Emulation

An introduction to CPU emulation

By John Hodge [TPG]

Introduction

- Emulation makes one computer or system look like another.
  - Using a wrapper API (wine)
  - Emulating the CPU and providing an API
  - Providing an entire “Virtual Machine”

Emulation Techniques

- Pure Interpreters
  - Step by step emulation of each instruction
- JIT
  - Converts blocks of code into native code
  - Far more bug prone

Emulation Techniques

- Composite
  - JITs simple/common instructions
  - Uses caching on common methods/loops
- AOT
  - Passes a JIT over the code once and stores the result
  - Doesn't work for some cases (annoyingly common cases)

How to make an interpreter

- A quick introduction to writing a CPU interpreter.
  - Step 1: Learn the architecture
  - Step 2: Planning
  - Step 3: Write!

Learn the Architecture

- Read the manuals and look at code for it
  - Also, writing code for the architecture helps you understand it.
- Find the things that could trip you up
  - "Unreal mode" in x86 is an example
- Look for patterns to keep your code clean
  - Example: x86's ALU operations
  - Seeing the pattern reduces the required code segments from sixty-four, to sixteen (eight for the operations, eight for the arguments)

Plan the structure

- Identify the CPU's registers and organise them.
- Think of what state will be needed during decoding
- Place this into a skeleton structure

Write!

- Create a simple, almost pseudo-code, structure
- Populate it with the patterns to decode the instructions
- Keep the main decoder function simple by abstracting common sections of code

Example: RME

- Real Mode Emulator
  - Emulates an x86 series processor that is in 16-bit mode.
  - Designed for use in operating system kernels.

Overview

- Machine State Structure
  - 8 GP Registers, 4 Segment Registers, IP and FLAGS
  - Decoder variables

Overview

- Error handling
  - All functions return zero on success
  - “ret = func(); if(ret) return ret;”
- Memory access abstraction
  - ReadMem(CPU, Segment, Offset, size, dest)

Instruction Formats

- Opcode
  - One or more bytes
    - 0x0F starts a multi-byte sequence
- Mod R/M Byte
  - Extra byte to encode source and destination, or extra opcode information
  - 2:3:3 – Mod : Reg : Mem
  - Mod selects the offset of the memory address, or marks Mem as a register.

Instruction Formats

- Mod/RM - ALU
  - Memory on Register, Register on Memory and Immediate on Accumulator (AX)
    - These three encode the operation in the opcode and use both Reg and Mem as points.
  - Immediate on Memory
    - Operation is encoded in the Reg field.

Instruction Formats

- Directly Encoded
  - PUSH and POP
    - Operands are encoded in the opcode
  - Conditional Jumps
  - STOS, LODS and MOVS
    - Implicitly uses a combo ES:[DI], DS:[SI] or AX (Depending on the instruction)

Catches

- Operand and Address overrides
  - Makes the code more complex
- Flag Values
  - Setting flag values correctly requires care

Tricks

- Adrian will hate me for some of these :D
- Macros are your friend
  - But make sure to use them wisely
- Inline functions
  - Where you start getting big macros, put them in an inline function.