The Ol' RISC vs. CISC Debate

Anna Cavender, Jessica Miller

CSE 548

January 5, 2005

Prerequisites

- Classic debate in architecture: Is it best to support HLLs (High-Level Languages) in hardware or software?
  - Continuum of solutions that affect instruction set architecture:

    software RISC <-> CISC <-> HLLC hardware
- Before 1980's great deal of focus on reducing the semantic gap between HLLs and MLs (Machine Languages)

First Paper: Retrospective on HLL Computer Architecture

- Major contribution: Enumerates and examines validity of justifications made for using HLLCs
- Structure of Paper:
  - Review of Justifications
  - Definition
  - Metrics
  - Attributes
  - Weird Section that doesn't quite fit (Scheme for Transparent Reversibility of Compilation Process)

The Arguments Against the Axioms

- Which of the authors' arguments still work today?
  - Arguments against code compactness, compiler complexity, and ease of programming still seem to work
  - Any that don't hold today?

Critiques/Questions

- Does their definition really help? It seems like anything would qualify as HLLC…
- Is the metric that is introduced reasonable?
- Um, that last section definitely needs to be reorganized…

Nothing RISC'd Nothing Gained: Second Paper (A VSLI RISC)

- Major contribution: Proof of concept that RISCs can be competitive with HLLCs and CISCs.
- Structure of Paper:
  - Motivation
  - Factors influencing design
  - Architecture of RISC I
  - Results and Discussion

Motivation

- Problems with CISCs:
  - Increased design time
  - Increased design errors
  - Inconsistent implementation
- Hypothesis: By reducing instruction set one can design a suitable VLSI architecture that uses scarce resources more effectively than CISCs.

Architecture of RISC I

- 31 instructions in four categories (arithmetic-logic, memory access, branch, misc.)
- Load/store use two cycles
- Register Windows
- Delayed Jump

Results

- Ran faster on benchmarks than all other systems in set
- Increased size of machine code at most by a factor of two

Open Questions

- How many instructions are too little or too much (today's RISCs have more instructions than RISC I)?
- Would RISC I be sufficient for today's HLLs (e.g. Java, Python)?
- Are there reasons other than legacy issues that popular chip families don't use RISC?
- Are there particular applications/domains that would be able to efficiently leverage complex instructions?