Getting Stuff Done in Assembly Language

Introduction

When you start out progrmaming for the SD-8516 it's basically going to be in Assembly if you want maximum power because any language running on the system is going to pare down to assembly eventually. So unless we get a good version of C up, it's assembly. Try this first for now.

But how do you get stuff done? How do you do real stuff? Well you're in luck son, because this is the guide for you!

The Golden Rule of Assembly

If you don't have a library for something, write it yourself first. Writing low level stuff where you put characters into the memory mapped screen area is possible, but if you did that you would just end up writing your own library to do it anyways. Use the provided libraries or make them better. Look for a library function that can help you first.

Always remember: *The libraries are your friend.*

Using an external editor

If you're in the C version, you can have an editor open in .vc4/files and that makes it easier to edit files. For the WASM version you will need to INCOPY your file into the system if you want to use an external editor.

Dealing with Text

Print a string (BASIC_IO)

```
; Example 1: Print using INT 5h
.address $C000
    LDBLX @msg
    LDAH $66    ; BASIC IO_PUTSTRING
    INT 0x05    ; INT 5 is the BASIC services library.
    RET         ; return control back to system.
msg:
    .bytes "Hello Potato Chips!", 13, 10, 0
```

Let's follow along at home:

```
  MKDIR A
  CD A
  ED a.asm
  a            <--- this will enter append mode
  <paste the above program>
  .
  w
  q
  AS a.asm a
  DLOAD a
  DUMP C000     <---- you can see it's been loaded
  SYS 49152
```

The above sequence of commands will allow you to enter a program. Alternately you can use INCOPY for the WASM version or just copy the files into the directory .vc4/files on the C version.

Run the program by typing SYS 49152. What do you see?

Obviously it will say hello potato chips.

INT 10h Print String

The same as above works with INT 10h.

```
  ;
================================================================================
  ; AH=18h - Print String at Cursor
  ; Input:  ELM = pointer to null-terminated string
  ; Output: Cursor advances
  ; Notes:  Works with current video mode
  ;
================================================================================
```

```
; Example 2: Print using INT 10h

.address $C000
    LDELM @msg
    LDAH $18    ; INT 10h Write String
    INT 0x10    ; INT 10h is the Terminal Services library.
    RET         ; return control back to system.
msg:
    .bytes "Hello from INT $10!", 13, 10, 0
```

Input

You might want to ask the user for input. For string input, we repurpose BASIC's INPUT command. IO_INPUT can be used as follows:

```
  ; ------------------------------------------------------------------------
----
  ; AH=$68: IO_INPUT - Read line of input from user
  ; Input:  None
  ; Output: ELM = pointer to input string (null-terminated, in @PATB_TBUF)
  ;         B = numeric value (parsed via atoi)
  ;         CF = 0 on success, 1 on empty input
  ; Notes:  Reads characters until ENTER (13).
  ;         Supports backspace for editing.
  ;         Echoes characters to screen.
  ;         ENTER is not echoed.
  ;         Max input length limited by PATB_TBUF size.
  ;         If ESC is detected it sets the B flag.
  ; ------------------------------------------------------------------------
----
```

```
; Example 3: IO_INPUT

.address $C000
    LDBLX @prompt
    LDAH $66    ; IO_PRINT
    INT 0x05
    LDAH $68    ; IO_INPUT
    INT 0x05
    ; ELM now contains the user-entered string.

    LDBLX @hellomr
    LDAH $66    ; IO_PRINT
    INT 0x05
    MOV BLX, ELM ; get the name in BLX
    INT 0x05    ; print the name

    LDAL #'!'   ; load a '!' character.
    LDAH $61    ; IO_PUTCHAR
    INT 0x05
    LDAH $64    ; IO_NEWLINE
    INT 0x05
    RET         ; return control back to system.

prompt:
    .bytes "What is your name? ", 0

hellomr:
    .bytes "Hello Mr. ", 0
```

Random Numbers

Random numbers are really useful. Here, we demonstrate the xorshift generator that comes standard in the INT $13 math services library.

```
; Example 4: Random Numbers

.address $C000

start:
    LDBLX @msg
    LDAH $66        ; IO_PRINT
    INT 0x05
loop:
    LDAH $00    ; 16 bit random number in B
    INT $13
    MOD B, #10 ; 0-9
    INC B      ; 1-10

    LDAH $63    ; IO_PUTNUM (prints number in B)
    INT 0x05

    LDAH $64    ; IO_NEWLINE
    INT 0x05
    LDAH $02    ; blocking getkey
    INT 0x10
    CMP AL, #'q'
    JZ @exit
    CMP AL, #'Q'
    JZ @exit
    JMP @loop
exit:
    RET
msg:
    .bytes "Random number program.", 13, 10
    .bytes Press any key or Q to quit.", 13, 10, 0
```

Now you see how to get a random number and how to do blocking getkey! Non-blocking getkey is AH=00.

About INT $13

As an aside to the above, here is the function header from INT $13, AH=$00.

```
  ;
```

```
;===============================================================================
  ; AH=00h - Random Number
  ; Input:  None
  ; Output: A, B = random 16-bit numbers (1-65535)
  ; Notes:  Uses xorshift
  ;
;===============================================================================
```

Getkey

```
  ;
;===============================================================================
  ; AH=02h - Read Character (Blocking)
  ; Input:  None
  ; Output: AL  = ASCII character
  ;         B  = number of keys that were in buffer before this call
  ;         K  = keyboard flags at time of press
  ; Note:   X, Y preserved for cursor position
  ;         This function BLOCKS until a key is pressed
  ;
;===============================================================================
```

Also there is a non-blocking version:

```
  ;
;===============================================================================
  ; AH=00h - Read Character (Non-blocking)
  ; Input:  None
  ; Output: AL = ASCII character (0 if no key)
  ;         B  = number of keys that //were// in buffer before this call (0
= no key)
  ;         K  = keyboard flags at time of press (1 = shift, 8 = caps, etc)
  ;         ZF = 1 if no key (B value is zero), ZF = 0 has key (non-zero B
value)
  ; *** NOTE: calls int10_apply_shift
  ;
;===============================================================================
```

Putting It All Together: Menus

The engine for all these games we love such as Bard's Tale, Ulima, Nethack, King's quest, Alice in wonderland, Maniac Mansion, all were based on text engines. If the engine is good, a modicum of graphics will suffice. The big deal with retro games is that it's all about the gameplay, and the graphics serve the gameplay. What people miss in today's games is that the graphics attempt to look so good that they can replace gameplay. But it goes without saying, that even so, King's Quest was a “better game” than Tomb Raider, even though Tomb Raider is such an amazing game in and of itself. It fell into that trap, replacing gameplay with graphics. I mean, Tomb Raider pulled it off because it was so awesome, but King's Quest, OMG, hard to say which one is better really.

The first thing you need to do is write some menus. Fundamentally printing the menus is easy, but you may wish to color the title of the menus, so you will need to learn how to do colored text. Also, you want to do input validation, so let's work on that too.

First, colored text. it's exactly the same as writing normal text, except you write a color byte. Here, i've added a special function to change the default color text is written in TTY mode. You can also set the color of text at an X and Y location, but that can become tedious because you have to do it character by character.

```
; Example 5: Menus and Colored Text

.address $C000

start:
    ; Clear screen.
    LDAH $10
    INT $10

    ; Set cursor position to X=0, Y=10.
    LDX #0
    LDY #10
    LDAH $15    ; set cursor position
    INT $10

    LDAH $50    ; Set teletype text default color
    LDAL $2F    ; color data in AL: BG=2, FG=F (here, orange)
    INT $10

    LDBLX @str_menu_title
    LDAH $66    ; BASIC IO_PUTSTRING
    INT 0x05

    LDAH $50    ; Set teletype text default color
    LDAL $29    ; color data default for mode 1
    INT $10

    LDBLX @str_menu
    LDAH $66    ; BASIC IO_PUTSTRING
    INT 0x05

    LDAH $02    ; blocking getkey
    INT 0x10

    RET


str_menu_title:
   .bytes "  Choose your class:", 13, 10, 13, 10, 0
```

```
str_menu:
   .bytes "    1) Human", 13, 10
   .bytes "    2) Elf", 13, 10
   .bytes "    3) Half-elf", 13, 10
   .bytes "    4) Dwarf", 13, 10
   .bytes "    5) Orc", 13, 10, 13, 10
   .bytes "  Your choice [1-5, q]? ", 0
```

Major Concept: The Cursor

Before we proceed, it's nice to be able to completely control the terminal. That means, instead of relying solely on PRINT and INPUT we have character level control.

Let's have a quick introduction to four important INT $10 (terminal services) routines.

- AH = 10h – Clear Screen
- AH = 16h – Get Cursor Position (X, Y)
- AH = 15h – Set Cursor Position (X, Y)
- AH = 17h – Print Character at Cursor (Teletype, AL=ascii code)
    - *Note that with print character at cursor, it's a teletype mode, meaning the cursor will auto-advance for you.*

Now let's make a simple demo program that clears the screen and writes ten random letters to the screen. This demo means we can now draw (using charaters) any text we want to the screen in any place – meaning, we can draw a typical 'game board' or 'game screen' in text. This is the first step before graphics programming, since we can edit the in-memory character set.

```
; Example 6: The Cursor

.address $C000

start:
    ; Set up variables, clear screen, etc.
    LDAH $10
    INT $10

    LDT #10     ; ten times for the loop
loop:
    ; 1. Choose random X and Y location on screen
    LDAH $00    ; 16 bit random number in B (and in A)
    INT $13
    MOV X, A
    MOV Y, B

    MOD X, #40 ; clamp to 0-39
    MOD Y, #24 ; clamp to 0-23

    ; 2. Get random letter in AL
    LDAH $00
```

```
    INT $13
    MOD A, #26 ; 0-25
    ADD A, #65 ; AL is now a random letter.

    ; 3. Set cursor position to X and Y
    ; X and Y already set
    LDAH $15
    INT $10

    ; 4. Print character.
    LDAH $17    ; write char (teletype)
    ; AL already has char
    INT $10

    DEC T
    JNZ @loop

done:
    ; 5. Set cursor position to X=0, Y=20.
    LDX #0
    LDY #20
    LDAH $15    ; set cursor position
    INT $10

    RET     ; return control to system.
```

Keeping Track Yourself

In a game, you don't really need the cursor. So instead of using a teletyle print function and calling INT $10, AH=$16, you can just give the values you want for your game screen. Here's an example of putting a character anywhere you want and using any color. Although it looks a bit trivial this ends up proving to be a very useful pattern.

```
; Example 7: put_ccxy

.address $C000

start:
    LDX #0
    LDY #10
    LDC $03
    LDB #'A'

    CALL @put_ccxy
    RET

; put_ccxy(B, X, Y, C)
```

```
; Put Character and Color at X, T
; INPUT:
;     B -- ascii code
;     X -- x location on screen
;     Y -- y location on screen
;     C -- hi nibble (BG) and low niblle (FG)
;
put_ccxy:
    ; Write character in AL at X, Y
    LDAH $11
    INT $10

    ; Set color at X, Y
    ; Input:  CL = color data already sey
    LDAH $13
    INT $10

    RET
```

DEFCHAR Graphics

Mode 1 Graphics is the first step in our graphics adventure. Mode 1 Graphics has two limitations. One, since you are redefining the character set to do graphics, everything must be drawn using 8×8 monocolor stamps (like characters). Two, you can only have 256 of them.

First, use int10h_font_set_char and int10h_glyph_pack to define the character:

```
  ; ------------------------------------------------------------------------
----
  ; AH=49h - glyph_pack -- pack an 8x8 grid and install it as a font glyph
  ;   Input:    AL  = character code (0..255)
  ;             ELM = pointer to 64 source chars (8 rows x 8 cols, row-
major)
  ;   Encoding: ' ' (space) or 0 -> bit clear; any other byte -> bit set
  ;             MSB = leftmost column
  ;   Writes:   8 bytes at petscii_data + charcode*8
  ; ------------------------------------------------------------------------
----
```

You can also use AH=48h if you have the pre-compiled bytes:

```
  ; ------------------------------------------------------------------------
----
  ; AH=48h - font_set_char -- install an 8-byte glyph into the font
  ;   Input:    AL  = character code (0..255)
  ;             ELM = pointer to 8 source bytes
  ;   Writes:   8 bytes at petscii_data + charcode*8
  ;   Destroys: A   (BLX/FLD/C/K preserved)
  ; ------------------------------------------------------------------------
```

```
----
```

Here's how it works:

```
; Example 8: DEFCHAR

.address $C000

start:
    LDELM @smiley
    LDAH $49
    LDAL #'!'  ; replace the ! character
    INT $10
    RET
    ; Now the ! is replaced by a smiley.
smiley:
    .bytes "        "
    .bytes "  #  #  "
    .bytes "        "
    .bytes " #    # "
    .bytes " #    # "
    .bytes "  ####  "
    .bytes "        "
    .bytes "        "
```

Unfortunately, the conversion to a smiley is permanent! At least, until you reboot the system (or change it back).

Using this concept in Mode 3

Now, interestingly enough, you can translate this graphics technique directly into Mode 3. First, let's adapt BASIC's DRAWCHAR (INT $18 AH=$60 and $61) so we can blit characters in Mode 3.

```
  ; ------------------------------------------------------------------------
----
  ; AH=61h -- stamp_char -- render a glyph onto mode-3, transparent
background
  ;   Input:  AL = character code (0..255)
  ;           X  = base X (pixel)
  ;           Y  = base Y (pixel)
  ;           CL = foreground color (only looks at low nibble)
  ;   Uses:   INT 18h AH=01h (plot pixel);
  ;           glyph source = VM1_CHAR_ROM ($01E000)
  ; ------------------------------------------------------------------------
----
```

We'll toss in some fancy arrow key movement and a q for quit key, to spice things up:

```
; Example 9: stamp_char

.address $C000

start:
    ; Switch to mode 3
    LDA $4003   ; 40h = set mode, 3 = mode 3
    INT $10

    ; Clear screen
    LDAH $20
    INT $10

    LDX #0
    LDY #0

loop:
    ; draw an 'x' at X, Y
    LDAL #'x'
    LDAH $61    ; stamp char
    LDC #9      ; dark blue
    INT $18

    LDAH $02    ; blocking getkey
    INT 0x10

    LDAH $60    ; erase char
    LDC #0      ; black background
    INT $18

    CMP AL, #'q'
    JZ @exit
    CMP AL, #'Q'
    JZ @exit

    CMP AL, #128
    JZ @move_left
    CMP AL, #129
    JZ @move_down
    CMP AL, #130
    JZ @move_up
    CMP AL, #131
    JZ @move_right

    JMP @loop

move_left:
    DEC X
```

```
    DEC X
    JNN @loop   ; if X didn't go negative, just continue.

    LDX #0      ; Don't allow X to go below 0.
    JMP @loop

move_right:
    INC X
    INC X
    CMP X, #312
    JNC @loop

    ; C is set if X was >= 312
    LDX #311
    JMP @loop


move_up:
    DEC Y
    DEC Y
    JNN @loop   ; if Y didn't go negative, just continue.

    LDY #0
    JMP @loop

move_down:
    INC Y
    INC Y
    CMP Y, #191
    JNC @loop

    LDY #191
    JMP @loop

exit:
    ; Switch back to mode 1
    LDA $4001
    INT $10
    RET
```

Multi-Color Character Graphics in Mode 3

The way to do multicolor graphics is to use different tiles for each color component. For example, you can have one tile with the blue pixels and the other tile with the yellow pixels. When you draw the tiles on top of each other, it will produce a multicolor effect.

Music

Randomly, in the middle of nowhere, we will explain to you; yes, you can play music in your game. Our PLAY statement is multi-voice; please see the relevant documentation for more information. For now, here's a short little ditty reminiscent of England in the summer of the 1600's.

```
10 PLAY "XV1 MN W5 XP3000 O4 L4 T180"
15 PLAY "XV2 MN W5 XP3500 O3 L4 T180"
20 PLAY "XV1 F1"
25 PLAY "XV2 R1"
30 PLAY "XV1 A4. B-8 > C4 C4 <"
35 PLAY "XV2 R1"
40 PLAY "XV1 B-4 A4 G4 F4"
45 PLAY "XV2 R1"
50 PLAY "XV1 > C2 C4. C8 <"
55 PLAY "XV2 R2 F4. F8"
60 PLAY "XV1 > C4 D2 C4 <"
65 PLAY "XV2 F4 < B-2 A4"
70 PLAY "XV1 B-4 A4 G4 G4"
75 PLAY "XV2 B-4 F4 > C4 C4"
80 PLAY "XV1 A1"
85 PLAY "XV2 < F1"
```

Now you may wonder, why incorporate a BASIC example? Here, we can use BASIC to quickly test musical ideas. When it's time to put them into a game, we do this:

```
; Example 10: Music

.address $C000

    LDELM @fair_phyllis
    LDAH $52        ; Play Song
    INT $11
    RET

fair_phyllis:
    .bytes "XV1 MN W5 XP3000 O4 L4 T180"
    .bytes "XV2 MN W5 XP3500 O3 L4 T180"
    .bytes "XV1 F1 "
    .bytes "XV2 R1 "
    .bytes "XV1 A4. B-8 > C4 C4 < "
    .bytes "XV2 R1 "
    .bytes "XV1 B-4 A4 G4 F4 "
    .bytes "XV2 R1 "
    .bytes "XV1 > C2 C4. C8 < "
    .bytes "XV2 R2 F4. F8 "
    .bytes "XV1 > C4 D2 C4 < "
```

```
    .bytes "XV2 F4 < B-2 A4 "
    .bytes "XV1 B-4 A4 G4 G4 "
    .bytes "XV2 B-4 F4 > C4 C4 "
    .bytes "XV1 A1 "
    .bytes "XV2 < F1", 0
```