**Discrete CPUs**

Isaac Hair

November 2020

Objective: Highlight your scientific progress and the interesting/difficult parts about the construction.

**Abstract**

ASDF make this short and brag.

**Introduction**

Brag a lot about the computer here. This is likely going to be read first, so it needs to be interesting and also a huge flex. Don’t assume that the computer speaks for itself, because it doesn’t. Without anchoring to form an opinion, this project can easily seem underwhelming even though it took a massive amount of creative and technical insight.

This bad boy should cover most of my journey through computer hardware and software design from October 2016 through November 2020. I guess a good place to start would be a description of the project. DESCRIPTION blah blah blah wanted to make discrete computer blah blah realized that you can’t even define something as a computer because there is no bright line etc. Objective is to create a computer that is functional but extremely low transistor count. Useful in applications where power consumption is critical, or just for low cost stuff. Like, you could put 2^24 of the map25.2 on a typical dye these days (with that sexy 5 nm technology). The one I created has 254,000 nm feature size and 5,000,000 nm transistor size, which is balls horrible. Also increase clock from 456 kHz to 5 GHz. Ok this needs to be moved to the other section. I will include the “useful” discoveries and inventions first, then the other crap. Note that notes from first few years were very lacking because everything was just burned into my memory (inventing everything from scratch will do that). I have included as much detail as possible in this thesis for all years. I tried to keep this purely based on my own inventive process, but of course I was polluted by outside sources. This is why I actually know what CISC, RISC, ALU, etc mean.

There are enough experiments here to write a dozen research papers at least. Should find a way to break it up and start publishing as useful stuff.

Many people underestimate the value of analyzing computation from a very low level. Research into the pure mathematics of computers and the cutting edge of software exists, but comparatively little attention is given to the nexus between the two: small scale computers and efficient design in both software and hardware. Closer analysis into this technology provides insights into the very construction of computers. The construction of the computer is literally the foundation for all software, so any application of the computer relies on intelligent hardware design. Furthermore, insights into hardware design provide insights into various programming paradigms. Example: procedural vs functional vs object oriented.

**Applications**

I’m just going to get this out of the way: this architecture is not meant to be very useful in the real world; it is mostly just an exploration into the nature of computing. However, I believe that offloading the work of the ALU into the rest of the system can prove useful in scenarios where all you need is for the cpu to be tiny.

This cpu architecture is actually extremely useful in low power applications. Also, if you just need to get something done without being too complicated. Also, extreme parallelism. Also, if just need something to direct the activity of peripherals (eg you offload the heavy lifting to the gpu, etc). This cpu is basically an exploration into the simplest possible architecture you can have that still does things. This architecture can just be dropped onto just about any chip, even a monolithic chip, wherever you need something to do something. You can use a diode array instead of eeprom for these sorts of applications.

This project also provided mathematical insights. For example, I hope that I have at least been able to further humanity’s understanding of the P vs NP problem a little bit through my own exploration into the nature of computing. Spoiler alert: the closest I came to defining a computer was to realize that the task is literally impossible. Actually, proving this would be cool. Hmmmmmm.

**Conjectures**

Conjecture A: Every computable program can be described as some combination of the four fundamental functions.

Conjecture B: Computational speed can be described as the speed at which the four basic functions are performed.

Conjecture C: The “effeciency” of a program can be described as the big O notation for increase in a) RAM b) Program length and c) time as a function of word size.

Conjecture D: The size of a program in ROM is a function of the computer’s physical architecture and the architecture of the machine code.

Conjecture E: “Or” logic on its own is enough to create all logic gates.

Conjecture F: Mask writes to registers is just as effective as program complexity increases as an entire ALU would be.

Conjecture G: Mask writes with rotations up and down is ONLY PROPORTIONALLY slower than a computer with an ALU.

Conjecture H: Increasing the complexity of the ALU or computer architecture in general only increases speed of defined functions; there will always be functions that must be explicitly defined down to a bitwise level.

Conjecture I: It is fundamentally impossible to define a “computer.”

Conjecture J: 100% ROM packing is impossible. (This might just apply to map 25.2).

Conjecture K: No computer will be capable of performing every function with maximum efficiency (maybe this should be an fpga???????) because the complexity blows up and the complexity of the computer is constant.

Conjecture L: Two layers is always enough on a pcb.

**Algorithms**

Algorithm A: Path indexing. For vsp 8

Algorithm B: Nested return matrix. For vsp 8

Algorithm C: Function stack and heap definitions. For vsp 8

Algorithm D: 3 types of addition for map 25.2

Algorithm E: Edge detect predecessor

**Table of Contents**

1. Theory of computational completeness
2. Every cpu design paradigm (25 total)
3. Virtual machine
4. Applications
5. Path Indexing
6. Successor function
7. Three methods of addition
8. Multiplication
9. Shit notetaking skills
10. Voltage levels
11. Reliability
12. Production methods
13. Logic type
14. Speed of processing
15. RAM and ROM
16. DPN evolution (eg control unit)
17. Instruction scheduling
18. BIOS and boot up procedures
19. WAVEFORMS DUDE
20. First manifestation (that single bit thing with actual alu)
21. Second manifestation (i think vsbp-7)
22. Third manifestation (VSP-8 aka VSBP-8)
23. Fourth manifestation (map25.2)
24. Comparison to other systems, eg my shitty wikipedia research
25. Positional Entropy
26. Instruction shape and wasted memory
27. Displays
28. Compiler and assembler
29. Corth
30. “Fake” C for the mandelbrot program
31. VSBP paradigm
32. MAP paradigm
33. CPU+ALU paradigm
34. Function stack
35. Ram heap, stack
36. Use of virtual machine to meet requirements
37. System “seizures”
38. Micro functions
39. Four basic functions and conjecture
40. Easy eda file manipulation
41. Easyeda in general
42. .R protocol evolution
43. Phase noise problems (eg collapse on vsp8 >800kHz)
44. Signal death if connection is even slightly flaky, and 10k connections easily
45. Not gate oscillator
46. Quartz crystal oscillator
47. Mathematics for frq from switches
48. Rigorous proof that 2 layers is enough for ANY layout
49. Other rigorous and sexy proofs
50. Gls29ee010 struggles and arduino uno being a bitch with digitalWrite()
51. Note taking strategies
52. Bit ordering eg little endian, big endian, random
53. Rom paging and messiness
54. Phase generator
55. Pcb design
56. Oscilloscopes (rigol and the p2)
57. Usb protocol for arduino
58. Current draw measurements
59. Lcds and displays (might have already listed this)
60. Tri state output
61. Flakiness constant
62. If else scheduling
63. Bus designs
64. Schmitt trigger failures and successes
65. Oscilloscope phasing
66. Choice of mosfets
67. Sram designs and dram sucks dick
68. Future design notes
69. Timing calculations
70. Why the speed is so bad
71. Test probes
72. Detection of short circuits
73. Soldering large blocks (and de-soldering)
74. Cutting wires
75. Soldering iron tips and design of that bitch
76. Ttl amplifiers (maybe ecl?)
77. 90 degrees phase shift correlates with integral of sin(x) so max voltage
78. Polycarbonate base and static
79. Diagnostic program designs
80. Identification of flakiness cause
81. Outside the box
82. Narrowing down to real problem
83. Capacitance points conjecture but WRONG (luckily)
84. Duty cycles
85. Iterative testing
86. Auto reset but it seems to not work
87. Vga attempts
88. Terminal display hack
89. Simulators of this stuff using javascript or something
90. Read the corth readme.txt
91. Collision checker
92. Probability of longest string of ones to calculate speed of logic gate and rotate “and”
93. Radio signal pickup when use audio cable over computer
94. Gas duster
95. Keeping soldering iron tip clean
96. Ad hoc design vs other stuff
97. Use eeprom to store micro instructions?
98. Breadboards are flaky as balls
99. Bus wire connections
100. Arduino-sram used as rom (and why it is so jank)
101. Dip tabs breaking and how to fix them
102. Why i should have elected to use zifs
103. Methods to actually remove and replace rom chips
104. Stop using diode clock because it is horrible
105. Wire gauge
106. Definition of an instruction
107. Separating lights into groups of 4
108. Being an “or” machine
109. Distribution of the ALU into the registers themselves, supplemented by branching
110. Extreme noise and power rail oscillations
111. How to solder cleanly
112. Bread board hot glue and pry off if need
113. Wrapping battery in foil to use as spacer
114. Internal inversion ->saves transistors in rare cases but is usually just cancerous
115. Physical machine creation procedures
116. Staple bridges on single sided
117. Blank, one-sided from fry’s is 5x more expensive than custom routed 2 sided from jlcpcb, and jlcpcb reliability >> custom etch reliability
118. Computers in terms of thermodynamics
119. Female vs male headers (eg why male > female)
120. Smd resistors are too big
121. Testing procedure
122. Big O efficiency for ram, rom, and time in terms of word size for operation
123. Depth of programming languages and a rigorous definition; eg windows operating system is the integral of python which is the integral of c which is the integral of assembly which is the integral of machine code which is the integral of cpu circuitry design which is the integral of electrical engineering which is the integral of applied mathematics which is the integral of number theory which is the integral of logical axioms that we assume to be true.
124. P vs np and how this research sheds light on that shit
125. Coupling capacitors
126. Proof that computation is inherently undefinable
127. Entropy and information density
128. Editing process and what i have learned about making shit actually editable
129. Copy examples from journals and github
130. Basically this should be a legible embodiment of the journals
131. Shielding
132. Soldering with flux
133. Mistake wire
134. Trace dimensions on pcbs and frying traces
135. Flakiness of card edge stuff
136. Tracing rules and ease of construction
137. Scaling (or “integration”) of ideas to increase complexity exponentially
138. Why masks are so sexy
139. Comparison to biology
140. Exponential difficulty because you have to build and design, then everything fails and thats 100 hours down the drain so you start again.
141. Cable management
142. Interpret in terms of state machines
143. Asnx and jzor
144. (work/clock : transistor) ratio

Notes

* Read other papers
* Ensure that you have rigor
* Only include inventions for the most part, not repetitions of what has already been done or notes
* Go back and re-learn how your first three cpus worked to get insight into the earlier days and thus more content