CSE 120
Day 4 Notes

Elijah Hantman

Review: Iron Law

$$E = \frac{Instructions}{Program} \times \frac{Cycles}{Instruction} \times \frac{Seconds}{Cycle} \tag{1}$$

$$E = \frac{CPI * IC}{CR} \tag{2}$$

> The CPI will vary per instruction type. This implies the average or global CPI will vary between programs with different compositions of instruction types.

Relative Performance

$Performance = \frac{1}{\text{Execution Time}}$

Relative Speedup = $Performance_{old}/Performance_{new}$

Instruction Set Architecture

ISA's define:

- Defines a protocol between hardware/software
- Defines the state of the system (registers, memory, etc)
- Functionality of each instruction
- encoding of each instruction

ISA's don't define:

- How instructions are implemented in hardware.
- How fast or slow instructions are
- how much power each instruction will cost.

A good ISA creates a stable platform that allows software and hardware to evolve seperately.

Classifying ISAs

- Basic differentiation: are the ALU operands coming from registers or RAM?
- Stack ISA
- Accumulator ISA
- Register-Memory ISA
- Register-Register/load-store ISA

Why registers and RAM and not just Registers?

Registers are made out of individual flip flops, each one is significantly faster than memory or RAM.

Registers are not memory dense and are expensive. In addition each additional register requires more complicated hardware and ISAs to encode and manage.

Physical distance is also a limiter on register speed.

Register-memory ISA

Can operate on both registers or memory, always outputs to a register.

Only accepts one operand from memory and one operand from a register. Generally the convention for ISAs is the register output is first, then the register source, then the Memory source.

An example add might look like

```
add Rd, Rs, M;
```

Register-Register ISA

Only operates with registers.

Requires explicit movement of values to and from memory. The memory writes and reads are directly exposed to the software.

The ISA type is based on **Only** the ALU's relationship with memory. They do not affect other memory or external operations.

Generally Register-Memory structures would be used for CISC computers, and Register-Register ISA's were used for RISC computers.

RISC computers tend to require greater numbers of instructions. In the early days of computing this was a big deal and led to CISC architectures becoming more dominant.

Instruction Length and Format

- Fixed Length ISAs
  Address of next instruction is easy to compute, however the code density suffers due to common and rare instructions requiring the same number of bits.
  It also has the benefit of easy prefetching.
- Variable Length ISAs
  Generally gives better code density, fewer memory fetches for instructions, more common instructions can be shorter.
  This makes the Fetching and decoding of instructions more difficult. You have to decode each instruction before you can fetch the next instruction.
  Prefetching is difficult and complicated.

CISC vs RISC

- CISC (Complex Instruction Set Computer)
  The goal is to get more done by adding sophisticated instructions. Shift the complexity onto the hardware in order to save on software.
  This includes ISAs like x86, IBM 360, Motorola 68K, etc.

  ```
  MUL mem2 <= mem0 * mem1
  ```

- RISC (Reduced Instruction Set Computer)
  Goal is to simplify ISA to allow for simple and fast hardware. It moves the complexity onto the Software to allow for less complicated hardware.
  Examples include MIPS, SUN, Sparc, RISC-V, etc.

  ```
  LOAD  reg0 <= mem0
  LOAD  reg1 <= mem1
  MUL   reg0 <= reg0 * reg1
  STORE mem2 <= reg0
  ```

Generally both CISC and RISC architectures have adapted for different use cases. RISC instruction sets for high performance have had to add additional instruction for hardware acceleration.

CISC computers use the concept of micro-ops to achieve more of the speed and parallelism that RISC lends itself to.

Due to how complicated ISAs and hardware has become, the only reliable way of benchmarking is repeated measurement.

CISC	RISC
HW difficult to implement	HW easy
Multi-cycle complex instructions	Simple, sincle clock instructions
Small Code Size	Large Code Size
High CPI	Low CPI
Variable Length instructions	Fixed instructions