Ideas on a continuous computer architecture

So the initial idea was for a minimal-instruction set computer architecture. I've done a NAND-based machine before, but shifting is a difficult operation. One solution is to memory-map a shift unit, and simply write data to that unit and read the shifted output from the output location. This leads to an alternate idea.

Consider an architecture based purely on memory transfer operations. Programs take the form:

$src_1, dst_1, srd_2, dst_2, src_3, dst_3, ...$

Each pair represents a single memory transfer, from src_n to dst_n. The machine begins with the first pair at addresses 0 and 1; performs the transfer; and then proceeds to addresses 2 and 3; and so on.

Operations such as addition, logical operations, shifts, and so on are memory-mapped into the address space, so, for example, to add the numbers at addresses 100 and 200 and save the result in address 200, one would use the instructions:

$100, ADD_A, 200, ADD_B, ADD_SUM, 200$

where ADD_A, ADD_B and ADD_SUM are the locations in memory of the two inputs and the output to the addition unit.

Additionally, mapping the program counter (PC) into the address space allows branches to occur under program control. And, of course, the program itself is mapped into the same address space, so that instructions can modify other instructions.

I considered writing a simulator for this, and wondered about how to wire-in the various hardware modules. I decided a simple text-file could list standard hardware devices and the addresses of their I/O lines. So in this case, in addition to writing the code for the program itself, one could also specify the hardware-map files. Now this starts to feel more like HW/SW co-compilation, which is a pretty fun area to explore.

But the PC seems out of place: it requires some magic piece of hardware that increments by default, whereas everything else in the system is just doing successive indirect reads and writes.

But there's a bigger issue for me. I've been thinking about this idea of a sequential computer, and how we count operations: it works well with the way we count events, or moments in time, as it forces countability on the set of operations the computer performs, even over an infinite period of time. But I believe it goes against the funndamental nature of the universe, where successions of events form an uncountable set. I also wonder if this

is a fundamental limitation of sequential computers in trying to model or undestand the universe...

So how to get rid of the PC?

How about, instead of performing transfers sequentially, if we perform them *all at the same time*? No more PC needed: we simply read all the sources, and copy to all the destinations, simultaneously. No problem, this is doable. But when then? Clearly we could repeat this process ("a second time"), but right there is the issue again: we're forcing a countable sequence of events, which may never be able to capture the behavior of a universe where the sequence of events - even over a finite period (say one second) - form an uncountable set.

So we could have these transfer occur continuously. src_1 could transfer to dst_1 and, as soon as it finishes, it begins again. But there's still a countable sequence there. So what if the transfers occcur gradually, rather than all at once? Thing of the contents of *src* and *dst* as fluids in containers, which slowly drains from *src* into *dst*. Hardware units occur as the inputs arrive and change, and the outputs fill or drain at the same time. But there's still a notion of "start" and "finish," which leads again to countability. So maybe we need the transfers to change the instructions too, as they are "executing," so that there isn't a start and an end, but a continuum of execution.

While we're at this, what if we go ahead and eliminate the discrete set of instructions, and instead have a single continuous *src*/*dst* pair, which affects the transfers among a continuous memory (no more discrete memory addresses); and, finally, let the program itself morph while everything else is morphing and flowing.

What would this look like? I'm not sure; but some sort of fluidic assembly that is self-affecting, with the option of certain parts of the fluid interacting with external hardware systems...