 LD D,B
    1961 59          5718                 LD E,C
    1962 3E02        5719                 LD A,2
    1964 CD1FBE      5720                 CALL PUT_VRAM
    1967 C9          5721                 RET
                     5722
    1968 211700      5723 DELAY_10        LD HL,1700H
    196B 1100FF      5724 TIMER_1         LD DE,255
    196E 1B          5725 TIMER_2         DEC DE
    196F 7A          5726                 LD A,D
    1970 B3          5727                 OR E
    1971 20FB        5728                 JR NZ,TIMER_2
    1973 2B          5729                 DEC HL
    1974 7C          5730                 LD A,H
    1975 B5          5731                 OR L
    1976 20F3        5732                 JR NZ,TIMER_1
    1978 C9          5733                 RET
                     5734  PROG
```

```
OCATION OBJECT CODE LINE     SOURCE LINE


                  ************************* EXTERNAL SYMBOLS ***************************

                  **********************************************************************

                  * EXTERNAL ROUTINES FROM OS

                                  ;EXT INIT_TABLE
                                  ;EXT PUT_VRAM
                                  ;EXT WRITE_REGISTER
                                  ;EXT WRITE_VRAM
                                  ;EXT VRAM_ADDR_TABLE
                                  ;EXT SPRITENAMETBL
                                  ;EXT SPRITEGENTBL
                                  ;EXT PATTRNNAMETBL
                                  ;EXT PATTRNGENTBL
                                  ;EXT COLORTABLE
                                  ;EXT LOAD_ASCII
                                  ;EXT FILL_VRAM
                                  ;EXT MODE_1

                  ************************** DEFINITIONS ***************************

                  ***************************** EXPORTS *******************************

                                  GLB GAME_OPT_

                  ******************** DISPLAY GAME OPTION SCREEN ********************

                  *  GAME_OPT_  DISPLAYS THE GAME OPTION SCREEN WITH WHITE LETTERS ON A
                  *             BLUE BACKGROUND.  VDP IS LEFT IN MODE 1 WITH THE VRAM
                  *             MEMORY MAP AS FOLLOWS.
                  *
                  ;                               VDP MEMORY MAP
                  ;               3800H-3FFFH  SPRITE GENERATOR TABLE
                  ;               2000H-37FFH  PATTERN COLOR TABLE
                  ;               1B00H-1B7FH  SPRITE ATTRIBUTE TABLE
                  ;               1800H-1AFFH  PATTERN NAME TABLE
                  ;               0000H-17FFH  PATTERN GENERATOR TABLE

                                  PROG
1979 210000       GAME_OPT_       LD HL,0                         ;ZERO VRAM
197C 114000                       LD DE,16384
197F 3E00                         LD A,0
1981 CD1F82                       CALL FILL_VRAM

                  *  SET UP VDP WITH MODE_1
1984 CD1F85                       CALL MODE_1

                  *  SET UP BACKGROUND COLOR
1987 060F                         LD B,0FH
1989 0E04                         LD C,4
198B CD1FD9                       CALL WRITE_REGISTER

                  ****************** WRITE OUT PATTERN GEN TABLE ******************

198E CD1F7F                       CALL LOAD_ASCII
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     5793
                     5794 ****************** WRITE OUT PATTERN_NAME_TABLE ****************
                     5795
                     5796 *  WRITE OUT PATTERN_NAME_TABLE
  1991 211A7C        5797              LD HL,LINE_1
  1994 110025        5798              LD DE,37
  1997 FD210016      5799              LD IY,22
  199B 3E02          5800              LD A,2
  199D CD1FBE        5801              CALL PUT_VRAM
                     5802
  19A0 211A92        5803              LD HL,LINE_2
  19A3 110065        5804              LD DE,101
  19A6 FD210017      5805              LD IY,23
  19AA 3E02          5806              LD A,2
  19AC CD1FBE        5807              CALL PUT_VRAM
                     5808
  19AF 1100C5        5809              LD DE,197
  19B2 CD1ACA        5810              CALL WRITE_L3
                     5811
  19B5 110105        5812              LD DE,261
  19B8 CD1ACA        5813              CALL WRITE_L3
                     5814
  19BB 110145        5815              LD DE,325
  19BE CD1ACA        5816              CALL WRITE_L3
                     5817
  19C1 110185        5818              LD DE,389
  19C4 CD1ACA        5819              CALL WRITE_L3
                     5820
  19C7 1101E5        5821              LD DE,485
  19CA CD1ACA        5822              CALL WRITE_L3
                     5823
  19CD 110225        5824              LD DE,549
  19D0 CD1ACA        5825              CALL WRITE_L3
                     5826
  19D3 110265        5827              LD DE,613
  19D6 CD1ACA        5828              CALL WRITE_L3
                     5829
  19D9 1102A5        5830              LD DE,677
  19DC CD1ACA        5831              CALL WRITE_L3
                     5832
  19DF 110105        5833              LD DE,261
  19E2 CD1AD7        5834              CALL WRITE_L4
                     5835
  19E5 110145        5836              LD DE,325
  19E8 CD1ADC        5837              CALL WRITE_L5
                     5838
  19EB 110185        5839              LD DE,389
  19EE CD1AE1        5840              CALL WRITE_L6
                     5841
  19F1 211AC2        5842              LD HL,LINE_7
  19F4 1101E5        5843              LD DE,485
  19F7 CD1AE4        5844              CALL WRITE_CHAR
                     5845
  19FA 211AC3        5846              LD HL,LINE_8
  19FD 110225        5847              LD DE,549
  1A00 CD1AE4        5848              CALL WRITE_CHAR
                     5849
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

   1A03 211AC4                  LD HL,LINE_9
   1A06 110265                  LD DE,613
   1A09 CD1AE4                  CALL WRITE_CHAR
   1A0C 211AC5                  LD HL,LINE_10
   1A0F 1102A5                  LD DE,677
   1A12 CD1AE4                  CALL WRITE_CHAR
   1A15 11010F                  LD DE,271
   1A18 CD1AD7                  CALL WRITE_L4
   1A1B 11014F                  LD DE,335
   1A1E CD1ADC                  CALL WRITE_L5
   1A21 11018F                  LD DE,399
   1A24 CD1AE1                  CALL WRITE_L6
   1A27 1101F1                  LD DE,497
   1A2A CD1AEE                  CALL WRITE_L11
   1A2D 110231                  LD DE,561
   1A30 CD1AEE                  CALL WRITE_L11
   1A33 110271                  LD DE,625
   1A36 CD1AEE                  CALL WRITE_L11
   1A39 110281                  LD DE,689
   1A3C CD1AEE                  CALL WRITE_L11
   1A3F 11022F                  LD DE,559
   1A42 CD1AD7                  CALL WRITE_L4
   1A45 11026F                  LD DE,623
   1A48 CD1ADC                  CALL WRITE_L5
   1A4B 1102AF                  LD DE,687
   1A4E CD1AE1                  CALL WRITE_L6
   1A51 1101FB                  LD DE,507
   1A54 CD1AFB                  CALL WRITE_L12
   1A57 11023B                  LD DE,571
   1A5A CD1AFB                  CALL WRITE_L12
   1A5D 11027B                  LD DE,635
   1A60 CD1AFB                  CALL WRITE_L12
   1A63 1102BB                  LD DE,699
   1A66 CD1AFB                  CALL WRITE_L12
        ****************** WRITE OUT COLOR_NAME_TABLE ****************
   1A69 2A73FA                  LD HL,[COLORTABLE]
   1A6C 110020                  LD DE,32
   1A6F 3EF4                    LD A,0F4H
   1A71 CD1F82                  CALL FILL_VRAM
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     5907 *************************** ENABLE DISPLAY **********************
                     5908
                     5909 * ENABLE DISPLAY
  1A74 0601          5910                 LD B,1
  1A76 0EC0          5911                 LD C,11000000B
  1A78 CD1FD9        5912                 CALL WRITE_REGISTER
  1A7B C9            5913                 RET
                     5914
                     5915 ******************************************************************
                     5916 *                                                                *
                     5917 *                         DATA TABLES                            *
                     5918 *                                                                *
                     5919 ******************************************************************
                     5920
                     5921 ********************** PATTERN_NAME_TABLE **********************
  1A7C 544F205345    5922 LINE_1 DEFB "TO SELECT GAME OPTION,"
  1A81 4C45435420
  1A86 4741404520
  1A8B 4F5054494F
  1A90 4E2C
  1A92 5052455353    5923 LINE_2 DEFB "PRESS BUTTON ON KEYPAD."
  1A97 2042555454
  1A9C 4F4E204F4E
  1AA1 2048455950
  1AA6 41442E
  1AA9 3120302053    5924 LINE_3 DEFB "1 = SKILL 1/ONE PLAYER"
  1AAE 48494C4C20
  1AB3 312F4F4E45
  1AB8 20504C4159
  1ABD 4552
  1ABF 32            5925 LINE_4 DEFB "2"
  1AC0 33            5926 LINE_5 DEFB "3"
  1AC1 34            5927 LINE_6 DEFB "4"
  1AC2 35            5928 LINE_7 DEFB "5"
  1AC3 36            5929 LINE_8 DEFB "6"
  1AC4 37            5930 LINE_9 DEFB "7"
  1AC5 38            5931 LINE_10 DEFB "8"
  1AC6 54574F        5932 LINE_11 DEFB "TWO"
  1AC9 53            5933 LINE_12 DEFB "S"
                     5934
                     5935 ******************************************************************
                     5936 *                                                                *
                     5937 *                      LOCAL SUBROUTINES                         *
                     5938 *                                                                *
                     5939 ******************************************************************
                     5940
  1ACA 211AA9        5941 WRITE_L3        LD HL,LINE_3
  1ACD FD210016      5942                 LD IY,22
  1AD1 3E02          5943                 LD A,2
  1AD3 CD1FBE        5944                 CALL PUT_VRAM
  1AD6 C9            5945                 RET
                     5946
  1AD7 211ABF        5947 WRITE_L4        LD HL,LINE_4
  1ADA 1808          5948                 JR WRITE_CHAR
                     5949
  1ADC 211AC0        5950 WRITE_L5        LD HL,LINE_5
  1ADF 1803          5951                 JR WRITE_CHAR
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     5952
  1AE1 211AC1        5953 WRITE_L6        LD HL,LINE_6
  1AE4 FD210001      5954 WRITE_CHAR      LD IY,1
  1AE8 3E02          5955                 LD A,2
  1AEA CD1FBE        5956                 CALL PUT_VRAM
  1AED C9            5957                 RET
                     5958
  1AEE 211AC6        5959 WRITE_L11       LD HL,LINE_11
  1AF1 FD210003      5960                 LD IY,3
  1AF5 3E02          5961                 LD A,2
  1AF7 CD1FBE        5962                 CALL PUT_VRAM
  1AFA C9            5963                 RET
                     5964
  1AFB 211AC9        5965 WRITE_L12       LD HL,LINE_12
  1AFE FD210001      5966                 LD IY,1
  1B02 3E02          5967                 LD A,2
  1B04 CD1FBE        5968                 CALL PUT_VRAM
  1B07 C9            5969                 RET
                     5970
                     5971  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                 ;TRUE            EQU          1
                            ;EXT  VRAM_WRITE,REG_WRITE,VRAM_READ
                            ;EXT  VDP_MODE_WORD
                            ;EXT  MUX_SPRITES
                            ;EXT  PARAM_
                            ;EXT  LOCAL_SPR_TBL,SPRITE_ORDER
                             GLB  INIT_TABLE_,GET_VRAM_,PUT_VRAM_,INIT_SPR_ORDER_,WR_SPR_NM_TBL
                             GLB  INIT_TABLEQ,GET_VRAMQ,PUT_VRAMQ,INIT_SPR_ORDERQ,WR_SPR_NM_TBLQ

                                  PROG
                 * PROCEDURE INIT_TABLEQ (TABLE_CODE:BYTE;TABLE_ADDRESS:INTEGER)

                 * THIS IS THE PASCAL ENTRY POINT TO INIT_TABLE_

 1B08 00020001   INIT_TABLE_P     DEFW            2,1,2
 1B0C 0002
                 * THIS IS THE PARAMETER DESCRIPTOR FOR INIT_TABLEQ

 1B0E            INIT_TABLEQ
 1B0E 011B08                      LD              BC,INIT_TABLE_P
 1B11 1173BA                      LD              DE,PARAM_AREA
 1B14 CD0098                      CALL            PARAM_
 1B17 3A73BA                      LD              A,[PARAM_AREA]
 1B1A 2A73BB                      LD              HL,[PARAM_AREA+1]

 1B1D            INIT_TABLE_                       ;INIT_TABLE_ INITIALIZES THE TABLE ADDRESSES
                                                  ;FOR VRAM TABLES. IT ALSO WRITES THE APPROPRIATE
                                                  ;BASE ADDRESS INTO THE RESPECTIVE VDP REGISTER.
                                                  ;
                                                  ;IN: VRAM ADDRESS IN HL
                                                  ;    TABLE CODE IN 'A' : 0=SPRITE NAME TABLE
                                                  ;    1=SPRITE GENERATOR TABLE, 2=PATTERN NAME
                                                  ;    TABLE,3= PATTERN GENERATOR TABLE, 4=
                                                  ;    COLOR TABLE

 1B1D 4F                     LD   C,A
 1B1E 0600                   LD   B,0             ;USE TABLE CODE AS INDEX
 1B20 DD2173F2               LD   IX,VRAM_ADDR_TABLE
 1B24 DD09                   ADD  IX,BC
 1B26 DD09                   ADD  IX,BC
 1B28 DD7500                 LD   [IX+0],L        ;SAVE VRAM ADDRESS IN TABLE
 1B2B DD7401                 LD   [IX+1],H
                                                  ;CHECK VDP GRAPHICS MODE
 1B2E 3A73C3                 LD   A,[VDP_MODE_WORD]
 1B31 CB4F                   BIT  1,A
 1B33 2827                   JR   Z,INIT_TABLE80
                             ;**  GRAPHICS MODE 2
 1B35 79                     LD   A,C             ;GET TABLE CODE
 1B36 FE03                   CP   3               ;CHECK IF GENERATOR TABLE
 1B38 2806                   JR   Z,CASE_OF_GEN
 1B3A FE04                   CP   4               ;CHECK IF COLOR TABLE
 1B3C 2810                   JR   Z,CASE_OF_COLOR
 1B3E 181C                   JR   INIT_TABLE80
 1B40            CASE_OF_GEN
 1B40 0604                   LD   B,4             ;VDP REGISTER TO WRITE
 1B42 7D                     LD   A,L             ;CHECK FOR WHICH 8K BOUNDARY IN 16K VRAM
 1B43 B4                     OR   H
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  1844 2004         6029               JR   NZ,CASE_OF_GEN10
  1846 0E03         6030               LD   C,3              ;VALUE TO WRITE FOR VRAM ADDRESS OF 00H
  1848 1828         6031               JR   INIT_TABLE90
  184A              6032 CASE_OF_GEN10
  184A 0E07         6033               LD   C,7              ;VALUE TO WRITE FOR VRAM ADDRESS OF 2000H
  184C 1824         6034               JR   INIT_TABLE90
  184E              6035 CASE_OF_COLOR
  184E 0603         6036               LD   B,3              ;REGISTER TO WRITE
  1850 7D           6037               LD   A,L
  1851 B4           6038               OR   H
  1852 2004         6039               JR   NZ,CASE_OF_CLR10
  1854 0E7F         6040               LD   C,7FH            ;VALUE TO WRITE FOR VRAM ADDRESS OF 00H
  1856 181A         6041               JR    INIT_TABLE90
  1858              6042 CASE_OF_CLR10
  1858 0EFF         6043               LD   C,0FFH           ;VALUE TO WRITE FOR VRAM ADDRESS OF 2000H
  185A 1816         6044               JR   INIT_TABLE90
                    6045
  185C              6046 INIT_TABLE80
                    6047               ;**  COMPUTE BASE ADDRESS (BASE_ADDRESS=TABLE_ADDRESS/FACTOR
                    6048               ;**  GET BIT SHIFT COUNT
  185C FD211B76     6049               LD   IY,BASE_FACTORS
  1860 FD09         6050               ADD  IY,BC
  1862 FD09         6051               ADD  IY,BC
  1864 FD7E00       6052               LD   A,[IY+0]         ;SHIFT COUNT NOW IN 'A'
  1867 FD4601       6053               LD   B,[IY+1]         ;REGISTER NUMBER TO WRITE IN 'B'
  186A              6054 DIVIDE        ;**  COMPUTE BASE ADDRESS
  186A CB3C         6055               SRL  H                ;SHIFT HI BYTE
  186C CB1D         6056               RR   L                ;SHIFT LO BYTE
  186E 3D           6057               DEC  A                ;DECREMENT SHIFT COUNT
  186F 20F9         6058               JR   NZ,DIVIDE
                    6059               ;**  WRITE TO VDP REGISTER
  1871 4D           6060               LD   C,L              ;VALUE TO WRITE IN 'C'
  1872              6061 INIT_TABLE90
  1872 CD1CCA       6062               CALL REG_WRITE
  1875 C9           6063               RET
                    6064
  1876              6065 BASE_FACTORS    ;BASE_FACTOR,REGISTER_NUMBER
                    6066
  1876 0705080604   6067               DEFB 7,5,11,6,10,2,11,4,6,3
  187B 0208040603
                    6068
                    6069 * PROCEDURE GET_VRAMQ (TABLE_CODE:BYTE;START_INDEX:BYTE;SLICE:BYTE;
                    6070 *                      VAR DATA:BUFFER;ITEM_COUNT:INTEGER);
                    6071
                    6072 * THIS IS THE PASCAL ENTRY POINT TO INIT_TABLE_
                    6073
  1880 00050001     6074 GET_VRAM_P     DEFW            5,1,1,1,-2,2
  1884 00010001
  1888 FFFE0002
                    6075 * THIS IS THE PARAMETER DESCRIPTOR FOR INIT_TABLEQ
                    6076
  188C              6077 GET_VRAMQ
  188C 011B80       6078                  LD              BC,GET_VRAM_P
  188F 11738A       6079                  LD              DE,PARAM_AREA
  1892 CD0098       6080                  CALL            PARAM_
  1895 3A738A       6081                  LD              A,[PARAM_AREA]
  1898 FD5B738A     6082                  LD              DE,[PARAM_AREA+1]
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  1B9C FD2A73BF                   LD      IY,[PARAM_AREA+5]
  1BA0 2A73BD                     LD      HL,[PARAM_AREA+3]


  1BA3                   GET_VRAM_                ;GETS A CERTAIN NUMBER OF BYTES FROM VRAM
                                                  ;AND PUTS THEM IN A BUFFER
                                                  ;IN: TABLE CODE IN A
                                                  ;    0=SPRITE NAME TABLE
                                                  ;    1=SPRITE GENERATOR TABLE, 2=PATTERN_NAME
                                                  ;    TABLE,3= PATTERN GENERATOR TABLE, 4=
                                                  ;    COLOR TABLE
                                                  ; START_INDEX IN DE,
                                                  ; DATA BUFFER IN HL, AND COUNT IN IY.

  1BA3 CD1BAA                     CALL SET_COUNT
  1BA6 CD1D3E                     CALL VRAM_READ
  1BA9 C9                         RET


  1BAA                   SET_COUNT                ;CALLED BY PUT_VRAM_ AND GET_VRAM_
                                                  ;SETS BYTE COUNT AND INDEX FOR
                                                  ;WRITES AND READS TO AND FROM VRAM
                                                  ; TABLE          BYTES/ITEM
                                                  ; SPRITE_NAME        4
                                                  ; SPRITE_GEN         8
                                                  ; PATTERN_NAME       1
                                                  ; PATTERN_GEN        8
                                                  ; COLOR (MODE 1)     1
                                                  ; COLOR (MODE 2)     8

  1BAA FD2273FE                   LD   [SAVED_COUNT],IY           ;SAVE COUNT
  1BAE DD2173F2                   LD   IX,VRAM_ADDR_TABLE         ;POINTER TO SAVED VRAM ADDRESSES
  1BB2 4F                         LD   C,A            ;SAVE TABLE CODE
  1BB3 0600                       LD   B,0            ;BC USED AS INDEX
  1BB5 FE04                       CP   4                          ;CHECK FOR CASE OF COLOR TABLE
  1BB7 2007                       JR   NZ,SET_COUNT10
                                  ;**  COLOR TABLE, CHECK IF MODE 1
  1BB9 3A73C3                     LD   A,[VDP_MODE_WORD]          ;CHECK FOR VDP GRAPHICS MODE
  1BBC CB4F                       BIT  1,A
  1BBE 282C                       JR   Z,SET_COUNT20
                                  ;**  NOT MODE 1, ADJUST ITEM COUNT AND INDEX
  1BC0                   SET_COUNT10
  1BC0 FD211BFF                   LD   IY,SHIFT_CT    ;GET ITEM COUNT CONVERSION FACTORS
  1BC4 FD09                       ADD  IY,BC
  1BC6 FD7E00                     LD   A,[IY+0]       ;SHIFT COUNT FOR MULTIPLICATION
  1BC9 FE00                       CP   0
  1BCB 281F                       JR   Z,SET_COUNT20
  1BCD                   ADJUST_INDEX                 ;MULTIPLY ITEM_INDEX TO GET BYTE_INDEX
  1BCD CB23                       SLA  E
  1BCF CB12                       RL   D
  1BD1 3D                         DEC  A
  1BD2 20F9                       JR   NZ,ADJUST_INDEX
  1BD4                   END_ADJ_INDEX
  1BD4 C5                         PUSH BC             ;SAVE TABLE_CODE/INDEX
  1BD5 ED4B73FE                   LD   BC,[SAVED_COUNT]
  1BD9 FD7E00                     LD   A,[IY+0]       ;SHIFT COUNT FOR MULTIPLICATION
  1BDC FE00                       CP   0
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

  1BDE 2808         6140              JR   Z,END_ADJ_COUNT
  1BE0              6141 ADJUST_COUNT                          ;MULTIPLY ITEM_COUNT TO GET BYTE COUNT
  1BE0 CB21         6142              SLA  C
  1BE2 CB10         6143              RL   B
  1BE4 3D           6144              DEC  A
  1BE5 20F9         6145              JR   NZ,ADJUST_COUNT
  1BE7 ED4373FE     6146              LD   [SAVED_COUNT],BC                  ;SAVE ADJUSTED COUNT
  1BEB              6147 END_ADJ_COUNT
                    6148
  1BEB C1           6149              POP  BC              ;RESTORE TABLE_CODE/INDEX
  1BEC              6150 SET_COUNT20
  1BEC E5           6151              PUSH HL
  1BED DD09         6152              ADD  IX,BC           ;GET TABLE ADDRESS IN VRAM
  1BEF DD09         6153              ADD  IX,BC
  1BF1 DD6E00       6154              LD   L,[IX+0]        ;LOW ORDER OF VRAM ADDRESS
  1BF4 DD6601       6155              LD   H,[IX+1]        ;HIGH ORDER OF VRAM ADDRESS
  1BF7 19           6156              ADD  HL,DE           ;ADD BYTE_INDEX TO GET VRAM START ADDRESS
  1BF8 EB           6157              EX   DE,HL           ;VRAM DESTINATION NOW IN DE
  1BF9 E1           6158              POP  HL              ;RESTORE DATA POINTER
  1BFA ED4B73FE     6159              LD   BC,[SAVED_COUNT] ;RESTORE ADJUSTED COUNT
  1BFE              6160 SET_COUNTX
  1BFE C9           6161              RET
                    6162
                    6163
  1BFF              6164 SHIFT_CT
  1BFF 0203000303   6165                   DEFB         2,3,0,3,3
                    6166
                    6167
                    6168 * PROCEDURE PUT_VRAMQ (TABLE_CODE:BYTE;START_INDEX:BYTE;SLICE:BYTE;
                    6169 *                       VAR DATA:BUFFER;ITEM_COUNT:INTEGER);
                    6170
                    6171 * THIS IS THE PASCAL ENTRY POINT TO INIT_TABLE_
                    6172
  1C04 00050001     6173 PUT_VRAM_P        DEFW         5,1,1,1,-2,2
  1C08 00010001
  1C0C FFFE0002
                    6174 * THIS IS THE PARAMETER DESCRIPTOR FOR INIT_TABLEQ
                    6175
  1C10              6176 PUT_VRAMQ
  1C10 011C04       6177                   LD           BC,PUT_VRAM_P
  1C13 11738A       6178                   LD           DE,PARAM_AREA
  1C16 CD0098       6179                   CALL         PARAM_
  1C19 3A738A       6180                   LD           A,[PARAM_AREA]
  1C1C ED5B738B     6181                   LD           DE,[PARAM_AREA+1]
  1C20 FD2A738F     6182                   LD           IY,[PARAM_AREA+5]
  1C24 2A738D       6183                   LD           HL,[PARAM_AREA+3]
                    6184
  1C27              6185 PUT_VRAM_                      ;WRITES A CERTAIN NUMBER OF BYTES TO VRAM
                    6186                                ;FROM A BUFFER.
                    6187                                ;IN: TABLE CODE IN A
                    6188                                ;     0=SPRITE NAME TABLE
                    6189                                ;     1=SPRITE GENERATOR TABLE, 2=PATTERN NAME
                    6190                                ;     TABLE,3= PATTERN GENERATOR TABLE, 4=
                    6191                                ;     COLOR TABLE
                    6192                                ; START INDEX IN DE,
                    6193                                ; DATA BUFFER IN HL, AND COUNT IN IY.
                    6194
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  1C27 F5                           PUSH  AF
                         *    IF (TABLE_CODE = SPRITE_NAME_TABLE) AND (MUX_SPRITES = TRUE) THEN
  1C28 FE00                         CP 0
  1C2A 2022                         JR NZ,ELSEZZ
  1C2C 3A73C7                       LD A,[MUX_SPRITES]
  1C2F FE01                         CP 1
  1C31 201B                         JR NZ,ELSEZZ

                         *      WRITE ENTRY TO LOCAL TABLE
  1C33 F1                           POP  AF                ; CLEAR STACK
  1C34 E5                           PUSH  HL  ; [SP] = DATA BUFFER

  1C35 2A8002                       LD  HL,[LOCAL_SPR_TBL] ; CALCULATE ADDRESS IN TABLE
  1C38 7B                           LD  A,E
  1C39 CB27                         SLA  A
  1C3B CB27                         SLA  A
  1C3D 5F                           LD  E,A
  1C3E 19                           ADD HL,DE
  1C3F EB                           EX  DE,HL

  1C40 FDE5                         PUSH  IY                    ; CALCULATE BYTE COUNT
  1C42 C1                           POP  BC
  1C43 79                           LD  A,C
  1C44 CB27                         SLA  A
  1C46 CB27                         SLA  A
  1C48 4F                           LD  C,A

  1C49 E1                           POP  HL                     ; STACK CLEAR
  1C4A EDB0                         LDIR                        ; PERFORM WRITE FROM BUFFER

  1C4C 1807                         JR END_IFZZ
                         *    ELSE
  1C4E                   ELSEZZ
  1C4E F1                           POP  AF
  1C4F CD1BAA                       CALL SET_COUNT
  1C52 CD1D01                       CALL VRAM_WRITE
                         *    END IF
  1C55                   END_IFZZ
  1C55 C9                           RET


                         * PROCEDURE INIT_SPR_ORDERQ  (SPRITE_COUNT:BYTE);

                         * THIS IS THE PASCAL ENTRY POINT TO THE INIT_SPR_ORDER_ ROUTINE.

  1C56 00010001          INIT_SPR_P        DEFW            1,1

        <1C5A>           INIT_SPR_ORDERQ EQU               $
  1C5A 011C56                          LD                  BC,INIT_SPR_P
  1C5D 1173BA                          LD                  DE,PARAM_AREA
  1C60 CD0098                          CALL                PARAM
  1C63 3A73BA                          LD                  A,[PARAM_AREA]

  1C66                   INIT_SPR_ORDER_                    ;INITIALIZES THE SPRITE DISPLAY ORDER
                                                            ;LIST IN RAM TO DEFAULT ORDER (0...31)
                                                            ;IN: NUMBER OF SPRITES TO ORDER IN 'A'
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

 1C66 47            6252            LD   B,A              ;SAVE SPRITE COUNT
 1C67 AF            6253            XOR  A
 1C68 2A8004        6254            LD   HL,[SPRITE_ORDER]
 1C6B               6255 INIT_SPR10
 1C6B 77            6256            LD   [HL],A
 1C6C 23            6257            INC  HL
 1C6D 3C            6258            INC  A
 1C6E B8            6259            CP   B
 1C6F 20FA          6260            JR   NZ,INIT_SPR10
 1C71 C9            6261            RET
                    6262
                    6263 * PROCEDURE WR_SPR_NM_TBLQ  (SPRITE_COUNT:BYTE);
                    6264
                    6265 * THIS IS THE PASCAL ENTRY POINT TO THE WR_SPR_NM_TBL_ ROUTINE.
                    6266
 1C72 00010001      6267 WR_SPR_P          DEFW           1,1
                    6268
        <1C76>      6269 WR_SPR_NM_TBLQ   EQU            $
 1C76 011C72        6270                   LD             BC,WR_SPR_P
 1C79 11738A        6271                   LD             DE,PARAM_AREA
 1C7C CD0098        6272                   CALL           PARAM_
 1C7F 3A738A        6273                   LD             A,[PARAM_AREA]
                    6274
 1C82               6275 WR_SPR_NM_TBL_                    ;WRITES SPRITE_NAME_TABLE TO VRAM
                    6276                                  ;USING THE SPRITE ORDER LIST.
                    6277                                  ;
                    6278                                  ;    NUMBER OF SPRITES TO WRITE IN 'A'
                    6279
                    6280 ;MODE_0_PORT      EQU            0BEH
                    6281 ;MODE_1_PORT      EQU            0BFH
                    6282
 1C82 DD2A8004      6283            LD   IX,[SPRITE_ORDER]  ;LIST OF DISPLAY ORDERS
                    6284 ;          LD   [SPRITE_CT],A     ;SAVE COUNT
 1C86 F5            6285            PUSH AF                ;SAVE COUNT
 1C87 FD2173F2      6286            LD   IY,VRAM_ADDR_TABLE
 1C8B FD5E00        6287            LD   E,[IY+0]
 1C8E FD5601        6288            LD   D,[IY+1]          ;VRAM SPRITE_NAME_TABLE ADDRESS NOW IN DE.
                    6289
                    6290            ;**  SET UP VDP TO RECEIVE DATA
                    6291
 1C91 7B            6292            LD   A,E
 1C92 D3BF          6293            OUT  [MODE_1_PORT],A
 1C94 7A            6294            LD   A,D
 1C95 F640          6295            OR   40H
 1C97 D3BF          6296            OUT  [MODE_1_PORT],A
                    6297
 1C99 F1            6298            POP  AF                ;RESTORE COUNT
 1C9A               6299 OUTPUT_LOOP_TABLE_MA
 1C9A 2A8002        6300            LD   HL,[LOCAL_SPR_TBL] ;RESTORE RAM SPRITE_NAME_TABLE POINTER
 1C9D DD4E00        6301            LD   C,[IX+0]          ;DISPLAY ORDER
 1CA0 DD23          6302            INC  IX                ;ADVANCE DISPLAY ORDER POINTER
 1CA2 0600          6303            LD   B,0
 1CA4 09            6304            ADD  HL,BC
 1CA5 09            6305            ADD  HL,BC
 1CA6 09            6306            ADD  HL,BC
 1CA7 09            6307            ADD  HL,BC
                    6308
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                     6309             ;**  OUTPUT TO VRAM THROUGH VDP
                     6310
 1CA8 0604           6311             LD   B,4             ;ELEM.NT COUNT FOR ONE SPRITE
 1CAA 0EBE           6312             LD   C,MODE_0_PORT   ;OUTPUT PORT IN 'C'
 1CAC                6313 OUTPUT_LOOP10
 1CAC EDA3           6314             OUTI
 1CAE 00             6315             NOP                  ;DELAY
 1CAF 00             6316             NOP
 1CB0 20FA           6317             JR   NZ,OUTPUT_LOOP10
                     6318 ;           LD   A,[SPRITE_CT]
                     6319 ;           DEC  A
                     6320 ;           LD   [SPRITE_CT],A
 1CB2 3D             6321             DEC  A
 1CB3 20E5           6322             JR   NZ,OUTPUT_LOOP_TABLE_MA
 1CB5 C9             6323             RET
                     6324
                     6325                  GLB          VRAM_ADDR_TABLE
                     6326                  GLB          SPRITENAMETBL
                     6327                  GLB          SPRITEGENTBL
                     6328                  GLB          PATTRNNAMETBL
                     6329                  GLB          PATTRNGENTBL
                     6330                  GLB          COLORTABLE
                     6331                  DATA
 73F2                6332 VRAM_ADDR_TABLE
 73F2                6333 SPRITENAMETBL    DEFS         2
 73F4                6334 SPRITEGENTBL     DEFS         2
 73F6                6335 PATTRNNAMETBL    DEFS         2
 73F8                6336 PATTRNGENTBL     DEFS         2
 73FA                6337 COLORTABLE       DEFS         2
                     6338 * THIS TABLE HOLDS THE BASE ADDRESSES OF ALL THE VRAM TABLES.
                     6339
                     6340
 73FC                6341 SAVE_TEMP        DEFS 2
 73FE                6342 SAVED_COUNT      DEFS 2
                     6343
                     6344 ;                COMN
                     6345
                     6346 ;PARAM_AREA       DEFS        6
                     6347 * THIS IS THE PARAMETER PASSING AREA FOR THE PASCAL ENTRY POINTS TO
                     6348 * ROUTINES IN THIS MODULE. IT IS HELD IN COMMON WITH OTHER SUCH ENTRY
                     6349 * POINTS FOR OTHER MODULES.
                     6350  PROG
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

                    6465 * THIS IS THE PARAMETER DESCRIPTOR FOR REG_WRITEQ
                    6466
                    6467 * BEGIN REG_WRITE
                    6468                  GLB           REG_WRITEQ
          <1CBC>    6469 REG_WRITEQ       EQU           $
 1CBC 011CB6        6470                  LD            BC,REG_WRITE_P
 1CBF 11738A        6471                  LD            DE,PARAM_AREA
 1CC2 CD0098        6472                  CALL          PARAM_
 1CC5 2A738A        6473                  LD            HL,[PARAM_AREA]
 1CC8 4C            6474                  LD            C,H
 1CC9 45            6475                  LD            B,L
                    6476
                    6477                  GLB           REG_WRITE
          <1CCA>    6478 REG_WRITE        EQU           $
                    6479
                    6480 *    VALUE =; CTRL_PORT
 1CCA 79            6481                  LD            A,C
 1CCB D38F          6482                  OUT           [CTRL_PORT],A
                    6483
                    6484 *    REGISTER + 80H =; CTRL_PORT
 1CCD 78            6485                  LD            A,B
 1CCE C680          6486                  ADD           A,80H
 1CD0 D38F          6487                  OUT           [CTRL_PORT],A
                    6488
                    6489 *    IF REGISTER = 0 THEN VDP_MODE_WORD[0] := VALUE
 1CD2 78            6490                  LD            A,B
 1CD3 FE00          6491                  CP            0
 1CD5 2004          6492                  JR            NZ,NOT_REG_0
 1CD7 79            6493                  LD            A,C
 1CD8 3273C3        6494                  LD            [VDP_MODE_WORD],A
          <1CDB>    6495 NOT_REG_0        EQU           $
                    6496
                    6497 *    IF REGISTER = 1 THEN VDP_MODE_WORD[1] := VALUE
 1CDB 78            6498                  LD            A,B
 1CDC FE01          6499                  CP            1
 1CDE 2004          6500                  JR            NZ,NOT_REG_1
 1CE0 79            6501                  LD            A,C
 1CE1 3273C4        6502                  LD            [VDP_MODE_WORD+1],A
          <1CE4>    6503 NOT_REG_1        EQU           $
                    6504
                    6505 * END REG_WRITE
 1CE4 C9            6506                  RET
                    6507
                    6508 * PROCEDURE  VRAM_WRITE (VAR DATA:BUFFER;DEST:INTEGER;COUNT:INTEGER)
                    6509
                    6510 * VAR DATA (POINTER TO DATA BUFFER) IS PASSED IN HL
                    6511 * DEST IS PASSED IN DE
                    6512 * COUNT IS PASSED IN BC
                    6513 * DESTROYS: ALL
                    6514
 1CE5 0003FFFE      6515 VRAM_WRITE_P     DEFW          3,-2,2,2
 1CE9 00020002
                    6516 * THIS IS THE PARAMETER DESCRIPTOR FOR VRAM_WRITEQ
                    6517
                    6518 * BEGIN VRAM_WRITE
                    6519                  GLB           VRAM_WRITEQ
          <1CED>    6520 VRAM_WRITEQ      EQU           $
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  1CED 011CE5      6521                LD          BC,VRAM_WRITE_P
  1CF0 11738A      6522                LD          DE,PARAM_AREA
  1CF3 CD0098      6523                CALL        PARAM
  1CF6 2A738A      6524                LD          HL,[PARAM_AREA]
  1CF9 ED5B738C    6525                LD          DE,[PARAM_AREA+2]
  1CFD ED4B738E    6526                LD          BC,[PARAM_AREA+4]
                   6527
                   6528                GLB         VRAM_WRITE
          <1D01>   6529 VRAM_WRITE     EQU         $
                   6530
                   6531 * DEST := DEST + 4000H
  1D01 E5          6532                PUSH        HL
  1D02 D5          6533                PUSH        DE
  1D03 E1          6534                POP         HL
  1D04 114000      6535                LD          DE,4000H
  1D07 19          6536                ADD         HL,DE
                   6537
                   6538 * LOW BYTE OF DEST =; CTRL_PORT
  1D08 7D          6539                LD          A,L
  1D09 D3BF        6540                OUT         [CTRL_PORT],A
                   6541
                   6542 * HIGH BYTE OF DEST =; CTRL_PORT
  1D0B 7C          6543                LD          A,H
  1D0C D3BF        6544                OUT         [CTRL_PORT],A
                   6545
                   6546 * DATA =; DATA_PORT
  1D0E C5          6547                PUSH        BC
  1D0F D1          6548                POP         DE
  1D10 E1          6549                POP         HL
  1D11 0EBE        6550                LD          C,DATA_PORT
  1D13 43          6551                LD          B,E
          <1D14>   6552 OUTPUT_LOOP    EQU         $
  1D14 EDA3        6553                OUTI
  1D16 00          6554                NOP
  1D17 00          6555                NOP
  1D18 C21D14      6556                JP          NZ,OUTPUT_LOOP
  1D1B 15          6557                DEC         D
  1D1C FA1D21      6558                JP          M,END_OUTPUT
  1D1F 20F3        6559                JR          NZ,OUTPUT_LOOP
                   6560
                   6561 * END VRAM_WRITE
          <1D21>   6562 END_OUTPUT     EQU         $
  1D21 C9          6563                RET
                   6564
                   6565 * PROCEDURE  VRAM_READ  (VAR DATA:BUFFER;SRCE:INTEGER;COUNT:INTEGER)
                   6566
                   6567 * VAR DATA (POINTER TO DATA BUFFER) IS PASSED IN HL
                   6568 * SRCE IS PASSED IN DE
                   6569 * COUNT IS PASSED IN BC
                   6570 * DESTROYS: ALL
                   6571
  1D22 0003FFFE    6572 VRAM_READ_P    DEFW        3,-2,2,2
  1D26 00020002
                   6573 * THIS IS THE PARAMETER DESCRIPTOR FOR VRAM_READQ
                   6574
                   6575 * BEGIN VRAM_READ
                   6576                GLB         VRAM_READQ
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

          <1D2A>  6577 VRAM_READQ      EQU       $
1D2A 011D22       6578                 LD        BC,VRAM_READ_P
1D2D 1173BA       6579                 LD        DE,PARAM_AREA
1D30 CD0098       6580                 CALL      PARAM_
1D33 2A73BA       6581                 LD        HL,[PARAM_AREA]
1D36 ED5B73BC     6582                 LD        DE,[PARAM_AREA+2]
1D3A ED4B73BE     6583                 LD        BC,[PARAM_AREA+4]
                  6584
                  6585                 GLB       VRAM_READ
          <1D3E>  6586 VRAM_READ       EQU       $
                  6587
                  6588 * LOW BYTE OF SRCE =; CTRL_PORT
1D3E 7B           6589                 LD        A,E
1D3F D3BF         6590                 OUT       [CTRL_PORT],A
                  6591
                  6592 * HIGH BYTE OF SRCE =; CTRL_PORT
1D41 7A           6593                 LD        A,D
1D42 D3BF         6594                 OUT       [CTRL_PORT],A
                  6595
                  6596 * DATA =; DATA_PORT
1D44 C5           6597                 PUSH      BC
1D45 D1           6598                 POP       DE
1D46 0EBE         6599                 LD        C,DATA_PORT
1D48 43           6600                 LD        B,E
          <1D49>  6601 INPUT_LOOP      EQU       $
1D49 EDA2         6602                 INI
1D4B 00           6603                 NOP
1D4C 00           6604                 NOP
1D4D C21D49       6605                 JP        NZ,INPUT_LOOP
1D50 15           6606                 DEC       D
1D51 FA1D56       6607                 JP        M,END_INPUT
1D54 20F3         6608                 JR        NZ,INPUT_LOOP
                  6609
                  6610 * END VRAM_READ
          <1D56>  6611 END_INPUT       EQU       $
1D56 C9           6612                 RET
                  6613
                  6614 * FUNCTION   REG_READ:BYTE
                  6615
                  6616 * FUNCTION OUTPUT RETURNED IN A
                  6617 * DESTROYS A ONLY
                  6618
                  6619 * BEGIN REG_READ
                  6620                 GLB       REG_READ
          <1D57>  6621 REG_READ        EQU       $
                  6622
                  6623 *   REG_READ := CTRL_PORT
1D57 DBBF         6624                 IN        A,[CTRL_PORT]
                  6625
                  6626 * END REG_READ
1D59 C9           6627                 RET
                  6628
                  6629  PROG
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                    6631
                    6632 * THIS IS A PACKAGE OF ROUTINES THAT ALLOW APPLICATIONS PROGRAMMERS TO
                    6633 * OPERATE ON SHAPE GENERATORS. EACH OF THEM TAKES, AS INPUTS, AN AREA
                    6634 * IN ONE OF THE GENERATOR TABLES IN WHICH THE GENERATORS TO BE OPERATED
                    6635 * UPON RESIDE, A COUNT OF THE GENERATORS TO BE USED, AND AN AREA OF THE
                    6636 * SAME TABLE INTO WHICH THE RESULTS ARE TO BE PUT.THE ONLY RAM AREA THEY
                    6637 * IS IN THE WORK_BUFFER A POINTER TO WHICH IS DECLARED AS AN EXTERNAL
                    6638 * AND DEFINED IN THE CARTRIDGE.
                    6639
                    6640 ************ NOTE:                                   ******************
                    6641 ************      THESE ROUTINES WRITE TO AND READ  ******************
                    6642 ************      WITHOUT POSSIBILITY OF DEFERAL    ******************
                    6643 ************      AND SHOULD NOT BE USED IN ANY     ******************
                    6644 ************      SITUATION WHERE THEY MAY BE       ******************
                    6645 ************      INTERRUPTED.                      ******************
                    6646 ********************************************************************
                    6647
                    6648 ;                EXT             WORK_BUFFER
                    6649 * POINTER TO THE WORK BUFFER DEFINED BY THE CARTRIDGE PROGRAMMER
                    6650
                    6651 ;                EXT             VDP_MODE_WORD
                    6652 * THE WORD IN OS RAM THAT DESCRIBES THE CURRENT GRAPHICS MODE.
                    6653
                    6654 ;                EXT             GET_VRAM_
                    6655 ;                EXT             PUT_VRAM_
                    6656 * EXTERNAL OS ROUTINES IN TABLE_MANAGER MODULE
                    6657
                    6658 ;                EXT             MIRROR_L_R
                    6659 ;                EXT             MIRROR_U_D
                    6660 ;                EXT             ROTATE
                    6661 ;                EXT             MAGNIFY
                    6662 ;                EXT             QUADRUPLE
                    6663 * EXTERNAL ROUTINES THAT PERFORM BLOCK OPERATIONS
                    6664
                    6665 ;TRUE            EQU             1
                    6666 ;FALSE           EQU             0
                    6667 * VALUES FOR BOOLEAN FLAGS
                    6668
         <0003>     6669 PATTERN_GEN      EQU             3
         <0004>     6670 COLOR_TABLE      EQU             4
                    6671 * VALUES FOR TABLE CODE
                    6672
                    6673 * PROCEDURE REFLECT_VERTICAL (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))
                    6674
                    6675 * REFLECT REFLECTS EACH OF A BLOCK OF GENERATORS FROM VRAM AROUND
                    6676 * THE VERTICAL AXIS. IF THE GENERATORS ARE FROM THE PATTERN PLANE
                    6677 * AND THE GRAPHICS MODE IS 2, THEN THE ROUTINE ALSO COPIES THE
                    6678 * CORRESPONDING COLOR GENERATORS., OTHERWISE IS ASSUMES THAT THE COLOR
                    6679 * DATA HAS ALREADY BEEN SET UP.
                    6680
                    6681 * BEGIN REFLECT_VERTICAL
                    6682                  GLB             RFLCT_VERT
1D5A                6683 RFLCT_VERT                       ; ACTUAL ROUTINE NAME EXISTS IN OS
                    6684                                  ; JUMP TABLE ONLY
                    6685
                    6686 *  SET OPERATION CODE
1D5A DD211D96       6687                  LD              IX,RFLCT_VERT
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                 *  CONTINUE BELOW
1D5E 1810                        JR              CONTINUE_GRAPHICS

                 * PROCEDURE REFLECT_HORIZONTAL (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))

                 * REFLECT_HORIZONTAL REFLECTS EACH OF A BLOCK OF GENERATORS FROM VRAM
                 * AROUND THE HORIZONTAL AXIS. IF THE GENERATORS ARE FROM THE PATTERN
                 * PLANE ANDTHE GRAPHICS MODE IS 2 THEN IT REFLECTS THE CORRESPONDING
                 * COLOR GENERATORS AS WELL.

                 * BEGIN REFLECT_HORIZONTAL
                                 GLB             RFLCT_HOR
1D60             RFLCT_HOR                       ; ACTUAL ROUTINE NAME EXISTS IN OS
                                                 ; JUMP TABLE ONLY

                 *  SET OPERATION CODE
1D60 DD211DB7                    LD              IX,RFLCT_HOR_

                 *  CONTINUE BELOW
1D64 180A                        JR              CONTINUE_GRAPHICS

                 * PROCEDURE ROTATE_90 (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))

                 * ROTATE_90 ROTATES EACH OF A BLOCK OF GENERATORS FROM VRAM 90 DEGREES
                 * CLOCKWISE. IF THE GENERATORS ARE FROM THE PATTERN PLANE AND THE
                 * GRAPHICS MODE IS 2 THEN THE ROUTINE COPIES THE CORRESPONDING COLOR
                 * ENTRIES AS WELL.

                 * BEGIN ROTATE_90
                                 GLB             ROT_90
1D66             ROT_90                          ; ACTUAL ROUTINE NAME EXISTS IN OS
                                                 ; JUMP TABLE ONLY

                 *  SET OPERATION CODE
1D66 DD211DE5                    LD              IX,ROT_90_

                 *  CONTINUE BELOW
1D6A 1804                        JR              CONTINUE_GRAPHICS

                 * PROCEDURE ENLARGE (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))

                 * ENLARGE TAKES EACH OF A BLOCK OF GENERATORS AND ENLARGES IT INTO
                 * A BLOCK OF FOUR GENERATORS WHEREIN EACH PIXEL OF THE ORIGINAL
                 * GENERATOR IS EXPANDED TO FOUR PIXELS IN THE NEW ONES. IF THE
                 * GENERATORS ARE FROMTHE PATTERN PLANE AND THE GRAPHICS MODE IS 2
                 * THE THE ROUTINE ALSO QUADRUPLES EACH OF THE CORRESPONDING COLOR
                 * GENERATORS AS WELL.

                 * BEGIN ENLARGE
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                                 GLB          ENLRG
 1D6C                  ENLRG                  ; ACTUAL ROUTINE NAME EXISTS IN OS
                                              ; JUMP TABLE ONLY


                       *  SET OPERATION CODE
 1D6C DD211E07                   LD           IX,ENLRG_


                       *  CONTINUE EXECUTION HERE
 1D70                  CONTINUE_GRAPHICS

                       *  SAVE ALL ENTRY PARAMETERS
 1D70 D9                         EXX
 1D71 08                         EX           AF,AF'
 1D72 DDE5                       PUSH         IX               ; [SP] = OPERATION CODE

                       *  REPEAT
 1D74                  MAIN_LOOP

                       *     GET_VRAM_ (TABLE_CODE,SOURCE,WORK_BUFFER[0..7],1)
 1D74 08                         EX           AF,AF'
 1D75 F5                         PUSH         AF
 1D76 08                         EX           AF,AF'
 1D77 F1                         POP          AF
 1D78 D9                         EXX
 1D79 D5                         PUSH         DE
 1D7A D9                         EXX
 1D7B D1                         POP          DE
 1D7C FD210001                   LD           IY,1
 1D80 2A8006                     LD           HL,[WORK_BUFFER]
 1D83 CD1BA3                     CALL         GET_VRAM_

                       *     EXECUTE ENCODED OPERATION BELOW
 1D86 DDE1                       POP          IX
 1D88 DDE5                       PUSH         IX
 1D8A DDE9                       JP           [IX]
         <1D8C>        RETURN_HERE     EQU    $

                       *     SOURCE : SUCC (SOURCE)
 1D8C 13                         INC          DE

                       *     COUNT := PRED (COUNT)
 1D8D 0B                         DEC          BC

                       *  UNTIL COUNT = 0
 1D8E 78                         LD           A,B
 1D8F B1                         OR           C
 1D90 D9                         EXX
 1D91 20E1                       JR           NZ,MAIN_LOOP

                       * END (ALL)
 1D93                  ALL_X
 1D93 DDE1                       POP          IX               ; CLEAR STACK
 1D95 C9                         RET

 1D96                  RFLCT_VERT_
```

```
OCATION OBJECT CODE LINE     SOURCE LINE

                 * OPERATIONS SPECIFIC TO REFLECT_VERTICAL ROUTINE

                 *   MIRROR_L_R(WORK_BUFFER[0..7],WORK_BUFFER[8..15])
  1D96 2A8006                    LD              HL,[WORK_BUFFER]
  1D99 010008                    LD              BC,8
  1D9C E5                        PUSH            HL
  1D9D D1                        POP             DE
  1D9E 09                        ADD             HL,BC
  1D9F EB                        EX              DE,HL
  1DA0 CD1F00                    CALL            MIRROR_L_R

                 *   PUT_VRAM (TABLE_CODE,DESTINATION,WORK_BUFFER[8..15],1)
  1DA3 CD1E72                    CALL            PUT_TABLE

                 *   IF COLOR_TEST THEN
  1DA6 CD1E5D                    CALL            COLOR_TEST
  1DA9 FE01                      CP              TRUE
  1DAB 2006                      JR              NZ,END_IF_1_GRAPHICS

                 *     GET_VRAM(COLOR_TABLE,SOURCE,WORK_BUFFER[0..7],1)
  1DAD CD1E89                    CALL            GET_COLOR

                 *     PUT_VRAM(COLOR_TABLE,DESTINATION,WORK_BUFFER[0..7],1)
  1DB0 CD1E9A                    CALL            PUT_COLOR

                 *   END IF
  1DB3           END_IF_1_GRAPHICS

                 *   DESTINATION := SUCC (DESTINATION)
  1DB3 D9                        EXX
  1DB4 23                        INC             HL

                 * END
  1DB5 1805                      JR              RETURN_HERE



  1DB7           RFLCT_HOR
                 * OPERATIONS SPECIFIC TO REFLECT_HORIZONTAL ROUTINE

                 *   MIRROR_U_D(WORK_BUFFER[0..7],WORK_BUFFER[8..15])
  1DB7 2A8006                    LD              HL,[WORK_BUFFER]
  1DBA 010008                    LD              BC,8
  1DBD E5                        PUSH            HL
  1DBE D1                        POP             DE
  1DBF 09                        ADD             HL,BC
  1DC0 EB                        EX              DE,HL
  1DC1 CD1F4E                    CALL            MIRROR_U_D

                 *   PUT_VRAM (TABLE_CODE,DESTINATION,WORK_BUFFER[8..15],1)
  1DC4 CD1E72                    CALL            PUT_TABLE

                 *   IF COLOR_TEST THEN
  1DC7 CD1E5D                    CALL            COLOR_TEST
  1DCA FE01                      CP              TRUE
  1DCC 2013                      JR              NZ,END_IF_2_GRAPHICS
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                          *     GET_VRAM(COLOR_TABLE,SOURCE,WORK_BUFFER[0..7],1)
 1DCE CD1E89                           CALL             GET_COLOR

                          *     MIRROR_U_D(WORK_BUFFER[0..7],WORK_BUFFER[8..15])
 1DD1 2A8006                           LD               HL,[WORK_BUFFER]
 1DD4 010008                           LD               BC,8
 1DD7 E5                               PUSH             HL
 1DD8 D1                               POP              DE
 1DD9 09                               ADD              HL,BC
 1DDA EB                               EX               DE,HL
 1DDB CD1F4E                           CALL             MIRROR_U_D

                          *     PUT_VRAM(COLOR_TABLE,DESTINATION,WORK_BUFFER[8..15],1)
 1DDE CD1E9A                           CALL             PUT_COLOR

                          *   END_IF
 1DE1                     END_IF_2_GRAPHICS

                          *   DESTINATION := SUCC (DESTINATION)
 1DE1 D9                               EXX
 1DE2 23                               INC              HL

                          * END
 1DE3 18A7                             JR               RETURN_HERE



 1DE5                     ROT_90
                          * OPERATIONS SPECIFIC TO THE ROTATE_90 ROUTINE


                          *   ROTATE(WORK_BUFFER[0..7],WORK_BUFFER[8..15])
 1DE5 2A8006                           LD               HL,[WORK_BUFFER]
 1DE8 010008                           LD               BC,8
 1DEB E5                               PUSH             HL
 1DEC D1                               POP              DE
 1DED 09                               ADD              HL,BC
 1DEE EB                               EX               DE,HL
 1DEF CD1F12                           CALL             ROTATE

                          *   PUT_VRAM (TABLE_CODE,DESTINATION,WORK_BUFFER[8..15],1)
 1DF2 CD1E72                           CALL             PUT_TABLE

                          *   IF COLOR_TEST THEN
 1DF5 CD1E5D                           CALL             COLOR_TEST
 1DF8 FE01                             CP               TRUE
 1DFA 2006                             JR               NZ,END_IF_3_GRAPHICS

                          *     GET_VRAM(COLOR_TABLE,SOURCE,WORK_BUFFER[0..7],1)
 1DFC CD1E89                           CALL             GET_COLOR

                          *     PUT_VRAM(COLOR_TABLE,DESTINATION,WORK_BUFFER[0..7],1)
 1DFF CD1E9A                           CALL             PUT_COLOR

                          *   END_IF
 1E02                     END_IF_3_GRAPHICS

```

```
XCATION OBJECT CODE LINE      SOURCE LINE

                    *   DESTINATION := SUCC (DESTINATION)
 1E02 D9                            EXX
 1E03 23                            INC             HL

                    * END
 1E04 C31D8C                        JP              RETURN_HERE



 1E07                ENLRG_
                    * OPERATIONS SPECIFIC TO THE ENLARGE ROUTINE

                    *   MAGNIFY(WORK_BUFFER[0..7],WORK_BUFFER[8..39])
 1E07 2A8006                        LD              HL,[WORK_BUFFER]
 1E0A 010008                        LD              BC,8
 1E0D E5                            PUSH            HL
 1E0E D1                            POP             DE
 1E0F 09                            ADD             HL,BC
 1E10 EB                            EX              DE,HL
 1E11 CD1EAB                        CALL            MAGNIFY

                    *   PUT_VRAM (TABLE_CODE,DESTINATION,WORK_BUFFER[8..39],4)
 1E14 08                            EX              AF,AF'
 1E15 F5                            PUSH            AF
 1E16 08                            EX              AF,AF'
 1E17 F1                            POP             AF
 1E18 D9                            EXX
 1E19 E5                            PUSH            HL
 1E1A D9                            EXX
 1E1B D1                            POP             DE
 1E1C 2A8006                        LD              HL,[WORK_BUFFER]
 1E1F 010008                        LD              BC,8
 1E22 09                            ADD             HL,BC
 1E23 FD210004                      LD              IY,4
 1E27 CD1C27                        CALL            PUT_VRAM_

                    *   IF COLOR_TEST THEN
 1E2A CD1E5D                        CALL            COLOR_TEST
 1E2D FE01                          CP              TRUE
 1E2F 2024                          JR              NZ,END_IF_4_GRAPHICS

                    *     GET_VRAM(COLOR_TABLE,SOURCE,WORK_BUFFER[0..7],1)
 1E31 CD1E89                        CALL            GET_COLOR

                    *     QUADRUPLE(WORK_BUFFER[0..7],WORK_BUFFER[8..39])
 1E34 2A8006                        LD              HL,[WORK_BUFFER]
 1E37 010008                        LD              BC,8
 1E3A E5                            PUSH            HL
 1E3B D1                            POP             DE
 1E3C 09                            ADD             HL,BC
 1E3D EB                            EX              DE,HL
 1E3E CD1EEA                        CALL            QUADRUPLE

                    *     PUT_VRAM(COLOR_TABLE,DESTINATION,WORK_BUFFER[8..39],4)
 1E41 3E04                          LD              A,COLOR_TABLE
 1E4                                EXX
 1E44 E5                            PUSH            HL
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  1E45 D9                               EXX
  1E46 D1                               POP          DE
  1E47 2A8006                           LD           HL,[WORK_BUFFER]
  1E4A 010008                           LD           BC,8
  1E4D 09                               ADD          HL,BC
  1E4E FD210004                         LD           IY,4
  1E52 CD1C27                           CALL         PUT_VRAM_

                            *    END_IF
  1E55                      END_IF_4_GRAPHICS

                            *    DESTINATION := DESTINATION + 4
  1E55 D9                               EXX
  1E56 23                               INC          HL
  1E57 23                               INC          HL
  1E58 23                               INC          HL
  1E59 23                               INC          HL

                            * END
  1E5A C31D8C                           JP           RETURN_HERE




  1E5D                      COLOR_TEST                      ; TESTS WHETHER PATTERN GENERATORS ARE
                                                            ; BEING MANIPULATED AND WHETHER THE
                                                            ; GRAPHICS MODE IS 2. IF SO THE ABOVE
                                                            ; ROUTINES NEED TO DEAL WITH THE COLOR
                                                            ; GENERATORS THAT CORRESPOND TO THE
                                                            ; PATTERN GENERATORS THEY ARE OPERATING
                                                            ; ON.

                                                            ; NOT INPUTS, RETURNS WITH TRUE (1) IN
                                                            ; A IF CONDITION IS TRUE, FALSE (0) IF
                                                            ; NOT.


                            * BEGIN COLOR_TEST

                            * CHECK TABLE CODE IN A'
  1E5D 08                               EX           AF,AF'
  1E5E F5                               PUSH         AF
  1E5F 08                               EX           AF,AF'
  1E60 F1                               POP          AF
  1E61 FE03                             CP           PATTERN_GEN
  1E63 200A                             JR           NZ,EXIT_FALSE

                            * CHECK MODE
  1E65 2173C3                           LD           HL,VDP_MODE_WORD
  1E68 CB4E                             BIT          1,[HL]
  1E6A 2803                             JR           Z,EXIT_FALSE

                            * EXIT HERE IF TRUE
  1E6C 3E01                             LD           A,TRUE
  1E6E C9                               RET

                            * EXIT HERE IF FALSE
```

```
CATION OBJECT CODE      SOURCE LINE

 1E6F 3E00         EXIT_FALSE   LD      A,FALSE
 1E71 C9                        RET
 1E72              PUT_TABLE
                                        ; PUTS THE CONTENTS OF WORK_BUFFER[8..15]
                                        ; IN VRAM AT THE GIVEN DESTINATION.
 1E72 08                        EX      AF,AF'
 1E73 F5                        PUSH    AF
 1E74 08                        EX      AF,AF'
 1E75 F1                        POP     AF
 1E76 D9                        EXX
 1E77 E5                        PUSH    HL
 1E78 D9                        EXX
 1E79 D1                        POP     DE
 1E7A 2A8006                    LD      HL,[WORK_BUFFER]
 1E7D 010008                    LD      BC,8
 1E80 09                        ADD     HL,BC
 1E81 FD210001                  LD      IY,1
 1E85 CD1C27                    CALL    PUT_VRAM_
 1E88 C9                        RET
 1E89              GET_COLOR
                                        ; GETS THE COLOR INFORMATION FROM
                                        ; THE APPROPRIATE PLACE IN VRAM
 1E89 3E04                      LD      A,COLOR_TABLE
 1E8B D9                        EXX
 1E8C D5                        PUSH    DE
 1E8D D9                        EXX
 1E8E D1                        POP     DE
 1E8F 2A8006                    LD      HL,[WORK_BUFFER]
 1E92 FD210001                  LD      IY,1
 1E96 CD1BA3                    CALL    GET_VRAM_
 1E99 C9                        RET
 1E9A              PUT_COLOR
                                        ; PUTS COLOR INFORMATION IN THE
                                        ; APPROPRIATE PLACE IN  VRAM
 1E9A 3E04                      LD      A,COLOR_TABLE
 1E9C D9                        EXX
 1E9D E5                        PUSH    HL
 1E9E D9                        EXX
 1E9F D1                        POP     DE
 1EA0 2A8006                    LD      HL,[WORK_BUFFER]
 1EA3 FD210001                  LD      IY,1
 1EA7 CD1C27                    CALL    PUT_VRAM_
 1EAA C9                        RET
                    PROG
```

```
LOCATION OBJECT CODE LINE      SOURCE LINE

                    
                    * THE ROUTINES IN THIS MODULE TAKE A SINGLE 8-BYTE BLOCK AS INPUT AND
                    * PRODUCE 4 8-BYTE BLOCKS AS OUTPUT. THEY PERFORM A 2-TO-1 EXPANSION
                    * AND A SIMPLE QUADRUPLING OPEATION RESPECTIVELY.



                                      GLB           MAGNIFY
                                      GLB           QUADRUPLE
                    * NAMES OF ENTRY POINTS




1EAB                MAGNIFY                         ; PERFORM A 2-TO-1 EXPANSION ON AN
                                                    ; 8-BYTE BLOCK OF DATA.

                                                    ; SOURCE POINTER IN HL, DESTINATION
                                                    ; POINTER IN DE.

                                                    ; DESTROYS IX,IY,AF,BC,DE,HL

         <000A>     BYTE_COUNT        EQU           BC
         <000E>     SOURCE            EQU           IX
         <000F>     DESTINATION       EQU           IY
                    * STANDARD NAMES FOR REGISTERS IN THIS ROUTINE

                    * BEGIN
1EAB E5                               PUSH          HL              ; SET UP POINTERS
1EAC DDE1                             POP           SOURCE
1EAE D5                               PUSH          DE
1EAF FDE1                             POP           DESTINATION

                    *   BYTE_COUNT := 8
1EB1 010008                           LD            BYTE_COUNT,8

                    *   REPEAT
1EB4                MAG_LOOP

                    *      EXPAND A BYTE FROM SOURCE
1EB4 DD7E00                           LD            A,[SOURCE+0]
1EB7 DD23                             INC           SOURCE
1EB9 57                               LD            D,A
1EBA 1E04                             LD            E,4
1EBC CB17           EXP_1             RL            A               ; DOUBLE BITS IN HIGH
1EBE CB14                             RL            H               ; ORDER NIBBLE
1EC0 CB12                             RL            D
1EC2 CB14                             RL            H
1EC4 1D                               DEC           E
1EC5 20F5                             JR            NZ,EXP_1
1EC7 1E04                             LD            E,4
1EC9 CB17           EXP_2             RL            A               ; DOUBLE BITS IN LOW
1ECB CB15                             RL            L               ; ORDER NIBBLE
1ECD CB12                             RL            D
1ECF CB15                             RL            L
1ED1 1D                               DEC           E
1ED2 20F5                             JR            NZ,EXP_2
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

                        *     WRITE IT TO DESTINATION
1ED4 FD7400                        LD          [DESTINATION+0],H
1ED7 FD7510                        LD          [DESTINATION+16],L
1EDA FD23                          INC         DESTINATION
1EDC FD7400                        LD          [DESTINATION+0],H
1EDF FD7510                        LD          [DESTINATION+16],L
1EE2 FD23                          INC         DESTINATION

                        *     DECREMENT BYTE_COUNT
1EE4 0B                            DEC         BYTE_COUNT

                        *   UNTIL BYTE_COUNT = 0
1EE5 79                            LD          A,C
1EE6 B0                            OR          B
1EE7 20CB                          JR          NZ,MAG_LOOP

                        * END
1EE9 C9                            RET

1EEA                    QUADRUPLE                ; PERFORM A QUADRUPLING ON AN
                                                 ; 8-BYTE BLOCK OF DATA.

                                                 ; SOURCE POINTER IN HL, DESTINATION
                                                 ; POINTER IN DE.

                                                 ; DESTROYS AF,BC,DE,HL,IY

                        * BEGIN

                        *   BYTE_COUNT := 16
1EEA 010010                        LD          BYTE_COUNT,16

                        *   SAVE SOURCE
1EED E5                            PUSH        HL

                        *   REPEAT
1EEE                    QUAD_LOOP

                        *     GET A BYTE FROM SOURCE
1EEE 7E                            LD          A,[HL]
1EEF 23                            INC         HL

                        *     WRITE IT TWICE TO DESTINATION
1EF0 12                            LD          [DE],A
1EF1 13                            INC         DE
1EF2 12                            LD          [DE],A
1EF3 13                            INC         DE

                        *     DECREMENT BYTE_COUNT
1EF4 0B                            DEC         BYTE_COUNT

                        *      IF BYTE_COUNT = 8 THEN RESTORE SOURCE
1EF5 79                            LD          A,C
1EF6 FE08                          CP          8
1EF8 2001                          JR          NZ,SKIPZZ
1EFA E1                            POP         HL
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

  1EFB                   SKIPZZ

                         *    UNTIL BYTE_COUNT = 0
  1EFB 79                              LD          A,C
  1EFC B0                              OR          B
  1EFD 20EF                            JR          NZ,QUAD_LOOP

                         * END
  1EFF C9                              RET
                          PROG
```

```
CATION OBJECT CODE LINE     SOURCE LINE

                   7210
                   7211 * THE ROUTINES IN THIS FILE TAKE A SINGLE 8-BYTE BLOCK AS INPUT
                   7212 * AND OPERATE ON IT PRODUCING A SINGLE 8-BYTE BLOCK AS OUTPUT.
                   7213 * THEY PERFORM MIRRORING AROUND THE VERTICAL AXIS, MIRRORING
                   7214 * AROUND THE HORIZONTAL AXIS, AND 90 DEGREE ROTATION.
                   7215
                   7216              GLB           MIRROR_L_R
                   7217              GLB           ROTATE
                   7218              GLB           MIRROR_U_D
                   7219
                   7220
  1F00             7221 MIRROR_L_R                 ; REFLECTS AN 8X8 PIXEL DATA BLOCK
                   7222                            ; AROUND THE VERTICAL AXIS.
                   7223
                   7224                            ; SOURCE IN HL, DEST IN DE
                   7225                            ; DESTROYS AF,BC,DE,HL
                   7226
  1F00 010008      7227              LD   BC,8          ;SET BLOCK BYTE COUNT
  1F03             7228 MIR_L_R10
  1F03 46          7229              LD   B,[HL]        ;GET SOURCE BYTE
  1F04 3E80        7230              LD   A,80H         ;SET WORK REGISTER MASK
  1F06             7231 MIR_L_R20
  1F06 CB10        7232              RL   B             ;PUT SOURCE IN CARRY
  1F08 1F          7233              RRA                ;PUT CARRY INTO WORK REGISTER
  1F09 30FB        7234              JR   NC,MIR_L_R20  ;CONTINUE UNTIL MASK BIT IS IN CARRY
  1F0B 12          7235              LD   [DE],A        ;WRITE MIRRORED BYTE TO DESTINATION
  1F0C 23          7236              INC  HL            ;ADVANCE POINTERS
  1F0D 13          7237              INC  DE
  1F0E 0D          7238              DEC  C
  1F0F 20F2        7239              JR   NZ,MIR_L_R10  ;CHECK COUNT
  1F11             7240 MIR_L_RX
  1F11 C9          7241              RET
                   7242
                   7243
                   7244
  1F12             7245 ROTATE                     ; ROTATE OBJECT 90 DEGREES
                   7246
                   7247                            ; SOURCE IN HL DESTINATION IN DE.
                   7248                            ; DESTROYS AF,BC,DE,HL
                   7249
  1F12 E5          7250              PUSH HL
  1F13 DDE1        7251              POP  IX
  1F15 EB          7252              EX   DE,HL
  1F16 010008      7253              LD   BC,8
  1F19             7254 TRANSP_10
  1F19 DDCB0016    7255              RL   [IX+0]        ;PUT HI BIT OF FIRST SOURCE BYTE IN CARRY
  1F1D CB1E        7256              RR   [HL]          ;PUT CARRY IN DESTINATION BYTE
  1F1F DDCB0116    7257              RL   [IX+1]
  1F23 CB1E        7258              RR   [HL]
  1F25 DDCB0216    7259              RL   [IX+2]
  1F29 CB1E        7260              RR   [HL]
  1F2B DDCB0316    7261              RL   [IX+3]
  1F2F CB1E        7262              RR   [HL]
  1F31 DDCB0416    7263              RL   [IX+4]
  1F35 CB1E        7264              RR   [HL]
  1F37 DDCB0516    7265              RL   [IX+5]
  1F3B CB1E        7266              RR   [HL]
```

```
LOCATION OBJECT CODE LINE      SOURCE LINE

 1F3D DDCB0616                   RL    [IX+6]
 1F41 CB1E                       RR    [HL]
 1F43 DDCB0716                   RL    [IX+7]
 1F47 CB1E                       RR    [HL]
 1F49 23                         INC   HL
 1F4A 0D                         DEC   C
 1F4B 20CC                       JR    NZ,TRANSP_10
 1F4D                      TRANSP_X
 1F4D C9                         RET



 1F4E                      MIRROR_U_D                        ;REFLECT 8X8 PIXEL BLOCK AROUND THE
                                                             ; HORIZONTAL AXIS

                                                             ; SOURCE IN HL, DESTINATION IN DE
                                                             ; DESTROYS AF,BC,DE,HL

                           * SOURCE   := SOURCE  + 7
 1F4E 010007                        LD              BC,7
 1F51 09                            ADD             HL,BC

                           * BYTE COUNT :=  8
 1F52 03                            INC             BC

                           * REPEAT
          <1F53>           REFLECT_LOOP     EQU             $

                           *    [DESTINATION] := [SOURCE]
 1F53 7E                            LD              A,[HL]
 1F54 12                            LD              [DE],A

                           *     DESTINATION  := SUCC (DESTINATION)
 1F55 13                            INC             DE

                           *     SOURCE  := PRED (SOURCE)
 1F56 2B                            DEC             HL

                           *     BYTE COUNT := PRED BYTE COUNT
 1F57 0B                            DEC             BC

                           * UNTIL BYTE COUNT = 0
 1F58 78                            LD              A,B
 1F59 B1                            OR              C
 1F5A 20F7                          JR              NZ,REFLECT_LOOP
                           * END
 1F5C C9                            RET

                                    ;Modified February 14, 1983.  Filler locations were
                                    ;changed to 0FFH to reflect OS_7PRIME.

 1F5D FFFFFFFF              HEX FF,FF,FF,FF               ;Filler
```

```
XCATION OBJECT CODE LINE     SOURCE LINE

                    ************************** ROM JUMP_TABLE ****************************
                    ;
                    ;JUMP_TABLE  THIS IS THE JUMP TABLE TO BE USED IN ACCESSING CODE
                    ;            RESIDING IN THE O.S. ROM.  THIS TABLE MUST HAVE ITS
                    ;            ORIGIN REDEFINED TO ACCOUNT FOR GROWTH. PILE NEW ROUTINES
                    ;            AT THE BEGINNING OF THE TABLE MAKING SURE TO INCREMENT
                    *            THE NO_OF_ROUTINES VALUE.

                    * NOTE ****

                    *                 **** NO DELETIONS SHOULD BE MADE FROM ****
                    *                 **** THIS TABLE                       ****

          <2000>    ROM_END            EQU           2000H
                    * THIS IS THE END OF OS ROM

          <0035>    NO_OF_ROUTINES  EQU 53
                    * THIS NUMBER KEEPS COUNT OF THE NUMBER OF ROUTINES ACCESSED THROUGH
                    * THE JUMP TABLE.

                    JUMP_TABLE        ORG  ROM_END-(NO_OF_ROUTINES*3)

 1F61 C30300        PLAY_SONGS        JP PLAY_SONGS_
 1F64 C30488        ACTIVATEP         JP ACTIVATEQ
 1F67 C306C7        PUTOBJP           JP PUTOBJQ
 1F6A C31D5A        REFLECT_VERTICAL JP RFLCT_VERT
 1F6D C31D60        REFLECT_HORIZONTAL JP RFLCT_HOR
 1F70 C31D66        ROTATE_90         JP ROT_90
 1F73 C31D6C        ENLARGE           JP ENLRG
 1F76 C3114A        CONTROLLER_SCAN JP CONT_SCAN
 1F79 C31188        DECODER           JP DECODER_
 1F7C C31979        GAME_OPT          JP GAME_OPT_
 1F7F C31927        LOAD_ASCII        JP LOAD_ASCII_
 1F82 C318D4        FILL_VRAM         JP FILL_VRAM_
 1F85 C318E9        MODE_1            JP MODE_1_
 1F88 C3116A        UPDATE_SPINNER   JP UPDATE_SPINNER_
 1F8B C31B0E        INIT_TABLEP       JP INIT_TABLEQ
 1F8E C31B8C        GET_VRAMP         JP GET_VRAMQ
 1F91 C31C10        PUT_VRAMP         JP PUT_VRAMQ
 1F94 C31C5A        INIT_SPR_ORDERP  JP INIT_SPR_ORDERQ
 1F97 C31C76        WR_SPR_NM_TBLP   JP WR_SPR_NM_TBLQ
 1F9A C30F9A        INIT_TIMERP       JP INIT_TIMERQ
 1F9D C30FB8        FREE_SIGNALP      JP FREE_SIGNALQ
 1FA0 C31044        REQUEST_SIGNALP  JP REQUEST_SIGNALQ
 1FA3 C3108F        TEST_SIGNALP      JP TEST_SIGNALQ
 1FA6 C31CBC        WRITE_REGISTERP  JP REG_WRITEQ
 1FA9 C31CED        WRITE_VRAMP       JP VRAM_WRITEQ
 1FAC C31D2A        READ_VRAMP        JP VRAM_READQ
 1FAF C30655        INIT_WRITERP      JP INIT_QUEUEQ
 1FB2 C30203        SOUND_INITP       JP INIT_SOUNDQ
 1FB5 C30251        PLAY_ITP          JP JUKE_BOXQ
 1FB8 C31B1D        INIT_TABLE        JP INIT_TABLE_
 1FBB C31BA3        GET_VRAM          JP GET_VRAM_
 1FBE C31C27        PUT_VRAM          JP PUT_VRAM_
 1FC1 C31C66        INIT_SPR_ORDER   JP INIT_SPR_ORDER_
 1FC4 C31C82        WR_SPR_NM_TBL    JP WR_SPR_NM_TBL_
 1FC7 C30FAA        INIT_TIMER        JP INIT_TIMER_
```

```
LOCATION OBJECT CODE LINE     SOURCE LINE

    1FCA C30FC4      7381 FREE_SIGNAL     JP FREE_SIGNAL_
    1FCD C31053      7382 REQUEST_SIGNAL  JP REQUEST_SIGNAL_
    1FD0 C310CB      7383 TEST_SIGNAL     JP TEST_SIGNAL_
    1FD3 C30F37      7384 TIME_MGR        JP TIME_MGR_
    1FD6 C3023B      7385 TURN_OFF_SOUND  JP ALL_OFF_
    1FD9 C31CCA      7386 WRITE_REGISTER  JP REG_WRITE_
    1FDC C31D57      7387 READ_REGISTER   JP REG_READ
    1FDF C31D01      7388 WRITE_VRAM      JP VRAM_WRITE
    1FE2 C31D3E      7389 READ_VRAM       JP VRAM_READ
    1FE5 C30664      7390 INIT_WRITER     JP INIT_QUEUE
    1FE8 C30679      7391 WRITER          JP WRITER_
    1FEB C311C1      7392 POLLER          JP POLLER_
    1FEE C30213      7393 SOUND_INIT      JP INIT_SOUND
    1FF1 C3025E      7394 PLAY_IT         JP JUKE_BOX
    1FF4 C3027F      7395 SOUND_MAN       JP SND_MANAGER
    1FF7 C304A3      7396 ACTIVATE        JP ACTIVATE_
    1FFA C306D8      7397 PUTOBJ          JP PUTOBJ_
    1FFD C3003B      7398 RAND_GEN        JP RAND_GEN_
                     7399

Errors=    0
```

LINE#	SYMBOL	TYPE	REFERENCES
7396	ACTIVATE	A	242
7347	ACTIVATEP	A	248
1610	ACTIVATEQ	P	1609,7347
1627	ACTIVATE_	P	1587,1683,7396
1607	ACTIVATE_P	P	1611
1915	ACT_OSPRT	P	1648
1917	ACT_1SPRT	P	1650
1662	ACT_CMPLX	P	1652
1903	ACT_MOBILE	P	1646
1694	ACT_SEMI	P	1644
884	ADD816	P	703,883
5152	ADDR_ADJ	P	5172
5171	ADD_8	P	5152
6141	ADJUST_COUNT	P	6145
6130	ADJUST_INDEX	P	6134
3723	AD_EXIT	P	3719
3718	AD_LP	P	3721
463	AFTER_RANDOM	P	
85	ALEN	A	
1052	ALL_OFF	P	992,7385
6797	ALL_X	P	
492	AMERICA	P	236
87	APS	A	
88	APSV	A	733
964	AREA_SONG_IS	P	963,1216
4504	ARM	A	4856,5027
5008	ARM_DBNCE	P	4859
5037	ARM_EXIT	P	5022,5028,5033
4522	ARM_MASK	A	4746,5011
4516	ARM_OLD	A	5013,5021,5034
5024	ARM_REG	P	5020
5030	ARM_ST1	P	5016
4517	ARM_STATE	A	5014,5026,5036
499	ASCII_TABLE	P	254
5458	ASCII_TBL	P	499,5080
5382	ASC_TABLE	P	5681
86	ASTEP	A	1373,1445,1464
79	ATN	A	1370
724	ATN_SWEEP	P	721,1229
1038	B1	P	1040
6065	BASE_FACTORS	P	6049
3211	BK_CLR	A	3384,3413,3600,3658,3744,3833
3209	BK_PTN	A	3351,3413,3547,3650,3735,3744,3833
368	BOOT_UP	P	
1980	BUFFER	D	2021,2069,2129
7107	BYTE_COUNT	S	
2689	CALC_OFFSET	P	2480,2644
457	CARRY_READY	P	454
261	CARTRIDGE	A	260,534
6042	CASE_OF_CLR10	P	6039
6035	CASE_OF_COLOR	P	6023
6025	CASE_OF_GEN	P	6021
6032	CASE_OF_GEN10	P	6029
5705	CENTER_PRT	P	5234,5244
75	CH	A	
99	CH0	A	
90	CH0END	A	

LINE#	SYMBOL	TYPE	REFERENCES
94	CHOREP	A	
100	CH1	A	
91	CH1END	A	
95	CH1REP	A	
101	CH2	A	
92	CH2END	A	
96	CH2REP	A	
102	CH3	A	
93	CH3END	A	
97	CH3REP	A	
4784	CHK_PLYR_1	P	4766,4780
4777	CHK_SEG_01	P	4772
4801	CHK_SEG_11	P	4796
4579	CINIT1	P	4587
1677	CMPLX4	P	1690
1691	CMPLX9	P	1676
2823	COLOR	A	
6337	COLORTABLE	D	5902,6330
2830	COLOR_AND_TAG	A	2936,3024,3064,3141
6670	COLOR_TABLE	A	6970,7059,7074
6997	COLOR_TEST	P	6817,6855,6903,6953
3201	COLR	A	3472,3602,3835
3559	COMBINE_LOOP	P	3593
3797	COM_PAT_COL	P	3576
3029	CONTINUE	P	2900,2988
6755	CONTINUE_GRAPHI	P	6690,6709,6730
4534	CONTROLLER_0	A	4611,4712
4535	CONTROLLER_1	A	
4570	CONTROLLER_INIT	P	555,4569
285	CONTROLLER_MAP	A	284,4573,4762
7353	CONTROLLER_SCAN	A	212
4609	CONT_READ	P	4720,4735
4615	CONT_READ1	P	4612
4617	CONT_READX	P	4614
4626	CONT_SCAN	P	4625,4760,7353
80	CTRL	A	1307
4531	CTRL_0_PORT	A	4613,4627,4635,4654
4530	CTRL_1_PORT	A	4616,4630,4638,4668
6432	CTRL_PORT	A	6433,6482,6487,6540,6544,6590,6594,6624
3594	C_LP_EXIT	P	3591
6431	DATA_PORT	A	6433,6550,6599
5050	DBNCE_BUFF	D	4576,4761
4092	DCR_L_MODE_TBL	P	
4104	DCR_L_RPT_TBL	P	4091
4129	DCR_S_MODE_TBL	P	4089
4086	DCR_TIMER	P	4072
833	DECLSN	P	689,735,741,832
850	DECMSN	P	849
7354	DECODER	A	213
4748	DECODERX	P	4728
4701	DECODER	P	4700,7354
4820	DECODE_0	P	4776,4800
4842	DECODE_0X	P	4835
4854	DECODE_1	P	4783,4807
4866	DECODE_1X	P	4863
4827	DEC_FIRE	P	4823
4861	DEC_KBD	P	4857

INE#	SYMBOL	TYPE	REFERENCES
4549	DEC_KBD_TBL	P	4739,4899
4715	DEC_PLYR	P	4713
4733	DEC_SEG1	P	4704
4833	DEC_SPNR	P	4829
59	DEDAREA	A	60,61,62,63,64,65
2181	DEFER	A	2184
630	DEFER_WRITES	D	238,559,2086,2091,2153,2183
4599	DELAY	P	4542
5723	DELAY_10	P	5258
7109	DESTINATION	S	
1535	DE_TO_DEST	P	1343,1441,1461,1487
5126	DISPLAY_LOGO	P	566,5082
5054	DIVIDE	P	6058
3304	DLP1	P	3324
3355	DLP2	P	3425
3479	DLP4	P	3542
3613	DLP5	P	3619
3638	DLP6	P	3642
4048	DONE	A	4473,4486
5167	DONE_LOGO	P	5147
1203	DONE_SNDMAN	P	1178
3120	DONT_PUT	P	2876,2879,2889,2892,2964,2967,2977,2980
2188	DO_PUTOBJ	P	2113,2185
1063	DUMAREA	P	992,1044,1168
1876	DUPLI	P	1878
3681	DVEX	P	3677
3676	DVLP	P	3680
1399	EFFECT	P	1342
1233	EFXOVER	P	1213
3705	ELSE04	P	3646
3223	ELSE1	P	3219
3530	ELSE10	P	3516
3608	ELSE13	P	3605
3810	ELSE18	P	3800
3282	ELSE2	P	3278
3841	ELSE23	P	3838
3339	ELSE5	P	3336
3400	ELSE6	P	3378
3463	ELSE8	P	3460
3501	ELSE9	P	3483
6227	ELSEZZ	P	6198,6201
2277	ELSE_1	P	2268
2702	ELSE_11	P	2694
2726	ELSE_12	P	2718
2536	ELSE_8	P	2503
2559	ELSE_9	P	2543,2544
3757	END04	P	3704
3225	END1	P	3221
3537	END10	P	3529
3587	END11	P	3584
3620	END12	P	3598
3610	END13	P	3607
3628	END14	P	3626
3750	END15	P	3728
3782	END16	P	3766
3781	END17	P	3773
3828	END18	P	3809

LINE#	SYMBOL	TYPE	REFERENCES
3815	END19	P	3813
3284	END2	P	3280
3821	END20	P	3819
3827	END21	P	3825
3872	END22	P	3831
3843	END23	P	3840
3852	END24	P	3847
3861	END25	P	3856
3870	END26	P	3865
3321	END3	P	3308
3313	END4	P	3311
3341	END5	P	3338
3420	END6	P	3399
3419	END7	P	3406
3466	END8	P	3462
3538	END9	P	3500
1389	ENDNOREP	P	1342,1382
1384	ENDREP	P	1342
73	ENDSDATA	A	1143,1177
6147	END_ADJ_COUNT	P	6140
6135	END_ADJ_INDEX	P	
376	END_BOOTUP	P	
6232	END_IFZZ	P	6225
2320	END_IF_1	P	2276
2610	END_IF_10	P	2584,2589
2707	END_IF_11	P	2698
2731	END_IF_12	P	2722
6828	END_IF_1_GRAPHI	P	6819
2344	END_IF_2	P	2326
6875	END_IF_2_GRAPHI	P	6857
2363	END_IF_3	P	2358
6914	END_IF_3_GRAPHI	P	6905
2419	END_IF_4	P	2395,2396
6982	END_IF_4_GRAPHI	P	6955
2572	END_IF_8	P	2532
2568	END_IF_9	P	2555
6611	END_INPUT	P	6607
6562	END_OUTPUT	P	6558
7352	ENLARGE	A	246
6746	ENLRG	P	6745,7352
6925	ENLRG_	P	6751
4051	EOT	A	4073,4215,4246,4407,4425,4464
2271	EQUAL_TO	P	2267
4305	EXIT	P	4264
7030	EXIT_FALSE	P	7018,7023
3155	EXIT_PUT_SPR	P	3118
7129	EXP_1	P	7134
7136	EXP_2	P	7141
526	FALSE	A	558,2090,7030
5616	FILL	P	5621
7357	FILL_VRAM	A	252,5780,5905
5610	FILL_VRAM_	P	5084,5129,7357
4501	FIRE	A	4828,4945
4926	FIRE_DBNCE	P	4831
4955	FIRE_EXIT	P	4940,4946,4951
4521	FIRE_MASK	A	4726,4929
4510	FIRE_OLD	A	4931,4939,4952

NE#	SYMBOL	TYPE	REFERENCES
.942	FIRE_REG	P	4938
.948	FIRE_ST1	P	4934
511	FIRE_STATE	A	4932,4944,4954
'811	FIRST_GEN_NAME	A	3104
;202	FLAGS	A	3225,3375,3596,3603,3644,3798,3829,3836
82	FPS	A	
83	FPSV	A	686
'817	FRAME	A	2926,3014,3055,3092
'814	FRAME_TABLE_PTR	A	2915,3003,3044,3081
050	FREE	A	4071,4222,4224,4248,4344,4432,4471,4483
214	FREE1	P	4219
237	FREE_COUNTER_	P	
311	FREE_EXIT	P	4216
221	FREE_MATCH	P	4212
309	FREE_SET	P	4223,4226,4228
381	FREE_SIGNAL	A	224
366	FREE_SIGNALP	A	229
200	FREE_SIGNALQ	P	4041,7366
206	FREE_SIGNAL_	P	4040,7381
197	FREE_SIG_PAR	P	4201
78	FREQ	A	807,811,814
671	FREQ_SWEEP	P	668,1230
203	FRM	A	3230,3441,3624
84	FSTEP	A	673,702,1372,1444
204	F_GEN	A	3269,3301,3623
333	GAME_NAME	A	332,5230,5232,5237,5247
355	GAME_OPT	A	250
777	GAME_OPT_	P	5761,7355
643	GET_BKGRND	P	2377,2642,3262
055	GET_COLOR	P	6822,6860,6908,6958
278	GET_NEXT	P	4249,4253,4258,4263,4277
294	GET_OLD	P	2291
376	GET_VRAM	A	189,3132,3380,3398,3411,3418
361	GET_VRAMP	A	194
)77	GET_VRAMQ	P	5980,7361
)87	GET_VRAM_	P	2659,5979,6776,7066,7376
)74	GET_VRAM_P	P	6078
305	GRAPHICS	A	2913,2914,3001,3002,3042,3043,3079,3080,3102,3103
?72	HEAD_ADDRESS	D	1997,2022,2032,2070
;85	IF11	P	3582
71	INACTIVE	A	1063,1180,1217,1351,1389,1521
321	INIT_80	P	1822
)58	INIT_QUEUE	P	2057,7390
)50	INIT_QUEUEQ	P	2049,7372
)45	INIT_QUEUE_P	P	2051
)26	INIT_SOUND	P	992,7393
)18	INIT_SOUNDQ	P	1011,7373
;85	INIT_SOUND_DATA	D	1020,1022,1024
)12	INIT_SOUND_PAR	P	1019
'55	INIT_SPR10	P	6260
;78	INIT_SPR_ORDER	A	191
;63	INIT_SPR_ORDERP	A	196
'42	INIT_SPR_ORDERQ	P	5980,7363
'48	INIT_SPR_ORDER_	P	5979,7378
'40	INIT_SPR_P	P	6243
;75	INIT_TABLE	A	188,5649,5654,5659,5664,5669
)46	INIT_TABLE80	P	6017,6024

LINE#	SYMBOL	TYPE	REFERENCES
6061	INIT_TABLE90	P	6031,6034,6041,6044
7360	INIT_TABLEP	A	193
5990	INIT_TABLEQ	P	5980,7360
5997	INIT_TABLE_	P	5979,7375
5987	INIT_TABLE_P	P	5991
7380	INIT_TIMER	A	223
7365	INIT_TIMERP	A	228
4174	INIT_TIMERQ	P	4039,7365
4180	INIT_TIMER_	P	4038,7380
4430	INIT_TIMER__EXI	P	4370,4396,4404
587	INIT_TIME_DATA	D	4176
4171	INIT_TIME_PAR	P	4175
7390	INIT_WRITER	A	203
7372	INIT_WRITERP	A	208
1807	INIT_XP_OS	P	1696,1905
6601	INPUT_LOOP	P	6605,6608
423	IRQ_INTERRUPT	P	
325	IRQ_INT_VECT	A	324,424
4502	JOY	A	4822,4986
4967	JOY_DBNCE	P	4825
4996	JOY_EXIT	P	4981,4987,4992
4523	JOY_MASK	A	4722,4970
4512	JOY_OLD	A	4972,4980,4993
4983	JOY_REG	P	4979
4989	JOY_ST1	P	4975
4513	JOY_STATE	A	4973,4985,4995
1102	JUKE_BOX	P	1075,1385,7394
1095	JUKE_BOXQ	P	1094,7374
1083	JUKE_BOX_PAR	P	1096
7344	JUMP_TABLE	A	
4505	KBD	A	4862,4903
4878	KBD_DBNCE	P	4865
4913	KBD_EXIT	P	4893,4904,4909
4520	KBD_MASK	A	4738,4882
4529	KBD_NULL	A	4550,4557,4561,4565
4518	KBD_OLD	A	4884,4892,4910
4895	KBD_REG	P	4891
4906	KBD_ST1	P	4887
4519	KBD_STATE	A	4885,4897,4912
1143	L1	P	1152
1229	L10	P	1222
1244	L12	P	1232,1240
1377	L13	P	1361
1394	L14	P	1379
1424	L15	P	1397
1448	L16	P	1429
1467	L17	P	1450
1482	L18	P	1479
1534	L19	P	
1176	L2	P	1202
684	L20	P	676
1528	L20_LOAD_HEX	P	1526
708	L21	P	691
766	L22	P	737,761
763	L23	P	743
793	L24	P	787
1306	L5	P	1302

LINE#	SYMBOL	TYPE	REFERENCES
4508	SEG_0	A	4771,4795
4509	SEG_1	A	4779,4803
1750	SEMI_BOT	P	1748
1755	SEMI_EXIT	P	1711,1753
1759	SEMI_GRI	P	1715
1745	SEMI_MID	P	1743
452	SET	P	446
6102	SET_COUNT	P	6097,6229
6124	SET_COUNT10	P	6118
6150	SET_COUNT20	P	6122,6129
6160	SET_COUNTX	P	
4145	SET_DONE_BIT	P	4103,4128,4136
2035	SET_UP_ENDIF	P	2024
1991	SET_UP_WRITE	P	2186
2824	SHAPE	A	
3575	SHFEX	P	3571
3570	SHFLP	P	3574
6164	SHIFT_CT	P	6125
4475	SIGNAL_FALSE	P	4465,4472
4470	SIGNAL_MATCH	P	4461
594	SIGNAL_NUM	D	4202,4204
4480	SIGNAL_TRUE	P	4474
4484	SIGNAL_TRUE1	P	4482
7199	SKIPZZ	P	7197
2317	SKIP_OLD	P	2293
1825	SM_BY_OLD	P	1814,1820
1138	SND_MANAGER	P	1135,7395
76	SONGNO	A	
7393	SOUND_INIT	A	215
7373	SOUND_INITP	A	220
7395	SOUND_MAN	A	218
69	SOUND_PORT	A	795,810,821,1055,1057,1059,1061,1303,1324
7108	SOURCE	S	
5390	SPACE	P	5688
4503	SPIN	A	4834,4838,4839
4514	SPIN_OLD	A	
4515	SPIN_STATE	A	
5051	SPIN_SW0_CT	D	4589,4655,4710,4775
5052	SPIN_SW1_CT	D	4590,4799
4524	SPNR_MASK	A	
5334	SPRITEGENTBL	D	6327
5333	SPRITENAMETBL	D	6326
2807	SPRITE_INDEX	A	3112,3129,3147
275	SPRITE_ORDER	A	274,6254,6283
2798	SPRITE_PTR	S	
997	SR1ATN	A	1054,1278,1279
996	SR1FRQ	A	1280
999	SR2ATN	A	1056,1284,1285
998	SR2FRQ	A	1286
1001	SR3ATN	A	1058,1290,1291
1000	SR3FRQ	A	1292
1004	SRNATN	A	1060,1296,1297
1003	SRNCTL	A	1314
578	STACK	D	233,371
290	START_GAME	A	289,541,5266
2806	STATUS	A	2865,2866,2924,2925,2953,2954,3012,3013,3034,3035,3053,3054,3070,3071,3090,3091
532	STRB_RST_PORT	A	4571,4641,4737

LINE#	SYMBOL	TYPE	REFERENCES
4533	STRB_SET_PORT	A	4633,4734
5053	STROBE_FLG	D	
4536	STROBE_RESET	A	
4537	STROBE_SET	A	4703
4266	SUBTRACT_4	P	4259
1831	SUP_GEN_CLR	P	1744,1749,1754
1894	SUP_UPDATE	P	1746,1751
2347	SV1	P	
2393	SV2	P	
573	SYSTEM_RAM_AREA	D	578
2260	S_OLD_SCRN	P	2255
1973	TAIL_ADDRESS	D	2071,2101,2130,2142
3981	TBLO	P	3978
588	TEMP1	D	4178
591	TEMP2	D	4179
4463	TEST1	P	4468
4489	TEST_EXIT	P	4478
7383	TEST_SIGNAL	A	226
7368	TEST_SIGNALP	A	231
4449	TEST_SIGNALQ	P	4045,7368
4455	TEST_SIGNAL_	P	4044,7383
601	TEST_SIG_NUM	D	4451,4453
4446	TEST_SIG_PARAM	P	4450
2802	THIS_SPRITE	S	
4343	TIMER1	P	4413,4428
4399	TIMER2	P	
5724	TIMER_1	P	5289,5732
5725	TIMER_2	P	5728
4147	TIMER_EXIT	P	4132,4157
598	TIMER_LENGTH	D	4334
4495	TIMER_TABLE_BAS	D	4030,4069,4181,4208,4243,4340,4457
7384	TIME_MGR	A	227
4067	TIME_MGRQ	P	4047
4068	TIME_MGR_	P	4046,7384
1319	TONE_OUT	P	1255,1282,1288,1294
5322	TRADEMARK	P	5190
7254	TRANSP_10	P	7273
7274	TRANSP_X	P	
527	TRUE	A	6818,6856,6904,6954,7026
7385	TURN_OFF_SOUND	A	216,548
1432	TYPE0	P	1342
1453	TYPE1	P	1342
1472	TYPE2	P	1342
1501	TYPE3	P	1342,1469
1625	T222	P	1623
783	UPATNCTRL	P	781,1306,1315,1327
4665	UPDATE_R0	P	4660
4679	UPDATE_R1	P	4674
4668	UPDATE_S1	P	4657,4663
7359	UPDATE_SPINNER	A	211
4653	UPDATE_SPINNER_	P	4652,7359
4681	UPDATE_SPINX	P	4670,4677
806	UPFREQ	P	805,1328
1166	UP_CH_DATA_PTRS	P	1121,1164,1241
608	VDP_MODE_WORD	D	235,1713,3216,6015,6120,6494,6502,7021
623	VDP_STATUS_BYTE	D	234
6332	VRAM_ADDR_TABLE	D	6009,6114,6286,6325

LINE#	SYMBOL	TYPE	REFERENCES
6586	VRAM_READ	P	2287,2316,6098,6585,7389
6577	VRAM_READQ	P	6576,7371
6572	VRAM_READ_P	P	6578
6529	VRAM_WRITE	P	1824,2417,6230,6528,7388
6520	VRAM_WRITEQ	P	6519,7370
6515	VRAM_WRITE_P	P	6521
1270	WRITE	A	
280	WORK_BUFFER	A	279,1873,2280,2361,2401,2859,2947,3215,3256,3268,3271,3291,3295,3300,3329,3347,3373,3382,3440,3451,3467 3471,3474,3545,3546,3599,3633,3649,3656,3657,3660,3703,3723,3737,3761,3774,3784,3785,6775,6805,6843,6863 6891,6929,6946,6961,6975,7047,7064,7079
7391	WRITER	A	204
2083	WRITER	P	2082,7391
5954	WRITE_CHAR	P	5844,5848,5852,5856,5948,5951
5959	WRITE_L11	P	5868,5871,5874,5877
5965	WRITE_L12	P	5889,5892,5895,5898
5941	WRITE_L3	P	5810,5813,5816,5819,5822,5825,5828,5831
5947	WRITE_L4	P	5834,5859,5880
5950	WRITE_L5	P	5837,5862,5883
5953	WRITE_L6	P	5840,5865,5886
5142	WRITE_LOOP	P	5165
5177	WRITE_NAMES	P	5169
7386	WRITE_REGISTER	A	199,5216,5263,5293,5638,5642,5674,5788,5912
7369	WRITE_REGISTERP	A	205
7388	WRITE_VRAM	A	201,3337
7370	WRITE_VRAMP	A	206
2135	WRTR_ELSE	P	2122
2145	WRTR_END_IF	P	2133
2149	WRTR_END_WHILE	P	2098
2094	WRTR_WHILE	P	2147
7379	WR_SPR_NM_TBL	A	192
7364	WR_SPR_NM_TBLP	A	197
6269	WR_SPR_NM_TBLQ	P	5980,7364
6275	WR_SPR_NM_TBL_	P	5979,7379
6267	WR_SPR_P	P	6270
2828	X	A	2910,2998,3039,3137
3200	XDISP	A	3239,3568
3208	XP_BK	A	3243,3260,3330,3681,3768,3788
2490	XP_NEG	P	2487
3206	XP_OS	A	3272,3762,3777
2498	X_IN_BOUNDS	P	
2818	X_LOCATION	A	2867,2955,3036
2827	Y	A	3075
3199	YDISP	A	3248,3549,3577,3579
3207	YP_BK	A	3252,3257,3259,3292,3348,3374,3634,3661,3685,3724,3769,3786,3789
3205	YP_OS	A	3296,3767,3775,3778
2819	Y_LOCATION	A	3072