Instruction Sets

Why Not Make My Own?

Paul Gentemann
10/21/2014
Objectives

● Learn about Instruction Sets (IS).
● Design something new - a special-purpose IS that handles probabilities.
● Explore alternatives to standard floating point numbers.
Types of Instructions

- **Data Handling/Memory**
  - Load memory or a value into a register.

- **Arithmetic/Logic**
  - Add, subtract, multiply, divide, bitwise operations, compare values.

- **Control Flow**
  - Branch, make a call, conditional branch.
Types of Instruction Sets

- CISC - Complex Instruction Set Circuit
- RISC - Reduced Instruction Set Circuit
- MISC - Minimal Instruction Set Circuit
- OISC - One Instruction Set Circuit
- ZISC - Zero Instruction Set Circuit
- Still waiting for negative numbers...
CISC - Complex

- Think x86 architecture.
- Named because instructions are capable of executing several low-level operations.
  - LEA EAX, [ ECX + EBX + 1234567 ]
- Designed to support high-level programming concepts.
- Too general-purpose for our needs.
RISC - Reduced

- For example, ARM.
- The name is meant to contrast CISC, though the gulf has narrowed over the years.
- Designed to use an optimized set of instructions.
  - Do one thing well philosophy.
  - Memory ops are generally separate from math ops.
MISC - Minimal

- ENIAC, or most ancient digital computers.
- Simpler design and implementation:
  - generally a stack-based machine
  - >1 instruction, <33 instructions
  - Not support general inputs
  - No hardware memory protections
  - Only Load/Store memory ops are supported
  - 64KB to 4GB memory, usually <1MB
  - No pipelines, branches, superscalar, register renaming
OISC - One

- Most “famous” is probably subleq:
  - Sample instruction: A B C
  - Means: B = B - A, jump to C if B<=0
- Cool, but not exactly the point of this project.

http://da.vidr.cc/projects/subleq/
ZISC - Zero

- Check out the CM1K.
- Massively hardwired, parallel processed, neural network stuff.
- “Zero” because no traditional instructions.
- Also not in scope, but still cool.
ZISC - Zero

- 15 registers
- 1024 neurons
- 4 “operations”
  - broadcast vector
  - recognize/learn vector
  - save knowledge
  - load knowledge

http://general-vision.com/documentation/TB_CM1K_simplest_API.pdf
Instruction Set Decision

● Either a simplified RISC, or a slightly extended MISC.

● Reasons for MISC:
  ○ Everything except the stack-focus.

● Reasons for RISC:
  ○ Uniform instruction format
  ○ Identical Registers
  ○ Simple addressing modes
Data Types

- Most integers and floats are $<-1$, or $>1$.
- Probability lies entirely in the range of $[0,1]$; except for quantum physicists, where $[-1,1]$ is perfectly acceptable and useful.
- Feynman illustrating how negative probabilities might be interpreted: http://cds.cern.ch/record/154856/files/pre-27827.pdf
Data Types - Denormal?

- Use only denormalized numbers?
  - Multiplication will always shrink, so accuracy will eventually reach 0.
  - In 32-bit, denormalized numbers exist in the range of $\pm 2^{126}$, or about $\pm 1.18\times10^{-38} \ldots$ so no such thing as .5, let alone 1.
Data Types - Fixed Precision?

- Fixed-precision floats?
  - Essentially, split up the number line into equally-spaced pieces, determined by number of bits.
  - Can add flag bits to assist with divide-by-zero, +/-inf, NaN, out-of-range [-1, 1].
  - -1, 0, and 1 are in the set, and thus accurately represented.
  - Epsilon circuit, which checks whether an operation has underflowed our precision.
Data Types - Fixed Precision?

- Ex: 3-bits splits number line into 7 values: -1, -\(\frac{2}{3}\), -\(\frac{1}{3}\), 0, \(\frac{1}{3}\), \(\frac{2}{3}\), 1.

- \(2^n - 1\) unique values, one excluded, balanced around zero.
Data Types - Fixed Precision?

- Denominator is (#values-1) / 2.
- Multiplication will often require more bits for accuracy.
  - No more than 2 bits, and that is squaring the smallest nonzero value.
Data Types - Analog-Digital

- Use analog circuits, where voltage on a given bit represents the probability that the bit is a 1.
- Fast, simple, power efficient.
- Just recently learned of this type, so requires more research.

https://www.cs.uaf.edu/2011/spring/cs641/proj1/rarutter/
Operations - Algebraic Analogues

● Algebraic operations’ probability analogues:
  ○ Addition - sum of probabilities of mutually exclusive events. Also known as OR.
  ○ Subtraction - sum of probabilities excluding other events (usually All events not including event A).
  ○ Multiplication - independ joint probability of events. Also known as AND.
  ○ Division - used in conditional probabilities.
Operations - Unique to Probability

● Conditional Probability -
  \[ P(A \text{ given } B) = \frac{P(A \text{ AND } B)}{P(B)} \]
  \[ = \frac{P(B \text{ given } A) \times P(A)}{P(B)} \]

● Mutually dependent events -
  \[ P(A \text{ OR } B) = P(A) + P(B) - P(A \text{ AND } B) \]

● Negation -
  \[ P(\sim A) = 1 - P(A) \]
Probability and CS haven’t communicated much. Here are the instructions I propose (let “val” be a number or register.

- add val, val
- sub val, val
- mul val, val
- div val, val
- con val, val, val, val
- mde val, val, val
- not val
Operations

● It might be valuable to make not special, so you can say “add not val, val”, but that may be too complicated for the circuit I intend to design.

● There are also considerations from Lawlor’s comments on my draft that need to be addressed.
Thanks!

- You may applaud.