```
;====================================================================
====================================
; C64 Brain Assembly Language Project - Caver Engine 0.3a
; 2016/17 - Peter 'Sig' Hewett aka RetroRomIcon
;====================================================================
====================================
#region "ChangeLog 0.1a"
; Changelog
; 0.1a - MLP PROJECT FRAMEWORK 1.3
;        Restarting from 1.3 Framework due to lack of focus,
messy code,
;        too many 'get back to it' moments. Starting more
methodically this time
;
;       - Setup VIC Bank Memory Map : Screens 0,1,6 Charsets 1,2
;         leaves a solid VIC Bank with room for 144 sprite
images starting at
;         SPRITE_BASE $70
;
;       - Split screen with raster to upper (gamescreen) and
lower (scoreboard)
;         using screens 0/1 (double buffered) and 6
respectively.
;         Configure screens to use MapChars for top screen and
ScoreChars for the bottom
;
;       - Rewrite ScreenClear for individual screens
(Screen0,Screen1,ScoreScreen,ColorRam)
;       - Rewrite DrawText routines for all 3 screens by passing
the screen address in WPARAM1
;
;       - Add FetchLineAddress helper routine to screen_routines
to fetch correct screen
;         (save typing out the screen detection code for
everything else)
;               - Add alt entry points to this routine to
automatically detect CURRENT_SCREEN
;                 and CURRENT_BUFFER (for double buffering)
;
;       - Screen line lookup tables moved to screen_routines.asm
;       - Rewrite DisplayByte to work on all 3 screens using
WPARAM1
;       - Rewrite Joystick detection code for better legibility
and flow
;
;       - Charpad integration
```

```
;       - Set aside initial memory for levels - permanent block
for level loading
;         at $8000-$8FFF (4k)
;
;       - Basic LoadLevel routine started
;               - Lookup tables for tile data and charset
addresses
;               - Load and save initial charset and map from
under banked memory at $E000
;               - Copy level character set to VIC RAM
;               - Copy Screen to Backbuffer
;       - DrawTile Routine : Draw tile screen at X/Y in block
coords
;       - DrawMap Routine - draws the current screen from the
map at X/Y position
;               sets up useful info: MAP_X_POS, MAP_Y_POS,
MAP_X_DELTA, MAP_Y_DELTA
;               MAP_POS_ADDRESS
;
;       - Added SetSpriteImage routine - rearranged sprite
handling data
;       - Sprite updating happens for both screens (foreground
and buffer)
;       - ScreenSwap routine written to exchange front and
backbuffer screens
#endregion

#region "Changelog 0.2a"
;
;       - Scrolling
;               - Test double buffer using sprite 8 to show
screen number (0,1)
;               - Setup raster for 0-7 hardware horizontal
scrolling
;               - ScrollRight routine and frame split for 1
pixel scroll
;               - ShiftChars for right scroll on backbuffer
;               - ShiftColor for right scroll
;               - Write and draw right scroll buffers
;               - ScrollLeft routine and frame split for 1 pixel
scroll
;               - ShiftChars for left scroll on backbuffer
;               - ShiftColor for left scroll.
;               - Combined buffer copy into one routine and
buffer for 'horizontal buffer'
;               - LeftBufferDraw and LeftBuffer Color routines
;
```

```
;               - Setup raster for 0-7 hardware vertical scroll
;               - Scroll down routine for 1 pixel scroll down
;               - Add 'IrqGlitchControl' to catch glitches and
stabilize the raster
;               - ShiftChars for down scroll
;               - ShiftColors for down scroll
;               - Write and draw vertical buffers for characters
and color
;
;               - Scroll up routine for 1 pixel scroll up
;               - ShiftChars for up scroll
;               - ShiftChars for down scroll
;               - Copy and write buffers for character and color
updates (vert scroll)
;
#endregion

#region "Changelog 0.3a"
;       - Basic Player Setup + Joystick movement
;               - Import 'MLP Speluker Sprites' to CBM PRG
STUDIO
;               - Created MOBS.asm to handle mobile objects
(players / enemies)
;               - PlayerSetup now handles setting up the player
sprite(s)
;               - Modified test map for better viewing with
sprites
;               - Setup test 'man' sprite with basic joystick
left/right control
;               - Capping left/right movement at characters $07
- $20
;               - Scroll horizontally if cap is hit
;               - BUG FIX : reset horizontal scroll to start
frame on stop
;               - BUG FIX : Horizontal scrolling buffer not
doing a correct lookup on direction
;                               change - splitting the routine for
left and right buffer reads
;               - Rewrote to right/left copy to horizontal copy
buffer routines
;               - Move player sprite when stopping and scrolling
is 'resetting' horizontally
;               - Lock horizontal scroll to map edges
;
;               - Setup test 'man' sprite with basic joystick
up/down control
;               - Cap downward movement at #$0F (char Y pos)
```

```
;               - Scroll vertically up if cap is hit
;               - Scroll vertically down if cap is hit
;               - BUG FIX : reset vertical scrolling to 'start
frame' on stop
;               + BUG FIX : we hit the 128 byte limit on
branching - time to break joystick
;                           input into seperate subroutines.
;               - BUG FIX : Badline jump at scroll count 1
vertical
;               - BUG FIX : Our original scroll routines are
based on tile aligned maps, we need
;                           to modify all the buffer copy
routines to work with map delta X and Y
;                           (0-3 the character within the tile)
;               - Lock vertical scroll to map edges
;               - BUG FIX - lock joystick input until
'fixscroll' has done its work : eliminate
;                           rapid vert - horizontal joystick
wiggles. I'll do a more elegant
;                           solution later if needed, though in
this game this shouldn't happen
;                           often (but happens a lot when
testing).

#endregion

#region "Changelog 0.4a"
;       - Collision Detection, Basic sprite to character
;       - Edit Map/Tile/Character data to add attributes and
create a 'test area'
;       - Original sprite to character collision routines
weren't on double buffered
;         screens, but I'm going to be testing collisions
against an attributes you
;         can use from Charpad and are stored in the upper half
of the color byte.
;         Also scrolling will effect the delta values we use for
testing against characters
;         so a complete rewrite is in order for the entire
system
;       - CheckMoveRight replaces CanMoveRight and takes into
consideration changing delta values
;         due to scrolling.
;       - BUG FIX : it seems I am very mistaken. The attributes
are NOT able to be read from color
;                   rams upper half byte (or I am doing
something very wrong, or VICE is
```

```
;                   doing something wrong - either way I can't
get it working), but luckily
;                   accessing that data directly given the
character num isn't a huge deal.
;       - Very basic start to replace TestBlocking. Attributes
are taken from Charpad data
;         (Edited in Charpad as 'Materials') and saved /
displayed on debug console
;
;       - CheckMoveLeft replacing CanMoveLeft, updating for
delta values with scrolling
;       - CheckMoveDown replacing CanMoveDown, updating for
delta values with scrolling
;       - CheckMoveUp replacing CanMoveUp, updating for delta
values with scrolling
;       - Adjust initial sprite delta offsets to tweak
collisions with main character
;       - Write new line lookup routine for
playscreen/collisions to fetch the line of the
;         current front screen so it works better/faster with
'CheckMove####' routines

#endregion

#region "Changelog 0.5a"
;       - Basic Player Logic.  Walking/Standing/falling +
Project Cleanup
;       - Moved all map files to a folder 'Maps'
;       - Renamed MOB.asm to Player.asm for better organizing.
Mobs will have a file later
;       - Created more variables for sprite handling:
;
;               SPRITE_IS_ACTIVE   - (0 or 1) test if a sprite
is currently being used
;               SPRITE_DIRECTION_X - direction a sprite is
heading on X axis (-1 , 0 , 1)
;               SPRITE_DIRECTION_Y - direction a sprite is
heading on Y axis (-1 , 0 , 1)
;               SPRITE_ANIM_COUNT  - current frame of animation
the sprite is using
;               SPRITE_ANIM_TIMER  - a timer mask to set the
speed of the animation
;               SPRITE_ANIM        - a pointer to the current
anim list
;
;       - Created basic anim lists for PLAYER_IDLE and
PLAYER_WALK_R for testing
```

```
;       - Reorganized sprite data in memory and in sprite editor
project to make things easier
;       - InitSpriteAnim routine for setting sprites up to be
animated
;       - AnimateSprite routine added - handling animlists and
looped animations
;         added play-once and ping-pong animation types.
;       - Created animlists for Walk Left, Walk Right, and Idle
anims. Made use of ping-pong type
;         to cut 2 anim frames off idle animation.
;
;       - Started work on player states.  As it stands, the test
code being used for joystick
;         reading/player movement/scrolling will quickly become
unmanagable once special case
;         blocks (ropes, moving platforms), player actions
(jumping, climbing), and other things
;         start getting involved. So I'll be breaking it down
into different states.
;         Being in a state of WALKING will only need to handle
things that effect walking
;         and things that would change that state to something
else.
;         I'll be using a system of jump tables to manage this.
;
;       - Added ChangePlayerState to put the player into a new
state and initialize it
;       - PlayerStateIdle framework and setup with animation
starting.
;       - PlayerStateWalk left/right framework and setup with
animation starting.
;       - Joystick inputs and state interaction done between the
3 initial states
;       - Seperate old JoyInput routine. Replaced with
MovePlayerRight so it can be called from
;         state code.
;       - Replace left section of JoyInput routine with
MovePlayerLeft called from state code
;       - Replace down section of JoyInput routine with
MovePlayerDown called from state code
;       - Implemented a very basic 'apply gravity'.  It simply
moves the player down one pixel
;         if called from a state (not all states will - jumping
for instance).
;         this system will be expanded to proper gravity at a
later time, once other issues
;         are dealt with
```

```
;
;       - BUGFIX - when testing basic falling it is possible to
be calling 2 scroll directions
;                  at once. This is a problem as my scroll
routine is barely a 4 way scroller
;                  (technically I'd call it 2 two way scrollers)
and certainly can't handle
;                  8 way scrolling with diagonals.
;                  A simple fix was to perform gravity checks
before joystick input in L/R
;                  walking states and skip L/R input if the
player is falling down.
;                  Not the best solution, but until I upgrade
the scrolling it does the job.
;
;
;       - Added PlayerStateFall. A basic fall routine
;       - Player can walk to lefthand map edge - I'm dubious of
the result and may need a bugfix
;       - Player can walk to righthand map edge
;       - Added PlayerStateStairR. climbing a one char 'stair'
from walking right - works nice.
;       - Added PLAYER_FALLCOUNT to count pixels falling before
kicking in the actual fall state
;       - Modified PlayerStateWalkR to handle climbing down a
'stair'
;       - Modified PlayerstateWalkL to handle climging down a
'stair'
;       - BUG FIX : MovePlayerLeft wasn't returning a 'clear
code'
;       - Code cleanup for release.
;
;
#endregion

#region "Changelog 0.6a"
;       - PLAYER LOGIC : Climbing Ropes

;       - added rope collision type, modified TestBlocking to
stop blocking movement
;       - reordered sprites to accomodate no 2nd sprite on climb
anims
;       - imported rope climbing sprites
;       - Created ClimbRopeUp AnimList
;       - Added PlayerStateRope for rope climbing
;       - Added CheckBlockUnder to check for special blocks
under the player sprite.
```

```
;       - PlayerStateRope also handles climbdown and returns to
PlayerStateIdle when your
;         feet touch a solid block
;       - Simple check added to PlayerStateWalk left and right
for basic 'dismount' of ropes
;       - Adjusted map to test simple dismount and climb
;       - Modified PlayerStateRope to align the player to the
rope so they can pass through 'holes'
;       - Added check to Idle state to allow player to climb
down through 'rope holes' and stop
;         false positives on ground collisions.
;       - Modified PlayerStateRope to doublecheck ending climb
at the end of the rope and returning
;         to idle state
;       - BUGFIX. Check right and left before exiting
PlayerStateRope to avoid 'airwalking'
;       - There is one last bug in here that I can't find yet,
but will get found when the state
;         system and scrolling are eventually refined further.
It occasionally misreads the character
;         the rope is on incorrectly as one character to the
right. I haven't been able to reliably
;         replicate it.


#endregion

#region "TO DO"
;-----------------------------------------------------------------
-----------------------------------
;
;
; TO DO :
;       - Map code needs an overall X Y coord in characters
;       - expand testblocking and CheckMove routines to return
more than 0 or 1
;       + When scroll hits the map edge, player can walk to the
screen edge
;       Update sprite postion registers in the interrupt (have
them all synced to scroll)
;       Variable speed scrolling
;       Portrait (32 x 64) sized maps
;       'Spawn' info on level start
;
;       Sprites vs Character/Actor Object
;
```

```
;       A better 'scroll fix' solution. Perhaps introducing
'fake frames' to catch up or a way
;       to better sync horizontal and vertical scrolling to make
it closer to an 8 way scroller.
;
;
;
; POSSIBLE OPTIMIZATIONS:
;       - States are being looked up every frame, every change,
for every sprite... and it's only
;         going to compound once enemies arrive. Maybe save the
address to the state and update it
;         when the state changes.
;
;  DONE - JSR is 6 cycles , RTS is 6 cycles, JMP is 3 (5 with
indirect addressing). Following the
;         'state trail' and changing JSR to JMP - thus making
the ChangeState line one big subroutine
;         when it runs could save a bunch of raster time.
;-----------------------------------------------------------------
-----------------------------------
#endregion
;================================================================
================
;
DIRECTIVES
;================================================================
================
Operator Calc         ; IMPORTANT - calculations are made BEFORE
hi/lo bytes
                      ;             in precidence (for
expressions and tables)
;================================================================
================
;
DEFINITIONS
;================================================================
================
IncAsm "VICII.asm"                      ; VICII register
includes
IncAsm "macros.asm"                     ; macro includes
;================================================================
================
;================================================================
================
;
CONSTANTS
```

```
;================================================================
===============

CONSOLE_TEXT = SPRITE_CONSOLE_TEXT

CONSOLE_DISPLAY = DisplaySpriteInfo



#region "Constants"
SCREEN_MEM    = $4000
SCREEN1_MEM   = $4000                  ; Bank 1 - Screen 0 ; $4000
SCREEN2_MEM   = $4400                  ; Bank 1 - Screen 1 ; $4400
SCORE_SCREEN = $5800                   ; Bank 1 - Screen 6 ; $5800

COLOR_MEM   = $D800                    ; Color mem never changes
CHAR_MEM    = $4800                    ; Base of character set
memory (set 1)
SPRITE_MEM = $5C00                     ; Base of sprite memory

COLOR_DIFF = COLOR_MEM - SCREEN_MEM   ; difference between color
and screen ram
                                       ; a workaround for CBM PRG
STUDIOs poor
                                       ; expression handling

SPRITE_POINTER_BASE = SCREEN_MEM + $3f8 ; last 8 bytes of screen
mem

SPRITE_BASE = $70                         ; the pointer to the
first image

SPRITE_0_PTR = SPRITE_POINTER_BASE + 0   ; Sprite pointers
SPRITE_1_PTR = SPRITE_POINTER_BASE + 1
SPRITE_2_PTR = SPRITE_POINTER_BASE + 2
SPRITE_3_PTR = SPRITE_POINTER_BASE + 3
SPRITE_4_PTR = SPRITE_POINTER_BASE + 4
SPRITE_5_PTR = SPRITE_POINTER_BASE + 5
SPRITE_6_PTR = SPRITE_POINTER_BASE + 6
SPRITE_7_PTR = SPRITE_POINTER_BASE + 7

SPRITE_DELTA_OFFSET_X = 8                ; Offset from SPRITE
coords to Delta Char coords
;SPRITE_DELTA_OFFSET_Y = 11               ; approx the center of
the sprite
SPRITE_DELTA_OFFSET_Y = 14
```

```
NUMBER_OF_SPRITES_DIV_4 = 3             ; This is for my personal
version, which
                                        ; loads sprites and
characters under IO ROM


LEVEL_1_MAP    = $E000                    ;Address of level 1
tiles/charsets
LEVEL_1_CHARS = $E800
#endregion

;=================================================================
====================================
;
ZERO PAGE VARIABLES
;=================================================================
====================================
#region "ZeroPage"
PARAM1 = $03                    ; These will be used to pass
parameters to routines
PARAM2 = $04                    ; when you can't use registers or
other reasons
PARAM3 = $05
PARAM4 = $06                    ; essentially, think of these as
extra data registers
PARAM5 = $07

TIMER = $08                     ; Timers - fast and slow, updated
every frame
SLOW_TIMER = $09

WPARAM1 = $0A                   ; Word length Params. Same as above
only room for 2
WPARAM2 = $0C                   ; bytes (or an address)
WPARAM3 = $0E

;---------------------------- $11 - $16 available

ZEROPAGE_POINTER_1 = $17        ; Similar only for pointers that
hold a word long address
ZEROPAGE_POINTER_2 = $19
ZEROPAGE_POINTER_3 = $21
ZEROPAGE_POINTER_4 = $23

CURRENT_SCREEN    = $25         ; Pointer to current front screen
CURRENT_BUFFER    = $27         ; Pointer to current back buffer
```

```
SCROLL_COUNT_X    = $29        ; Current hardware scroll value
SCROLL_COUNT_Y    = $2A
SCROLL_SPEED      = $2B        ; Scroll speed (not implemented
yet)
SCROLL_DIRECTION = $2C         ; Direction we are scrolling in
SCROLL_MOVING     = $2D        ; are we moving? (Set to direction
of scrolling)
                               ; This is for resetting back to
start frames

                              ; All data is for the top left
corner of the visible map area
MAP_POS_ADDRESS = $2E         ; (2 bytes) pointer to current
address in the level map
MAP_X_POS         = $30       ; Current map x position (in tiles)
MAP_Y_POS         = $31       ; Current map y position (in tiles)
MAP_X_DELTA       = $32       ; Map sub tile delta (in characters)
MAP_Y_DELTA       = $33       ; Map sub tile delta (in characters)

#endregion

;================================================================
================
;
BASIC KICKSTART
;================================================================
================
KICKSTART
; Sys call to start the program - 10 SYS (2064)

*=$0801

        BYTE
$0E,$08,$0A,$00,$9E,$20,$28,$32,$30,$36,$34,$29,$00,$00,$00




;================================================================
==================================
;
PROGRAM START
;================================================================
==================================
*=$0810
```

```
PRG_START

        lda #0                          ; Turn off sprites
        sta VIC_SPRITE_ENABLE

        lda VIC_SCREEN_CONTROL          ; turn screen off with
bit 4
        and #%11100000                  ; mask out bit 4 -
Screen on/off
        sta VIC_SCREEN_CONTROL          ; save back - setting
bit 4 to off

        ;-------------------------------------------------------
---------------
        ;
VIC BANK SETUP
        ;-------------------------------------------------------
---------------
#region "VIC Setup"
        ; To set the VIC bank we have to change the first 2 bits
in the
        ; CIA 2 register. So we want to be careful and only
change the
        ; bits we need to.

        lda VIC_BANK            ; Fetch the status of CIA 2
($DD00)
        and #%11111100          ; mask for bits 2-8
        ora #%00000010          ; the first 2 bits are your
desired VIC bank value
                                ; In this case bank 1 ($4000 -
$7FFF)
        sta VIC_BANK
        ;-------------------------------------------------------
---------------
        ;                                           CHARACTER SET
AND SCREEN MEM
        ;-------------------------------------------------------
---------------
        ; Within the VIC Bank we can set where we want our
screen and character
        ; set memory to be using the VIC_MEMORY_CONTROL at $D018
        ; It is important to note that the values given are
RELATIVE to the start
        ; address of the VIC bank you are using.
```

```
        lda #%00000010    ; bits 1-3 (001) = character memory 2 :
$0800 - $0FFF
                          ; bits 4-7 (000) = screen memory 0 :
$0000 - $03FF

        sta VIC_MEMORY_CONTROL

        ; Because these are RELATIVE to the VIC banks base
address (Bank 1 = $4000)
        ; this gives us a base screen memory address of $4000
and a base
        ; character set memory of $4800
        ;
        ; Sprite pointers are the last 8 bytes of screen memory
(25 * 40 = 1000 and
        ; yet each screen reserves 1024 bytes). So Sprite
pointers start at
        ; $4000 + $3f8.

        ; After alloction of VIC Memory for Screen, backbuffer,
scoreboard, and
        ; 2 character sets , arranged to one solid block of mem,
        ; Sprite data starts at $5C00 - giving the initial image
a pointer value of $70
        ; and allowing for up to 144 sprite images

#endregion
        ;-------------------------------------------------------
----------------
        ;
SYSTEM SETUP
        ;-------------------------------------------------------
----------------
#region "System Setup"
System_Setup

        ; Here is where we copy level 1 data from the start
setup to under
        ; $E000 so we can use it later when the game resets.
        ; A little bank switching is involved here.
        sei

        ; Here you load and store the Processor Port ($0001),
then use
        ; it to turn off LORAM (BASIC), HIRAM (KERNAL), CHAREN
(CHARACTER ROM)
```

```
        ; then use a routine to copy your sprite and character
mem under there
        ; before restoring the original value of $0001 and
turning interrupts
        ; back on.

        lda PROC_PORT                   ; store ram setup
        sta PARAM1

        lda #%00110000                  ; Switch out BASIC,
KERNAL, CHAREN, IO
        sta PROC_PORT

        ; When the game starts, Level 1 tiles and characters are
stored in place to run,
        ; However, when the game resets we will need to restore
these levels intact.
        ; So we're saving them away to load later under the
KERNAL at $E000-$EFFF (4k)
        ; To do this we need to do some bank switching, copy
data, then restore as
        ; we may use the KERNAL later for some things.

        loadPointer ZEROPAGE_POINTER_1, MAP_MEM         ; source
        loadPointer ZEROPAGE_POINTER_2, LEVEL_1_MAP     ;
destination

        jsr CopyChars                   ; CopyChars for charsets
copys 2048 bytes of character
                                        ; data, the same size as
our tile maps, so we use that
                                        ; routine

        loadPointer ZEROPAGE_POINTER_1, CHAR_MEM
        loadPointer ZEROPAGE_POINTER_2, LEVEL_1_CHARS

        jsr  CopyChars


        lda PARAM1                      ; restore ram setup
        sta PROC_PORT

        cli
#endregion
        ;-------------------------------------------------------
---------------------------------
```

```
        ;
SCREEN SETUP
        ;-------------------------------------------------------
-----------------------------------
#region "Screen Setup"
Screen_Setup
        lda #COLOR_BLACK
        sta VIC_BORDER_COLOR            ; Set border and
background to black
        sta VIC_BACKGROUND_COLOR

        lda #COLOR_LTBLUE
        sta VIC_CHARSET_MULTICOLOR_1
        lda #COLOR_BLUE
        sta VIC_CHARSET_MULTICOLOR_2

        loadPointer CURRENT_SCREEN,SCREEN1_MEM
        loadPointer CURRENT_BUFFER,SCREEN2_MEM

        lda #$40                        ; Use #$40 as the fill
character on GameScreen
        jsr ClearScreen1                ; Clear both screens
(double buffer)
        jsr ClearScreen2

        lda #32                         ; Use #32 (space) as the
fill character on
        jsr ClearScoreScreen            ; Score Screen

        lda #COLOR_BLUE                 ; Fill entire visible
color ram with COLOR_BLUE
        jsr ClearColorRam
        ;-------------------------------------------------
SCROLLING
                                        ; VERTICAL
        lda #3                          ; start with vertical
scroll of 3
        sta SCROLL_COUNT_Y              ; which is the default

                                        ; HORIZONTAL
        lda #4                          ; Start centered on
character
        sta SCROLL_COUNT_X              ; at X #4

        lda #SCROLL_STOP                ; direction = up
        sta SCROLL_DIRECTION
        lda #0                          ; speed = 0
```

```
        sta SCROLL_SPEED                ; (not implemented yet)

        ;-------------------------------------------------
CHARPAD LEVEL SETUP
        lda #1                          ; Start Level = 1
        sta CURRENT_LEVEL

        jsr LoadLevel                   ; load level 1 data

        ldx #0                          ; X start pos (in tile
coords)
        ldy #20                         ; Y start pos (in tile
coords)

        jsr DrawMap                     ; Draw the level map
(Screen1)
                                        ; And initialize it

        jsr CopyToBuffer                ; Copy to the
backbuffer(Screen2)

        ;-------------------------------------------------
DEBUG CONSOLE

                                        ; Display the Debug
Console Text
        loadpointer ZEROPAGE_POINTER_1, CONSOLE_TEXT

        lda #0                          ; PARAM1 contains X
screen coord (column)
        sta PARAM1
        lda #19                         ; PARAM2 contains Y
screen coord (row)
        sta PARAM2
        lda #COLOR_WHITE                ; PARAM3 contains the
color to use
        sta PARAM3
        jsr DisplayText                 ; Then we display the
text

        jsr DisplaySpriteInfoNow        ; Now update it with the
debug info

        ;-------------------------------------------------
RASTER SETUP
        jsr WaitFrame
```

```
        jsr InitRasterIRQ               ; Setup raster
interrupts
        jsr WaitFrame


        lda #%00011011                  ; Default (Y scroll = 3
by default)                                     ;
        sta VIC_SCREEN_CONTROL

#endregion


        ;-------------------------------------------------------
-----------------------------------
        ;
SPRITE SETUP
        ;-------------------------------------------------------
-----------------------------------
#region "Sprite Setup"

        lda #0
        sta VIC_SPRITE_ENABLE           ; Turn all sprites off
        sta VIC_SPRITE_X_EXTEND         ; clear all extended X
bits
        sta SPRITE_POS_X_EXTEND         ; in registers and data

        ;------------------------------------------------ SETUP
AND DISPLAY PLAYER
        jsr PlayerSetup

                                        ; Use Sprite 7 to test
double buffering
        lda #50
        sta VIC_SPRITE_X_POS + 14       ; Set sprite X pos for
sprite 7 in registers
        sta SPRITE_POS_X + 7            ; Set sprite X pos for
sprite 7 in variables

        lda VIC_SPRITE_X_EXTEND         ; Set X extended bit for
sprite 7
        ora #%10000000                  ; (mask it into existing
values)
        sta VIC_SPRITE_X_EXTEND

        lda SPRITE_POS_X_EXTEND         ; Set X extended bit for
sprite 7 in variables
        ora #%10000000
```

```
        sta SPRITE_POS_X_EXTEND

        lda #60                         ; Set Y pos for sprite 7
in registers and variables
        sta VIC_SPRITE_Y_POS + 14
        sta SPRITE_POS_Y + 7

        lda #COLOR_VIOLET               ; Set sprite color for
sprite 7
        sta VIC_SPRITE_COLOR + 7

        lda #SPRITE_BASE + 41           ; Set image. We give it
a different image for screen 1
        sta SPRITE_7_PTR                ; and screen 2 in the
pointers, so when the screen flips,
        lda #SPRITE_BASE + 42           ; the number changes
with zero overhead
        sta SPRITE_POINTER_BASE2 + 7

        lda #%10000011                  ; Turn on sprites 0 1
and 7
        sta VIC_SPRITE_ENABLE


#endregion


;=================================================================
============================
        ;
MAIN LOOP

;=================================================================
============================
        ; The main loop of the program - timed to the verticle
blanking period
        ;-----------------------------------------------------
----------------
MainLoop
        jsr WaitFrame                   ; wait for the vertical
blank period

        lda #COLOR_YELLOW               ; Raster time indicator
- turn the border yellow
        sta VIC_BORDER_COLOR            ; before the main loop
starts
```

```
        jsr UpdateTimers
        jsr UpdateScroll
        jsr UpdatePlayer

;        jsr CONSOLE_DISPLAY              ; Display simple debug
info

        lda #COLOR_BLACK                ; Restore the border to
black - this gives a visual
        sta VIC_BORDER_COLOR            ; on how much 'raster
time' you have to work with

        jmp MainLoop


;================================================================
============================

;================================================================
============================
        ;
ROUTINES

;================================================================
============================
        incAsm "raster.asm"                         ; raster
interrupts
        incAsm "core_routines.asm"                  ; core
framework routines
        incAsm "sprite_routines.asm"                ; sprite
handling
        incAsm "collision_routines.asm"             ; sprite
collision routines
        incAsm "screen_routines.asm"                ; screen
drawing and handling
        incAsm "Scrolling.asm"                      ; Screen
scrolling routines
        incAsm "CharPadTools.asm"                   ; Tools for
CharPad levels
        incAsm "Player.asm"                           ; Mobile
Object/Player handing


        ;-------------------------------------------------------
-----------------------------------
        ;
DISPLAY DEBUG INFO
```

```
        ;-------------------------------------------------------
-----------------------------------
DisplaySpriteInfo
                                            ; Only update if
Joystick is used
;        lda JOY_X
;        bne DisplaySpriteInfoNow
;        lda JOY_Y
;        bne DisplaySpriteInfoNow
;        rts
                                                    ; Display Sprite
debug info
DisplaySpriteInfoNow

        loadPointer WPARAM1,SCORE_SCREEN

        lda SPRITE_POS_X                            ; Display sprite
X and Y coords
        ldx #19
        ldy #7
        jsr DisplayByte


        lda SPRITE_POS_Y
        ldx #19
        ldy #18
        jsr DisplayByte

                                                    ; check the
extended x bit
        lda SPRITE_POS_X_EXTEND
        and #$01                                    ; mask bit one
        bne @extend                                 ; if it's set,
display an *

        lda #' '                                    ; if not,
display a space
        sta SCORE_SCREEN + #770
        lda #COLOR_WHITE
        sta COLOR_MEM + #770
        jmp @displayCharCoords                      ; display the
next bunch of info

@extend

        lda #'*'
        sta SCORE_SCREEN + #770
```

```
        lda #COLOR_WHITE
        sta COLOR_MEM + #770


@displayCharCoords

        lda SPRITE_CHAR_POS_Y
        ldx #19
        ldy #37
        jsr DisplayByte

        lda SPRITE_CHAR_POS_X
        ldx #19
        ldy #28
        jsr DisplayByte

        lda SPRITE_POS_X_DELTA
        ldx #20
        ldy #7
        jsr DisplayByte

        lda SPRITE_POS_Y_DELTA
        ldx #20
        ldy #18
        jsr DisplayByte

                                                                ;
Scrolling map debug info
        lda MAP_X_POS
        ldx #21
        ldy #28
        jsr DisplayByte

        lda MAP_Y_POS
        ldx #21
        ldy #37
        jsr DisplayByte

        lda SCROLL_COUNT_X
        ldx #22
        ldy #7
        jsr DisplayByte

        lda SCROLL_COUNT_Y
        ldx #22
        ldy #18
        jsr DisplayByte
```

```
        lda MAP_X_DELTA
        ldx #22
        ldy #28
        jsr DisplayByte

        lda MAP_Y_DELTA
        ldx #22
        ldy #37
        jsr DisplayByte

        lda COLLIDER_ATTR
        ldx #23
        ldy #7
        jsr DisplayByte

        rts

;=================================================================
==================================
;
PROGRAM DATA
;=================================================================
==================================

SPRITE_CONSOLE_TEXT
        byte ' xpos:$     ypos:$    chrx:$   chry:$    /'
        byte ' dltx:$     dlty:$ /'
        byte '                      mapx:$   mapy:$    /'
        byte ' sclx:$     scly:$    mdlx:$   mdly:$    /'
        byte ' attr:',0
SCROLL_CONSOLE_TEXT
        byte ' xpos:$     ypos:$    dltx:$   dlty:$    / scrlx:
scrly:    chrx:    chry:',0
;-----------------------------------------------------------------
----------------------------------
;
JOYSTICK
JOY_X                           ; current positon of Joystick(2)
        byte $00                ; -1 0 or +1
JOY_Y
        byte $00                ; -1 0 or +1

BUTTON_PRESSED                  ; holds 1 when the button is
held down
        byte $00
```

```
                                        ; holds 1 when a single press is
made (button released)
BUTTON_ACTION
        byte $00

;-------------------------------------------------------------
-----------------------------------
; Bit Table
; Take a value from 0 to 7 and return it's bit value
BIT_TABLE
        byte 1,2,4,8,16,32,64,128


*=$4000
;=============================================================
================
;                                                           VIC
MEMORY BLOCK
;                                                           CHARSET
AND SPRITE DATA
;=============================================================
================
; Charset and Sprite data directly loaded here.

VIC_DATA_INCLUDES

; VIC VIDEO MEMORY LAYOUT - BANK 1 ($4000 - $7FFF)
; SCREEN_1      = $4000 - $43FF         (Screen 0)      ; Double
buffered
; SCREEN_2      = $4400 - $47FF         (Screen 1)      ; game
screen
; MAP_CHARS     = $4800 - $5FFF         (Charset 1)     ; game
chars (tiles)
; SCORE_CHARS   = $5000 - $57FF         (Charset 2)     ;
Scoreboard chars
; SCORE_SCREEN  = $5800 - $5BFF         (Screen 6)      ;
Scoreboard Screen
; SPRITES       = $5COO - $7FFF         (144 Sprite Images)

;---------------------
; CHARACTER SET SETUP
;---------------------
; Going with the 'Bear Essentials' model would be :
;
; 000 - 063    Normal font (letters / numbers / punctuation,
sprite will pass over)
; 064 - 127    Backgrounds (sprite will pass over)
```

```
; 128 - 143    Collapsing platforms (deteriorate and eventually
disappear when stood on)
; 144 - 153    Conveyors (move the character left or right when
stood on)
; 154 - 191    Semi solid platforms (can be stood on, but can
jump and walk through)
; 192 - 239    Solid platforms (cannot pass through)
; 240 - 255    Death (spikes etc)
;
; I would prefer to follow this model for organization, but it
is useful to note that
; Charpad allows the setting the upper 4 bits of Color data
(which is ignored by the VIC)
; to use as 16 'attribute' values.  Something I am taking
advantage of.
;

*=$4800
MAP_CHAR_MEM                            ; Character set for map
screen
;incbin "MapChars.cst",0,255
;incbin "Maps/CPChar6.bin"
incbin "Maps/MWChset.bin"
*=$5000
SCORE_CHAR_MEM
incbin "ScoreChars.cst",0,255           ; Character set for
scoreboard

;-------------------------------------------------------
SPRITE DATA
*=$5C00
incbin "CaverMan.spt",1,14,true         ; Walking right (0 - 13)
incbin "CaverMan.spt",15,28,true        ; Walking left (14-27)
incbin "CaverMan.spt",29,34,true        ; idle (28,33)
incbin "CaverMan.spt",53,54,true        ; falling (34,35)
incbin "CaverMan.spt",47,50,true        ; rope climb (36-39)
incbin "Sprites.spt",1,16,true          ; Numbers 0-9 A-F
SPRITES 40 - 56

;=======================================================
====================================
;
LEVEL DATA
;=======================================================
====================================
; Each Level has a character set (2k) an attribute/color list
(256 bytes) 64 4x4 tiles (1k)
```

```
; and a 64 x 32 (or 32 x 64) map (2k).

; The current level map will be put at $8000 with Attribute
lists (256 bytes) and Tiles (1k)
; Starting after it at 8800

; With additional levels to be stored at $9000 and $C000 and if
new attributes and Tiles are needed
; I'll find a place to put them
;
; In order for the world to reset, the first map and chars will
be backed up at $D000
; under the VIC registers IO with bank switching at startup, and
restored at game restart
;
; New levels are loaded into these spaces.
;-----------------------------------------------------------------
-----------------------------------
*=$8000
MAP_MEM
;incbin"Maps/CPMap6.bin"
incbin"Maps/MW_Map1.bin"
ATTRIBUTE_MEM
;incbin "Maps/CPAttr6.bin"
incbin "Maps/MWChset_Attr1.bin"
TILE_MEM
;incbin "Maps/CPTiles6.bin"
incbin "Maps/MWTiles1.bin"
```