Smol-16 Technical Documentation

Corwin McKnight

March 27th, 2017

Contents

1 Introduction

Smol-16 is a Fantasy Game Console. It's what I think a 16 bit system could be, given they didn't die out and had a bit more room to grow.

I love the look of Super Nintendo and Sega Genesis games, and the like. They're ”bitty”, but still have a lot of depth. 8-bit to me is too restrictive, and 32-bit is too complex. 16-bit feels like the perfect balance.

1.1 History

Originally, I wanted carts to be written in Lua, but then I realized, that's not restrictive enough! Lua isn't easily rate limited, so programs would be able to run *way* faster then I wanted.

Other limitations (like on code size) weren't feasible as well. In the Lua version, I also had memory, but there was no reason to use it (after all, you had Lua variables).

So, I've decided to make an actual system, with IO registers, Memory, devices, banks, etc.

1.2 Model

The SNES, and PICO-8 were the models for this system. Most of the specs were borrowed or used as inspiration for Smol-16's specifications.

1.3 Hardware

Since Smol-16 is attempting to be a 16 bit console, modern hardware won't do. All the hardware must be created. The specs are modeled after the Super Nintendo.

2 Features

3 The CPU

The CPU is a custom 16-bit RISC processor that runs at `10.24 MHz`. The modern day equivalent to it would be an ARM chip. Infact, it's design is directly inspired (or even ripped) from ARM chips.

3.1 Registers

The CPU has 13 General Purpose Registers, `R0-12` . In addition, it has a `PC` register (R15), a Stack Pointer (R14), and a Flags register (R13).

Each register is 16-bits long. Some operations work on 8-bit registers, and therefore can access half-word versions of registers. This effectively allows **double** the registers (32 instead of 16).

3.2 Interrupts

The CPU can issue interrupts at any time. These immediately stop the processor and push all registers to the stack.

Then, the CPU runs one double length instruction from the `IVT`, based on the index of the IRQ. There are 32 possible interrupts, so the `IVT` takes up 64 bytes of memory. It is located at address `0x0000`.

Note: The CPU is put into a special mode where all interrupts are queued. When the CPU `ret`'s, it exits this mode. If an interrupt is in the queue, the processor will not execute the next instruction, but will immediately raise another IRQ. The processor will start from the lowest IRQ and move up (IRQ0 to IRQ31).

3.3 Instructions

Instructions are 16-bits in length, generally. There are some that extend to 32-bits (double length instructions).

Layout in memory generally depends on the instruction, however, these are always the same:

3.3.1 Opcode

The actual opcode is stored within the first byte. There are 256 possible instructions. Some instructions have multiple forms / parameters, based on the second byte.

3.3.2 Conditionals

Some instructions can be conditional. This means that they will be executed under certain conditions. If the test fails, they effectively are a `nop`

Bits	Mnemonic	Description
0000	EQ	Z flag set (equal)
0001	NE	Z flag clear (not equal)
0010	HS	C flag set (unsigned higher or same)
0011	LO	C flag clear (unsigned lower)
0100	MI	N flag set (negative)
0101	PL	N flag clear (non-negative)
0110	VS	V flag set (overflow)
0111	VC	V flag clear (no overflow)
1000	HI	C flag set and Z flag clear
1001	LS	C flag clear or Z flag set
1010	GE	N flag set and V set or N clear and V clear
1011	LT	N set and V clear or N clear and V set
1100	GT	Z clear with (either N or V set), or N clear and V set
1101	LE	Z set or (N set and V clear), or N clear and V set
1110	AL	Always (no condition)
1111	RS	Reserved

Table 1: Conditional flag bits

If this seems familiar, it's because it is directly taken from ARM chips.

3.3.3 Registers

Registers are encoded as 4 or 5 bits. Their position depends on the instruction.

Bits	Mnemonic
0000	R0
0001	R1
0010	R2
0011	R3
0100	R4
0101	R5
0110	R6
0111	R7
1000	R8
1001	R9
1010	R10
1011	R11
1100	R12
1101	R13 (FLAGS)
1110	R14 (SP) (Stack Pointer)
1111	R15 (PC) (Program Counter)

Table 2: Register bits

Generally, registers take up 4 bytes, but if the instruction requires 8-bit registers, it then needs a fifth bit. This bit is whether it's referring to the high byte. So, 1011, 1, means means "Register R11, high byte".

3.3.4 Literal Operands

If literal operands are required, they are generally pushed to the second word. There is a limit of 1 literal operand per instruction, otherwise, the instruction would be too large. The exception is if the literal operands are 8-bit, in which case there can be 2.

3.4 ALU

The ALU in the CPU is 16-bit. It can do operations on 8-bit and 16-bit numbers. It can do signed and unsigned operations as well.

It has limited 32-bit operational capacity, but can do basic operations on 32-bit numbers. 64-bit numbers are not supported.

While most CPU's from the era support BCD (Binary Coded Decimal) support, I will not include it.

4 Picture Processing Unit

The Picture Processing Unit (PPU) is a hardware component in the Smol-16 system. It is responsible for outputting graphics.

The screen size is 256px x 224px, with a total of 256 colors on screen at once. It also has hardware accelerated drawing functions, callable by the user.

4.1 Address Space

The PPU address space is not directly addressable. It must be accessed through registers. It has 2MB of addressable space in total. This is split between the VRAM, Sprites, Palette indexes, and so on. 0x100 * 0xf

Start	End	Description
0x0000	0xDFFF	Video RAM
0xE000	0xEFFF	Sprite Data
0xF000	0xF1FF	Global Screen Palette
0xF200	0xF3FF	Sprite Palette Tables
0xF400	0xF5FF	Sprite Data Tables
0xFF00	0xFFFF	Registers

4.2 Drawing

The general drawing routine is:

1. Wait for `PPU_VBLANK`
2. Clear Screen
3. Draw Sprite Layer
4. Signal PPU that frame can be rendered.

Note: The PPU will not accept drawing commands until `PPU_VBLANK` is 0.

Note: The PPU waits for a write to `PPU_VBLANK` to allow itself to draw the next frame. It will then wait for the next `VBLANK`. This means that if you go over your frame budget (16ms for 60fps), you will be VSYNC'd to 30

4.3 Registers

The register offset is at `0xFF00`. Take all registers here as offsets of this value

4.4 Video RAM

The VRAM is accessible through a system interrupt. It is `0xE000` bytes long, with each byte being a palette index. This allows for software rendering. It is slower then using hardware accelerated sprites, however.

4.5 PCT

4.5.1 Color format

Colors are stored in BGR-15 format. This means there are 32,768 possible colors that the PPU can display.

4.5.2 Index format

Each color index is simply a byte. This byte refers to the location of the color data relative to the PCT.

$$y = 2x + o$$

, Where x is the index, and o is the location of the PCT in memory.

4.6 Screen Palette Table

The Screen Palette table is a 512 byte long table of BGR-15 color values. Each color value is the value the PPU replaces when given an index.

Note: Color 0 is always transparent when used in hardware accelerated rendering.

4.7 Sprites

Sprites can have 16 colors, and are 8x8 pixels. There can be a total of 128 sprites loaded into the PPU at any one time. There is no limit on how many sprites can actually be displayed at once, due to the fact that the software must instruct the PPU to render a sprite. This just requires additional passes.

Sprites are stored as a series of indexes. Sprites are 8x8, so they are 32 bytes each (4 bits per index). Each index refers to the offset in the Sprite Palette Table that gives its color.

4.7.1 Sprite Palette Table

A Sprite Palette Table is a table 16 bytes long. It defines mappings between sprite data and actual palette colors.

There are 32 of them available, so they take up `0x200` bytes

4.7.2 Sprite Data Table

The Sprite Data table is a group of tables stored in memory.

When the program system calls a sprite to be rendered, properties that it'll need to be rendered are stored in the SDT with the index it is called at.

The SDT is laid out like this:

Byte	Description
0x0	X position (- 64)
0x1	Y position (- 64)
0x2	Sprite ID
0x3	Flags (bit 1 = flip x, bit 2 = flip y)

There is a limit of 128 defined sprites, so the size is `0x200` bytes.

4.8 Layers

5 Sound Processing Unit

5.1 Channels

The SPU has 8 hardware channels available. Each one can have Channel effects applied to it. However, channels 6 & 7 are by default reserved for sound effects. Music that's playing will lose whatever samples are being played on channels 6 & 7.

5.1.1 Effects

5.2 PCM Samples

5.3 Sound Effects

6 Memory

The processor has 128kb of ram at it's disposal. However, due to addressing limitations, only 64kb is visible at any one time.

Memory is laid out in Little Endian format. The processor can only access up to words normally.

6.1 Banks

6.2 Address Space

7 Input

7.1 Controller

7.2 Mouse

8 Carts

8.1 Header

8.2 Data

8.3 IO

9 Graphics Backends

9.1 None

The Emulator is capable of outputting with no graphic backend.

9.2 SDL

10 Instruction Set

11 Glossary

Table 3: Terms

Term	Meaning
CPU	Central Processing Unit
PPU	Graphics Controller
SPU	Sound Processor (DSP)
PCT	Palette Color Table
SPT	Sprite Palette Table
Layer	A separate image that is drawn over the screen

12 Credits and Used Libraries