Author: Tenton Lien

Last Updated: 10/5/2018

The document is still in translation.

The Zen Virtual Machine

Preface

This document introduces the specific implementation of the Zen virtual machine and its related tool chains. If you have already learned C and NASM, it will be much easier of your reading.

Tenton Lien

CONTENT

1 Introduction 1
1.1 Features of the Zen Virtual Machine 1
1.2 Get Started with the Zen Virtual Machine 1
1.3 The ZenVM and JVM Comparison 1
2 The Structure of the Zen Virtual Machine 2
2.1 Data Unit 2
2.2 Data Segment 2
2.3 Instruction Segment 2
2.4 Stack 3
2.5 Virtual Registers 3
2.5.1 Data Registers 3
2.5.2 Pointer Registers 3
2.5.3 Flag Registers 4
2.6 Interrupts 4
3 The Zen Bytecode File Format 5
3.1 Basic Structure 5
3.2 Header 5
3.2.1 Magic Number 5
3.2.2 Bytecode Version 5
3.3 Data Section 6
3.4 Code Section 6
3.4.1 Instruction Format 6
4 The Interpretation Procedure 7
4.1 Loading the bytecode from disk 7
4.1 Verifying the File Format 7
4.2 Handling with Data Section 7
4.2.1 The Initialization of the Stack 7
4.2.2 The Initialization of the Data Segment 7
4.3 Handling with Code Section 7
5 Memory Management of the Zen Virtual Machine 8
5.1 Pointers 8

5.2 Auto Garbage Collection 8
6 Zen Assembly Language 9
6.1 Zen Assembler 9
6.2 ZASM Syntax 9
7 The Zen Virtual Machine Instruction Set 10
7.1 Transfer Instructions 10
7.2 Arithmetic Instructions 10
7.3 Logical Instruction 11
7.4 Control Instructions 11
8 The Zen Virtual Machine Interrupts 14
8.1 System 14
8.2 I/O 14
8.3 File System 14
8.4 Date and Time 14
8.5 Network 15
8.6 Multithread 15
9 The Technical Details of the Zen Tool Chains 16
9.1 The implementation of the Zen Virtual Machine 16
9.1.1 The Implementation of the registers 16
9.1.2 The Decoding of Instructions 16
9.2 The Implementation of the Zen Assembler 16
9.3 The implementation of the Zen Compiler 16
Appendix 1: The Zen Virtual Machine Errors 17

1 Introduction

The Zen Virtual Machine is a high-level language virtual machine designed for Zen programming language. In Zen development procedure, the source code written by developers will be compiled into the bytecode, which can be interpreted by the Zen Virtual Machine.

1.1 Features of the Zen Virtual Machine

The Zen Virtual Machine is a register-based virtual machine.

1.2 Get Started with the Zen Virtual Machine

The Zen Virtual Machine is an open source project with MIT license so you can get the source code and binary output on GitHub. As the Zen Virtual Machine mainly supports Linux platform at present, I suggest you use it on Linux or the Linux subsystem of Windows.

The main program of the virtual machine is under the `\out` directory. To view the version information, just enter this directory and use the command `zvm -version`.

1.3 The ZenVM and JVM Comparison

As we know, Java is a popular programming language, and the design of Java virtual machine is also very successful. Therefore, I list the distinctions between ZenVM and JVM below.

The JVM is a stack-based virtual machine, while the Zen VM is a register-based machine.

The JVM owns the Java bytecode standard so that some programming languages such as Java, Scala, Kotlin can base on the JVM platform. The Zen VM doesn't use the Java bytecode standard but redefines a new bytecode standard to fit with itself well. For academic research, the design of the Zen VM gets closer to the theory system of computer hardware. There are still some traditional concepts such as registers, interrupts and even an assembly language.

Java is a pure object-oriented programming language, while Zen supports several kinds of programming paradigm especially functional programming. Therefore， when loading a bytecode file， the JVM requires a public class containing a main method as the entry of a program, while the ZenVM requires a main function.

As for memory management, the Java Virtual Machine does all the trivial work for developers and removes the pointer which is quite important in C/C++. Nevertheless, the pointer in the Zen VM still exists for complex environment. At this pointer, we can say the Zen VM is more similar with .NET CLR.

2 The Structure of the Zen Virtual Machine

2.1 Data Unit

A data unit in the Zen Virtual Machine stores a single variable's type and value. Its structure in memory can be represented as `struct data_unit {int type; unsigned long long value}`.

The Zen Virtual Machine operates on two kinds of type: primitive types and reference types.

	Zen Data Types	ZenVM Data Types	Size (byte)
Literal Type	bool	byte	1
	char	byte	1
	i8, u8	byte	1
	i16, u16	short	2
	i32, u32	int	4
	i64, u64	long long	8
	f32	float	4
	f64	double	8

2.2 Data Segment

The data segment stores the variables and constant used by program. The constant values are written into data segment during the initialization. The space in data segment for variables is preserved.

The data segment can contain 4,096 data units at most. As all the data in the Zen Virtual Machine is managed in the data segment, in most cases we use a 12-bit long relative memory address instead of a real physical address. The relative memory address of the first data unit is 0, etc.

Due to different life cycles of each variable and constant, the memory space can be reused after the optimization by compiler.

2.3 Instruction Segment

2.4 Stack

The max length of the stack is 65536. Each element uses 64-bit space.

When the stack pointer moves out of the head of stack, an overflow error occurs; when the stack pointer moves out of the end of stack, an underflow error occurred. These two errors are both fatal errors, which will make the virtual machine crash. However, the checking of overflow and underflow error will work during the compilation, so developers needn't pay much attention on these kinds of errors.

2.5 Virtual Registers

In order not to depend too much on the specified hardware platform, The Zen Virtual Machine doesn't directly operate the registers on the processor, But I still remain the concept of registers and implement them on memory. Indeed, they are virtual registers.

Type	Name	Size (bit)	Relative Memory Address
Data Registers	ax	64	0x00
	bx	64	0x01
	cx	64	0x02
	dx	64	0x03
Pointer Registers	bp	64	-
	ip	64	-
	sp	64	-
	di	64	0x04
	si	64	0x05
Flag Registers	sf	1	-
	zf	1	-

2.5.1 Data Registers

The Zen Virtual Machine has 4 data registers: ax, bx, cx, dx. Each register can store 64 bits data and has no difference in usage. Data registers are especially used to store parameters in interrupt operation.

2.5.2 Pointer Registers

The Zen Virtual Machine has 5 pointer registers; they are the base pointer register, instruction pointer register, stack pointer register, destination index register and source index register.

BP (Base Pointer Register), stores the address of the first unit of the data segment. The value in the base pointer register is not changeable while executing.

IP (Instruction Pointer Register), whose default value is zero, stores the sequence number of the current executing instruction. The control instructions (loop, jmp, je, etc.) can change the value in the IP register to help the program jump.

SP (Stack Pointer Register), stores the sequence number of current valid position of the stack. In the PUSH operation the value plus one, while in the POP operation the value minus one.

DI (Destination Index Register) is one of the index registers that are used to fetch any element in an array.

SI (Source Index Register).

2.5.3 Flag Registers

The Zen Virtual Machine has 2 flag registers at present. Each register only has two states, 0 or 1.

SF (Sign Flag). If the result of the arithmetic is positive (or negative), the value is the SF register is 0 (or 1). The execution of CMP instruction influences the register.

ZF (Zero Flag). If the result of the arithmetic is zero, the value in the register is 1; otherwise it will be 0.

2.6 Interrupts

The Zen Virtual Machine makes use of system calls by interrupts. The steps for using interrupts are listed below.

- Store the arguments in the data registers.
- Call the relevant interrupt.
- The result is returned in the data registers.

3 The Zen Bytecode File Format

Zen executable file stores the bytecode for The Zen Virtual Machine. The data in the bytecode uses little-endian representation: the order where less significant byte comes before more significant byte in memory.

3.1 Basic Structure

Sections	Items	Length (byte)
Header	Magic number	4
	Bytecode version	4
Data Section	Stack size	4
	Data segment size	2
	Constant amount	2
	All constants' value	8m
Code Section	Instruction amount	4
	All instructions	4n

3.2 Header

3.2.1 Magic Number

The Magic number identifies the ZEF file format. For example, Java bytecode file's magic number is CA FE BA BE. Similarly, the magic number of Zen executable file is 5A 45 4E 21, the ASCII code of "Zen!".

3.2.2 Bytecode Version

The bytecode version is stored in a four-byte integer. You can get the version number by dividing the value by 100. For example, 01 00 00 00 represents version 0.01， 64 00 00 00 represents version 1.0.

3.3 Data Section

The data section records the details of stack and data segment.

数据段中前 6 个存储单位属于虚拟寄存器的映射区域，因此数据段的最小容量为 6。

3.4 Code Section

Code Section is the core of the bytecode, storing the instructions that can be executed by the Zen Virtual Machine. 该表起始标志为 0xFF，随后跟随指令表的大小，单位为字节，数值本身占用 4 个字节。

3.4.1 Instruction Format

The Zen Virtual Machine uses fixed length instruction set. Each instruction is 32 bits long, containing one or two operands.

3.4.1.1 Opcodes

每条指令中操作码占用前 6 位，最多可表示 64 种操作码。

3.4.1.2 Addressing Modes

The addressing mode follows the opcode，占用 2 位，表示 4 种寻址方式，分别为：

- Immediate Addressing
- Memory Addressing
- Memory-Immediate Addressing
- Memory-Memory Addressing

3.4.1.3 Operands

Every Instruction has a destination operand but not always a source operand.

需要注意的是，在 ZenVM 指令中，允许两个操作数均为内存地址，这一点和 x86 指令有所区别。

4 The Interpretation Procedure

4.1 Loading the bytecode from disk

4.1 Verifying the File Format

检查程序文件是否包含魔数，字节码版本号是否在虚拟机支持的范围内。如果不符合条件，虚拟机将拒绝执行此程序。

4.2 Handling with Data Section

4.2.1 The Initialization of the Stack

读取栈的大小，并为其分配相应的内存空间。同时设置栈指针寄存器 sp 的值为 0，即指向栈的首个位置。

4.2.2 The Initialization of the Data Segment

数据段

4.3 Handling with Code Section

读取到代码段的标志字节 0xFF 之后，读取指令条数，并循环读取指令建立内存中的指令表。

5 Memory Management of the Zen Virtual Machine

5.1 Pointers

I have mentioned that in most cases the Zen Virtual Machine uses relative memory address to visit data because memory access via pointers is unsafe. However, in order to make the Zen Virtual Machine efficient in memory management, I still remain the real pointer.

5.2 Auto Garbage Collection

6 Zen Assembly Language

ZASM, the Zen assembly language, is especially designed for The Zen Virtual Machine.

6.1 Zen Assembler

Zen 汇编工具 (Zen Assembler) 可以将汇编语句翻译成字节码，方便调试。

调用 ZASM 的命令。

```
zasm source.asm
```

此时 ZASM 会读取 source.asm 的内容，将其编译为可执行的字节码并保存到同路径下的 source.zef 中。

调用 The Zen Virtual Machine 运行生成的字节码文件。

```
zen source.zef
```

6.2 ZASM Syntax

The syntax of ZASM mainly refers to NASM (The Netwide Assembly Language).以下是输出 Hello World 字符串的 ZASM 汇编程序。

```
section .data
str char[] 'Hello World!'

section .code
mov ax, 8h    ;输出数据的类型为字符数组
mov bx, str   ;输出数据的内存地址
int 11h   ;调用中断输出结果到屏幕
int 0h   ;结束程序
```

7 The Zen Virtual Machine Instruction Set

7.1 Transfer Instructions

0x00: MOV (move)

Format: MOV DST, SRC

Operation: (DST) ← (SRC)

0x01: XCHG (exchange)

Format: XCHG OPR1, OPR2

Operation: (OPR1) ↔ (OPR2)

0x02: PUSH (push onto the stack)

Format: PUSH SRC

Operation: (SP) ← (SP) + 1

0x03: POP

Format: POP DST

Operation: (SP) ← (SP) – 1

7.2 Arithmetic Instructions

0x10: ADD (add)

Format: ADD DST, SRC

Operation: (DST)←(SRC) + (DST)

0x11: INC (increment)

Format: INC OPR

Operation: (OPR) ←(OPR) + 1

0x12: SUB (subtract)

Format: SUB DST, SRC

Operator: (DST)←(SRC) + (DST)

0x13: DEC (decrement)

Format: DEC OPR

Operator: (OPR) ← (OPR) – 1

0x14: CMP (compare)

Format: CMP ORP1, OPR2

Operator: (OPR1) - (OPR2)

0x15: MUL (multiple)

Format: MUL OPR1, OPR2

Operator: (OPR1) ←(OPR1) * (OPR2)

0x16: DIV (divide)

Format: DIV OPR1, OPR2

Operator: (OPR1) ←(OPR1) / (OPR2)

7.3 Logical Instruction

0x20: AND (and)

Format: AND DST, SRC

Operation: (DST) ← (DST) ∧ (SRC)

7.4 Control Instructions

0x30: INT (interrupt)

Format: INT OPR

Operation:

0x31: LOOP (loop)

Format: LOOP OPR

Operation:

0x32: JMP (jump)

Format: JMP OPR

Operation:

0x33: JE (jump if equal)

Format: JE OPR

Operation:

0x34: JNE (jump if not equal)

Format: JNE OPR

Operation:

0x35: JG (jump if greater)

Format: JG OPR

Operation:

0x36: JL (jump if less)

Format: JL OPR

Operation:

0x37: JGE (jump if greater or equal)

Format: JGE OPR

Operation:

0x38: JLE (jump if less or equal)

Format: JLE OPR

Operation:

8 The Zen Virtual Machine Interrupts

8.1 System

0x01: Exit program

Read: None | Write: None

Interrupt 01H is used to shut down the program. The memory space for stack, data segment and instruction segment will be released at once then the process ends.

8.2 I/O

0x10: Input

Read: AX {1, 2, 3, 4}, BX {1, 2} | Write: DX

The interrupt 10H is used to fetch the data input form the keyboard. The AX register stores the data format you want to get. The BX register stores the …

0x11: Output

Read: AX {1, 2, 3, 4}, BX {1, 2} | Write: DX

The interrupt 11H will print the value in DX register on the console.

8.3 File System

8.4 Date and Time

0x30: System time

The interrupt 30H will return the value in DX.

30H interrupt 用于获取当前的系统时间。当寄存器 ax 的值为 0 时，该中断将包含完整日期和时间的字符串的物理内存地址写入寄存器 bx 中。字符串的格式如下：

Mon Sep 17 20:17:23 2018

8.5 Network

8.6 Multithread

9 The Technical Details of the Zen Tool Chains

9.1 The implementation of the Zen Virtual Machine

The Zen Virtual Machine is written in C (2011 Standard) and compiled by GCC. 因为部分功能需要调用系统函数，因此 ZenVM 目前只支持遵循 POSIX 标准的类 UNIX 操作系统，比如 UNIX, Linux, macOS 等。Windows 用户建议在 WSL (Windows Subsystem for Linux) 中运行本虚拟机。

9.1.1 The Implementation of the registers

为减少虚拟机的跨平台移植的工作量，虚拟机本身并没有使用汇编语言进行编写，无法直接控制 CPU 核内的寄存器。因此虚拟机的寄存器均为虚拟寄存器，其值实际存储在内存中。

9.1.2 The Decoding of Instructions

通过位运算解析出 64 位指令中的操作码、寻址方式及操作数。

opcode = (instruction & 0xFC00000000) >> 58

addressing =

9.2 The Implementation of the Zen Assembler

The Zen Assembler is written in C++.

9.3 The implementation of the Zen Compiler

The Zen Compiler is written in C++.

Appendix 1: The Zen Virtual Machine Errors

Error Code	Details
0	Load bytecode failed...
1	Not a valid Zen executable file.
2	The version of current ZenVM is too low to interpret this executable file. Version higher than xxx needed.
3	Invalid instruction: xxx.
4	
5	
6	
7	