SD-8516 Programmer's Reference Guide

- NOTE: If this is your first time reading about the SD-8516, you may wish to read the SD-8516 User's Guide first!
- Also see: Tutorials and Guides

1. Introduction

The **Stellar Dynamics SD-8516** represents a categoretroical reimagining of microprocessor architecture. This 16-bit CPU, implemented in AssemblyScript for the VC-3 computer system, delivers performance exceeding conventional silicon constraints through advanced cross-boundary resonance microcascades.

The SD-8516 is intended to be an easy to learn architecture which remains era-authentic.

CPU	Opcodes	Assembler	Notes
SD-8516	56 opcodes	105 opcodes	
6809	59 opcodes	154+	“the most elegant 8 bit CPU ever designed”
8086	117 opcodes	117	standard of the era
6502	151 opcodes	151	standard of the era
Z80	158 opcodes	hundreds	prefix machine–158 base opcodes
8080	244 opcodes		

Key Specifications

- 16-bit architecture with 16 general-purpose registers
- 24-bit addressing with register pairing system
- Memory: 256KB addressable via 4-bank system
- ~40× performance improvement over legacy 8510 design

Default Performance:

- Standard clock speed: 4MHz (1.28 MIPS)
- Overclocked: 10 MHz (3 MIPS)
- Memory bandwidth: 16 MB/s
- Sound system overhead: < 5% CPU time
- Video refresh: 60 Hz (16.67ms frame time)
- BASIC execution speed: 3,000 lines/sec.
- Forth: 360,000 words/sec

Technical Implementation

- **Architecture:** WebAssembly-based virtual CPU
- **Languages:** AssemblyScript (CPU core), JavaScript (I/O systems)
- **Memory Model:** 4 banks of 64k RAM

- **Audio Backend:** SD-450 4 voice polyphonic 5 waveform Audio System
- **Video Backend:** 9 mode Text and Graphics pixel-perfect render engine

4. Advanced PEEK and POKE

Commands such as DEFCHAR and DRAWCHAR are merely glorified PEEK and POKE commands. At 2,500 lines of BASIC per second on a 4 MHz system, large numbers of PEEK and POKE commands become very useful to control the computer. They are not slow at all!

WAIT

In fact, if you were to write a MIDI player or graphics application in BASIC, you may find the need to time your programs accurately or slow things down – perhaps for animation. To do this you can use the WAIT command:

```
WAIT 50 : REM THIS WILL WAIT FOR 50 MILLISECONDS
```

PEEK Memory Map

BANK	LOCATION	DESCRIPTION
BANK 1	PEEK(61184)	Get current video mode.
BANK 1	PEEK(61185)	Get max columns of current video mode.
BANK 1	PEEK(61189)	Video clock low byte (four bytes in length).
BANK 1	PEEK(61198)	Cursor column.
BANK 1	PEEK(61199)	Cursor row.
BANK 1	PEEK(61201)	Number of keys in keyboard buffer.
BANK 1	PEEK(61202)	First byte pair of keyboard buffer.
BANK 1	PEEK(61251)	Palette mode.
BANK 1	PEEK(61440)	TEXT 0,0 in mode 1
BANK 1	PEEK(63488)	COLOR 0,0 in mode 1
BANK 1	PEEK(61440)	TEXT 0,0 in mode 2
BANK 1	PEEK(63488)	COLOR 0,0 in mode 2

POKE memory map

You can also POKE to various locations to affect the sound system.

BANK 1	POKE 61312,0	Set low byte of voice 1 frequency (length: 3 bytes)
BANK 1	POKE 61328,0	Set low byte of voice 2 frequency (length: 3 bytes)
BANK 1	POKE 61344,0	Set low byte of voice 3 frequency (length: 3 bytes)
BANK 1	POKE 61360,0	Set low byte of voice 4 frequency (length: 3 bytes)

For more information on the sound system see Appendix 5 Sound System.

5. Machine Language

WHAT IS MACHINE LANGUAGE?

At the heart of every microcomputer is a central microprocessor. It is a very special microchip that acts as the “brain” of the computer. The SD-8516/VC-3 is no exception. Every microprocessor understands its own language of instructions. These instructions are called *machine language instructions.*

More precisely, machine language is the **only** programming language that your VC-3 understands. It is the native language of the machine.

If machine language is the only language that the 8516 understands, then how does it understand STELLAR BASIC? BASIC is **not** the machine language of the 8516. What, then, makes the 8516 understand BASIC instructions like `PRINT` and `GOTO`?

To answer this question, you must first see what happens inside your 8516. Apart from the microprocessor – which is the brain of the machine – there is also a machine language program stored in a special type of memory that cannot be changed. More importantly, it does not disappear when the 8516 is turned off, unlike a program that you may have written.

This machine language program is called the **Operating System** of the SD-8516. Your SD-8516 knows what to do when it is turned on because its Operating System program is automatically “run.”

WHAT DOES MACHINE CODE LOOK LIKE?

WHAT DOES MACHINE CODE LOOK LIKE? Machine code stored in computer memory as a series of numbers. The computer decides what to do based on those numbers. So, a simple program might look like this:

```
$C000: 00 34 10 C0 00 00 20 66
$C008: 86 05 00 20 64 86 05 85
$C010: 48 45 4C 4C 4F 20 57 4F
$C018: 52 4C 44 21 00 00 00 00
```

To enter and RUN this style of program you can use a program like 'wozmon' for the Apple I, MONITOR for the Commodore 128, or DEBUG for 8086 computers using MS-DOS. You could also write a simple BASIC program to help you enter the numbers.

HEXMON

HEXMON

Your SD-8516 comes with an entry program in ROM called HEXMON. To start HEXMON type “MON” at the ready prompt. For more information on HEXMON see: HEXMON.

If you enter the above program (just type the lines in) and then type:

```
    C000R
```

The program will run. C000 is the memory address and R is the traditional wozmon command to run code starting at that location. What does this program do? Try it now!

ASSEMBLY LISTINGS

ASSEMBLY LISTINGS

Another way to write the code above would be a full assembly/disassembly listing (a full, or complete program listing). Here's an example of the same program shown as a full listing:

```
     ADDR    BYTES          INSTRUCTION          COMMENT
    ------------------------------------------------------------------------
--------
    $C000:   00 34          LDBLX @msg           ; Load pointer to string
into BLX
             10 C0 00                            ; (Bank 0, address $C0 10)
    $C005:   00 20 66       LDAH $66             ; IO_PRINT_STR function
    $C008:   86 05          INT $05              ; BASIC services library
    $C00A:   00 20 64       LDAH $64             ; IO_NEWLINE function
    $C00D:   86 05          INT $05              ; Call BASIC services
    $C00F:   85             RET                  ; Return to caller
    $C010:                  msg:                 ; String data label
(equates to $C000 above)
    $C010:   48 45 4C 4C    .bytes "HELLO        ; H E L L
    $C014:   4F 20 57 4F           WORLD!", 0    ; O   W O
    $C018:   52 4C 44 21                         ; R L D !
    $C01C:   00                                  ; null terminator (zero
terminated string)
```

This full listing can also be used to assemble the program in the monitor; just type in the program in the monitor as usual:

```
    $C000:   00 34 10 C0 00
    $C005:   00 20 66
    $C008:   86 05
    $C00A:   00 20 64
    $C00D:   86 05
    $C00F:   85
    $C010:   48 45 4C 4C
    $C014:   4F 20 57 4F
    $C018:   52 4C 44 21
    $C01C:   00
```

This is the exact same style of listing generated by the DUMP command, or the range examine command in HEXMON (ex. `C000.C020`). It's just broken up by instruction.

LOAD and SAVE for Assembly

HEXMON is equipped with several commands that help you load, save and publish machine language programs.

Command	Function
L	Load a machine language file using the address in the file header. This only works with programs saved using W (see below), not S.
####L	Load a machine language file explicitly at the address given. This ignores the address in the program header, if there is one.
####.####S	Save bytes in range to file. You can use this to save data or code you will load using the L command.
####.####W	Save machine code in range to file. You can use this to save your programs to disk, because it will include a header allowing it to be loaded back into the same location.
####.####P	Publish. This will create a file with a publishable listing including checksum data.

The important distinction is between the S and W command. S saves bytes only, W includes a header. If you want to be able to click on the Load button or just “L,filename” in MON, then you must save the machine code with W to include the address header.

For more information on HEXMON see: HEXMON.

ASSEMBLY FROM BASIC

STELLAR BASIC ASSEMBLE COMMAND

The above “hexmon” style machine language listings are convenient for print media, because they are compact. But they are not ideal for learning and study. Thankfully for learning and general programming purposes, one may use the STELLAR BASIC v1.0 **ASSEMBLE** command.

This system works because the VC-3 KERNAL does not tokenize basic. It keeps whatever you type in a string in memory, and the ASSEMBLE command uses this to pass the data to the assembler. The same sort of idea was used on the Atari-ST Assembler (please see: https://www.atarimania.com/documents/The-Atari-Assembler.pdf).

On the SD-8516, it works like this:

```
10 ASSEMBLE
20 LDBLX @msg
30 LDAH $66 ; BASIC IO_PRINT_STR
40 INT $05
50 LDAH $64 ; BASIC IO_NEWLINE
60 INT $05
70 RET
```

```
    80 msg:
    90 .bytes "HELLO WORLD!", 0
```

If you enter this like you would a STELLAR BASIC V1.0 computer program and type **RUN**, the system will assemble your program at memory address $030100. You can view your program yourself, by typing

```
    DUMP 030100
```

Next try running the program by typing **SYS**. This will jump to $030100.

This is the exact same program as before with one exception; the automatically assembled version will begin with:

```
  $00C000:   00 34 **10 C0 00**  ; string is at Bank $00
```

but the mon-entered one will show:

```
  $030100:   00 34 **10 01 03**  ; string is at Bank $03
```

And the reason for this is that in the listing we typed in on the monitor we used the address 00 C0 00 (i.e. $C000) so the string began at $C010. But for the automatically assembled version, it begins at 03 01 00 so the string address is 03 01 10.

- $00C000: 00 34 **10 C0 00** ; 00 is LD, 34 is BLX (00 34 is LD BLX) – and $C010 in little endian
- $030100: 00 34 **10 01 00** ; different address!

Otherwise they are all the exact same program, whether you enter it in machine code or in BASIC assembly!

***NOTE: Because this program is listed in BASIC memory space, you can load and save it using the LOAD and SAVE commands.**

Your First Program: LDA

The SD-8516's registers are 16-bit word-length and may be used for any general purpose. Here is your first program!

```
    10 ASSEMBLE
    15 .address $C000
    20 LDA #2
    30 LDB #5
    40 MUL A, B
    50 MOV B, A
    60 LDAH $63
```

```
70 INT $5
80 INC AH
90 INT $5
100 RET
```

Fix Me! under construction

Appendix 1. Lore

Since the days of the first minicomputers, Stellar Dynamics has been at the forefront of microarchitecture design. The SD-8516 is not simply an iteration upon its predecessors; it is a categorical reimagining of what a “processor” can be when unshackled from quantum locality.

While our earliest designs struggled with resonance cascade instability, the SD-8516 delivers stable, predictable cross-boundary resonance microcascades at clock rates exceeding the theoretical limits of conventional silicon.

These advancements position the Stellar Dynamics SD-8516 as the definitive architecture for next-generation computation: a bridge between classical logic engines and the emergent technologies of multidimensional processing.

> This SD-8516 PROGRAMMER'S REFERENCE GUIDE has been developed as a working tool and reference source for those of you who want to maximize your use of the built-in capabilities of your VC-3 Computer System. This manual contains the information you need for your programs, from the simplest example all the way to the most complex. The PROGRAMMER'S REFERENCE GUIDE is designed so that everyone from the beginning BASIC programmer to the professional experienced in SD-8516 machine language can get information to develop his or her own creative programs. At the same time this book shows you how clever your SD-8516 really is.
>
> This REFERENCE GUIDE is not designed to teach the BASIC programming language or the SD-8516 machine language. There is, however, an extensive glossary of terms and a “semi-tutorial” approach to many of the sections in the book. If you don't already have a working knowledge of BASIC and how to use it to program, we suggest that you study the SD-8516 USER'S GUIDE that came with your computer. The USER'S GUIDE gives you an easy to read introduction to the BASIC programming language. If you still have difficulty understanding how to use BASIC then turn to the back of this book (or Appendix N in the USER'S GUIDE) and check out the Bibliography.
>
> The SD-8516 PROGRAMMER'S REFERENCE GUIDE is just that; a reference. Like most reference books, your ability to apply the information creatively really depends on how much knowledge you have about the subject. In other words if you are a novice programmer you will not be able to use all the facts and figures in this book until you expand your current programming knowledge.
>
> What you can do with this book is to find a considerable amount of valuable programming reference information written in easy to read, plain English with the programmer's jargon explained. On the other hand the programming professional will find all the information needed to use the capabilities of the SD-8516 effectively.

WHAT'S INCLUDED?

- Our complete “BASIC dictionary” includes Stellar BASIC 1.0 language commands, statements and functions listed in alphabetical order. We've created a “quick list” which contains all the words and their abbreviations. This is followed by a section containing a more detailed definition of each word along with sample BASIC programs to illustrate how they work.
- If you need an introduction to using machine language with BASIC programs our layman's overview will get you started (See: Chapter 5).
- A powerful feature of all VC systems is called the KERNAL. It helps insure that the programs you write today can also be used on the VC-3 system of tomorrow.
- The Input/Output Programming section gives you the opportunity to use your computer to the limit. It describes how to hook-up and use everything from disk drives, to telecommunication devices called modems.
- You can explore the world of SPRITES, programmable characters, and high resolution graphics for the most detailed and advanced animated pictures in the microcomputer industry.
- You can also enter the world of music synthesis and create your own songs and sound effects with the best built-in synthesizer available in any personal computer.
- If you're an experienced programmer, the soft load language section gives you information about the SD-8516's ability to run C and other high level languages. This is in addition to BASIC.

Think of your SD-8516 PROGRAMMER'S REFERENCE GUIDE as a useful tool to help you and you will enjoy the hours of programming ahead of you.

Appendix 2. CPU Architecture

Register Set

The SD-8516 features sixteen 16-bit registers:

Register	Name	Primary Use
R0	A	Accumulator
R1	B	Accumulator
R2	X	Index/General
R3	Y	Index/General
R4	I	Loop/General
R5	J	Loop/General
R6	K	Loop/General
R7	T	Temporary/General
R8	M	Memory Pointer
R9	D	Memory Pointer
R10	E	Extra/General
R11	C	Counter/General
R12	F	Function Register
R13	G	General Purpose

Register	Name	Primary Use
R14	L	General Purpose
R15	Z	General Purpose

Byte Access

Each register's high and low bytes are individually addressable using H/L suffixes: AH/AL, BH/BL, XH/XL, etc.

32-bit Pairs

Adjacent registers can be combined for certain 32-bit operations using concatenated names: - AB = A (high) + B (low) - CD = C (high) + D (low) - EF, GI, JK, LM, TY, XZ

This is simulated 32 bit access; changing the value of a 32 bit pair will corrupt the underlying 16 bit registers, and so forth. Secondly, access is only marginally faster than 16 bit access; for memory loads, stores and compares it is usually faster to use native 16-bit mode.

24-bit Pointers

Memory addressing uses a bank byte plus 16-bit offset. The naming convention is `[low-byte][offset]`: - BLX = BL (bank) + X (address) - ELM = EL (bank) + M (address) - FLD = FL (bank) + D (address) - GLK = GL (bank) + K (address)

Eight bank registers (BL, EL, FL, GL, IL, JL, LL, TL) each pair with eight address registers (A, C, D, K, M, X, Y, Z), yielding 64 possible 24-bit pointer combinations.

Register Overlap

As with their 32-bit counterparts, 24-bit pointers share components. ELM and ELD both use the EL bank byte. FLD and GLD both use the D address register. Modifying one affects the other – a common source of bugs. Always verify pointer independence when using multiple pointers simultaneously.

Flags Register

The 16-bit FLAGS register contains:

Arithmetic Flags (Byte 1):

- Bit 0: **Z** (Zero) - Result was zero
- Bit 1: **N** (Negative) - Result was negative (bit 15 set)
- Bit 2: **C** (Carry) - Unsigned overflow/borrow
- Bit 3: **V** (Overflow) - Signed overflow
- Bits 4-7: Reserved

Control Flags (Byte 2):

- Bit 8: **H** (Halt) - CPU stopped, waiting for interrupt
- Bit 9: **T** (Trace) - Single-step debugging mode
- Bit 10: **B** (Breakpoint) - Breakpoint mode active
- Bit 11: **E** (Exception) - Sticky error flag
- Bit 12: **P** (Protected) - Protected mode enabled
- Bit 13: **I** (Interrupt) - Interrupt enable/disable
- Bit 14: **S** (Sound) - Sound System Interrupt enable
- Bit 15: Reserved

Layout: `Z N C V - - - - H T B E P I S -`

Appendix 3. Memory Map

Placed in it's own section for easy reference:

- Appendix 3 Memory Map

Appendix 4. Instruction Set Architecture

- Moved to: Appendix 4 Instruction Set Architecture

Appendix 5. Sound System (SD-450)

This has been moved to it's own page for easy reference.

Please see: Appendix 5 Sound System

Appendix 6. Video System

Video Modes

The VC-3 supports both text and graphics modes:

Mode	Resolution	Colors	Description
1	40×25 text	16	Character mode, COLORDORE palette
2	80×25 text	16	80 column mode, CGA 5153 palette
3	320×200	16	Packed pixels (4-bit)
4	*320×200*	*256*	Supercolor mode (8bpp)
5	*256×224*	*256*	N-Type “MODE 5” graphics (not implemented yet)
6	80×25 text	16	DYNATERM 8800 mode (C) 1984 Stellar Dynamics
8	*128×128*	*16*	Low-res “Cart Mode” (not implemented yet)

Text Mode Architecture

Mode 1 (40×25):

- Character buffer: `$F000-$F3E7` (1000 bytes)
- Color buffer: `$F800-$FBE7` (1000 bytes)
- Character ROM: `$E800-$E8FF` (256 characters × 8 bytes)

Color byte format: `(bg_color << 4) | fg_color`

Mode 2 (80×25):

- Same layout, 2000 bytes each
- Scanline doubling for 640×400 output

Graphics Mode Architecture

Mode 3 (320×200×16):

- Framebuffer: Bank 2, `$00000-$07CFF` (32,000 bytes)
- Palette: Bank 2, `$F000-$F02F` (16 colors × 3 bytes RGB)
- Pixel packing: 2 pixels per byte (high/low nibbles)

Pixel addressing:

```
offset = (y × 160) + (x ÷ 2)
address = Bank 2 + offset
```

Mode 4 (256×224×256):

- Framebuffer: Bank 2, `$00000-$DFFFF` (57,344 bytes)
- Palette: Bank 2, `$F000-$F2FF` (256 colors × 3 bytes RGB)
- 1 byte per pixel (256 colors)

Palette Format

Each palette entry is 3 bytes (RGB): `Offset +0: Red (0-255) Offset +1: Green (0-255) Offset +2: Blue (0-255)`

Appendix 7. KERNAL Functions

Moved to it's own section;

- VC-3 System Interrupt Table
- Appendix 7 Kernal Functions

—

SD-8516 Technical Manual - Revision 1.0

All specifications subject to change as quantum resonance research continues.