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:

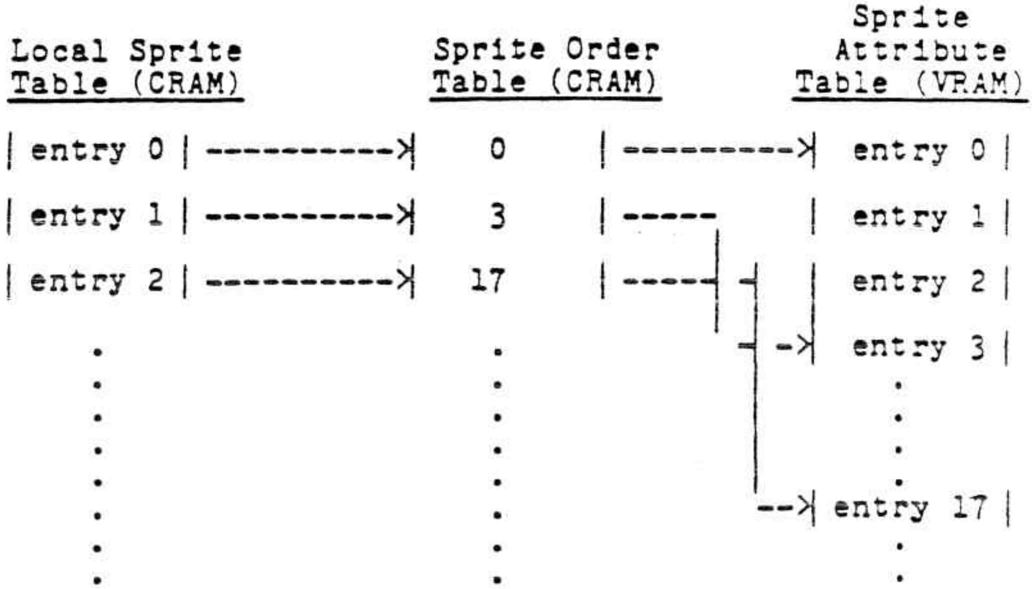

Figure 3-8

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.