```
; adapted from a sample found on http://wiki.superfamicom.org

.include "header.inc"
.include "initsnes.asm"

.macro ConvertX
; Data in: our coord in A
; Data out: SNES scroll data in C (the 16 bit A)
.rept 5
asl a               ; multiply A by 32
.endr
rep #%00100000      ; 16 bit A
eor #$FFFF    ; this will do A=1-A
inc a               ; A=A+1
sep #%00100000      ; 8 bit A
.endm

.macro ConvertY
; Data in: our coord in A
; Data out: SNES scroll data in C (the 16 bit A)
.rept 5
asl a               ; multiply A by 32
.endr
rep #%00100000      ; 16 bit A
eor #$FFFF    ; this will do A=1-A
sep #%00100000      ; 8 bit A
.endm

.bank 0 slot 0
.org 0
.section "Vblank"
;--------------------------------------
VBlank:
lda $4212       ; get joypad status
and #%00000001      ; if joy is not ready
bne VBlank    ; wait
lda $4219       ; read joypad (BYSTudlr)
sta $0201       ; store it
cmp $0200     ; compare it with the previous
bne +               ; if not equal, go
rti                     ; if it's equal, then return

+ sta $0200     ; store
and #%00010000      ; get the start button
                    ; this will be the delete key
beq +               ; if it's 0, we don't have to delete
ldx #$0000
- stz $0000,x   ; delete addresses $0000 to $0008
inx
cpx #$09        ; this is 9. Guess why (homework : ) )
bne -
stz $0100       ; delete the scroll
```

```
stz $0101       ; data also

+ lda $0201    ; get back the temp value
and #%11000000       ; Care only about B and Y
beq +           ; if empty, skip this
; so, B or Y is pressed. Let's say B is O,
; and Y is X.
cmp #%11000000      ; both are pressed?
beq +           ; then don't do anything
cmp #%10000000      ; B?
bne ++          ; no, try Y
; B is pressed, write an O ($08)
; we have to tell the cursor position,
; and calculate an address from that
; Formula: Address=3*Y+X
lda $0101       ; get Y
sta $0202       ; put it to a temp value
clc
adc $0202       ; multiply by 3 - an easy way
adc $0202       ; A*3=A+A+A : )
adc $0100       ; add X
; Now A contains our address
ldx #$0000     ; be on the safe side
tax
lda #$08
sta $0000,x     ; put $08 to the good address
jmp +           ; done with this

++              ; now for Y
cmp #%01000000      ; Y?
bne +           ; no, jump forward (this should not happen)
; Y is pressed, write an X ($0A)
lda $0101       ; get Y
sta $0202       ; put it to a temp value
clc
adc $0202       ; multiply by 3 - an easy way
adc $0202       ; A*3=A+A+A : )
adc $0100       ; add X
; Now A contains our address
ldx #$0000     ; be on the safe side
tax
lda #$0A
sta $0000,x     ; put $0A to the good address
+               ; finished putting tiles

; cursor moving comes now
lda $0201       ; get control
and #%00001111       ; care about directions
sta $0201       ; store this

cmp #%00001000      ; up?
bne +           ; if not, skip
```

```
lda $0101       ; get scroll Y
cmp #$00        ; if on the top,
beq +           ; don't do anything
dec $0101       ; sub 1 from Y
+

lda $0201       ; get control
cmp #%00000100      ; down?
bne +           ; if not, skip
lda $0101
cmp #$02        ; if on the bottom,
beq +           ; don't do anything
inc $0101       ; add 1 to Y
+

lda $0201       ; get control
cmp #%00000010      ; left?
bne +           ; if not, skip
lda $0100
cmp #$00        ; if on the left,
beq +           ; don't do anything
dec $0100       ; sub 1 from X
+

lda $0201       ; get control
cmp #%00000001      ; right?
bne +           ; if not, skip
lda $0100
cmp #$02        ; if on the right,
beq +           ; don't do anything
inc $0100       ; add 1 to X
+
rti             ; F|NisH3D!
;-------------------------------------
.ends
```

; Palette 1 nach CGRAM kopieren:

```
; Dieser Code ist der Einstiegspunkt ins Programm..
; Hier wird A als 8 Bit Register und X/Y als 16 Bit Register verwendet.
; Das CGRAM- Adress-Register $2121 zeigt wegen dem Code in InitSNES schon auf Adresse 0.
; $2121 ist also ein Zeiger in den CG-RAM und er zeigt auf 0.
; Mit einem STA an das CG-RAM-Schreibe-Register $2122 werden 8 Farben aus der Datei tiles.inc
; in den Farbspeicher (CGRAM) an Adresse 0, 1, ... 7 übertragen. Bei einem Schreiben an $2122
; wird das Adress-Register automatisch erhöht. Es ist also kein 65x Opcode notwendig.
.bank 0 slot 0
.org 0
.section "Main"
;-------------------------------------
Start:
 InitSNES
rep #%00010000      ;16 bit xy
sep #%00100000      ;8 bit ab
```

```
ldx #$0000
- lda UntitledPalette.l,x
sta $2122
inx
cpx #8
bne -

; Palette 2 nach CGRAM kopieren:
; Auch hier werden Farben in den Farbspeicher geladen. Diesmal jedoch nicht an Adresse
; 0, ...7 sondern an Adresse 33 und 34.  Eine Farbe besteht übrigens aus 2 Byte. Hier wird also nur
; eine Farbe kopiert (aus Palette2)
; TODO: lda #33 verstehen.
; Meine Theorie: An Adresse 32 liegt 00, was die Hintergrundfarbe für  den zweiten Hintergrund ist.
;I'll explain this later
;We'll have two palettes, only one color is needed for the second:
lda #33                 ;The color we need is the 33rd
sta $2121
lda.l Palette2
sta $2122
lda.l Palette2+1
sta $2122

; Grafiken/Tiles nach VRAM kopieren:
; Die Pixel werden hier nicht durch direktes schreiben in den VRAM übertragen (was auch ginge),
; sondern wie DMA. Deshalb sieht der Code hier etwas anders aus. Die DMA-Register $4302,
; $4303 und $4304 speichern die 3 Byte (24 Bit) A-BUS-Adresse von der gelesen wird. In unserem
; Fall  die Symbolische Adresse UntitledData. Dort liegen die Grafiken.
; Das DMA-Register $4305 und $4306 enthält eine Zahl: Die Anzahl von Bytes die übertragen
; werden. Befinden sich also mehr Tiles an der Adresse UntitledData, so muss die Zahl in Y
; erhöht werden!
; Die Bits im DMA-Register $4300 beschrieben den Übertragungsmodus.
; In unserem Fall lda #%00000001
; Bit 7: bei 0 A to B (bei 1 wäre es umgekehrt)
; Bit 3-4: bei 00 Autoinkrement der Adresse. (bei 10 wäre es Dekrement)
; Bit 0-2: bei 001 2 Bytes zu 2 Bytes (Es werden also immer 2 Bytes übertragen)
; Erklärung: in CGRAM und OAM sind die Speicherstellen 16 Bit!
; Im „Normalen Speicher“ und VRAM 8 Bit
: Das DMA-Register $4301 setzt die Adresse für den (8 Bit) B-BUS. Also in unserem Fall
; das wir die Grafiken in den VRAM speichern wollen. Das ist NICHT die Adresse, sondern die
; Adresse des VRAM-Write Registers $2118. d.h wir schreiben eine 18!
; Die eigentliche Adresse landet im V-RAM-Adress-Register $2116.
; Nachdem alles eingestellt ist startet das schreiben an $420B den Vorgang mit Channel 0.
; (Wir haben die ganze Zeit nur mit Channel 0 (es gibt 8) gearbeitet)
; lda #%00000001: Das oberste bit ist Channel 7 … das unterste Channel 0
; TODO:  lda #:UntitledData verstehen.
; Meine Theorie: Die Syntax mit dem Doppelpunkt schnappt  sich irgendwie die obersten Bytes der
; Adresse und packt sie in A. Die untern beiden sind schon in X.
; TODO: $211[89]: Kommentar verstehen
; Meine Theorie: bezieht sich auf $2118 und $2119. Das sind die VRAM-Write und VRAM-
; Read Register.
ldx #UntitledData       ; Address
```

```
lda #:UntitledData      ; of UntitledData
ldy #(15*16*2)          ; length of data
stx $4302       ; write
sta $4304       ; address
sty $4305       ; and length
lda #%00000001      ; set this mode (transferring words)
sta $4300
lda #$18        ; $211[89]: VRAM data write
sta $4301       ; set destination
ldy #$0000      ; Write to VRAM from $0000
sty $2116
lda #%00000001      ; start DMA, channel 0
sta $420B
```

; TileMap im VRAM anlegen:

```
; Das hier wird wieder manuell (also ohne DMA gemacht). Die Map beschreibt für einen
; Hintergrund aus welchen Tiles er zusammen gesetzt ist. Tiles im BG heißen Characters.
; Das VRAM-Adress-Register $2116 wird hierbei auf $4000 gesetzt.
; Mit schreiben ins VRAM-Write Register $2118 (oder $2119) wird dann fortlaufend
; ab Adresse $4000 in den VRAM geschrieben. Hier werden nur die Nummern der Tiles
; angegeben. Später sagen wir dann dem SNES wo die Grafiken anfangen (wir haben sie im
; vorherigen abschnitt mit DMA ab Adresse 0 in den VRAm geschrieben.
; Die WLA-Assembler-Instruktionen .rep X wiederholen die Instruktionen in dem Block bis .endr
; Setzen des Wertes  10000000 an VMANIC Register $2115 erhöht automatisch die Adresse in
; $2116 um 1 wenn 2 Byte geschrieben wurden.
lda #%10000000      ; VRAM writing mode
sta $2115
ldx #$4000      ; write to vram
stx $2116       ; from $4000

;ugly code starts here - it writes the # shape I mentioned before.
.rept 2
  ;X|X|X
  .rept 2
    ldx #$0000; tile 0 ( )
    stx $2118
    ldx #$0002; tile 2 (|)
    stx $2118
  .endr
  ldx #$0000
  stx $2118
  ;first line finished, add BG's
  .rept 27
    stx $2118   ; X=0
  .endr
  ;beginning of 2nd line
  ;-+-+-
  .rept 2
    ldx #$0004; tile 4 (-)
    stx $2118
    ldx #$0006; tile 6 (+)
    stx $2118
```

```
   .endr
   ldx #$0004   ; tile 4 (-)
   stx $2118
   ldx #$0000
   .rept 27
     stx $2118
   .endr
.endr
.rept 2
  ldx #$0000    ; tile 0 ( )
  stx $2118
  ldx #$0002    ; tile 2 (|)
  stx $2118
.endr
```

;Zweite TileMap im VRAM anlegen:
; Hier wird eine 1 Byte Tilemap ab Adresse $6000 angelegt. **Das ist der Cursor.**

```
ldx #$6000      ; BG2 will start here
stx $2116
ldx #$000C      ; And will contain 1 tile
stx $2118
```

; Tile und Tilemap position dem SNES mitteilen:

```
; Farben, Tiles und Tilemap liegen im Speicher. Nun müssen wir einen Modus wählen und
; alles (je nach Modus) den Hintergründen zuweisen.
; Das Schrieben von 00110000 an das BG-Mode Register $2105 setzt das SNES auf Mode 0
; (es gibt 8 Modes) und die Charactes (Tiles)  von BG 1 und BG 2 auf 16x16
; Bit 0-2: Mode
; Bit 4-7: BG1 – BG4 Tiles size (0=8x8, 1=16x16)
; Das BG1-Soruce Register $2107 wird auf  01000000 gestezt. Damit beginnt die Tilemap
; bei $4000 im VRAM. Werte werden mit 1000 multipliziert.
; Das BG2-Soruce Register $2107 wird auf  01100000 gesetzt. Damit beginnt die Tilemap
; bei $6000 im VRAM. Werte werden mit 1000 multipliziert.
; Das BG12-Name-Base-Address-Register wird auf 0 gesetzt. Dort soll nach den Tiles geschaut
; werden und dort haben wir die vorhin mit DMA auch hingeschrieben.
; Ein schreiben an das Though-Main Register $212C setzt Hintergrund BG1 und B2 auf Sichtbar
: Die Bits (von 0 nach 5): BG1, BG2, BG3, BG4, OBJ
; Das Register $210E bestimmt den V-Scroll Wert von B1
; Das Register $2110 bestimmt den V-Scroll Wert von B2
; Das INITDISP Register $2100 setzt die Helligkeit (bit0-3) und schaltet das Forced-Blanking aus
; TODO: ich komme hier nicht auf $4000 und $6000
: Theorie: Evtl nur hohen Bytes? Also 0100 = 4 und 0110=6 (passt nicht zum SNES manual)
; TODO: nicht ganz klar was die Scroll-Register exakt machen
; Theorie: Der Kursor wird durch Scrollen des ganzen BG2 bewegt. Evtl. beeinflusst der
; Wert hier die Richtung und weite der Bewegung? Vermutlich wird hier nur an der oberen
; linken Ecke ausgerichtet (das ist komplett geraten)
;set up the screen
lda #%00110000      ; 16x16 tiles, mode 0
sta $2105       ; screen mode register
lda #%01000000      ; data starts from $4000
sta $2107       ; for BG1
lda #%01100000      ; and $6000
```

```
sta $2108       ; for BG2
stz $210B       ; BG1 and BG2 use the $0000 tiles
lda #%00000011      ; enable bg1&2
sta $212C

;The PPU doesn't process the top line, so we scroll down 1 line.
rep #$20        ; 16bit a
lda #$07FF      ; this is -1 for BG1
sep #$20        ; 8bit a
sta $210E       ; BG1 vert scroll
xba
sta $210E
rep #$20        ; 16bit a
lda #$FFFF      ; this is -1 for BG2
sep #$20        ; 8bit a
sta $2110       ; BG2 vert scroll
xba
sta $2110
lda #%00001111      ; enable screen, set brightness to 15
sta $2100

lda #%10000001      ; enable NMI and joypads
sta $4200

; Das ist die Hauptschleife. Alles hiervor in Main wird nur einmal gemacht. Das hier wiederholt
; sich. Mit Hilfe der Marco ConvertX und ConvertY wird hier der X/Y Wert aus dem WRAM
; gelesen (unser selbstgebauter Spielzustand) und der Hintergrund BG2 entsprechend verschoben.
; Es bewegt sich also NICHT die Grafik (der Cursor), sondern der gesamte Hintergrund!
forever:
wai
rep #%00100000      ; get 16 bit A
lda #$0000          ; empty it
sep #%00100000      ; 8 bit A
lda $0100           ; get our X coord
 ConvertX           ; WLA needs a space before a macro name
sta $210F           ; BG2 horz scroll
xba
sta $210F           ; write 16 bits

;now repeat it, but change $0100 to $0101, and $210F to $2110
rep #%00100000      ; get 16 bit A
lda #$0000          ; empty it
sep #%00100000      ; 8 bit A
lda $0101           ; get our Y coord
 ConvertY           ; WLA needs a space before a macro name
sta $2110           ; BG2 vert scroll
xba
sta $2110           ; write 16 bits
;-------------------------------------
ldx #$0000          ; reset our counter
-
rep #%00100000              ; 16 bit A
```

```
lda #$0000              ; empty it
sep #%00100000                  ; 8 bit a
lda VRAMtable.l,x       ; this is a long indexed address, nice : )
rep #%00100000
clc
adc #$4000              ; add $4000 to the value
sta $2116               ; write to VRAM from here
lda #$0000              ; reset A while it's still 16 bit
sep #%00100000                  ; 8 bit A
lda $0000,x             ; get the corresponding tile from RAM
; VRAM data write mode is still %10000000
sta $2118               ; write
stz $2119               ; this is the hi-byte
inx
cpx #9                  ; finished?
bne -                   ; no, go back
jmp forever

;-------------------------------------
.ends

; Hier endet die Hauptschleife und der ganze MAIN Block
; Das ist alles WLA spezifisch und hat nix mit dem SNES direkt zu tun.
; Tile-Daten einbinden.
.bank 1 slot 0          ; We'll use bank 1
.org 0
.section "Tiledata"
.include "tiles.inc"    ; If you are using your own tiles, replace this
.ends

.bank 2 slot 0
.org 0
.section "Conversiontable"
VRAMtable:
.db $00,$02,$04,$40,$42,$44,$80,$82,$84
.ends
```