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 E 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 deferred write condition will not be recognised by PUT_MOBILE.

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

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

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

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

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

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

The calling sequence for Mobile-Objects is:

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

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

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

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

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

7.5 PUT_SPRITE

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

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

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

7.6 PUT_COMPLEX

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

PUT_COMPLEX takes the following steps to display Complex objects:

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

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

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

8.0 DATA STRUCTURE SUMMARY FOR SEMI_MOBILE OBJECTS

*** ROM DATA AREAS ***

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

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

Graphics for semi_mobile objects are defined as follows:

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

The data for each frame is organized as follows:

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

The pattern generators are stored as follows:

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

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

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

GRAPHICS MODE II:

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

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

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

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

GRAPHICS MODE I:

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

*** RAM DATA AREAS ***

The status area for semi_mobile objects is as follows:

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

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

```
OLD_SCREEN      DEFS    1       ;X_PAT_POS, column in which the upper left corner
                                ;of saved screen lies.  This byte 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    m       ;Where m = 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.
                                      ;these 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) continue in the numbering sequence. (i.e. if there are 8 generators in ROM, numbers 0-7, then generator number 8 reffers to the first generator in a table starting at (NEW_GEN), etc.)

*** RAM DATA AREAS ***

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

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

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

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

10.0 DATA STRUCTURE SUMMARY FOR SPRITE OBJECTS

*** ROM DATA AREAS ***

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

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

Graphics for a sprite object are defined as follows:

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

The data for each frame is organised as follows:

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

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

The pattern generators are stored as follows:

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

*** RAM DATA AREAS ***

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

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

11.0 DATA STRUCTURE SUMMARY FOR COMPLEX OBJECTS

*** ROM DATA AREAS ***

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

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

Graphics for a Complex object are defined as follows:

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

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

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

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

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

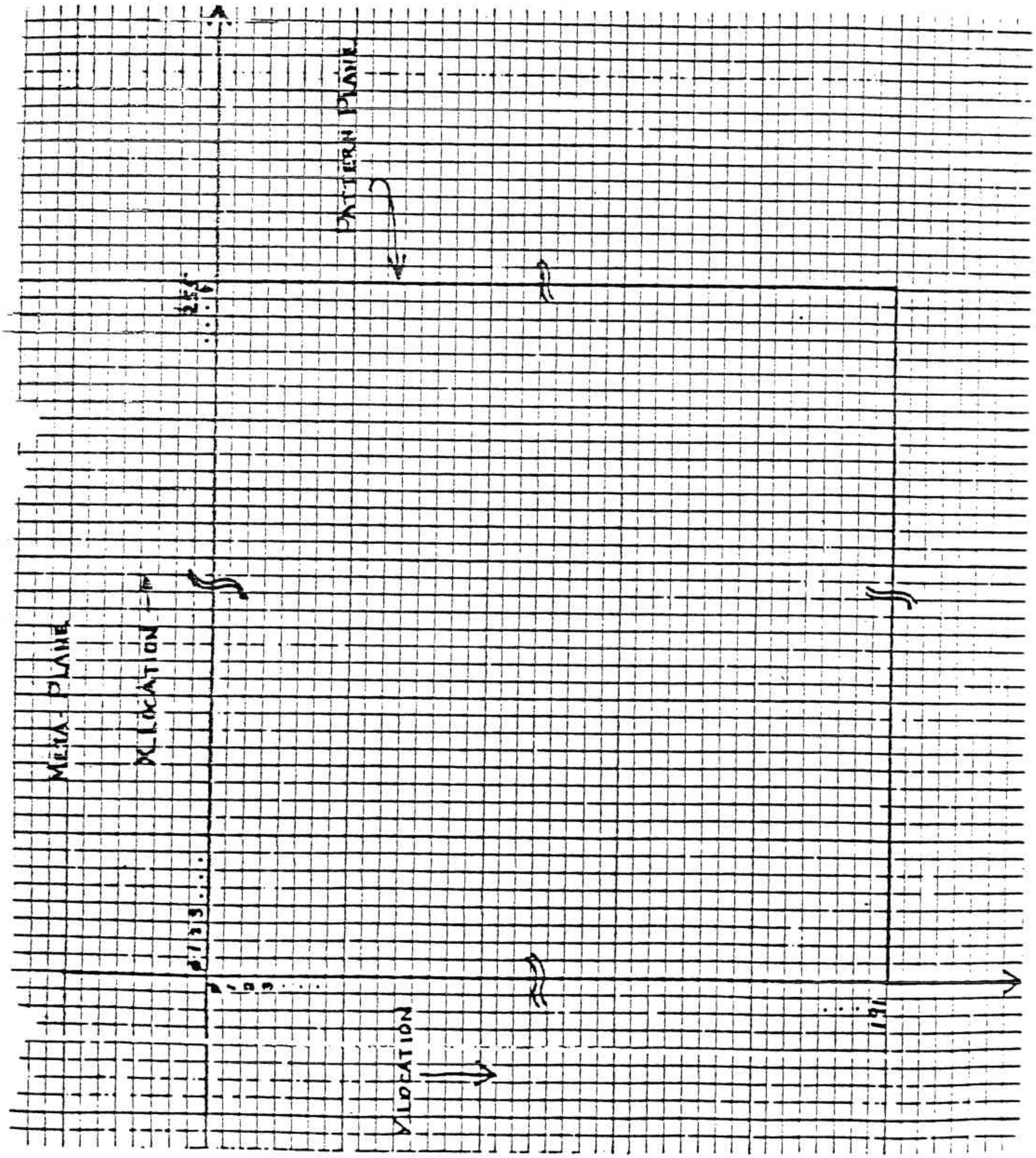
*** RAM DATA AREA ***

Status area for a Complex object:

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

Figure 1

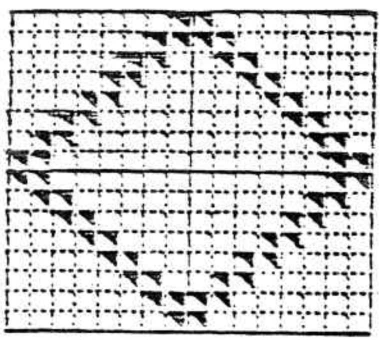
This figure illustrates the relationship between the Meta-Plane and the Pattern Plane. X_LOCATION and Y_LOCATION are sixteen bit signed variables which determin the location of an object's upper-left corner. These variables are part of each object's STATUS area in CPU RAM.



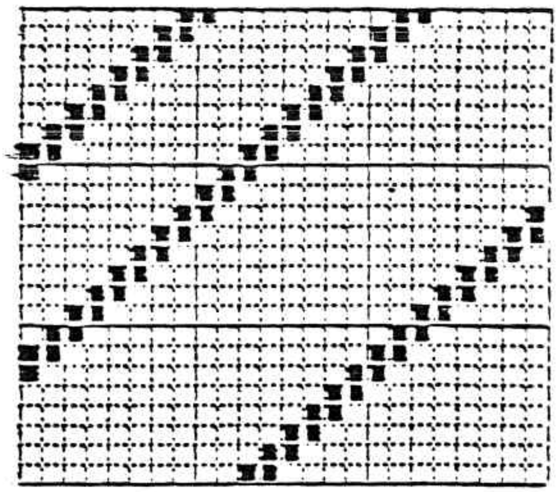
The following figures represent the four methods which PUT_MOBILE uses to superimpose a Mobile object upon a background. A parameter passed to PUT_MOBILE in register B, selects which of the four methods will be used.

Legend:

- represents the color0 of the background
- represents the color1 of the background
- represents the color of the Mobile object
- represents the backdrop color



Mobile object.



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

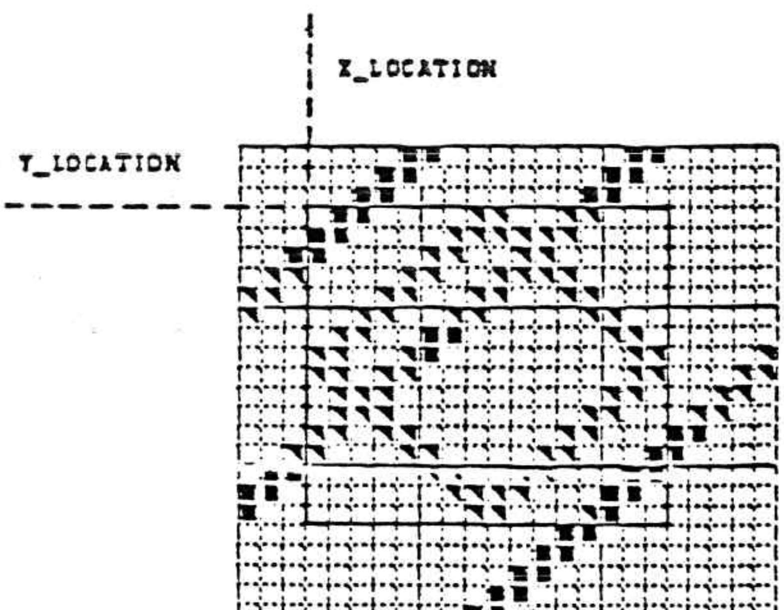


Figure 2: Mobile object superimposed on surround when parameter passed in register B = 0.

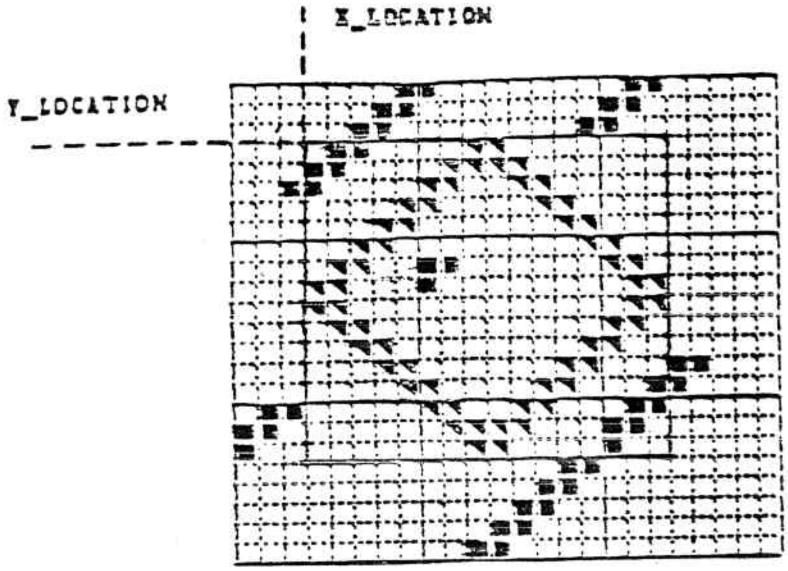


Figure 3: Mobile object superimposed on surround when parameter passed in register B = 1.

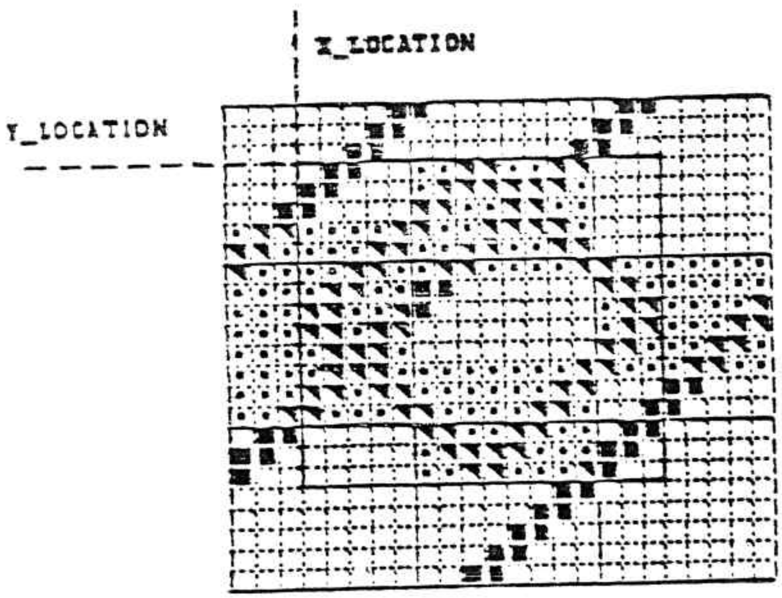


Figure 4: Mobile object superimposed on surround when parameter passed in register B = 2.

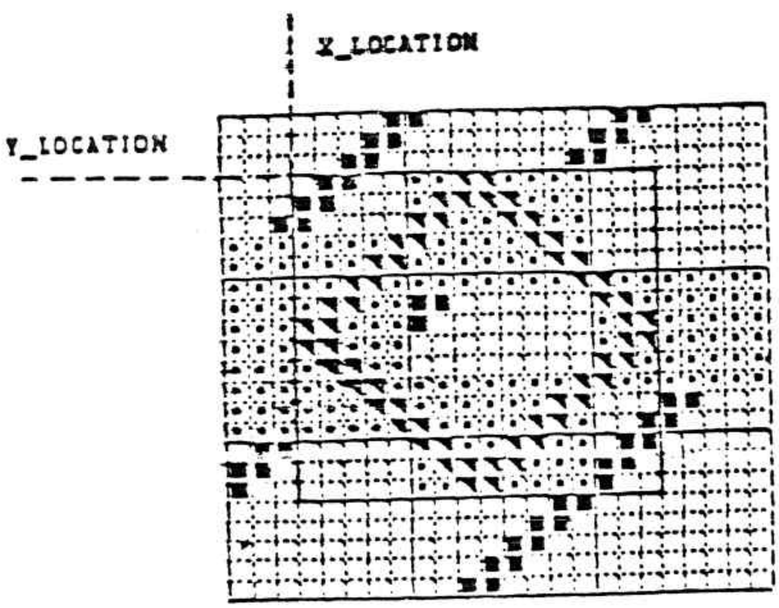


Figure 5: Mobile object superimposed on surround when parameter passed in register B = 3.