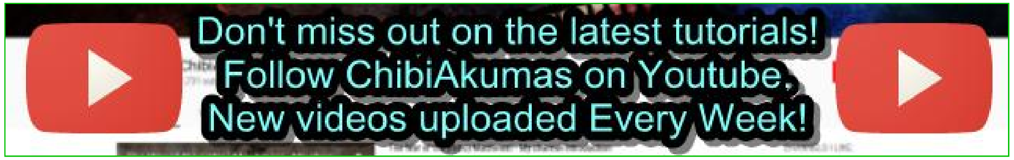


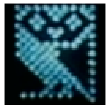
View Options
Default Dark
Simple (Hide this menu)
Print Mode (white background)

Top Menu
Main Menu
Youtube channel
Forum
AkuSprite Editor
Dec/Bin/Hex/Oct/Ascii Table

Z80 Content
Z80 Tutorial List
Learn Z80 Assembly
Hello World
Advanced Series
Multiplatform Series
Platform Specific Series
ChibiAkumas Series
Grime Z80
Z80 Downloads
Z80 Cheatsheet
Sources.7z
DevTools kit
Z80 Platforms
Amstrad CPC
Elan Enterprise
Gameboy & Gameboy Color
Master System & GameGear
MSX & MSX2
Sam Coupe
TI-83
ZX Spectrum

Learn Multi platform 6502 Assembly Programming... For Monsters!

Hello World Series



Lesson H1 - Hello World on the BBC Micro!

In this episode we'll learn how to create a simple Hello world example on the BBC... To make things easy, we'll use the firmware functions to print characters.
We'll then compile our program and transfer it to a disk image.

Lets learn how!

BBC_HelloWorld.asm

Showing A 'Hello World' Message

Starting our program, we're going to define the Origin at $3000

We're also going to define two symbols from the firmware to do tasks for use:
$FFE3 will print a character in A to the screen
$FFE7 will start a new line.

```
PrintChar equ $FFE3 ;OSASCI - Print Ascii Character to scrn
NewLine   equ $FFE7 ;OSNEWL - New Line

    ORG $3000          ;Start of our program code.
```

We're going to 'extend' this to make a PrintString routine...

We'll use Zeropage pair $20/1 to store an address which will point to a string in memory...

We use Y as an offset to the start address, and We'll print characters to the screen, until we get a character 255...

```
PrintStr:
    ldy #0                  ;Set Y to zero
PrintStr_again:
    lda ($20),y             ;Load a character from addr in $20+Y

    cmp #255                ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar           ;Print Character
    iny                     ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

HelloWorld:                 ;255 terminated string
    db "Hello World",255
```

We need to load the High and Low bytes of our address into the $20/1 zero page entries to define the address of our string, and then call our PrintString function - this will show our string to the screen.

```
    ;Load in the address of the Message into the zero page
    lda #>HelloWorld
    sta $21                 ;H Byte
    lda #<HelloWorld
    sta $20                 ;L Byte

    jsr PrintStr            ;Show to the screen

    jsr NewLine             ;Start a new line

    rts                     ;Return to basic
```

Our Hello World message will be shown to screen.

Getting Hello World to the screen isn't much, but it's a vital step! Once we can get a program running, we can develop it into something much better.

In our Bitmap Series, we went directly to the graphics hardware and used our own font, but We've used the firmware in this example for speed.

Running our Program

We need to compile our program using VASM... we need to specify some command switches

```
\vasm6502_oldstyle_win32.exe %BuildFile% -chklabels -nocase -Dvasm=1 -L \BldBBC\Listing.txt -DBuildBBC=1 -Fbin -o "\BldBBC\$.Boot"
```

We need to specify a **ASM file to build**
We need to specify an **output file**, and that we need it to be **Binary**...
We're specifying some **Symbols** we want defined on the command line... you probably don't need these
We're also outputting a **listing file** ... this is for debugging.
We're also disabling **Case sensitivity**, and telling VASM to **check our labels don't look like commands** (Usually because we've missed a tab!)

Once we've built our binary, we need to get it into a disk to run on the bbc, we use **BBCIM** to do this

Spectrum NEXT
Camputers Lynx

6502 Content
6502 Tutorial List
Learn 6502 Assembly
Advanced Series
Platform Specific Series
Hello World Series
Grime 6502
6502 Downloads
6502 Cheatsheet
Sources.7z
DevTools kit
6502 Platforms
Apple IIe
Atari 800 and 5200
Atari Lynx
BBC Micro
Commodore 64
Commander x16
Super Nintendo (SNES)
Nintendo NES / Famicom
PC Engine (Turbografx-16)
Vic 20

68000 Content
68000 Tutorial List
Learn 68000 Assembly
Hello World Series
Platform Specific Series
Grime 68000
68000 Downloads
68000 Cheatsheet
Sources.7z
DevTools kit
68000 Platforms
Amiga 500
Atari ST
Neo Geo
Sega Genesis / Mega Drive
Sinclair QL
X68000 (Sharp x68k)

8086 Content
Learn 8086 Assembly
Platform Specific Series

First we **add** our $.Boot file to the disk (the name is important!)

```
rem Add file to disk image
\Utils\BBCIM.EXE -a disk.ssd $.Boot
```

Next we set the disk to **autoboot**.

```
rem Set Boot Option
\Utils\BBCIM.EXE -boot disk.ssd RUN
```

We can use the command line to start the disk image with BeebEm

```
copy disk.ssd \RelBBC
\Emu\BeebEm\BeebEm.exe \RelBBC\disk.ssd
```

Debugging Tools

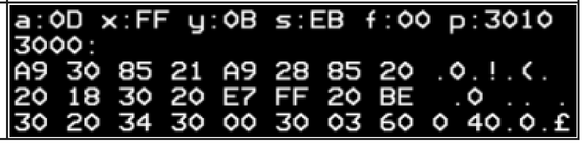
It's relatively easy to add support for my 'Monitor Tools' ... these will allow you to see the state of the processor or memory easily.

```
    ;Basic macros for ASM tasks
    include "\SrcAll\BasicMacros.asm"

z_Regs      equu $20
SPpage equ $0100

   ;Debugging tools
   include "\SrcAll\monitor.asm"
   ;Basic commands for ASM tasks
   include "\SrcAll\BasicFunctions.asm"
```

We can use the Monitor to see the processor registers, or specify a memory address, and a number of lines to show.

```
  jsr monitor              ;Show registers to screen

  jsr MemDump              ;Show Some Ram to screen
  word $3000                   ;Address to show
  byte $3                      ;Lines
```

We will see the result to screen.



Lesson H2 - Hello World on the C64

Lets learn how to show a Hello World message on the C64... we'll learn how to build our example as a PRG and an CRT Cartridge

C64_HelloWorld.asm

Showing Hello World to the screen

We're going to create a PRG file... these need a header to start the program - we'll never need to change this provided we don't want to change the start address,

Ours starts at $0810

```
;Init Routine
*=$0801
    db $0E,$08,$0A,$00,$9E,$20,$28,$32,$30,$36,$34,$29,$00,$00,$00
*=$0810 ;Start at $0810
```

We're going to use the firmware function $FFD2 to print characters (known as ChrOut)

Unfortunately this function does not use normal ASCII! and it doesn't have lower case letters... we'll need to do some converting to fix this!

```
PrintChar:  ;DefaultFont
    cmp #96                 ;Check if character >96
    bcc PrintCharOK
    and #%11011111          ;Convert to uppercase
PrintCharOK:
    jmp $ffd2               ;CHROUT - Output a character
```

We're going to 'extend' this to make a PrintString routine...

Hello World Series
8086 Downloads
8086 Cheatsheet
Sources.7z
DevTools kit
8086 Platforms
Wonderswan
MsDos

ARM Content
Learn ARM Assembly
Platform Specific Series
ARM Downloads
ARM Cheatsheet
Sources.7z
DevTools kit
ARM Platforms
Gameboy Advance
Nintendo DS
Risc Os

Risc-V Content
Learn Risc-V Assembly
Risc-V Downloads
Risc-V Cheatsheet
Sources.7z
DevTools kit

PDP-11 Content
Learn PDP-11 Assembly
PDP-11 Downloads
PDP-11 Cheatsheet
Sources.7z
DevTools kit

TMS9900 Content
Learn TMS9900 Assembly
TMS9900 Downloads
TMS9900 Cheatsheet
Sources.7z
DevTools kit
TMS9900 Platforms
Ti 99

6809 Content
Learn 6809 Assembly
6809 Downloads

We'll use Zeropage pair $20/1 to store an address which will point to a string in memory...

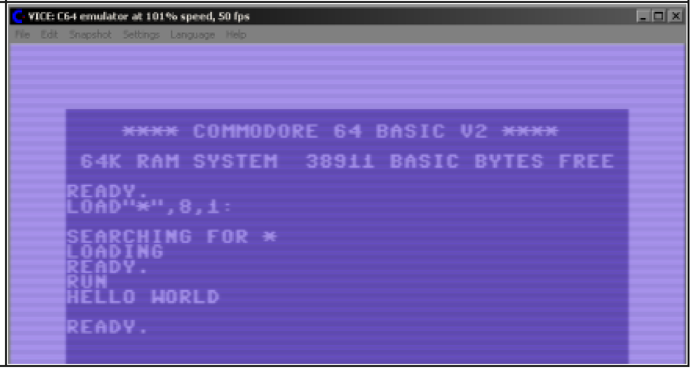
We use Y as an offset to the start address, and We'll print characters to the screen, until we get a character 255...

```
PrintStr:
    ldy #0                  ;Set Y to zero
PrintStr_again:
    lda ($20),y             ;Load a character from addr in $20+Y

    cmp #255                ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar           ;Print Character
    iny                     ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

HelloWorld:                 ;255 terminated string
    db "Hello World",255
```

We need to load the High and Low bytes of our address into the $20/1 zero page entries to define the address of our string, and then call our PrintString function - this will show our string to the screen.

```
    ;Load in the address of the Message into the zero page
    lda #>HelloWorld
    sta $21                 ;H Byte
    lda #<HelloWorld
    sta $20                 ;L Byte

    jsr PrintStr            ;Show to the screen

    jsr NewLine             ;Start a new line

    rts                     ;Return to basic
```

Our Hello World message will be shown to screen



Upper and Lower case fonts

The C64 has an alternate font, which while not ASCII allows for upper and lower case.

To enable it we just write character $0E (14) to the screen

```
    lda #$0e            ;Full Charset (not just uppercase)
    jsr $ffd2           ;CHROUT - Output a character
```

We need a different PrintChar routine for the new font.

```
PrintChar:  ;Upper+LowerCase Font
    cmp #64                 ;Check if character >96
    bcc PrintCharOKB
    eor #%00100000          ;Convert to uppercase
PrintCharOKB:
    jmp $ffd2               ;CHROUT - Output a character
```

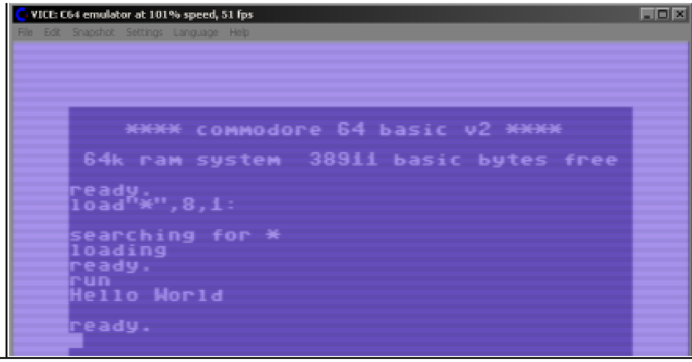
We can now print Upper and Lower Case!

6809/6309 Cheatsheet
Sources.7z
DevTools kit
6809 Platforms
Dragon 32/Tandy Coco
Fujitsu FM7
TRS-80 Coco 3
Vectrex

My Game projects
Chibi Aliens
Chibi Akumas

Work in Progress
Learn 65816 Assembly
Learn eZ80 Assembly

Misc bits
Ruby programming



Building a PRG file with Vasm

I use VASM to compile the ASM file into a usable PRG

```
n\vasm6502_oldstyle_win32.exe %1 -cbm-prg -chklabels -nocase -L \BldC64\Listing.txt -Dvasm=1 -DBuildC64=1 -Fbin -o "\BldC64\Program.prg"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** that's a **PRG** file
We need to specify the **Destination file name**
We include **some symbols** (some of my code uses these - you won't need them)
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)

We can start the VICE emulator with the PRG from the command line	`x64.exe \BldC64\Program.prg`

Converting to a cartridge

We can convert our PRG to a cartridge instead - we need to replace the PRG header with a cartridge one,

The header shown should work OK

```
; CRT format cartridge header
org $7FB0
byte "C64 CARTRIDGE   "              ; Cartridge Signature
byte $00,$00,$00,$40                 ; Header length $00000040
byte $01,$00                         ; Version (1.00)
byte $00,$00                         ; Cartridge Type... $0000 = normal
byte $00                             ; Exrom Status... $00 = none
byte $00                             ; Game Line Status... $00 = none
byte $00,$00,$00,$00,$00,$00         ; Unused
;     12345678901234567890123456789012
byte "CHIBIAKUMAS.COM                 "  ; 32 byte cartridge name

;******************************
; Chip Packet Header ($10)
;******************************

org $7FF0
byte "CHIP"
byte $00, $00, $40, $10   ; Chip Packet Length $00002010
byte $00, $00             ; Chip type 0 = ROM, 1 = RAM
byte $00, $00             ; Bank Location $0000 = normal cartridge
byte $80, $00             ; Load location $8000
byte $40, $00             ; Rom image size $4000

org $8000                 ;Start of ROM

word Startup; Startup Vector
word Startup; Restore Vector
byte $C3, $C2, $CD, $38, $30
```

Our ROM will effectively bypass basic, so we need to do the tasks BASIC was previously doing, the calls shown will do this

We can then just include our program as usual

```
Startup:

    jsr $FF84        ;IOINIT. Initialize CIA's, SID
    jsr $FF87        ;RAMTAS. Clear memory addresses

    jsr $FF8A        ;RESTOR. Fill vector table
    jsr $FF81        ;SCINIT. Initialize VIC;

;Your Program Starts Here!
```

We still use VASM to assemble, we just remove the switch -CBM-PRG, and change the output file to a CRT

```
\vasm6502_oldstyle_win32.exe %1 -chklabels -nocase -Dvasm=1 -L \BldC64\Listing.txt -DBuildC64=1 -DBuildC64_CRT=1 -Fbin -o "\BldC64\Program.CRT"
```

We need to pad our cartridge to 16k, My BinaryTools program can do this:

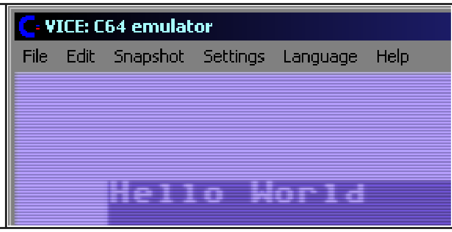
```
\Utils\BinaryTools.exe fill \BldC64\Program.CRT 16464 1 0
```

We also Change our VICE command line:

```
x64.exe -cartcrt "\BldC64\Program.CRT"
```

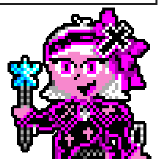
"Look MA! no basic!"

We've run our program straight from a cartridge!



Running from a cartridge may make things easier, as our program will no longer be running from low memory (0-$3FFF)..

This area is used by the screen hardware,so it's 'premium' storage space!



Recent New Content
Amiga - ASM PSET and POINT for Pixel Plotting
Learn 65816 Assembly: 8 and 16 bit modes on the 65816

Debugging Tools

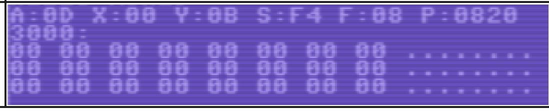
It's relatively easy to add support for my 'Monitor Tools' ... these will allow you to see the state of the processor or memory easily.

```
z_Regs       equ $20          ;Fake Registers
SPpage       equ $0100        ;Stackpointer Address

    ;Basic macros for ASM tasks
    include "\SrcAll\BasicMacros.asm"


 include "\SrcAll\monitor.asm"          ;Debugging tools
 include "\SrcAll\BasicFunctions.asm"   ;Basic commands for ASM tasks
```

We can use the Monitor to see the processor registers, or specify a memory address, and a number of lines to show.

```
  jsr monitor           ;Show registers to screen

  jsr MemDump           ;Show Some Ram to screen
  word $3000                ;Address to show
  byte $3                   ;Lines
```

We will see the result to screen.



Lesson H3 - Hello World on the VIC-20

Being it's predecessor, The Vic 20 shares many of the basics of the c64...

For this reason, Writing Hello World on the VIC is pretty similar, with just a few changes... Lets learn how!

VIC_HelloWorld.asm

Showing Hello World to the screen

We're going to create a PRG file... these need a header to start the program - we'll never need to change this provided we don't want to change the start address,

Ours starts at $100A

```
* = $1001
        ; BASIC program to boot the machine language code
        db $0b, $10, $0a, $00, $9e, $34, $31, $30, $39, $00, $00, $00
```

We're going to use the firmware function $FFD2 to print characters (known as ChrOut)

Unfortunately this function does not use normal ASCII! and it doesn't have lower case letters... we'll need to do some converting to fix this!

```
PrintChar:  ;DefaultFont
    cmp #96               ;Check if character >96
    bcc PrintCharOK
    and #%11011111        ;Convert to uppercase
PrintCharOK:
    jmp $ffd2             ;CHROUT - Output a character
```

We're going to 'extend' this to make a PrintString routine...

We'll use Zeropage pair $20/1 to store an address which will point to a string in memory...

We use Y as an offset to the start address, and We'll print characters to the screen, until we get a character 255...

```
PrintStr:
    ldy #0                ;Set Y to zero
PrintStr_again:
    lda ($20),y           ;Load a character from addr in $20+Y

    cmp #255              ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar         ;Print Character
    iny                   ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

HelloWorld:               ;255 terminated string
    db "Hello World",255
```

We need to load the High and Low bytes of our address into the $20/1 zero page entries to define the address of our string, and then call our PrintString function - this will show our string to the

SNES - ASM PSET and POINT for Pixel Plotting

ARM Assembly Lesson H3

Lesson P65 - Mouse reading on the Sam Coupe

Mouse Reading in MS-DOS

Risc-V Assembly Lesson 3 - Bit ops and more maths!

Mouse reading on the MSX

Hello World on RISC-OS

Atari 800 / 5200 - ASM PSET and POINT for Pixel Plotting

Apple 2 - ASM PSET and POINT for Pixel Plotting

Making a 6502 ASM Tron game...

Photon1 - Introduction and Data Structures

Gaming + more:

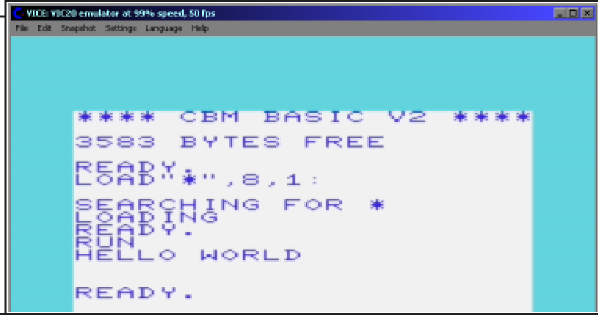
Emily The Strange (DS) - Live full playthrough

$150 calculator: Unboxing the Ti-84 Plus CE (eZ80 cpu)

screen.

```
;Load in the address of the Message into the zero page
lda #>HelloWorld
sta $21             ;H Byte
lda #<HelloWorld
sta $20             ;L Byte

jsr PrintStr        ;Show to the screen

jsr NewLine         ;Start a new line

rts                 ;Return to basic
```

Our Hello World message will be shown to screen



Building a PRG file with Vasm

I use VASM to compile the ASM file into a usable PRG

```
vasm6502_oldstyle_win32.exe %1 -cbm-prg -chklabels -nocase -Dvasm=1 -L \BldVIC\Listing.txt -DBuildVIC=1 -Fbin -o "\BldVIC\Program.prg"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** that's a **PRG** file
We need to specify the **Destination file name**
We include **some symbols** (some of my code uses these - you won't need them)
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)

We can start the VICE emulator with the PRG from the command line

```
xvic.exe \BldVIC\Program.prg
```

Converting to a cartridge

We can convert our PRG to a cartridge instead - we need to replace the PRG header with a cartridge one,

The header shown should work OK

```
* = $A000
    ;Cartridge Header
    dw ProgramStart
    dw ProgramStart
    db $41,$30,$C3,$C2,$CD        ;ROM Header
ProgramStart:
```

Our ROM will effectively bypass basic, so we need to do the tasks BASIC was previously doing, the calls shown will do this

We can then just include our program as usual

```
  ;Initialise hardware (Basic normally does this)
  JSR $FD8D        ;RAMTAS - Initialise System Constants
  JSR $FD52        ;RESTOR - Restore Kernal Vectors (at 0314)
  JSR $FDF9        ;IOINIT - Initialize I/O registers
  JSR $E518        ;CINT1  - Initialize I/O
```

We still use VASM to assemble, we just remove the switch -CBM-PRG, and change the output file to a CRT

```
\vasm6502_oldstyle_win32.exe %1 -chklabels -nocase -Dvasm=1  -L \BldVIC\Listing.txt -DBuildVIC=1 -DBuildVIC_Rom=1 -Fbin -o "\BldVIC\Program.rom"
```

We need to pad our cartridge to 8k, My BinaryTools program can do this:

```
\Utils\BinaryTools.exe fill \BldVIC\Program.rom 8191 1 0
```

We also Change our VICE command line:

```
x64.exe -cartcrt "\BldC64\Program.CRT"
```

"Look MA! no basic!"

We've run our program straight from a cartridge!

The VIC has very little RAM, so using cartridges will make things a lot easier, and is something you'll almost certainly want to do, unless your program is very small and simple.

Grime 6502 was too big for a PRG, and that was tiny!

Debugging Tools

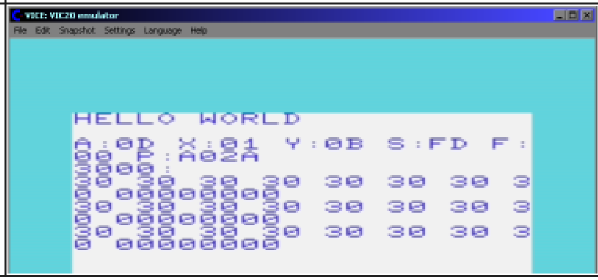
It's relatively easy to add support for my 'Monitor Tools' ... these will allow you to see the state of the processor or memory easily.

```
z_Regs      equ $20           ;Fake Registers
SPpage      equ $0100         ;Stackpointer Address

    ;Basic macros for ASM tasks
    include "\SrcAll\BasicMacros.asm"


include "\SrcAll\monitor.asm"           ;Debugging tools
include "\SrcAll\BasicFunctions.asm"    ;Basic commands for ASM tasks
```

We can use the Monitor to see the processor registers, or specify a memory address, and a number of lines to show.

```
  jsr monitor            ;Show registers to screen

  jsr MemDump            ;Show Some Ram to screen
  word $3000                ;Address to show
  byte $3                   ;Lines
```


We will see the result to screen.



Lesson H4 - Hello World on the Atari 800 / 5200

The Atari 5200 and 800 are almost the same, we just need to change some addresses so we can work with both.

Lets make a hello world for these systems!

A52_HelloWorld.asm

Setting up the Cartridge and initializing the screen.

We're going to need some defined symbols.

The GTIA is at a different memory address on the 5200 and 8000 - it handles some of the graphics functions.
The Font's Character address is also different.
Finally the Cartridge starts at a different memory address.

We also need two zero page entries for the X and Y cursor positions.

```
Cursor_X equ $40
Cursor_Y equ $41

    ifdef BuildA80          ;Atari 800 settings
GTIA equ $D000              ;GTIA address
ChrAddrH equ $E0            ;Font at $E000
    org     $A000           ;Start of cartridge area

    else                    ;Atari 5200 settings
GTIA equ $C000              ;GTIA address
ChrAddrH equ $F8            ;Font at $F800
    org     $4000           ;Start of cartridge area
    endif
```

We need to set up the screen...

The graphics display is defined by a 'display list' (it defines the screen settings of each line of the screen)... we need to point to this display list, and set $D402/3 to the 16 bit address (labeled 'DisplayList' in our code)

We need to set the 'Character Base' - this is the address of the Font in Ram/Rom

Now we need to enable the screen... we do this by seting bits 1 and 5 of $D400

Finally we set the colors of the background and foreground - these are done by the GTIA (the address is different on the 800/5200)

```
ProgramStart:
    sei                     ;Disable interrupts

    lda #<DisplayList
    sta $D402               ;DLISTL - Display list lo
    lda #>DisplayList
    sta $D403               ;DLISTH - Display list hi

    lda #ChrAddrH
    sta $D409               ;CHBASE - Character set base

    lda #%00100010
    sta $D400               ;DMACTL - DMA Control (screen on)

    lda #$0F                ;Set color PF1 (foreground)
    sta GTIA+ $17           ;COLPF1 equ

    lda #$00                ;Set color PF2 (background)
    sta GTIA+ $18           ;COLPF2
```

The Display List needs to be a fairly fixed format... we're defining all the lines as screen mode 2.

You won't want to change any of this unless you're trying to do something clever - so you should probably leave it alone!

```
Smode equ 2
DisplayList:                    ;Display list data
    db $70,$70,$70;$70 7= 8 blank lines 0= blank lines
    db $40+2                    ;$40+2

    dw $1800                    ;Screen starts at &1800

    db $02,$02,$02,$02,$02,$02  ;Screen mode (2) lines
    db $02,$02,$02,$02,$02,$02
    db $02,$02,$02,$02,$02,$02
    db $02,$02,$02,$02,$02

    db $41                      ;Loop
    dw DisplayList
```

Finally we need a footer for the cartridge... it has to be at $BFFD, the first byte should be $FF... next is the address of the start of code to execute.

```
;   Rom Header
        org $bffd
        db $FF              ;Disable Atari Logo
        dw ProgramStart;program Start
```

Showing Hello World to the screen

If we want to print a character to the screen, we need to set a byte of the screen memory to the character number.

To calculate the memory address to change for a XY position, we need to use the formula below:

Address = ScreenBase + (Ypos * ScreenWidth) + Xpos
Address = $1800 + (Ypos * 40) + Xpos

To effect the multiplication, we do bitshifting... to 'Multiply' by 40, we bitshift to get Y*8, then to Y*32 - and add the two together!

We calculate the memory address of the next character location

```
PrintChar:
    sta $24             ;Character to show
    txa
    pha
        lda #0          ;Zero High byte (Use A as Low)
        sta $23

        lda Cursor_Y
        ;Y* 40 =        %00101000    (Y*8 + Y*32)
        asl             ;00000001
        rol $23         ;00000010    *2
        asl
        rol $23         ;00000100    *4
        asl
        rol $23         ;00001000    *8
        sta $22

        lda $23         ;Back up Y*8 for later
        sta $25

        lda $22         ;00001000
        asl
        rol $23         ;00010000    *16
        asl
        rol $23         ;00100000    *32

        clc
        adc Cursor_X    ;Add Xpos
        adc $22
        sta $22

        lda $23         ;Get Y*32
        adc $25         ;Add Y*8... Result=Y*40

        ora #$18        ;Screen Base at $1800
        sta $23
```

We now know know the position to change (in zero page entry $22/3)

Unfortunately the font is not ASCII - it has no lowercase letters - we can convert the se by subtracting 32 ($20) from the character number when the character is over 96

After writing our character onscreen, we Increment our X position, and check if we're at the end of the line.

```
        lda $24         ;Get back Character to show
        cmp #96         ;Check if character >96
        bcs PrintCharOK
        sec
        sbc #$20        ;Fix Uppercase
PrintCharOK:
        ldx #0
        sta ($22,x)     ;Store in video memory

        inc Cursor_X    ;Inc Xpos
        lda Cursor_X

        cmp #40         ;Screen is 40 chars wide
        bne PrintChar_NotNextLine
        jsr NewLine
PrintChar_NotNextLine:
    pla
    tax
    rts
```

We can extend this function into a 'PrintString routine'

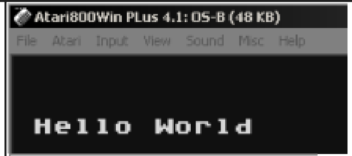
We print consecutive characters to the screen, until we get a character 255.

```
NewLine:
    lda #0
    sta Cursor_X          ;Reset X
    inc Cursor_Y          ;Inc Y
    rts


PrintStr:
    ldy #0                ;Set Y to zero
PrintStr_again:
    lda ($20),y           ;Load a character from addr in $20+Y

    cmp #255              ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar         ;Print Character
    iny                   ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts
```

To print a string We load it's address into zero page entries $20/1, before calling 'PrintStr'

```
 ;Load in the address of the Message into the zero page
 lda #>HelloWorld
 sta $21              ;H Byte
 lda #<HelloWorld
 sta $20              ;L Byte

 jsr PrintStr         ;Show to the screen
```

The code can work on the Atari 800 or 5200 - we just need to define symbol 'BuildA80' for the Atari 800



Building and running our cartridge

I compile the code with VASM in a batch file.

```
\vasm6502_oldstyle_win32.exe %1 -chklabels -nocase -Dvasm=1 -DBuildA80=1 -L \BldA52\Listing.txt -Fbin -o "\BldA52\Program.rom"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** that's a **PRG** file
We need to specify the **Destination file name**
We include **some symbols** (You'll need BuildA80 if you're building for Atari 800)
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)

We can start the cartridge with the emulator of our choice.

```
Jum52_Win32.exe "\BldA52\Program.rom"


Atari800Win.exe -atari -cart "\BldA52\Program.rom"
```

Debugging Tools

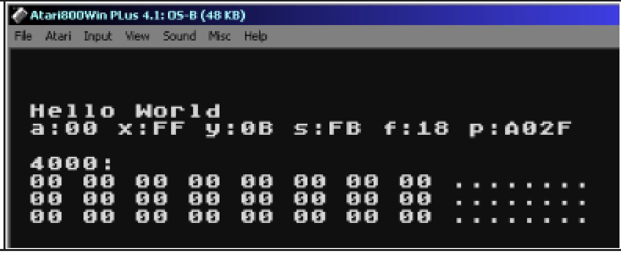
It's relatively easy to add support for my 'Monitor Tools' ... these will allow you to see the state of the processor or memory easily.

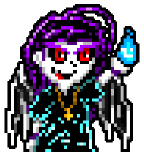
```
z_Regs      equ $20          ;Fake Registers
SPpage      equ $0100        ;Stackpointer Address

    ;Basic macros for ASM tasks
    include "\SrcAll\BasicMacros.asm"

 include "\SrcAll\monitor.asm"          ;Debugging tools
 include "\SrcAll\BasicFunctions.asm"   ;Basic commands for ASM tasks
```

We can use the Monitor to see the processor registers, or specify a memory address, and a number of lines to show.

```
  jsr monitor           ;Show registers to screen

  jsr MemDump           ;Show Some Ram to screen
  word $3000                ;Address to show
  byte $3                   ;Lines
```

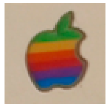
We will see the result to screen.





The 5200 and 800 are almost the same system - the only reason they don't work the same is so people couldn't buy console games for the home computer - the whole decision was a big evil scheme by the accountants !

What can I say... Whoever came up with the idea of moving the GTIA, they deserve a slow painful death!!



Lesson H5 - Hello World on the Apple II

Lets take a look at the Apple II this time, it's OS will be able to help us get text to the screen, so making 'Hello World' should be pretty easy!

Lets learn how!

AP2_HelloWorld.asm

Showing Hello World on the Apple II

Ok, let's start our program!

We're going to start our program at $0C00, and we're going to define a symbol which we'll use as the newline command...
We'll need that new line straight away as the program will start with the cursor still on the line that ran the program, for clarity we'll start a new line.

```
NewLine equ $FC62    ;CR - Carriage Return to Screen

    ORG $0C00       ;Program Start

    jsr NewLine           ;Start a new line
```

We're going to be using a pair of firmware functions to help us in this episode...

We'll use $FC62 to start a new line.
We'll use $FDF0 to draw a character... unfortunately, the Apple II fonts are a little weird, but we'll fix them by adding 128 to the character number, which will solve the problem!

```
PrintChar:
    pha
        clc
        adc #128        ;Correction for weird character map!
        jsr $FDF0       ;COUT1 - Output Character to Screen
    pla
    rts
```

We'll Extend this PrintChar routine into a PrintString routine.

We use CHR 255 terminated strings in this tutorial.

```
PrintStr:
    ldy #0                  ;Set Y to zero
PrintStr_again:
    lda ($40),y             ;Load a character from addr in $20+Y

    cmp #255                ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar           ;Print Character
    iny                     ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts
```

We use this NewLine Function to show our hello world message to the screen.

Once we've shown our message, we're done... so we just return to basic with a RET command!

```
    ;Load in the address of the Message into the zero page
    lda #>HelloWorld
    sta $41                 ;H Byte
    lda #<HelloWorld
    sta $40                 ;L Byte

    jsr PrintStr            ;Show to the screen
    rts

HelloWorld:                 ;255 terminated string
    db "Hello World",255
```

Our text will be shown to the screen.

Building our program onto a Disk on the Apple II

I build my files with VASM via a batch file.

```
\vasm6502_oldstyle_win32.exe %1 -c02 -chklabels -nocase -Dvasm=1 -L \BldAP2\Listing.txt -DBuildAP2=1 -Fbin -o "\BldAP2\Prog."
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** file
We need to specify the **Destination file name**
We include **some symbols**
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)
The Apple II has a 65c02 CPU - to enable the extra features we use the **-c02** switch

We're going to start with a **template blank disk**, we'll add our file to it

To **add the file to our disk**, We use the A2In function of **A2Tools**

Finally we'll **start our AppleWin emulator** with the Disk image - we'll also use a save state to speed things up (so we don't need to type the BRUN command)

```
copy \ResAp2\blank.dsk \BldAP2\Test.dsk

cd \BldAP2

\Utils\a2in b.0C00 TEST.DSK PROG prog

copy \BldAP2\Test.dsk \RelAP2\Test.dsk

cd \Emu\AppleWin\
copy Test.aws.yaml \RelAP2\SaveState.aws.yaml

echo *** type: BRUN PROG ***
\Emu\AppleWin\Applewin.exe -d1 \RelAP2\Test.dsk -load-state \RelAP2\SaveState.aws.yaml
```

Debugging Tools

We can use the Debugging tools that were build in the Multiplatform series, we just need to include a few files and settings.

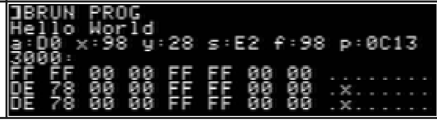
```
 ;Basic macros for ASM tasks
 include "\SrcAll\BasicMacros.asm"

z_Regs equ $40
SPpage equ $0100

 ;Debugging tools
 include "\SrcAll\monitor.asm"
 ;Basic commands for ASM tasks
 include "\SrcAll\BasicFunctions.asm"
```

We can use our 'Monitor' function to show the registers,

We can use the MemDump function to show an area of memory.

```
 jsr monitor          ;Show registers to screen

 jsr MemDump          ;Show Some Ram to screen
 word $3000             ;Address to show
 byte $3                ;Lines
```

We'll see the register contents, and the memory area we chose.



We've got some text to the screen without too much difficulty, We've covered bitmap fonts in the Platform specific series... Next time in the Simple series, we'll learn how to get bitmaps to the screen.

Lesson H6 - Hello World on the Atari Lynx

The Lynx doesn't have any firmware to help us, so we'll have to use a bitmap font to show our 'Hello World' message

Lets learn how to make a simple Lynx Cartridge Lets Learn how!

LNX_HelloWorld.asm

Starting a Lynx Cartridge

Our cartridge needs a header
This will start our program in ram at address $0200

```
 org $200-10
 db $80,$08,$02,$00,$40,$0A,$42,$53,$39,$33
 ;Our program starts at $0200
```

We're going to need some zero page values for our work.
We'll define these using symbols

```
z_Regs          equ $20

z_HL   equ z_Regs          ;Zeropage Values for our use
z_L    equ z_Regs
z_H    equ z_Regs+1
z_DE   equ z_Regs+4
z_E    equ z_Regs+4
z_D    equ z_Regs+5
z_As   equ z_Regs+6
z_ixl  equ z_Regs+8

Cursor_X        equ $40       ;Text position for next char
Cursor_Y        equ Cursor_X+1
```


bit modes on the 65816

SNES - ASM PSET and POINT for Pixel Plotting

ARM Assembly Lesson H3

Lesson P65 - Mouse reading on the Sam Coupe

Mouse Reading in MS-DOS

Risc-V Assembly Lesson 3 - Bit ops and more maths!

Mouse reading on the MSX

Hello World on RISC-OS

Atari 800 / 5200 - ASM PSET and POINT for Pixel Plotting

Apple 2 - ASM PSET and POINT for Pixel Plotting

Making a 6502 ASM Tron game... Photon1 - Introduction and Data Structures

Gaming + more:

Emily The Strange (DS) - Live full playthrough

$150 calculator: Unboxing the Ti-84 Plus CE (eZ80 cpu)

When we write to the 'Suzy' graphics chip , we MUST write low bytes first.

We're going to set the address in RAM to show as the screen with $FD94/5 - we're setting this to $C000

```
;ScreenInit -    SUZY chip needs low byte setting first
                    ;OR IT WILL WIPE THE HIGH BYTE!

    ;Set screen ram pointer to $C000
    stz $FD94          ;DISPADR    Display Address L (Visible)
    lda #$C0
    sta $FD95          ;DISPADR    Display Address H (Visible)
```

We need to set up some colors!
We'll set the background (Color 0) to blue... we'll Color 15 to Yellow (used by our font)

The palette is defined by addresses $FDA0+ - each color definition uses two bytes

```
    ;Do the palette
    ;lda #%00000000 ;Palette Color 0 ----GGGG
    stz $FDA0
    lda #%01110000  ;Palette Color 0 BBBBRRRR
    sta $FDB0

    lda #%00001111  ;Palette Color 15 ----GGGG
    sta $FDAF
    ;lda #%00001111 ;Palette Color 15 BBBBRRRR
    sta $FDBF
```

We're now ready to start our program!

Drawing a character to the screen

We're going to use a bitmap font to print characters to the screen... each character in our font is 8 bytes of black and white pixels... we'll need to convert this 1bpp to 4bpp font

```
Bitmapfont:           ;Chibiakumas bitmap font (1bpp)
    incbin "\ResALL\Font96.FNT"
```

We need a "PrintCharacter" routine...

Our font starts from Character 32, so we need to subtract 32 from the character we want to print...

Next we need to calculate the address in the font of the character..
As each character has 8 bytes of data, we multiply the Character number by 8... we do this with 3 bitshifts, then we add the address of our font...

z_HL now contains the address of the bitmap data of the character we want

```
PrintChar:
    sec
    sbc #32               ;No Characters below 32 in our font

    phx
    phy
    ldx z_h               ;Back up registers and Zeropage
    phx
    ldx z_l
    phx
    ;Calculate font pos = BitmapFont + ( (Char-32) *8)

        stz z_H           ;%00000000 00000001
        asl
        rol z_H           ;%00000000 00000010
        asl
        rol z_H           ;%00000000 00000100
        asl
        rol z_H           ;%00000000 00001000
        clc
        adc #<BitmapFont
        sta z_l

        lda z_h
        adc #>BitmapFont
        sta z_h
```

We now need to calculate the address of the screen position for the character we want to draw - our screen is 80 bytes wide ($50) and our characters are 8 lines tall, so our formula is:
Address=$C000 + ($280*CursorY) + CursorX

We don't have a multiply command in 65c02, so we achieve this by two bit shifting operations.
We then add the Xpos and the screen base ($C0)

```
	stz z_e

	lda Cursor_Y ;Ypos*$280 (Ypos * $00000010 10000000)
	lsr
	ror z_e
	sta z_d

	lda Cursor_Y
	asl
	adc z_d
	sta z_d

	lda Cursor_X      ;Add Xpos
	asl
	asl
	clc
	adc z_e
	sta z_e

	lda z_d
	adc #$C0          ;ScreenBase=$C000
	sta z_d
```

We're going to read in a line from our font...

In our font, each bit is a pixel... but in screen ram, each pixel is represented by a nibble of the byte...

We want our font to use color 15, to achieve this we shift two bits out of the font, and copy these bits to fill all 4 bits of the nibble.

We repeat this 4 times, to fill all 8 pixels of the font,

we then repeat for all 8 lines

```
		ldy #0
nextFontLine:
		phy
			lda (z_HL),y     ;Get Byte from font $76543210
			ldy #00
			sta z_AS
MoreFontLine:
			lda #0
			rol z_As     ;Shift a Bit out $6543210- 7
			rol              ;$-------7
			asl              ;$------7-
			asl              ;$-----7--
			asl              ;$----7---
			rol z_As     ;Shift a Bit out $543210-- 6
			rol              ;$---7---6
			sta z_ixl
			asl              ;$--7---6-
			ora z_ixl        ;$--77--66
			asl              ;$-77--66-
			ora z_ixl        ;$-777-666
			asl              ;$777-666-
			ora z_ixl        ;$77776666
			sta (z_DE),Y ;Write byte to screen
			iny
			cpy #4
			bne MoreFontLine
			clc
			lda #$50         ;Move Down 1 Line ($50 Bytes)
			adc z_e
			sta z_e
			lda #0
			adc z_d
			sta z_d
SkipFontLine:
		ply
		iny
		cpy #8           ;Next Y line of character
		bne nextFontLine
```

Printing a string to screen

We're going to use our PrintChar function in a printstring routine, this will print characters until it reaches a CHR 255

```
PrintStr:
    ldy #0              ;Set Y to zero
PrintStr_again:
    lda (z_hl),y            ;Load a character from addr in $20+Y

    cmp #255            ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar       ;Print Character
    iny                 ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

NewLine:
        stz Cursor_X    ;Zero X
        inc Cursor_Y    ;Increase Y
    rts
```

We can load z_HL with the address of a string to show it to the screen

```
    ;Load in the address of the Message into the zero page
    lda #>HelloWorld
    sta z_h             ;H Byte
    lda #<HelloWorld
    sta z_l             ;L Byte

    jsr PrintStr        ;Show to the screen

    jsr NewLine         ;Start a new line
    jsr PrintStr        ;Show to the screen

    jmp *                   ;Infinite Loop

HelloWorld:
    db "Hello World",255
```

The Hello World will be shown to the screen.

Building an unencrypted cartridge

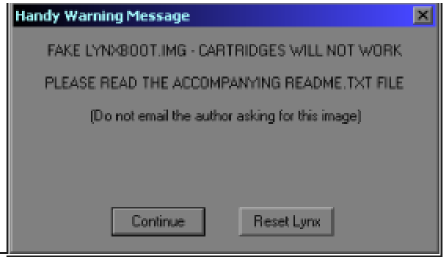
I build my files with VASM via a batch file.

```
vasm6502_oldstyle_win32.exe %1 -c02 -chklabels -nocase -Dvasm=1  -L \BldLNX\Listing.txt -DBuildLNX=1 -Fbin -o "\RelLNX\cart.o"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** file
We need to specify the **Destination file name**
We include **some symbols**
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)
The Lynx has a 65c02 CPU - to enable the extra features we use the **-c02** switch

We can load the Handy emulator with this file just fine, though handy will give a warning because we don't have a rom image (we can just ignore it!)

```
handy.exe "\RelLNX\cart.o"
```

Real Atari Lynx cartridges need to be encrypted, but we would need a proper OS ROM to run them (Which cannot legally be distributed!)

For our purposes these unencrypted .O files will work just fine!

Debugging Tools

We can include the 'Monitor tools' we looked at in the multiplatform series, these can help us develop on a new system.	`;Basic macros for ASM tasks` `include "\SrcAll\BasicMacros.asm"` `;Debugging tools` `include "\SrcAll\monitor.asm"` `;Basic commands for ASM tasks` `include "\SrcAll\BasicFunctions.asm"`
We can use the Monitor to show the contents of the CPU registers, We can use the Memdump to output lines of ram to the screen.	`jsr monitor      ;Show registers to screen` `jsr MemDump      ;Dump address to screen` `    word $3000       ;Address` `    byte $3       ;Lines`
The register contents, and the memory address specified will be shown with these commands.	

Lesson H7 - Hello World on the Nes / Famicom

Like most Tile based systems, the Famicom doesn't have a font built in, we'll have to create our own font and character printing routines to show text to screen

Lets learn how!

NES_HelloWorld.asm


Starting a Nes or Famicom Cartridge

We need a header for our cartridge - the settings shown will work for a simple program.

```
org $BFF0

db "NES",$1a            ;ID
db $01                  ;Rom pages (16k each)
db $0                   ;CHR-ROM pages
db %01000010            ;mmmmFTBM        mmmm = mapper no bottom 4 bits ,
db %00000000            ;mmmm--PV        mapper (top 4 bits...  Pc10 arcade,
db 0                    ;Ram pages
db 0,0,0,0,0,0,0
                        ;We selected Mapper 4 - it has 8k VRAM , 8K Sram and 128k rom
```

We also need a footer... this has definitions pointing to the start of the program and interrupt handlers

```
;Cartridge Footer
    org $FFFA
    dw nmihandler            ;FFFA - Interrupt handler
    dw ProgramStart          ;FFFC - Entry point
    dw irqhandler            ;FFFE - IRQ Handler
```

We're going to need a few bytes in the zero page to store data, we also need an IRQ handler of some kind (a return in this case)

Vblank (The point when the screen is not being drawn) is important, this is the only time we can write to VRAM... so we can detect when this is possible we use zero page entry $7F as a marker.. and alter this when vblank occurs

```
Cursor_X     equ $40      ;Position of next printed character
Cursor_Y     equ Cursor_X+1

vblanked     equ $7F      ;Zero page address of Vblank count

nmihandler:      ;This procuedure runs after each frame (See footer.asm)
    php
    inc vblanked          ;Alter Vblank Zero page entry
    plp
irqhandler: ;Do nothing
    rti
```

We're ready to start our program!... First we need to set up the font...

We need to define the tiles that will make up each character... These are written to the 'Pattern Table' at VRAM address $0000

Each tile uses 2 bitplanes (4 color)... The 8 line of Bitplane 0 of the tile come first... then the 8 lines of Bitplane 1

to convert our black and white font to 4 colors we need to write the same 8 bytes of data to both bitplanes

This will set our font to color 3 in the palette

Note... at this stage the screen is not on, so we don't need to worry about VBlank at this time

```
    lda #$00        ;Reset Cursor pos
    sta Cursor_X
    sta Cursor_Y
    tay

    ;Pattern table &0000
    sta $2006       ;PPUADDR H
    sta $2006       ;PPUADDR L

    lda #BitmapFont&255 ;Address of font
    sta $20
    lda #BitmapFont/256
    sta $21

    ldx #3  ;Y=0 (768 lines total)
fontchar_loop:
    txa
    pha
fontchar_loop2:
        tya
        pha
            jsr Font_DoBitplane ;Bitplane 0
        pla
        tay
        jsr Font_DoBitplane     ;Bitplane 1

        tya
        bne fontchar_loop2      ;Repeat until Y=0
    inc $21
    pla
    tax
    dex
    bne fontchar_loop


Font_DoBitplane:
    ldx #8          ;8 bytes per tile bitplane
FontFillAgain_Plane1:
    lda ($20),y
    sta $2007       ;Write data to data-port
    iny
    dex
    bne FontFillAgain_Plane1
    rts

Bitmapfont:         ;Chibiakumas bitmap font (1bpp B/W)
    incbin "\ResALL\Font96.FNT"
```

Now we've got a font, we need to set up our palette...
The palette is in VRAM Addresses $3F00 onwards... each palette of 4 colors uses 4 bytes.

We load in the four bytes of the palette in from our palette definition... we use X as an offset in the palette, so the colors are read in backwards

We define the background as Blue, and Color 3 as Yellow

```
    lda #$3F        ;Select Palette ram &3F00
    sta $2006       ;PPUADDR H
    txa ;X=0
    sta $2006       ;PPUADDR L

    ldx #4
PaletteAgain
    lda Palette-1,x
    sta $2007       ;PPUDATA
    dex
    bne PaletteAgain

Palette:
;   Color   3   2   1   0
        db $38,$21,$15,$02
```

We're finally done! we need to turn on the layers, and enable the Vblank

```
;Turn ON the screen
    lda #%00011110      ;(Sprite enable/back enable
    sta $2001           ;PPUMASK

    lda #$80            ;NMI enable (Vblank)
    sta $2000           ;PPUCTRL - VPHB SINN
```

Waiting for Vblank

Now the screen is on, we need to wait for Vblank before we write to VRAM...

To do this we write 0 to zeropage entry 'Vblanked' (defined by a symbol)... then we wait for it to change... when vblank occurs, the value will be nonzero

```
waitframe:
    pha
        lda #$00
        sta vblanked        ;Zero Vblanked
waitloop:
        lda vblanked        ;Wait for the interrupt to change it
        beq waitloop
    pla
    rts
```

Getting A Character to the screen

We're going to print a character to the screen... Because our font has no characters below 32 we need to subtract 32 from the character number.

```
PrintChar:
    sec
    sbc #32             ;No character below 32 in our font
    sta $26

    txa
    pha
    tya
    pha
```

We need to calculate the VRAM address of the next tile we want to change...
The Tilemap starts at VRAM address $2000, and the tilemap is 32 tiles wide, and each tile is 1 byte in memory, so our formula is:

Address= $2000 + (Ypos*32) + Xpos

We need to multiply the CursorY by 32... we do this by repeated bitshifts.

```
    lda Cursor_Y
    asl                 ;%00000111
    asl
    asl
    asl
    asl                 ;%11100000
    ora Cursor_X
    tay                 ;L Byte

    lda Cursor_Y
    lsr                 ;%11111000
    lsr
    lsr                 ;%00011111
    clc
    adc #$20            ;Tilemap Base (Nametable) = $2000
    tax                 ;Hbyte
```

Now that the screen is on, Before we write to vram, we need to wait until vblank with our 'WaitFrame' function.

Then we select our calculated Vram Address with port $2006 - finally, we write the actual tile number (Character) to Vram with port $2007

Any write to VRAM will mess up the scrolling of the tilemap... so we need to reset it with port $2005

```
    jsr waitframe       ;Can only Write to VRAM in Vblank

    stx $2006           ;PPUADDR High byte
    sty $2006           ;PPUADDR Low byte

    lda $26             ;Write Tile Number to VRAM
    sta $2007           ;PPUDATA

    ;Need to reset scroll each write
    lda #0              ;Scroll X
    sta $2005           ;PPUSCROLL
    lda #0-8            ;Scroll y
    sta $2005           ;PPUSCROLL
```

The Screen is 32 tiles wide, once we're at the end of the screen, we need to do a newline to start the next line.

```
    inc Cursor_X        ;Move across screen
    lda Cursor_X
    cmp #32             ;At end of line?
    bne PrintChar_NotNextLine
    jsr NewLine
PrintChar_NotNextLine:
    pla
    tay
    pla
    tax
    rts
```

Printing a string to screen

We're going to extend our PrintChar command to print strings... Our strings will be char 255 terminated.

We also need a NewLine command, this needs to Zero the Cursor_X and increase Cursor_Y

```
PrintStr:
    ldy #0              ;Set Y to zero
PrintStr_again:
    lda ($20),y         ;Load a character from addr in $20+Y

    cmp #255            ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar       ;Print Character
    iny                 ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

NewLine:
    lda #0
    sta Cursor_X
    inc Cursor_Y
    rts
```

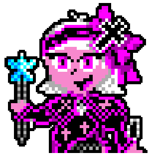
We need to load the address of the string into Zeropage entries $20/1... we then call our Printstring routine to show it to the screen

Once we're done, we use a JMP * to halt the processor.

```
;Load in the address of the Message into the zero page
    lda #>HelloWorld
    sta $21             ;H Byte
    lda #<HelloWorld
    sta $20             ;L Byte

    jsr PrintStr        ;Show to the screen

    jsr NewLine         ;Start a new line

    jmp *

HelloWorld:             ;255 terminated string
    db "Hello World",255
```

Our Hello World message will be shown to screen

Because we're waiting for VBlank each write, the text will be rather slow,
It's enough for this beginner series, but we need a buffer for real games... we covered this in the Platform Specific series



Building our NES cartridge

I build my files with VASM via a batch file.

```
vasm6502_oldstyle_win32.exe %1 -chklabels -nocase -Dvasm=1 -L \BldNES\Listing.txt -DBuildNES=1 -Fbin -o "\RelNES\cart.nes"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** file
We need to specify the **Destination file name**
We include **some symbols**
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)

Once we've compiled our cartridge, we can load it with our emulator.	`nestopia.exe \RelNES\cart.nes`

Debugging Tools

We can include the 'Monitor tools' we looked at in the multiplatform series, these can help us develop on a new system.

```
    include "\SrcALL\BasicMacros.asm"

z_Regs      equ $60
Cursor_X    equ $40
Cursor_Y    equ Cursor_X+1
SpPage      equ $0100
UserRam     equ $200
vblanked    equ $7F     ;Zero page address of Vblank count


    ;Debugging tools
    include "\SrcAll\monitor.asm"
    ;Basic commands for ASM tasks
    include "\SrcAll\BasicFunctions.asm"
```

We can use the Monitor to show the contents of the CPU registers,

We can use the Memdump to output lines of ram to the screen.

```
    jsr monitor         ;Show registers to screen

    jsr MemDump         ;Dump address to screen
        word $3000          ;Address
        byte $3         ;Lines
```

The register contents, and the memory address specified will be shown with these commands.

Lesson H8 - Hello World on the SNES / Super Famicom

Conceptually Hello World on the SNES is pretty similar to the NES , but of course the code is rather different... lets learn how to get Hello World on the snes!

SNS_HelloWorld.asm

Starting a SNES/SFC Cartridge

Our cartridge needs to start at address $8000, When our program starts, we'll disable interrupts

We also need to define some symbols - we'll need two zeropage bytes for the current cursor position	<pre>Cursor_X equ $40 Cursor_Y equ Cursor_X+1 org $8000 ;Start of ROM SEI ;Stop interrupts</pre>
Our cartridge also needs a footer - the one here will work for our purposes.	<pre> org $FFC0 ; "123456789012345678901" db "www.ChibiAkumas.com " ; Program title (21 byte Ascii string) db $20 ;Rom mode/speed (bits 7-4 = speed, bits 3-0 = map mode) db $00 ;Rom type (bits 7-4 = co-processor, bits 3-0 = type) db $01 ;Rom size in banks (1bank=32k) db $00 ;Ram size (0=none) db $00 ;Country/video refresh (ntsc 60hz, pal 50hz) (0=j 1=us/eu) db $00 ;Developer id code db $00 ;Rom version number db "cc" ;Complement check db "cs" ;Checksum ;65816 mode vectors dw $0000 ;Reserved dw $0000 ;Reserved dw $0000 ;Cop vector (cop opcode) dw $0000 ;Brk vector (brk opcode) dw $0000 ;Abort vector (unused) dw $0000 ;Vblank interrupt handler dw $0000 ;Reset vector (unused) dw $0000 ;Irq vector (h/v-timer/external interrupt) ;6502 mode vectors dw $0000 ;Reserved dw $0000 ;Reserved dw $0000 ;Cop vector (cop opcode) dw $0000 ;Brk vector (unused) dw $0000 ;Abort vector (unused) dw $0000 ;Vblank interrupt handler dw $8000 ;Reset vector (cpu is always in 6502 mode on reset) dw $0000 ;Irq/brk vector</pre>
We're going to need to set up our screen... First we need to initialize the tilemap, we need to set the base address in VRAM ($0000) and the tilemap size (32x32) We can only write to VRAM during Vblank while the screen is on, so we turn the screen off during our initialization	<pre> ;aaaabbbb -aaa=base addr for BG2 bbb=base addr for BG1 lda #%00010001 sta $210B ;BG1 & BG2 VRAM location register [BG12NBA] ; xxxxxxss - xxx=address… ss=SC size 00=32x32 stz $2107 ;BG1SC - BG1 Tilemap VRAM location ; abcdefff - abcd=tile sizes e=pri fff=mode def lda #%00001001 sta $2105 ;BGMODE - Screen mode register ; x000bbbb - x=screen disable (1=disable) lda #%10000000 ;Screen off sta $2100 ;INIDISP - Screen display register</pre>
We need to set the palette next... we only need two colors for this test... the background (Color 0 - blue) and the font (color 15 - yellow) We select a color by writing to $2121, and RGB bytes to $2122	

```
;Background (Color 0)
    stz $2121       ;CGADD - Colour selection  (0=Back)
                ;gggrrrrr
    stz $2122       ;CGDATA - Colour data register
                ;?bbbbbgg
    lda #%00111100
    sta $2122       ;CGDATA
;Font (Color 15)
    lda #15         ;Color 15=Font
    sta $2121       ;CGADD - Colour selection  (15=Font)
                ;gggrrrrr
    lda #%11111111
    sta $2122       ;CGDATA - Colour data register
                ;?bbbbbgg
    lda #%00000011
    sta $2122       ;CGDATA
```

We need to configure the $2118/9 ports ... we write zero to $2115... we're setting the Vram address to AutoInc on a write to $2118

```
;TileDefs
    ;     i000abcd - I 0=inc on $2118 or $2139 1=$2119 or $213A.. abcd=move size
    stz $2115       ;VMAIN - Video port control (Inc on write to $2118)
```

We now need to load in our font... The SNES font uses 4 bitplanes for each 8 pixel wide line of the tile... The 8 lines of bitplanes 0+1 come first... then the 8 lines of lines 2+3 come next.

We set the address to write to with ports $2116/7 - The Tile patterns are at address $1000

we read in from our 1bpp font, and write each byte 4 times... because bitplanes (0,1) and (2,3) are split, we write the 8 lines once... reset Y and do the same again!

```
    stz $2116       ;VRAM MemL
    lda #$10
    sta $2117       ;VRAM MemH

    lda #BitmapFont&255
    sta $20
    lda #BitmapFont/256
    sta $21

    ldx #3          ;96 sprites * 8 lines = 768
    ldy #0
fontchar_loopx:
    phx
fontchar_loop:
        phy
            jsr Font_DoBitplane ;Bitplane 0+1
        ply
        jsr Font_DoBitplane     ;Bitplane 2+3
        tya
        bne fontchar_loop
        inc $21                 ;Inc to byte of address
    plx
    dex
    bne fontchar_loopx

Font_DoBitplane:
        ldx #8              ;8 Lines
fontchar_loopL:
        lda ($20),y
        sta $2119           ;Write Word data to data-port
        sta $2118
        iny
        dex
        bne fontchar_loopL
        rts

Bitmapfont:                 ;Chibiakumas bitmap font
    incbin "\ResALL\Font96.FNT"
```

Right! Our font is ready,

but we now need to initialize the Tilemap... we need to reset the scroll position with $210D/E

We also need to clear the tilemap... we do this by writing zeros to all the tiles in the tilemap

The Tilemap starts at $0000 - and there are 1024 pairs of bytes to zero (32x32 tiles)

```
;Set Scroll position
    stz $210D        ;BG1HOFS BG1 horizontal scroll
    stz $210D        ;BG1HOFS

    lda #-1
    sta $210E        ;BG1VOFS BG1 vertical scroll
    stz $210E        ;BG1VOFS

;Clear Screen
    stz $2116        ;MemL -Video port address [VMADDL/VMADDH]
    stz $2117        ;MemH

    ldy #4           ;Tilemap Size: 32*32 = 1024
    ldx #0
ClearTilemap:
    stz $2119        ;Zero all Tiles in Tilemap
    stz $2118
    dex
    bne ClearTilemap
    dey
    bne ClearTilemap
```

We're finally done... We now just need to actually turn on the screen!

Phew! that was hard work!

```
;Turn on the screen
        ; ---S4321 - S=sprites 4-1=enable Bgx
    lda #%00000001   ;Turn on BG1
    sta $212C        ;Main screen designation [TM]

    ;      x000bbbb - x=screen disable (1=disable) bbbb=brightness (15=max)
    lda #%00001111   ;Screen on
    sta $2100        ;INIDISP - Screen display register
```

We may be able to skip the clear screen part on some emulators, but emulators like Mesen-S will fill the ram with random data on power-up to force us to do things properly!... how cheeky!

Waiting for Vblank

Now that the screen is enabled, we need to wait for Vblank before writing to the screen...

Vblank is the time during redraw when the screen has finished drawing, and the next frame hasn't started.

we can check if the screen is in Vblank by reading $4212

```
WaitVblank:
    lda $4212        ;HVBJOY - Status
        ; xy00000a   - x=vblank state y=hblank state a=joypad ready
    and #%10000000
    beq WaitVblank   ;Wait until we get nonzero - this means we're in VBLANK
    rts
```

Getting A Character to the screen

We want to print the character in A to the screen...

Our font doesn't have a character below 32... so we subtract 32 from the character number.

We now need to calculate the address of the tile we want to change... our tilemap is 32 tiles wide and starts from memory address $0000 , so our formula is:
;Address= (Ypos*32) + Xpos

We achieve the multiplication by bitshifting.

```
PrintChar:
    sec
    sbc #32                 ;No Charactrers below 32 in our font
    phx
    phy
        pha
            ldx $21             ;Backup $21
;Address= (Ypos*32) + Xpos
            lda Cursor_Y
            sta $21             ;%YYYYYYYY 00000000
            lda #0
            lsr $21             ;%0YYYYYYY Y00000000
            ror
            lsr $21             ;%00YYYYYY YY00000000
            ror
            lsr $21             ;%000YYYYY YYY0000000
            ror
            adc Cursor_X
```

We need to wait for Vblank before doing any VRAM writing... we use the function we wrote before.

Now we select the address we want to write to using ports $2116/7... Then we write the two bytes for that address with ports $2119/8

note... we must write them in this order - as the address will autoinc when we write to #2118)

```
        jsr WaitVblank  ;Can only write to vram during Vblank
        sta $2116       ;MemL -Video port address [VMADDL/VMADDH]
        lda $21
        sta $2117       ;MemH
    pla
    sta $2119       ;Top Tile Byte (0)
    sta $2118       ;Bottom Tile Byte (TileNum)
```

We now need to increase our X position, then we check if we're at the end of the line... we need to move down a line if we've reached character 32 (the right hand of the screen)

```
        inc Cursor_X
        lda Cursor_X
        cmp #32             ;Tilemap is 32 tiles wide
        bne PrintChar_NotNextLine
        jsr NewLine
PrintChar_NotNextLine:
        stx $21             ;Resotre $21
    ply
    plx
    rts
```

Printing a string to screen

We're going to extend our PrintChar command to print strings... Our strings will be char 255 terminated.

We also need a NewLine command, this needs to Zero the Cursor_X and increase Cursor_Y

```
PrintStr:
    ldy #0              ;Set Y to zero
PrintStr_again:
    lda ($20),y         ;Load a character from addr in $20+Y

    cmp #255            ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar       ;Print Character
    iny                 ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

NewLine:
    stz Cursor_X        ;Clear Xpos
    inc Cursor_Y        ;Increase Ypos
    rts
```

We need to load the address of the string into Zeropage entries $20/1... we then call our Printstring routine to show it to the screen

Once we're done, we use a JMP * to halt the processor.

```
    lda #>HelloWorld
    sta $21             ;H Byte
    lda #<HelloWorld
    sta $20             ;L Byte

    jsr PrintStr        ;Show to the screen

    jsr NewLine         ;Start a new line

    jmp *               ;Infinite Loop
```

Our Hello World message will be shown to screen

Building a SNES / Super Famicom Cartridge

I build my files with VASM via a batch file.

```
\vasm6502_oldstyle_win32.exe %BuildFile% -c02 -chklabels -nocase -Dvasm=1 -L \BldSNS\Listing.txt -DBuildSNS=1 -Fbin -o "\RelSNS\cart.sfc"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** file
We need to specify the **Destination file name**
We include **some symbols**
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)
The SNES has a 65c02 CPU - to enable the extra features we use the **-c02** switch

Once we've compiled our cartridge, we can load it with our emulator.

```
\Emu\Snes9x\snes9x.exe \RelSNS\cart.sfc

\Emu\mesen-s\Mesen-S.exe \RelSNS\cart.sfc
```

Debugging Tools

We can include the 'Monitor tools' we looked at in the multiplatform series, these can help us develop on a new system.

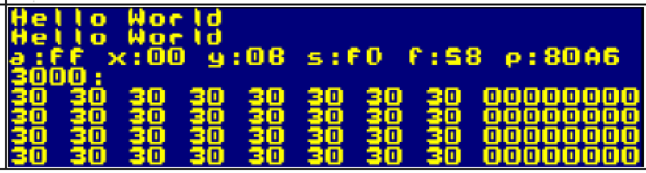
```
z_Regs      equ $20
Cursor_X    equ $40
Cursor_Y    equ Cursor_X+1

sppage equ $0100
UserRam equ $0200

    include "\SrcAll\BasicMacros.asm" ;Basic macros for ASM tasks

    ;Debugging tools
    include "\SrcAll\monitor.asm"
    ;Basic commands for ASM tasks
    include "\SrcAll\BasicFunctions.asm"
```

We can use the Monitor to show the contents of the CPU registers,

We can use the Memdump to output lines of ram to the screen.

```
  jsr monitor        ;Show registers to screen

  jsr MemDump        ;Dump address to screen
      word $3000         ;Address
      byte $3        ;Lines
```

The register contents, and the memory address specified will be shown with these commands.



Lesson H9 - Hello World on the PC Engine/TurboGrafx-16 Card

The PC Engine uses a tilemap for its background graphics... to show Hello World we'll need to define our font as tiles, then use those tiles to show the letters of our message!

PCE_HelloWorld.asm

Starting a PC Engine/TurboGrafx-16 Card

Our program will start at $E000

We'll also define some symbols we'll need for our cursor position

```
Cursor_X     equ $40      ;Used for Printchar
Cursor_Y     equ $41

    org $e000        ;bank $0
ProgramStart:
```

We also need a footer, it's just a word pointing to the start of our program

```
  org $fffe
  dw ProgramStart          ;Reset Vector
```

When our program starts, we need to set a lot of things up!

First we turn off interrupts, set highspeed mode, and clear the decimal flag.

Next we need to 'Page in the RAM and IO banks - this configures parts of the addressable memory, pointing them to underlying hardware.... we do this with a special 6280 command called TAM

We also set up the stack pointer... finally we turn the interrupots off with port $1402

```
ProgramStart:
    sei                 ;Disable interrupts
    csh                 ;Highspeed Mode
    cld                 ;Clear Decimal mode

    lda #$f8            ;map in RAM
    tam #%00000010      ;TAM1 (2000-3FFF)

    lda #$ff            ;map in I/O (#$ff)
    tam #%00000001      ;TAM0 (0000-1FFF)
    tax
    txs                 ;Init stack pointer

        ;      T12 - TIQ, IRQ1, IRQ2
    lda #%00000111
    sta $1402           ;IRQ mask... 1=Off
```

We need to set up the Tilemap... we need to select the video registers with the special command ST0... then set values for those registers with ST1 and ST2

First we turn the tilemap on... next we set the tilemap size - we set it to 32x32, Finally we reset the position - so that the first tile in the tilemap is the top left corner of the screen.

```
;   ScreenInit
    st0 #5                  ;RegSelect 5
          ;BSXXIIII  Backgroundon Spriteon eXtendedsync Interruptenable
    st1 #%10000000          ;Background ON, Sprites On
    st2 #0

    st0 #9
        ; 0BBB0000
    st1 #%00000000          ;BACKGROUND Tilemap size (32x32)
    st2 #0

;Reset Background scroll registers
    st0 #7                  ;Background X-scroll (------XX XXXXXXXX)
    st1 #0
    st2 #0

    st0 #8                  ;Background Y-scroll (-------Y YYYYYYYY)
    st1 #248                ;Move Byte pos 0 to top left of screen
    st2 #0
```

Next we're going to set up the palette... we select a color to change with registers $0402/3 and set the new RGB value with registers $0404/5

The background is Color 0 - we set it to blue... the foreground is Color 15 - we set it to yellow

	`` ;Background Color stz $0402 ;Palette address L stz $0403 ;Palette address H lda #%00000111 ;GGRRRBBB sta $0404 stz $0405 ;-------G ;Font color lda #15 sta $0402 ;Palette address L stz $0403 ;Palette address H lda #%11111000 sta $0404 ;GGRRRBBB lda #%00000001 sta $0405 ;-------G ```
We need to copy our font into tile ram... We're going to use tiles 256+ - which appears at Vram Address $1000 onwards... we need to set ST0 to #0 to tell the hardware we want to change the address - then write $1000 to ST1/2... finally we set ST0 to #2 to select that we're going to send data. Our font is 1bpp, but the PC engine uses 4 bitplanes - split into two halves - we need to send the 8 lines bitplanes 0/1 first, then the 8 lines of Bitplanes 1/2 As our font is 1bpp - we send the same data for all 4. When we want to send data in A - we use $0102 and $0103 - these are the equivalent of ST 1/2 when our value is in the accumulator	``` st0 #0 ;set Address reg to $1000 st1 #$00 ;we'll put our font there (tiles 256+) st2 #$10 st0 #2 ;Select Data reg lda #>Bitmapfont ;Address of our font sta $61 lda #<Bitmapfont sta $60 ldx #3 ;96*8=256*3 ldy #0 FontNextChar: phx phy jsr FontPart ;Do Bitplanes 0/1 ply jsr FontPart ;Do Bitplanes 2/3 plx cpy #0 BNE FontNextChar inc $61 dex bne FontNextChar FontPart: ldx #8 FontPartAgain: lda ($60),Y ;sta_00 $02 ;I use my macro here - I need to write to VramDataWrite at $0002 sta $0102 ;This does not work, as the CPU redirects it to $2002 sta $0103 ;just set second plane to 0 iny dex BNE FontPartAgain ;Write the first 8 lines rts Bitmapfont: incbin "\ResALL\Font96.FNT" ```
We now need to clear our tilemap, and set all the starting tiles to zero... We select the address for the destination of our tilemap with Graphics Reg 0 - we select address $0000 We need to write 1024 tiles to fill our 32x32 tilemap - we need to write	

```
    st0 #0          ;VDP reg 0 (address)
    st1 #$00        ;L - Start of tilemap $0000
    st2 #$00        ;H

    st0 #2          ;Select VDP Reg2 (data)

    ldx #4
    ldy #0          ;1024 tiles total (32x32)
ClsAgain:
    st1 #0          ;Fill the entire area with our "Space tile"
    st2 #%00000001      ;(tile 256)
    dey
    bne ClsAgain
    dex
    bne ClsAgain
```

The PC Engine has lots of special commands... most important for us here are ST0 , ST1 and ST2... these save fixed values to the graphics hardware, and are equivalent to STA $0100, STA $0102 and STA $0103

ST0 Selects a register, and ST1/2 save values to that register... Register 0 is the 'address select' register... Register 2 is Data write - sending data to the address selected with Register 0
It may sound confusing, but don't worry too much if you don't understand it yet - just copy the code here for now.

Getting A Character to the screen

We're going to print a character to the screen!
We need to work out the next cursor position... as the tilemap is at VRAM address $0000 and each line is 32 tiles wide, our formula is:

Address=(Ypos *32) + X

We multiply Y by 32 by bishifts, and select the calculated address...

We need to subtract 32 from our character number, as our font has no characters below 32 then write the tilenumber.

```
PrintChar:
    pha
        st0 #0          ;Reg0=Select Addr
;Address=(Ypos *32)+X
        lda Cursor_Y
        asl             ;%00000111
        asl
        asl
        asl
        asl             ;%11100000
        ora Cursor_X
        sta $0102       ;Address L

        lda Cursor_Y
        lsr             ;%11111000
        lsr
        lsr             ;%00011111
        sta $0103       ;Address H
    pla
    pha
        st0 #2          ;Reg2=Write Byte Data
        sec
        sbc #32     ;We have no characters below 32

        sta $0102       ;Store Char
        st2 #%00000001  ;Font Tile are 256+
```

Once we've drawn our letter, we increase Cursor X, and check if we've got to the end of a line - if we have, we use our NewLine function to start the next line

```
        inc Cursor_X
        lda Cursor_X
        cmp #32         ;Are we at end of line
        bne PrintChar_NotNextLine
        jsr NewLine
PrintChar_NotNextLine:
    pla
    rts
```

Printing a string to screen

We're going to extend our PrintChar command to print strings... Our strings will be char 255 terminated.

We also need a NewLine command, this needs to Zero the Cursor_X and increase Cursor_Y

```
PrintStr:
    ldy #0                  ;Set Y to zero
PrintStr_again:
    lda ($60),y             ;Load a character from addr in $60+Y

    cmp #255                ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar           ;Print Character
    iny                     ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

NewLine:
    stz Cursor_X            ;Clear Xpos
    inc Cursor_Y            ;Increase Ypos
    rts
```

We need to load the address of the string into Zeropage entries $20/1... we then call our Printstring routine to show it to the screen

Once we're done, we use a JMP * to halt the processor.

```
;Load in the address of the Message into the zero page
    lda #>HelloWorld
    sta $61                 ;H Byte
    lda #<HelloWorld
    sta $60                 ;L Byte

    jsr PrintStr            ;Show to the screen

    jsr NewLine             ;Start a new line

    jmp *                   ;Infinite Loop

HelloWorld:
    db "Hello World",255
```

Our Hello World message will be shown to screen

Building a PC Engine/TurboGrafx-16 Card

I build my files with VASM via a batch file.

```
vasm6502_oldstyle_win32.exe %1 -6280 -chklabels -nocase -Dvasm=1 -L \BldPCE\Listing.txt -DBuildPCE=1 -Fbin -o "\RelPCE\cart.PCE"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** file
We need to specify the **Destination file name**
We include **some symbols**
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)
We need to be able to use the 6280 exclusive opcodes - to enable the extra features we use the **-6280** switch

Once we've compiled our cartridge, we can load it with our emulator.

```
\Emu\Ootake\Ootake.exe "\RelPCE\cart.PCE"
```

Debugging Tools

We can include the 'Monitor tools' we looked at in the multiplatform series, these can help us develop on a new system.

```
    include "\SrcAll\BasicMacros.asm"  ;Basic macros for ASM tasks

z_Regs      equ $60     ;Fake Registers
SPpage      equ $2100   ;StackPointer is at an odd address on the PCE!
Cursor_X    equ $40     ;Used for Printchar
Cursor_Y    equ $41
```

```
;Debugging tools
include "\SrcAll\monitor.asm"
;Basic commands for ASM tasks
include "\SrcAll\BasicFunctions.asm"
```

We can use the Monitor to show the contents of the CPU registers,

We can use the Memdump to output lines of ram to the screen.

```
jsr monitor         ;Show registers to screen

jsr MemDump         ;Dump address to screen
    word $3000          ;Address
    byte $3         ;Lines
```

The register contents, and the memory address specified will be shown with these commands.

Lesson H10 - Hello World on the Commodore PET

The PET is the predecessor of the VIC-20, it's only capable of 'text graphics' and cannot allow custom characters.

Lets learn how to make the PET to say 'Hello'

PET_HelloWorld.asm

Showing Hello World to the screen

We're going to create a PRG file... these need a header to start the program - we'll never need to change this provided we don't want to change the start address,

Ours starts at $100A

```
    ;Basic program to execute our ASM binary
*=$0401
    db $0e,$04,$0a,$00,$9e,$20,$28, $31,$30,$34,$30,$29,$00,$00,$00
;org $0410
```

We're going to use the firmware function $FFD2 to print characters (known as ChrOut)

Unfortunately this function does not use normal ASCII... and it doesn't have lower case letters... we'll need to do some converting to fix this!

```
PrintChar:  ;DefaultFont
    cmp #96             ;Check if character >96
    bcc PrintCharOK
    and #%11011111      ;Convert to uppercase
PrintCharOK:
    jmp $ffd2           ;CHROUT - Output a character
```

We're going to 'extend' this to make a PrintString routine...

We'll use Zeropage pair $20/1 to store an address which will point to a string in memory...

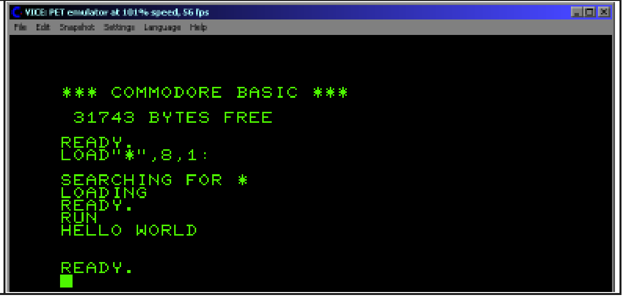
We use Y as an offset to the start address, and We'll print characters to the screen, until we get a character 255...

```
PrintStr:
    ldy #0              ;Set Y to zero
PrintStr_again:
    lda ($20),y         ;Load a character from addr in $20+Y

    cmp #255            ;If we got 255, we're done
    beq PrintStr_Done

    jsr PrintChar       ;Print Character
    iny                 ;Inc Y and repeat
    jmp PrintStr_again
PrintStr_Done:
    rts

HelloWorld:             ;255 terminated string
    db "Hello World",255
```

We need to load the High and Low bytes of our address into the $20/1 zero page entries to define the address of our string, and then call our PrintString function - this will show our string to the screen.

```
;Load in the address of the Message into the zero page
lda #>HelloWorld
sta $21              ;H Byte
lda #<HelloWorld
sta $20              ;L Byte

jsr PrintStr         ;Show to the screen

jsr NewLine          ;Start a new line

rts                  ;Return to basic
```

Our Hello World message will be shown to screen



Building a PRG file with Vasm

I use VASM to compile the ASM file into a usable PRG

```
.vasm6502_oldstyle_win32.exe %1 -cbm-prg -chklabels -nocase -L \BldPET\Listing.txt -Dvasm=1 -DBuildPET=1 -Fbin -o "\BldPET\Program.prg"
```

We have to specify a **Source ASM** file.
We need to tell VASM we want to create a **BINary** that's a **PRG** file
We need to specify the **Destination file name**
We include **some symbols** (some of my code uses these - you won't need them)
We're specifying an output **Listing file**
We're also disabling **case sensitivity**, and telling **VASM to check our labels** don't look like commands (in case we forgot a tab on one of our commands)

We can start the VICE emulator with the PRG from the command line

```
xpet.exe \BldPET\Program.prg
```

Debugging Tools

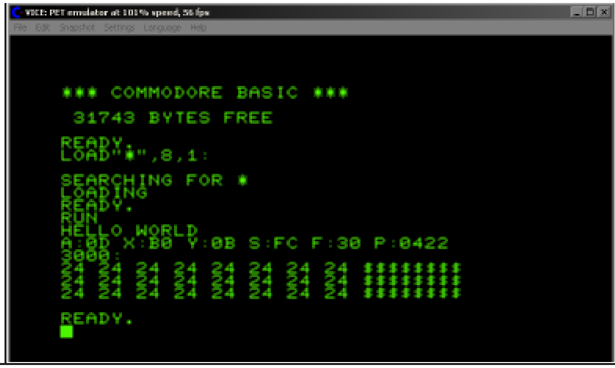
It's relatively easy to add support for my 'Monitor Tools' ... these will allow you to see the state of the processor or memory easily.

```
z_Regs      equ $20         ;Fake Registers
SPpage      equ $0100       ;Stackpointer Address

    ;Basic macros for ASM tasks
    include "\SrcAll\BasicMacros.asm"

include "\SrcAll\monitor.asm"           ;Debugging tools
include "\SrcAll\BasicFunctions.asm"    ;Basic commands for ASM tasks
```

We can use the Monitor to see the processor registers, or specify a memory address, and a number of lines to show.

```
jsr monitor          ;Show registers to screen

jsr MemDump          ;Show Some Ram to screen
word $3000               ;Address to show
byte $3                  ;Lines
```

We will see the result to screen.


VICE: PET emulator at 101% speed, 56 fps
File Edit Snapshot Settings Language Help
*** COMMODORE BASIC ***
31743 BYTES FREE
READY.
LOAD"*",8,1:
SEARCHING FOR *
LOADING
READY.
RUN
HELLO WORLD
A:0D X:B0 Y:0B S:FC F:30 P:0422
3000:
24 24 24 24 24 24 24 24 $$$$$$$$
24 24 24 24 24 24 24 24 $$$$$$$$
24 24 24 24 24 24 24 24 $$$$$$$$
READY.