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 given 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. ACTIVATL 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.

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.

2.1 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 conceptualy 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.

4 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.

2	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 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 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 initialised 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
        CALL ACTIVATE              ; MOVES GENERATORS TO VRAM
         :

        -or-

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

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,W_BUF+88H   ; 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
```

W_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 initialized 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_n         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) contiaue 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_m    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
```

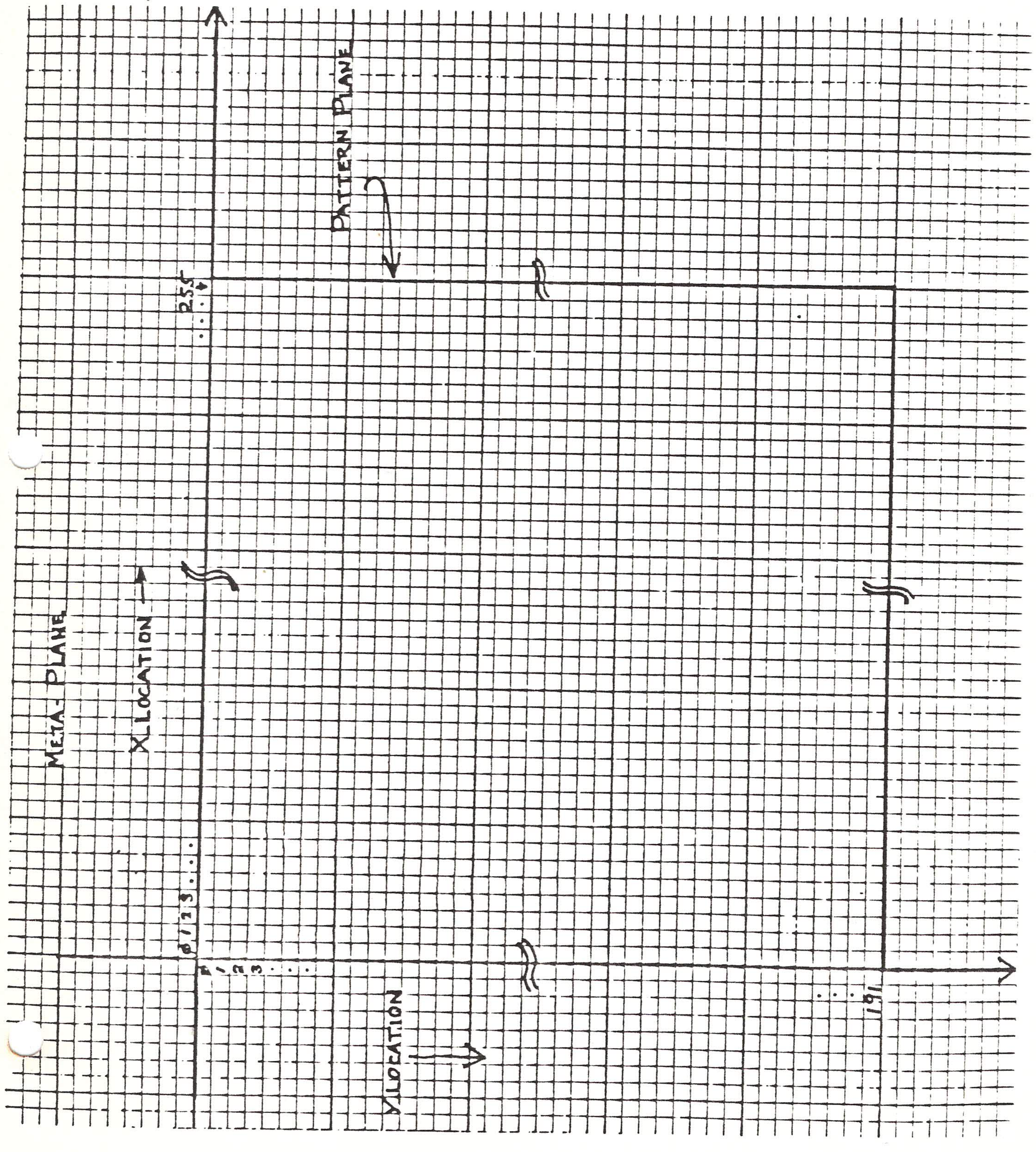
*** RAM DATA AREA ***

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
```

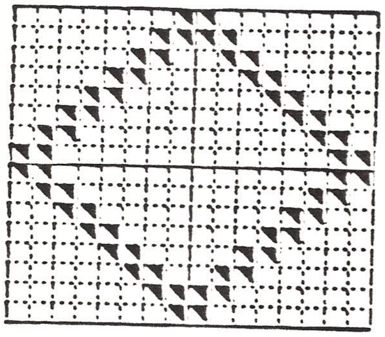
Figure 1

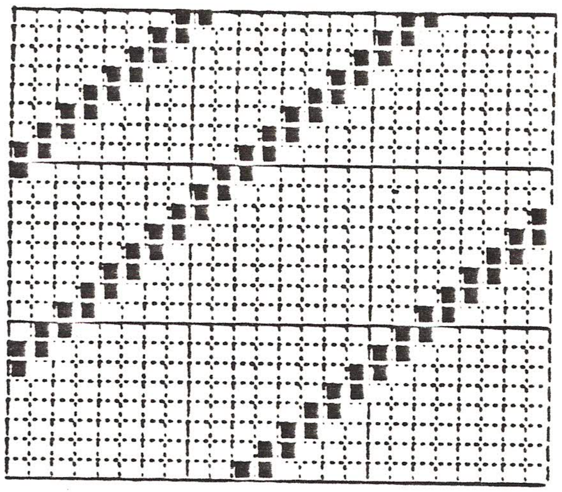
This figure illustrates the relationship between the Meta-Plane and the Pattern Plane. X_LOCATION and Y_LOCATION are sixteen bit signed variables which determin the location of an object's upper-left corner. These variables are part of each object's STATUS area in CPU RAM.



The following figures represent the four methods which PUT_MOBILE uses to superimpose a Mobile object apon a background. A paramenter 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.

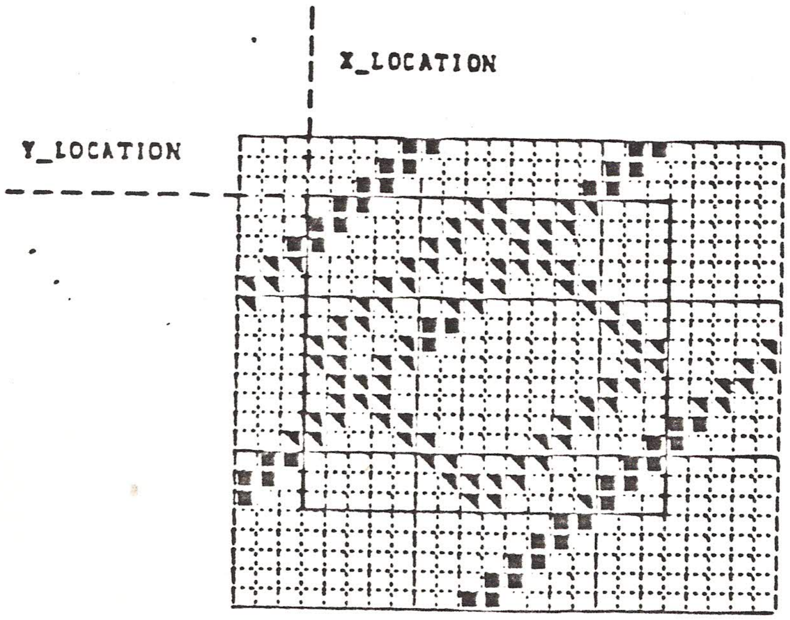


Figure 2: Mobile object superimposed on surround when parameter passed in register B = 0.

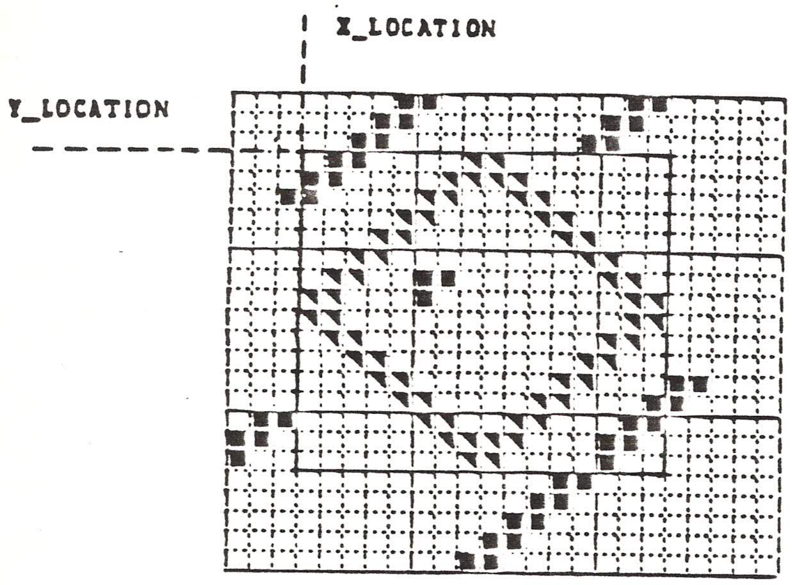


Figure 3: Mobile object superimposed on surround when parameter passed in register B = 1.

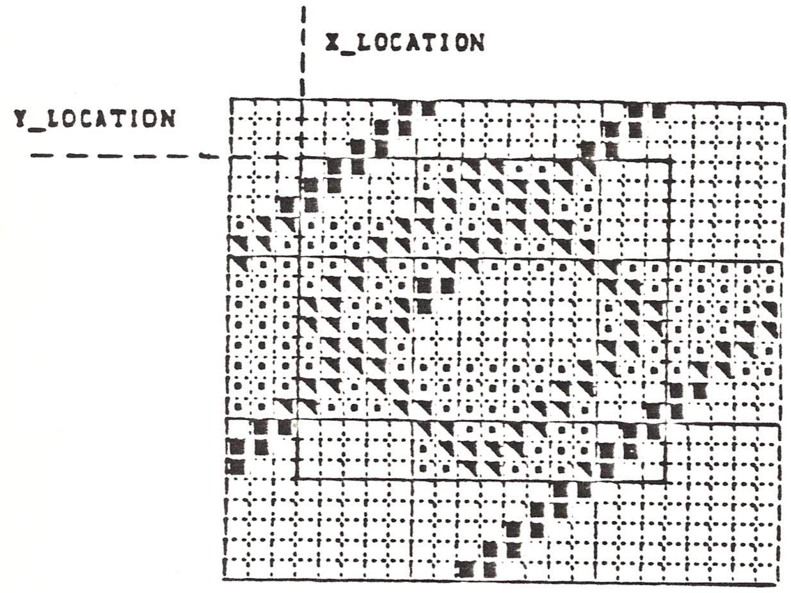


Figure 4: Mobile object superimposed on surround when parameter passed in register B = 2.

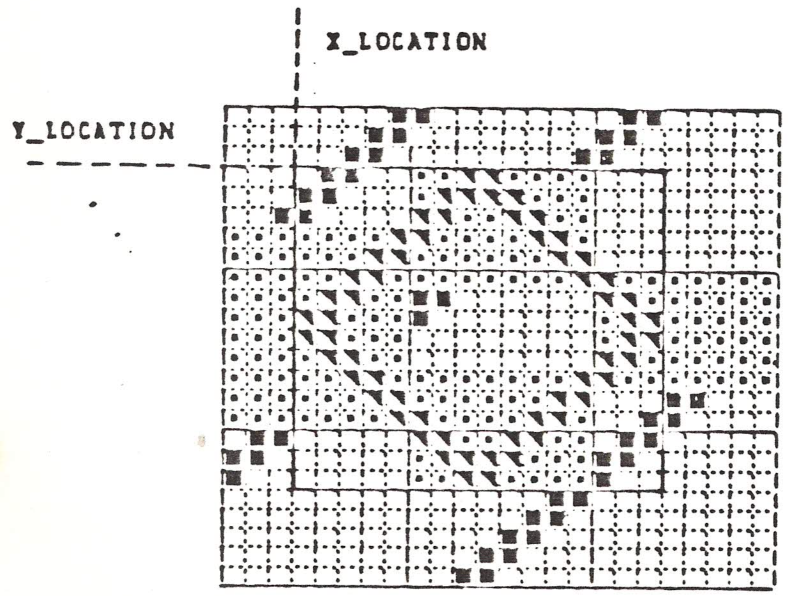


Figure 5: Mobile object superimposed on surround when parameter passed in register B = 3.