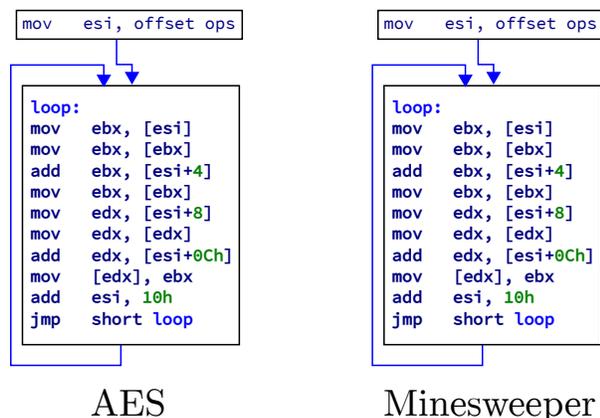


5 A Crisis of Existential Import; or, Putting the VM in M/o/Vfuscator

by Chris Domas



A programmer writes code. That is his purpose: to define the sequence of instructions that must be carried out to perform a desired action. Without code, he serves no purpose, fulfills no need. What then would be the effect on our existential selves if we found that all code was the same, that every program could be written and executed exactly as every other? What if the net result of our century of work was precisely ... nothing?

Here, we demonstrate that all programs, on all architectures,³² can be reduced to the same instruction stream; that is, the sequence of instructions executed by the processor can be made identical for every program. On careful analysis, it is necessary to observe that this is subtly distinct from prior classes of research. In an interpreter, we might say that the same instructions (those that compose the VM) can execute multiple programs, and this is correct; however, in an interpreter the sequence of the instructions executed by the processor changes depending on the program being executed—that is, the instruction streams differ. Alternatively, we note that it has been shown that the x86 MMU is itself Turing-complete, allowing a program to run with no instructions at all.³³

In this sense, on x86, we could argue that any program, compiled appropriately, could be reduced to *no* instructions—thereby inducing an equivalence in their instruction streams. However, this peculiar-

ity is unique to x86, and it could be argued that the MMU is then performing the calculations, even if the processor core is not—different calculations are being performed for different programs, they are just being performed “elsewhere.”

Instead, we demonstrate that all programs, on any architecture, could be simplified to a single, universal instruction stream, in which the computations performed are precisely equivalent for every program—if we look only at the instructions, rather than their data.

In our proof of concept, we will illustrate reducing any C program to the same instruction stream on the x86 architecture. It should be straightforward to understand the adaptation to other languages and architectures.

We begin the reduction with a rather ridiculous tool called the M/o/Vfuscator. The M/o/Vfuscator allows us to compile any C program into only x86 `mov` instructions. That is not to say the instructions are all the same—the registers, operands, addressing modes, and access sizes vary depending on the program—but the instructions are all of the `mov` variety. What would be the point of such a thing? Nothing at all, but it does provide a useful beginning for us—by compiling programs into only `mov` instructions, we greatly simplify the instruction stream, making further reduction feasible. The `mov` instructions are executed in a continuous loop, and compiling a program³⁴ produces an instruction stream as follows:

```
1 start:
2 mov ...
3 mov ...
4 mov ...
5 ...
6 mov ...
7 mov ...
8 mov ...
9 jmp start
```

³²Perhaps it is necessary to specify, Turing-complete architecture.

³³See *The Page-Fault Weird Machine: Lessons in Instruction-less Computation* by Julian Bangert et al., USENIX WOOT’13 or the 29C3 talk “The Page Fault Liberation Army or Gained in Translation” by Bangert & Bratus

³⁴`movcc -Wf-no-mov-loop program.c -o program`

Fine Fun For Winter Nights



Dull evenings are unknown where there's a *New Mirroscope*. Simply hang a sheet, darken the room and have a picture show of your own. Guessing games, puzzles, illustrated songs—there are hundreds of ways to entertain yourself and friends with

The New Mirroscope

The 1916 Models have improved lenses and lighting system and exclusive adjustable card holder. Prices range from \$2.50 to \$25.00. Six sizes. Made for electricity, acetylene and natural or artificial gas. Every *New Mirroscope* fully guaranteed.

FREE: The *New Mirroscope* Booklet of shows and entertainments. Send for it.

You can buy the *New Mirroscope* at most department and toy stores, at many photo supply and hardware stores. Ask for the *New Mirroscope* and look



for the name. If no dealer is near you, we will ship direct on receipt of price.

The
Mirroscope Co.
16902 Waterloo Road
Cleveland, O.

But our mov instructions are of all varieties—from simple `mov eax, edx` to complex `mov dl, [esi+4*ecx+0x19afc09]`, and everything in between. Many architectures will not support such complex addressing modes (in any instruction), so we further simplify the instruction stream to produce a uniform variety of movs. Our immediate goal is to convert the diverse x86 movs to a simple, 4-byte, indexed addressing varieties, using as few registers as possible. This will simplify the instruction stream for further processing and mimic the simple load and store operations found on RISC type architectures. As an example, let us assume `0x10000` is a 4-byte scratch location, and `esi` is kept at 0. Then

```
1 mov eax, edx
```

can be converted to

```
1 mov [0x10000+esi], edx
  mov eax, [0x10000+esi]
```

We have replaced the register-to-register mov variety with a standard 4-byte indexed memory read and write. Similarly, if we pad our data so that an oversized memory read will not fault, and pad our scratch space to allow writes to spill, then

```
mov al, [0x20000]
```

can be rewritten

```
1 mov [0x10000+esi], eax
  mov edi, [0x20000-3+esi]
3 mov [0x10000-3+esi], edi
  mov eax, [0x10000+esi]
```

For more complex addressing forms, such as `mov dx, [eax+4*ebx+0xdeadbeef]`, we break out the extra bit shift and addition using the same technique the M/o/Vfuscator uses—a series of movs to perform the shift and sum, allowing us to accumulate (in the example) `eax+4*ebx` into a single register, so that the mov can be reduced back to an indexed addressing `eax+0xdeadbeef`.

With such transforms, we are able to rewrite our diverse-mov program so that all reads are of the form `mov esi/edi, [base + esi/edi]` and all writes of the form `mov [base + esi/edi], esi/edi`, where

base is some fixed address. By inserting dummy reads and writes, we further homogenize the instruction stream so that it consists only of alternating reads and writes. Our program now appears as (for example):

```

start:
2 ...
mov esi, [0x149823 + edi]
4 mov [0x9fba09 + esi], esi
mov edi, [0x401ab5 + edi]
6 mov [0x3719ff + esi], edi
...
8 jmp start

```

The only variation is in the choice of register and the base address in each instruction. This simplification in the instruction stream now allows us to more easily apply additional transforms to the code. In this case, it enables writing a non-branching mov interpreter. We first envision each mov as accessing “virtual,” memory-based registers, rather than CPU registers. This allows us to treat registers as simple addresses, rather than writing logic to select between different registers. In this sense, the program is now

```

start:
2 ...
MOVE [_esi], [0x149823 + [_edi]]
4 MOVE [0x9fba09 + [_esi]], [_esi]
MOVE [_edi], [0x401ab5 + [_edi]]
6 MOVE [0x3719ff + [_esi]], [_edi]
...
8 jmp start

```

where `_esi` and `_edi` are labels on 4-byte memory locations, and `MOVE` is a pseudo-instruction, capable of accessing multiple memory addresses. With the freedom of the pseudo-instruction `MOVE`, we can simplify all instructions to have the exact same form:

```

start:
2 ...
MOVE [0 + [_esi]], [0x149823 + [_edi]]
4 MOVE [0x9fba09 + [_esi]], [0 + [_esi]]
MOVE [0 + [_edi]], [0x401ab5 + [_edi]]
6 MOVE [0x3719ff + [_esi]], [0 + [_edi]]
...
8 jmp start

```

We can now define each `MOVE` by its tuple of memory addresses:

```

{0, _esi, 0x149823, _edi}
2 {0x9fba09, _esi, 0, _esi}
{0, _edi, 0x401ab5, _edi}
4 {0x3719ff, _esi, 0, _edi}

```

and write this as a list of operands:

```

operands:
2 .long 0, _esi, 0x149823, _edi
.long 0x9fba09, _esi, 0, _esi
4 .long 0, _edi, 0x401ab5, _edi
.long 0x3719ff, _esi, 0, _edi

```

We now write an interpreter for our pseudo-mov. Let us assume the physical `esi` register now holds the address of a tuple to execute:

```

1 ; a pseudo-move
3 ; Read the data from the source.
mov ebx, [esi+0] ; Read the address of the
5 ; virtual index register.
mov ebx, [ebx] ; Read the virtual index
7 ; register.
add ebx, [esi+4] ; Add the offset and
9 ; index registers to
; compute a source
11 ; address.
mov ebx, [ebx] ; Read the data from the
13 ; computed address.
15 ; Write the data to the destination.
mov edx, [esi+8] ; Read the address of the
17 ; virtual index register.
mov edx, [edx] ; Read the virtual index
19 ; register.
add edx, [esi+12] ; Add the offset and
21 ; index registers to
; compute a destination
23 ; address.
mov [edx], ebx ; Write the data to the
25 ; destination address.

```

Leedawl COMPASS Make Your Boy a Leader
 Give him a Leedawl Compass for Christmas and let him lead "the boys" through the woods, over a trail or on a tramp.
It's the only Guaranteed Jeweled Compass for \$1.00.
If your dealer does not have them, write us for folder C-12.
Taylor Instrument Companies, Rochester, N. Y.
 Makers of Scientific Instruments of Superiority.

Finally, we execute this single MOVE interpreter in an infinite loop. To each tuple in the operand list, we append the address of the next tuple to execute, so that `esi` (the tuple pointer) can be loaded with the address of the next tuple at the end of each transfer iteration. This creates the final system:

```

1 mov esi, operands
  loop:
3 mov ebx, [esi+0]
  mov ebx, [ebx]
5 add ebx, [esi+4]
  mov ebx, [ebx]
7 mov edx, [esi+8]
  mov edx, [edx]
9 add edx, [esi+12]
  mov [edx], ebx
11 mov esi, [esi+16]
   jmp loop

```

The operand list is generated by the compiler, and the single universal program appended to it. With this, we can compile all C programs down to this exact instruction stream. The instructions are simple, permitting easy adaptation to other architectures. There are no branches in the code, so the precise sequence of instructions executed by the processor is the same for all programs. The logic of the program is effectively distilled to a list of memory addresses, unceremoniously processed by a mundane, endless data transfer loop.

So, what does this mean for us? Of course, not so much. It is true, all “code” can be made equivalent, and if our job is to code, then our job is not so interesting. But the essence of our program remains—it had just been removed from the processor, diffused instead into a list of memory addresses. So rather, I suppose, that when all logic is distilled to nothing, and execution has lost all meaning—well, then, a programmer’s job is no longer to “code,” but rather to “data!”

This project, and the proof of concept reducing compiler, can be found at Github³⁵ and as an attachment.³⁶ The full code elaborates on the process shown here, to allow linking reduced and non-reduced code. Examples of AES and Minesweeper running with identical instructions are included.

³⁵[git clone https://github.com/xoreaxeaxeax/reducto](https://github.com/xoreaxeaxeax/reducto)

³⁶[unzip pocorgtfo12.pdf reducto.tgz](#)

Helps to Spring Fun

The Second BOYS' BOOK OF MODEL AEROPLANES

By Francis Arnold Collins

The book of books for every lad, and every grown-up too, who has been caught in the fascination of model aeroplane experimentation, covering up to date the science and sport of model aeroplane building and flying, both in this country and abroad.

There are detailed instructions for building fifteen of the newest models, with a special chapter devoted to parlor aviation, full instructions for building small paper gliders, and rules for conducting model aeroplane contests.

The illustrations are from interesting photographs and helpful working drawings of over one hundred new models.

The price, \$1.20 net, postage 11 cents

The Author's Earlier Book THE BOYS' BOOK OF MODEL AEROPLANES

It tells just how to build “a glider,” a motor, monoplane and biplane models, and how to meet and remedy common faults—all so simply and clearly that any lad can get results. The story of the history and development of aviation is told so accurately and vividly that it cannot fail to interest and inform young and old.

Many helpful illustrations

The price, \$1.20 net, postage 14 cents

All booksellers, or send direct to the
publishers :

THE CENTURY CO.