APPENDIX B

COLECOVISION

GRAPHICS USERS' MANUAL

11/30/82 V1.0

COLECOVISION
GRAPHICS USERS' MANUAL

Table of contents

Page	Section
1	1.0 Introduction
1	1.1 Summary of Object Types
2	1.2 Data structure Overview
2	1.3 Graphics Routines Overview
3	1.4 Some Key Concepts
	discussion of object types
5	2.0 Semi-Mobile Objects
6	2.1 Detailed Description of Semi-Mobile Object Data Structure
12	3.0 Mobile Objects
13	3.1 Detailed Description of Mobile Object Data Structure
17	4.0 Sprite Objects
18	4.1 Detailed Description of Sprite Object Data Structure
21	5.0 Complex Objects
21	5.1 Detailed Description of Complex Object Data Structure
	discussion of graphics routines
24	6.0 Activate
24	6.1 Description of Activate Routine
28	7.0 Put_Object
28	7.1 Description of Put_Object
28	7.2 Description of Put_Semi
29	7.3 Put_Mobile
30	7.4 Description of Put_Mobile
34	7.5 Put_Sprite
34	7.6 Put_Complex
	data structure summary
36	8.0 Data Structure Summary for Semi_Mobile Objects
39	9.0 Data Structure Summary for Mobile Objects
41	10.0 Data Structure Summary for Sprite Objects
42	11.0 Data Structure Summary for Complex Objects
	figures
44	figure 1
45	figure 2
46	figures 3,4,5

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 givun 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.

2.2 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 backgrounds. (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.

CRAPHICS 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.