Kevin Martin
Syracuse University
Summer 2020 – CIS655, Tuesday @ 9:00pm EST

Homework 1

1. Research and Reading Assignment

The four papers summarized all presented their take on one overall common theme: simplicity in instruction sets versus more complex implementations. Two papers favored complexity, one favored simplicity, and the last sought to comment on the third. Interestingly, the papers that were in support of complexity were focused on new hardware systems and were written more as technical papers on their merits. Both machines discussed, the B5000 (Lonergan and King) and the IBM System/360 (Amdahl et. al) chose to take the "complex" architecture approach, and as such their respective papers champion this decision. The Case for the Reduced Instruction Set Computer (Patterson and Ditzel) forges its own path with instruction set theory, yet specifically references the System/360 and some of the potential issues with it. Finally, Comments on "The Case for the Reduced Instruction Set Computer" (Clark and Strecker) picks specific arguments with the referenced paper, but also takes a more balanced view than the two product centered reports.

The first three papers were written independently, but there are several common themes that each addresses and solutions proposed to the problems most salient at the time. One of the most interesting takeaways I had from reading these was the enthusiasm that each side can describe on two completely opposing approaches. Without having a good background in some of these topics, a reader could assume that either approach is correct and the other is entirely incorrect. I began, randomly, with the paper from Patterson & Ditzel, and I believe that colored my opinion on many of the subjects discussed. I found their postulation about favoring a reduced instruction set computer (RISC) as opposed to a complex instruction set computer (CISC) to be more compelling than any of the other arguments presented. The criticisms levied by Clark and Strecker are valid, but I believe they poke holes more in the granular aspects like definitions as opposed to presenting convincing evidence to favor a CISC model.

Additionally, these papers are all quite dated. This is important because it provides valuable context. When they were written, the topics discussed did not have definitive answers, nor was there concrete evidence to support one path over another. In theory, either approach (RISC or CISC) could have been the victor. However, because we now have the benefit of hindsight, we know RISC implementations have proven to be more successful *in general*. Modern x86 processors for example are of course in the CISC family, but not to the degree that was utilized by either the B5000 or the System/360. In the current age of computing, there are not hard lines that distinguish the two philosophies, but rather gradients of how much of each concept a processor embodies. In discussing these papers, I also want to highlight a couple of points that were brought up on both sides and what events actually unfolded. In doing so, I hope to illuminate which ideas were better thought out than others.

A perfect example comes from the System/360 paper when describing the choice of whether to use (the then recently adopted) ASCII character size or the legacy BCD system. The ASCII system implements a 7-bit character size, whereas BCD uses 8-bit. At this time, the ASCII system was under "final consideration" by the International Standards Organization for international standards recommendation. Obviously now we know that not only did ASCII become adopted shortly thereafter, but it has proven to be a success for over 50 years. The designers at IBM did not know this and opted to include both

sets. They created their CPU to accept both instructions, with “program-selectable BCD or ASCII modes.” In doing so, the construction of the CPU becomes much more complicated, but also allows for greater flexibility.

Similarly, the B 5000 has multiple operating modes which can accommodate both fixed-length words or variable-length words. The relevant paper mentions the benefit as “by combining both abilities in one processor, a processor can operate in the mode most desirable for the operation at hand.” While this is true, it does not elaborate on the design difficulties needed in achieving this. More importantly, it does not further identify why this approach is more desirable than only allowing fixed-length words. The paper describes the different modes (character mode and word mode, respectively), but does little to convince the reader this was the optimal choice. Only in the conclusion can we see what the designers had intended, and that was for a machine to be a one-size-fits-all solution. An ambitious goal and one certainly with merit, but perhaps not the best goal to be focusing on.

The designers make it seem like this decision, and others like it, were easy to implement, almost as if it was just a simple flip of a switch. However, this is not the case, and implementing multiple solutions to catch every scenario is impractical for several reasons. First, as Patterson & Ditzel note, this lengthens design times. Now not one but two sets of instructions must be tested and debugged. This slows down the design cycle and potentially allows for competitors to catch up. A second problem is increased design error. Specifically, with the System/360 there was a “bad experience with errors” and as a result this paved the way for future design decisions.

Another issue, or at least difference, between the papers is that the two focusing on products are very much concerned with problem solving in the present. Patterson & Ditzel focus on long term implementation goals and look towards the future. By analyzing various performance metrics, they argue that perhaps not all operations need to be supported, or not supported in the same (legacy) way. Performing one complicated step may actually be slower than multiple simpler steps. Indeed, we see this paradigm in assembly, which has faithfully stuck to many of its original design implementations. While there has been some evolution, the modifications are modest, and have thus allowed this design to flourish. The authors had foresight by expressing the desire to keep instruction sets simpler and allow for growth above the chipset level.

Following on this idea, the authors also present the idea of high-level language support. Here, one of the great mantras of modern programming is stated: “at no time need the programmer be aware of any lower levels in the writing or the debugging of a program.” This is a very eloquent way of describing abstraction, a design philosophy that has consistently shaped the programming field. Again, the foresight of the authors is apparent as this abstraction concept proved to be the correct way of thinking. By bringing up the compiler, the authors touch on a very important point. Complex instruction sets do not ease the burden for compiler designers. This is a fact, and one that has been supported by decades of evidence.

Finally, I’d like to bring up the last paper by Patterson and Ditzel. Personally, I do not find their paper convincing in utilizing CISC over RISC, nor do I believe that was their intent. They were simply highlighting some of the potential fallacies and over-generalizations that one may find in reading Patterson & Ditzel’s paper. Even still, I do not find their complaints compelling. Issues are raised, but no solutions are provided. For instance, the “complexity vs. size” section argues that the definitions of reduced and complex are not well suited to be compared. That just doesn’t make sense, of course they are suited to be compared. But their point about how to describe instruction complexity is more valid, and the idea that original paper contains no formal definition of a RISC or CISC is also true. However,

I do not think that an absence of a formal definition discredits the arguments. It would be helpful in identifying certain systems for future use, but the core concept is: rely on simple instructions and allow growth on top, or try to build a catch all model for every scenario? The answer, as history shows, is obvious.
The other complaints listed, such as time to design and increasing of design errors, are only supported through mere anecdotal evidence. If the authors wish to poke holes in a paper that does not contain enough concrete data, they should provide some concrete data. I find that my frustration with this paper is again largely supported by the gift of hindsight and looking at these complaints now seems unfair.

In summary, I am glad to have read the Patterson & Ditzel paper first as it is the most correct of all of them. Seeing each machine lauded for its designs is an interesting glimpse into the past, but none of the ideas presented proved to be of merit. I commend Patterson & Ditzel for their grasp on programmer/hardware dichotomy, and an uncanny ability to predict the future of computer design.

2. Q1.1

Assume a wafer yield of 100% and N = 13.5
Updated chart with correction for IBM Power5 defects:

Chip	**Die Size** (mm^2)	**Est. Defect rate** (cm^2)	**Manufacturing size** (nm)	**Transistors** (millions)
IBM Power5	389	0.03	130	276
Sun Niagra	380	0.75	90	279
AMD Opteron	199	0.75	90	233

a. What is the yield of the IBM Power5?
N = 13.5
wafer yield = 100%
Defects = .03
die area = 389/100 = 3.89 (convert mm^2 to cm^2)

Yield = wafer yield x 1 / (1 + Defects x die area)^N
Yield = 1 x 1 / (1 + (.03 x 3.89)$^{13.5}$
Yield = 1 / (1 + 0.1167)$^{13.5}$
Yield = 1 / 4.4375
Yield = **0.2253**

b. Why does the IBM Power5 have a lower defect rate than the Niagra and Opteron?
The IBM Power5 uses a larger transistor size, which is an older technology. Older technologies have had more time to work out issues, thus the expected error rate should decrease as time goes on.

3. Q1.5 (part a)

A cooling door for a rack costs $4000 and dissipates 14 KW (into the room; additional cost is required to get it out of the room). How many servers with an Intel Pentium 4 processor, 1 GB 240-pin DRAM, and a single 7200 rpm hard drive can you cool with one cooling door?

Intel Pentium 4 processor: 48.9-66W
1 GB 240-pin DRAM (Kingston D2N3 1GB): 2.3W
7200 rpm hard drive (DiamondMax 9): 7.9W read/seek, 4.0W idle

Assume worst case (maximum power draw). Peak power consumption = 66 + 2.3 + 7.9 = 76.2W

Cooling door can dissipate 14KW, therefore: 14,000W/76.2W = 183.72
The maximum number of servers that can be cooled by one cooling door is **183**