The Balanced Ternary Machines of Soviet Russia

Andrew Buntine Nov 16, 2016

ternary history computer-science

> "Perhaps the prettiest number system of all is the balanced ternary notation" - Donald E. Knuth, The Art of Computer Programming, Vol. 2.

It's pretty common knowledge that computers store and operate on data using the binary number system. One of the main reasons for this can be found in the circuitry of modern computers, which are made up of billions of easily mass-producible transistors and capacitors that can together represent two states: **high voltage** (`1`) and **low voltage** (`0`).

Such a design is so ubiquitous nowadays that it's hard to imagine that computers could operate in any other way. But, in Soviet Russia during the 1950s, they did. Enter **Setun**, a balanced ternary computer developed in 1958 by a small team led by Nikolay Brusentsov at the Moscow State University.

OK, so before we continue talking any more about Brusentsov and Setun, let me explain a little bit about balanced ternary.

Balanced Ternary

Ternary, or base-3, is a number system in which there are *three* possible values: `0`, `1` and `2`. In balanced ternary, these three possibilities are `-1`, `0` and `+1`; often simplified to `-`, `0` and `+`, respectively.

So, in its balanced form, we can think of the ternary values as being "balanced" around the mid-point of `0`. The same rules apply to ternary as to any other numeral system: The right-most symbol, `R`, has it's own value and each successive symbol has it's value multiplied by the base, `B`, raised to the power of it's distance, `D` from `R`.

Uhh, right, maybe I'll just give you an example. Let's encode `114`:

```
+++-0 = (1 * 3^4) + (1 * 3^3) + (1 * 3^2) + (-1 * 3^1) + 0
      = 81 + 27 + 9 + -3
      = 114
```

And in binary (base-2):

```
1110010 = (1 * 2^6) + (1 * 2^5) + (1 * 2^4) + 0 + 0 + (1 * 2^1) + 0
        = 64 + 32 + 16 + 2
        = 114
```

And, just to be sure, here are the same rules applied in decimal (base-10):

```
114 = (1 * 10^2) + (1 * 10^1) + (4 * 10^0)
    = 100 + 10 + 4
    = 114
```

Cool?

Now, what if we wanted to represent `-114`? Well, in binary and decimal, we need to introduce a new symbol: the sign. In the main memory of a binary computer, this would either translate to storing a leading bit to specify the sign or significantly reduce the amount of numbers we can represent [1]. This is the reason we speak about `signed` and `unsigned` integers in programming languages.

But in balanced ternary, as we're about to discover, to represent the inverse of a number we simply have to swap all `+`'s with `-`'s and vice versa. We don't need any extra information to represent the sign!

Check it out:

```
---+0 = (-1 * 3^4) + (-1 * 3^3) + (-1 * 3^2) + (1 * 3^1) + 0
      = -81 + -27 + -9 + 3
      = -114
```

In a moment we will see that this property (and a couple others) of balanced ternary give us some very interesting computational efficiencies. But, for now, let's go back to talking about the Setun computer.

The Birth of Setun

The late 50's were an exciting time in computers: Nathaniel Rochester and his team at IBM had recently developed the first mass-produced, stored-program (a.k.a "modern") computer - the IBM 701. John Backus and his team had invented FORTRAN, the first high-level programming language that was seeing widespread use. And, perhaps most important of all, the first fully-transistorized computers, such as the TX-0 and the Philco Transac S-2000, were starting to be developed. The path was set for the development of the binary computing machines that have dominated the market ever since.

But that was North America.

At the same time in Russia, a group of mathematicians and engineers under the guidance of Nikolay Brusentsov and his colleague Sergei Sobolev were devising different ways in which computers could operate [2]. Brusentsov and co. surveyed the array of computers and technological advancements emerging from the West and comprehended their use of transistors to represent binary data. But, let's remember, this was Russia - transistors were not readily available behind the Iron Curtain. And vacuum tubes sucked as much in Russia as they did in the West!

So Brusentsov devised an element from miniature ferrite cores and semiconductor diodes that was able to act as a controlled current transformer, which proved to be an effective base for implementing ternary logic [3]. It was found that these elements, compared with their binary counterparts, provided more speed and reliability and required less power to operate.

The team of ten literally built Setun from scratch, working in a small room filled with laboratory tables (which they also built!). Every morning, the team members began their day by assembling five basic machine elements. They started with a ferrite core and, using an ordinary sewing needle, wound 52 coils of wire onto it. The cores were then handed off to technicians who completed the assembly process and mounted them onto blocks.

Ternary logic was implemented by combining two of these ferrite elements and wiring them in such a way that they could represent three stable states. This approach was successful but did nothing to reduce the number of elements required as, in reality, those two ferrite cores could potentially represent two binary bits, which represents more information (`2^2`) than one ternary "trit" (`3^1`). Alas, at least their power consumption was reduced!

Setun operated on numbers of up to `18` trits, meaning one could represent anything between `-387,420,489` and `387,420,489`. A binary computer, on the other hand, would need at least `29` bits to reach this capacity.

Overall, development of Setun lasted two years - although it was operational after just ten days of testing, which was unheard of at the time. In total, only about 50 machines were produced. And even though Setun machines proved capable of operating successfully for years-on-end in Russia's extreme climate, the entire project was racked with controversy.

This was primarily due to the fact the manufacturing plant was unable to justify the mass-production of what they deemed as a cheap intellectual endeavor and the "fruit of university fantasy". I think it's safe to assume that Russia was simply not ready to comprehend the eventual importance of computing machines. Ultimately, the Setun machines were replaced with binary counterparts that were able to calculate with similar efficiency but at more than twice the cost of operation!

But why Ternary?

As I demonstrated briefly above, in Ternary, there is no need to store a leading bit, ahem trit, to indicate sign. Therefore, there is no concept of `signed` or `unsigned` integers - everything is just an integer. So subtraction is achieved by simply inverting the operand and applying addition (which is implemented similarly to binary machines). This plus-minus consistency is also able to cut down on the number of carries required in multiplication operations.

Another useful trait of balanced ternary (or any balanced numeral system for that matter!) is the fact that one can implement rounding on floating-point numbers by blatant truncation, which allows for a simplified implementation of division. This is because of the way balanced ternary represents the fractional portion of real numbers.

Let me give you a simple example. Encoding `0.2` looks like this:

```
0.+--+ = 0 + (1 * (3^-1)) + (-1 * (3^-2)) + (-1 * (3^-3)) + (1 * (3^-4))
       = 0.33 + -0.11 + -0.03 + 0.01
       = 0.2
```

And to encode `0.8` we must start with a `+` in the most significant digit and

then simply inverse the fractional portion (e.g `1 + -0.2`):

```
+.-++- = 1 + (-1 * (3^-1)) + (1 * (3^-2)) + (1 * (3^-3)) + (-1 * (3^-4))
       = 1 + -0.33 + 0.11 + 0.03 + -0.01
       = 0.8
```

Above we can see that truncating the trits to the right of the radix point is equivalent to rounding: `0.2` becomes `0` whereas `0.8` becomes `1`. Cool!

Programming with trits and trytes!

OK, back to Setun one last time. In the late 60's, Brusentsov developed a more modern machine called "Setun-70" that embodied it's *ternarity* more directly. The "Tryte" was introduced, which accounted for `6` trits (roughly `9.5` bits). The Setun-70 computer was a stack machine, and so rather than having machine instructions that specifically named registers for input and output, all operations worked on two stacks - one for operands (input) and one for return values (output). In order to accommodate this design, machine instructions were written in reverse Polish notation (a.k.a postfix).

In the late 70's, Brusentsov and some of his students developed a programming language for the Setun-70 computer called the Dialog System for Structured Programming (DSSP). In my research [4], I have been able to discover that it is a stack-based language (no surprise there) similar to Forth that uses reverse Polish notation. This allows one to write programs in a relatively high-level language yet still feel "close to the metal". So close, in fact, that the authors had the following to say:

> DSSP was not invented. It was found. That is why DSSP has not versions, but only extensions.

Take the following DSSP program that sums a bunch of numbers:

```
1 2 3 4  DEEP 1- DO +
```

Let's try to break it down. In the first column we have the instruction, in the second we have the machine state after execution (the operand stack), and in the third I've provided an explanation:

```
1    [1]           Push 1 to the stack.
```

```
2    [2 1]          Push 2 to the stack.
3    [3 2 1]        Push 3 to the stack.
4    [4 3 2 1]      Push 4 to the stack.
DEEP [4 4 3 2 1]    Push "the depth of the stack" (4) to the stack.
1-   [-1 4 4 3 2 1] Push -1 to the stack.
DO   [4 3 2 1]      Initiate LOOP, pop top two items from stack. To control
                    the loop, the first item is applied to the second item until
                    it reaches 0.
+    []             Apply the "+" operation repeatedly until the LOOP terminates,
                    each time popping the top item from the operand stack stack,
                    applying + and then pushing the output to the return stack.
```

At the end of execution, the operand stack will be empty and the return stack will contain `[10]`.

You can read a bit more about DSSP over at the website of Ivan Tikhonov (original authors Sidorov S.A. and Shumakov M.N.).

The future

The development of balanced ternary machines has all but faded into a small footnote in the annals of computer history. And whilst research into memory cells able to efficiently represent three distinct states has been relatively minimal, there have been some efforts in this area.

Particularly, researchers in Japan in the late 90's described the possibility of using a Josephson Junction to implement ternary logic. This could be achieved by circulating superconducting currents, either *clockwise*, *counterclockwise*, or *off*. They found that this gave the memory cells a "capability of high speed computation, low power consumption and very simple construction with less number of elements due to the ternary operation".

But, for the moment at least, I don't think that you'll be hearing much more about balanced ternary computers in the near future. Nor do I think DSSP is going to be the next big thing in programming language fanboyism. But I do believe that much wisdom can be sought by looking into the past [5].

Thanks for making it this far. Corrections and feedback is very welcome. You can also read more here, here and here. :)

1. This depends on how the machine in question represents numbers. Two's complement is a common notation that would allow one to represent `-((2^n) / 2)` to `((2^n) / 2) - 1` in `n` bits.
2. Although Setun was the first electronic device to operate on balanced

ternary, it's worth mentioning that the idea of using balanced ternary in computing devices was first popularized over 100 years before. In 1840, Thomas Fowler built a calculating machine entirely from wood that operated on data in the balanced ternary notation.

3. A far more accurate description can be found on the Russian Computer Museum website.
4. Reference material for DSSP in English is not widely available so I'd like to provide a warning here that my knowledge is very limited and *just may* contain some guesswork.
5. My own contribution can be seen at computerpionee.rs.
6. Cover image, originally from the Moscow University Supercomputing Center, depicts one of the Setun machines in operation.

Andrew Buntine

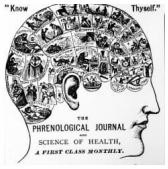
Conjurer of computational spells and trickery.

buntine bunts.io

You may also like

How to write an object oriented program that doesn't suck

Naveen on November 17, 2016



What Can Phrenology Teach Us About Imposter Syndrome?

Ben Halpern on May 23, 2016

The Practical Dev - copyright 2016 Follow @ThePracticalDev on the Twitters