15:11  X86 is Turing-Complete without Data Fetches

by Chris Domas

One might expect that to compute, we must first somehow access data. Even the most primitive Turing tarpits generally provide some type of load and store operation. It may come as a surprise, then, that most modern architectures are Turing-complete without reading data at all!

We begin with the (somewhat uninspiring) observation that the effect of any traditional data fetch can be accomplished with a pure instruction fetch instead.

That fetch in pure code would be a move sourced from an immediate value.

With this, let us then model memory as an array of “fetch cells,” which load data through instruction fetches alone.

So to read a memory cell, without a data fetch, we’ll jmp to these cells after saving a return address. By using a jmp, rather than a traditional function call, we can avoid the indirect data fetches from the stack that occur during a ret.

A data write, then, could simply modify the immediate used in the read instruction.

Of course, for a proof of concept, we should actually compute something, without reading data. As is typical in this situation, the BrainFuck language is an ideal candidate for implementation — our fetch cells can be easily adapted to fit the BF memory model.

Reads from the BF memory space are performed through a jmp to the BF data cell, which loads an immediate, and jumps back. Writes to the BF memory space are executed as self modifying code, overwriting the immediate value loaded by the data cell. To satisfy our “no data fetch” requirement, we should implement the BrainFuck interpreter without a stack. The I/O BF instructions (., and,), which use an int 0x80, will, at some point, use data reads of course, but this is merely a result of the Linux implementation of I/O.

First, let us create some macros to help with the simulated data fetches:

Next, we’ll compose the skeleton of a basic BF interpreter:

Then, we’ll implement each BF instruction without data fetches.
Finally, let us construct the unusual memory tape and system state. In its data-fetchless form, it looks like this.

```
fetch_ip:
    db 0xb8
ip:
    dd 0
jmp esi
fetch_dp:
    db 0xb8
mov eax, xxxxxxxx
mov ebx, 0
int 0x80
```

For brevity, we’ve omitted the I/O functionality from this description, but the complete interpreter source code is available.43

And behold! A functioning Turing machine on x86, capable of execution without ever touching the data read pipeline. Practical applications are nonexistent.

43git clone https://github.com/xoreaxeaxeax/tiresias || unzip pocorgtfo15.pdf tiresias.zip