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

FRAME_0...FRAME_n - cartridge ROM or CPU RAM

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

```
byte:
0        list of 4 pattern names.  The four names specify which of the
         object's patterns are to be displayed in each of the object's
         four quadrants as follows:
         name               quadrant
          1                 upper left
          2                 lower left
          3                 upper right
          4                 lower right
         The value of the name specifies the object's generators as
         follows:
         0 = first ROMed pattern
         1 = second ROMed pattern
          :
         NUMGEN-1 = last ROMed pattern
         Name values greater than or equal to NUMGEN refer to patterns
         stored in the location starting at NEW_GEN in CPU RAM.  A
         value of NUMGEN refers to the first pattern in that area.  A
         value of NUMGEN + 1 refers to the second pattern etc.
         The MSN of this byte determines the color of the object (i.e.
         the color1 of the object) and the LSN must be 0.
```

*** Generators - cartridge ROM

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

```
byte:

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

*** STATUS - CPU RAM

A Mobile object's status area consists of 4 elements

byte:	
0	FRAME - bits 0-6 indicate which frame is to be displayed. Bit 7 is used by the PUT_MOBILE routine and should not be altered when changing the frame number.
1	X_LOCATION
3	Y_LOCATION - The X and Y_LOCATION variables specify the coordinate position of the upper left corner of the object on a "metaplane" which includes the pattern plane.
5	pointer to NEW_GEN area - This pointer points to the next available space for adding new generators. It will be initialized by ACTIVATE with the 16 bit value NEW_GEN from the parameter segment of the object's GRAPHICS area. Routines which add generators to this area should increment the pointer by 8 each time a new generator is added if subsequent generators are not to overwrite previous ones.

*** OLD_SCREEN - VRAM or CPU RAM

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

byte:	
0	X_PAT_POS - pattern position column in which the upper left corner of the saved "background frame" lies
1	Y_PAT_POS - pattern position row in which the upper left corner lies
2-11	the nine background names

4.0 SPRITE OBJECTS

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

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

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

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

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

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

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

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

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

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

4.1 DETAILED DESCRIPTION OF SPRITE OBJECT DATA STRUCTURE

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

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

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

A detailed description of each structure follows.

*** SPROBJ cartridge ROM

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

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

*** GRAPHICS - cartridge ROM

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

*** Parameters and Pointers

byte:

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

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

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

3	address of GENERATORS - the start of the ROMed pattern generators in the object's graphics data area.
4	NUMGEN - indicates how many pattern generators are defined (stored) in the graphics data area. This is the number of generators which will be moved to the VRAM generator table.
5	address of FRAME_TABLE - This is a pointer to the table containing shape and color information for each frame of the object.

*** FRAME_TABLE - cartridge ROM or CPU RAM

The FRAME_TABLE contains a pair of bytes for each frame. The first byte of each pair determines the color and the second byte points to the generator (or set of four generators in the case of sizel 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 sizel sprites, the value of SHAPE must be a multiple of 4.
2	COLOR for frame 1
3	SHAPE for frame 1
:	: :
2n	COLOR for frame n
2n+1	SHAPE for frame n

*** GENERATORS - cartridge ROM

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

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

*** STATUS - CPU RAM

An object's status area consists of 4 elements:

byte:

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

1 2 bytes for X_LOCATION

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

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

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

5.0 COMPLEX OBJECTS

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

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

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

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

5.1 DETAILED DESCRIPTION OF COMPLEX OBJECT DATA STRUCTURE

Complex objects are defined in cartridge ROM by:
  1) COM_OB - the object's high level definition
  2) GRAPHICS - which contains frame and offset parameters. For each frame of the Complex object, these parameters define the frame numbers for the component objects and the positional relationship between the component objects.

and in CPU ram by:
  1) STATUS - which contains the frame and position variables for the object.

The following is a detailed description of the data structure:

*** COM_OB - cartridge ROM

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

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

*** GRAPHICS - cartridge ROM

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

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

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

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

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

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

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

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

*** STATUS - CPU RAM

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

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

6.0 ACTIVATE

Calling sequence:

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

            -or-

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

Registers used:

Uses all registers

RAM Used (starting at WORK_BUFFER):

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

6.1 DESCRIPTION OF ACTIVATE ROUTINE

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

1) Activating Semi-Mobile objects

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

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

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

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

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

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

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

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

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

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

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

2) Activating Mobile objects

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

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

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

3) Activating Sprite objects

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

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

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

4) Activating Complex objects

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

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

7.0 PUT_OBJECT

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

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

Registers used:

Uses all registers

RAM used (starting at WORK_BUFFER):

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

2. Mobile
   See discussion of PUT_MOBILE below.

3. Sprite
   4 bytes.

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

7.1 DESCRIPTION OF PUT_OBJECT

PUT_OBJECT places a frame of an object on the display. Which frame is displayed and where it is placed on the display are specified in that object's STATUS area. When PUT_OBJECT is called, it will determine what type of object is to be "put" on the display and ther 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