The Case for the Reduced Instruction Set Computer

David Patterson and David Ditzel
Context

• It’s the early 80s
• Single-chip processors have just become feasible (area is expensive, but seems it might get cheaper)
• High-level languages (i.e. LISP) are all the rage
• Hand-coded assembly is falling from favor. Compilers are on the rise.
• We’ve seen a few generations of machines in a single family come out. There were lessons to learned.
  • Design lessons
  • Use lessons
  • Lots of people have built lots of different systems
Complexity drivers

• It’s expensive to fetch instructions because memory is slow.
  • No caches is old machines
  • (note that they used a micro-code store instead, which is basically a form of statically managed icache)
• Adding microcode instructions is cheap
• Code Density
• Marketing
• Backward compatibility
• High-level language support
• Partly, it was not clear that going complex was bad. Why stop?
Costs of Complexity

- Irrationality
  - The “tailor-made” instruction is often slower than the equivalent sequence of instructions.
- Increased design time
- Increased design errors
- Won’t fit on a single chip (i.e., area costs)
- Design time increases
- It’s a poor use of chip area.
RISC Today

• This paper is about simplicity
• RISC has really turned into two arguments
  • A simple interface to the hardware
    • Good for compilers -- rational, regular, consistent
    • Increased flexibility for implementors
  • Hiding complexity and targeting at things that matter today
    • Parallelism
    • Pipelining
    • Spend complexity where it pays
• Modern machines are not simple by any means
  • But a simple machine would not be faster.
  • Mostly because of memory.