COLECOVISION CODING GUIDE

WITH THE ABSOLUTE COLECO BIOS LISTING

PRELIMINARY RELEASE

CONFIDENTIAL DOCUMENT - DO NOT COPY

Document written by Daniel Bienvenu
Last modification date: 03/07/2010

ACKNOWLEDGEMENTS

From ADAM™ Technical Reference Manual

Note: The following text was written in every official Coleco documents.

Coleco makes no representations or warranties whatsoever, including without limitation any implied warranties of merchantability and fitness for a particular purpose, in connection with the materials contained herein, and sush materials are disclosed as is. Coleco shall have no liability for any losses caused to recipients of these materials by reason of any changes or modifications made by Coleco in these materials after their disclousure herein, in addition, Coleco shall have no liability for any consequential, special, indiret or incidental damages or losses whatsoever, including loss of profits, in connection with the use of the materials disclosed herein.

Please note that a lot of time and effort has been put into this documentation. If you intend to use it for commercial purpose, please ask for permission first.

PREFACE

This document is a source of technical information for software designers. This preliminary release includes the most essential information.

Section I, BEFORE PROGRAMMING, shows an overview of all the necessary informations to know before programming a real ColecoVision game.

Section II, OS 7' ROUTINES SPECIFICATIONS, covers all the functions of the ColecoVision BIOS with necessary information to properly use them.

Section III, OS 7' ABSOLUTE LISTING, supplies listing for the ColecoVision BIOS code.

Section IV, APPENDIX, contains suplement of information about the ColecoVision itself.

TABLE OF CONTENTS

BEFORE PROGRAMMING

Source : ColecoVision Programmers' Manual

The following text shows what to know before programming a real ColecoVision game.

Defined Reference Locations

In the OS ROM area, it's IMPORTANT to know that the application programs (games) should only use the OS entry points and global symbols listed in the appendix. Accessing to the OS otherwise is illegal and may cause program malfunction when hardware configuration changes or OS routines relocated due to update. This warning still important today to keep the compatibility of the games with the actual and future ColecoVision clones.

Europe/America Byte

The European TV uses PAL system (625-line format) which requires interrupt at the end of each active-display scan every 1/50 second, as opposed to every 1/60 second for the American model (NTSC, 525-line format). ColecoVision cartridges must be interchangeable between both systems. If a real-time display (such as a clock) must be implemented, the program will have to access the Europe/America byte (0069h) to determine the current line frequency: 60 (3Ch) for America-based units, and 50 (32h) for European-based units.

Graphics Tables

There are two(2) graphics tables in the OS available to the user. The pointers to these tables are defined in the locations ASCII_TABLE (006Ah) and NUMBER_TABLE (006Ch). It's IMPORTANT to get the pointers to the ASCII and number generator tables by using these locations.

The ASCII table contains pattern generators for all 26 upper and pseudo-lower (half-size upper) case letters plus eleven special characters in 5x7 dot matrix form. The number table contains pattern generators for the numbers from 0 to 9 plus seven special characters.

Cartridge ROM

At the beginning of the Cartridge ROM, locations are reserved for testing cartridge presence (8000h; AAh 55h to display OS logo screen, 55h AAh to not display OS logo screen), plus a number of pointers which points to tables, buffers and start of the game (see the Appendix for details). After the pointers, it's the programmable restart and interrupt vectors. And after the vectors, at GAME_NAME location (8024h) it's normally the space (up to 60 bytes) reserved to show game informations on the logo screen.

Of course, we know now that the space for the game informations on the logo screen could be bigger than 60 bytes, however, this size of 60 bytes is really enough.

RAM Areas

Eleven(11) bytes are reserved for OS sound data starting at 7020h; seventy-one(71) bytes at the high end of memory are used by various OS routines. The top of the stack is sitting at address 73B9h which grows in the decrementing direction. Between stack and user buffer there are 942 bytes available for the application program. However, care should be exercised in both size and boundary when using RAM as scratch pad.

Special note: the stack pointer address is set by the OS before running the game, and it's a good practice to never set yourself the stack pointer specially if the game uses the OS functions.

Of course, if the game don't use the OS sound routines, the reserved locations 7020h-702Ah can be used for any purpose.

Dealing with OS Bugs

WRITE_VRAM AND READ_VRAM

It works as advertised for byte counts less than 256 (00h-FFh) and for byte counts that are multiples of 256 (100h, 200h,...). For other values, it substracts 256 (100h) from the actual byte count. Programmers should deal with this problem by always sending numbers of bytes that are less than or even multiples of 256 (100h). They should not deal with it by padding their byte counts as this may lead to cartridges that fail when the bug is fixed. Note : PUT_VRAM and GET_VRAM call these routines, so take care.

SEMI-MOBILE OBJECT IN GRAPHIC MODE 1

In the graphic mode 1, ACTIVATE writes the pattern generators for a semi-mobile object to VRAM properly, but miscalculates the number and placement in VRAM of the corresponding color bytes when operating on generators in the upper half of the stable. Programmers should avoid using ACTIVATE to write pattern generators to VRAM in graphic mode 1, - OR - , first of all, count on having to write the color table seperatly, and second, count on guarding against VRAM corruption by isolating the color table.

PUT_MOBILE PATCH

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. Calling the fixed version inside the cartridge instead of the version in the OS has two side effects: mobile objects may not be components of a complex object, the defered write condition will not be recognized by PUTMOBILE.

The following code is the section of PUT_MOBILE to be part of the cartridge program.
Remark : the red color is used here to show the difference between two(2) versions of PUTMOBILE, one for graphic mode 1 and a second for graphic mode 2.

```
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

;
; PUTMOBILE version GRAPHICS MODE I
;
PUTMOBILE:          LD IY,[WORK_BUFFER]
                    RES 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
                    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 not equal 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
                    SRL A
                    LD E,A                ; DE gets pointer to object's
                    LD D,0                ;    color gens in VRAM
```

```
                    LD HL,WORK_BUFFER+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
                    JP PUT_VRAM

;
; PUTMOBILE version GRAPHICS MODE II
;
PUTMOBILE:          LD IY,[WORK_BUFFER]
                    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           ; Call rest of OS PUT_MOBILE routine

; The calling sequence for mobile objects is :
;
;                   LD IX,HIGH_LEVEL_DEFINITION
;                   LD HL,GRAPHICS
;                   LD B,MODE
;                   CALL PUTMOBILE
```

OS 7' ROUTINES SPECIFICATIONS

Written by Daniel Bienvenu and Steve Bégin.
Verified and completed with the ColecoVision programmers' manual. (in progress)

SOUND ROUTINES

Except TURN_OFF, the sound routines in the jump table require a specific sound data format and/or sound tables in RAM and ROM.

More information about the sound format, song data areas and more at the sections SOUND DATA FORMAT, SOUND TABLES, SONG TABLE IN ROM, OUTPUT TABLE IN RAM in pages 190-192.

A quick reference guide is available in page 22.

FREQ_SWEEP

INPUT:

IX = address of byte 0 of a song data area.

FUNCTION(S):

If frequency not swept, so decrement NLEN (note length) value and RET.
Otherwise,

- Decrement FPSV counter
- If FPSV timed out, then :
  - Reload FPSV counter and decrement NLEN value.
  - Add frequency step FSTEP to frequency value FREQ if note not over.

OUTPUTS:

Z flag is reset if sweep in progress or note not over, Z flag is set if note over.

CALLS:

DECLSN
MSNTOLSN
ADD816

CALLED BY:

PROCESS_DATA_AREA

NOTES:

None

ATN_SWEEP

INPUT:

IX = address of byte 0 of a song data area.

FUNCTION(S):

OUTPUTS:

Z flag is set if sweep is over or note was never sweep, Z flag is reset sweep in progress.

CALLS:

DECLSN
MSNTOLSN

CALLED BY:

PROCESS_DATA_AREA

NOTES:

None

UPATNCTRL

INPUT:

IX = address of byte 0 of a song data area.
C = formatted channel attenuation code (MSN).

FUNCTION(S):

Output attenuation or noise control data to sound port.

OUTPUTS:

None

CALLS:

None

CALLED BY:

PLAY_SONGS
TONE_OUT

NOTES:

None

UPFREQ

INPUT:

IX = address of byte 0 of a song data area.
D = formatted channel frequency code (MSN).

FUNCTION(S):

Output frequency data (into 2 bytes) to sound port.

OUTPUTS:

None

CALLS:

None

CALLED BY:

TONE_OUT

NOTES:

None

PT_IX_TO_SxDATA

INPUT:

B = song number.

FUNCTION(S):

Point IX to byte 0 in a song data area calculated with song number (SONGNO) passed in B.

OUTPUTS:

IX = address of byte 0 song data area used by that song number (SONGNO).

CALLS:

None

CALLED BY:

FREQ_SWEEP

NOTES:

DE = IX
HL is pointing to MSN SxDATA entry in LST_OF_SND_ADDRS.

LEAVE_EFFECT

INPUT:

B = song number.

FUNCTION(S):

Restores the SONGNO to which the effect note belongs to B5-B0 of byte 0 in the effect's data area, and loads byte 1 and 2 with the address of the next note in the song. The address of the 1 byte SONGNO (saved by the effect when it was first called) is passed to DE. The 2 byte address of the next note in the song, also saved by the effect, is passed in HL. IX is assumed to be pointing to byte 0 of the data area to which the song number is to be restored. Bits 7 & 6 of the saved SONGNO byte are not stored into byte 0, and therefore may be used during the course of the effect to store any useful flag information.

OUTPUTS:

None

CALLS:

None

CALLED BY:

Called by a special sound effect routine when it's finished.

NOTES:

None

AREA_SONG_IS

INPUT:

IX = address of byte 0 of a song data area.

FUNCTION(S):

Retrive in A the song # playing in a specific song data area pointed to by IX.

OUTPUTS:

Accumulator =
song # of the song using that area;
FF if inactive;
62 if special effect and HL = address of the special sound effect routine.

CALLS:

None

CALLED BY:

PROCESS_DATA_AREA

NOTES:

None

SOUND_INIT

ADDRESS : 1FEE

INPUT:

HL = LST_OF_SND_ADDRS, address in RAM to song data areas.
B = # of song data areas to initialize.

FUNCTION(S):

Set pointer PTR_TO_LST_OF_SND_ADDRS to LST_OF_SND_ADDRS.
Store inactive code FF at byte 0 of the song data areas.
Store 00 at end of song data areas.
Sets the 4 channel sound pointers to a dummy, inactive data area.
Initialize SAVE_CTRL to inactive code FF.
Turn off sound

OUTPUTS:

None

CALLS:

ALL_OFF (local routine name for TURN_OFF_SOUND)

CALLED BY:

Under user program control. Should be called immediately after power on.

NOTES:

IX and IY are preserved

TURN_OFF_SOUND

ADDRESS : 1FD6

INPUT:

None

FUNCTION(S):

Turn off all 4 sound generators.

OUTPUTS:

None

CALLS:

None

CALLED BY:

POWER_UP

NOTES:

Only the Accumulator is affected

PLAY_IT

ADDRESS : 1FF1

INPUT:

B = song number to play.

FUNCTION(S):

If the song is already playing, do nothing.
Otherwise,

- Load 1st note and set NEXT_NOTE_PTR.
- Update channel data pointers.

OUTPUTS:

None

CALLS:

PT_IX_TO_SxDATA
LOAD_NEXT_NOTE
UP_CH_DATA_PTRS

CALLED BY:

Under user program control

NOTES:

None

SOUND_MAN

ADDRESS : 1FF4

INPUT:

None

FUNCTION(S):

Update all song data areas.

- Update counters : decrement sound duration and sweep timers.
- Apply sound effect : modify swept frequency and attenuation values.
- Call special effects routines where necessary.
- Update the channel data area pointers if necessary.
- Restart the sound if indicated.

OUTPUTS:

None

CALLS:

PT_IX_TO_SxDATA
PROCESS_DATA_AREA

CALLED BY:

Under user program control. Should be called every VDP interrupt, after PLAY_SONGS.

NOTES:

None

UP_CH_DATA_PTRS

INPUT:

None

FUNCTION(S):

Set all 4 channel data pointers to dummy inactive area.
Scan all song data areas to store song data area's byte 0 address to proper channel data pointer.

OUTPUTS:

None

CALLS:

PT_IX_TO_SxDATA

CALLED BY:

JUKE_BOX (local routine name for PLAY_IT)
EFXOVER (sub-routine in PROCESS_DATA_AREA)

NOTES:

IX returned intact

PROCESS_DATA_AREA

INPUT:

IX = address of byte 0 of a song data area.

FUNCTION(S):

If byte 0 is inactive code FF, do nothing.
If byte 0 is special sound effect code 3E, do 1pass thru effect.
Otherwise, process attenuation and frequency sweep data, if any.
If note is over, run sub-routine EFXOVER.

OUTPUTS:

None

CALLS:

AREA_SONG_IS
ATN_SWEEP
FREQ_SWEEP
EFXOVER

CALLED BY:

SND_MANAGER (local routine name for SOUND_MAN)

NOTES:

None

EFXOVER

INPUT:

IX = address of byte 0 of a song data area.

FUNCTION(S):

Load next note and update channel data pointers if needed.

OUTPUTS:

None

CALLS:

LOAD_NEXT_NOTE
UP_CH_DATA_PTRS

CALLED BY:

PROCESS_DATA_AREA

NOTES:

None

PLAY_SONGS

ADDRESS : 1F61

INPUT:

None

FUNCTION(S):

Prepare and pitch the actual song notes to sound chip.

- Current frequency and attenuation data is output to each tone generator, if sound on that channel is active; otherwise that generator is turned off.
- Noise generator is sent current attenuation data and control data, if new.
- Modifes SAVE_CTRL if necessary

OUTPUTS:

None

CALLS:

TONE_OUT
UPATNCTRL
UPFREQ

CALLED BY:

Under user program control. Should be called every VDP interrupt, before SOUND_MAN.

NOTES:

None

TONE_OUT

INPUT:

IX = pointer to byte 0 of a song data area
A = formatted channel mute code
C = formatted channel attenuation code (needed for UPATNCTRL) (MSN)
D = formatted channel frequency code (needed for UPFREQ) (MSN)

FUNCTION(S):

Pitch the current frequency and attenuation to sound chip.

OUTPUTS:

None

CALLS:

UPATNCTRL
UPFREQ

CALLED BY:

PLAY_SONGS

NOTES:

None

LOAD_NEXT_NOTE*

INPUT:

IX = pointer to byte 0 of a song data area

FUNCTION(S):

Update song data area based on next note code.

OUTPUTS:

None

CALLS:

JUKE_BOX (local name for PLAY_IT)
Note: is called only to reload 1st note when reading a REPEAT note code.

CALLED BY:

PLAY_IT
EFXOVER

NOTES:

AF , HL, DE, BC and IY are affected. Additional may be changed by called special effect subroutines.

QUICK REFERENCE: HOW TO USE THE OS 7' SOUND ROUTINES

When needed, usually before playing game, turn off sound with the following routine :

TURN_OFF_SOUND (1FD6) : Turn off all sound channels. No setup requiered.

Setup the sound tables in RAM as soon as possible in your code with the following routine :

SOUND_INIT (1FEE) : Initialize sound tables in CPU RAM.
Input : HL = Address of the song table, B = Number of output tables to be used.
Note: The first entry in the song table (in the game code) contains the address of the 1st song data area in RAM.

During the game, use the following routine to play sounds :

PLAY_IT (1FF1) : Play a specific song or sound effect.
Input : B = Song number to be started

And, to play songs at a regular speed, add the following routines in this order in the user NMI routine :

PLAY_SONGS (1F61) : Pitch data to sound generator chip.
SOUND_MAN (1FF4) : Update sound tables by applying sweep effects, loading next note or set the END flag.

The song table in ROM is composed into many entries of two addresses (a pointer to the song data in ROM and a pointer to the song data area in RAM). Each entry correspond to a song. The 1st entry in the song table have to use obligatory a pointer to the 1st song data area. The memory allocation to a free zone in RAM for the song data areas is the responsibility of the coder. Because the sound chip can't play more than one song on the same sound channel, there is a concept of priority : higher is the number of the song data area, higher is the priority.

Each song in ROM is encoded into a specific sound data format, uses only one channel, and obligatory ends with an END or REPEAT code. To play a music that uses more than one sound channel, the coder have to encode his music into separate song entries, one for each sound channel needed, of course that needs more than one song entry in the song table where each entry uses a different song data area.

Expert tips : To stop a song playing, you can play another song that uses the same song data area, or write the INACTIVE code 0FFh at the 1st byte of its song data area.

OBJECT ROUTINES

The following routines are for objects. An object can be represented as a set of characters and/or sprites.

Note: Almost every Non-Coleco programmers didn't use these routines. Object graphic is a good concept but these routines are complex, slow, need too much RAM space.

ACTIVATE

ADDRESS : 1FF7

INPUT:

HL = pointer to the object data.
Carry flag = is set to move graphic data to VRAM, reset otherwise.

FUNCTION(S):

Initialize the RAM status area for the passed object
If carry flag is set, move object pattern and color generators to the PATTERN and COLOR generator tables in VRAM.

OUTPUTS:

None

CALLS:

PUT_VRAM_ (local name for PUT_VRAM)
VRAM_WRITE (local name for WRITE_VRAM)

CALLED BY:

Under user program control

NOTES:

VDP_MODE_WORD (73C3), WORK_BUFFER (8006) may be needed by called subroutines.
AF , HL, DE, BC and IY are affected. Additional may be changed by called subroutines.

OBJECT HEADER : pointer to OBJ GEN CPU ROM, pointer to OBJ CPU RAM.
The OBJ GEN CPU ROM header starts with less significant nibble (LSN) as OBJ TYPE.
OBJ TYPE : 0=semi mobile, 1=mobile, 2=0sprite, 3=1sprite, other=complex.

BUG Error in subroutine for semi-mobile type objects in graphic mode I.

SET_UP_WRITE

INPUT:

IX = data pointer.
B = parameter.

FUNCTION(S):

Sets up defered VRAM operation.

OUTPUTS:

None

CALLS:

None

CALLED BY:

PUTOBJ

NOTES:

Destroys all.

INIT_WRITER

ADDRESS : 1FE5

INPUT:

A = size of the defered write queue.
HL = location in RAM of the defered write queue.

FUNCTION(S):

Initialize defered write queue in RAM :

- QUEUE_SIZE (73CA) = A
- QUEUE_HEAD (73CB) = 0
- QUEUE_TAIL (73CC) = 0
- HEAD_ADDRESS (73CD-73CE) = HL
- TAIL_ADDRESS (73CF-73D0) = HL

OUTPUTS:

None

CALLS:

None

CALLED BY:

Under user program control

NOTES:

Only the Accumulator is affected

WRITER

ADDRESS : 1FE8

INPUT:

None.

FUNCTION(S):

Temporary reset defered write flag and write data at queue to VRAM.

OUTPUTS:

None

CALLS:

DO_PUTOBJ

CALLED BY:

Under user program control

NOTES:

Destroys all.

PUTOBJ

ADDRESS : 1FFA

DESCRIPTION:

According to flag DEFER_WRITES (73C6), this function updates the object specifications (position x/y, pattern, color) of object IX on screen or puts the object in the queue for updating later.

INPUT:

IX = Object data pointer
B = Object parameter, selector for methods of combining object generators with background generators (for mobile objects only)

FUNCTION(S):

Check if DEFER_WRITES flag (73C6) is true.
If true, set up for defered write by calling SET_UP_WRITE
If not, process object by calling DO_PUTOBJ

OUTPUTS:

None

CALLS:

SET_UP_WRITE
DO_PUTOBJ

CALLED BY:

Under user program control

NOTES:

Assume this routine destroys all registers.

DO_PUTOBJ

INPUT:

IX = Object data pointer.

FUNCTION(S):

Get object graphics address and object type.
Call the appropriate put object routine based on object type.

OUTPUTS:

None

CALLS:

PUTSEMI
PUT0SPRITE
PUT1SPRITE
PUT_MOBILE
PUTCOMPLEX

CALLED BY:

WRITER
PUTOBJ

NOTES:

Assume this routine destroys all registers.

PUTSEMI

DESCRIPTION:

Puts semi-mobile objects on screen. A semi-mobile object is a box of names (characters) that can be positioned anywhere on screen.

FUNCTION(S):

It calls PX_TO_PTRN_POS and CALC_OFFSET to calculate the top-left current screen XY position of the box of chars in RAM.
Checks if chars that will be overwritten must be saved (OLD_SCREEN in Semi Object Table.)
If Yes (< 8000h):

It recalls the names (characters) that were overwritten the last time (if present) from RAM (OLD_SCREEN) and put them back on screen..
It calls GET_BKGRND to save the names (characters) that the new box will overwrite in RAM (again, 3rd word in Semi Object Table.)
It finally writes new box of names (characters) on screen by calling PUT_VRAM.
Notes: OLD_SCREEN can be in RAM (7100h to 7FFFh) or in VRAM (0000h-3FFFh).
Be sure to use the ranges of addresses specified here (possibilty of problems).
If using VRAM for OLD_SCREEN, it uses the WORK_BUFFER (8006) pointer for temporary storage.

Else (>= 8000)

Only write new box of names (characters) on screen by calling PUT_VRAM

PX_TO_PTRN_POS

INPUT:

DE = signed 16bit number.

FUNCTION(S):

Devides DE reg by 8,
If signed result > 127 then E = max signed positive number 127.
If signed result < -128 then E = min negative number -128.

OUTPUTS:

E = DE/8, except if DE/8 < -128 or DE/8 > 127 then E equals respectively -128 or 127.

CALLS:

None.

CALLED BY:

PUTSEMI
PUT_MOBILE

NOTES:

HL is restored.

PUT_FRAME

DESCRIPTION:

Puts a box of names (characters) on screen. It prevents bleeding outside visible screen.

INPUT:

HL = address of list of names (characters that compose the frame)
E = X_PAT_POS
D = Y_PAT_POS
C = X_EXTENT
B = Y_EXTENT

FUNCTION(S):

The names which constitute a frame are moved to the name table in VRAM.
The upper left hand corner of the frame is positioned at X_PAT_POS, Y_PAT_POS.

OUTPUTS:

None.

CALLS:

CALC_OFFSET
PUT_VRAM

CALLED BY:

PUTSEMI
PUT_MOBILE

NOTES:

GET_BKGRND

DESCRIPTION:

Gets a box of names (characters) from screen.

INPUT:

HL = location in CPU RAM to copy names from VRAM
E = X_PAT_POS
D = Y_PAT_POS
C = X_EXTENT
B = Y_EXTENT

FUNCTION(S):

Gets the names from name table which constitute the background in which an object is to be moved at X_PAT_POS, Y_PAT_POS.

OUTPUTS:

None.

CALLS:

CALC_OFFSET
GET_VRAM

CALLED BY:

PUTSEMI
PUT_MOBILE

NOTES:

CALC_OFFSET

INPUT:

D = Y_PAT_POS.
E = X_PAT_POS.

FUNCTION(S):

This routine calculate the proper offset into the name table for the pattern position given by X_PAT_POS, Y_PAT_POS. The formula used is : offset = 32* Y_PAT_POS + X_PAT_POS

OUTPUTS:

DE = offset.

CALLS:

None.

CALLED BY:

PUT_FRAME
GET_BKGRND

NOTES:

DE is affected.

PUT0SPRITE

PUT1SPRITE

PUT_MOBILE

PUTCOMPLEX

TIMER ROUTINES

The users have to reserve two free RAM spaces to use timers. The first RAM space is for the timer table itself, the second one is for extra data block needed for repeating long timers. The following data structures are how the timers looks like in RAM.

General timer data structure (3 bytes) :

TIMER

DONE	REPEAT	FREE	EOT	LONG	-	-	-
?							
?							

Specific timer data structures :

ONE-TIME SHORT TIMER

DONE	0	0	0	0	-	-	-
Unsigned counter value							
-							

REPEATING SHORT TIMER

DONE	1	0	0	0	-	-	-
Unsigned current counter value							
Unsigned original counter value							

ONE-TIME LONG TIMER

DONE	0	0	0	1	-	-	-
Unsigned counter value - low-part LSB							
Unsigned counter value - high-part MSB							

REPEATING LONG TIMER

DONE	1	0	0	1	-	-	-
Pointer to a data block for extra timer information							

DATA BLOCK

Unsigned current counter value - low-part LSB
Unsigned current counter value - high-part MSB
Unsigned original counter value - low-part LSB
Unsigned original counter value - high-part MSB

Note: The following routines have been done by Ken Lagace and Rob Jepson in March '82.

TIME_MGR

ADDRESS : 1FD3

INPUT:

None.

FUNCTION(S):

Get timer table address from TIMER_TABLE_BASE (73D3-73D4).
Update all timers in timer table.

OUTPUTS:

None.

CALLS:

None.

CALLED BY:

Under user program control

NOTES:

DE and HL are affected.

INIT_TIMER

ADDRESS : 1FC7

INPUT:

HL = base address in CPU RAM for timer table.
DE = base address in CPU RAM for data block.

FUNCTION(S):

Store given base address for timer table in TIMER_TABLE_BASE (73D3-73D4) and for data block in NEXT_TIMER_DATA_BYTE (73D5-73D6).

OUTPUTS:

None.

CALLS:

None.

CALLED BY:

Under user program control

NOTES:

DE and HL are switched.

FREE_SIGNAL

ADDRESS : 1FCA

INPUT:

A = signal (timer) number to be freed. 0 = 1st signal, 1 = 2nd signal, etc.

FUNCTION(S):

Finds signal (timer) A, stops it by setting bit 5 (FREE) and release its data block if exists.

OUTPUTS:

None.

CALLS:

None.

CALLED BY:

Under user program control

NOTES:

REQUEST_SIGNAL

ADDRESS : 1FCD

INPUT:

HL = length of timer
A = repeating timer if 0, non repeating type if not.

FUNCTION(S):

Search for a free signal (timer) and initialize it with HL and A.
Return signal (timer) number used in A.

OUTPUTS:

A = signal (timer) number.

CALLS:

None.

CALLED BY:

Under user program control

NOTES:

TEST_SIGNAL

ADDRESS : 1FD0

INPUT:

A = signal (timer) number to be tested.

FUNCTION(S):

Check if the signal (timer) number exists.
If so, return A = true if bit 7 (DONE) is set and free up the signal (timer) if non repeating counter.
Otherwise, return A = false

OUTPUTS:

A = true if signal bit 7 (DONE) is set, false otherwise.

CALLS:

None.

CALLED BY:

Under user program control

NOTES:

Destroys BC and HL.

CONTROLLER ROUTINES

CONTROLLER_INIT

INPUT:

None

FUNCTION(S):

Initialize controller to strobe reset.
Clear controller memory and debounce status buffer.
Clear remaining variables :

- SPIN_SW0_CT (73EB) : Spinner counter port #1
- SPIN_SW1_CT (73EC) : Spinner counter port #2
- S0_C0 (73EE) : Segment #0 data, port #1
- S0_C1 (73EF) : Segment #0 data, port #2
- S1_C0 (73F0) : Segment #1 data, port #1
- S1_C1 (73F1) : Segment #1 data, port #2

OUTPUTS:

None

CALLS:

None

CALLED BY:

POWER_UP

NOTES:

CONTROLLER_MAP (8008), DBNCE_BUFF (73D7-73D8) are needed.
A, B, IX, IY are affected.

CONT_READ

INPUT:

H = 0 for player #1, 1 for player #2.

FUNCTION(S):

Return the complement value of the controller port H segment 0 data (joystick data).

OUTPUTS:

A = Raw data from controller H

CALLS:

None

CALLED BY:

DECODER

NOTES:

Because the returned value in register A is the complement of the controller port data, bits 1 mean data, bits 0 mean no data.
Only register A is affected.

CONTROLLER_SCAN

ADDRESS : 1F76

INPUT:

None

FUNCTION(S):

Update SO_CO, S0_C1, S1_C0, S1_C1 by reading segment 0 and 1 from both controller ports.

OUTPUT:

None

CALLS:

None

CALLED BY:

POLLER or under user program control.

NOTES:

Because the returned value in register A is the complement of the controller port data, bits 1 mean data, bits 0 mean no data.
Destroys A

UPDATE_SPINNER

ADDRESS : 1F88

INPUT:

None

FUNCTION(S):

Update counters pointed by SPIN_SW0_CT and SPIN_SW1_CT by reading bit 4 and 5 from segment 1 of both controller ports.

OUTPUT:

None

CALLS:

None

CALLED BY:

Under user program control.

NOTES:

Destroys A, HL

DECODER

ADDRESS : 1F79

INPUT:

H = Controller number (0 for player #1, 1 for player #2)
L = Segment number (0 for fire+joystick, 1 for arm+keyboard)

FUNCTION(S):

If L = segment number 0
- Load spinner counter SPIN_SW0_CT or SPIN_SW1_CT in E
- Call CONT_READ
- Load joystick data in L (A AND 0F)
- Load Fire state in H (A AND 40)

Else L = segment number 1
- Call CONT_READ
- Load decoded key value in L (0 = key 0,… , 9 = key 9, A = key *, B = key #, F = no key)
- Load Arm state in H (A AND 40)

OUTPUT:

H = State of Fire (if segment number = 0), Arm (if segment number = 1)
L = State of Joystick (if segment number = 0), Keyboard (if segment number = 1)
E = Spinner (if segment number = 0)

CALLS:

None

CALLED BY:

Under user program control.

NOTES:

Destroys All

MISCELLANEOUS

BOOT_UP

ADDRESS : 0000

INPUT:

None

FUNCTION(S):

Set stack (= 073B9h)
Continue the execution by calling POWER_UP

OUTPUTS:

None

CALLS:

POWER_UP

CALLED BY:

RESET or POWER ON

NOTES:

None

RAND_GEN

ADDRESS : 1FFD

INPUT:

None

FUNCTION(S):

Set of bit operations on RAND_NUM to calculate next pseudo random value.

OUTPUTS:

HL = (RAND_NUM)
A = L

CALLS:

None

CALLED BY:

Under user program control

NOTES:

None

POWER_UP

INPUT:

None

FUNCTION(S):

Check for the presence of a game cartridge (at 08000h)
If cartridge (rom) first two bytes are 055h and 0AAh, then start game immidiatly.
Otherwise,

- Turn off sound
- Initialize pseudo random
- Initialize controller to strobe reset
- Set no defered writes to VRAM
- Set no sprites multiplexing
- Continuing the execution by displaying the logo screen

OUTPUTS:

None

CALLS:

TURN_OFF_SOUND
CONTROLLER_INIT
DISPLAY_LOGO

CALLED BY:

BOOT_UP

NOTES:

None

DECLSN

ADDRESS : 0190

INPUT:

HL = pointer to a byte value.

FUNCTION(S):

Decrement low nibble (LSN) of a byte pointed to by HL without affecting the high nibble part (MSN).

OUTPUTS:

(HL) = old MSN | new LSN
Z flag set if decrement LSN results in 0, reset otherwise.
C flag set if decrement LSN results in -1 (F), reset otherwise.
A = 0 | new LSN

CALLS:

None

CALLED BY:

FREQ_SWEEP
ATN_SWEEP

NOTES:

HL is preserved

DECMSN

ADDRESS : 019B

INPUT:

HL = pointer to a byte value.

FUNCTION(S):

Decrement high nibble (MSN) part of a byte pointed to by HL without affecting the low nibble part (LSN).

OUTPUTS:

(HL) = new MSN | old LSN
Z flag set if decrement MSN results in 0, reset otherwise.
C flag set if decrement MSN results in -1 (F), reset otherwise.
A = 0 | new MSN

CALLS:

None

CALLED BY:

None

NOTES:

HL is preserved

MSNTOLSN

ADDRESS : 01A6

INPUT:

HL = pointer to a byte value.

FUNCTION(S):

Copy high nibble (MSN) part of a byte value pointed to by HL to the low nibble part (LSN) opf that byte.

OUTPUTS:

(HL) = MSN | MSN

CALLS:

None

CALLED BY:

FREQ_SWEEP
ATN_SWEEP

NOTES:

HL is preserved

ADD816

ADDRESS : 01B1

INPUT:

HL = pointer to a word value.
A = signed byte value [-128,127].

FUNCTION(S):

Adds 8 bit two's complement signed value passed in A to the 16 bit value pointed to by HL.

OUTPUTS:

(HL) = (HL) + A

CALLS:

None

CALLED BY:

FREQ_SWEEP

NOTES:

HL is returned intact

DISPLAY_LOGO

DESCRIPTION:

Displays the Coleco logo screen with COLECOVISION on a black background. If no cartridge is detected, a default message is displayed during 60 seconds, instructing the operator to turn game off before inserting cartridge or expansion module. Otherwise, the game title, manufacturer, and copyright year are obtained from the cartridge, and overlayed onto the logo screen for a period of 10 seconds before game starts.

INPUT:

None

FUNCTION(S):

Clean up the 16K VRAM (0000h-4000h)
Set default screen mode 1 by calling MODE_1
Load default ASCII by calling LOAD_ASCII
Load logo pattern
Put logo on screen
Add ™ beside the logo
Put year 1982 centered at the bottom
Load logo colors
Enable display
Test if a cartridge is present :

1. If it's present,
   - Add game informations from cartridge to screen (title, company, year)
   - Wait 10 seconds
   - Turn off display and start game
2. Otherwise, display default message during 60 seconds then turn off display

OUTPUTS:

None

CALLS:

FILL_VRAM
PUT_VRAM
WRITE_REGISTER
MODE_1
LOAD_ASCII
(START_GAME)

CALLED BY:

POWER_UP

NOTES:

None

OS 7' ABSOLUTE LISTING

OS 7' Listing from the ColecoVision Programmers' Manual rev. 5 © Coleco Industries, Inc. 1982
Disassembled with DASMx v1.30 © Copyright 1996-1999 Conquest Consultants.
Restored by Daniel Bienvenu October, 2004.

OPERATING SYSTEM

```
;      Author:       Coleco Indutries Inc.
;                    Advanced Research & Development - Software Engineering
;
;      UserID:       OS
;
;      Starting date:      A long long time ago in a galaxy far far away . . .
;
;      Prom release date:  24 Nov 1982. For internal use only
;      Prom release rev:   7B
;
;      Prom release date:  December 28, 1982
;      Prom release rev:   7PRIME
;
;      Header Rev: 2
;
; ***************************************************************************
; *                                                                         *
; *                       ColecoVision Operating System                     *
; *                      Absolute Listing ( REV 7PRIME )                    *
; *                         © Coleco Industries 1982                        *
; *                                                                         *
; *                            *** Confidential ***                         *
; *                                                                         *
; ***************************************************************************
;
;      This listing has the actual address of the start of OS routines
;
;      Rev History (one line note indicating the change)
;
;      Rev.   Date                 Change
;      4      14feb1983            Filler locations changed to 0FFH to
;                                  reflect OS_7PRIME. Prom release date
;                                  changed to December 28, 1982 from May
;                                  1982.  Name change to OS_7PRIME to
;                                  reflect majority of versions in the
;                                  field at this date.
;
;      3      24nov1982            Timing change to shorten LOGO delay
;                                  Title changes to JMPTABLES and OSSR_EQU
;
;      2      6oct1982             Minor comment modifications
;
;      1      23sept1982           OS_7 as one absolute file
;
;      0      may 1982             OS_7 listing by module
;
; ===========================================================================
; =                                 EXPORTS                                 =
; ===========================================================================
;
; * ENTRY POINTS TO OS ROUTINES
;
;      GLB    INIT_TABLE                ; TABLE MANAGER
;      GLB    GET_VRAM
;      GLB    PUT_VRAM
;      GLB    INIT_SPR_ORDER
;      GLB    WR_SPR_NM_TBL
;      GLB    INIT_TABLEP               ; PASCAL CALLS
;      GLB    GET_VRAMP
```

```
;       GLB    PUT_VRAMP
;       GLB    INIT_SPR_ORDERP
;       GLB    WR_SPR_NM_TBLP
;
;       GLB    WRITE_REGISTER            ; VIDEO DRIVERS
;       GLB    READ_REGISTER
;       GLB    WRITE_VRAM
;       GLB    READ_VRAM
;       GLB    INIT_WRITER
;       GLB    WRITER
;       GLB    WRITE_REGISTERP           ; PASCAL CALLS
;       GLB    WRITE_VRAMP
;       GLB    READ_VRAMP
;       GLB    INIT_WRITERP
;
;       GLB    POLLER              ; CONTROLLER ROUTINES
;       GLB    UPDATE_SPINNER
;       GLB    CONTROLLER_SCAN
;       GLB    DECODER
;
;       GLB    SOUND_INIT                ; SOUND ROUTINES
;       GLB    TURN_OFF_SOUND
;       GLB    PLAY_IT
;       GLB    SOUND_MAN
;       GLB    PLAY_SONGS
;       GLB    SOUND_INITP               ; PASCAL CALLS
;       GLB    PLAY_ITP
;
;       GLB    INIT_TIMER                ; TIME MGMT ROUTINES
;       GLB    FREE_SIGNAL
;       GLB    REQUEST_SIGNAL
;       GLB    TEST_SIGNAL
;       GLB    TIME_MGR
;       GLB    INIT_TIMERP               ; PASCAL CALLS
;       GLB    FREE_SIGNALP
;       GLB    REQUEST_SIGNALP
;       GLB    TEST_SIGNALP
;
;       GLB    STACK                     ; MISC GLOBALS
;       GLB    VDP_STATUS_BYTE
;       GLB    VDP_MODE_WORD
;       GLB    AMERICA
;       GLB    MUX_SPRITES
;       GLB    DEFER_WRITES
;       GLB    RAND_GEN            ; Can be called from PASCAL or ASM language
;
;       GLB    PUTOBJ              ; GRAPHICS ROUTINES
;       GLB    ACTIVATE
;       GLB    REFLECT_VERTICAL
;       GLB    REFLECT_HORIZONTAL
;       GLB    ROTATE_90
;       GLB    ENLARGE
;       GLB    PUTOBJP                   ; PASCAL CALLS
;       GLB    ACTIVATEP
;
;       GLB    GAME_OPT            ;GAME OPTIONS DISPLAY
;       GLB    LOAD_ASCII          ;LOADS ASCII CHARACTER GENERATORS
;       GLB    FILL_VRAM           ;FILLS DESIGNATED AREA OF VRAM WITH VALUE
;       GLB    MODE_1              ;SETS UP A DEFAULT GRAPHICS MODE 1
;       GLB    ASCII_TABLE         ;POINTER TO TABLE OF ASCII GENERATORS
;       GLB    NUMBER_TABLE  ;POINTER TO TABLE OF 0-9 PATTERN GENERATORS
;
; ===========================================================================
```

```
; =                        CARTRIDGE ROM DATA AREA                        =
; =======================================================================
;
CARTRIDGE           EQU    08000H
; This is the memory location tested to see if a cartridge is plugged
; in. If it contains the pattern AA55H the OS assumes that a game
; cartridge is present. If it contains the pattern 55AAH, the OS
; assumes that a test cartridge is present (bypass Coleco logo screen).
;
LOCAL_SPR_TABLE     EQU    08002H
; This is a pointer to the CPU RAM copy of the sprite name table. The
; table copy is used whenever one level of indirection is desired in
; addressing the VRAM table. For example when using the OS sprite
; multiplexing software.
;
SPRITE_ORDER        EQU    08004H
; This is a pointer to the CPU RAM sprite order table. This table is
; used to order the local sprite name table.
;
WORK_BUFFER         EQU    08006H
; This is a pointer to a free buffer space in RAM. The object oriented
; graphics routines used this buffer for temporary storage.
;
CONTROLLER_MAP      EQU    08008H
; This is a pointer to the controller memory map that is maintained by
; the high-level controller scanning and debounce software.
;
START_GAME          EQU    0800AH
; This is a pointer to the start of the game
;
; =======================================================================
; =                     RESTART AND INTERRUPT VECTORS                     =
; =======================================================================
;
RST_8H_RAM          EQU    0800CH
; This is the restart 8 soft vector.
;
RST_10H_RAM         EQU    0800FH
; This is the restart 10 soft vector.
;
RST_18H_RAM         EQU    08012H
; This is the restart 18 soft vector.
;
RST_20H_RAM         EQU    08015H
; This is the restart 20 soft vector.
;
RST_28H_RAM         EQU    08018H
; This is the restart 28 soft vector.
;
RST_30H_RAM         EQU    0801BH
; This is the restart 30 soft vector.
;
IRQ_INT_VECTOR      EQU    0801EH
; This is the maskable interrupt soft vector.
;
NMI_INT_VECTOR      EQU    08021H
; This is the non maskable interrupt (NMI) soft vector.
;
GAME_NAME           EQU    08024H
; From here to START_GAME there should be a string of ASCII characters
; names that has the following form:
;
;      NAME_OF_THIS_GAME/MAKER_OF_THIS_GAME/COPYWRITE_YEAR.
```

```
;
;      For example:
;
;             "DONKEY KONG/NINTENDO/1982"
;
; IMPORTANT NOTE *****
;
;                   **** IT IS THE RESPONSIBILITY OF THE ****
;                   **** CARTRIDGE PROGRAMMER TO PLACE   ****
;                   **** THESE CODES IN CARTRIDGE ROM    ****
;
; ========================================================================
; =                         OPERATING SYSTEM ROM CODE                    =
; ========================================================================
;
; ************************** PAGE ZERO *****************************
; * PAGE ZERO CONTAINS THE RESTART VECTORS, INTERRUPT VECTORS, AND
; * THE INTERRUPT VECTORING SOFTWARE, AS WELL AS THE DEFAULT HANDLERS
; * FOR INTERRUPTS AND RESTARTS.
;
;      .IDENT OS           ;includes BOOT_UP,RAND_GEN_,PARAM_
;
; #Globals
;      GLB    BOOT_UP,RAND_GEN_,PARAM_
;
; #Externals
;      EXT    TURN_OFF_SOUND,CONTROLLER_INIT,DISPLAY_LOGO
;
; INCLUDE    OSSR_EQU:OS:0 ;equates
;
; #Defines
STACK               EQU    073B9H
DEFER_WRITES        EQU    073C6H
MUX_SPRITES         EQU    073C7H
RAND_NUM            EQU    073C8H
;
; BOOT-UP ROUTINE
;
; The BOOT-UP routine handles power on resets and restarts to 0.
; It initializes the stack and jumps to the POWER_UP routine.
;
BOOT_UP:
;      * Kick stack
       ld     sp,STACK
;      * jump to POWER_UP
       jp     POWER_UP
;
filler_0006:
       db     0FFH,0FFH
;
;
; RESTART VECTORS
;
; The following are the 8 programmable restarts. For each of the
; restart locations below ther is a vector in cartridge ROM.
; To use a restart, the programmer must place the address of the
; routine which he/she wishes to access through the restart at the
; corresponding vector. Thereafter every time that restart is
; executed, the cartridge programmer's routine will be called.
;
RST_8H:
       jp     RST_8H_RAM
;
```

```
filler_000B:
       db     0FFH,0FFH,0FFH,0FFH,0FFH
;
RST_10H:
       jp     RST_10H_RAM
;
filler_0013:
       db     0FFH,0FFH,0FFH,0FFH,0FFH
;
RST_18H:
       jp     RST_18H_RAM
;
filler_001B:
       db     0FFH,0FFH,0FFH,0FFH,0FFH
;
RST_20H:
       jp     RST_20H_RAM
;
filler_0023:
       db     0FFH,0FFH,0FFH,0FFH,0FFH
;
RST_28H:
       jp     RST_28H_RAM
;
filler_002B:
       db     0FFH,0FFH,0FFH,0FFH,0FFH
;
RST_30H:
       jp     RST_30H_RAM
;
filler_0033:
       db     0FFH,0FFH,0FFH,0FFH,0FFH
;
; MASKABLE INTERRUPT VECTORING SOFTWARE
;
; A maskable interrupt occuring in the system is equivalent to a
; restart to 38H. Thus, the maskable interrupt is vectored in exactly
; the same way as the various restarts given above. In order to use
; the interrupt, the cartridge must place the address of his/her
; interrupt handler in the IRQ_INT_VECT location in cartridge ROM.
;
; The cartridge programmer is responsible for saving any registers
; his/her own interrupt handlers may use, and for re-enabling
; interrupts if he/she needs to be re-enabled.
;
IRQ_INTERRUPT:
       jp     IRQ_INT_VECT
;
; ============================================================================
; =                          RANDOM NUMBER GENERATOR                         =
; ============================================================================
;
; (PLACED HERE FOR PURPOSES OF CODE COMPACTION)
;
; Random number generator (pseudo) for a 16 bit value
; This routine 'exclusive or's the 15th and 8th bit
; together. It then rotates the entire quantity to the
; left and inserts the 'exclusive or'ed bit into the
; rightmost bit. Upon leaving it stores the random number
; in a specified memory location.
;
; The random number can be accessed from the global location
; RAND_NUM or the HL pair or the Accumulator.
```

```
;
RAND_GEN_:
       ld     hl,(RAND_NUM)
       bit    7,h
       jr     z,NOT_ON
       bit    0,h
       jr     z,SET
       jr     RESET
;
NOT_ON:
       bit    0,h
       jr     z,RESET
SET:
       scf
       jr     CARRY_READY
;
RESET:
       or     a
CARRY_READY:
       rl     l
       rl     h
       ld     (RAND_NUM),hl
       ld     a,l
       ret
;
filler_0059:
       db     0FFH,0FFH,0FFH,0FFH,0FFH
       db     0FFH,0FFH,0FFH,0FFH,0FFH
       db     0FFH,0FFH,0FFH
;
; THE NMI VECTORING SOFTWARE AND DEFAULT HANDLER
;
; When an NMI is raised by the VDP in the ColecoVision system, it
; causes the CPU to restart to 66h. The vectoring software for the
; NMI is identical to that for the maskable interrupt except that
; it gets its vector from NMI_INT_VECT instead of IRQ_INT_VECT.
;
; Again the cartridge programmer is responsible, in his/her own
; interrupt handlers for saving and restoring the processor state
; when necessary, and for cleaning the VDP condition by reading the
; VDP status register.
;
NMI_INTERRUPT:
       jp     NMI_INT_VECT
;
; ===============================================================================
; =                            OS ROM DATA AREA                                 =
; ===============================================================================
;
AMERICA:
       db     60
; This byte should be used whenever the cartridge programmer wants to
; set up real-time counters. It has a value of 60 for ColecoVisions
; marketed in the USA and 50 for european untis. Use of this byte
; ensures cartridge compatibility at least where real-time counting
; is concerned
;
ASCII_TABLE:
       dw     ASCII_TBL
; This is the address of the Rom pattern generators for uppercase
; ASCII which are contained within the operating system.
;
NUMBER_TABLE:
```

```
       dw     NUMBER_TBL
; This is the address of the ROM pattern generators for the numbers
; 0-9 which are contained within the operating system.
;
; ******************** POWER ON BOOT SOFTWARE ************************
; BOOT_UP       SINCE THE VIDEO GAME SYSTEM MAY BE STARTED UP WITH A
;               GAME CARTRIDGE, KEYBOARD MODULE, OR BOTH (OR NOTHING)
;               INSTALLED AT BOOT UP, THE SOFTWARE MUST PERFORM THE
;               FOLLOWING:
;               A.  INITIALIZE THE INTERRUPT VECTORS.
;               B.  INITIALIZE RESTART VECTORS.
;               C.  TURN OFF THE SOUND CHIP.
;               D.  DETERMINE IF A CARTRIDGE IS PLUGGED IN.
;                   IF SO, BRANCH TO THE CARTRIDGE PROGRAM
;                   ELSE, WAIT FOR CARTRIDGE.
;
FALSE         EQU          0
TRUE          EQU          1
; * VALUES FOR BOOLEAN FLAGS
;
; * BEGIN OF POWER_UP
POWER_UP:
; * IF CARTRIDGE = 55AAH THEN EXIT TO START_GAME (TEST)
       ld     hl,(CARTRIDGE)
       ld     a,l
       cp     055H
       jp     nz,NO_TEST_
       ld     a,h
       cp     0AAH
       jp     nz,NO_TEST_
       ld     hl,(START_GAME)
       jp     (hl)          ;INFO: index jump
; * ELSE
NO_TEST_:
; * TURN OFF SOUND CHIP
       call   TURN_OFF_SOUND
; * INITIALIZE RANDOM NUMBER GENERATOR
       ld     hl,00033H
       ld     (RAND_NUM),hl
; * CLEAR CONTROLLER BUFFER AREAS
       call   CONTROLLER_INIT
; * DEFER_WRITES := FALSE
       ld     a,000H
       ld     (DEFER_WRITES),a
; * MUX_SPRITES := FALSE
       ld     (MUX_SPRITES),a
; * EXIT TO DISPLAY LOGO AND TEST FOR CARTRIDGE
       jp     DISPLAY_LOGO
;
; COMMON PARAMETER PASSING ROUTINE
; To copy PASCAL functions parameters to CPU RAM (complex)
;
; * BEGIN OF PARAM_
PARAM_:
       pop    hl
       ex     (sp),hl
       push   hl
       ld     a,(bc)
       ld     l,a
       inc    bc
       ld     a,(bc)
       inc    bc
       ld     h,a
```

```
       ex     (sp),hl
       push   de
L00A3:
       ld     e,(hl)
       inc    hl
       ld     d,(hl)
       inc    hl
       push   hl
       ld     a,e
       or     d
       jp     nz,L00B7
       pop    hl
       ld     e,(hl)
       inc    hl
       ld     d,(hl)
       inc    hl
       push   hl
       ex     de,hl
       ld     e,(hl)
       inc    hl
       ld     d,(hl)
L00B7:
       inc    bc
       ld     a,(bc)
       rlca
       jp     nc,L00DA
       inc    bc
       pop    hl
       ex     (sp),hl
       ld     (hl),e
       inc    hl
       ld     (hl),d
       inc    hl
L00C4:
       pop    de
       ex     (sp),hl
       dec    hl
       xor    a
       cp     h
       jp     nz,L00D0
       cp     l
       jp     z,L00D6
L00D0:
       ex     (sp),hl
       push   hl
       ex     de,hl
       jp     L00A3
;
L00D6:
       pop    hl
       ex     de,hl
       ex     (sp),hl
       jp     (hl)          ;INFO: index jump
;
L00DA:
       pop    hl
       ex     (sp),hl
       push   hl
       rrca
       ld     h,a
       dec    bc
       ld     a,(bc)
       ld     l,a
```

```
       ex     (sp),hl
       inc    bc
       inc    bc
L00E5:
       ld     a,(de)
       ld     (hl),a
       inc    hl
       inc    de
       ex     (sp),hl
       dec    hl
       xor    a
       cp     l
       jp     nz,L00F4
       cp     h
       jp     z,L00F8
L00F4:
       ex     (sp),hl
       jp     L00E5
;
L00F8:
       pop    hl
       jp     L00C4
;
; ============================================================================
; =                              SYSTEM RAM AREA                             =
; ============================================================================
;
SYSTEM_RAM_AREA            EQU    073BAH
; This is the RAM area dedicated to the basic OS needs. It includes the
; stack, various status variables, and all the variables used by OS
; routines.
;
STACK                      EQU    SYSTEM_RAM_AREA-1
; This is the TOP of the STACK
;
PARAM_AREA                 EQU    073BAH ; 9 bytes
; This is the common parameter passing area and the hole in the data
; area that is provided to make room for it.
; * To extract the parameters (PASCAL CALLS)
; * To initialize sound and timer data
; * ETC.
;
TIMER_LENGTH        EQU    073C0H
TEST_SIG_NUM        EQU    073C2H
;
VDP_MODE_WORD       EQU    073c3H ; 2 bytes
; The VDP mode word contains a copy of the data in the 1st two VDP
; registers. By examining this data, the OS and cartridge programs
; can make mode-dependent decisions about the sprite size or VRAM
; table arrangement. This word is maintained by the WRITE_REGISTER
; routine whenever the contents of registers 0 or 1 are changed.
;
; IMPORTANT NOTE *****
;
;                  **** IT IS THE RESPONSIBILITY OF THE ****
;                  **** CARTRIDGE PROGRAMMER TO MAKE    ****
;                  **** SURE THAT NON-STANDARD USE OF   ****
;                  **** THE VDP REGISTERS DOES NOT MAKE ****
;                  **** THE DATA IN THIS WORD INVALID   ****
;
VDP_STATUS_BYTE            EQU    073c5H
; The default handler for the NMI, which must read the VDP status
; register to clear the interrupt condition, places its contents
```

```
; here. This byte is the most accurate representation of the actual
; VDP status that is available to the cartridge programmer provided
; that the VDP interrupt is enabled on-chip
;
DEFER_WRITES        EQU    073C6H
; This is a boolean flag which is set to FALSE at power up time,
; should be set to true only if the cartridge programmer wishes
; to defer writes to VRAM. If this flag is true then the writer
; routine must be called regularly to perform defered writes.
;
MUX_SPRITES                EQU    073C7H
; This boolean flag with default FALSE value should be set to TRUE if
; the cartridge programmer wishes one level of indirection to be
; inserted into sprite processing by having all sprites written to
; a local SPRITE_NAME_TABLE before being written to VRAM. This aids
; sprite multiplexing solution to the 5th sprite problem.
;
RAND_NUM                   EQU    073C8H ; 2 bytes
; This is the shift register used by the random number generator.
; It is initialized at power-up.
```

SOUND ROUTINE EQUATES

```
;      Operating System Sound Routine EQUATES
;      FILE NAME: OSSR.EQU
;      *** Equates ***
;
; Dedicated Cartridge RAM locations
DEDAREA             EQU    07020H
; * DEDAREA is the start of the RAM area dedicated to sound routines
PTR_TO_LST_OF_SND_ADDRS          EQU    DEDAREA+0
PTR_TO_S_ON_0               EQU    DEDAREA+2
PTR_TO_S_ON_1               EQU    DEDAREA+4
PTR_TO_S_ON_2               EQU    DEDAREA+6
PTR_TO_S_ON_3               EQU    DEDAREA+8
SAVE_CTRL                        EQU    DEDAREA+10
;
; Attenuation level codes
OFF           EQU    00FH   ; [no sound]
;
; Sound output port
SOUND_PORT          EQU    0FFH   ; data to sound chip thru this port
;
; Special byte 0 codes
INACTIVE            EQU    0FFH
SEFFECT             EQU    62     ; special sound effect code
ENDSDATA            EQU    0
;
; Offsets within an SxDATA song data area
CH            EQU    0      ; channel
SONGNO EQU    0      ; song number
NEXTNOTEPTR         EQU 1
FREQ          EQU    3      ; frequency
ATN           EQU    4      ; attenuation
CTRL          EQU    4
NLEN          EQU    5      ; noise
FPS           EQU    6      ; frequency sweep
FPSV          EQU    6
FSTEP         EQU    7
ALEN          EQU    8      ; attenuation sweep
ASTEP         EQU    8
APS           EQU    9
APSV          EQU    9
;
; Song end codes
CH0END        EQU    010H
CH1END        EQU    050H
CH2END        EQU    090H
CH3END        EQU    0D0H
CH0REP        EQU    018H
CH1REP        EQU    058H
CH2REP        EQU    098H
CH3REP        EQU    0D8H
;
; Channel numbers, B7-B6
CH0           EQU    000H
CH1           EQU    040H
CH2           EQU    080H
CH3           EQU    0C0H
```

FREQ SWEEP RTN

```
;       .IDENT FREQSWE      ;includes FREQ_SWEEP
;
; INCLUDE     OSSR_EQU:OS:0 ;equates
; #Globals
;       GLB    FREQ_SWEEP
; #Externals
;       GLB    DECLSN,DECMSN,MSNTOLSN,ADD816
; #Defines
FSTEP         EQU    007H
FPSV          EQU    006H
NLEN          EQU    005H
;*************************
;*       FREQ_SWEEP       *
;*************************
;.COMMENT }
;See User's Manual for description
;RETs Z SET: if note over
;RETs Z RESET: if sweep in progress or note not over
;}
FREQ_SWEEP:
;       * if freq not swept, dec NLEN and RET [setting Z flag]
        ld     a,(ix+FSTEP)  ;check for no sweep code
        cp     000H                ;set Z flag if FSTEP = 0
;       if     [psw,is,zero] ;note not to be swept
        jr     nz,L20
        ld     a,(ix+NLEN)         ;dec NLEN and
        dec    a                   ;SET Z flag if NLEN = 0
        ret    z                   ;leave if note over with Z flag SET
        ld     (ix+NLEN),a         ;store decremented NLEN
        ret                        ;RET with Z flag RESET [note not over]
;       * sweep going, so decrement FPSV
L20:
        push   ix                  ;point HL to FPSV
        pop    hl
        ld     e,FPSV
        ld     d,000H
        add    hl,de
        call   DECLSN       ;decrement FPSV
;       if     [psw,is,zero] ;FPSV  has timed out
        jr     nz,L21
;       * decrement NLEN and leave if sweep is over
        call   MSNTOLSN            ;reload FPSV from FPS
        dec    hl                  ;point to NLEN [# steps in the sweep]
        ld     a,(hl)       ;decrement NLEN and
        dec    a                   ;SET Z flag if NLEN = 0
        ret    z                   ;leave if sweep over with Z flag set
;       * sweep not over, so add FSTEP to FREQ
        ld     (hl),a       ;store decremented NLEN
        dec    hl                  ;point HL to FREQ
        dec    hl
        ld     a,(ix+007H)         ;A = FSTEP [two's complement step size]
        call   ADD816       ;FREQ = FREQ + FSTEP
        inc    hl                  ;point HL to hi FREQ
        res    2,(hl)       ;RESET B2 in hi FREQ in case add cased > 10 bit
        or     0FFH                ;RESET Z flag, sweep not over yet
L21:
        ret
```

ATTENUATION SWEEP RTN

```
;       .IDENT ATNSWEE      ;includes ATN_SWEEP
;
; INCLUDE     OSSR_EQU:OS:0 ;equates
; #Globals
;       GLB    ATN_SWEEP
; #Externals
;       GLB    DECLSN,DECMSN,MSNTOLSN
; #Defines
APSV          EQU    009H
;*************************
;*       ATN_SWEEP        *
;*************************
;.COMMENT }
;See User's Manual for description
;RETs Z SET: if byte 8 is 0 [means sweep is over, or note was never swept]
;RETs Z RESET: if sweep in progress
;}
ATN_SWEEP:
;       * RET with Z SET if byte 8 = 00
        ld     a,(ix+008H)         ;check byte 8 for no sweep code
        cp     000H                ;Z is set if byte 8=0
        ret    z                   ;leave if Z set, sweep not going
;       * sweep going, so dec APSV
        push   ix                  ;point HL to APSV
        pop    hl
        ld     d,000H
        ld     e,APSV
        add    hl,de
        call   DECLSN       ;dec APSV [LSN of byte 9]
;       if     [psw,is,zero];APSV has timed out
        jr     nz,L22
;       * decrement ALEN to see if sweep over
        call   MSNTOLSN            ;reload APSV from APS
        dec    hl                  ;point to ALEN [# of steps in the sweep]
        call   DECLSN       ;dec ALEN [LSN byte 8]
        jr     z,L23
;       * add ASTEP to ATN
        ld     a,(hl)       ;MSN A = ASTEP
        and    0F0H                ;mask LSN
        ld     e,a                 ;E = ASTEP | 0
        dec    hl                  ;point HL to ATN
        dec    hl
        dec    hl
        dec    hl
        ld     a,(hl)       ;MSN A = ATN
        and    0F0H                ;A = ATN | 0
        add    a,e                 ;MSN A = [ASTEP + ATN] | 0
        ld     e,a                 ;Saved in E
        ld     a,(hl)       ;A = ATN | freq or CTRL
        and    00FH                ;mask old ATN  A = 0 | freq or CTRL
        or     e                   ;OR in new ATN
        ld     (hl),a       ;store updated value back into song data area
        or     0FFH                ;RESET Z flag, sweep not over yet
        jr     L22
;       ELSE
L23:
        ld     (hl),000H           ;set byte 8 to 0 to indicate end sweep
L22:
        ret
```

UTILITY

```
;      .IDENT UTIL          ;includes UPATNCTRL,UPFREQ,
;                           ;DECLSN,DECMSN,MSNTOLSN,ADD816,PT_IX_SxDATA,
;                           ;LEAVE_EFFECT,AREA_SONG_IS
;
; #Globals
;      GLB    UPATNCTRL,UPFREQ
;      GLB    DECLSN,DECMSN,MSNTOLSN
;      GLB    ADD816
;      GLB    PT_IX_SxDATA
;      GLB    LEAVE_EFFECT
;      GLB    AREA_SONG_IS
;
; INCLUDE     OSSR_EQU:OS:0 ;equates
;
; #Defines
;INACTIVE     EQU    0FFH
FREQ          EQU    003H
SOUND_PORT    EQU    0FFH

;*************************
;*       UPATNCTRL       *
;*************************
;.COMMENT }
;Perform single byte update of the snd chip noise control register or any
;attanuation register. IX is passed pointing to byte 0 of a song data area, MSN
;register C = formatted channel attenuation code.
;}
UPATNCTRL:
       ld     a,(ix+004H)          ;MSN A=ATN, LSN may be CTRL data
       bit    4,c                  ;test for ATN
;      if     [psw,is,nzero]       ;ATN is to be sent, move it to LSN
       jr     z,L24
       rrca                        ;swap nibbles
       rrca
       rrca
       rrca
L24:
       and    00FH                 ;mask MSN
       or     c                    ;a = formatted register# | ATN or CTRL
       out    (SOUND_PORT),a       ;output ATN or CTRL data
       ret
;
;*************************
;*         UPFREQ        *
;*************************
;.COMMENT }
;Perform double byte update of a sound chip frequency register. IX is passed
;pointing to byte0 of a song data area, MSN register D = formatted channel
;frequency code.
;}
UPFREQ:
       ld     a,(ix+FREQ)      ;A =  F2  F3  F4  F5  F6  F7  F8  F9
       and    00FH             ;A =   0   0   0   0  F6  F7  F8  F9
       or     d                ;A = FORMATTED REG# | F6  F7  F8  F9
       out    (SOUND_PORT),a   ;output 1st freq byte
       ld     a,(ix+FREQ)      ;A =  F2  F3  F4  F5  F6  F7  F8  F9
       and    0F0H             ;A =  F2  F3  F4  F5   0   0   0   0
       ld     d,a              ;save in D
       ld     a,(ix+FREQ+1)    ;LSN A = 0 0 F0 F1
```

```
        and     00FH            ;A =   0   0   0   0   0   0  F0  F1
        or      d               ;A =  F2  F3  F4  F5   0   0  F0  F1
        rrca                    ;swap nibbles
        rrca
        rrca
        rrca                    ;A =   0   0  F0  F1  F2  F3  F4  F5
        out     (SOUND_PORT),a  ;output 2nd [most significant] freq byte
        ret
;
;*************************
;*         DECLSN        *
;*************************
;.COMMENT }
;Without affecting the MSN, decrement the LSN of the byte pointed to by HL.
;HL remains the same.
;RET with Z flag set if dec LSN results in 0, reset otherwise.
;RET with C flag set if dec LSN results in -1, reset otherwise.
;}
DECLSN:
        ld      a,000H
        rrd                     ;A = 0 | LSN [HL]
        sub     001H            ;Z flag set if dec to 0, C flag if dec to -1
        push    af              ;save Z and C flag
        rld                     ;[HL] = old MSN | new LSN
        pop     af              ;restore Z and C flags, A = 0 | new LSN
        ret
;
;*************************
;*         DECMSN        *
;*************************
;.COMMENT }
;Without affecting the LSN, decrement the MSN of the byte pointed to by HL.
;HL remains the same.
;RET with Z flag set if dec MSN results in 0, reset otherwise.
;RET with C flag set if dec MSN results in -1, reset otherwise.
;}
DECMSN:
        ld      a,000H
        rld                     ;A = 0 | MSN [HL]
        sub     001H            ;Z flag set if dec to 0, C flag if dec to -1
        push    af              ;save Z and C flag
        rrd                     ;[HL] = new MSN | old LSN
        pop     af              ;restore Z and C flags, A = 0 | new MSN
        ret
;
;*************************
;*        MSNTOLSN       *
;*************************
;.COMMENT }
;Copy MSN of the byte pointed to by HL to the LSN of that byte.
;HL remains the same.
;}
MSNTOLSN:
        ld      a,(hl)          ;A = MSN | LSN to be changed
        and     0F0H            ;A = MSN | 0
        ld      b,a             ;save in B
        rrca                    ;swap nibbles
        rrca
        rrca
        rrca                    ;A = 0 | MSN
        or      b               ;A = MSN | MSN
        ld      (hl),a          ;[HL] = MSN | MSN
        ret
```

```
;
;*************************
;*         ADD816        *
;*************************
;.COMMENT }
;Adds 8 bit two's complement signed value passed in A to the 16 bit location
;pointed to by HL.
;}
ADD816:
       ld     b,000H        ;set B for positive value in A
       bit    7,a           ;if A is positive
       jr     z,POS         ;skip
       ld     b,0FFH        ;A is neg: extend sign bit thru B
POS:
       add    a,(hl)        ;do 8 bit add [and set Carry]
       ld     (hl),a        ;store result into LSN 16 bits number
       inc    hl            ;put MSB
       ld     a,(hl)        ;into A
       adc    a,b           ;A = MSB + Carry + B [B is 0 or FF]
       ld     (hl),a        ;store result into MSN
       dec    hl            ;re-point HL to LSB 16 bit number
       ret
;
;*************************
;*    PT_IX_TO_SxDATA    *
;*************************
;.COMMENT }
;SONGNO passed in B.
;Point IX to byte 0 in SONGNO's song data area.
;RET with both DE and IX pointing to SxDATA,
;HL pointing to MSB SxDATA entry in LST_OF_SND_ADDRS.
;}
PT_IX_TO_SxDATA:
; * IX & DE := addr of byte 0 in SONGNO's song data area,
; ; HL pointing to MSB SxDATA entry in LST_OF_SND_ADDRS.
       ;point HL to start LST_OF_SND_ADDRS
       ld     hl,(PTR_TO_LST_OF_SND_ADDRS)
       dec    hl                  ;init HL for addition
       dec    hl
       ld     c,b                 ;from 4*SONGNO in C
       ld     b,000H
       rlc    c
       rlc    c
       add    hl,bc               ;HL pts to SxDATA's entry in LST_OF_SND_ADDRS
       ld     e,(hl)        ;move addr SxDATA to IX thry DE
       inc    hl
       ld     d,(hl)
       push   de
       pop    ix
       ret
;
;*************************
;*      LEAVE_EFFECT     *
;*************************
;.COMMENT }
;LEAVE_EFFECT, called by a special sound effect routine when it's finished,
;restores the SONGNO of song to which the effect note belongs to B5-B0 of
;byte 0 in the effect's data area, and loads bytes 1 and 2 with the address of
;the next note in the song.  The address of the 1 byte SONGNO (saved by the
;effect when 1st called) is passed in DE.  The 2 byte address of the next note
;in the song, also saved by the effect, is passed in HL.  IX is assumed to be
;pointing to byte 0 of the data area to which the song number is to be
;restored. Bits 7 and 6 of the saved SONGNO are ignored, and therefore may be
```

```
;used by the effect to store flag information during the course of the note.
;}
LEAVE_EFFECT:
       ld     (ix+001H),l   ;LSB NEXT_NOTE_PTR := LSB addr next note in song
       ld     (ix+002H),h   ;MSB NEXT_NOTE_PTR := MSB addr next note in song
       ld     a,(de)      ;A := x x SONGNO (i.e., the saved, original SONGNO)
       and    03FH          ;A := 0 0 SONGNO
       ld     b,a           ;Saved in B
       ld     a,(ix+000H)   ;A := CH# | 62 (all effect notes have SONGNO = 62)
       and    0C0H          ;A := CH# 0 0 0 0 0 0
       or     b             ;A := CH# | SONGNO
       ld     (ix+000H),a   ;restore song number
       ret
;
;*************************
;*      AREA_SONG_IS      *
;*************************
;.COMMENT }
;The address of byte 0 of a song data area is passed in IX.  The song # of
;the song using that area is returned in A [0FFH if inactive].  If a special
;effect was using that area, 62 is returned in A and HL is returned with the
;address of the special sound effect routine.
;}
AREA_SONG_IS:
       ld     a,(ix+000H)         ;A := CH# | SONGNO or 62, or A := FF
       cp     0FFH
       ret    z                   ;leave if A = FF (area inactive)
       and    03FH                ;mask CH#
       cp     03EH
       ret    nz                  ;leave with A = SONGNO (not a special effect)
       push   ix                  ;point HL to byte 1
       pop    hl
       inc    hl
       ld     e,(hl)              ;save LSB effect addr in E
       inc    hl                  ;HL to byte 2
       ld     d,(hl)              ;save MSB effect addr in D
       ex     de,hl               ;HL := addr special effect
       ret
```

INIT SOUND

```
;      .IDENT INITSOU      ;includes INIT_SOUND,ALL_OFF
;*************************
;*       INIT_SOUND       *
;*************************
;.COMMENT }
;see Users' Manual for description; includes ENTRY POINT ALL_OFF
;addr LST_OF_SND_ADDRS passed in HL
;n = # of song data areas to init, passed in B
;}
;
; #Globals
;      GLB    INIT_SOUND,ALL_OFF,DUMAREA
;
; INCLUDE     OSSR_EQU:OS:0 ;equates
;
; #Defines
OFF           EQU    0FFH
SR1ATN EQU    090H
SR2ATN EQU    0B0H
SR3ATN EQU    0D0H
SRNATN EQU    0F0H
SR1FRQ EQU    080H
SR2FRQ EQU    0A0H
SR3FRQ EQU    0C0H
SRNCTL EQU    0E0H

SOUND_PORT    EQU    0FFH

INIT_SOUND_PAR:
       dw     00002H
       dw     00001H
       dw     00002H
;
INIT_SOUNDQ:
       ld     bc,INIT_SOUND_PAR
       ld     de,PARAM_AREA
       call   PARAM_
       ld     a,(PARAM_AREA)
       ld     b,a
       ld     hl,(PARAM_AREA+1)
INIT_SOUND:
;      * initialize PTR_TO_LST_OF_SND_ADDRS with value passed in HL
       ld     (PTR_TO_LST_OF_SND_ADDRS),hl
;      * store inactive code at byte 0 of each of the n data areas [B=n]
       inc    hl            ;pt HL to song 1 data area entry in LST_OF_SND_ADDRS
       inc    hl
       ld     e,(hl)        ;pt DE to byte 0 in first song data area
       inc    hl
       ld     d,(hl)
       ex     de,hl         ;pt HL to byte 0 in first song data area
       ld     e,00AH        ;set DE for 10 byte increment
       ld     d,000H
B1:
       ld     (hl),0FFH     ;deactivate area
       add    hl,de         ;pt HL to byte 0 next area (10 bytes away)
       djnz   B1            ;do this for n (passed in B) data areas
;      * store end of data area code (0) at 1st byte after last song data area
       ld     (hl),000H     ;store end of data area code in byte 0 data area n+1
;      * set the 4 channel data area pointers to a dummy, inactive data area
```

```
        ld      hl,DUMAREA    ;point HL to inactive byte below [after the RET]
        ld      (PTR_TO_S_ON_0),hl  ;store addr DUMAREA at PTR_TO_S_ON_0
        ld      (PTR_TO_S_ON_1),hl  ;store addr DUMAREA at PTR_TO_S_ON_1
        ld      (PTR_TO_S_ON_2),hl  ;store addr DUMAREA at PTR_TO_S_ON_2
        ld      (PTR_TO_S_ON_3),hl  ;store addr DUMAREA at PTR_TO_S_ON_3
;       * initialize SAVE_CTRL
        ld      a,0FFH        ;note: this is only time MSN SAVE_CTRL
                              ; will be non zero,
        ld      (SAVE_CTRL),a ;thus ensuring PLAY_SONGS will output
                              ; 1st real CTRL data
ALL_OFF:
;       * turn off all 4 sound generators
        ld      a,SR1ATN+OFF        ;form off code for tone generator 1
        out     (SOUND_PORT),a      ;send it out
        ld      a,SR2ATN+OFF        ;form off code for tone generator 2
        out     (SOUND_PORT),a      ;send it out
        ld      a,SR3ATN+OFF        ;form off code for tone generator 3
        out     (SOUND_PORT),a      ;send it out
        ld      a,SRNATN+OFF        ;form off code for noise generator, N
        out     (SOUND_PORT),a      ;send it out
        ret
;
DUMAREA:
        db      0FFH
```

JUKEBOX

```
;      .IDENT JUKEBOX      ;includes JUKE_BOX
;*************************
;*        JUKE_BOX        *
;*************************
;.COMMENT }
;see Users' Manual for description
;SONGNO passed in B
;}
;
; #Globals
;      GLB    JUKE_BOX
;      GLB    JUKE_BOXQ
;
; #Externals
;      EXT    PT_IX_TO_SxDATA,LOAD_NEXT_NOTE,UP_CH_DATA_PTRS
;
; INCLUDE     OSSR_EQU:OS:0 ;equates
;

JUKE_BOX_PAR:
       dw     00001H
       dw     00001H
;
JUKE_BOXQ:
       ld     bc,JUKE_BOX_PAR
       ld     de,PARAM_AREA
       call   PARAM_
       ld     a,(PARAM_AREA)
       ld     b,a
JUKE_BOX:
;      * RET if song already in progress
       push   bc                        ;save SONGNO on stack
       call   PT_IX_TO_SxDATA           ;point IX to SONGNO's song data area
       ld     a,(ix+000H)               ;A := CH# [if any] | SONGNO [if any]
       and    03FH                      ;A := 0 0 SONGNO
       pop    bc                        ;B := SONGNO
       cp     b                         ;test if already in progress
       ret    z                         ;if so, leave
;      * load 1st note and set NEXT_NOTE_PTR [thru LOAD_NEXT_NOTE]
       ld     (ix+000H),b   ;store SONGNO in byte 0
       dec    hl            ;-HL left by PT_IX_TO_SxDATA
                            ; pointing to MSB SxDATA
       dec    hl            ;-entry in LST_OF_SND_ADDRS; point HL to note list
       ld     d,(hl)        ;-starting addr entry in LST_OF_SND_ADDRS
                            ; and save this
       dec    hl            ;-addr in DE
       ld     e,(hl)        ;DE now has the initial value for NEXT_NOTE_PTR
       ld     (ix+001H),e         ;set NEXT_NOTE_PTR for 1st note in song
       ld     (ix+002H),d
       call   LOAD_NEXT_NOTE      ;load note, byte 0 := CH#|SONGNO,
                                  ; set new NEXT_NOTE_PTR
       call   UP_CH_DATA_PTRS     ;new song, so update channel data ptrs
       ret
```

SOUND MANAGER

```
;       .IDENT SNDMAN;includes TONE_OUT
;*************************
;*       SONG_MANAGER      *
;*************************
;.COMMENT }
;see Users' Manual for description
;}
;
; #Globals
;       GLB     SND_MANAGER
;       GLB     UP_CH_DATA_PTRS
;       GLB     PROCESS_DATA_AREA
;       GLB     EFXOVER
;
; #Externals
;       EXT     PT_IX_TO_SxDATA,AREA_SONG_IS
;       EXT     DUMAREA
;       EXT     LOAD_NEXT_NOTE,ATN_SWEEP,FREQ_SWEEP
;
; INCLUDE     OSSR_EQU:OS:0;equates
ENDSDATA            EQU     000H
INACTIVE            EQU     0FFH

SND_MANAGER:
;       * IX := addr of song #1 data area [S1DATA]
        ld      b,001H          ;pt IX to byte 0 song data area for song# 1
        call    PT_IX_TO_SxDATA
;       LOOP until end of song data areas
L1:
        ld      a,ENDSDATA              ;check for end of song data areas
        cp      (ix+000H)               ;set Z flag if negative
        ret     z                       ;leave [Z set],
; if all data areas have been processed
;       * process active song data areas
        call    PROCESS_DATA_AREA   ;update counters of call effect; get next note
;       * point IX to byte 0 next song data area
        ld      e,00AH
        ld      d,000H
        add     ix,de
        jr      L1      ; repeat loop
;*************************
;*    UP_CH_DATA_PTRS     *
;*************************
;.COMMENT }
;For each active data area, starting with S1DATA and proceeding in order, load
;the associated channel data area pointer [PTR_TO_S_ON_x] with the address of
;byte 0.  This routine is called by JUKE_BOX, when a song starts and
;PROCESS_DATA_AREA when the channel using a data area has changed as a result
;of calling LOAD_NEXT_NOTE [this happens when a song finishes and when it
;switches back and forth between noise and tone notes].
;}
UP_CH_DATA_PTRS:
        push    ix                  ;save curent IX
        ld      hl,DUMAREA          ;set all 4 ch data ptrs to dummy inactive area
        ld      (PTR_TO_S_ON_0),hl
        ld      (PTR_TO_S_ON_1),hl
        ld      (PTR_TO_S_ON_2),hl
        ld      (PTR_TO_S_ON_3),hl
        ld      b,001H
```

```
        call   PT_IX_TO_SxDATA
;       LOOP until end of song data areas
L2:
        ld     a,(ix+000H)
        cp     ENDSDATA            ;check for end of song data areas
        jr     z,DONE_SNDMAN       ;leave loop if all data areas checked
;       * if area active, set appropriate channel data area pointer
        cp     INACTIVE            ;check for inactive data area:
                                   ; don't up date ptr if so
;       if     [PSW,IS,ZERO];area is active: update channel data ptrs
        jr     z,L9
        ld     a,(ix+000H)         ;get CH# in A
        and    0C0H                ;B7 - B6 in A = CH#
        rlca                       ;form CH# * 2 in A,i.e., the offset from
        rlca                       ;PTR_TO_S_ON0 of channel data area pointer
        rlca                       ;that points to channel CH#
        ld     e,a                 ;add offset to addr of PTR_TO_C_ON_0
        ld     d,000H
        ld     hl,PTR_TO_S_ON_0
        add    hl,de               ;HL points to proper channel data area pointer
        push   ix                  ;store this song data area's byte 0 addr there
        pop    de
        ld     (hl),e
        inc    hl
        ld     (hl),d
;       * point IX to byte 0 next song data area
L9:
        ld     e,00AH
        ld     d,000H
        add    ix,de
        jr     L2      ;repeat loop
;
DONE_SNDMAN:
        pop    ix      ;restore IX
        ret
;
;*************************
;*    UP_CH_DATA_PTRS     *
;*************************
;.COMMENT }
;See Users' Manual for description
;Terminology: SFX = address of sound effect routine
;}
PROCESS_DATA_AREA:
        call   AREA_SONG_IS ;return area's SONGNO in A [and addr SFX in HL]
        cp     INACTIVE            ;test for inactive code
        ret    z                   ;RET, no processing if area inactive
;       * if special effect, call it to process the data area
        cp     03EH                ;test for special sound effect
        jr     nz,L10
        ld     e,007H        ;pt HL to SFX+7, starting adr of the effect's code
        ld     d,000H
        add    hl,de
        jp     (hl)          ;do 1 pass thru effect, RET from effect
;
;       * else process a non-effect note
L10:
        call   ATN_SWEEP           ;process atn sweep data, if any
        call   FREQ_SWEEP    ;proc frq sweep data, if any, & note dura timers
;       if [psw,is,zero]           ;note is over
        jr     nz,L12
EFXOVER:
        ld     a,(ix+000H)         ;A := CH# | SONGNO this note
```

```
        push    af                  ;save on stack
        call    LOAD_NEXT_NOTE      ;load data for next note
        pop     bc                  ;B := CH# | SONGNO previous note
        ld      a,(ix+000H)         ;A := CH# | SONGNO new note [may be inactive]
        cp      b                   ;check against new note's CH# | SONGNO
;       if      [psw,is,nzero]      ;change to/from tone/efx/noise
        jr      z,L12
        call    UP_CH_DATA_PTRS     ;to maintain data area priority system
L12:
        ret
```

PLAY SONG

```
;      .IDENT PLAYSON      ;includes TONE_OUT
;*************************
;*       PLAY_SONGS_       *
;*************************
;.COMMENT }
;see Users' Manual for description
;SFX refers to the beginning address of a special sound effect routine
;}
;
; #Globals
;      GLB    PLAY_SONGS_,TONE_OUT
;
; #Externals
;      EXT    UPATNCTRL,UPFREQ
;
; INCLUDE     OSSR_EQU:OS:0 ;equates
;
; #Defines
OFF           EQU    00FH
SR1ATN        EQU    090H
SR2ATN        EQU    0B0H
SR3ATN        EQU    0D0H
SRNATN        EQU    0F0H
SR1FRQ        EQU    080H
SR2FRQ        EQU    0A0H
SR3FRQ        EQU    0C0H
SRNCTL        EQU    0E0H

SOUND_PORT    EQU    0FFH

PLAY_SONGS_:
;      * output CH1 attenuation and frequency
       ld     a,SR1ATN+OFF        ;format CH1 OFF byte into A
       ld     c,SR1ATN                   ;format MSN C for CH1 attenuation
       ld     d,SR1FRQ                   ;format MSN D for CH1 frequency
       ld     ix,(PTR_TO_S_ON_1)  ;point IX to byte 0 data area
                                  ; of song for CH1
       call   TONE_OUT
;      * output CH2 attenuation and frequency
       ld     a,SR2ATN+OFF        ;format CH2 OFF byte into A
       ld     c,SR2ATN                   ;format MSN C for CH2 attenuation
       ld     d,SR2FRQ                   ;format MSN D for CH2 frequency
       ld     ix,(PTR_TO_S_ON_2)  ;point IX to byte 0 data area
; of song for CH2
       call   TONE_OUT
;      * output CH3 attenuation and frequency
       ld     a,SR3ATN+OFF        ;format CH3 OFF byte into A
       ld     c,SR3ATN                   ;format MSN C for CH3 attenuation
       ld     d,SR3FRQ                   ;format MSN D for CH3 frequency
       ld     ix,(PTR_TO_S_ON_3)  ;point IX to byte 0 data area
                                  ; of song for CH3
       call   TONE_OUT
;      * output CH0 [noise] ATN [and CTRL, if different from last time]
       ld     a,SRNATN+OFF        ;format CH0 OFF byte into A
       ld     c,SRNATN                   ;format MSN C for CH0 attenuation
       ld     ix,(PTR_TO_S_ON_0)  ;point IX to byte 0 data area
                                  ; of song for CH0
       ld     e,(ix+000H)                ;look for inactive code 0FFH
       inc    e                          ;this sets Z flag if E = 0FFH
```

```
;       if      [psw,is,zero]       ; song data area is inactive
        jr      nz,L5
        out     (SOUND_PORT),a                  ;turn off CH0
        jr      L6
;       ELSE
L5:
        call    UPATNCTRL                       ;send out current ATN
        ld      a,(ix+004H)                     ;LSN A = current CTRL data
        and     00FH                            ;mask MSN
        ld      hl,SAVE_CTRL        ;point to last CTRL data sent
        cp      (hl)                            ;compare
;       if      [psw,is,nzero]                  ;CTRL has changed
        jr      z,L6
        ld      (hl),a              ;SAVE_CTRL = new CTRL data
        ld      c,SRNCTL                        ;send new CTRL data
        call    UPATNCTRL
L6:
        ret
;
TONE_OUT:
        ld      e,(ix+000H)                     ;look for inactive code, oFFH
        inc     e                               ;this sets Z flag if E = 0FFH
;       if      [psw,is,zero]       ;song data area is inactive
        jr      nz,L7
        out     (SOUND_PORT),a                  ;turn off CHx
        jr      L8
;       ELSE                                    ;send out current ATN and FREQ
L7:
        call    UPATNCTRL                       ;send out attenuation
        call    UPFREQ              ;send out frequency
L8:
        ret
```

LOAD NEXT NOTE

```
;       .IDENT LOADNEX      ;includes LOAD_NEXT_NOTE
;*************************
;*      LOAD_NEXT_NOTE     *
;*************************
;.COMMENT }
;see Users' Manual for description
;SFX refers to the beginning address of a special sound effect routine
;}
;
; #Globals
;       GLB    LOAD_NEXT_NOTE
;       GLB    REST,ENDREP,ENDNOREP,EFFECT,TYPE0,TYPE1,TYPE2,TYPE3
;       GLB    MODB0,DE_TO_DEST,PASS1
;
; #Externals
;       EXT    JUKE_BOX
;
; INCLUDE     OSSR_EQU:OS:0 ;equates
;
; #Defines
ATN           EQU    4
NLEN          EQU    5
FSTEP         EQU    7
ASTEP         EQU    8
INACTIVE      EQU    0FFH

LOAD_NEXT_NOTE:
       ; * deactivate area, save SONGNO on stack
       ld     a,(ix+000H)          ;A := byte 0
       and    03FH                 ;mask CH#, if any
       push   af                   ;save SONGNO on stack
       ld     (ix+000H), INACTIVE ;deactivate area
       ; A := header new note
       ld     l,(ix+001H)          ;HL := addr new note in ROM
       ld     h,(ix+002H)
       ld     a,(hl)        ;A := header new note
       ; * save header of new note in song on stack
       ;   and load its data CASE note type
       ld     b,a                  ;save header new not in B
;      - test for rest
       bit    5,a                  ;test for rest
;      if     [psw,is,nzero]       ;note is rest
       jr     z,L13
;      --CASE-- rest
REST:
       push   bc                   ;save header on stack
       and    01FH                 ;mask all but duration bits
       inc    hl            ;HL = addr of the header of the note after this note
       ld     (ix+001H),l          ;store in NEXT_NOTE_PTR
       ld     (ix+002H),h
; move this note's data and fill in bytes where necessary
       ld     (ix+ATN),0F0H;set stn off
       ld     (ix+NLEN),a          ;NLEN := 5 bit duration
       ld     (ix+FSTEP),000H      ;indicate freq not to be swept
       ld     (ix+ASTEP),000H      ;indicate atn not to be swept
       jp     MODB0
;
;      - test for end of song
L13:
```

```
        bit     4,a                 ;test for end
;       if      [psw,is,nzero]      ;end of song
        jr      z,L14
        bit     3,a                 ;test for repeat
;       if      [PSW,is,nzero]      ;end of song
        jr      z,ENDNOREP
;       --CASE-- end song, repeat
ENDREP:
        pop     bc                  ;B := SONGNO
        call    JUKE_BOX            ;to reload 1st note of this song
        ret                         ;to PROCESS_DATA_AREA, don't save header
;
;       --CASE-- end song, no repeat
ENDNOREP:
        ld      a, INACTIVE
        push    af                  ;save inactive code to end song
        jp      MODB0               ;to load byte 0
;
;       - test for special sound effect
L14:
        and     03CH                ;mask irrelevant bits
        cp      004H                ;test for B5 - B2 = 0001
;       if      [psw,is,zero];note is a special effect
        jr      nz,L15
;       --CASE-- special effect
EFFECT:
        pop     iy                  ;IY := SONGNO
        push    iy                  ;put SONGNO back on stack
        push    bc      ;save header on stack; NEXT_NOTE_PTR := SFX, DE := SFX
        inc     hl              ;-pt HL to next byte [LSB addr SFX]
        ld      e,(hl) ;-E := LSB SFX
        ld      (ix+001H),e     ;-put LSB of SFX in byte 1 of SxDATA [NEXT_NOTE_PTR]
        inc     hl              ;-pt HL to NSB SFX
        ld      d,(hl) ;-D := NSB SFX
        ld      (ix+002H),d     ;-put NSB SFX in byte 2 of SxDATA
        inc     hl                      ;point HL to next note [after this new note]
        push    iy                      ;A := SONGNO
        pop     af
        push    de                      ;PASS1 on the stack
        pop     iy
        ld      de,PASS1        ;create "CALL [IY]" with RET to PASS1 by storing
        push    de                      ;PASS1 on stack
        jp      (iy)            ;1st 7 bytes SFX will save addr next note & SONGNO
;
PASS1:
        ld      d,000H          ; in same fashion, create a "CALL (IY+7)"
        ld      e,007H          ; to allow SFX to load initial values
        add     iy,de
        ld      de,MODB0                ;RET to MODB0
        push    de
        jp      (iy)            ;INFO: index jump
;       - if here, note is type 0 - 3
L15:
        push    bc                      ; save header on stack
        ld      a,b                     ; A := fresh copy header
        and     003H                    ; mask all but type number
        cp      000H                    ; test for type 0
;       if      [psw,is,zero]; note is type 0: fixed freq and atn
        jr      nz,L16
;       --CASE-- note type 0
;       * set up NEXT_NOTE_PTR
TYPE0:
        inc     hl              ; next note [after this new note] is 4 bytes away,
```

```
        inc     hl                  ; point HL to it
        inc     hl
        inc     hl
        ld      (ix+001H),l         ; put addr in NEXT_NOTE_PTR
        ld      (ix+002H),h
        ; move new note data and fill in bytes where necessary
        dec     hl                  ; point HL back to 1st ROM data to move, NLEN
        ld      de,00005H           ; point DE to destination: bytes 5,4, and 3
        call    DE_TO_DEST
        ld      bc,00003H           ; move 3 bytes
        lddr
        ld      (ix+FSTEP),000H     ; set for no freq sweep
        ld      (ix+ASTEP),000H     ; set for no atn sweep
        jr      MODB0
;
L16:
        cp      001H                ; test for type 1
;       if      [psw,is,zero]; note is type 1: swept freq, fixed attenuation
        jr      nz,L17
;       --CASE-- not type 1
;       * set up  NEXT_NOTE_PTR
TYPE1:
        ld      e,006H        ; note after this note is 6 bytes away,
        ld      d,000H        ; pt HL to it
        add     hl,de
        ld      (ix+001H),l         ; store in NEXT_NOTE_PTR
        ld      (ix+002H),h
        ; move new note data and fill in bytes where necessary
        dec     hl                  ; point HL back to 1st ROM data to move, FSTEP
        inc     e                   ; E:=7; point DE to destination: bytes 7 - 3
        call    DE_TO_DEST
        ld      bc,00005H           ; move 5 bytes
        lddr
        ld      (ix+ASTEP),000H     ; set for no atn sweep
        jr      MODB0
;
L17:
        cp      002H                ; test for type 2
;       if      [psw,is,zero]; note is type 2: fixed freq, swept attenuation
        jr      nz,TYPE3
;       --CASE-- note type 2
;       * set up NEXT_NOTE_PTR
TYPE2:
        ld      e,006H        ; pt HL to note after this note
; since it's 6 bytes away,
        ld      d,000H        ; pt HL to it by adding 6
        add     hl,de
        pop     af                  ; A := header this note [CH# |SONGNO]
        push    af                  ; put back on stack
        and     0C0H                ; mask SONGNO, leaving CH#
;       if      [psw,is,zero]       ; This is a noise note,
                                    ; which is only 5 ROM bytes long
        jr      nz,L18
        dec     hl                  ; so move HL back 1 byte
L18:
        ld      (ix+001H),l         ; put addr in NEXT_NOTE_PTR
        ld      (ix+002H),h
; move new note data and fill in bytes where necessary
        dec     hl                  ; point HL back to 1st ROM data to move, APS
        ld      e,009H        ; point DE to destination: bytes 9,8,5 - 3
        call    DE_TO_DEST
        ld      bc,00002H           ; move 2 bytes
```

```
       lddr                        ; when done, DE points to FSTEP, HL to ROM
NLEN
       ld     a,000H
       ld     (de),a               ; FSTEP := 0 for no freq sweep
       dec    de                   ; pt DE to RAM NLEN
       dec    de
       ld     c,003H               ; move last 3 ROM bytes
                                   ; if this is a noise note, garbage
       lddr                        ; will be loaded into byte 3, buts that's OK
       jr     MODB0
;
TYPE3:
       ld     e,008H       ; note after this note is 8 bytes away
       ld     d,000H       ; pt HL to it
       add    hl,de
       ld     (ix+001H),l          ; put addr in NEXT_NOTE_PTR
       ld     (ix+002H),h
       ; move new note data and fill in bytes where necessary
       dec    hl                   ; Point HL back to 1st ROM data to move, APS
       push   ix                   ; Point DE to destination: bytes 9-3
       pop    iy                   ; IY := Addr byte 0 [and DE = 6]
       ld     e,009H               ; DE := 9
       add    iy,de                ; IY := Addr byte 9 [APS]
       push   iy
       pop    de                   ; DE := addr APS
       ld     bc,00007H            ; move 7 bytes
       lddr
MODB0:
       push   ix                   ;pt HL to byte 0
       pop    hl
       pop    af                   ; A := Header new note
       pop    bc                   ; B := SONGNO
       cp     0FFH                 ; Test for inactive [song over, as detected
above]
       ret    z
       ld     d,a                  ; Save header in D
       and    03FH                 ; Rid channel bits
       cp     004H                 ; Special effect
       jr     nz,L20_LOAD_NEX
       ld     b,03EH
L20_LOAD_NEX:
       ld     a,d                  ; Restore A to header
       and    0C0H                 ; A := CH# 0 0 0 0 0 0
       or     b                    ; A := new CH# | SONGNO
       ld     (hl),a               ; Store back in byte 0
L19:
       ret
;
DE_TO_DEST:
;DE passed = offset from byte 0, RETed with address byte offset
       push   ix
       pop    iy                   ; IY := Addr byte 0 (and DE = offset)
       add    iy,de                ; IY := Addr byte 0 + offset
       push   iy
       pop    de                   ; DE := Addr of destination byte in SxDATA
       ret
```

ACTIVATE

```
;      .IDENT ACTIVATE
;      .ZOP
;      .EPOP
;      .COMMENT }
;****************** ACTIVATE ************************************************
;                                                                    4/22/82
;                                                                    13:50:00
;
; The following changes/revisions were made:
;
;      1. Eliminate code placing OLD_SCREEN address in status area
;      2. Init X_PAT_POS in OLD_SCREEN when in VRAM as well as when in CPU RAM
;      3. Use VDP_MODE_WORD to test graphics mode
;      4. Add code to expand one color generator byte to 8
;      5. Added C_BUFF defs 8 for color expanding code
;      6. Fix color gen move in mode 1 (5/02)
;      7. Use CONTROLER_MAP for buffer area
;
;
; ACTIVATE is used to initialize the RAM status area for the passed
; object and move its pattern and color generators to the PATTERN and
; COLOR generator tables in VRAM_  The second function is enabled or
; disabled by setting or resetting the carry flag in the PSW_  this is
; necessary to prevent sending the same graphics data to VRAM more than
; once when creating identical objects_  The calling sequence for
; activating an object is as follows:
;
;      LD     HL,OBJ_n      ; ->OBJ to activate
;      SCF                  ; Signal MV to VRAM
;      CALL   ACTIVATE
;
; OR
;
;      LD     HL,OBJ_n      ; ->OBJ to activate
;      OR     A             ; Don't MV to VRAM
;      CALL ACTIVATE
;
; #External
;      EXT    PUT_VRAM_,VRAM_WRITE,VDP_MODE_WORD
;      EXT    WORK_BUFFER
;
; #Global
;      GLB    ACTIVATEQ, ACTIVATE_
;
; Register usage: Following will be changed by activate, additional
; may be changed by called SUBR.
;                   AF,HL,DE,BC,IY
;
; PROCEDURE ACTIVATEQ[VAR OBJ:OBJECT;MOVE:BOOLEAN];
;
; ACTIVATEQ is the Pascal entry point to ACTIVATE
;
;      EXT    PARAM_
; The Pascal parameter passing procedure
; #Common
;      PRM_DATA:    DEFS   3      ; Moved to OS
; This is the common parameter passing area

ACTIVATE_P:
```

```
        dw      00002H
        dw      0FFFEH
        dw      00001H
;
ACTIVATEQ:
        ld      bc,ACTIVATE_P
        ld      de,PARAM_AREA
        call    PARAM_
        ld      hl,(PARAM_AREA)
        ld      e,(hl)
        inc     hl
        ld      d,(hl)
        ex      de,hl
        ld      a,(PARAM_AREA+2)
        cp      000H
        jr      z,NTZZZ_
        scf
        jr      TZZZ_
;
NTZZZ_:
        or      a
TZZZ_:
ACTIVATE_:
; SUP pointers etc_ common to all subcases
;       HL -> OBJ DEF CPU ROM
;       C FLG = SUP VRAM FLG
        ld      e,(hl)              ;->OBJ GEN CPU ROM
        inc     hl
        ld      d,(hl)
        inc     hl
        ld      c,(hl)              ;->OBJ CPU RAM
        inc     hl
        ld      b,(hl)
        inc     hl
        ld      a,000H              ;ZERO FRAME
        ld      (bc),a
        ld      a,(de)              ;GET OBJ_TYPE
        push    af                  ;SV OBJ_TYPE & FLG
        and     00FH                ;GET OBJ_TYPE NUM
        jp      z,ACT_SEMI                  ;TYPE=0
                                            ; SEMI-MOBILE
        dec     a
        jp      z,ACT_MOBILE                ;TYPE=1
                                            ; MOBILE
        dec     a
        jp      z,ACT_0SPRT                 ;TYPE=2
                                            ; 0SPRITE
        dec     a
        jp      z,ACT_1SPRT                 ;TYPE=3
                                            ; 1SPRITE
        dec     a
        jr      z,ACT_CMPLX                 ;TYPE=4
                                            ; COMPLEX
        pop     af                  ;SUBCASE ELSE
        ret
;ON ENTRY TO SUBCASE
;       STACK=OBJ_TYPE & SUP VRAM FLG
;       HL->OBJ_n+4
;       DE->OBJ GRAPHICS+0
;       BC->OBJ STATUS+0
;       A=0
;
ACT_CMPLX:
```

```
; SUBCASE Complex
       ld     a,(de)              ;GET COMP_CNT
       rra
       rra
       rra
       rra
       and    00FH
       ld     b,a                 ;SET CNTR
       ld     e,(hl)              ;DE->comp ptrs list
       inc    hl
       ld     d,(hl)
       inc    hl
       or     a                   ;? EMPTY
       jr     z,CMPLX9
CMPLX4:
       pop    af                  ;Sup call, comp obj
       push   af
       push   hl
       push   bc
       ex     de,hl
       call   ACTIVATE_
       pop    bc                  ;Restore pntrs
       pop    hl
       ld     e,(hl)
       inc    hl
       ld     d,(hl)
       inc    hl
       djnz   CMPLX4              ;? More, reloop
CMPLX9:
       pop    af                  ;Clear stack for rtn
       ret                        ;Technacally should jmp to rtn
;
ACT_SEMI:
; SUBCASE Semi_Mobile
       call   INIT_XP_OS          ;X_PAT_POS := 80H
       ld     a,(de)              ;A := FIRST_GEN_NAME
       ld     l,a
       inc    de
       ld     a,(de)              ;A := NUMGEN
       add    a,l
       ld     (iy+005H),a         ;NEXT_GEN := FIRST_GEN_NAME + NUMGEN
       ld     h,000H              ;HL=FIRST_GEN_NAME
;At this point:
;      STACK=OBJ_TYPE & SUP VRAM FLG
;      HL=FIRST_GEN_NAME
;      DE->NUMGEN
;      BC:FREE
;SUP FOR VRAM INIT
       pop    af                  ;If sup VRAM flg on
       jr     nc,SEMI_EXIT
       push   af
       ld     a,(VDP_MODE_WORD)   ;See which graphics mode
       bit    1,a                 ;If GR II mode
       jr     z,SEMI_GRI          ;-, GO GRI
       ex     de,hl               ;DE=FIRST_GEN_NAME
       ld     b,h                 ;SV->NUMGEN
       ld     c,l
       ld     l,(hl)              ;CALC source offset
       ld     h,000H
       push   hl
       add    hl,hl
       add    hl,hl
       add    hl,hl
```

```
       push   hl
       inc    bc                  ;HL->source buffer
       ld     a,(bc)
       ld     l,a
       inc    bc
       ld     a,(bc)
       ld     h,a
       pop    bc
       pop    iy
       pop    af
;At this point:
;      HL->SOURCE BUFFER, PTRN GNRTRS
;      DE=INDEX TO START OF VRAM ENTRIES
;      IY=NUMBER OF ITEMS TO READ FROM VRAM
;      BC=OFFSET TO COLOR SOURCE BUFFER 2
;      AF=OBJ_TYPE [& SUP VRAM FLG, UNNEEDED]
;Fill as needed top, mid, and bot PTRN_GNRTRS & DITTO for COLOR_GNRTRS
       bit    7,a                 ;If bit 7 OBJ_TYPE ON (top)
       jr     z,SEMI_MID          ;-, GO HNDL MID
       call   SUP_GEN_CLR
SEMI_MID:
       call   SUP_UPDATE
       bit    6,a                 ;If bit 6 OBJ_TYPE ON (mid)
       jr     z,SEMI_BOT
       call   SUP_GEN_CLR
SEMI_BOT:
       call   SUP_UPDATE
       bit    5,a                 ;If bit 5 OBJ_TYPE ON (bot)
       jr     z,SEMI_EXIT
       call   SUP_GEN_CLR
SEMI_EXIT:
       ret
;
SEMI_GRI:
       ex     de,hl               ;HL->NUMGEN
       ld     c,(hl)              ;IY=NUMGEN
       ld     b,000H
       push   bc
       pop    iy
       inc    hl                  ;HL->PTRN_GNRTRS
       ld     a,(hl)
       inc    hl
       ld     h,(hl)
       ld     l,a
       push   hl                  ;SAVE FOR RESTORE
       push   bc
       push   de
       push   iy
       ld     a,003H              ;SIGNAL PTRN GEN FILL
       call   PUT_VRAM_
       pop    bc                  ;BC:=NUMGEN
       pop    hl                  ;HL:=FIRST_GEN_NAME
       ld     e,l
       ld     d,h                 ;DE:=HL
       add    hl,bc               ;HL:=FIRST_GEN_NAME+NUMGEN
       dec    hl
       srl    h
       rr     l
       srl    h
       rr     l
       srl    h
       rr     l                   ;HL:=(FIRST_GEN_NAME+NUMGEN-1)/8
       sra    e
```

```
       sra    e
       sra    e                   ;DE:=FIRST_GEN_NAME/8
       or     a                   ;Clear carry
       sbc    hl,de
       inc    hl            ;HL:=(F_G_N+NMGN-1)/8-F_G_N/8+1=NUMBER_COLR GENS
       push   hl
       pop    iy
       pop    hl                  ;Restore reg
       add    hl,hl               ;Step over PTRN_GNRTRS
       add    hl,hl
       add    hl,hl
       pop    bc
       add    hl,bc               ;HL->COLOR GNRTR SOURCE
       ld     a,004H              ;SIGNAL PTRN COLOR TBL
       call   PUT_VRAM_
       pop    af                  ;FIX STACK
       ret
;Internal routine to initialize X_PAT_POS in OLD_SCREEN
INIT_XP_OS:
       push   bc
       pop    iy                  ;IY->STATUS
       push   de                  ;SAVE->GRAPHICS
       ld     e,(hl)              ;DE:=OLD_SCREEN ADDRESS
       inc    hl
       ld     d,(hl)
       bit    7,d                 ;? OLD SCRN IN CPU ROM
       jr     nz,SM_BY_OLD
       ld     a,d                 ;OLD_SCREEN IN VRAM?
       cp     070H
       jr     c,OS_IN_VRAM
       ld     a,080H              ;INIT X_PAT_POS = 80H
       ld     (de),a
       jr     SM_BY_OLD
;
INIT_80:
       db     080h
OS_IN_VRAM:
       ld     hl,INIT_80
       ld     bc,00001H           ;ONE BYTE TO MOVE TO VRAM
       call   VRAM_WRITE
SM_BY_OLD:
       pop    de                  ;DE->GRAPHICS
       inc    de                  ;DE->FIRST_GEN_NAME
       ret
;
;Internat rtn to setup Ptrn Gen VRAM & Color Gen VRAM
SUP_GEN_CLR:
       push   af                  ;SAVE FOR RESTORE
       push   bc
       push   iy
       push   de
       push   hl
       ld     a,003H              ;Signal PTRN gen fill
       call   PUT_VRAM_
       pop    hl                  ;Restore
       pop    de
       pop    iy
       pop    bc
       pop    af
       push   af
       push   bc                  ;Save for retor
       push   iy
       push   de
```

```
       push   hl
       bit    4,a                ;How many color gen bytes?
       jr     nz,ONE_BYTE
       add    hl,bc              ;HL->Color gen source
       ld     a,004H             ;Signal PTRN color fill
       call   PUT_VRAM_
O_B_RET:
       pop    hl
       pop    de
       pop    iy
       pop    bc
       pop    af
       ret
; For each item to send, duplicate the color byte 8 times [in C_BUFF]
; then send this generator to VRAM color table indexed by DE
ONE_BYTE:
       add    hl,bc              ;HL->Color byte
       ld     c,l
       ld     b,h                ;BC->Color byte
       push   iy
       pop    hl                 ;HL = Item count
NEXT_COLOR:
       push   hl                 ;Save counter
       ld     a,(bc)             ;Get color byte
       push   bc                 ;Save pointer to color
       ld     bc,00008H          ;Create 8 duplicates
       ld     hl,(WORK_BUFFER)
       add    hl,bc              ;Place then here, starting at end of buffer
       ld     b,008H
DUPLI:
       dec    hl
       ld     (hl),a
       djnz   DUPLI
       push   de                 ;Save index into tables
       ld     iy,00001H          ;1 item to send
       ld     a,004H             ;Color table code
       call   PUT_VRAM_
       pop    de                 ;Get index back
       pop    bc                 ;Pointer to color byte
       inc    de                 ;Increment index
       inc    bc                 ;Increment color pointer
       pop    hl                 ;Get item counter
       dec    hl
       ld     a,h
       or     l
       jr     nz,NEXT_COLOR
       jr     O_B_RET
;Internal RTN to update to next VRAM index screen area
SUP_UPDATE:
       push   bc
       ld     bc,00100H
       ex     de,hl
       add    hl,bc
       ex     de,hl
       pop    bc
       ret
;
ACT_MOBILE:
; SUBCASE Mobile
       call   INIT_XP_OS         ;X_PAT_POS := 80H
; Insert new_generator address in object CPU RAM
       inc    de
       ld     a,(de)
```

```
       ld     (iy+005H),a
       inc    de
       ld     a,(de)
       ld     (iy+006H),a         ;Init NEW_GEN in status
       pop    af
       ret
;
ACT_0SPRT:
; SUBCASE Sprite size 0
ACT_1SPRT:
; SUBCASE Sprite size 1
       inc    bc                  ;->NEXT_GEN in CPU RAM
       inc    bc
       inc    bc
       inc    bc
       inc    bc
       ex     de,hl               ;HL->FIRST_GEN_NAME
       inc    hl
       ld     a,(hl)
       ld     e,a                 ;SV index to VRAM
       ld     d,000H
       push   de
       inc    hl                  ;DE=PTRN_PTR
       ld     e,(hl)
       inc    hl
       ld     d,(hl)
       inc    hl                  ;CALC & SET NEXT_GEN CPU RAM
       add    a,(hl)
       ld     (bc),a
       ld     c,(hl)
       ld     b,000H
       push   bc
       pop    iy
       ex     de,hl               ;HL->SOURCE PTRN GEN
       pop    de                  ;DE=INDEX TO PTRN GEN VRAM
       pop    af
       ret    nc
       ld     a,001H              ;Signal sprite PRTN gen fill
       call   PUT_VRAM_
       ret
```

PUT/DEFRD PUT OBJ

```
;****************** PUTOBJ ************************************************
;
;DESCRIPTION: Putobj vectors to one of 5 specific routines for placing the
;                    different object types on the display.
;
;INPUT:              IX = Address of object to be processed
;                     B = Parameter to be passed specific put routines
;
; In addition, this module contains routines which allow VRAM operations
; to be defered, typically until an interrupt occurs, and performed
; in a block by central writer routine.
;
;*****************************************************************************
;
; DATA
; QUEUE_SIZE        DEFS   1      ; 73CA
; This is the size of the defered write queue. It is set by the
; cartridge programmer. It has range 0 - 255.
;
; QUEUE_HEAD        DEFS   1      ; 73CB
; QUEUE_TAIL        DEFS   1      ; 73CC
; These are the indices of the head and tail of the write queue.
;
; HEAD_ADDRESS      DEFS   2      ; 73CD
; TAIL_ADDRESS      DEFS   2      ; 73CF
; These are the addresses of the queue head and tail.
;
; BUFFER            DEFS   2      ; 73D1
; This is a pointer to the beginning of the defered write queue. The
; cartridge programmer is responsible for providing a RAM area to hold
; the queue, and passing its location and size to init_queue.
;
; DEFER_WRITES      DEFS   1      ; 73C6
;
; #Defines
TRUE   EQU    1
FLASE  EQU    0
; Values for boolean deferal_flag.
;
; #Common
; PARAM_AREA        DEFS   3      ; 73BA
; PARAM_AREA is the common parameter passing area for PASCAL entry pts.

;
SET_UP_WRITE:
; SET_UP_WRITE sets up defered VRAM operation.
;
; Put data in QUEUE_HEAD
       push   ix
       ld     hl,(HEAD_ADDRESS)
       pop    de
       ld     (hl),e       ; Put data pointer
       inc    hl
       ld     (hl),d
       inc    hl
       ld     (hl),b       ; Store PUTOBJ      parameter
       inc    hl
       ex     de,hl               ; HEAD address in DE
       ld     a,(QUEUE_HEAD)
```

```
       inc    a                    ; new HEAD in A
; If QUEUE_HEAD = QUEUE_SIZE then
       ld     hl,QUEUE_SIZE
       cp     (hl)
       jr     nz,NOT_TOO_BIG
; QUEUE_HEAD := 0
       ld     a,000H
       ld     (QUEUE_HEAD),a
; If HEAD_ADDRESS := BUFFER
       ld     hl,(BUFFER)
       ld     (HEAD_ADDRESS),hl
       jr     SET_UP_ENDIF
; Else
NOT_TOO_BIG:
; Store new QUEUE_HEAD
       ld     (QUEUE_HEAD),a
; Store HEAD_ADDRESS
       ld     (HEAD_ADDRESS),de
; End if
SET_UP_ENDIF:
       ret
;
; Procedure INIT_QUEUE (SIZE:BYTE;VAR A_QUEUE:QUEUE)
;
; SIZE passed in A, LOCATION passed in HL
; Destroys: A
INIT_QUEUE_P:
       dw     00002H
       dw     00001H
       dw     0FFFEH
; This is the parameter descriptor for INIT_QUEUEQ
;
; Begin INIT_QUEUE
INIT_QUEUEQ:
       ld     bc,INIT_QUEUE_P
       ld     de,PARAM_AREA
       call   PARAM_
       ld     a,(PARAM_AREA)
       ld     hl,(PARAM_AREA+1)
INIT_QUEUE:
                                   ; QUEUE_SIZE := SIZE
       ld     (QUEUE_SIZE),a
                                   ; QUEUE_HEAD := QUEUE_TAIL := 0
       ld     a,000H
       ld     (QUEUE_HEAD),a
       ld     (QUEUE_TAIL),a
                           ; BUFFER := TAIL_ADDRESS := HEAD_ADDRESS := LOCATION
       ld     (BUFFER),hl
       ld     (HEAD_ADDRESS),hl
       ld     (TAIL_ADDRESS),hl
; End INIT_QUEUE
       ret
;
; Procedure WRITER_
;
; Takes no parameters
; Destroys: ALL
;
; Begin WRITER_
WRITER_:
                           ; Save deferal flag
       ld     a,(DEFER_WRITES)
       push   af
```

```
                                ; DEFER_WRITES := FALSE
       ld     a,FALSE
       ld     (DEFER_WRITES),a
                                ; While QUEUE_TAIL <> QUEUE_HEAD Do
WRTR_WHILE:
       ld     a,(QUEUE_TAIL)
       ld     hl,QUEUE_HEAD
       cp     (hl)
       jr     z,WRTR_END_WHILE
                                ; Write data at QUEUE_TAIL to VRAM
       ld     hl,(TAIL_ADDRESS)
       ld     e,(hl)        ; ; Get object pointer
       inc    hl
       ld     d,(hl)
       inc    hl
       ld     b,(hl)        ; ; Get parameter
       inc    hl
                                ; Process object in QUEUE
       push   de
       pop    ix
       push   hl                ;; Save QUEUE_TAIL address
       call   DO_PUTOBJ
                                ; Increment QUEUE_TAIL
       ld     a,(QUEUE_TAIL)
       inc    a
                                ; If QUEUE_TAIL = QUEUE_SIZE Then
       ld     hl,QUEUE_SIZE
       cp     (hl)
       jr     nz,WRTR_ELSE
                                ; QUEUE_TAIL := 0
       ld     a,000H
       ld     (QUEUE_TAIL),a
                                ; TAIL_ADDRESS := BUFFER
       ld     hl,(BUFFER)
       ld     (TAIL_ADDRESS),hl
       pop    hl                ; ; Restore stack pointer
       jr     WRTR_END_IF
; Else
WRTR_ELSE:
                                ; Store new QUEUE_TAIL
       ld     (QUEUE_TAIL),a
                                ; TAIL_ADRESS := TAIL_ADRESS + 3
       pop    hl
       ld     (TAIL_ADDRESS),hl
                                ; End if
WRTR_END_IF:
       jr     WRTR_WHILE
; End While
WRTR_END_WHILE:
                                ; Restore deferal flag
       pop    af
       ld     (DEFER_WRITES),a
; End WRITER_
       ret
;
; #External
; EXT  PUTSEMI, PUT_MOBILE, PUT0SPRITE, PUT1SPRITE, PUTCOMPLEX
; EXT  DEFER_WRITES
; EXT  PARAM_
PUTOBJ_PAR:
       dw     00002H
       dw     00002H
       dw     00001H
```

```
;
; Procedure PUT_OBJP (VAR DATA:BUFFER;PARAM:BYTE);
; This is the PASCAL entry point to the PUT_OBJ routine
; Input: IX := BUFFER, B := PARAM
;
PUTOBJQ:
       ld     bc,PUTOBJ_PAR
       ld     de,PARAM_AREA
       call   PARAM_
       ld     ix,(PARAM_AREA)
       ld     a,(PARAM_AREA+2)
       ld     b,a
PUTOBJ_:
       ld     a,(DEFER_WRITES)    ; Check if defered write is desired
       cp     TRUE
       jr     nz,DO_PUTOBJ ; If not, process object
       call   SET_UP_WRITE ; If so, set up for defered write
       ret
;
DO_PUTOBJ:
       ld     h,(ix+001H)         ; Get address of graphics for OBJ_n
       ld     l,(ix+000H)
       ld     a,(hl)        ; A := OBJ_TYPE
       ld     c,a                 ; SAVE COPY in C
       and    00FH                ; Mask for OBJ_TYPE number
       jp     z,PUTSEMI           ;  0 = SEMI_MOBILE
       dec    a
       jp     z,PUT_MOBILE ;  1 = MOBILE
       dec    a
       jp     z,PUT0SPRITE ;  2 = SPRITE0
       dec    a
       jp     z,PUT1SPRITE ;  3 = SPRITE1
       jp     PUTCOMPLEX          ;  4+= COMPLEX
```

PUT_SEMI

```
;*****************************  PUT_SEMI  ***********************************
;Description:  Puts semi_mobile objects on screen
;
;Input:              IX = Address of object to be processed
;                    HL = Address of object's graphics tables in ROM
;*******************************************************************************
;
;      GLB    PUTSEMI, PX_TO_PTRN_POS, PUT_FRAME, GET_BKGRND
;
PUTSEMI:
       ld     d,(ix+003H)          ; Get address of status
       ld     e,(ix+002H)
       push   de                   ; And put it into IY
       pop    iy
       ld     d,(iy+002H)          ; Get X_LOCATION
       ld     e,(iy+001H)
       call   PX_TO_PTRN_POS
       ld     c,e                  ; C := Pattern plane col.
       ld     d,(iy+004H)          ; Get Y_LOCATION
       ld     e,(iy+003H)
       call   PX_TO_PTRN_POS
       ld     b,e                  ; B := Pattern plane row
       ld     e,(iy+000H)          ; Get frame number
;
; HL = GRAPHICS_n, IX = OBJ_n, IY = STATUS_n, C = COL., B = ROW, E = FRAME
;
       ld     d,000H        ; DE has frame number
       add    hl,de
       add    hl,de                ; 2*Frame number + Addr of graphics_n
       ld     e,005H        ; Frame pointer offset
       add    hl,de                ; HL now points to location holding address
                                   ; of frame
       ld     e,(hl)        ; Get address into DE
       inc    hl
       ld     d,(hl)
       ex     de,hl                ; HL := Address of frame
       push   bc
       pop    de                   ; DE := Y_PAT_POS & X_PAT_POS
       ld     c,(hl)        ; C := X_EXTENT
       inc    hl
       ld     b,(hl)        ; B := Y_EXTENT
       inc    hl                   ; HL points to 1st name in list
;
; Test to see if OLD_SCREEN is to be saved
;
       ld     a,(ix+005H)          ; Get high byte of OLD_SCREEN address
       bit    7,a                  ; Test bit 15 of OLD_SCREEN address
       jr     z,S_OLD_SCRN
       call   PUT_FRAME
       ret
;
S_OLD_SCRN:
       push   bc                   ; Save regs
       push   de
       push   hl
       cp     070H
       jr     z,EQUAL_TO
       jr     c,ELSE_1
;      IF     [.A,GE,70H}          ; Then OLD_SCREEN in CPU RAM
```

```
EQUAL_TO:
       ld     h,a
       ld     l,(ix+004H)          ; HL := OLD_SCREEN address
       ld     a,(hl)
       jr     END_IF_1
;
ELSE_1:                            ; OLD_SCREEN in VRAM
       ld     hl,(WORK_BUFFER)     ; Get address of free buffer space
       ld     d,(ix+005H)          ; DE := OLD_SCREEN address
       ld     e,(ix+004H)
       push   hl                   ; Save 2 copies free buffer addr.
       push   de                   ; Save OLD_SCREEN addr.
       push   hl
       ld     bc,00004H            ; Read 4 bytes [X,Y_PAT_POSs,X,Y_EXTENTs]
       call   VRAM_READ
       pop    hl                   ; HL := Free buffer addr.
       ld     a,(hl)
       cp     080H
       jr     nz,GET_OLD
       pop    de
       jr     SKIP_OLD
;
GET_OLD:
       inc    hl
       inc    hl
       ld     b,(hl)        ; B := X_EXTENT of OLD_SCREEN
       inc    hl
       ld     e,(hl)        ; E := Y_EXTENT
       ld     d,000H
       inc    hl
       ex     de,hl                ; Multiply X_EXTENT*Y_EXTENT in HL
       jr     M_XY+1 ;L0772
;
M_XY:
       add    hl,hl
;L0772:      ;M_XY+1
       djnz   M_XY
       push   hl
       pop    bc                   ; BC := Number of bytes to read
       ex     de,hl                ; HL := Free buff addr + 4
       pop    de                   ; DE := OLD_SCREEN addr.
       inc    de
       inc    de
       inc    de
       inc    de
       call   VRAM_READ            ; Read saved names for background
SKIP_OLD:
       pop    hl                   ; HL := free buffer addr.
END_IF_1:
       ld     a,(hl)        ; A := X_PAT_POS
       cp     080H
       jr     z,END_IF_2
;      IF     [_A,NE,80H]          ; Then there is an OLD_SCREEN
       ld     e,(hl)        ; E := X_PAT_POS
       inc    hl
       ld     d,(hl)        ; D := Y_PAT_POS
       inc    hl
       ld     c,(hl)        ; C := X_EXTENT
       inc    hl
       ld     b,(hl)        ; B := Y_EXTENT
       inc    hl                   ; HL points to 1st name in list
       push   ix                   ; Save object pointer
       call   PUT_FRAME            ; Restore OLD_SCREEN to display
```

```
       pop    ix                  ; Restore object pointer
END_IF_2:
SV1:
       pop    hl                  ; HL := Addr. of 1st name in frame
       pop    de                  ; DE := Y,X_PAT_POSs
       pop    bc                  ; BC := Y,X_EXTENTs
       push   bc
       push   de
       push   hl
       ld     h,(ix+005H)         ; HL := OLD_SCREEN addr.
       ld     l,(ix+004H)
       ld     a,070H
       cp     h
       jr     c,END_IF_3
;      IF     [_H,LT,70H]         ; The OLD_SCREEN now in free buffer
       ld     hl,(WORK_BUFFER)    ; Therefore, move background to buffer
END_IF_3:
       ld     (hl),e       ; OLD_SCREEN +  0 := X_PAT_POS
       inc    hl
       ld     (hl),d       ; OLD_SCREEN +  1 := Y_PAT_POS
       inc    hl
       ld     (hl),c       ; OLD_SCREEN +  2 := X_EXTENT
       inc    hl
       ld     (hl),b       ; OLD_SCREEN +  3 := Y_EXTENT
       inc    hl                  ; HL := Addr. to store names
       push   ix                  ; Save object pointer
       call   GET_BKGRND
       pop    ix                  ; Restore object pointer
       pop    hl                  ; Where names are in CPU RAM
       pop    de                  ; Where to move then in VRAM [NAME TABLE]
       pop    bc                  ; How many to move
       push   ix                  ; Save object pointer
       call   PUT_FRAME
       pop    ix                  ; Restore object pointer
       ld     d,(ix+005H)         ; See if saved background to be moved to VRAM
SV2:
       ld     a,070H
       cp     d
       jr     z,END_IF_4
       jr     c,END_IF_4
;SV2:: IF     [_D,LT,70H]
       ld     e,(ix+004H)         ; DE := OLD_SCREEN addr.
       exx                        ; Use 'reg for calculation
       ld     hl,(WORK_BUFFER)    ; Where next OLD_SCREEN data is
       push   hl
       inc    hl
       inc    hl
       ld     e,(hl)       ; E := X_EXTENT
       ld     d,000H
       inc    hl
       ld     b,(hl)       ; B := Y_EXTENT
       ex     de,hl               ; HL := X_EXTENT
       jr     M_XY2+1
;
M_XY2:
       add    hl,hl               ; HL := X_EXTENT*Y_EXTENT
;M_XY2+1:
       djnz   M_XY2
       push   hl
       exx
       pop    bc                  ; BC := Number of bytes to write
       pop    hl                  ; HL := Free buffer addr.
       call   VRAM_WRITE
```

```
END_IF_4:
       ret
; ***************************** PX_TO_PTRN_POS ******************************
; Description: Divides reg by 8, If signed result > 127 Then E := Max signed
;              Positiv number. If result < -128, Then E := MIN negative num
; Input:    DE = 16 bit signed number
; Output:   DE/8 < -128    E = -128
;   -128 <= DE/8 <= 127    E = DE/8
;    127 <  DE/8           E = 127
; ***************************************************************************
PX_TO_PTRN_POS:
       push   hl                  ; HL used to test magnitude
       sra    d                   ; 16 bit shift left
       rr     e
       sra    d
       rr     e
       sra    d                   ; x3
       rr     e
       bit    7,d                 ; Is result negative
       jr     nz,NEGTV
       ld     hl,0FF80H           ; Is result > -128
       add    hl,de
       pop    hl
       ret    nc
       ld     e,07FH        ; IF > 128 then E := MAX signed + Num
       ret
;
NEGTV:
       ld     hl,00080H
       add    hl,de
       pop    hl
       ret    c
       ld     e,080H        ; IF < -128 then E := MIN signed - Num
       ret
; ************** PUT_FRAME ************************************************
; Description:  The names which constitute a frame are moved to the name table
;               in VRAM. The upper left hand corner of the frame is positioned
;               at X_PAT_POS, Y_PAT_POS.
; Input:        HL = Addr. of list of names [in CPU RAM]
;               D,E = Y_PAT_POS, X_PAT_POS
;               B,C = Y_EXTENT, X_EXTENT
; ***************************************************************************
PUT_FRAME:
       push   bc                  ; Copy parameters into primed registers
       push   de
       push   hl                  ; And frame address into DE'
       exx
       pop    hl
       pop    de
       pop    bc
       call   CALC_OFFSET
       exx
; %%%%%
; Test for the following condition:
;              (X_PAT_POS sle 32) and (X_PAT_POS + X_EXTENT sgt 0)
PF1:
       ld     a,e                 ; Is X_PAT_POS < 0?
       bit    7,a
       jr     nz,XP_NEG
       cp     020H                ; Is X_PAT_POS < 32?
       ret    nc                  ; If not, return
XP_NEG:
       add    a,c                 ; A := X_PAT_POS + X_EXTENT
```

```
       bit    7,a                ; Is A Neg?
       ret    nz                 ; If Yes, return
       or     a                  ; Is A = 0?
       ret    z                  ; If Yes, return
X_IN_BOUNDS:
;      IF     [.E,IS,MINUS]; IF x_PAT_POS < 0, Frame bleeding on from left
       bit    7,e
       jr     z,ELSE_8
       ld     a,c                ; Calculate amount of frame on screen
       add    a,e                ; A := X_EXTENT + X_PAT_POS
       push   de
       cp     021H               ; If number of names > 32
       jr     c,LT33
       ld     a,020H       ; Then number of names := 32
LT33:
       ld     e,a
       ld     d,000H
       push   de                 ; Get count into IY
       pop    iy
       pop    de                 ; Restore DE
       ld     a,e                ; A := X_PAT_POS
       exx                       ; Now adjust starting points in frame list and
                                 ; name table
       push   bc                 ; Save X and Y extent
       neg                       ; 2's compliment of X_PAT_POS
       ld     c,a
       ld     b,000H
       add    hl,bc              ; Add displacement to frame pointer
       ex     de,hl
       add    hl,bc              ; Add displacement to name table pointer
       ex     de,hl
       pop    bc
       exx
       jr     END_IF_9
;
ELSE_8:
PF2:
       ld     a,e                ; Is X_PAT_POS + X_EXTENT > 31
       add    a,c
       cp     01FH
       jr     z,ELSE_9
       jr     c,ELSE_9
       ld     a,020H       ; Substract X_PAT_POS from 31
       sub    e
       push   de
       ld     e,a                ; Get this number into IY
       ld     d,000H
       push   de
       pop    iy
       pop    de
       jr     END_IF_9
;      ELSE                      ; Both ends of frame within pattern plane
ELSE_9:
PF3:
       push   bc
       ld     b,000H
       push   bc
       pop    iy
       pop    bc
END_IF_9:
END_IF_8:
       ld     e,000H
;      REPEAT             ; Y_EXTENT-1 times
```

```
RPT_1:
PF4:
       ld     a,d                 ; Get Y_PAT_POS
       add    a,e                 ; Add Y
;      IF     [.A,IS,PLUS]
       bit    7,a
       jr     nz,END_IF_10
;      IF     [.A,LE,23]          ; Is 0 <= Y_PAT_POS + Y <= 23
       cp     018H
       jr     nc,END_IF_10
       push   bc
       push   de
       exx
       push   bc
       push   de
       push   hl
       push   iy
       ld     a,002H        ; Code for pattern name table added 4/20
       call   PUT_VRAM_
       pop    iy
       pop    hl
       pop    de
       pop    bc
       exx
       pop    de
       pop    bc
END_IF_10:
       exx
       push   bc
       ld     b,000H        ; Increment pointer into frame by X_EXTENT
       add    hl,bc
       ex     de,hl
       ld     bc,00020H           ; Increment offset by 32
       add    hl,bc
       ex     de,hl
       pop    bc
       exx
       inc    e
;      UNTIL [.E,EQ,.B]           ; Until Y = Y_EXTENT
       ld     a,e
       cp     b
       jr     nz,RPT_1
       ret
; ************** GET_BKGRND ************************************************
; Description:  This routine gets the names from the name table which
;               constitute the background in which an object is to be moved
;               at X_PAT_POS, Y_PAT_POS.
; Input:        HL = Location in CPU RAM to which the names are moved
;               E = X_PAT_POS [Left hand column]
;               D = Y_PAT_POS [Top row of pattern]
;               B,C = Y_EXTENT and X_EXTENT of pattern
; ***************************************************************************
GET_BKGRND:
       call   CALC_OFFSET         ; Offset into name table of position of
       push   bc                  ;        upper left   hand pattern
       ld     b,000H        ; Get X_EXTENT into IY
       push   bc                  ;    number of names per row
       pop    iy
       pop    bc
;      REPEAT              ; Y_EXTENT-1 times
RPT_2:
       push   bc
       push   de
```

```
       push   hl
       push   iy
       ld     a,002H       ; Table code for pattern name table
       call   GET_VRAM_
       pop    iy
       pop    hl
       pop    de
       pop    bc
       push   bc
       ld     b,000H       ; BC := X_EXTENT
       add    hl,bc              ; Point HL to beginning of next row
       ld     bc,00020H
       ex     de,hl
       add    hl,bc              ; Increment offset by 32
       ex     de,hl
       pop    bc
       dec    b
;      UNTIL [.B,EQ,0]
       jr     nz,RPT_2
       ret
; ************** CALC_OFFSET ************************************************
; Description:  This routine calculates the proper offset into the name table
;               for the pattern position given by X_PAT_POS, Y_PAT_POS.  The
;               formula used is: offset = 32* Y_PAT_POS + X_PAT_POS
; Input:        D,E = Y_PAT_POS, X_PAT_POS
; Output:       DE = Offset
; ***************************************************************************
CALC_OFFSET:
       push   hl
;      IF     [.D,IS,MINUS]; EXTEND SIGN
       bit    7,d
       jr     z,ELSE_11
       ld     h,0FFH
       jr     END_IF_11
;
ELSE_11:
       ld     h,000H
END_IF_11:
       ld     l,d                ; Offset = 32*Y_PAT_POS + X_PAT_POS
       add    hl,hl              ; HL=2*Y_PAT_POS
       add    hl,hl              ;    4*   "
       add    hl,hl              ;    8*   "
       add    hl,hl              ;   16*   "
       add    hl,hl              ;   32*   "
;      IF     [.E,IS,MINUS]; Extend sign
       bit    7,e
       jr     z,ELSE_12
       ld     d,0FFH
       jr     END_IF_12
;
ELSE_12:
       ld     d,000H
END_IF_12:
       add    hl,de              ; HL := 32*Y_PAT_POS + X_PAT_POS
       ex     de,hl              ; DE := 32*Y_PAT_POS + X_PAT_POS
       pop    hl
       ret
; ******************* EXTERNALS ******************
; EXT  GET_VRAM_
; EXT  PUT_VRAM_
; EXT  VRAM_READ
; EXT  VRAM_WRITE
; EXT  WORK_BUFFER
```

PUT_SPRITE RTN

```
;*****************************************************************************
;
;DESCRIPTION: This module contains code for the PUT1SPRITE and PUT0SPRITE
;             routines. These routines turn out to be essentially the same
;             code with two slihntly different entry points
;
;INPUT:       IX = Address of the sprite object
;
; The format for sprite objects is
;
; SPRITE_OBJECT = RECORD
;   GRAPHICS:^SPRITE_GRAPHICS
;   STATUS:^SPRITE_STATUS
;   SPRITE_INDEX:BYTE            (SPRITE_NAME_TABLE index of this sprite)
; END SPRITE_OBJECT
;
; SPRITE_GRAPHICS = RECORD
;   OBJECT_TYPE:BYTE
;   FIRST_GEN_NAME:BYTE                 (Name of the 1st sprite generators)
;   PTRN_POINTER:^PATTERN_GENERATOR     (Pointer to ROM'ed generators)
;   NUMGEN:BYTE                         (Number of ROM'ed generators)
;   FRAME_TABLE_PTR:^ARRAY[0..nn] of FRAME (table of animation frames)
; END SPRITE_ROM_GRAPHICS
;
; SPRITE_STATUS = RECORD
;   FRAME:BYTE                  (Current animation frame)
;   X_LOCATION:INTEGER
;   Y_LOCATION:INTEGER
;   NEXT_GEN:BYTE               (Index of free space in generator table)
; END SPRITE_STATUS
;
; FRAME = RECORD
;   COLOR:BYTE             (Sprite's color for this frame)
;   SHAPE:BYTE             (This frame's offset from name from FIRST_GEN_NAME)
; END FRAME
;
; SPRITE = RECORD
;   Y:BYTE
;   X:BYTE
;   NAME:BYTE
;   COLOR_AND_TAG:BYTE
; END SPRITE
;
;************************* DICTIONARY ***************************************
;
; #External
; EXT  WORK_BUFFER
; WORK_BUFFER is a pointer in cartridge ROM, located at 8006h, to the
; free buffer area to be used by the graphics routines.
;
; #Defines
;
; SPRITE_PTR  EQU    IY
; SPRITE_PTR is a pointer to the new sprite name table entry being
; build by this routine.
;
; THIS_SPRITE EQU    IX
; THIS_SPRITE is a pointer to the sprite object being put.
;
```

```
GRAPHICS            EQU    0
STATUS       EQU    2
SPRITE_INDEX EQU    4
; Field offsets for SPRITE_OBJECT records.
OBJECT_TYPE         EQU    0
FIRST_GEN_NAME      EQU    1
PTRN_POINTER EQU    2
NUMGEN       EQU    4
FRAME_TABLE_PTR     EQU    5
; Field offsets for SPRITE_GRAPHICS records.
FRAME               EQU    0
X_LOCATION          EQU    1
Y_LOACTION          EQU    3
NEXT_GEN            EQU    5
; Field offsets for SPRITE_STATUS records.
COLOR               EQU    0
SHAPE               EQU    1
; Field offsets for FRAME records.
Y                   EQU    0
X                   EQU    1
NAME                EQU    2
COLOR_AND_TAG EQU    3
; Field offsets for SPRITE records.
;********************** External Procedures ***********************************
;
; #External
; EXT  PUT_VRAM, GET_VRAM
; PUT_VRAMP (TABLE_CODE:BYTE;START_INDEX,SLICE;BYTE;
;            VAR DATA:BUFFER;ITEM_COUNT:INTEGER);
; GET_VRAMP (TABLE_CODE:BYTE;START_INDEX,SLICE;BYTE;
;            VAR DATA:BUFFER;ITEM_COUNT:INTEGER);
;
; PUT_VRAM sends a block of data to the table specified by TABLE_CODE.
; The SLICE, START_INDEX, and ITEM_COUNT are table dependant.
; GET_VRAM does the inverse operation.
;
; TABLE_CODE is passed in A
; START_INDEX,SLICE in DE
; DATA buffer address in HL
; ITEM_COUNT passed in IY
;************************ PROCEDURE BODY ************************************
;
; #Global
; GLB  PUT0SPRITE, PUT1SPRITE

; Begin PUT0SPRITE
PUT0SPRITE:
       ld     iy,(WORK_BUFFER)     ; SPRITE_PTR :=[WORK_BUFFER]
; With THIS_SPRITE^,SPRITE_PTR^ DO
; If (STATUS^.X_LOCATION > -8) And (STATUS^.X_LOCATION < 256) And
;    (STATUS^.Y_LOCATION > -8) And (STATUS^.Y_LOCATION < 192) Then
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,X_LOCATION
       add    hl,de                ; [HL] = X_LOCATION
       ld     c,(hl)
       inc    hl
       ld     b,(hl)               ; BC = X_LOCATION
       ld     a,b                  ; Compare BC with -8
       cp     000H
       jr     z,OK__1
       cp     0FFH   ; -1
       jp     nz,DONT_PUT
```

```
       ld     a,c
       cp     0F9H   ; -7
       jp     m,DONT_PUT
OK__1:
       inc    hl                  ; [HL] = Y_LOCATION
       ld     c,(hl)
       inc    hl
       ld     b,(hl)              ; BC = Y_LOCATION
       ld     a,b                 ; Compare BC with -8
       cp     000H
       jr     z,OK__2
       cp     0FFH   ; -1
       jp     nz,DONT_PUT
       ld     a,c
       cp     0F9H   ; -7
       jp     m,DONT_PUT
OK__2:
; If (STATUS^.X_LOCATION < 0) Then
       dec    hl
       dec    hl                  ; [HL] = HI(X_LOCATION)
       ld     a,(hl)              ; Compare with 0
       cp     000H
       jp     z,CONTINUE
; X := BYTE(STATUS^.X_LOCATION) + 8
       dec    hl                  ; [HL] = ^X_LOCATION
       ld     c,(hl)
       inc    hl
       ld     b,(hl)
       ld     hl,00008H
       add    hl,bc
       ld     a,l
       ld     (iy+X),a
; COLOR_AND_TAG := GRAPHICS^.FRAME_TABLE[STATUS^.FRAME].COLOR Or 80h
       ld     l,(ix+GRAPHICS)
       ld     h,(ix+GRAPHICS+1)
       ld     de,FRAME_TABLE_PTR
       add    hl,de               ; [HL] = FRAME_TABLE_PTR
       ex     de,hl
       ld     a,(de)
       ld     l,a
       inc    de
       ld     a,(de)
       ld     h,a                 ; [HL] = FRAME_TABLE_PTR^
       push   hl
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,FRAME
       add    hl,de               ; [HL] = FRAME
       ld     a,(hl)              ; Calculate offset of
       sla    a                   ; COLOR entry
       ld     bc,00000H
       ld     c,a
       pop    hl
       add    hl,bc               ; [HL] = COLOR
       ld     a,(hl)              ; Or in 80h
       or     080H
       ld     (iy+COLOR_AND_TAG),a
       jp     PUT_Y_AND_NAME
; Else
; ********* Continue below
; Begin PUT1SPRITE
PUT1SPRITE:
       ld     iy,(WORK_BUFFER)    ; SPRITE_PTR :=[WORK_BUFFER]
```

```
; With THIS_SPRITE^,SPRITE_PTR^ DO
; If (STATUS^.X_LOCATION > -32) And (STATUS^.X_LOCATION < 256) And
;    (STATUS^.Y_LOCATION > -32) And (STATUS^.Y_LOCATION < 192) Then
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,X_LOCATION
       add    hl,de               ; [HL] = X_LOCATION
       ld     c,(hl)
       inc    hl
       ld     b,(hl)              ; BC = X_LOCATION
       ld     a,b                 ; Compare BC with -32
       cp     000H
       jr     z,OK__3
       cp     0FFH   ; -1
       jp     nz,DONT_PUT
       ld     a,c
       cp     0E1H   ; -31
       jp     m,DONT_PUT
OK__3:
       inc    hl                  ; [HL] = Y_LOCATION
       ld     c,(hl)
       inc    hl
       ld     b,(hl)              ; BC = Y_LOCATION
       ld     a,b                 ; Compare BC with -32
       cp     000H
       jr     z,OK__4
       cp     0FFH   ; -1
       jp     nz,DONT_PUT
       ld     a,c
       cp     0E1H   ; -31
       jp     m,DONT_PUT
OK__4:
; If STATUS^.X_LOCATION < 0 Then
       dec    hl
       dec    hl                  ; [HL] = HI(X_LOCATION)
       ld     a,(hl)              ; Compare with 0
       cp     000H
       jp     z,CONTINUE
; X := BYTE(STATUS^.X_LOCATION) + 32
       dec    hl                  ; [HL] = ^X_LOCATION
       ld     c,(hl)
       inc    hl
       ld     b,(hl)
       ld     hl,00020H    ; 32
       add    hl,bc
       ld     a,l
       ld     (iy+X),a
; COLOR_AND_TAG := GRAPHICS^.FRAM_TABLE[STATUS^.FRAME].COLOR Or 80h
       ld     l,(ix+GRAPHICS)
       ld     h,(ix+GRAPHICS+1)
       ld     de,FRAME_TABLE_PTR
       add    hl,de               ; [HL] = FRAME_TABLE_PTR
       ex     de,hl
       ld     a,(de)
       ld     l,a
       inc    de
       ld     a,(de)
       ld     h,a                 ; [HL] = FRAME_TABLE_PTR^
       push   hl
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,FRAME
       add    hl,de               ; [HL] = FRAME
```

```
       ld     a,(hl)              ; Calculate offset of
       sla    a                   ; COLOR entry
       ld     bc,00000H
       ld     c,a
       pop    hl
       add    hl,bc               ; [HL] = COLOR
       ld     a,(hl)              ; Or in 80h
       or     080H
       ld     (iy+COLOR_AND_TAG),a
       jr     PUT_Y_AND_NAME
; Else
; ********* Continue from here
CONTINUE:
; X := BYTE(STATUS^.X_LOCATION)
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,X_LOCATION
       add    hl,de               ; [HL] = X_LOCATION
       ld     a,(hl)
       ld     (iy+X),a
; COLOR_AND_TAG := GRAPHICS^.FRAME_TABLE[STATUS^.FRAME].COLOR
       ld     l,(ix+GRAPHICS)
       ld     h,(ix+GRAPHICS+1)
       ld     de,FRAME_TABLE_PTR
       add    hl,de               ; [HL] = FRAME_TABLE_PTR
       ex     de,hl
       ld     a,(de)
       ld     l,a
       inc    de
       ld     a,(de)
       ld     h,a                 ; [HL] = FRAME_TABLE_PTR^
       push   hl
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,FRAME
       add    hl,de               ; [HL] = FRAME
       ld     a,(hl)              ; Calculte offset of
       sla    a                   ; COLOR entry
       ld     bc,00000H
       ld     c,a
       pop    hl
       add    hl,bc               ; [HL] = Color
       ld     a,(hl)
       ld     (iy+COLOR_AND_TAG),a
; End if
PUT_Y_AND_NAME:
; Y := BYTE(STATUS^.Y_LOCATION)
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,Y_LOCATION
       add    hl,de               ; [HL] = Y_LOCATION
       ld     a,(hl)
       ld     (iy+Y),a
; NAME := GRAPHICS^.FRAME_TABLE[STATUS^.FRAME].SHAPE
;         + GRAPHICS^.FIRST_GEN_NAME
       ld     l,(ix+GRAPHICS)
       ld     h,(ix+GRAPHICS+1)
       ld     de,FRAME_TABLE_PTR
       add    hl,de               ; [HL] = FRAME_TABLE_PTR
       ex     de,hl
       ld     a,(de)
       ld     l,a
       inc    de
```

```
       ld     a,(de)
       ld     h,a                 ; [HL] = FRAME_TABLE_PTR^
       push   hl
       ld     l,(ix+STATUS)
       ld     h,(ix+STATUS+1)
       ld     de,FRAME
       add    hl,de               ; [HL] = FRAME
       ld     a,(hl)              ; Calculate offset of
       sla    a                   ; SHAPE entry
       ld     bc,00000H
       ld     c,a
       pop    hl
       add    hl,bc
       inc    hl                  ; [HL] = SHAPE
       ld     a,(hl)
       ld     l,(ix+GRAPHICS)
       ld     h,(ix+GRAPHICS+1)
       ld     de,FIRST_GEN_NAME
       add    hl,de               ; [HL] = FIRST_GEN_NAME
       add    a,(hl)
       ld     (iy+NAME),a
; PUT_VRAM (0,THIS_SPRITE^.SPRITE_INDEX,SPRITE_PTR,1)
       xor    a
       ld     d,000H
       ld     e,(ix+SPRITE_INDEX)
       push   iy
       pop    hl
       ld     iy,00001H           ; Count of one item
       call   PUT_VRAM
       jr     EXIT_PUT_SPR
; Else
DONT_PUT:    ; Put sprite off the screen by setting its X and early cloack
; GET_VRAM (0,THIS_SPRITE^.SPRITE_INDEX,SPRITE_PTR,1)
       push   iy                  ; Save index regs.
       push   ix
       push   iy
       push   iy
       xor    a
       ld     d,000H
       ld     e,(ix+SPRITE_INDEX)
       pop    hl
       ld     iy,00001H           ; Count of one item
       call   GET_VRAM
; SPRITE_PTR.X :=0
       ld     a,000H
       pop    iy
       ld     (iy+X),a
; SPRITE_PTR.COLOR_AND_TAG := 80h
       ld     a,080H
       ld     (iy+COLOR_AND_TAG),a
; PUT_VRAM (0,THIS_PRITE^.SPRITE_INDEX,SPRITE_PTR,1)
       xor    a
       ld     d,000H
       pop    ix
       ld     e,(ix+SPRITE_INDEX)
       pop    hl
       ld     iy,00001H           ; Count of one item
       call   PUT_VRAM
; End if
; End PUT0SPRITE,PUT1SPRITE
EXIT_PUT_SPR:
       ret
```

PUT MOBILE

```
;*************** MODIFIED VERSION TO RUN ON HP ASSEMBLER **********************
;                                                                      4/16/82
;                                                                      13:50:00
;****************** PUT_MOBILE ************************************************
;
;DESCRIPTION: This procedure places a mobile object on the pattern plane
;             at the X,Y pixel localisation specified in that objet's RAM
;             status AREA.
;
;             A buffer area of 204 bytes (graphics mode II) or 141 bytes
;              (gaphics mode I) is required for forming the new generators
;             representing the object on it's background_ the procedure
;             uses RAM starting at (F_BUF_SPACE) for this buffer
;
;INPUT:       IX = Address of object to be processed
;             HL = Address of object's graphics tables in ROM
;             B = Selector for methode of combining object generators
;                 with background generators
;
;-------------------------------------------------------------------------------
;METHODE OF COMBINING OBJECT GENERATORS:
;
;             1 = Object pattern gens ored with background pattern gens
;                 color1 of background changed to mobile object's color
;                 if corresponding pattern byte not zero
;
;             2 = Replace background pattern gens with object pattern gens
;                 treat color same as #1
;
;             3 = Same as #1 except color0 changed to transparent
;
;             4 = Same as #2 except color0 changed to transparent
;
;*******************************************************************************
;}
;      EXT    READ_VRAM,WRITE_VRAM,WORK_BUFFER,GET_VRAM,PUT_VRAM
;      EXT    PX_TO_PTRN_POS,GET_BKGRND,VDP_MODE_WORD,PUTFRAME
;      GLB    PUT_MOBILE
;
; #Defines
; The following are offsets from the start of the free buffer area
; These locations used to store variables and pattern and color data
YDISP         EQU    0      ;Y Displacement
XDISP         EQU    1      ;X Displacement
COLR          EQU    2      ;Color
FLAGS         EQU    3      ;BITS 0,1=Selector#, BIT X = Graphics Mode (I/II)
FRM           EQU    4      ;Frame to be Displayed
F_GEN         EQU    5      ;Name of 1st generator in object's gen table
YP_OS         EQU    7      ;Y_PAT_POS of OLD_SCREEN
XP_OS         EQU    6      ;X_PAT_POS of OLD_SCREEN
YP_BK         EQU    18     ;Y_PAT_POS of BACKGROUND   ;12h
XP_BK         EQU    17     ;X_PAT_POS of BACKGROUND   ;11h
BK_PTN EQU    28     ;Start of background pattern generators ; 1Ch
OBJ_PTN       EQU    100    ;Start of object's pattern generators   ; 64h
BK_CLR EQU    132    ;Start of background color generators   ; 84h

PUT_MOBILE:
       ld     iy,(WORK_BUFFER)    ; Get start of free buffer area
       ld     a,(VDP_MODE_WORD)   ; Find out which graphics mode we are in
```

```
       bit    1,a
       jr     nz,ELSE1            ; Then Mode I
       res    7,b
       jr     END1
;
ELSE1:                     ; Else Mode II
       set    7,b
END1:
       ld     (iy+FLAGS),b        ; Save Selector
       push   hl                  ; Save graphics address
       ld     h,(ix+003H)         ; HL := ADDR_ of status
       ld     l,(ix+002H)
       ld     a,(hl)              ; Get frame #
       ld     (iy+FRM),a          ; And save
       xor    080H                ; Complement table_in_use flag
       ld     (hl),a              ; Save back in status area
       inc    hl                  ; point to X_LOCATION
       ld     e,(hl)              ; E := Low X_LOCATION
       ld     a,e
       and    007H                ; A := #Pixels to right of pattern boundary
       neg
       add    a,008H              ; Amount to shift pattern left
; from next pat boundary
       ld     (iy+XDISP),a        ; Save
       inc    hl
       ld     d,(hl)              ; DE := X_LOCATION
       call   PX_TO_PTRN_POS      ; Calculate X_PAT_POS of background
       ld     (iy+XP_BK),e        ; And Save
       inc    hl                  ; Point to Y_LOCATION
       ld     e,(hl)              ; E := Low Y_LOCATION
       ld     a,e
       and    007H                ; A := #Pixels to right of pattern boundary
       ld     (iy+YDISP),a ; Save
       inc    hl
       ld     d,(hl)              ; DE := Y_LOCATION
       call   PX_TO_PTRN_POS      ; Calculate Y_PAT_POS
       ld     (iy+YP_BK),e        ; And Save
; Now get the nine names that constitute the background on which
; the mobile object will be superimposed
PM1:
       ld     hl,(WORK_BUFFER)
       ld     de,YP_BK+1          ; Point to space for background names
       add    hl,de
       ld     d,(iy+YP_BK)        ; D := Y_PAT_POS
       ld     e,(iy+XP_BK)        ; E := X_PAT_POS
       ld     bc,00303H           ; B := Y_EXTENT, C := X_EXTENT
       call   GET_BKGRND          ; Get background names
; Read old screen into buffer and get COLOR and 1st GEN_NAME
PM2:
       ld     d,(ix+005H)         ; DE := OLD_SCREEN address
       ld     e,(ix+004H)
       ld     a,(ix+006H)         ; Get first GEN_NAME
       pop    ix                  ; IX := Address of graphics
       ld     iy,(WORK_BUFFER)
       ld     (iy+F_GEN),a        ; Save in buffer
       push   de                  ; Save OLD_SCREEN address
       ld     hl,(WORK_BUFFER)    ; HL := Addr of start of buffer
       ld     bc,XP_OS            ; Space to move OLD_SCREEN to
       add    hl,bc
       ld     bc,0000BH           ; Get 9 names from VRAM
                                  ; Then OLD_SCREEN is in VRAM
       ld     a,d
       cp     070H
```

```
       jr     nc,ELSE2
       call   READ_VRAM
       jr     END2
; ELSE                      ; OLD_SCREEN in CPU RAM
ELSE2:
       ex     de,hl
       ldir
END2:

; At this point, IX = GRAPHICS, [SP] = OLD_SCREEN
; BACKGROUND pattern position and names starting at YP_BK
; OLD_SCREEN pattern position and names starting at YP_OS
; Find all names in background which
;      belong to this object's pattern generator
; And replace with name from OLD_SCREEN which
;      corresponds to that pattern position
PM3:
       ld     hl,(WORK_BUFFER)    ; HL := BUFFER BASE
       ld     de,YP_BK+1
       add    hl,de               ; Point to 1st of background names
       exx
       ld     de,(WORK_BUFFER)    ; DE' := BUFFER BASE
       ld     hl,YP_OS+1
       add    hl,de
       ex     de,hl               ; Points to 1st of OLD_SCREEN names
       exx
       ld     iy,(WORK_BUFFER)
       ld     c,(iy+F_GEN)        ; C := FIRST_GEN_NAME
       ld     b,009H
DLP1:
       ld     a,(hl)              ; Get a name
       sub    c                   ; Substract FIRST_GEN_NAME
; If [_A,LT,18]                   ; Then name falls in range of names for object
       cp     012H
       jr     nc,END3
; If [_A,GE,9]                    ; Then sub 9 to find correct
       cp     009H
       jr     c,END4
       sub    009H                ; Position in OLD_SCREEN
END4:  ; ENDIF
       exx
       ld     l,a                 ; Form a pointer into OLD_SCREEN names
       ld     h,000H
       add    hl,de
       ld     a,(hl)              ; Get OLD_SCREEN NAME
       exx
       ld     (hl),a              ; Replace background name with OLD_SCREEN name
END3:
       inc    hl                  ; Point to next name in background
       djnz   DLP1

; Now new version of background names will not contain any names of this object
; Replace previous version of OLD_SCREEN with this new background
PM4:
       pop    de                  ; DE := OLD_SCREEN address
       ld     hl,(WORK_BUFFER)    ; HL := BUFFER base
       ld     bc,XP_BK
       add    hl,bc               ; HL points to background position and names
       ld     bc,0000BH           ; Number of bytes to move
;      if [_D,LT,70H]             ; Then move data to VRAM
       ld     a,d
       cp     070H
       jr     nc,ELSE5
```

```
       call   WRITE_VRAM
       jr     END5
;      Else
ELSE5:                      ; Move to CPU RAM
       Ldir
;      ENDIF
END5:
       push   ix                   ; Save graphics pointer
PM5:
       ld     de,(WORK_BUFFER)     ; DE := BUFFER base
       ld     hl,YP_BK+1           ; Displacement to beginning of BKGND names
       add    hl,de
       ex     de,hl                ; DE point to first background name
       ld     bc,BK_PTN-8          ; HL points to background PAT GET storage area
-8
       add    hl,bc                ;  will be incremented before 1st use
       ld     b,009H
DLP2:
       ld     a,(de)               ; Get name
       inc    de                   ; point to next name
       push   de                   ; Save name pointer
       ld     de,00008H            ; increment BUFFER pointer
       add    hl,de
       push   hl                   ; Save BUFFER pointer
       ld     e,a                  ; Index into pattern table
       ld     d,000H
       ld     c,a                  ; Save pattern name in C
       push   bc                   ; Save counter
       ld     a,009H               ; ADD [b-9]/3 to YP_BK to find out
       sub    b                    ;  which third of tables GENs are in
       ld     b,000H
PM52:
       sub    003H
       jr     c,PM51
       inc    b
       jr     PM52
;
PM51:
       ld     a,b
       ld     iy,(WORK_BUFFER)
       add    a,(iy+YP_BK)
       bit    7,(iy+FLAGS)         ; Test graphics mode
       ld     iy,00001H            ; Number of elements read
PM6:   ; IF [PSW,IS,ZERO]          ; Then MODE I
       jr     nz,ELSE6
       ld     a,003H               ; Code for pattern generator table
       call   GET_VRAM
       pop    bc                   ; Get count and name
       ld     hl,(WORK_BUFFER)
       push   bc
       ld     de,BK_CLR            ; Displacement to COLOR GEN AREA
       add    hl,de                ; Point to it
       ld     e,c                  ; Pattern name
       srl    e                    ; Divide name by 8
       srl    e
       srl    e
       ld     d,000H
       ld     a,009H               ; Calc position in buffer to move GEN to
       sub    b
       ld     c,a
       ld     b,000H
       add    hl,bc
       ld     iy,00001H
```

```
       ld     a,004H              ; Color generator table code
       call   GET_VRAM
       jr     END6
;
ELSE6: ; ELSE       ; MUST BE MODE II
       sra    a                   ; Divide Y_POS by 8
       sra    a
       sra    a                   ; A:= Third of table, 0=1st, 1=2nd, 2=3rd
PM7:   ; IF [_A,LT,3]              ; If A>2, Then Y_POS > 23 And Therefore off
screen
       cp     003H
       jr     nc,END7
       ld     d,a                 ; DE := 256*A + NAME
       push   de                  ; Save it
       push   hl
       ld     a,003H              ; Pattern generator table code
       call   GET_VRAM
       pop    hl                  ; HL := Pattern buffer address
       ld     de,BK_CLR-BK_PTN    ; Displacement between pattern and color
buffers
       add    hl,de               ; HL := Pointer to color buffer
       pop    de
       ld     iy,00001H
       ld     a,004H              ; Code for color table
       call   GET_VRAM
END7:
END6:
       pop    bc                  ; Restore registers
       pop    hl
       pop    de
       djnz   DLP2

; Now the pattern and color generators are in their respective buffers
; So get the four generators for this frame of the object

       pop    ix                  ; Restore graphics pointer
PM8:
       exx
       ld     d,(ix+003H)
       ld     e,(ix+002H)         ; DE' := NEW_GEN
       ld     b,(ix+005H)
       ld     c,(ix+004H)         ; BC' := PTRN_POINTER
       exx
       push   ix
       pop    hl                  ; HL := Address of graphics
       ld     iy,(WORK_BUFFER)
       ld     a,(iy+FRM)          ; A := FRM #
       add    a,a                 ; X2
       ld     c,a
       ld     b,000H
       ld     de,00006H           ; Frame pointers start at this offset
       add    hl,de
       add    hl,bc               ; HL := Points to FRAME_POINTER for frame
       ld     e,(hl)
       inc    hl
       ld     d,(hl)              ; DE := Address of frame names
       ld     hl,(WORK_BUFFER)    ; HL := Buffer base address
       ld     bc,OBJ_PTN+24       ; Use location for last GEN to store names
       add    hl,bc
       push   hl                  ; Save for later use
       push   bc                  ; Save offset
       ld     bc,00005H
;      IF [_D,LT,70H]              ; Then names are in VRAM
```

```
        ld      a,d
        cp      070H
        jr      nc,ELSE8
        call    READ_VRAM           ; Get the 4 names
        jr      END8
;       ELSE                        ; Names in CPU memory space
ELSE8:
        ex      de,hl
        ldir
END8:
        ld      iy,(WORK_BUFFER)    ; Get color byte
        pop     bc                  ; Offset to 1st name
        add     iy,bc
        ld      a,(iy+004H)         ; A := Color Byte
        ld      iy,(WORK_BUFFER)
        ld      (iy+COLR),a         ; COLOR := Color Byte
PM9:
        pop     de                  ; DE := Address of 1st name in buffer
        ld      hl,(WORK_BUFFER)    ; HL := buffer base address
        ld      bc,OBJ_PTN          ; Start of object's pattern buffer
        add     hl,bc
        ld      b,004H              ; Get 4 patterns corresponding to the 4 names
DLP4:
        ld      a,(de)              ; Get name
        cp      (ix+001H)           ; Compare to NUMGEN
        push    de                  ; Save pointer to names
;       IF [PSW,IS,CARRY]           ; The name < numgen, therefore
        jr      nc,ELSE9
        exx                         ; Pattern part of graphics tables
        add     a,a
        add     a,a
        add     a,a                 ; A := 8*Name
        ld      l,a
        ld      h,000H
        add     hl,bc               ; HL := Pointer to Pattern
        push    hl
        exx
        pop     de                  ; DE :=   "          "
        ex      de,hl
        push    bc
        ld      bc,00008H           ; Number of bytes to move
        ldir
        pop     bc
        ex      de,hl
        jr      END9
;       ELSE                ; Name => Numgen, therefore not part of ROMed Gens
ELSE9:
        sub     (ix+001H)           ; Substract Numgen from NAME
        exx
        add     a,a
        add     a,a
        add     a,a                 ; A := 8*NAME
        ld      l,a
        ld      h,000H              ; HL := Pointer to pattern
        add     hl,de
        push    hl
        exx
        pop     de
;       IF [_D,LT,70H]              ; Then pattern in VRAM
PM10:
        ld      a,d
        cp      070H
        jr      nc,ELSE10
```

```
        push   bc
        push   hl                  ; Save buffer address
        push   de                  ; Save generator address
        ld     bc,00008H           ; Number bytes to move
        call   READ_VRAM
        ld     bc,00008H           ; Increment pointers by 8
        pop    hl
        add    hl,bc               ; Gen addr := Gen addr + 8
        ex     de,hl
        pop    hl
        add    hl,bc               ; Bug addr := Buf addr + 8
        pop    bc
        jr     END10
;       ELSE                       ; Pattern in CPU RAM
ELSE10:
        ex     de,hl
        push   bc
        ld     bc,00008H
        ldir
        pop    bc
        ex     de,hl
END10:
END9:
        pop    de
        inc    de
        djnz   DLP4

; Combine object pattern  generators with background
PM11:
        ld     iy,(WORK_BUFFER)
        ld     de,(WORK_BUFFER)    ; DE := Buffer base addr_
        ld     hl,BK_PTN
        add    hl,de               ; HL points to start of background generators
        ld     c,(iy+YDISP)  ; BC :=
        ld     b,000H       ; Displacement of object below pattern generators
        add    hl,bc               ; HL points to 1st row in background
;     to be overlayed
        push   hl
        pop    ix                  ; IX := pointer to row in pattern buffer
        ld     hl,OBJ_PTN
        add    hl,de               ; HL points to 1st byte in object's pattern
gen
        push   hl
        ld     a,010H              ; Use A' as loop counter
        ex     af,af'
COMBINE_LOOP:
        pop    hl                  ; Point to object pattern byte
        ld     d,(hl)              ; Get pattern byte
        inc    hl
        push   hl
        ld     bc,0000FH
        add    hl,bc               ; Point to adjacent pattern byte
        ld     e,(hl)
        ex     de,hl               ; HL has 16 bit row of object's pattern
        ld     b,(iy+XDISP)        ; Shift left pattern by this amount
        xor    a                   ; Clear A
SHFLP:
        dec    b
        jp     m,SHFEX
        add    hl,hl
        rla
        jr     SHFLP
;
```

```
SHFEX:
       ld     e,a                 ; Save left byte in E
       call   COM_PAT_COL
       ld     a,(iy+YDISP)        ; Increment YDISP
       inc    a
       ld     (iy+YDISP),a
       cp     008H
       jr     z,IF11
       cp     010H
       jr     nz,END11
IF11:
       ld     bc,00010H
       add    ix,bc               ; Beginning of next row
END11:
       inc    ix                  ; Point to next gen byte
       ex     af,af'
       dec    a                   ; Decrement loop counter
       jr     z,C_LP_EXIT
       ex     af,af'
       jr     COMBINE_LOOP
;
C_LP_EXIT:
       pop    hl                  ; POP Pointer off stack
       bit    7,(iy+FLAGS)        ; Test for mode 1
;      IF [PSW,IS,ZERO]           ; Then update mode 1 color Gens
       jr     nz,END12
       ld     hl,(WORK_BUFFER)
       ld     bc,BK_CLR           ; Offset for color buffer
       add    hl,bc               ; HL points to start of color buffer
       ld     d,(iy+COLR)         ; Get object color
       bit    1,(iy+FLAGS)        ; Color0 = background or transparent ?
;      IF [PSW,IS,ZERO]           ; Then use background color0
       jr     nz,ELSE13
       ld     c,00FH              ; Mask for color0 of background
       jr     END13
;      ELSE
ELSE13:
       ld     c,000H              ; Mask replace color0 with transparent
;      ENDIF
END13:
       ld     b,009H              ; Change all 9 color bytes
DLP5:
       ld     a,(hl)              ; Get background color gen
       and    c                   ; Mask out color1
       or     d                   ; Add object color1
       ld     (hl),a              ; Update color generator
       inc    hl                  ; Point to next color gen
       djnz   DLP5
END12:

; Decide which part of object's pattern and color tables to use
       ld     a,(iy+F_GEN)        ; Get name of 1st generator in object's table
       bit    7,(iy+FRM)          ; Test which part of table to use
;      IF [PSW,IS,NZERO]          ; Then use upper half of tables
       jr     z,END14
       add    a,009H
;      ENDIF
END14:
; Change names in background buffer to those of the object's patterns
; This will then constitute a new frame displaying the object at pattern plane
; location [YP_BK],[XP_BK]
       ld     c,a                 ; Save index
       ld     hl,(WORK_BUFFER)    ; HL := Buffer base
```

```
       ld     de,YP_BK+1          ; Point to 1st background name
       add    hl,de
       ld     b,009H
DLP6:
       ld     (hl),a
       inc    a
       inc    hl
       djnz   DLP6
; Move newly formed generators to object's pattern and color gen tables
PM12:
       bit    7,(iy+FLAGS)        ; Test which mode
;      IF [PSW,IS,ZERO]           ; Then mode 1
       jr     nz,ELSE04
       ld     e,c
       ld     d,000H              ; DE := Index into pattern generator table
       ld     hl,(WORK_BUFFER)
       ld     bc,BK_PTN
       add    hl,bc               ; HL points to 1st generator
       ld     iy,00009H           ; 9 elements to send
       ld     a,003H              ; Code for pattern generator table
       call   PUT_VRAM
; Set up pointers to object's pattern names and to color gen bytes
       ld     iy,(WORK_BUFFER)
       ld     hl,(WORK_BUFFER)    ; HL := buffer base
       ld     bc,BK_CLR           ; Offset to color buffer
       add    hl,bc               ; HL points to color buffer
       ld     ix,(WORK_BUFFER)    ; IX := buffer base
       ld     bc,YP_BK+1          ; Offset to 1st name for object
       add    ix,bc               ; IX points to 1st name
       ld     b,009H
; Repeat
RPT1:                             ; Move colors to color generator table
       ld     a,(ix+000H)         ; Get name
       inc    ix                  ; Point to next name
       srl    a                   ; Divide by 8
       srl    a
       srl    a
       ld     e,a
       ld     d,000H              ; DE := Offset into color gen table
       push   bc                  ; Save counter
       ld     a,009H              ; Test whether this pattern position is
       sub    b                   ;    on the screen or not (i.e. is YP.BK
       ld     b,000H              ;    + (B-9)/3 between 0 and 23, and is
DVLP:
       cp     003H                ;    XP.BAK + (B-9)/3 mod 3 between 0 and 31)
       jr     c,DVEX
       sub    003H
       inc    b
       jr     DVLP
;
DVEX:
       add    a,(iy+XP_BK)        ; A := X_POS + (B-9)/3 mod 3
       cp     020H
       jr     nc,NOTOS
       ld     a,b
       add    a,(iy+YP_BK)        ; A := Y_POS + (B-9)/3
       cp     018H                ; Is this position on screen 7
       jr     nc,NOTOS
       push   ix                  ;      And pointers
       push   hl
       ld     a,004H              ; Code for color generator table
       ld     iy,00001H           ; Number of items to send
       call   PUT_VRAM
```

```
       pop    hl                  ; Restore pointers
       pop    ix
NOTOS:
       pop    bc                  ;      And counter
       inc    hl                  ; Increment pointers
       inc    de
       dec    b                   ; Decrement counter
       ld     a,b
       cp     000H
       jr     nz,RPT1
;      UNTIL [_B,EQ,0]
       ld     iy,(WORK_BUFFER)    ; Restore buffer addr
       jr     END04
;
ELSE04:
PM13:
       ld     b,000H
RPT2:
       push   bc                  ; Save counter and index
       ld     a,c                 ; Add 3xcounter to index,
       add    a,b                 ;    where next gens to be moved
       add    a,b
       add    a,b
       ld     c,a                 ; C := Index to table location for next 3 gens
       ld     hl,00000H           ; Calculate offsets from start of pattern
       ld     de,00018H           ;   and color generator tables
       ld     a,b
AD_LP:
       dec    a
       jp     m,AD_EXIT
       add    hl,de
       jr     AD_LP
; HL := Offset from start of ptrn and color buffers
AD_EXIT:
       ld     iy,(WORK_BUFFER)
       ld     a,(iy+YP_BK)  ; Get Y_PAT_POS to see which 3rd of tables to use
       add    a,b           ; each group of 3 in next pattern plane row
;      IF [_A,LT,24]       ;'A' must countain valid Y_PAT_POS [0-23]
       cp     018H
       jr     nc,END15
       srl    a                   ; This number / 8 indicates which 1/3 of
       srl    a                   ;     table to use
       srl    a
       ld     d,a
       ld     e,c                 ; DE := Index into pattern and color tables
       push   de                  ; Save index
       ld     bc,BK_PTN           ; Form pointer to generators in HL
       add    hl,bc
       ld     bc,(WORK_BUFFER)    ; Get buffer base addr
       add    hl,bc
       push   hl                  ; Save this pointer
       ld     iy,00003H           ; Number of elements to move
       ld     a,003H              ; Pattern generator table code
       call   PUT_VRAM
       pop    hl                  ; Get pointer back
       ld     de,BK_CLR-BK_PTN    ; Offset between buffers
       add    hl,de         ; HL points to start of next 3 color generators
       pop    de                  ; Get index into gen tables
       ld     iy,00003H
       ld     a,004H
       call   PUT_VRAM
END15:
       pop    bc                  ; Restore counter and index
```

```
       inc    b
       ld     a,b
       cp     003H
       jr     nz,RPT2
;      IF [_B,EQ,3] ; Repeat 3 times
END04:

; Restore OLD_SCREEN if it's Y_PAT_POS and X_PAT_POS differs from the
; Y_PAT_POS and X_PAT_POS for the object
PM14:
       ld     iy,(WORK_BUFFER)
       ld     b,(iy+XP_OS)        ; Test for valid OLD_SCREEN data
;      IF [_B,NE,80H]             ; Then there is valid data
       ld     a,b
       cp     080H
       jr     z,END16
       ld     c,(iy+YP_OS)        ; Test if OS position same as current position
       ld     h,(iy+XP_BK)
       ld     l,(iy+YP_BK)
       or     a                   ; Clear the carry
       sbc    hl,bc               ; Is there any difference?
;      IF     [PSW,IS,NZERO]      ; Then position has changed
       jr     z,END17
       ld     hl,(WORK_BUFFER)    ; Get buffer base
       ld     de,YP_OS+1
       add    hl,de               ; Point to OLD_SCREEN names
       ld     e,(iy+XP_OS)
       ld     d,(iy+YP_OS)        ; DE := X and YPAT_POS
       ld     bc,00303H           ; BC := X and Y EXTENT
       call   PUT_FRAME
END17:
END16:
; Place object on screen
       ld     iy,(WORK_BUFFER)
       ld     hl,(WORK_BUFFER)    ; HL := Buffer base addr
       ld     de,YP_BK+1          ; Point to names for object
       add    hl,de
       ld     e,(iy+XP_BK)
       ld     d,(iy+YP_BK)        ; DE := A and Y PAT_POS
       ld     bc,00303H           ; BC := X and Y EXTENT
       call   PUT_FRAME
; ****************** END OF PUT_MOBILE *****************************
       ret
; Regs A, H and L contain 24 bit pattern to be combined with background
; generators. IX points to the 1st of 3 generator bytes to be combined
; with A, H and L
COM_PAT_COL:
       bit    0,(iy+FLAGS)        ;'OR' gens or replace
;      IF     [PSW,IS,ZERO]       ; Then 'OR'
       jr     nz,ELSE18
       or     (ix+000H)           ; 'OR' left byte with background
       ld     (ix+000H),a         ; Substitute for that generator byte
       ld     a,h                 ; Now do middle byte
       or     (ix+008H)
       ld     (ix+008H),a
       ld     a,l                 ; And now the right hand byte
       or     (ix+010H)
       ld     (ix+010H),a
       jr     END18
;      ELSE                       ; Replace background with non-zero bytes
ELSE18:
       or     a                   ; Is byte non-zero
;      IF     [PSW,IS,NZERO]
```

```
       jr     z,END19
       ld     (ix+000H),a         ; Yes, then replace background with object
END19:
       ld     a,h                 ; Same for middle
       or     a
       jr     z,END20
       ld     (ix+008H),a
END20:
       ld     a,l                 ; Same for right hand byte
       or     a
       jr     z,END21
       ld     (ix+010H),a
END21:
END18:
PM15:
       bit    7,(iy+FLAGS)        ; Find out which graphics mode used
;      IF     [PSW,IS,ZERO]       ; Then mode 2
; [mode 1 colors done after combined loop]
       jr     z,END22
       push   ix                  ; Save background pointer
       ld     bc,BK_CLR-BK_PTN    ; Change IX to point to color generators
       add    ix,bc
       ld     b,(iy+COLR)         ; Get object color
       bit    1,(iy+FLAGS)        ; color0 = background or transparent ?
;      IF     [PSW,IS,ZERO]       ; Then use background color0
       jr     nz,ELSE23
       ld     c,00FH              ; Mask for color0 of background
       jr     END23
;
ELSE23:
       ld     c,000H              ; Mask replace color0 with transparent
END23:
       ld     a,e                 ; Get 1st object's pattern byte
       or     a                   ; Are there any '1' bits?
       jr     z,END24
       ld     a,(ix+000H)         ; Get background color gen
       and    c                   ; Mask out color1
       or     b                   ; Add object color1
       ld     (ix+000H),a         ; Update color generator
END24:
       ld     a,h                 ; Same for middle byte
       or     a
       jr     z,END25
       ld     a,(ix+008H)
       and    c
       or     b
       ld     (ix+008H),a
END25:
       ld     a,l                 ; Same for right hand byte
       or     a
       jr     z,END26
       ld     a,(ix+010H)
       and    c
       or     b
       ld     (ix+010H),a
END26:
       pop    ix                  ; Restore background pointer
END22:
       ret
```

PUT COMPLEX

```
;      .IDENT PUTCOMP
;      .ZOP
;      .EPOP
;      .IF1 ,.INSERT B:SPZ80.ASM
;      .COMMENT }
;                                                                    4/15/82
;                                                                    10:40:00
;****************** PUT_COMPLEX ************************************************
;
;DESCRIPTION: The position and frame number of each of a complex object's
;             component objects is updated.  Then put_object is called for
;             each of the component objects.
;
;INPUT:              IX = Address of object to be processed
;                    HL = Address of object's graphics tables in ROM
;                     B = Selector for methode of combining object generators
;                         with background generators
;                     C = Object type, and number of components
;
;-------------------------------------------------------------------------------
;METHODE OF COMBINING OBJECT GENERATORS:
;
;              1 = Object pattern gens ored with background pattern gens
;                 color1 of background changed to mobile object's color
;                 if corresponding pattern byte not zero
;
;              2 = Replace background pattern gens with object pattern gens
;                 treat color same as #1
;
;              3 = Same as #1 except color0 changed to transparent
;
;              4 = Same as #2 except color0 changed to transparent
;
;*******************************************************************************
;}
;      EXT    PUTOBJ
;      GLB    PUTCOMPLEX

PUTCOMPLEX:
; Update the frame number and the X and Y location in each of the component
; object's status areas
       push   bc                  ; Save selector and component count [COMP_CNT]
       exx                        ; Use primed regs for X_LOC and Y_LOC
       ld     h,(ix+003H)         ; High-byte of status
       ld     l,(ix+002H)         ; Low-byte of status
       ld     a,(hl)        ; A := FRAME
       inc    hl
       ld     c,(hl)        ; BC' := X_LOCATION
       inc    hl
       ld     b,(hl)
       inc    hl
       ld     e,(hl)        ; DE' := Y_LOCATION
       inc    hl
       ld     d,(hl)
       exx
       add    a,a                 ; FRAME := 4*FRAME
       add    a,a
       ld     e,a                 ; from pointer to frame and offset pointers
       ld     d,000H
```

```
        inc     hl                  ; Point to 1st of FRA_OFFSET_PNTR pairs
        add     hl,de               ; Point to FRAME pointer
        ld      c,(hl)          ; BC := FRAME pointer
;         [POINTER TO LIST OF FRAME #'s]
        inc     hl
        ld      b,(hl)
        inc     hl
        ld      e,(hl)          ; DE := Offset pointer [PNTR TO LIST OF OFFSETS]
        inc     hl
        ld      d,(hl)
        ld      h,b
        ld      l,c                 ; HL := FRAME pointer
;
; DE' = Y_LOC, BC' = X_LOC, HL = PNTR to frame list, DE = PNTR to offset list
; IX = Addr of OBJ, [SP] = COMP_CNT & SELECTOR
; FOR N=0 TO COMP_CNT-1: COMP_OBJ[N] FRAME := FRAME#[N] FROM FRAME LIST
;             COMP_OBJ[N] X_LOCATION := CMPLX_OBJ.X_LOCATION + X_OFFSET[N]
;             COMP_OBJ[N] Y_LOCATION := CMPLX_OBJ.Y_LOCATION + Y_OFFSET[N]
;
        pop     bc
        ld      a,c                 ; Get component count into B
        ld      c,b                 ; Save selector in C
        srl     a                   ; Get count into low nibble
        srl     a
        srl     a
        srl     a
        ld      b,a                 ; # := component count
        push    bc                  ; Save counter and selector on stack
        push    ix                  ; Save addr of OBJ
LP1:
        push    hl                  ; Save pntr to frame list
        push    de                  ; Save pntr to offset list
        ld      l,(ix+004H)
        ld      h,(ix+005H)         ; HL := Addr of component OBJ
        inc     ix                  ; Point to next object pointer
        inc     ix
        inc     hl
        inc     hl                  ; HL points to status pointer
        ld      e,(hl)
        inc     hl
        ld      d,(hl)          ; DE := Addr of status for component object
        push    de
        pop     iy                  ; IY := Addr of status for component object
        pop     de                  ; DE := pntr to offset list
        pop     hl                  ; HL := pntr to frame list
        ld      a,(hl)          ; Get frame number
        bit     7,(iy+000H)         ; Preserve bit 7 of frame
        jr      z,TBL0          ; used by mobile objects to indicate which
                                    ; VRAM tables in use.
        set     7,a
TBL0:
        ld      (iy+000H),a         ; Move to components status area
        inc     hl                  ; Point to next frame number
        ld      a,(de)          ; Get X_Offset
        exx
        ld      l,a
        ld      h,000H          ; HL' := X_Offset
        add     hl,bc               ; HL' := X_Offset + X_Location
        ld      (iy+001H),l
        ld      (iy+002H),h         ; Component's X_Location := X_Offset +
X_Location
        exx
        inc     de                  ; Point to Y_Offset
```

```
        ld      a,(de)
        exx
        ld      l,a
        ld      h,000H          ; HL := Y_Offset
        add     hl,de           ; HL := Y_Offset + Y_Location
        ld      (iy+003H),l
        ld      (iy+004H),h     ; Component's Y_Location := Y_Offset +
Y_Location
        exx
        inc     de              ; Point to next offset pair
        djnz    LP1
;
; Call PUT_OBJECT for each of the component objects, pass selector in B
        pop     iy              ; Get object address back
        ld      bc,00004H
        add     iy,bc           ; IY points to pointer to 1st component object
        pop     de              ; DE := Counter and Selector
LP2:
        ld      l,(iy+000H)
        ld      h,(iy+001H)     ; HL := Address of component object
        inc     iy
        inc     iy              ; IY := Points to next component object
pointer
        push    hl
        pop     ix              ; IX := Address of component object
        push    iy              ; Save pointer
        push    de              ; Save counter and selector
        ld      b,e             ; B := Selector
        call    PUTOBJ
        pop     de              ; Get counter and selector
        pop     iy              ; Get address of next component object pointer
        dec     d
        jr      nz,LP2
        ret
```

TIME MANAGER

```
; ; Ken Lagace and Rob Jepson 3/82
; #Global
; GLB  TIMER_TABLE_BASE          ; Is this necessary ??
; GLB  NEXT_TIMER_DATA_BYTE      ; Or this??
;
; #Extern
; EXT  PARAM_ ; Parameter passing routine needed
;             ; for setting up pascal interfaces
;
; #Global
; GLB  INIT_TIMER_
; GLB  INIT_TIMERQ
; GLB  FREE_SIGNAL_
; GLB  FREE_SIGNALQ
; GLB  REQUEST_SIGNAL_
; GLB  REQUEST_SIGNALQ
; GLB  TEST_SIGNAL_
; GLB  TEST_SIGNALQ
; GLB  TIME_MGR_
; GLB  TIME_MGRQ
;
; #Define
DONE   EQU    7
REPEAT EQU    6
FREE   EQU    5
EOT    EQU    4
LONG   EQU    3
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;                                                                              ;
;                 . . . . . .  MODE BIT  . . . . . .                          ;
;          TO CHECK BIT *****  BIT ?  JR Z = JUMP IF BIT IS 0 !                ;
;                                                                              ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;        ;        ;        ;        ;;        ;        ;        ;        ;
;  DONE  ; REPEAT ;  FREE  ;  E O T ;;  LONG  ;   --   ;   --   ;   --   ;
;   7    ;   6    ;   5    ;    4   ;;   3    ;   2    ;   1    ;   0    ;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;

TIME_MGRQ:
TIME_MGR_:
       ld     hl,(TIMER_TABLE_BASE)      ; Current time address
NEXT_TIMER0:
       bit    FREE,(hl)           ; Free?
       call   z,DCR_TIMER         ; If not, decrement
       bit    EOT,(hl)            ; End of table?
       jr     nz,SCRAM            ; If it is, we're done.
       inc    hl                  ; Otherwise get next timer
       inc    hl                  ; and start over.
       inc    hl
       jr     NEXT_TIMER0
SCRAM:
       ret
;
DCR_TIMER:
       push   hl                  ; Save current timer.
       bit    LONG,(hl)           ; Long?
       jr     z,DCR_S_MODE_TBL    ; Short, non-repeating
       bit    REPEAT,(hl)         ; Repeat?
       jr     nz,DCR_L_RPT_TBL    ; Long, repeating
```

```
DCR_L_MODE_TBL:                  ; Long non-repeating
       inc    hl
       ld     e,(hl)        ; Move the counter to DE
       inc    hl
       ld     d,(hl)
       dec    de                ; Decrement.
       ld     a,e
       or     d                 ; Check if 0.
       jr     nz,SAVE_2_BYTES   ; If not save'm.
       pop    hl                ; Otherwise, get mode byte
       push   hl                ; and set it's done bit.
       jr     SET_DONE_BIT
;
DCR_L_RPT_TBL:                   ; Long-repeating timer
       inc    hl
       ld     e,(hl)        ; Move the counter to DE
       inc    hl
       ld     d,(hl)
       ex     de,hl             ; Exchange and
       ld     e,(hl)        ; load addr. Into DE
       inc    hl
       ld     d,(hl)
       dec    de                ; Decrement
       ld     a,e
       or     d                 ; Check for 0.
       jr     nz,SAVE_2_BYTES   ; Save if not.
       inc    hl                ; Otherwise, reload
       ld     e,(hl)        ; original counter #.
       inc    hl
       ld     d,(hl)        ; jockey all over to
       dec    hl                ; perform said task!
       dec    hl
       ld     (hl),d
       dec    hl
       ld     (hl),e
       pop    hl
       push   hl
       jr     SET_DONE_BIT ; Then set done bit.
;
DCR_S_MODE_TBL:
       inc    hl
       dec    (hl)
       jr     nz,TIMER_EXIT
       pop    hl
       push   hl
       bit    REPEAT,(hl)       ; Repeat?
       jr     z,SET_DONE_BIT    ; If not, leave.
       inc    hl                ; Otherwise, jockey
       inc    hl                ; around again and
       ld     a,(hl)        ; reload original #.
       dec    hl
       ld     (hl),a
       dec    hl
       pop    hl
       push   hl
SET_DONE_BIT:
       set    DONE,(hl)
TIMER_EXIT:
       pop    hl
       ret
;
SAVE_2_BYTES:
       ld     (hl),d
```

```
       dec    hl
       ld     (hl),e
       jr     TIMER_EXIT
;
; Procedure Init Timer
; HL has address of Timer table
; DE has address of Timer Data Table
;
; #Common
; INIT_TIME_DATA:
; TEMP1: DEFS 2
; TEMP2: DEFS 2
;
INIT_TIMER_PARAM:
       dw     00002H
       dw     00002H
       dw     00002H
;
INIT_TIMERQ:
       ld     bc,INIT_TIMER_PARAM
       ld     de,PARAM_AREA
       call   PARAM_
       ld     hl,(PARAM_AREA)
       ld     de,(PARAM_AREA+2)
INIT_TIMER_:
       ld     (TIMER_TABLE_BASE),hl      ; Store given base address
                                         ;    for timer table
       ld     (hl),030H                  ; Set first byte in timer table to free
; and last timer
       ex     de,hl
       ld     (NEXT_TIMER_DATA_BYTE),hl  ; Store given base address
; for data block
       ret
;
; Procedure Free Signal
; Acc has signal number to be freed
; No output is generated
;
; #Common
; SIGNAL_NUM: DEFS 1
;
FREE_SIG_PARAM:
       dw     00001H
       dw     00001H
;
FREE_SIGNALQ:
       ld     bc,FREE_SIG_PARAM
       ld     de,PARAM_AREA+4
       call   PARAM_
       ld     a,(PARAM_AREA+4)
FREE_SIGNAL_:
       ld     c,a                        ; Put Free code into C register
       ld     hl,(TIMER_TABLE_BASE)      ; Get Timer Base address
       ld     b,a                        ; Timer offset already available
       ld     de,00003H                  ; Setup offset to subsequent timers
       or     a                   ; See if the first timer is requested
       jr     z,FREE_MATCH        ; If so we have a match
                                  ; Otherwise
FREE1:                            ; Loop to find requested signal
       bit    EOT,(hl)            ; Check if end of table
       jr     nz,FREE_EXIT        ; If so go no further
       add    hl,de               ; If not then offset to next timer
       dec    c                   ; Decrement signal request
```

```
       jr     nz,FREE1            ; If requested signal not zero
; then go try the next
; Else
FREE_MATCH:                               ; Here when the requested signal
; matches current signal
       bit    FREE,(hl)                   ; Check if already free
       jr     nz,FREE_SET                 ; If so then just reset LONG
       set    FREE,(hl)                   ; Set current timer to free
       bit    REPEAT,(hl)                 ; Check for repeating timer
       jr     z,FREE_SET                  ; If not then go free it
       bit    LONG,(hl)                   ; Check also for long timer
       jr     z,FREE_SET                  ; If not long then go free it
; call FREE_COUNTER_              ; Else release timer data before freeing it
;
; FREE (DELETE) COUNTER
; KENL 3/82
; NEEDS   "TO-DELETE":counter addr.in DE
;
FREE_COUNTER_:
                                  ; HL should countain timer to delete addr.
       inc    hl                  ; Get next after mode byte
       ld     e,(hl)              ; into DE
       inc    hl
       ld     d,(hl)
       push   de                         ; Save them for later.
       ld     hl,(TIMER_TABLE_BASE)
       push   hl                         ; Save beginning of table.
NEXT:
       bit    EOT,(hl)                    ; End of table?
       jr     nz,MOVE_IT
       bit    FREE,(hl)           ; Free?
       jr     nz,GET_NEXT         ; If so we don't want it
       ld     a,(hl)
       and    048H                ; Repeating and long?
       cp     048H
       jr     nz,GET_NEXT         ; If NOT we don't want it
       inc    hl
       inc    hl
       ld     a,(hl)
       cp     d
       jr     c,GET_NEXT          ; If so we don't want it
       jr     nz,SUBSTRACT_4      ; However, if larger, change it
       dec    hl
       ld     a,(hl)
       cp     e                   ; Smaller?
       jr     c,GET_NEXT          ; If so we don't want it
       jr     z,FREE_EXIT         ; Error if equal
       inc    hl                  ; Setup HL for SUBSTRACT_4
SUBSTRACT_4:
       ld     d,(hl)
       dec    hl
       ld     e,(hl)
       dec    de                  ; Reduce this addr. By 4.
       dec    de
       dec    de
       dec    de
       ld     (hl),e              ; Replace reduced addr.
       inc    hl                  ; Back where we got it.
       ld     (hl),d
       jr     GET_NEXT
;
GET_NEXT:
       pop    hl                  ; Now we can get next timer.
```

```
        inc     hl
        inc     hl
        inc     hl
        push    hl
        jr      NEXT
;
MOVE_IT:
        ld      b,000H
        or      a                           ; CLEAR CARRY
        pop     hl
        pop     de                          ; Get addr. of timer to delete
        push    hl
        ld      hl,(NEXT_TIMER_DATA_BYTE)   ; Find # of bytes
        sbc     hl,de                       ; to move by substraction
        ld      c,l                         ; Save in counter reg.
        ld      l,e                         ; Copy into HL.
        ld      h,d
        inc     hl                          ; Find source addr.
        inc     hl
        inc     hl
        inc     hl
        ldir                                ; Move it!
        ld      bc,00008H     ; Adjust next available by -4 from LDIR dest.
        sbc     hl,bc         ; (or -8 from source of LDIR! [saves instrs.]).
        ld      (NEXT_TIMER_DATA_BYTE),hl
        pop     hl
FREE_SET:
        ; RES LONG,(HL)             ; Reset repeat bit just in case
FREE_EXIT:
        ret                         ; Return
;
; Procedure Request Signal
; HL pair has length of timer
; Acc has zero for repeating timer any other value for a non_repeating type
; Signal number is returned in the Accumulator
; #Common
; REPEAT_SIG_CODE:  DEFS   1
; TIMER_LENGTH:     DEFS   2
REQUEST_SIG_PARAM:
        dw      00002H
        dw      00001H
        dw      00002H
;
REQUEST_SIGNALQ:
        ld      bc,REQUEST_SIG_PARAM
        ld      de,PARAM_AREA+5
        call    PARAM_
        ld      hl,(TIMER_LENGTH)
        ld      a,(PARAM_AREA+5)
REQUEST_SIGNAL_:
        ld      c,a                         ; Put Repeat Code into C register
        ex      de,hl                       ; Put length of timer into DE
        ld      hl,(TIMER_TABLE_BASE)       ; Get Timer Base Address
        xor     a                           ; Into offset to First Table value
        ld      b,a
TIMER1:
        bit     FREE,(hl)             ; See if current timer free
        jr      z,NEXT_TIMER1         ; If not go get the next timer
        push    hl
        ld      a,(hl)
        and     010H
        or      020H
        ld      (hl),a
```

```
       xor    a
       or     d
       jr     nz,LONG_TIMER
       ; RES  FREE,[HL]           ; Reset free bit
       ; OR   D                   ; Check for zero
       ; JR   NZ,LONG_TIMER; If non_zero then it's a long timer
       ; RES  REPEAT,[HL]         ; Set for a NON_Repeating timer
       ; RES  LONG,[HL]
       ; LD   A,C                 ; Check for a short repeating timer
       or     c
       jr     z,NOT_A_REPEAT_TIMER       ; Don't reset repeat bit in mode byte
                                         ; if non_repeating
       set    REPEAT,(hl)                ; Set repeat bit
NOT_A_REPEAT_TIMER:
       inc    hl                 ; Go to next table location
       ld     (hl),e             ; Store timer length
       inc    hl
       ld     (hl),e             ; Store timer length again in case of repeat
       jr     INIT_TIMER__EXIT   ; All done so let's exit
;
LONG_TIMER:
       set    LONG,(hl)          ; Set long timer bit
       ld     a,c                ; Check for a long repeat timer
       or     a
       jr     z,NOT_A_LONG_REPEAT ; If zero then go to non_repeating timer
       push   de                        ; Store timer length temporary
       ex     de,hl                     ; Swap registers
       ld     hl,(NEXT_TIMER_DATA_BYTE) ; To get free space
                                        ; in long timer table
       ex     de,hl              ; Then swap back
       set    REPEAT,(hl)        ; Set mode byte to repeating
       inc    hl
       ld     (hl),e             ; Store low-byte of timer address
                                 ; into the value word
       inc    hl
       ld     (hl),d             ; Store high-byte of timer address
       ex     de,hl              ; Move address of data area into HL
       pop    de                 ; Get back the length of timer
       ld     (hl),e             ; Store that in the data table
       inc    hl
       ld     (hl),d
       inc    hl
       ld     (hl),e             ; Store it again
       inc    hl
       ld     (hl),d
       inc    hl
       ld     (NEXT_TIMER_DATA_BYTE),hl  ; Store the next available data area
                                         ; for future use
       jr     INIT_TIMER__EXIT
;
NOT_A_LONG_REPEAT:
       inc    hl
TIMER2:
       ld     (hl),e             ; Store it again
       inc    hl
       ld     (hl),d
       inc    hl
       jr     INIT_TIMER__EXIT
;
NEXT_TIMER1:
       bit    EOT,(hl)
       jr     nz,MAKE_NEW_TIMER
       inc    hl                 ; Go to next mode byte
```

```
        inc     hl
        inc     hl
        inc     b                       ; Count to next offset
        jr      TIMER1                  ; Go back up to init. timer
;
MAKE_NEW_TIMER:                         ; Maximum of 255 signals allowed
        push    de                      ; Save DE for a work register
        push    hl                      ; Save current timer address
        inc     hl                      ; Go to next available memory location
        inc     hl                      ; in the Timer Table
        inc     hl
        inc     b
        ld      (hl),030H
        ex      de,hl                   ; Save momentarily
        pop     hl                      ; Get back original timer
        res     EOT,(hl)                ; Reset previous last timer
        ex      de,hl                   ; Get back current last timer
        pop     de                      ; Restore DE register
        jr      TIMER1                  ; Go back up and initialize counter for use
;
INIT_TIMER__EXIT:
        pop     hl
        res     FREE,(hl)
        ld      a,b                     ; Put the offset into the Accumulator
; for the user of routine
        ret
;
; Procedure Test Signal
; Acc has the Signal number to be tested
; A value of True(1) or False(0) is returned in the Accumulator for the
;                 Signal given.
;
; #Common
; TEST_SIG_NUM:     DEFS   1
TEST_SIG_PARAM:
        dw      00001H
        dw      00001H
;
TEST_SIGNALQ:
        ld      bc,TEST_SIG_PARAM
        ld      de,TEST_SIG_NUM
        call    PARAM_
        ld      a,(TEST_SIG_NUM)
TEST_SIGNAL_:
        ld      c,a                     ; Put Signal Code into C register
        ld      hl,(TIMER_TABLE_BASE)       ; Get Timer Base Address
        ld      b,a                     ; Save Signal
        ld      de,00003H               ; Setup offset for next timer
        or      a                       ; See if first timer is a match
        jr      z,SIGNAL_MATCH          ; If so go check it
TEST1:                                  ; Loop to match timer table to desired timer
        bit     EOT,(hl)                ; Check for end of table
        jr      nz,SIGNAL_FALSE         ; If so then return a not done
        add     hl,de                   ; Now index to next timer
        dec     c                       ; Decrement to the timer desired
        jr      nz,TEST1                ; If not a timer desired timer then go back
SIGNAL_MATCH:                           ; Here with a timer match
        bit     FREE,(hl)
        jr      nz,SIGNAL_FALSE
        bit     DONE,(hl)               ; Check for timer done
        jr      nz,SIGNAL_TRUE          ; If so then go return a True
SIGNAL_FALSE:                           ; Here to return a false for either
                                        ; a not done or non_existent timer
```

```
       xor    a                   ; Put a false in Acc
       jr     TEST_EXIT           ; Got to the exit
;
SIGNAL_TRUE:                      ; Here when timer is finished
       bit    REPEAT,(hl)         ; Check for repeating timer
       jr     nz,SIGNAL_TRUE1     ; If so then just return True
       set    FREE,(hl)           ; Free current timer since not repeating
SIGNAL_TRUE1:
                                  ; ***** Start add 4/30/82****
       res    DONE,(hl)           ; Reset current timer to not done
                                  ; ***** End add 4/30/82****
       ld     a,001H              ; Put a true in the Acc
TEST_EXIT:                        ; Return
       or     a
       ret
```

CONTROLLER SOFTWARE

```
;
; #Defines
NUM_DEV       EQU    005H
KDB_NULL      EQU    00FH
CONTROLLER_0  EQU    000H
STROBE_SET    EQU    001H
FIRE_MASK     EQU    040H
JOY_MASK      EQU    00FH
ARM_MASK      EQU    040H
KBD_MASK      EQU    00FH
PLYR_0        EQU    00002H
PLYR_1        EQU    00007H
SEG_0         EQU    007H
SEG_1         EQU    018H
FIRE          EQU    000H
JOY           EQU    001H
SPIN          EQU    002H
ARM           EQU    003H
KBD           EQU    004H
FIRE_OLD      EQU    000H
FIRE_STATE            EQU    001H
JOY_OLD       EQU    002H
JOY_STATE             EQU    003H
SPIN_OLD      EQU    004H
SPIN_STATE            EQU    005H
ARM_OLD       EQU    006H
ARM_STATE             EQU    007H
KBD_OLD       EQU    008H
KBD_STATE             EQU    009H
;
; #Defines (PORTS)
STRB_RST_PORT EQU    0C0H
STRB_SET_PORT EQU    080H
CTRL_0_PORT   EQU    0FCH
CTRL_1_PORT   EQU    0FFH
;
; #Global
; GLB  CONTROLLER_INIT     ; Initialize controller to strobe reset
; GLB  CONT_READ           ; Controller read routine
; GLB  CONT_SCAN           ; Controller scanner routine
; GLB  UPDATE_SPINNER_     ; Update spinner switch routine
; GLB  DECODER_            ; Decodes raw, undebounced data
; GLB  POLLER_             ; Polling routine for all devices in controller

; Decoder table for keyboard (keypad)
DEC_KBD_TBL:
       db     KDB_NULL
       db     006H   ; '6'
       db     001H   ; '1'
       db     003H   ; '3'
       db     009H   ; '9'
       db     000H   ; '0'
       db     00AH   ; '*'
       db     KDB_NULL
       db     002H   ; '2'
       db     00BH   ; '#' (RESET)
       db     007H   ; '7'
       db     KDB_NULL
       db     005H   ; '5'
```

```
        db      004H    ; '4'
        db      008H    ; '8'
        db      KDB_NULL
;
CONTROLLER_INIT:
; Initialize controller to strobe reset
        out     (STRB_RST_PORT),a
        xor     a
        ld      ix,(CONTROLLER_MAP)
        inc     ix
        inc     ix
        ld      iy,DBNCE_BUFF
        ld      b,NUM_DEV*2
CINIT1:
        ; Clear controller memory and debounce status buffer
        ld      (ix+000H),a
        inc     ix
        ld      (iy+000H),a
        inc     iy
        ld      (iy+000H),a
        inc     iy
        dec     b
        jr      nz,CINIT1
        ; Clear remaining variables
        ld      (SPIN_SW0_CT),a
        ld      (SPIN_SW1_CT),a
        ld      (S0_C0),a
        ld      (S0_C1),a
        ld      (S1_C0),a
        ld      (S1_C1),a
        ret
;
DELAY:
        ; Delay after strobe, before read
        nop
        ret
;
; Controller read routine
; Input: H - Controller number
; Output: A - Raw data
CONT_READ:
        ld      a,h
        cp      CONTROLLER_0  ; If controller<>0
        jr      nz,CONT_READ1 ; then read player 1
        in      a,(CTRL_0_PORT)      ; else read player 0
        jr      CONT_READX
;
CONT_READ1:
        in      a,(CTRL_1_PORT)
CONT_READX:
        cpl                          ; complement data so 0=no data, 1=data
        ret
;
; Controller scanner routine
CONT_SCAN:
        in      a,(CTRL_0_PORT)      ; Read segment 0, both players
        cpl
        ld      (S0_C0),a
        in      a,(CTRL_1_PORT)
        cpl
        ld      (S0_C1),a
        out     (STRB_SET_PORT),a    ; Strobe segment 1
        call    DELAY                ; wait 10 microsecs
```

```
        in      a,(CTRL_0_PORT)      ; Read segment 1, both players
        cpl
        ld      (S1_C0),a
        in      a,(CTRL_1_PORT)
        cpl
        ld      (S1_C1),a
        out     (STRB_RST_PORT),a    ; Reset to segment 0
        ret
;
; Update spinner switch routine
UPDATE_SPINNER_:
        in      a,(CTRL_0_PORT)      ; Get data
        ld      hl,SPIN_SW0_CT       ; address of spinner 0 count
        bit     4,a                  ; If int bit set
        jr      nz,UPDATE_S1  ; Then spinner 1
UPDATE_S0:
        bit     5,a                  ; If bit 5 is set
        jr      nz,UPDATE_R0  ; Then going right
                                     ; Else left
        dec     (hl)                 ; Decrement spinner counter
        jr      UPDATE_S1            ; Go check spinner 1
;
UPDATE_R0:
        inc     (hl)                 ; Right, spinner increment counter
UPDATE_S1:
        in      a,(CTRL_1_PORT)      ; Look at spinner 1 data
        bit     4,a                  ; If int bit set
        jr      nz,UPDATE_SPINX      ; Then not spinner 1
        inc     hl                   ; Else spinner 1, bump HL
; Check direction
        bit     5,a                  ; If bit 5 is set
        jr      nz,UPDATE_R1  ; Then going right
                                     ; Else left
        dec     (hl)                 ; Decrement spinner counter
        jr      UPDATE_SPINX
;
UPDATE_R1:
        inc     (hl)                 ; Right, increment spinner counter
UPDATE_SPINX:
        ret
;
; Decoder routine
; Desc.: This routine returns decoded raw, undebounced data
;        and may or not be required by O/S
;
; Input: H - Controller number
;        L - Segment number
;
; Output:
;                    SEGMENT 0     SEGMENT 1
;        H - BYTE 1 FIRE          ARM
;        L - BYTE 2 JOYSTICK      KBD
;        E - BYTE 3 SPINNER
DECODER_:
        ld      a,l
        cp      STROBE_SET    ; If L=1 then decode segment 1
        jr      z,DEC_SEG1
; Segment 0 (FIRE BUTTON, JOYSTICK)
; Return H=FIRE BUTTON, L=JOYSTICK, E=SPINNER
; Do spinner first
        ld      bc,SPIN_SW0_CT
        ld      a,h
        cp      CONTROLLER_0  ; If player=0 then go decode
```

```
       jr     z,DEC_PLYR
       inc    bc                  ; Else increment BC to spinner 1
DEC_PLYR:
       ld     a,(bc)              ; Get spinner switch count
       ld     e,a                 ; Return it in E
       xor    a
       ld     (bc),a              ; Clear out device data for player
       call   CONT_READ           ; Get other device data for player
       ld     d,a                 ; Save it
       and    JOY_MASK            ; Mask out joystick data
       ld     l,a                 ; Return it in L
       ld     a,d                 ; Restore data
       and    FIRE_MASK           ; Mask out fire button data
       ld     h,a                 ; Return it in H
       jr     DECODERX
;
; Segment 1 (ARM BUTTON, KEYBOARD)
; Return H=ARM BUTTON, L=KEYBOARD
DEC_SEG1:
       out    (STRB_SET_PORT),a   ; Strobe segment 1
       call   CONT_READ           ; Read segment 1 player data
       ld     d,a                 ; Save it
       out    (STRB_RST_PORT),a   ; Reset back to segment 0
       and    KBD_MASK            ; Mask out keyboard data
       ld     hl,DEC_KBD_TBL      ; Get decoder table address
       ld     b,000H
       ld     c,a
       add    hl,bc               ; Compute offset
       ld     l,(hl)              ; Return keyboard data in L
       ld     a,d                 ; Restore data
       and    ARM_MASK            ; Mask out ARM button data
       ld     h,a                 ; Return it in H
DECODERX:
       ret
;
; Polling routine for all devices in controller
POLLER_:
       call   CONT_SCAN                ; Go scan all data first
       ld     iy,DBNCE_BUFF            ; Debounce buffer pointer
       ld     ix,(CONTROLLER_MAP)      ; controller memory pointer
       push   ix
       ld     a,(ix+000H)              ; Get player 0 status
       bit    7,a                      ; If player 0 is not active
       jr     z,CHK_PLYR_1             ; Then check player 1
                                       ; Else (player 0 is active)
       ld     b,a                      ; Save status
       ld     de,PLYR_0                ; Compute address for player 0
       add    ix,de                    ; CONTROLLER_MEMORY
       and    SEG_0                    ; If segment 0 is not active
       jr     z,CHK_SEG_01             ; Then check segment 1
                                       ; Else (segment 0 is active)
       ld     a,(S0_C0)
       ld     hl,SPIN_SW0_CT
       call   DECODE_0                 ; Decode data for segment 0
CHK_SEG_01:
       ld     a,b                      ; Restore player 0 status
       and    SEG_1                    ; If segment 1 is not active
       jr     z,CHK_PLYR_1             ; Then check player 1
                                       ; Else (segment 1 is active)
       ld     a,(S1_C0)
       call   DECODE_1                 ; Decode data for segment 1
CHK_PLYR_1:
       pop    ix
```

```
       ld     a,(ix+001H)               ; Get Player 1 status
       bit    7,a                       ; If player 1 is not active
       jr     z,POLLER_X                ; Then exit all done
                                        ; Else (player 1 is active)
       ld     b,a                       ; Save player 1 status
       ld     de,2*NUM_DEV              ; Compute address of debounce buffer
       add    iy,de                     ; for player 1
       ld     de,PLYR_1                 ; Compute address of CONTROLLER_MEMORY
       add    ix,de                     ; for player 1
       and    SEG_0                     ; If segment 0 is not active
       jr     z,CHK_SEG_11              ; Then check segment 1
       ld     a,(S0_C1)
       ld     hl,SPIN_SW1_CT
       call   DECODE_0                  ; Decode data for segment 1
CHK_SEG_11:
       ld     a,b                       ; Restore status for player 1
       and    SEG_1                     ; If segment 1 is not active
       jr     z,POLLER_X                ; Then exit, all done
                                        ; Else (segment 1 is active)
       ld     a,(S1_C1)
       call   DECODE_1                  ; Decode data for segment 1
POLLER_X:
       ret
;
; Decode routine for segment 0
; Input: A - Data
;        B - Device status byte for current player
;       HL - Address of spinner data
;       IX - Pointer to controller memory
;       IY - Pointer to debounce status buffer
DECODE_0:
       ld     c,a                ; Save data
       bit    JOY,b              ; If joystick not active
       jr     z,DEC_FIRE         ; Then check fire button
                                 ; Else joystick active
       call   JOY_DBNCE          ; Debounce joystick data
       ld     a,c
DEC_FIRE:
       bit    FIRE,b             ; If fire button is not active
       jr     z,DEC_SPIN         ; Then check spinner
                                 ; Else (fire button active)
       call   FIRE_DBNCE         ; Debounce fire button
       ld     a,c
DEC_SPIN:
       bit    SPIN,b             ; If spinner is not active
       jr     z,DECODE_0X        ; Then exit decoder
                                 ; Else (spinner active)
       ld     a,(hl)             ; Save spinner count
       add    a,(ix+SPIN)
       ld     (ix+SPIN),a        ; in controller memory
       xor    a
       ld     (hl),a             ; Clear counter
DECODE_0X:
       ret
;
; Decoder routine for segment 1
; Input: A - Data
;        B - Device status byte for current player
;       IX - Pointer to controller memory
;       IY - Pointer to debounce status buffer
DECODE_1:
       ld     c,a                ; Save data
       bit    ARM,b              ; If ARM button not active
```

```
       jr     z,DEC_KBD           ; Then check keyboard
                                  ; Else (ARM button active)
       call   ARM_DBNCE           ; Debounce ARM button
       ld     a,c
DEC_KBD:
       bit    KBD,b               ; If keyboard not active
       jr     z,DECODE_1X         ; Then exit decoder
                                  ; Else keyboard active
       call   KBD_DBNCE           ; Debounce keyboard
DECODE_1X:
       ret
;
; Keyboard debounce routine
; Input: A - Raw data
;        IX - Controller memory pointer
;        IY - Debounce status buffer
KBD_DBNCE:
       push   bc
       push   de
       push   hl
       and    KBD_MASK            ; Mask out valid data
       ld     e,a                 ; and save it
       ld     b,(iy+KBD_OLD)      ; Get old data
       ld     a,(iy+KBD_STATE)    ; and current status
       cp     000H                ; If state<>0
       jr     nz,KBD_ST1          ; Then must be state=1
KBD_ST0:                          ; Else (state=0)
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,KBD_REG           ; Then saw data twice in sequence
       ld     (iy+KBD_OLD),e      ; Else first time, save current data
       jr     KBD_EXIT
;
KBD_REG:
       ld     a,001H        ; Set state=1
       ld     (iy+KBD_STATE),a
       ld     hl,DEC_KBD_TBL      ; Decode table address
       ld     d,000H        ; D/E raw data
       add    hl,de               ; Compute address into table
       ld     a,(hl)        ; Do table lookup
       ld     (ix+KBD),a          ; Save in controller memory 2kbd
       jr     KBD_EXIT
;
KBD_ST1:
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,KBD_EXIT          ; No change in state
       ld     (iy+KBD_OLD),e      ; Else save current data
       xor    a                   ; Set state=0
       ld     (iy+KBD_STATE),a
KBD_EXIT:
       pop    hl
       pop    de
       pop    bc
       ret
;
; Fire debounce routine
; Input: A - Raw data
;        IX - Controller memory pointer
;        IY - Debounce status buffer
FIRE_DBNCE:
       push   bc
       push   de
```

```
       and    FIRE_MASK           ; Mask out valid data
       ld     e,a                 ; and save it
       ld     b,(iy+FIRE_OLD)     ; Get old data
       ld     a,(iy+FIRE_STATE)   ; and current status
       cp     000H                ; If state<>0
       jr     nz,FIRE_ST1         ; Then must be state=1
FIRE_ST0:                         ; Else (state=0)
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,FIRE_REG          ; Then saw data twice in sequence
       ld     (iy+FIRE_OLD),e     ; Else first time, save current data
       jr     FIRE_EXIT
;
FIRE_REG:
       ld     a,001H              ; Set state=1
       ld     (iy+FIRE_STATE),a
       ld     (ix+FIRE),e         ; Save in controller memory 2fire
       jr     FIRE_EXIT
;
FIRE_ST1:
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,FIRE_EXIT         ; No change in state
       ld     (iy+FIRE_OLD),e     ; Else save current data
       xor    a                   ; Set state=0
       ld     (iy+FIRE_STATE),a
FIRE_EXIT:
       pop    de
       pop    bc
       ret
;
; Joystick debounce routine
; Input: A - Raw data
;        IX - Controller memory pointer
;        IY - Debounce status buffer
JOY_DBNCE:
       push   bc
       push   de
       and    JOY_MASK            ; Mask out valid data
       ld     e,a                 ; and save it
       ld     b,(iy+JOY_OLD)      ; Get old data
       ld     a,(iy+JOY_STATE)    ; and current status
       cp     000H                ; If state<>0
       jr     nz,JOY_ST1          ; Then must be state=1
JOY_ST0:                          ; Else (state=0)
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,JOY_REG           ; Then saw data twice in sequence
       ld     (iy+JOY_OLD),e      ; Else first time, save current data
       jr     JOY_EXIT
;
JOY_REG:
       ld     a,001H              ; Set state=1
       ld     (iy+JOY_STATE),a
       ld     (ix+JOY),e          ; Save in controller memory 2joy
       jr     JOY_EXIT
;
JOY_ST1:
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,JOY_EXIT          ; No change in state
       ld     (iy+JOY_OLD),e      ; Else save current data
       xor    a                   ; Set state=0
```

```
       ld     (iy+JOY_STATE),a
JOY_EXIT:
       pop    de
       pop    bc
       ret
;
; ARM button debounce routine
; Input: A - Raw data
;        IX - Controller memory pointer
;        IY - Debounce status buffer
ARM_DBNCE:
       push   bc
       push   de
       and    ARM_MASK            ; Mask out valid data
       ld     e,a                 ; and save it
       ld     b,(iy+ARM_OLD)      ; Get old data
       ld     a,(iy+ARM_STATE)    ; and current status
       cp     000H                ; If state<>0
       jr     nz,ARM_ST1          ; Then must be state=1
ARM_ST0:                          ; Else (state=0)
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,ARM_REG           ; Then saw data twice in sequence
       ld     (iy+ARM_OLD),e      ; Else first time, save current data
       jr     ARM_EXIT
;
ARM_REG:
       ld     a,001H              ; Set state=1
       ld     (iy+ARM_STATE),a
       ld     (ix+ARM),e          ; Save in controller memory 2joy
       jr     ARM_EXIT
;
ARM_ST1:
       ld     a,e                 ; Get current data
       cp     b                   ; If old=new
       jr     z,ARM_EXIT          ; No change in state
       ld     (iy+ARM_OLD),e      ; Else save current data
       xor    a                   ; Set state=0
       ld     (iy+ARM_STATE),a
ARM_EXIT:
       pop    de
       pop    bc
       ret
```

DISPLAY LOGO

```
; Description:
; Displays the Coleco logo screen with COLECOVISION on a
; black background.  The game title, manufacturer, and
; copyright year are obtained from the cartridge and
; overlayed onto the logo screen.  The logo is then
; displayed for 10 seconds after which time a jump to
; the game start address is executed.
;
; If no cartridge is present a default message is
; displayed, instructing the operator to:
;
;       "TURN GAME OFF"
;       "BEFORE INSERTING CARTRIDGE"
;       "OR EXPANSION MODULE."
;       "© 1982 COLECO"
;
; This message is displayed for 60 seconds, the screen
; is then blanked and finally a soft halt is executed
; locking up the program until the unit is reset.
;
; DISPLAY_LOGO exists with the VDP in mode 1, the screen
; blanked, and the ASCII character set in VRAM.
;
; The memory map is as follows:
;
;              VDP MEMORY MAP
;   3800H-3FFFH  SPRITE GENERATOR TABLE
;   2000H-201FH  PATTERN COLOR TABLE
;   1B00H-1B7FH  SPRITE ATTRIBUTE TABLE
;   1800H-1AFFH  PATTERN NAME TABLE
;   0000H-17FFH  PATTERN GENERATOR TABLE

;
; #External
; EXT  PUT_VRAM
; EXT  GAME_NAME
; EXT  WRITE_REGISTER
; EXT  COLORTABLE
; EXT  START_GAME

;
; #Define
DATA_PORT  EQU  0BEH
CTRL_PORT  EQU  0BFH
CARTRIDGE  EQU  08000H

;
; #Global
; GLB  ASCII_TBL    ; Pointer to uppercase ASCII generators
; GLB  NUMBER_TBL   ; Pointer to 0-9 generators
; GLB  DISPLAY_LOGO ; Display COLECOVISION logo
; GLB  LOAD_ASCII_  ; Load PATTERN_GEN_TABLE with full ASCII set
; GLB  FILL_VRAM_   ; Fill VRAM with a value
; GLB  MODE_1_      ; Set up MODE_1 graphics
;

;
DISPLAY_LOGO:
       ld    hl,00000H
```

```
       ld     de,04000H
       ld     a,000H
       call   FILL_VRAM_
       call   MODE_1_
       call   LOAD_ASCII_
       ld     hl,OBJ_TABLE
       ld     de,00060H
WRITE_LOOP:
       push   hl
       push   de
       ld     a,(hl)
       cp     0FFH
       jr     z,DONE_LOGO
       ld     b,a
       inc    b
       ld     hl,LOGO_GEN
       ld     de,00008H
ADDR_ADJ:
       djnz   ADD_8
       pop    de
       push   de
       ld     iy,00001H
       ld     a,003H
       call   PUT_VRAM
       pop    de
       pop    hl
       inc    de
       inc    hl
       jr     WRITE_LOOP
;
DONE_LOGO:
       pop    de
       pop    hl
       jr     WRITE_NAMES
;
ADD_8:
       add    hl,de
       jr     ADDR_ADJ
;
WRITE_NAMES:
       ld     hl,LOGO_NAMES
       ld     de,00085H
       ld     iy,00016H
       ld     a,002H
       call   PUT_VRAM
       ld     hl,LOGO_NAMES+22
       ld     de,000A5H
       ld     iy,00016H
       ld     a,002H
       call   PUT_VRAM
       ld     hl,TRADEMARK
       ld     de,0009BH
       ld     iy,00002H
       ld     a,002H
       call   PUT_VRAM
; SET UP DEFAULT COPYRIGHT MESSAGE
       ld     hl,S_C_1982_COLECO
       ld     de,002AAH
       ld     iy,0000DH
       ld     a,002H
       call   PUT_VRAM
; WRITE OUT COLOR_NAME_TABLE
       ld     hl,LOGO_COLORS
```

```
        ld      de,00000H
        ld      a,004H
        ld      iy,00012H
        call    PUT_VRAM
; ENABLE DISPLAY
        ld      b,001H
        ld      c,0C0H
        call    WRITE_REGISTER
; CARTRIDGE TEST
        ld      hl,CARTRIDGE
        ld      a,(hl)
        cp      0AAH
        jr      nz,NO_CARTRIDGE
        inc     hl
        ld      a,(hl)
        cp      055H
        jr      nz,NO_CARTRIDGE
; CARTRIDGE PRESENT
; DISPLAY GAME TITLE
        ld      hl,GAME_NAME
        call    PARSE
        ld      de,GAME_NAME
        ld      hl,00201H
        call    CENTER_PRT
; DISPLAY COMPANY NAME:
        ld      hl,GAME_NAME
        call    PARSE
        inc     hl
        ld      d,h
        ld      e,l
        call    PARSE
        ld      hl,L01C1
        call    CENTER_PRT
; CHANGE DATE
        ld      hl,GAME_NAME
        call    PARSE
        inc     hl
        call    PARSE
        inc     hl
        ld      de,002ACH
        ld      iy,00004H
        ld      a,002H
        call    PUT_VRAM
; DISPLAY 10 SECONDS
        call    DELAY_10
; TURN OFF DISPLAY
        ld      b,001H
        ld      c,080H
        call    WRITE_REGISTER
        ld      hl,(START_GAME)
        jp      (hl)          ;INFO: index jump
;
NO_CARTRIDGE:
        ld      hl,S_TURN_GAME_OFF
        ld      de,001AAH
        ld      iy,0000DH
        ld      a,002H
        call    PUT_VRAM
        ld      hl,S_BEFORE_INSERTING_CARTRIDGE
        ld      de,001E4H
        ld      iy,0001AH
        ld      a,002H
        call    PUT_VRAM
```

```
       ld     hl,S_OR_EXPANSION_MODULE
       ld     de,00227H
       ld     iy,00014H
       ld     a,002H
       call   PUT_VRAM
; DISPLAY 60 SECONDS
       ld     hl,08A00H
       call   TIMER_1
; TURN OFF DISPLAY
       ld     b,001H
       ld     c,080H
       call   WRITE_REGISTER
SOFT_HALT:
       jr     SOFT_HALT
;
; *************************
; *                       *
; *      DATA TABLES      *
; *                       *
; *************************
;
; **** COLOR_NAME_TABLE ****
LOGO_COLORS:
       db     000H, 000H, 000H, 0F0H, 0F0H, 0F0H,     0F0H,  0F0H
       db     0F0H,  0F0H,  0F0H,  0F0H,  0D0H,  080H,  090H,  0B0H
       db     030H,  040H
; **** PATTERN_NAME_TABLE ****
LOGO_NAMES:
       db     060H, 061H, 068H, 069H, 070H, 071H, 078H, 079H, 080H, 081H, 088H
       db     089H, 064H, 065H, 06CH, 074H, 075H, 07CH, 084H, 085H, 08CH, 08DH
       db     062H, 063H, 06AH, 06BH, 072H, 073H, 07AH, 07BH, 082H, 083H, 08AH
       db     08BH, 066H, 067H, 06DH, 076H, 077H, 07DH, 086H, 087H, 08EH, 08FH
S_TURN_GAME_OFF:
       db     "TURN GAME OFF"
S_BEFORE_INSERTING_CARTRIDGE:
       db     "BEFORE INSERTING CARTRIDGE"
S_OR_EXPANSION_MODULE:
       db     "OR EXPANSION MODULE."
S_C_1982_COLECO:
       db     01DH,  " 1982 COLECO" ; 01DH = COPYRIGHT
TRADEMARK:
       db     01EH,  01FH
; **** PATTERN_GENERATOR_TABLES ****
LOGO_GEN:
       db     000H, 000H, 000H, 000H, 000H, 000H, 000H, 000H
       db     03FH, 07FH, 0FFH, 0FFH, 0F3H, 0F3H, 0F0H, 0F0H
       db     000H, 080H, 0C0H, 0C0H, 0C0H, 0C0H, 000H, 000H
       db     03FH, 07FH, 0FFH, 0FFH, 0F3H, 0F3H, 0F3H, 0F3H
       db     000H, 080H, 0C0H, 0C0H, 0C0H, 0C0H, 0C0H, 0C0H
       db     0F0H, 0F0H, 0F0H, 0F0H, 0F0H, 0F0H, 0F0H, 0F0H
       db     0FFH, 0FFH, 0FFH, 0F0H, 0F0H, 0FFH, 0FFH, 0FFH
       db     0C0H, 0C0H, 0C0H, 000H, 000H, 000H, 000H, 000H
       db     0F1H, 0F1H, 0F1H, 07BH, 07BH, 07BH, 03FH, 03FH
       db     0E0H, 0E0H, 0E0H, 0C0H, 0C0H, 0C0H, 080H, 080H
       db     01FH, 03FH, 07FH, 079H, 078H, 07FH, 07FH, 03FH
       db     080H, 0C0H, 0E0H, 0E0H, 000H, 080H, 0C0H, 0E0H
       db     0F3H, 0F3H, 0FBH, 0FBH, 0FBH, 0FFH, 0FFH, 0FFH
       db     0C0H, 0C0H, 0C0H, 0C0H, 0C0H, 0C0H, 0C0H, 0C0H
       db     0F3H, 0F3H, 0FFH, 0FFH, 07FH, 03FH, 000H, 000H
       db     0C0H, 0C0H, 0C0H, 0C0H, 080H, 000H, 000H, 000H
       db     0F0H, 0F0H, 0FFH, 0FFH, 0FFH, 0FFH, 000H, 000H
       db     000H, 000H, 0C0H, 0C0H, 0C0H, 0C0H, 000H, 000H
       db     03FH, 01FH, 01FH, 01FH, 00EH, 00EH, 000H, 000H
```

```
        db      080H, 000H, 000H, 000H, 000H, 000H, 000H, 000H
        db      0F0H, 0F0H, 0F0H, 0F0H, 0F0H, 0F0H, 000H, 000H
        db      01FH, 001H, 079H, 07FH, 03FH, 01FH, 000H, 000H
        db      0E0H, 0E0H, 0E0H, 0E0H, 0C0H, 080H, 000H, 000H
        db      0FFH, 0F7H, 0F7H, 0F7H, 0F3H, 0F3H, 000H, 000H
        db      0C0H, 0C0H, 0C0H, 0C0H, 0C0H, 0C0H, 000H, 000H
ASC_TABLE:
        db      07EH, 081H, 0BDH, 0A1H, 0A1H, 0BDH, 081H, 07EH ; ©
        db      01FH, 004H, 004H, 004H, 000H, 000H, 000H, 000H ; t (trade)
        db      044H, 06CH, 054H, 054H, 000H, 000H, 000H, 000H ; m (mark)
SPACE:
        db      000H, 000H, 000H, 000H, 000H, 000H, 000H, 000H ;   (space)
        db      020H, 020H, 020H, 020H, 020H, 000H, 020H, 000H ; !
        db      050H, 050H, 050H, 000H, 000H, 000H, 000H, 000H ; "
        db      050H, 050H, 0F8H, 050H, 0F8H, 050H, 050H, 000H ; #
        db      020H, 078H, 0A0H, 070H, 028H, 0F0H, 020H, 000H ; $
        db      0C0H, 0C8H, 010H, 020H, 040H, 098H, 018H, 000H ; %
        db      040H, 0A0H, 0A0H, 040H, 0A8H, 090H, 068H, 000H ; &
        db      020H, 020H, 020H, 000H, 000H, 000H, 000H, 000H ; '
        db      020H, 040H, 080H, 080H, 080H, 040H, 020H, 000H ; (
        db      020H, 010H, 008H, 008H, 008H, 010H, 020H, 000H ; )
        db      020H, 0A8H, 070H, 020H, 070H, 0A8H, 020H, 000H ; *
        db      000H, 020H, 020H, 0F8H, 020H, 020H, 000H, 000H ; +
        db      000H, 000H, 000H, 000H, 020H, 020H, 040H, 000H ; ,
        db      000H, 000H, 000H, 0F8H, 000H, 000H, 000H, 000H ; -
        db      000H, 000H, 000H, 000H, 000H, 000H, 020H, 000H ; .
        db      000H, 008H, 010H, 020H, 040H, 080H, 000H, 000H ; /
NUMBER_TBL:
        db      070H, 088H, 098H, 0A8H, 0C8H, 088H, 070H, 000H ; 0
        db      020H, 060H, 020H, 020H, 020H, 020H, 070H, 000H ; 1
        db      070H, 088H, 008H, 030H, 040H, 080H, 0F8H, 000H ; 2
        db      0F8H, 008H, 010H, 030H, 008H, 088H, 070H, 000H ; 3
        db      010H, 030H, 050H, 090H, 0F8H, 010H, 010H, 000H ; 4
        db      0F8H, 080H, 0F0H, 008H, 008H, 088H, 070H, 000H ; 5
        db      038H, 040H, 080H, 0F0H, 088H, 088H, 070H, 000H ; 6
        db      0F8H, 008H, 010H, 020H, 040H, 040H, 040H, 000H ; 7
        db      070H, 088H, 088H, 070H, 088H, 088H, 070H, 000H ; 8
        db      070H, 088H, 088H, 078H, 008H, 010H, 0E0H, 000H ; 9
        db      000H, 000H, 020H, 000H, 020H, 000H, 000H, 000H ; :
        db      000H, 000H, 020H, 000H, 020H, 020H, 040H, 000H ; ;
        db      010H, 020H, 040H, 080H, 040H, 020H, 010H, 000H ; <
        db      000H, 000H, 0F8H, 000H, 0F8H, 000H, 000H, 000H ; =
        db      040H, 020H, 010H, 008H, 010H, 020H, 040H, 000H ; >
        db      070H, 088H, 010H, 020H, 020H, 000H, 020H, 000H ; ?
        db      070H, 088H, 0A8H, 0B8H, 0B0H, 080H, 078H, 000H ; @
ASCII_TBL:
        db      020H, 050H, 088H, 088H, 0F8H, 088H, 088H, 000H ; A
        db      0F0H, 088H, 088H, 0F0H, 088H, 088H, 0F0H, 000H ; B
        db      070H, 088H, 080H, 080H, 080H, 088H, 070H, 000H ; C
        db      0F0H, 088H, 088H, 088H, 088H, 088H, 0F0H, 000H ; D
        db      0F8H, 080H, 080H, 0F0H, 080H, 080H, 0F8H, 000H ; E
        db      0F8H, 080H, 080H, 0F0H, 080H, 080H, 080H, 000H ; F
        db      078H, 080H, 080H, 080H, 098H, 088H, 078H, 000H ; G
        db      088H, 088H, 088H, 0F8H, 088H, 088H, 088H, 000H ; H
        db      070H, 020H, 020H, 020H, 020H, 020H, 070H, 000H ; I
        db      008H, 008H, 008H, 008H, 008H, 088H, 070H, 000H ; J
        db      088H, 090H, 0A0H, 0C0H, 0A0H, 090H, 088H, 000H ; K
        db      080H, 080H, 080H, 080H, 080H, 080H, 0F8H, 000H ; L
        db      088H, 0D8H, 0A8H, 0A8H, 088H, 088H, 088H, 000H ; M
        db      088H, 088H, 0C8H, 0A8H, 098H, 088H, 088H, 000H ; N
        db      070H, 088H, 088H, 088H, 088H, 088H, 070H, 000H ; O
        db      0F0H, 088H, 088H, 0F0H, 080H, 080H, 080H, 000H ; P
        db      070H, 088H, 088H, 088H, 0A8H, 090H, 068H, 000H ; Q
```

```
        db      0F0H, 088H, 088H, 0F0H, 0A0H, 090H, 088H, 000H ; R
        db      070H, 088H, 080H, 070H, 008H, 088H, 070H, 000H ; S
        db      0F8H, 020H, 020H, 020H, 020H, 020H, 020H, 000H ; T
        db      088H, 088H, 088H, 088H, 088H, 088H, 070H, 000H ; U
        db      088H, 088H, 088H, 088H, 088H, 050H, 020H, 000H ; V
        db      088H, 088H, 088H, 0A8H, 0A8H, 0D8H, 088H, 000H ; W
        db      088H, 088H, 050H, 020H, 050H, 088H, 088H, 000H ; X
        db      088H, 088H, 050H, 020H, 020H, 020H, 020H, 000H ; Y
        db      0F8H, 008H, 010H, 020H, 040H, 080H, 0F8H, 000H ; Z
        db      0F8H, 0C0H, 0C0H, 0C0H, 0C0H, 0C0H, 0F8H, 000H ; [
        db      000H, 080H, 040H, 020H, 010H, 008H, 000H, 000H ; \
        db      0F8H, 018H, 018H, 018H, 018H, 018H, 0F8H, 000H ; ]
        db      000H, 000H, 020H, 050H, 088H, 000H, 000H, 000H ; ^
        db      000H, 000H, 000H, 000H, 000H, 000H, 000H, 0F8H ; _
        db      040H, 020H, 010H, 000H, 000H, 000H, 000H, 000H ; `
        db      000H, 000H, 070H, 088H, 0F8H, 088H, 088H, 000H ; a
        db      000H, 000H, 0F0H, 048H, 070H, 048H, 0F0H, 000H ; b
        db      000H, 000H, 078H, 080H, 080H, 080H, 078H, 000H ; c
        db      000H, 000H, 0F0H, 048H, 048H, 048H, 0F0H, 000H ; d
        db      000H, 000H, 0F0H, 080H, 0E0H, 080H, 0F0H, 000H ; e
        db      000H, 000H, 0F0H, 080H, 0E0H, 080H, 080H, 000H ; f
        db      000H, 000H, 078H, 080H, 0B8H, 088H, 070H, 000H ; g
        db      000H, 000H, 088H, 088H, 0F8H, 088H, 088H, 000H ; h
        db      000H, 000H, 0F8H, 020H, 020H, 020H, 0F8H, 000H ; i
        db      000H, 000H, 070H, 020H, 020H, 0A0H, 0E0H, 000H ; j
        db      000H, 000H, 090H, 0A0H, 0C0H, 0A0H, 090H, 000H ; k
        db      000H, 000H, 080H, 080H, 080H, 080H, 0F8H, 000H ; l
        db      000H, 000H, 088H, 0D8H, 0A8H, 088H, 088H, 000H ; m
        db      000H, 000H, 088H, 0C8H, 0A8H, 098H, 088H, 000H ; n
        db      000H, 000H, 0F8H, 088H, 088H, 088H, 0F8H, 000H ; o
        db      000H, 000H, 0F0H, 088H, 0F0H, 080H, 080H, 000H ; p
        db      000H, 000H, 0F8H, 088H, 0A8H, 090H, 0E0H, 000H ; q
        db      000H, 000H, 0F8H, 088H, 0F8H, 0A0H, 090H, 000H ; r
        db      000H, 000H, 078H, 080H, 070H, 008H, 0F0H, 000H ; s
        db      000H, 000H, 0F8H, 020H, 020H, 020H, 020H, 000H ; t
        db      000H, 000H, 088H, 088H, 088H, 088H, 070H, 000H ; u
        db      000H, 000H, 088H, 088H, 090H, 0A0H, 040H, 000H ; v
        db      000H, 000H, 088H, 088H, 0A8H, 0D8H, 088H, 000H ; w
        db      000H, 000H, 088H, 060H, 020H, 060H, 088H, 000H ; x
        db      000H, 000H, 088H, 050H, 020H, 020H, 020H, 000H ; y
        db      000H, 000H, 0F8H, 010H, 020H, 040H, 0F8H, 000H ; z
        db      038H, 040H, 020H, 0C0H, 020H, 040H, 038H, 000H ; {
        db      040H, 020H, 010H, 008H, 010H, 020H, 040H, 000H ; |
        db      0E0H, 010H, 020H, 018H, 020H, 010H, 0E0H, 000H ; }
        db      040H, 0A8H, 010H, 000H, 000H, 000H, 000H, 000H ; ~
        db      0A8H, 050H, 0A8H, 050H, 0A8H, 050H, 0A8H, 000H ; (deleted)
OBJ_TABLE:
        db      001H,  002H,  00EH,  00FH,  008H,  009H,  012H,  013H
        db      003H,  004H,  00EH,  00FH,  005H,  014H,  000H,  000H
        db      005H,  000H,  010H,  011H,  00AH,  00BH,  015H,  016H
        db      006H,  007H,  010H,  011H,  005H,  014H,  000H,  000H
        db      001H,  002H,  00EH,  00FH,  003H,  004H,  00EH,  00FH
        db      003H,  004H,  00EH,  00FH,  00CH,  00DH,  017H,  018H
        db      0FFH   ; END OF TABLE INDICATOR
;
; Desc.: Writes DE number of time a byte value in vram at HL address
; INPUT : HL = INDEX IN VRAM, A = BYTE TO COPY, DE = COUNT
; AFFECT AF, DE (reset to 0) and C (equal the input value A)
;
FILL_VRAM_:
        ld      c,a
        ld      a,l
        out     (CTRL_PORT),a
```

```
        ld      a,h
        or      040H
        out     (CTRL_PORT),a
FILL:
        ld      a,c
        out     (DATA_PORT),a
        dec     de
        ld      a,d
        or      e
        jr      nz,FILL
        call    READ_REGISTER
        ret
;
MODE_1_:
        ld      b,000H
        ld      c,000H
        call    WRITE_REGISTER
        ld      b,001H
        ld      c,080H
        call    WRITE_REGISTER
        ld      a,002H
        ld      hl,01800H
        call    INIT_TABLE
        ld      a,004H
        ld      hl,02000H
        call    INIT_TABLE
        ld      a,003H
        ld      hl,00000H
        call    INIT_TABLE
        ld      a,000H
        ld      hl,01B00H
        call    INIT_TABLE
        ld      a,001H
        ld      hl,03800H
        call    INIT_TABLE
        ld      b,007H
        ld      c,000H
        call    WRITE_REGISTER
        ret
;
; Desc.: Writes out ASCII character generators to the pattern
;        generator table. INIT_TABLE must be used to set up
;        the table address.
;
LOAD_ASCII_:
        ld      hl,ASC_TABLE  ; Location of generators
        ld      de,0001DH     ; Offset to place ASC_TABLE
        ld      iy,00060H     ;  in the correct location
        ld      a,003H
        call    PUT_VRAM
        ld      hl,SPACE
        ld      de,00000H
        ld      iy,00001H
        ld      a,003H
        call    PUT_VRAM
        ret
;
PARSE:
        ld      bc,00000H     ; From HL increment BC until
P_LOOP:
        ld      a,(hl) ; [HL] = "/"
        cp      02FH
        ret     z
```

```
       inc    hl
       inc    bc
       jr     P_LOOP
;
CENTER_PRT:
       push   bc     ; BC = LEN$
       pop    iy     ; IY = #items to be transferred in PUT_VRAM
       ld     a,020H ; DE = Location of start of string
       sbc    a,c    ;  A = 32-C
       rra           ;  DIV 2
       ld     b,000H
       ld     c,a
       add    hl,bc
       ld     b,h
       ld     c,l
       ld     h,d
       ld     l,e
       ld     d,b
       ld     e,c
       ld     a,002H
       call   PUT_VRAM
       ret
;
DELAY_10:
       ld     hl,01700H
TIMER_1:
       ld     de,000FFH
TIMER_2:
       dec    de
       ld     a,d
       or     e
       jr     nz,TIMER_2
       dec    hl
       ld     a,h
       or     l
       jr     nz,TIMER_1
       ret
```

GAME OPTION

```
; Displays the game option screen with white letters on a blue background.
; VDP is left in mode 1 with the VRAM memory map as follows.
;
;              VDP MEMORY MAP
;   3800H-3FFFH  SPRITE GENERATOR TABLE
;   2000H-201FH  PATTERN COLOR TABLE
;   1B00H-1B7FH  SPRITE ATTRIBUTE TABLE
;   1800H-1AFFH  PATTERN NAME TABLE
;   0000H-17FFH  PATTERN GENERATOR TABLE
;
; #External
; EXT  PUT_VRAM
; EXT  WRITE_REGISTER
; EXT  COLORTABLE
; EXT  LOAD_ASCII
; EXT  FILL_VRAM
; EXT  MODE_1
;
; #Global
; GLB  GAME_OPT_
;

; *********** Display game option screen ***********
GAME_OPT_:
; ***************** CLEAR 16K VRAM *****************
       ld     hl,00000H
       ld     de,04000H
       ld     a,000H
       call   FILL_VRAM
; ************* Set up VDP with mode 1 *************
       call   MODE_1
; ************ Set up background color *************
; Note: 0 = Black (default background color)
;       F = White (default forecolor not used in mode 1)
       ld     b,00FH
       ld     c,004H
       call   WRITE_REGISTER
; ********** Write out PATTERN_GEN_TABLE ***********
       call   LOAD_ASCII
; ********** Write out PATTERN_NAME_TABLE **********
       ld     hl,LINE_1
       ld     de,00025H
       ld     iy,00016H
       ld     a,002H
       call   PUT_VRAM
       ld     hl,LINE_2
       ld     de,00065H
       ld     iy,00017H
       ld     a,002H
       call   PUT_VRAM
       ld     de,000C5H
       call   WRITE_L3
       ld     de,00105H
       call   WRITE_L3
       ld     de,00145H
       call   WRITE_L3
       ld     de,00185H
       call   WRITE_L3
       ld     de,001E5H
```

```
        call    WRITE_L3
        ld      de,00225H
        call    WRITE_L3
        ld      de,00265H
        call    WRITE_L3
        ld      de,002A5H
        call    WRITE_L3
        ld      de,00105H
        call    WRITE_L4
        ld      de,00145H
        call    WRITE_L5
        ld      de,00185H
        call    WRITE_L6
        ld      hl,LINE_7
        ld      de,001E5H
        call    WRITE_CHAR
        ld      hl,LINE_8
        ld      de,00225H
        call    WRITE_CHAR
        ld      hl,LINE_9
        ld      de,00265H
        call    WRITE_CHAR
        ld      hl,LINE_10
        ld      de,002A5H
        call    WRITE_CHAR
        ld      de,0010FH
        call    WRITE_L4
        ld      de,0014FH
        call    WRITE_L5
        ld      de,0018FH
        call    WRITE_L6
        ld      de,001F1H
        call    WRITE_L11
        ld      de,00231H
        call    WRITE_L11
        ld      de,00271H
        call    WRITE_L11
        ld      de,002B1H
        call    WRITE_L11
        ld      de,0022FH
        call    WRITE_L4
        ld      de,0026FH
        call    WRITE_L5
        ld      de,002AFH
        call    WRITE_L6
        ld      de,001FBH
        call    WRITE_L12
        ld      de,0023bH
        call    WRITE_L12
        ld      de,0027BH
        call    WRITE_L12
        ld      de,002BBH
        call    WRITE_L12
; *********** Write out COLOR_NAME_TABLE ***********
; Note: F = White, 4 = Blue
        ld      hl,(COLORTABLE)
        ld      de,00020H
        ld      a,0F4H
        call    FILL_VRAM
; ***************** Enable display *****************
        ld      b,001H
        ld      c,0C0H
        call    WRITE_REGISTER
```

```
       ret
; ****************** DATA TABLES ******************
LINE_1:
       db     "TO SELECT GAME OPTION,"
LINE_2:
       db     "PRESS BUTTON ON KEYPAD."
LINE_3:
       db     "1 = SKILL 1/ONE PLAYER"
LINE_4:
       db     "2"
LINE_5:
       db     "3"
LINE_6:
       db     "4"
LINE_7:
       db     "5"
LINE_8:
       db     "6"
LINE_9:
       db     "7"
LINE_10:
       db     "8"
LINE_11:
       db     "TWO"
LINE_12:
       db     "S"
; *************** LOCAL SUBROUTINES ****************
WRITE_L3:
       ld     hl,LINE_3
       ld     iy,00016H
       ld     a,002H
       call   PUT_VRAM
       ret
;
WRITE_L4:
       ld     hl,LINE_4
       jr     WRITE_CHAR
;
WRITE_L5:
       ld     hl,LINE_5
       jr     WRITE_CHAR
;
WRITE_L6:
       ld     hl,LINE_6
WRITE_CHAR:
       ld     iy,00001H
       ld     a,002H
       call   PUT_VRAM
       ret
;
WRITE_L11:
       ld     hl,LINE_11
       ld     iy,00003H
       ld     a,002H
       call   PUT_VRAM
       ret
;
WRITE_L12:
       ld     hl,LINE_12
       ld     iy,00001H
       ld     a,002H
       call   PUT_VRAM
       ret
```

TABLE MANAGER

```
; #Define
; TRUE EQU 1
;
; #External
; EXT  VRAM_WRITE,REG_WRITE,VRAM_READ
; EXT  VDP_MODE_WORD
; EXT  MUX_SPRITES
; EXT  PARAM_
; EXT  LOCAL_SPR_TBL,SPRITE_ORDER
;
; #Global
; GLB  INIT_TABLE_,GET_VRAM_,PUT_VRAM_,INIT_SPR_ORDER_,WR_SPR_NM_TBL_
; GLB  INIT_TABLEQ,GET_VRAMQ,PUT_VRAMQ,INIT_SPR_ORDERQ,WR_SPR_NM_TBLQ
;
DATA_PORT EQU 0BEH
CTRL_PORT EQU 0BFH

;
; PROCEDURE INIT_TABLEQ (TABLE_CODE:BYTE;TABLE_ADDRESS:INTEGER)
;
INIT_TABLE_P:
       dw     00002H
       dw     00001H
       dw     00002H
;
INIT_TABLEQ:
       ld     bc,INIT_TABLE_P
       ld     de,PARAM_AREA
       call   PARAM_
       ld     a,(PARAM_AREA)
       ld     hl,(PARAM_AREA+1)
;
; INIT_TABLE_
; Desc.: Initializes the table addresses for VRAM tables.
;        Writes the appropriate base address into the
;        respective VDP register.
;
INIT_TABLE_:
       ld     c,a
       ld     b,000H
       ld     ix,VRAM_ADDR_TABLE
       add    ix,bc
       add    ix,bc
       ld     (ix+000H),l
       ld     (ix+001H),h
       ld     a,(VDP_MODE_WORD)
       bit    1,a
       jr     z,INIT_TABLE80
       ld     a,c
       cp     003H
       jr     z,CASE_OF_GEN
       cp     004H
       jr     z,CASE_OF_COLOR
       jr     INIT_TABLE80
;
CASE_OF_GEN:
       ld     b,004H
       ld     a,l
       or     h
```

```
        jr      nz,CASE_OF_GEN10
        ld      c,003H
        jr      INIT_TABLE90
;
CASE_OF_GEN10:
        ld      c,007H
        jr      INIT_TABLE90
;
CASE_OF_COLOR:
        ld      b,003H
        ld      a,l
        or      h
        jr      nz,CASE_OF_CLR10
        ld      c,07FH
        jr      INIT_TABLE90
;
CASE_OF_CLR10:
        ld      c,0FFH
        jr      INIT_TABLE90
;
; ** Compute base address (BASE_ADDRESS=TABLE_ADDRESS/FACTOR)
; ** Get bit shift count
INIT_TABLE80:
        ld      iy,BASE_FACTORS
        add     iy,bc
        add     iy,bc
        ld      a,(iy+000H)
        ld      b,(iy+001H)
DIVIDE:
        srl     h
        rr      l
        dec     a
        jr      nz,DIVIDE
        ld      c,l
INIT_TABLE90:
        call    REG_WRITE
        ret
;
BASE_FACTORS:
        db      007H
        db      005H
        db      00BH
        db      006H
        db      00AH
        db      002H
        db      00BH
        db      004H
        db      006H
        db      003H

;
; PROCEDURE GET_VRAMQ (TABLE_CODE:BYTE;START_INDEX:BYTE;SLICE:BYTE;
;                      VAR_DATA:BUFFER;ITEM_COUNT:INTEGER)
;
GET_VRAM_P:
        dw      00005H
        dw      00001H
        dw      00001H
        dw      00001H
        dw      0FFFEH
        dw      00002H
;
GET_VRAMQ:
```

```
        ld      bc,GET_VRAM_P
        ld      de,PARAM_AREA
        call    PARAM_
        ld      a,(PARAM_AREA)
        ld      de,(PARAM_AREA+1)
        ld      iy,(PARAM_AREA+5)
        ld      hl,(PARAM_AREA+3)
;
; GET_VRAM_
; Desc.: Gets a certain number of bytes from VRAM
;        and puts them in a buffer.
; Input: TABLE_CODE in A
;          0=SPRITE_NAME_TABLE
;          1=SPRITE_GENERATOR_TABLE
;          2=PATTERN_NAME_TABLE
;          3=PATTERN_GENERATOR_TABLE
;          4=COLOR_TABLE
;        START_INDEX in DE
;        DATA_BUFFER in HL
;        COUNT in IY
;
GET_VRAM_:
        call    SET_COUNT
        call    VRAM_READ
        ret
;
; SET_COUNT
; Desc.: Called by PUT_VRAM_ and GET_VRAM_
;        Sets byte count and index for writes
;        and reads to and from VRAM
;
;   TABLE         BYTES/ITEM
;   SPRITE_NAME       4
;   SPRITE_GEN        8
;   PATTERN_NAME      1
;   PATTERN_GEN       8
;   COLOR (MODE 1)    1
;   COLOR (MODE 2)    8
;
SET_COUNT:
        ld      (SAVED_COUNT),iy
        ld      ix,VRAM_ADDR_TABLE
        ld      c,a
        ld      b,000H
        cp      004H
        jr      nz,SET_COUNT10
        ld      a,(VDP_MODE_WORD)
        bit     1,a
        jr      z,SET_COUNT20
SET_COUNT10:
        ld      iy,SHIFT_CT
        add     iy,bc
        ld      a,(iy+000H)
        cp      000H
        jr      z,SET_COUNT20
ADJUST_INDEX:
        sla     e
        rl      d
        dec     a
        jr      nz,ADJUST_INDEX
END_ADJ_INDEX:
        push    bc
        ld      bc,(SAVED_COUNT)
```

```
        ld      a,(iy+000H)
        cp      000H
        jr      z,END_ADJ_COUNT
ADJUST_COUNT:
        sla     c
        rl      b
        dec     a
        jr      nz,ADJUST_COUNT
        ld      (SAVED_COUNT),bc
END_ADJ_COUNT:
        pop     bc
SET_COUNT20:
        push    hl
        add     ix,bc
        add     ix,bc
        ld      l,(ix+000H)
        ld      h,(ix+001H)
        add     hl,de
        ex      de,hl
        pop     hl
        ld      bc,(SAVED_COUNT)
        ret
;
SHIFT_CT:
        db      002H
        db      003H
        db      000H
        db      003H
        db      003H

;
; PROCEDURE PUT_VRAMQ (TABLE_CODE:BYTE;START_INDEX:BYTE;SLICE:BYTE;
;                      VAR_DATA:BUFFER;ITEM_COUNT:INTEGER)
;
PUT_VRAM_P:
        dw      00005H
        dw      00001H
        dw      00001H
        dw      00001H
        dw      0FFFEH
        dw      00002H
;
PUT_VRAMQ:
        ld      bc,PUT_VRAM_P
        ld      de,PARAM_AREA
        call    PARAM_
        ld      a,(PARAM_AREA)
        ld      de,(PARAM_AREA+1)
        ld      iy,(PARAM_AREA+5)
        ld      hl,(PARAM_AREA+3)
;
; PUT_VRAM_
; Desc.: Writes a certain number of bytes to VRAM
;        from a buffer.
; Input: TABLE_CODE in A
;          0=SPRITE_NAME_TABLE
;          1=SPRITE_GENERATOR_TABLE
;          2=PATTERN_NAME_TABLE
;          3=PATTERN_GENERATOR_TABLE
;          4=COLOR_TABLE
;        START_INDEX in DE
;        DATA_BUFFER in HL
;        COUNT in IY
```

```
;
PUT_VRAM_:
       push   af
       cp     000H
       jr     nz,ELSEZZ
       ld     a,(MUX_SPRITES)
       cp     001H
       jr     nz,ELSEZZ
       pop    af
       push   hl
       ld     hl,(LOCAL_SPR_TABLE)
       ld     a,e
       sla    a
       sla    a
       ld     e,a
       add    hl,de
       ex     de,hl
       push   iy
       pop    bc
       ld     a,c
       sla    a
       sla    a
       ld     c,a
       pop    hl
       ldir
       jr     END_IFZZ
;
ELSEZZ:
       pop    af
       call   SET_COUNT
       call   VRAM_WRITE
END_IFZZ:
       ret
;
; PROCEDURE INIT_SPR_ORDERQ (SPRITE_COUNT:BYTE)
;
INIT_SPR_P:
       dw     00001H
       dw     00001H
;
INIT_SPR_ORDERQ:
       ld     bc,INIT_SPR_P
       ld     de,PARAM_AREA
       call   PARAM_
       ld     a,(PARAM_AREA)
;
; INIT_SPR_ORDER_
; Desc.: Initializes the sprite display order list in RAM
;        to default order (0..31)
; Input: Number of sprites to order in A
;
INIT_SPR_ORDER_:
       ld     b,a
       xor    a
       ld     hl,(SPRITE_ORDER)
INIT_SPR10:
       ld     (hl),a
       inc    hl
       inc    a
       cp     b
       jr     nz,INIT_SPR10
       ret
;
```

```
; PROCEDURE WR_SPR_NM_TBLQ (SPRITE_COUNT:BYTE)
;
WR_SPR_P:
       dw     00001H
       dw     00001H
;
WR_SPR_NM_TBLQ:
       ld     bc,WR_SPR_P
       ld     de,PARAM_AREA
       call   PARAM_
       ld     a,(PARAM_AREA)
;
; WR_SPR_NM_TBL_
; Desc.: Writes SPRITE_NAME_TABLE to VRAM
;        using the sprite order list.
; Input: Number of sprites to write in A
;
WR_SPR_NM_TBL_:
       ld     ix,(SPRITE_ORDER)
       push   af
       ld     iy,VRAM_ADDR_TABLE
       ld     e,(iy+000H)
       ld     d,(iy+001H)
       ld     a,e
       out    (CTRL_PORT),a
       ld     a,d
       or     040H
       out    (CTRL_PORT),a
       pop    af
OUTPUT_LOOP_TABLE_MA:
       ld     hl,(LOCAL_SPR_TABLE)
       ld     c,(ix+000H)
       inc    ix
       ld     b,000H
       add    hl,bc
       add    hl,bc
       add    hl,bc
       add    hl,bc
       ld     b,004H
       ld     c,DATA_PORT
OUTPUT_LOOP10:
       outi
       nop                          ; delay
       nop
       jr     nz,OUTPUT_LOOP10
       dec    a
       jr     nz,OUTPUT_LOOP_TABLE_MA
       ret
```

DRIVERS FOR 9928 VDG

```
; The video drivers provide a standard protocol for the low-level communication
; with the 9918/28 VDP. There are four basic driver routines which between
; them allow the programmer to write a value to a VDP register, read the
; VDP status register, write a RAM or ROM buffer to a specified address
; in VRAM, and read a RAM buffer from a specified address in VRAM.
;
; The four routines outlined above are:
;
;   Procedure Reg_Write
;
;     Reg_Write takes a VDP register number (0..7) in the B register
;     and a byte value to be written to it in the C register. It writes
;     the value to the given VDP register and returns.
;
;     If the specified register is one of the VDP mode control registers,
;     ie. 0 or 1, the Reg_Write also writes the given value to the
;     corresponding half of the VDP_More_Word in RAM. All mode dependant
;     decisions made by the operating system are made by referencing the
;     contents of this word. Thus it is important for the cartridge
;     programmer to maintain it should he/she choose not to use
;     Reg_Write in accessing the VDP registers.
;
;     In addition to the BC pair, Reg_Write also makes use of AF.
;
;   Procedure Reg_Read
;
;     Reg_Read reads the VDP status register and returns its contents
;     in the A register.
;
;     It uses no other registers
;
;     NOTE ****** While this routine has no side effects with respect to
;                 the CPU, it should be used with caution since reads
;                 to the status register have the effect of resetting the
;                 VDP interrupt flag and may cause field interrupts to be
;                 missed.
;
;   Procedure Vram_Write
;
;     Vram_Write takes a pointer to the beginning of the data buffer in the
;     HL pair, the VRAM destination address in the DE pair, and a byte
;     count in the BC pair.
;
;     It writes the specified number of bytes from the buffer to VRAM
;     starting at the destination address.
;
;     The AF,BC,DE, and HL register pairs are all affected.
;
;     NOTE ****** This procedure is not re-entrant.
;
;   Procedure Vram_Read
;
;     Vram_Read takes a pointer to the beginning of the data buffer in the
;     HL pair, the VRAM source address in the DE pair, and a byte count
;     in the BC pair.
;
;     It reads the specified number of bytes into the buffer from VRAM
;     starting at the destination address.
;
```

```
;     The AF,BC,DE, and HL register pairs are all affected.
;
;     NOTE ****** This procedure is not re-entrant.
;
; For each of the routines listed above, there is an additional entry
; point which allows the routine to be called using the standard pascal
; 64000 parameter passing protocol and passing the parameters through
; a common data area into the registers. It should not be noted that use of
; these routines in this fashion any cause problems in an interrupt
; driven environment. They should therefor be used with care. If the
; name of a given routine is Name, the name of the additional entry point
; is NameQ for the actual routine named NameP when called through the
; jump table in OS ROM. "Q" entry points destroy all registers.
; ************************************************************************

; ***************************** DICTIONARY *********************************
; #Define
DATA_PORT  EQU  0BEH
CTRL_PORT  EQU  0BFH
;
; #External
; EXT  PARAM_
;
; #Common
; PARAM_AREA  DEFS   6
;
; #Global
; GLB  DATA_PORT,CTRL_PORT
; GLB  REG_WRITE_P ; ??
; GLB  REG_WRITE,VRAM_WRITE,VRAM_READ
; GLB  REG_WRITEQ,VRAM_WRITEQ,VRAM_READQ
; ************************************************************************

; ********************** PROCEDURES AND FUNCTIONS ************************
;
; PROCEDURE REG_WRITEQ (REGISTER:BYTE;VALUE:BYTE)
; - REGISTER is passed to B register.
; - VALUE is passed to C register.
; - DESTROYS: A

REG_WRITE_P:
       dw     00002H
       dw     00001H
       dw     00001H
;
REG_WRITEQ:
       ld     bc,REG_WRITE_P
       ld     de,PARAM_AREA
       call   PARAM_
       ld     hl,(PARAM_AREA)
       ld     c,h
       ld     b,l
REG_WRITE:
       ld     a,c
       out    (CTRL_PORT),a
       ld     a,b
       add    a,080H
       out    (CTRL_PORT),a
       ld     a,b
       cp     000H
       jr     nz,NOT_REG_0
       ld     a,c
       ld     (VDP_MODE_WORD),a
```

```
NOT_REG_0:
       ld     a,b
       cp     001H
       jr     nz,NOT_REG_1
       ld     a,c
       ld     (VDP_MODE_WORD+1),a
NOT_REG_1:
       ret

; PROCEDURE VRAM_WRITEQ (VAR_DATA:BUFFER;DEST:INTEGER;COUNT:INTEGER)
; - VAR_DATA (pointer to data buffer) is passed in HL
; - DEST is passed in DE
; - COUNT is passed in BC
; - DESTROYS: ALL

VRAM_WRITE_P:
       dw     00003H
       dw     0FFFEH
       dw     00002H
       dw     00002H
;
WRITE_VRAMQ:
       ld     bc,VRAM_WRITE_P
       ld     de,PARAM_AREA
       call   PARAM_
       ld     hl,(PARAM_AREA)
       ld     de,(PARAM_AREA+2)
       ld     bc,(PARAM_AREA+4)
VRAM_WRITE:
       push   hl
       push   de
       pop    hl
       ld     de,04000H
       add    hl,de
       ld     a,l
       out    (CTRL_PORT),a
       ld     a,h
       out    (CTRL_PORT),a
       push   bc
       pop    de
       pop    hl
       ld     c, DATA_PORT
       ld     b,e
OUTPUT_LOOP:
       outi
       nop
       nop
       jp     nz,OUTPUT_LOOP
       dec    d
       jp     m,END_OUTPUT
       jr     nz,OUTPUT_LOOP
END_OUTPUT:
       ret

; PROCEDURE VRAM_READQ (VAR_DATA:BUFFER;SRCE:INTEGER;COUNT:INTEGER)
; - VAR_DATA (pointer to data buffer) is passed in HL
; - SRCE is passed in DE
; - COUNT is passed in BC
; - DESTROYS: ALL

VRAM_READ_P:
       dw     00003H
       dw     0FFFEH
```

```
        dw      00002H
        dw      00002H
;
READ_VRAMQ:
        ld      bc,VRAM_READ_P
        ld      de,PARAM_AREA
        call    PARAM_
        ld      hl,(PARAM_AREA)
        ld      de,(PARAM_AREA+2)
        ld      bc,(PARAM_AREA+4)
VRAM_READ:
        ld      a,e
        out     (CTRL_PORT),a
        ld      a,d
        out     (CTRL_PORT),a
        push    bc
        pop     de
        ld      c, DATA_PORT
        ld      b,e
INPUT_LOOP:
        ini
        nop
        nop
        jp      nz,INPUT_LOOP
        dec     d
        jp      m,END_INPUT
        jr      nz,INPUT_LOOP
END_INPUT:
        ret
;
REG_READ:
        in      a,(0BFH)
        ret
```

GRAPHICS PRIM PKG

```
; This is a package of routines that allow applications programmers to
; operate on shape generators. Each of them takes, as inputs, an area
; in one of the generator tables in which the generators to be operated
; upon reside, a count of the generators to be used, and an area of the
; same table into which the results are to be put. The only RAM area they
; is in the WORK_BUFFER a pointer to which is declared as an external
; and defined in the cartridge.

; ************ NOTE:                                        ************
; ************      THESE ROUTINES WRITE TO AND READ       ************
; ************      WITHOUT POSSIBILITY OF DEFERAL         ************
; ************      AND SHOULD NOT BE USED IN ANY          ************
; ************      SITUATION WHERE THEY MAY BE            ************
; ************      INTERRUPTED.                           ************
; ****************************************************************

; #Define
; TRUE          EQU  1
; FALSE         EQU  0
; PATTERN_GEN   EQU  3
; COLOR_TABLE   EQU  4
;
; #External
; EXT  WORK_BUFFER
;
; #Global
; GLB  RFLCT_VERT,RFLCT_HOR,ROT_90,ENLRG

;
; PROCEDURE:
; REFLECT_VERTICAL (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))
;
; It reflects each of a block of generators from VRAM around the vertical axis.
; If the generators are from the pattern plane and the graphics mode is 2, then
; the routine also copies the corresponding color generators., otherwise is
; assumes that the color data has already been set up.
;
; BEGIN REFLECT_VERTICAL
;
RFLCT_VERT:
       ld     ix,RLFCT_VERT_
       jr     CONTINUE_GRAPHICS
;
; PROCEDURE:
; REFLECT_HORIZONTAL (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))
;
; It reflects each of a block of generators from VRAM around the horizontal
; axis. If the generators are from the pattern plane and the graphics mode is
; 2, then it reflects the corresponding color generators as well.
;
; BEGIN REFLECT_HORIZONTAL
;
RFLCT_HOR:
       ld     ix,RLFCT_HOR_
       jr     CONTINUE_GRAPHICS
;
; PROCEDURE:
; ROTATE_90 (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))
;
```

```
; It rotates each of a block of generators from VRAM 90 degrees clockwise.
; If the generators are from the pattern plane and the graphics mode is 2, then
: it copies the corresponding color entries as well.
;
; BEGIN ROTATE_90
;
ROT_90:
       ld     ix,ROT_90_
       jr     CONTINUE_GRAPHICS
;
; PROCEDURE:
; ENLARGE (TABLE_CODE(A),SOURCE(DE),DESTINATION(HL),COUNT(BC))
;
; It takes each of a block of generators and enlarges it into a block of four
; generators where each pixel of the original generator is expanded to four
; pixels in the new ones. If the generators are from the pattern plane and the
; graphics mode is 2 then it also quadruples each of the corresponding color
; generators as well.
;
; BEGIN ENLARGE
;
ENLRG:
       ld     ix,ENLRG_
CONTINUE_GRAPHICS:
       exx
       ex     af,af'
       push   ix
MAIN_LOOP:
       ex     af,af'
       push   af
       ex     af,af'
       pop    af
       exx
       push   de
       exx
       pop    de
       ld     iy,00001H
       ld     hl,(WORK_BUFFER)
       call   GET_VRAM_
       pop    ix
       push   ix
       jp     (ix)          ;INFO: index jump
;
RETURN_HERE:
       inc    de
       dec    bc
       ld     a,b
       or     c
       exx
       jr     nz,MAIN_LOOP
       pop    ix
       ret
;
; RLFCT_VERT_
; Desc.: Operations specific to the REFLECT_VERTICAL routine
;
RLFCT_VERT_:
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       push   hl
       pop    de
       add    hl,bc
       ex     de,hl
```

```
       call   MIRROR_L_R
       call   PUT_TABLE
       call   COLOR_TEST
       cp     001H
       jr     nz,END_IF_1_GRAPHICS
       call   GET_COLOR
       call   PUT_COLOR
END_IF_1_GRAPHICS:
       exx
       inc    hl
       jr     RETURN_HERE
;
; RLFCT_HOR_
; Desc.: Operations specific to the REFLECT_HORIZONTAL routine
;
RLFCT_HOR_:
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       push   hl
       pop    de
       add    hl,bc
       ex     de,hl
       call   MIRROR_U_D
       call   PUT_TABLE
       call   COLOR_TEST
       cp     001H
       jr     nz,END_IF_2_GRAPHICS
       call   GET_COLOR
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       push   hl
       pop    de
       add    hl,bc
       ex     de,hl
       call   MIRROR_U_D
       call   PUT_COLOR
END_IF_2_GRAPHICS:
       exx
       inc    hl
       jr     RETURN_HERE
;
; ROT_90_
; Desc.: Operations specific to the ROTATE_90 routine
;
ROT_90_:
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       push   hl
       pop    de
       add    hl,bc
       ex     de,hl
       call   ROTATE
       call   PUT_TABLE
       call   COLOR_TEST
       cp     001H
       jr     nz,END_IF_3_GRAPHICS
       call   GET_COLOR
       call   PUT_COLOR
END_IF_3_GRAPHICS:
       exx
       inc    hl
       jp     RETURN_HERE
;
```

```
; ENLRG_
; Desc.: Operations specific to the ENLARGE routine
;
ENLRG_:
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       push   hl
       pop    de
       add    hl,bc
       ex     de,hl
       call   MAGNIFY
       ex     af,af'
       push   af
       ex     af,af'
       pop    af
       exx
       push   hl
       exx
       pop    de
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       add    hl,bc
       ld     iy,00004H
       call   PUT_VRAM_
       call   COLOR_TEST
       cp     001H
       jr     nz,END_IF_4_GRAPHICS
       call   GET_COLOR
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       push   hl
       pop    de
       add    hl,bc
       ex     de,hl
       call   QUADRUPLE
       ld     a,004H
       exx
       push   hl
       exx
       pop    de
       ld     hl,(WORK_BUFFER)
       ld     bc,00008H
       add    hl,bc
       ld     iy,00004H
       call   PUT_VRAM_
END_IF_4_GRAPHICS:
       exx
       inc    hl
       inc    hl
       inc    hl
       inc    hl
       jp     RETURN_HERE
;
; COLOR_TEST
; Desc.: Tests whether pattern generators are being manipulated and wheter the
;        graphics mode is 2. If so the above routines need to deal with the
;        color generators that correspond to the pattern generators they are
;        operating on.
; Input: no
; Output: A = 1 if true, 0 if not
;
COLOR_TEST:
       ex     af,af'
```

```
        push   af
        ex     af,af'
        pop    af
        cp     003H
        jr     nz,EXIT_FALSE
        ld     hl,VDP_MODE_WORD
        bit    1,(hl)
        jr     z,EXIT_FALSE
        ld     a,001H
        ret
;
EXIT_FALSE:
        ld     a,000H
        ret
;
; PUT_TABLE
; Desc.: Puts the contents of WORK_BUFFER[8..15]
;        in vram at the given destination.
;
PUT_TABLE:
        ex     af,af'
        push   af
        ex     af,af'
        pop    af
        exx
        push   hl
        exx
        pop    de
        ld     hl,(WORK_BUFFER)
        ld     bc,00008H
        add    hl,bc
        ld     iy,00001H
        call   PUT_VRAM_
        ret
;
; GET_COLOR
; Desc.: Gets the color information from the appropriate place in vram.
;
GET_COLOR:
        ld     a,004H
        exx
        push   de
        exx
        pop    de
        ld     hl,(WORK_BUFFER)
        ld     iy,00001H
        call   GET_VRAM_
        ret
;
; PUT_COLOR
; Desc.: Puts the color information in the appropriate place in vram.
;
PUT_COLOR:
        ld     a,004H
        exx
        push   hl
        exx
        pop    de
        ld     hl,(WORK_BUFFER)
        ld     iy,00001H
        call   PUT_VRAM_
        ret
```

EXPANSION ROUTINES

```
; The routines in this module take a single 8-byte block as input and
; produce 4 8-byte blocks as output. They perform a 2-to-1 expansion
; and a simple quadrupling operation respectively

; #Define
; BYTE_COUNT    EQU BC register
; SOURCE        EQU IX register
; DESTINATION   EQU IY register
;
; #Global
; GLB  MAGNIFY,QUADRULE

;
; MAGNIFY
; Desc.: Perform a 2-to-1 expansion on an 8-byte block of data.
; Input: HL = Source pointer, DE = Destination pointer
; Destroy: IX, IY, AF, BC, DE, HL
;
MAGNIFY:
       push   hl
       pop    ix
       push   de
       pop    iy
       ld     bc,00008H
MAG_LOOP:
       ld     a,(ix+000H)
       inc    ix
       ld     d,a
       ld     e,004H
EXP_1:
       rl     a
       rl     h
       rl     d
       rl     h
       dec    e
       jr     nz,EXP_1
       ld     e,004H
EXP_2:
       rl     a
       rl     l
       rl     d
       rl     l
       dec    e
       jr     nz,EXP_2
       ld     (iy+000H),h
       ld     (iy+010H),l
       inc    iy
       ld     (iy+000H),h
       ld     (iy+010H),l
       inc    iy
       dec    bc
       ld     a,c
       or     b
       jr     nz,MAG_LOOP
       ret
;
; QUADRUPLE
; Desc.: Perform a quadrupling on an 8-byte block of data.
; Input: HL = Source pointer, DE = Destination pointer
```

```
; Destroy: AF, BC, DE
;
QUADRUPLE:
       ld     bc,00010H
       push   hl
QUAD_LOOP:
       ld     a,(hl)
       inc    hl
       ld     (de),a
       inc    de
       ld     (de),a
       inc    de
       dec    bc
       ld     a,c
       cp     008H
       jr     nz,SKIPZZ
       pop    hl
SKIPZZ:
       ld     a,c
       or     b
       jr     nz,QUAD_LOOP
       ret
```

MIRROR/ROTATE RTN

```
; The routines in this file take a single 8-byte block as input
; and operate on it producing a single 8-byte block as output
; They perform mirroring arround the vertical axis, mirroring
; arround the horizontal axis, and 90 degree rotation

; #Global
; GLB  MIRROR_L_R,MIRROR_U_D,ROTATE

;
; MIRROR_L_R
; Desc.: Reflets an 8x8 pixel data block arround the vertical axis.
; Input: HL = Source pointer, DE = Destination pointer
; Destroy: AF, BC, DE, HL
;
MIRROR_L_R:
       ld     bc,00008H
MIR_L_R10:
       ld     b,(hl)
       ld     a,080H
MIR_L_R20:
       rl     b
       rra
       jr     nc,MIR_L_R20
       ld     (de),a
       inc    hl
       inc    de
       dec    c
       jr     nz,MIR_L_R10
       ret
;
; ROTATE
; Desc.: Rotate object 90 degrees (clockwise).
; Input: HL = Source pointer, DE = Destination pointer
; Destroy: IX, AF, BC, DE, HL
;
ROTATE:
       push   hl
       pop    ix
       ex     de,hl
       ld     bc,00008H
TRANSP_10:
       rl     (ix+000H)
       rr     (hl)
       rl     (ix+001H)
       rr     (hl)
       rl     (ix+002H)
       rr     (hl)
       rl     (ix+003H)
       rr     (hl)
       rl     (ix+004H)
       rr     (hl)
       rl     (ix+005H)
       rr     (hl)
       rl     (ix+006H)
       rr     (hl)
       rl     (ix+007H)
       rr     (hl)
       inc    hl
       dec    c
```

```
        jr      nz,TRANSP_10
        ret
;
; MIRROR_U_D
; Desc.: Reflets 8x8 pixel block arround the horizontal axis.
; Input: HL = Source pointer, DE = Destination pointer
; Destroy: AF, BC, DE, HL
;
MIRROR_U_D:
        ld      bc,00007H
        add     hl,bc
        inc     bc
REFLECT_LOOP:
        ld      a,(hl)
        ld      (de),a
        inc     de
        dec     hl
        dec     bc
        ld      a,b
        or      c
        jr      nz,REFLECT_LOOP
        ret
;
; Modified february 14, 1983.
; Filler locations were changed to 0FFH to reflect OS_7PRIME
;
filler_1f5d:
        db      0FFH
        db      0FFH
        db      0FFH
        db      0FFH
```

JUMP TABLE

```
; This is the jump table to be used in accessing code residing in the OS ROM.
; This table must have its origin redefined to account for growth. Pile new
; routines at the beginning of the table making sure to increment the
; NO_OF_ROUTINES value.
;
; NOTE ****
;             **** NO DELETIONS SHOULD BE MADE FROM ****
;             **** THIS TABLE                       ****
; #Define
ROM_END         EQU 2000H   ; This is the end of OS ROM
NO_OF_ROUTINES EQU 53       ; This number keeps count of the number of routines
                            ; accessed through the jump table.
; #Origin
JUMP_TABLE    ORG    ROM_END-(NO_OF_ROUTINES*3)

PLAY_SONGS:
       jp     PLAY_SONGS_          ; ($1F61) See page 86
ACTIVATEP:
       jp     ACTIVATEQ            ; ($1F64) See page 93
PUTOBJP:
       jp     PUTOBJQ              ; ($1F67) See page 102
REFLECT_VERTICAL:
       jp     RFLCT_VERT           ; ($1F6A) See page 168
REFLECT_HORIZONTAL:
       jp     RFLCT_HOR            ; ($1F6D) See page 168
ROTATE_90:
       jp     ROT_90               ; ($1F70) See page 169
ENLARGE:
       jp     ENLRG                ; ($1F73) See page 169
CONTROLLER_SCAN:
       jp     CONT_SCAN            ; ($1F76) See page 140
DECODER:
       jp     DECODER_             ; ($1F79) See page 141
GAME_OPT:
       jp     GAME_OPT_            ; ($1F7C) See page 155
LOAD_ASCII:
       jp     LOAD_ASCII_          ; ($1F7F) See page 153
FILL_VRAM:
       jp     FILL_VRAM_           ; ($1F82) See page 152
MODE_1:
       jp     MODE_1_              ; ($1F85) See page 153
UPDATE_SPINNER:
       jp     UPDATE_SPINNER_      ; ($1F88) See page 141
INIT_TABLEP:
       jp     INIT_TABLEQ          ; ($1F8B) See page 158
GET_VRAMP:
       jp     GET_VRAMQ            ; ($1F8E) See page 159
PUT_VRAMP:
       jp     PUT_VRAMQ            ; ($1F91) See page 161
INIT_SPR_ORDERP:
       jp     INIT_SPR_ORDERQ      ; ($1F94) See page 162
WR_SPR_NM_TBLP:
       jp     WR_SPR_NM_TBLQ       ; ($1F97) See page 163
INIT_TIMERP:
       jp     INIT_TIMERQ          ; ($1F9A) See page 133
FREE_SIGNALP:
       jp     FREE_SIGNALQ         ; ($1F9D) See page 133
REQUEST_SIGNALP:
       jp     REQUEST_SIGNALQ      ; ($1FA0) See page 135
```

```
TEST_SIGNALP:
       jp     TEST_SIGNALQ        ; ($1FA3) See page 137
WRITE_REGISTERP:
       jp     REG_WRITEQ          ; ($1FA6) See page 165
WRITE_VRAMP:
       jp     WRITE_VRAMQ         ; ($1FA9) See page 166
READ_VRAMP:
       jp     READ_VRAMQ          ; ($1FAC) See page 167
INIT_WRITERP:
       jp     INIT_QUEUEQ         ; ($1FAF) See page 100
SOUND_INITP:
       jp     INIT_SOUNDQ         ; ($1FB2) See page 80
PLAY_ITP:
       jp     JUKE_BOXQ           ; ($1FB5) See page 82
INIT_TABLE:
       jp     INIT_TABLE_         ; ($1FB8) See page 158
GET_VRAM:
       jp     GET_VRAM_           ; ($1FBB) See page 160
PUT_VRAM:
       jp     PUT_VRAM_           ; ($1FBE) See page 162
INIT_SPR_ORDER:
       jp     INIT_SPR_ORDER_     ; ($1FC1) See page 162
WR_SPR_NM_TBL:
       jp     WR_SPR_NM_TBL_      ; ($1FC4) See page 163
INIT_TIMER:
       jp     INIT_TIMER_         ; ($1FC7) See page 133
FREE_SIGNAL:
       jp     FREE_SIGNAL_        ; ($1FCA) See page 133
REQUEST_SIGNAL:
       jp     REQUEST_SIGNAL_     ; ($1FCD) See page 135
TEST_SIGNAL:
       jp     TEST_SIGNAL_        ; ($1FD0) See page 137
TIME_MGR:
       jp     TIME_MGR_           ; ($1FD3) See page 131
TURN_OFF_SOUND:
       jp     ALL_OFF             ; ($1FD6) See page 81
WRITE_REGISTER:
       jp     REG_WRITE           ; ($1FD9) See page 165
READ_REGISTER:
       jp     REG_READ            ; ($1FDC) See page 167
WRITE_VRAM:
       jp     VRAM_WRITE          ; ($1FDF) See page 166
READ_VRAM:
       jp     VRAM_READ           ; ($1FE2) See page 167
INIT_WRITER:
       jp     INIT_QUEUE          ; ($1FE5) See page 100
WRITER:
       jp     WRITER_             ; ($1FE8) See page 100
POLLER:
       jp     POLLER_             ; ($1FEB) See page 142
SOUND_INIT:
       jp     INIT_SOUND          ; ($1FEE) See page 80
PLAY_IT:
       jp     JUKE_BOX            ; ($1FF1) See page 82
SOUND_MAN:
       jp     SND_MANAGER         ; ($1FF4) See page 83
ACTIVATE:
       jp     ACTIVATE_           ; ($1FF7) See page 93
PUTOBJ:
       jp     PUTOBJ_             ; ($1FFA) See page 102
RAND_GEN:
       jp     RAND_GEN_           ; ($1FFD) See page 68
```

APPENDIX

OS 7' JUMP TABLE

These symbols are entry points to OS 7' routines. They can be dirrectly called by programmers.

Legend:
P (at the end): special entry points for Pascal programs.

```
1F61 > 0300 : PLAY_SONGS            1FB2 > 0203 : SOUND_INITP
1F64 > 0488 : ACTIVATEP             1FB5 > 0251 : PLAY_ITP
1F67 > 06C7 : PUTOBJP               1FB8 > 1B08 : INIT_TABLE
1F6A > 1D5A : REFLECT_VERTICAL      1FBB > 1BA3 : GET_VRAM
1F6D > 1D60 : REFLECT_HORIZONTAL    1FBE > 1C27 : PUT_VRAM
1F70 > 1D66 : ROTATE_90             1FC1 > 1C66 : INIT_SPR_ORDER
1F73 > 1D6C : ENLARGE               1FC4 > 1C82 : WR_SPR_NM_TBL
1F76 > 114A : CONTROLLER_SCAN       1FC7 > 0FAA : INIT_TIMER
1F79 > 118B : DECODER               1FCA > 0FC4 : FREE_SIGNAL
1F7C > 1979 : GAME_OPT              1FCD > 1053 : REQUEST_SIGNAL
1F7F > 1927 : LOAD_ASCII            1FD0 > 10CB : TEST_SIGNAL
1F82 > 18D4 : FILL_VRAM             1FD3 > 0F37 : TIME_MGR
1F85 > 18E9 : MODE_1                1FD6 > 023B : TURN_OFF_SOUND
1F88 > 116A : UPDATE_SPINNER        1FD9 > 1CCA : WRITE_REGISTER
1F8B > 1B0E : INIT_TABLEP           1FDC > 1D57 : READ_REGISTER
1F8E > 1B8C : GET_VRAMP             1FDF > 1D01 : WRITE_VRAM
1F91 > 1C10 : PUT_VRAMP             1FE2 > 1D3E : READ_VRAM
1F94 > 1C5A : INIT_SPR_ORDERP       1FE5 > 0664 : INIT_WRITER
1F97 > 1C76 : WR_SPR_NM_TBLP        1FE8 > 0679 : WRITER
1F9A > 0F9A : INIT_TIMERP           1FEB > 11C1 : POLLER
1F9D > 0FB8 : FREE_SIGNALP          1FEE > 0213 : SOUND_INIT
1FA0 > 1044 : REQUEST_SIGNALP       1FF1 > 025E : PLAY_IT
1FA3 > 10BF : TEST_SIGNALP          1FF4 > 027F : SOUND_MAN
1FA6 > 1CBC : WRITE_REGISTERP       1FF7 > 04A3 : ACTIVATE
1FA9 > 1CED : WRITE_VRAMP           1FFA > 06D8 : PUTOBJ
1FAC > 1D2A : READ_VRAMP            1FFD > 003B : RAND_GEN
1FAF > 0655 : INIT_WRITERP
```

GLOBAL OS 7' SYMBOLS

SYMBOLS IN ALPHABETIC ORDER

ADDRESS	NAME	DESCRIPTION
01B1	ADD816	Add signed 8bit value A to 16bit [HL]
0069	AMERICA	60 = NTSC, 50 = PAL
006A	ASCII_TABLE	Pointer to uppercase ASCII pattern
012F	ATN_SWEEP	Attenuation sweep
08C0	CALC_OFFSET	Returns DE := offset for the coordinates (E,D)
8000	CARTRIDGE	Cartridge starting address
8008	CONTROLLER_MAP	Pointer to controller memory map
1D43	CTRL_PORT_PTR	(in READ_VRAM, equal I/O port# BF)
1D47	DATA_PORT_PTR	(in READ_VRAM, equal I/O port# BE)
0190	DECLSN	Decrement low nibble (in UTILITY)
019B	DECMSN	Decrement high nibble (in UTILITY)
73C6	DEFER_WRITES	Boolean flag to defer writes to VRAM
02EE	EFXOVER	(in PROCESS_DATA_AREA to get next note)
1D6C	ENLRG	It's the local name of the ENLARGE routine
00FC	FREQ_SWEEP	Frequency sweep
8024	GAME_NAME	String of ASCII characters
0898	GET_BKGRND	Copy a block of names from VRAM to RAM
801E	IRQ_INT_VECT	Software interrupt vector (RST 38H)
01D5	LEAVE_EFFECT	Called by a special sound effect function when done
8002	LOCAL_SPR_TABLE	Pointer to sprite name table
01A6	MSNTOLSN	Copy high nibble to low nibble (in UTILITY)
73C7	MUS_SPRITES	Boolean flag to sprite multiplexing
8021	NMI_INT_VECT	NMI soft vector
006C	NUMBER_TABLE	Pointer to numbers 0-9 pattern
080B	PUT_FRAME	Copy a block of names to VRAM
07E8	PX_TO_PTRN_POS	Pixel to pattern plane position
73C9	RAND_NUM	Pointer to pseudo random number value
800F	RST_10H_RAM	Reset 10 soft vector
8012	RST_18H_RAM	Reset 18 soft vector
8015	RST_20H_RAM	Reset 20 soft vector
8018	RST_28H_RAM	Reset 28 soft vector
801B	RST_30H_RAM	Reset 30 soft vector
800C	RST_8H_RAM	Reset 8 soft vector
8004	SPRITE_ORDER	Pointer to sprite order table
73B9	STACK	Stack pointer address
800A	START_GAME	Pointer to game start code
73C3	VDP_MODE_WORD	Copy of the first two VDP registers
73C5	VDP_STATUS_BYTE	Contents of default NMI handler
8006	WORK_BUFFER	Pointer to temporary storage in RAM

SYMBOLS ORDERED BY ADDRESSES

ADDRESS	NAME	DESCRIPTION
0069	AMERICA	60 = NTSC, 50 = PAL
006A	ASCII_TABLE	Pointer to uppercase ASCII pattern
006C	NUMBER_TABLE	Pointer to numbers 0-9 pattern
00FC	FREQ_SWEEP	Frequency sweep
012F	ATN_SWEEP	Attenuation sweep
0190	DECLSN	Decrement low nibble (in UTILITY)
019B	DECMSN	Decrement high nibble (in (UTILITY)
01A6	MSNTOLSN	Copy high nibble to low nibble (in UTILITY)
01B1	ADD816	Add signed 8bit value A to 16bit [HL]
01D5	LEAVE_EFFECT	Called by a special sound effect function when done
02EE	EFXOVER	(in PROCESS_DATA_AREA to get next note)
07E8	PX_TO_PTRN_POS	Pixel to pattern plane position
080B	PUT_FRAME	Copy a block of names to VRAM
0898	GET_BKGRND	Copy a block of names from VRAM to RAM
08C0	CALC_OFFSET	Returns DE := offset for the coordinates (E,D)
1D43	CTRL_PORT_PTR	(in READ_VRAM, equal I/O port# BF)
1D47	DATA_PORT_PTR	(in READ_VRAM, equal I/O port# BE)
1D6C	ENLRG	It's the local name of the ENLARGE routine
73B9	STACK	Stack pointer address
73C3	VDP_MODE_WORD	Copy of the first two VDP registers
73C5	VDP_STATUS_BYTE	Contents of default NMI handler
73C6	DEFER_WRITES	Boolean flag to defer writes to VRAM
73C7	MUS_SPRITES	Boolean flag to sprite multiplexing
73C9	RAND_NUM	Pointer to pseudo random number value
8000	CARTRIDGE	Cartridge starting address
8002	LOCAL_SPR_TABLE	Pointer to sprite name table
8004	SPRITE_ORDER	Pointer to sprite order table
8006	WORK_BUFFER	Pointer to temporary storage in RAM
8008	CONTROLLER_MAP	Pointer to controller memory map
800A	START_GAME	Pointer to game start code
800C	RST_8H_RAM	Reset 8 soft vector
800F	RST_10H_RAM	Reset 10 soft vector
8012	RST_18H_RAM	Reset 18 soft vector
8015	RST_20H_RAM	Reset 20 soft vector
8018	RST_28H_RAM	Reset 28 soft vector
801B	RST_30H_RAM	Reset 30 soft vector
801E	IRQ_INT_VECT	Software interrupt vector (RST 38H)
8021	NMI_INT_VECT	NMI soft vector
8024	GAME_NAME	String of ASCII characters

Note : Programmers are responsible to use these symbols properly.

MEMORY MAP

From ADAM™ Technical Reference Manual

Note for ADAM users: The ADAM computer can be reset in either computer mode or in game mode. When the cartridge (or ColecoVision) reset switch is pressed, ADAM resets to game mode. In this mode, 32K of cartridge ROM are switched into the upper bank of memory, and OS 7' plus 24K of intrinsic RAM are switched into the lower bank of memory. So, it's possible to create a ColecoVision game with additional options if plugged into an ADAM computer and then use the extra RAM space and the ADAM peripherics.

COLECOVISION GENERAL MEMORY MAP

From ColecoVision FAQ

ADDRESS	DESCRIPTION
0000-1FFF	ColecoVision BIOS OS 7'
2000-5FFF	Expansion port
6000-7FFF	1K RAM mapped into 8K. (7000-73FF)
8000-FFFF	Game cartridge

GAME CARTRIDGE HEADER

From The Absolute OS 7' Listing

ADDRESS	NAME	DESCRIPTION
8000-8001	CARTRIDGE	Test bytes. Must be AA55 or 55AA.
8002-8003	LOCAL_SPR_TABLE	Pointer to RAM copy of the sprite name table.
8004-8005	SPRITE_ORDER	Pointer to RAM sprite order table.
8006-8007	WORK_BUFFER	Pointer to free buffer space in RAM.
8008-8009	CONTROLLER_MAP	Pointer to controller memory map.
800A-800B	START_GAME	Pointer to the start of the game.
800C-800E	RST_8H_RAM	Restart 8h soft vector.
800F-8011	RST_10H_RAM	Restart 10h soft vector.
8012-8014	RST_18H_RAM	Restart 18h soft vector.
8015-8017	RST_20H_RAM	Restart 20h soft vector.
8018-801A	RST_28H_RAM	Restart 28h soft vector.
801B-801D	RST_30H_RAM	Restart 30h soft vector.
801E-8020	IRQ_INT_VECTOR	Mask-able interrupt soft vector (38h).
8021-8023	NMI_INT_VECTOR	Non mask-able interrupt (NMI) soft vector.
8024-80XX	GAME_NAME	String with two delimiters "/" as "LINE2/LINE1/YEAR"

COMPLET OS 7' RAM MAP

ADDRESS	NAME	DESCRIPTION
7020-7021	PTR_LST_OF_SND_ADDRS	Pointer to list (in RAM) of sound addrs
7022-7023	PTR_TO_S_ON_0	Pointer to song for noise
7024-7025	PTR_TO_S_ON_1	Pointer to song for channel#1
7026-7027	PTR_TO_S_ON_2	Pointer to song for channel#2
7028-7029	PTR_TO_S_ON_3	Pointer to song for channel#3
702A	SAVE_CTRL	CTRL data (byte)
73B9	STACK	Beginning of the stack
73BA-73BF	PARAM_AREA	Common passing parameters area (PASCAL)
73C0-73C1	TIMER_LENGTH	Length of timer
73C2	TEST_SIG_NUM	Signal Code
73C3-73C4	VDP_MODE_WORD	Copy of data in the 1st 2 VDP registers
73C5	VDP_STATUS_BYTE	Contents of default NMI handler
73C6	DEFER_WRITES	Deferred sprites flag
73C7	MUX_SPRITES	Multiplexing sprites flag
73C8-73C9	RAND_NUM	Pseudo random number value
73CA	QUEUE_SIZE	Size of the deferred write queue
73CB	QUEUE_HEAD	Indice of the head of the write queue
73CC	QUEUE_TAIL	Indice of the tail of the write queue
73CD-73CE	HEAD_ADDRESS	Address of the queue head
73CF-73D0	TAIL_ADDRESS	Address of the queue tail
73D1-73D2	BUFFER	Buffer pointer to deferred objects
73D3-73D4	TIMER_TABLE_BASE	Timer base address
73D5-73D6	NEXT_TIMER_DATA_BYTE	Next available timer address
73D7-73EA	DBNCE_BUFF	Debounce buffer. 5 pairs (old and state) of fire, joy, spin, arm and kbd for each player.
73EB	SPIN_SW0_CT	Spinner counter port#1
73EC	SPIN_SW1_CT	Spinner counter port#2
73ED	-	(reserved)
73EE	S0_C0	Segment 0 data, Controller port #1
73EF	S0_C1	Segment 0 data, Controller port #2
73F0	S1_C0	Segment 1 data, Controller port #1
73F1	S1_C1	Segment 1 data, Controller port #2
73F2-73FB	VRAM_ADDR_TABLE	Block of VRAM table pointers
73F2-73F3	SPRITENAMETBL	Sprite name table offset
73F4-73F5	SPRITEGENTBL	Sprite generator table offset
73F6-73F7	PATTERNNAMETBL	Pattern name table offset
73F8-73F9	PATTERNGENTBL	Pattern generator table offset
73FA-73FB	COLORTABLE	Color table offset
73FC-73FD	SAVE_TEMP	(no more used - in VRAM routines)
73FE-73FF	SAVED_COUNT	Copy of COUNT for PUT_VRAM & GET_VRAM

OS 7' AND EOS SIMILARITIES

Based on The Hackers' Guide to ADAM™ and The Absolute OS 7' Listing
List of OS 7' symbols (calls & variables) similar to the EOS ones. For ColecoVision/ADAM programmers.

EOS	OS 7'	SYMBOL
FD1A	1FDF	WRITE_VRAM
FD1D	1FE2	READ_VRAM
FD20	1FD9	WRITE_REGISTER
FD23	1FDC	READ_REGISTER
FD26	1F82	FILL_VRAM
FD29	1FB8	INIT_TABLE
FD2C	1FBE	PUT_VRAM
FD2F	1FBB	GET_VRAM
FD32	08C0	CALC_OFFSET
FD35	07E8	PX_TO_PTRN_POS
FD38	1F7F	LOAD_ASCII
FD3B	1FC4	WR_SPR_NM_TBL
FD3E	1F76	CONTROLLER_SCAN
FD41	1F88	UPDATE_SPINNER
FD44	0190	DECLSN
FD47	019B	DECMSN
FD4A	01A6	MSNTOLSN
FD4D	01B1	ADD816
FD50	1FEE	SOUND_INIT
FD53	1FD6	TURN_OFF_SOUND
FD56	1FF1	PLAY_IT
FD59	1F61,1FF4	PLAY_SONGS + SOUND_MAN
FD5C	01D5? 02EE?	LEAVE_EFFECT? or EFXOVER?
FD61-FD62	73C3-73C4	VDP_MODE_WORD
FD63	73C5	STATUS_MODE_BYTE
FD64-FD65	73F2-73F3	SPRITENAMETBL
FD66-FD67	73F4-73F5	SPRITEGENTBL
FD68-FD69	73F6-73F7	PATTRNNAMETBL
FD6A-FD6B	73F8-73F9	PATTRNGENTBL
FD6C-FD6D	73FA-73FB	COLORTABLE
FE58	73EB	SPIN_SW0_CT
FE59	73EC	SPIN_SW1_CT
FE5A-FE65	????-????	*CONTROLLER_MAP
FE6E-FE6F	7020-7021	PTR_TO_LST_OF_SND_ADDRS
FE70-FE71	7024-7025	PTR_TO_S_ON_1 **NOT SURE**
FE72-FE73	7026-7027	PTR_TO_S_ON_2 **NOT SURE**
FE74-FE75	7028-7029	PTR_TO_S_ON_3 **NOT SURE**
FE76-FE77	7022-7023	PTR_TO_S_ON_0 **NOT SURE**
FE78	702A	SAVE_CTRL

* CONTROLLER_MAP (in the cartridge header) points to the controller memory map used by the bios routine POLLER. The joystick data are decoded into 12 bytes : player#1 enable, player#2 enable, p1 fire, p1 joy, p1 spinner, p1 arm, p1 keypad, p2 fire, p2 joy, p2 spinner, p2 arm, p2 keypad.

Z80 I/O PORTS ASSIGNMENTS

Extracted from ADAM™ Technical Reference Manual

Note: Ports # 000H through 07FH are reserved for the ADAM computer. They are mentionned here for those who wants to elaborate special Coleco games with extra options when plugged in the ADAM.

Video Display Processor

Data port	0BEH	
Register port	0BFH	

Sound Generator

Data port	0FFH	(write-only)

Game Controller

Strobe Set port	080H	(write-only)
Strobe Reset port	0C0H	(write-only)
Controller#1 port	0FCH	(read-only)
Controller#2 port	0FFH	(read-only)

MODEM

Data port	05EH	
Control port	05FH	
Auto Dialer	01EH	

Expansion connector #2

Data port	04FH	

Memory Map

Control port	07FH	

Network reset

	03FH	(Performed by setting and resetting bit 0)

EOS enable

	03FH	(Performed by setting bit 1)

EOS disable

	03FH	(Performed by resetting bit 1)

GAME CONTROLLERS

From ADAM™ Technical Reference Manual

The game controller contains an 8-position joystick, two push buttons (Fire and Arm) and a 12-key keypad. Remark: Extra push buttons (Fire 3 and Fire 4) were done after the Coleco bios for the SuperAction controllers, it's why they are not decoded by calling DECODER or POLLER. The information from a controller is read by the CPU on eight input lines through a single port. Once a port has been read, the input data must be decoded. See CONT_SCAN (page 140) in OS 7' Listing for details.

CONTROLLER CONFIGURATION

	D_6	D_5	INT	D_3	D_2	D_1	D_0	
Fire	X							Common 0
North				X				Enabled
N-E				X	X			By ports
East					X			FD, FF
S-E					X	X		
South						X		
S-W						X	X	
West							X	
N-W				X			X	
Arm (Fire 2)	X							Common 1
2				X				Enabled
3						X	X	By ports
6							X	FC, FE
9					X			
8				X	X	X		
7				X		X		
4				X	X		X	
1						X		
5				X	X			
0					X		X	
*					X	X		
#				X			X	
Fire 3					X	X	X	
Fire 4				X		X	X	
Spinner A			X					Always
Spinner B (presently used in Expansion Mod #2)		X						Function

1	2	3
4	5	6
7	8	9
*	0	#

Remark : D_4 is named INT in the ColecoVision official documentations, including the absolute OS 7' bios listing.

Note : When a spinner is spinning, INT bit (Spinner A) is reset and D_5 bit (Spinner B) is set or reset depending on the way the spinner is spinning.

SOUND GENERATOR

From Daniel Bienvenu's CV programming documentation.
For more technical information, read Texas Instrument SN76489AN.

The ColecoVision uses the Texas Instruments SN76489A sound generator chip as the output port 0ffh. It contains three programmable tone generators, each with its own programmable attenuator, and a noise source with its own attenuator.

TONE GENERATORS

Each tone generator consists of a frequency synthesis section requiring 10 bits of information to define half the period of the desired frequency. The frequency can be calculated with the following formula:

frequency = 3.579MHz / 32n, where n is the 10-bit value.

NOISE GENERATOR

The noise generator consists of a noise source that is a shift register with an exclusive OR feedback network.

Noise Feedback Control:

Feedback (FB)	Configuration
0	Periodic Noise "buzz"
1	White Noise "shhh"

Noise Generator Frequency Control:

NF1	NF0	Shift Rate
0	0	N/512
0	1	N/1024
1	0	N/2048
1	1	Tone gen. #3 output

CONTROL REGISTERS

The SN76489A has 8 internal registers which are used to control the 3 tone generators and the noise source. During all data transfers to the SN76489A, the first byte contains a 3 bits field which determines the channel and the control/attenuation. The channel codes are shown below:

R1	R0	Destination Control Register
0	0	Tone 1
0	1	Tone 2
1	0	Tone 3
1	1	Noise

The output of the frequency flip-flop feeds into a 4 stage attenuator. The attenuator values, along with their bit position in the data word, are shown below. Multiple attenuation control bits may be true simultaneously. Thus, the maximum attenuation is 28 db.

A3	A2	A1	A0	Weight
0	0	0	1	2 db
0	0	1	0	4 db
0	1	0	0	8 db
1	0	0	0	16 db
1	1	1	1	OFF

Remark: Louder is the note, lower is the attenuation value.

SOUND CONTROL DATA FORMATS

This is the data formats to be send directly to the sound port 0ffh.

Frequency

1	Reg. Addr			Data			
	R1	R0	0	F3	F2	F1	F0

0	X	Data					
		F9	F8	F7	F6	F5	F4

Noise Control

1	Reg. Addr			X	FB	Shift	
	1	1	0			NF1	NF0

Attenuator

1	Reg. Addr			Data			
	R1	R0	1	A3	A2	A1	A0

SOUND CONTROL NUMBERS TABLE

	Pitch		Volume
	First byte	Second byte	High - Off
Voice 1	80 - 8F	00 - 3F	90 - 9F
Voice 2	A0 - AF	00 - 3F	B0 - BF
Voice 3	CO - CF	00 - 3F	D0 - DF
Noise	E0 - E3 (E3 = Voice 3)	Periodic "buzz"	
	E4 - E7 (E7 = Voice 3)	White "shhh"	
	F0 - FF (FF = Off)	Attenuation	

SOUND DATA FORMAT

From the Hackers' Guide to ADAM™

The following information is the sound data format to be used in the songs encoded in coleco games. Channel # (2-bit values): 00 = Noise, 01 = Tone 1 Generator, 10 = Tone 2 Generator, 11 = Tone 3 Generator. A noisy sound needs a specific sound data format. Numbers 7 to 0 indicate bits position.

REST

(no sound)

7	6	5	4	3	2	1	0
Channel #		1	Length				

SIMPLE NOTE

7	6	5	4	3	2	1	0
Channel #		0	0	0	0	0	0
Frequency (8 lower bits)							
Volume in 4 bits				0	0	Frequency (2 hi-bits)	
Length							

FREQUENCY SWEPT NOTE

7	6	5	4	3	2	1	0
Channel #		0	0	0	0	0	1
Frequency (8 lower bits)							
Volume in 4 bits				0	0	Frequency (2 hi-bits)	
Number of steps in sweep							
Step length				1st step length			
Step size							

VOLUME SWEPT NOTE

7	6	5	4	3	2	1	0
Channel #		0	0	0	0	1	0
Frequency (8 lower bits)							
Volume in 4 bits				0	0	Frequency (2 hi-bits)	
Length of note							
Step size				Number of steps			
Step length				1st step length			

VOLUME AND FREQUENCY SWEPT NOTE

7	6	5	4	3	2	1	0
Channel #		0	0	0	0	1	1
Frequency (8 lower bits)							
Volume in 4 bits				0	0	Frequency (2 hi-bits)	
Number of steps in sweep							
Frequency step length				1st frequency step length			
Frequency step size							
Volume step				Number of volume step			
Volume step length				1st volume step length			

NOISE

7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0
Unused byte							
Volume in 4 bits				0	FB	NF1	NF0
Length of note							

For the special meaning of FB, NF1 and NF0, read section SOUND GENERATOR (page 188).

NOISE VOLUME SWEEP

7	6	5	4	3	2	1	0
0	0	0	0	0	0	1	0
Initial volume				0	FB	NF1	NF0
Length of note							
Step size				Number of volume step			
Step length				1st volume step length			

For the special meaning of FB, NF1 and NF0, read section SOUND GENERATOR (page 188).

SPECIAL EFFECT

7	6	5	4	3	2	1	0
Channel #		0	0	0	1	0	0
Address of the special effect sub-routine (in 2 bytes of course)							

END OR REPEAT

7	6	5	4	3	2	1	0
Channel #		0	1	Repeat?	0	0	0

SOUND TABLES

Based on The Hackers' Guide to ADAM[tm] and The Absolute OS 7' Listing

SONG DATA AREAS IN RAM

With SOUND_INIT, the user initialize the song data areas in the cpu RAM. These song data areas are 10 bytes long and formated like this:

7	6	5	4	3	2	1	0
Channel #		Song number					
Address next note (8 lower bits)							
Address next note (8 higher bits)							
Frequency (8 lower bits)							
Volume in 4 bits				0	0	Frequency (2 hi-bits)	
Length							
Frequency step length				1st frequency step length			
Frequency step size							
Volume step				Number of volume step			
Volume step length				1st volume step length			

Right after the last song data area, a special END (00h) code indicates the end of the song data areas. It's the resposability of the user to allocate enough free RAM space for the song data areas.

SONG TABLE IN ROM

A song table consists of a number of entries. Each entry is composed into two addresses. The first address is a pointer to the song data in ROM encoded into the OS 7' sound data format. The second address is a pointer to the song data area in RAM. The first entry in the song table contains the address to the 1st song data area. More higher is the address of the song data area used by the song, more higher is the priority of this song.

OUTPUT TABLE IN RAM

This table in RAM at 7020-702A contains pointers to the song table and to the active song data areas played through the sound channels. For the structure information of this table, see the section COMPLET OS 7' RAM MAP at page 184.

NOTES AND FREQUENCIES

From Daniel Bienvenu's CV programming documentation.

NOTE, FREQUENCY CONVERSION TABLE

	Hz	HEX	Hz	HEX	Hz	HEX	Hz	HEX	Hz	HEX
A	110.00	3F8	220.00	1FC	440.00	0FE	880.00	07F	1760.0	03F
$A^{\#}/B^{b}$	116.54	3BF	233.08	1DF	466.16	0EF	932.33	077	1864.6	03B
B	123.47	389	246.94	1C4	493.88	0E2	987.77	071	1975.5	038
C	130.81	356	261.63	1AB	523.25	0D5	1046.5	06A	2093.0	035
$C^{\#}/D^{b}$	138.59	327	277.18	193	554.36	0C9	1108.7	064	2217.5	032
D	146.83	2F9	293.66	17C	587.33	0BE	1174.7	05F	2349.3	02F
$D^{\#}/E^{b}$	155.56	2CE	311.13	167	622.25	0B3	1244.5	059	2489.0	02C
E	164.81	2A6	329.63	153	659.25	0A9	1318.5	054	2637.0	02A
F	174.61	280	349.23	140	698.46	0A0	1396.9	050	2793.8	028
$F^{\#}/G^{b}$	185.00	25C	370.00	12E	739.99	097	1480.0	04B	2960.0	025
G	196.00	23A	391.99	11D	783.99	08E	1568.0	047	3136.0	023
$G^{\#}/A^{b}$	207.65	21A	415.30	10D	830.61	086	1661.2	043	3322.4	021

Remark: Higher is the frequency, lower is its hex corresponding value.

SCALES

A Scale is a series of notes which we define as "correct" or appropriate for a song.

Examples of various Scales (Root = "C"):

Name	**C**	**D^{b}**	**D**	**E^{b}**	**E**	**F**	**G^{b}**	**G**	**A^{b}**	**A**	**B^{b}**	**B**
Major	1		2		3	4		5		6		7
Major Triad	1				2			3				
Minor	1		2	3		4		5	6		7	
Minor Triad	1			2				3				
Harmonic Minor	1		2	3		4		5	6			7
Melodic Minor (asc)	1		2	3		4		5		6		7
Melodic Minor (desc)	1		2	3		4		5	6		7	
Enigmatic	1	2			3		4		5		6	7
Flamenco	1	2		3	4	5		6	7		8	

VDP - VIDEO DISPLAY PROCESSOR

From Texas Instrument documentation

VDP has 8 control registers (0-7) and 1 status register.

VDP REGISTERS

Control registers	Register Bits							
	7	6	5	4	3	2	1	0
0	-	-	-	-	-	-	M2	EXT
1	4/16K	BL	GINT	M1	M3	-	SI	MAG
2	-	-	-	-	PN13	PN12	PN11	PN10
3	CT13	CT12	CT11	CT10	CT9	CT8	CT7	CT6
4	-	-	-	-	-	PG13	PG12	PG11
5	-	SA13	SA12	SA11	SA10	SA9	SA8	SA7
6	-	-	-	-	-	SG13	SG12	SG11
7	TC3	TC2	TC1	TC0	BD3	BD2	BD1	BD0

M1, M2, M3	Select screen mode
EXT	Enables external video input
4/16K	Selects 16K Video RAM if set
BL	Blank screen if reset
SI	16x16 sprites if set; 8x8 if not
MAG	Sprites enlarged if set (double sized: sprite pixels are 2x2)
GINT	Generate interrupts if set
PN	Address for pattern name table (screen) = R2 * 400h
CT	Address for colour table (special meaning in M2) = R3 * 40h
PG	Address for pattern generator table (special meaning in M2) = R4 * 800h
SA	Address for sprite attribute (y, x, pattern, colour) table = R5 * 80h
SG	Address for sprite generator table = R6 * 800h
TC	Text colour (foreground)
BD	Backdrop (background + border)

STATUS REGISTER

INT	5S	C	FS4	FS3	FS2	FS1	FS0

FS	Fifth sprite (first sprite not displayed). Valid if 5S is set
C	Sprite collision detected
5S	Fifth sprite (not displayed) detected
INT	Set at each screen update (refresh)

VDP REGISTER ACCESS

The status register can't be write. After reading the status register, INT (bit#7) and C (bit#5) are reset.

```
ASM:    in      a,(0bfh)  ; get register value
```

COLECO BIOS : 1FDC ; Output A = vdp status register value.

The control registers can't be read. Two bytes must be writen:

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	V7	V6	V5	V4	V3	V2	V1	V0
Byte 1	1	-	-	-	-	R2	R1	R0

Legend

V* Value to be writen in the register. (V7-V0)
R* Register number. (R2-R0)

```
ASM:    ld      a, value
        out     (0bfh),a  ; set value
        ld      a, register_number
        add     a,80h
        out     (0bfh),a  ; write value in register
```

COLECO BIOS : 1FD9 ; Input C = data, B = register #.

VDP MEMORY ACCESS

To read or write video memory data, two bytes must be written to set the video memory offset address.

	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Byte 0	A7	A6	A5	A4	A3	A2	A1	A0
Byte 1	0	R/W	A13	A12	A11	A10	A9	A8

Legend

A* Memory address. (A13-A0)
R/W Flag is set to write, reset to read.

```
; HL = Video Memory Offset.
ASM:    ld      a,l
        out     (0bfh),a  ; low-addr.
        ld      a, h
        add     a,40h     ; write flag
        out     (0bfh),a  ; hi-addr.
```

After setting the video memory address, simply read or write data through the video data port (0beh). The offset address in video memory is auto-incremented after each 1-byte transfer.

COLECO BIOS : 1FDF ; Write VRAM : DE = offset, HL = pointer to data buffer, BC = count
COLECO BIOS : 1FE2 ; Read VRAM : DE = offset, HL = pointer to data buffer, BC = count

VRAM MEMORY ACCESS DELAY TIMES

The amount of time necessary for the CPU to transfer a byte of data to or from VRAM memory can vary from 2 to 7.95 microseconds. Once the VDP has been told to read or write a byte of data to or from VRAM it takes approximately 2 microseconds until the VDP is ready to make the data transfer. In addition to this delay, the VDP must wait for a CPU access window; i.e., the period of time when the VDP is not occupied with the memory refresh or screen display and is available to read or write data. Summary of these delay times in microseconds are showed in the following table.

Screen Mode	Condition	VDP Delay	Time waiting for an access window	Total Time
Graphics I, II M0, M2	Active Display Area	2	0 - 5.95*	2 - 7.95*
Text M1	Active Display Area	2	0 - 1.1	2 - 3.1
Multicolor M3	Active Display Area	2	0 - 1.5	2 - 3.5
All	Screen is Blanked	2	0	2
All	4300 microseconds after Vertical Interrupt Signal	2	0	2

*: The worst case time between windows occurs during the Graphics I or II screen mode when sprites are being used.

Two situations occur where the time waiting for an access window is effectively zero:

1. Screen is blanked by reseting the blank bit of register 0.
2. VDP is in the vertical refresh mode. This mode came right after the active display area period. An interrupt output signal (NMI) indicates that the VDP is entering the vertical refresh mode and that for the next 4300 microseconds. The program that monitors the interrupt output must allow for its own delays in responding to the interrupt signal (NMI) and recognize how much time it has left during the refresh period. The CPU must set the interrupt enable bit (GINT) of Register 1 in order to enable the interrupt for each frame, and then read the status register each time an interrupt is issued to clear the interrupt output. See NMI section for details.

NMI - Non Maskable Interrupt

The VDP output pin is used to generate an interrupt at the end of each active-display scan which is about every 1/60 second for the TMS9928A (NTSC) and 1/50 second for the TMS9929A (PAL). The interupt output signal is active when the generate interrupts bit (GINT) in VDP Register 1 is set and the bit 7 (INT) of the status register is set. Interrupts are cleared when status register is read.

In other words:

- After a vertical retrace (refresh done), the bit 7 of the status register is set.
- If GINT (bit 5 of control register#1) is set, the NMI interrupts the normal execution.
- When it's time again to refresh, the bit 7 of the status register is reset.

NMI can be used to execute something again and again at a regular speed like updating graphics, sounds or calling the game engine (game loop).

COLOR PALETTE

COLOR #	COLOR	Y	R-Y	B-Y
0	Invisible	-	-	-
1	Black	0.00	0.47	0.47
2	Medium Green	0.53	0.07	0.20
3	Light Green	0.67	0.17	0.27
4	Dark blue	0.40	0.40	1.00
5	Light blue	0.53	0.43	0.93
6	Dark Red (brown)	0.47	0.83	0.30
7	Cyan	0.73	0.00	0.70
8	Medium Red	0.53	0.93	0.27
9	Light Red (Pink/orange)	0.67	0.93	0.27
10 (A)	Dark Yellow (Yellow)	0.73	0.57	0.07
11 (B)	Light Yellow (Yellow + Light Grey)	0.80	0.57	0.17
12 (C)	Dark Green	0.47	0.13	0.23
13 (D)	Magenta	0.53	0.73	0.67
14 (E)	Grey (Light Grey)	0.80	0.47	0.47
15 (F)	White	1.00	0.47	0.47

TMS9928 color palette calculated by Richard F. Drushel, based on the TMS9928 technical documentation.

TMS9938 color palette calculated by Marat Fayzullin

TMS9928 color palette used in MESS emulator

The default color palette used in ADAMEM is the one based on the TMS9928 technical documentation.

The color palette used in COLEM is the one calculated by Marat Fayzullin.

The color palette I see in my Commodore monitor model 1802 looks like the one used in MESS emulator.

More information about Texas Instruments TMS99n8 color palette.

URL:

http://junior.apk.net/~drushel/pub/coleco/twwmca/wk961118.html
http://junior.apk.net/~drushel/pub/coleco/twwmca/wk961201.html
http://junior.apk.net/~drushel/pub/coleco/twwmca/wk970202.html

VIDEO DISPLAY SUMMARY

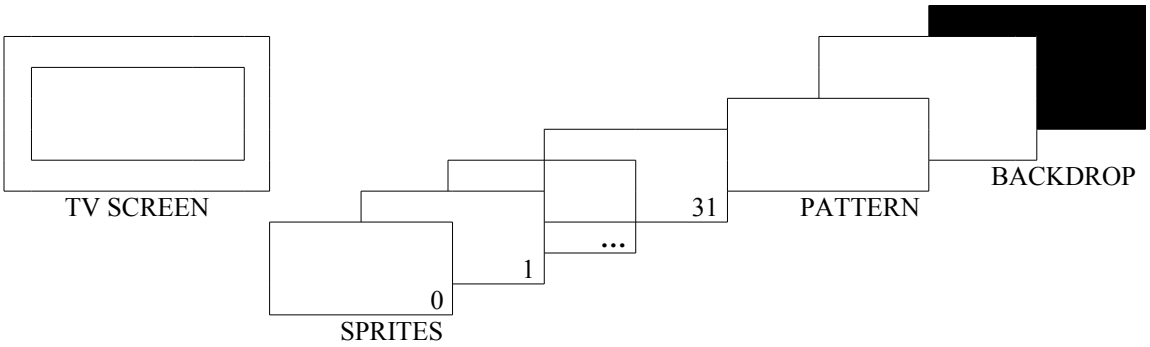
Based on The Hackers' Guide to ADAM™ and Texas Instrument documentation

The VDP displays an image on the screen that can be envisioned as a set of display planes sandwiched together. The objects on planes closest to the viewer have higher priority. In cases where two or more entities on different planes are occupying the same spot on screen, the entity on the higher priority plane will show at that point.

The first 32 planes each may contain a single sprite. Since the coordinates of the sprite are in terms of pixels, the sprite can be positioned and moved about very accurately. Sprites are available in three sizes: 8x8 pixels, 16x16 pixels, and 32x32 pixels. The sprites are showed in Multicolor and Graphics modes only. There is also a restriction on the number of sprites on a line: only 4 sprites can be active on any horizontal line, the additional sprites will be automatically made transparent.

Behind the sprite planes is the pattern plane. The pattern plane is used for textual and graphics images generated by the different screen modes. The pattern plane is broken into group of pixels called pattern positions. Since the full image is 256x192 pixels, there are 32x24 pattern positions (of 8x8 pixels) on the screen in the Graphics modes, 40x24 positions (of 6x8 pixels) on the screen in the Text mode, and 64x48 positions (of 4x4 pixels) on the screen in the Multicolor mode.

Behind the pattern plane is the backdrop, which is larger in area than the other planes so that it forms a border around the other planes. The backdrop consists of a single color used for the display borders and as the default color for the active display area. The default color is stored in the VDP register 7. When the backdrop color is transparent, the backdrop automatically defaults to black.



VDP Screen modes

Mode 0 - Graphics I

Description: 32x24 characters, two colors per 8 characters, sprites active.

Mode 1 - Text

Description: 40x24 characters (6x8), colors set in control register#7, sprites inactive.

Mode 2 - Graphics II

Description: 32x24 characters, 256x192 pixels, two colors per line of 8 pixels, sprites active.

Special meaning for CT* and PG*:

At control register#3, only bit 7 (CT13) sets the address of the color table (address: 0000 or 2000). Bits 6 - 0 are an AND mask over the top 7 bits of the character number.

At control register#4, only bit 2 (PG13) sets the address of the pattern table (address: 0000 or 2000). Bits 1 and 0 are an AND mask over the top 2 bits of the character number. If the AND mask is:

- 00, only one set (the first one) of 256 characters is used on screen.
- 01, the middle of the screen (8 rows) use another set (the second one) of 256 characters.
- 10, the bottom of the screen (8 rows) use another set (the third one) of 256 characters.
- 11, three set of 256 characters are used on screen: set one at the top 8 rows, set two in the middle 8 rows, and set three at the bottom 8 rows. This particular mode is normally used as a bitmap mode screen. The bitmap mode screen is in fact all three characters sets (top, middle and bottom) showed on screen at the same time by filling the screen with all the characters.

Mode 3 - Multicolor

Description: 64x48 big pixels (4x4), sprites active.

SPRITES

From Daniel Bienvenu's CV programming documentation.

The sprites are easy to use because you can place them anywhere on screen. Each sprite can be identified like a layer on screen. Normally, the size of a sprite is 16x16 but there is also the 8x8 format. The limits for using sprites are : never more than 4 sprites in a row, on the same scan line and never more than 32 sprites on screen at the same time. All sprites can be magnified by 2 (by changing size of the pixels in sprites).

To display a sprite on screen, you need a vector of 4 bytes (Position Y, Position X, Pattern and Colour) in the right video memory location.

SPRITES COLOR

The bits 0 are already replaced by the transparent color so there is only one color per sprite.
To use more than one color, you have two solutions:

- Use more than one sprite (one for each color)
- Use a combination of sprites and characters.

SPRITES LOCATIONS ON SCREEN

The Y location of a sprite can be any values between 0 and 255 except 208. The special value 208 tells the video chip to stop checking for sprites to display on screen. If you want to not show sprite#1 but you want to show sprite#2, use a value like 207 for the Y location of sprite#1.

SPRITES PATTERN

8x8 SPRITE

A 8x8 sprite looks like a character in screen mode 0 but all bits 0 are colored with the invisible color 0.

16x16 SPRITE

A 16x16 sprite is a combination of four (4) 8x8 patterns. These patterns are displayed like this:

1	3
2	4

SPRITE 8x8 SAMPLE

Spaceship Pattern								Pattern Codes
0	0	0	1	1	0	0	0	18
1	0	0	1	1	0	0	1	99
1	0	0	1	1	0	0	1	99
1	0	1	1	1	1	0	1	BD
1	1	1	0	0	1	1	1	E7
1	1	1	0	0	1	1	1	E7
1	0	1	1	1	1	0	1	BD
0	0	1	1	1	1	0	0	3C

Spaceship Color	
Color	Code
	4

SPRITE 16x16 SAMPLE

Face Pattern																Pattern Codes	
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	00
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	03	E0
0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0F	F8
0	0	0	1	1	0	0	1	1	1	0	0	1	1	0	0	19	CC
0	0	1	1	0	1	1	0	1	0	1	1	0	1	1	0	36	B6
0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	3F	FE
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7F	FF
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7F	FF
0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	7F	FF
0	1	1	1	0	0	0	0	0	0	0	0	0	1	1	1	70	07
0	0	1	1	0	0	0	0	0	0	0	0	0	1	1	0	30	06
0	0	1	1	1	0	0	0	0	0	0	0	1	1	1	0	38	0E
0	0	0	1	1	1	1	0	0	0	1	1	1	1	0	0	1E	3C
0	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0F	F8
0	0	0	0	0	0	1	1	1	1	1	0	0	0	0	0	03	E0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00	00

Face Color	
Color	Code
	A

This sprite pattern is coded like this in the video memory (VRAM) :

```
00, 03, 0F, 19, 36, 3F, 7F, 7F, 7F, 70, 30, 38, 1E, 0F, 03, 00, 00, E0,
F8, CC, B6, FE, FF, FF, FF, 07, 06, 0E, 3C, F8, E0, 00.
```

CHARACTERS

From Daniel Bienvenu's CV programming documentation.

The characters, also named names in the official coleco documents, are normally used as a background or semi-mobile objects. All the letters, numbers and symbols are characters. A character is 8x8 pixels sized except for screen mode 1 where a character is only 6x8. The characters can be copied many times on screen but only at specific positions. Normally, there are 768 spaces (32 columns x 24 rows) on screen where characters can be placed except for screen mode 1 (40 columns x 24 rows).

VIDEO MEMORY FOR CHARACTERS

The names of the three tables in Video RAM for characters are:

NAME: The screen. It contains characters # for all the spaces on screen.
PATTERN: The characters pattern (256 characters # : HEX values from 00 to FF)
COLOR: The characters color(s)

A character has the same pattern and color anywhere on screen (one exception with screen mode 2). So you can't use the same character to print a blue 'A' and a red 'A' side-by-side. The solution is using two characters with the same pattern 'A' but with different colors, one blue, one red.

You must understand the difference between the ASCII code and the symbol. In the ASCII code, the character '1' is not the character # 1 but the character # 49. But, by changing the patterns, you can make the character #1 looks like a '1' on screen if you like.

Now, if the color of the character (# 65, 41 in HEX value) 'A' is blue and if you change the pattern of the character (# 49, 31 in HEX value) '1' to looks like an 'A' but with a red color, then you just have to put the characters 'A' and '1' (values # 65 and # 49) side-by-side on screen to see two 'A's side-by-side but one blue and one red.

In the NAME table, there is HEX values 41 for the 'A' and 31 for the '1'.
In the PATTERN table, there are identical pattern data for the character 'A' and '1'.
In the COLOR table, there are different values to set a blue color for the character 'A' and a red color for the character '1' that looks now like an 'A'.

- Screen mode 0: there are only two colors (one color for bits 1 and one color for bits 0) for each bloc of 8 characters in the character set.
- Screen mode 1: there are only two colors (one color for bits 1 and one color for bits 0) for all the characters. These colors are set in VDP register 7.
- Screen mode 2: there are two colors (one color for bits 1 and one color for bits 0) per line of 8 pixels for all the characters in the character set.

For the screen mode 0 and 2, if you want to see the background color set in the VDP register, you have to use the transparent color code.

CHARACTER PATTERN

Characters are named names in the official Coleco documents, and tiles by some programmers. A character on screen is defined by a number that represent a pattern in the PATTERN GENERAOR table in VRAM at a specific location in the NAME table in VRAM. A character pattern is a 8x8 pixels graphic, 6x8 for screen mode 1.

CHARACTER PATTERN SAMPLE

Screen Mode 0 Character Sample

A 8x8 character pattern sample that use only two colors (one for bits 1 and one for bits 0).

Spaceship Pattern	Pattern Codes	Spaceship Colors		
		Bits 1	Bits 0	Code
0 0 0 1 1 0 0 0	18			E1
1 0 0 1 1 0 0 1	99			
1 0 0 1 1 0 0 1	99			
1 0 1 1 1 1 0 1	BD			
1 1 1 0 0 1 1 1	E7			
1 1 1 0 0 1 1 1	E7			
1 0 1 1 1 1 0 1	BD			
0 0 1 1 1 1 0 0	3C			

Screen Mode 1 Character Sample

A 6x8 character pattern that use only the colors set in the vdp control register #7.

Spaceship Pattern	Pattern Codes	Spaceship Colors		
		Bits 1	Bits 0	Code
0 0 1 1 0 0 0 0	30			E1
1 0 1 1 0 1 0 0	B4			
1 0 1 1 0 1 0 0	B4			
1 1 1 1 1 1 0 0	FC			
1 1 0 0 1 1 0 0	CC			
1 1 0 0 1 1 0 0	CC			
1 1 1 1 1 1 0 0	FC			
0 1 1 1 1 0 0 0	78			

Screen Mode 2 Character Sample

A 8x8 character pattern that use two colors per line.

Spaceship Pattern	Pattern Codes	Spaceship Colors		
		Bits 1	Bits 0	Codes
0 0 0 1 1 0 0 0	18			81
1 0 0 1 1 0 0 1	99			A1
1 0 0 1 1 0 0 1	99			E1
1 0 1 1 1 1 0 1	BD			E1
1 1 1 0 0 1 1 1	E7			EF
1 1 1 0 0 1 1 1	E7			E7
1 0 1 1 1 1 0 1	BD			E1
0 0 1 1 1 1 0 0	3C			81

COLECO ASCII TABLE

DEC: 0-63
HEX: 00-3F

DEC	HEX	CHARACTER	DEC	HEX	CHARACTER
0	00		32	20	Space
1	01		33	21	!
2	02		34	22	"
3	03		35	23	#
4	04		36	24	$
5	05		37	25	%
6	06		38	26	&
7	07		39	27	'
8	08		40	28	(
9	09		41	29	)
10	0A		42	2A	*
11	0B		43	2B	+
12	0C		44	2C	,
13	0D		45	2D	-
14	0E		46	2E	.
15	0F		47	2F	/
16	10		48	30	0
17	11		49	31	1
18	12		50	32	2
19	13		51	33	3
20	14		52	34	4
21	15		53	35	5
22	16		54	36	6
23	17		55	37	7
24	18		56	38	8
25	19		57	39	9
26	1A		58	3A	:
27	1B		59	3B	;
28	1C		60	3C	<
29	1D	©	61	3D	=
30	1E	T	62	3E	>
31	1F	M	63	3F	?

DEC: 64-127
HEX: 40-7F

DEC	HEX	CHARACTER	DEC	HEX	CHARACTER
64	40	@	96	60	`
65	41	A	97	61	a
66	42	B	98	62	b
67	43	C	99	63	c
68	44	D	100	64	d
69	45	E	101	65	e
70	46	F	102	66	f
71	47	G	103	67	g
72	48	H	104	68	h
73	49	I	105	69	i
74	4A	J	106	6A	j
75	4B	K	107	6B	k
76	4C	L	108	6C	l
77	4D	M	109	6D	m
78	4E	N	110	6E	n
79	4F	O	111	6F	o
80	50	P	112	70	p
81	51	Q	113	71	q
82	52	R	114	72	r
83	53	S	115	73	s
84	54	T	116	74	t
85	55	U	117	75	u
86	56	V	118	76	v
87	57	W	119	77	w
88	58	X	120	78	x
89	59	Y	121	79	y
90	5A	Z	122	7A	z
91	5B	[	123	7B	{ (brace left)
92	5C	\	124	7C	| (broken vertical)
93	5D	]	125	7D	} (brace right)
94	5E	^	126	7E	~ (tilde)
95	5F	_ (underline)	127	7F	▒

GLOSSARY

ATN: Attenuator
LSB: Less Significant Byte
LSN: Less Significant Nibble
MSB: Most Significant Byte
MSN: Most Significant Nibble

Nibble: 4-bit (half-byte)
Byte: 8-bit
Word: 16-bit