sim6502 - 6502 Assembly Testing Framework

Barry Walker

January 28, 2026

Contents

1 Introduction **2**

2 System Emulation **2**
- 2.1 Available Systems . . . 2
- 2.2 System Declaration . . . 2
- 2.3 C64 System . . . 2
- 2.4 Loading ROMs . . . 3

3 Processor Selection (Legacy) **3**
- 3.1 Specifying Processor Type (Deprecated) . . . 4
- 3.2 65C02 Additional Instructions . . . 4
  - 3.2.1 Stack Operations . . . 4
  - 3.2.2 Store Zero . . . 4
  - 3.2.3 Branch Always . . . 5
  - 3.2.4 Accumulator Increment/Decrement . . . 5
  - 3.2.5 Bit Manipulation . . . 5
  - 3.2.6 Zero Page Indirect Addressing . . . 5
- 3.3 6510 I/O Port . . . 5
  - 3.3.1 C64 Memory Banking . . . 5
- 3.4 System Examples . . . 6
  - 3.4.1 Generic 6502 Example . . . 6
  - 3.4.2 Generic 6510 Example . . . 6
  - 3.4.3 C64 Example . . . 6
  - 3.4.4 65C02 Example . . . 7

4 Test Suite Structure **7**

5 License **7**

1 Introduction

sim6502 is a tool to help you unit test your 6502 assembly language programs. It works by running your assembled programs with a 6502 simulator and then allowing you to make assertions on memory and CPU state.

2 System Emulation

sim6502 supports both simple processor emulation and full system emulation. The `system()` declaration is the recommended way to configure your test environment, as it provides proper memory mapping for specific systems.

2.1 Available Systems

- **c64** - Commodore 64 with full memory banking support
- **generic_6502** - Flat 64KB RAM with MOS 6502 processor
- **generic_6510** - Flat 64KB RAM with 6510 I/O port at $00-$01
- **generic_65c02** - Flat 64KB RAM with WDC 65C02 processor

2.2 System Declaration

Use the `system()` statement at the beginning of a suite:

```
suites {
  suite("C64 Tests") {
    system(c64)

    test("banking-test", "Test C64 memory banking") {
      // Full C64 memory banking available
    }
  }
}
```

2.3 C64 System

The C64 system provides accurate memory banking controlled by the 6510 processor port at $01:

- **LORAM (bit 0)** - BASIC ROM visibility at $A000-$BFFF
- **HIRAM (bit 1)** - KERNAL ROM visibility at $E000-$FFFF
- **CHAREN (bit 2)** - Character ROM vs I/O registers at $D000-$DFFF

Critical behavior: Writes *always* go to RAM underneath, even when ROM is visible. This enables "under-ROM" RAM techniques commonly used in C64 programs.

```
suites {
  suite("C64 Banking Example") {
    system(c64)

    test("write-under-rom", "Write to RAM under BASIC ROM") {
      ; Write to RAM at $A000 (BASIC ROM region)
      $A000 = $42

      ; Bank out BASIC: $01 = $35 (LORAM=0)
      $01 = $35

      ; Execute code
      $0300 = $60  ; RTS
      jsr($0300, stop_on_rts = true, fail_on_brk = false)

      ; Now RAM is visible at $A000
      assert(peekbyte($A000) == $42, "RAM value visible")
    }
  }
}
```

2.4 Loading ROMs

For systems that support ROMs (like C64), use the `rom()` declaration:

```
suites {
  suite("C64 with Custom ROMs") {
    system(c64)
    rom("basic", "basic.rom")
    rom("kernal", "kernal.rom")

    test("with-roms", "Test with loaded ROMs") {
      // ROMs are now available for banking
    }
  }
}
```

3 Processor Selection (Legacy)

The `processor()` declaration is still supported for backward compatibility but is deprecated. Use `system()` instead for new code.

sim6502 supports multiple processor variants in the 65xx family:

- **6502** - Original NMOS MOS Technology 6502 (default)
- **6510** - Same ISA as 6502, adds I/O port at $00-$01 (Commodore 64)
- **65C02** - WDC CMOS variant with additional instructions

3.1 Specifying Processor Type (Deprecated)

Use the `processor()` statement at the beginning of a suite:

```
suites {
  suite("My 65C02 Tests") {
    processor(65c02)  ; Deprecated - use system(generic_65c02)

    test("test-1", "Uses 65C02 opcodes") {
      // PHX, PLX, STZ, BRA, etc. available
    }
  }
}
```

If no `processor()` or `system()` statement is specified, the default is generic_6502.

3.2 65C02 Additional Instructions

The 65C02 adds the following instructions not available on the 6502/6510:

- **PHX, PLX** - Push/Pull X register
- **PHY, PLY** - Push/Pull Y register
- **STZ** - Store Zero
- **BRA** - Branch Always
- **INC A, DEC A** - Increment/Decrement Accumulator
- **TRB, TSB** - Test and Reset/Set Bits
- Zero Page Indirect addressing mode: `LDA ($50)`

3.2.1 Stack Operations

The 65C02 adds dedicated stack operations for the X and Y registers:

```
PHX     ; Push X register onto stack
PLX     ; Pull X register from stack
PHY     ; Push Y register onto stack
PLY     ; Pull Y register from stack
```

These instructions behave like PHA/PLA but operate on the X and Y registers respectively.

3.2.2 Store Zero

The STZ instruction stores zero to a memory location, supporting multiple addressing modes:

```
STZ $50         ; Zero page
STZ $50,X       ; Zero page, X
STZ $1234       ; Absolute
STZ $1234,X     ; Absolute, X
```

3.2.3 Branch Always

The BRA instruction provides an unconditional relative branch:

```
BRA label       ; Always branch to label
```

This is more efficient than using a conditional branch that always succeeds.

3.2.4 Accumulator Increment/Decrement

The 65C02 adds INC and DEC instructions that operate directly on the accumulator:

```
INC A           ; Increment accumulator
DEC A           ; Decrement accumulator
```

3.2.5 Bit Manipulation

TRB and TSB provide efficient bit testing and manipulation:

```
TRB $50         ; Test and Reset Bits (clear bits where A has 1s)
TSB $50         ; Test and Set Bits (set bits where A has 1s)
```

Both instructions set the Zero flag based on the result of A AND memory (before modification).

3.2.6 Zero Page Indirect Addressing

The 65C02 adds a new addressing mode that uses zero page indirect addressing without an index register:

```
LDA ($50)       ; Load A from address pointed to by $50-$51
STA ($50)       ; Store A to address pointed to by $50-$51
```

This mode is available for: LDA, STA, ORA, AND, EOR, ADC, CMP, and SBC.

3.3 6510 I/O Port

The 6510 (used in the Commodore 64) has an I/O port at addresses $00 and $01:

- **$00** - Data Direction Register (DDR)
- **$01** - Data Port

When using `processor(6510)`, reads and writes to these addresses access the I/O port registers instead of RAM. The DDR controls which bits are inputs (0) or outputs (1).

3.3.1 C64 Memory Banking

On the Commodore 64, the 6510 I/O port controls memory banking:

```
; Set DDR - bits 0-5 as outputs
LDA #$2F
STA $00

; Set data port - standard C64 configuration
LDA #$37
STA $01
```

3.4 System Examples

3.4.1 Generic 6502 Example

```
suites {
  suite("Standard 6502 Tests") {
    system(generic_6502)  ; Optional - this is the default

    test("basic-test", "Basic 6502 operation") {
      a = $42
      $0300 = $60  ; RTS
      jsr($0300, stop_on_rts = true, fail_on_brk = false)
      assert(a == $42, "Accumulator test")
    }
  }
}
```

3.4.2 Generic 6510 Example

```
suites {
  suite("6510 Tests") {
    system(generic_6510)

    test("io-port", "Test 6510 I/O port") {
      $00 = $2F   ; Set DDR
      $01 = $37   ; Set data port
      $0300 = $60  ; RTS
      jsr($0300, stop_on_rts = true, fail_on_brk = false)
      assert(peekbyte($00) == $2F, "DDR configured")
      assert(peekbyte($01) == $37, "Port configured")
    }
  }
}
```

3.4.3 C64 Example

```
suites {
  suite("C64 Tests") {
    system(c64)

    test("banking", "Test C64 memory banking") {
      $A000 = $42    ; Write RAM under BASIC
      $01 = $35      ; Bank out BASIC
      $0300 = $60    ; RTS
      jsr($0300, stop_on_rts = true, fail_on_brk = false)
      assert(peekbyte($A000) == $42, "RAM visible")
    }
  }
}
```

3.4.4 65C02 Example

```
suites {
  suite("WDC 65C02 Tests") {
    system(generic_65c02)

    test("enhanced-ops", "Test 65C02 enhancements") {
      a = $41
      $0310 = $1a  ; INC A
      $0311 = $60  ; RTS
      jsr($0310, stop_on_rts = true, fail_on_brk = false)
      assert(a == $42, "INC A worked")
    }
  }
}
```

4 Test Suite Structure

For information on the complete DSL syntax, test structure, and other features, please refer to the project README.md file.

5 License