Basics of Instruction Set Design

Shaping the Outline of an Architecture

General or Special?

- What is the goal of the architecture?
  - General purpose, but better
    - Faster, lower power, more secure
  - Special purpose, for an application or domain

General

- Covers all domains well
- Broad set of data types
- Instructions for all common operations
- Meant to improve upon what exists
  - Most recent significant commercial attempt at a clean-sheet design is Itanium
  - RISC architectures of the 80s
- Backwards compatibility leads to evolution

Special

- Many domains have distinct data types and operations, or unique constraints
- Signal processing, embedded control, GPU, network routing, database, mobile, high security, vector processing, gaming, neural nets
- Sometimes clean sheet, often customize an “IP” core

Basics

- Word size (e.g., 32 bits)
- Data types
- Registers
- Execution model

Word Size

- Affects all aspects
  - Data types, instructions, addressing, fetch
- Typically a power of 2 (8, 16, 32, 64 bit)
- Many 64-bit designs pack 2 32-bit instructions into a word (Itanium packed 3 40-bit instructions in 128 bits)
- Not advisable to buck this tradition

Basics

- Word size (32 bits)
- Data types
- Registers
- Execution model

Data Types

- Integers: 8, 16, 32, 64 bits
- Floating point: 32, 64, 80 bits (IEEE 754 standard)
- Chars: 8, 16 bits
- Vectors: 64 words, streams (MMX/SSE or Cray style)
- Pixels: splitting of integers into fields
- Strings, structs, objects, pointers, packets, complex, rationals, etc.

Implications of Types

- Each type necessitates another set of instructions (been tried: tagged memory)
- May need separate register bank for each type, especially if formats incompatible
- Smaller Int types usually packed within larger Int types (e.g., 64-bit registers with a pair of 32-bit values, 4 16-bit, 8 bytes)

Choosing Types

- What is necessary and unchanging?
- What can be efficiently programmed on top of a simpler set of types?
- Are there any special types that are easily described and that will boost performance?
- Are they expressible in a high level language in a way that a compiler will recognize and be able to use?
- How do they relate to word size?

Floating Point

- Better to keep in separate registers due to much different format from Ints
  - Avoids some NaN exceptions
- 64-bit regs can hold 2 32-bit values, but have to be careful about how operations use the registers. (Stick to one size of FP unless you need both.)
- IEEE 754 standard essentially rules this space

MMX/SSE Vector

- Word divided into smaller Ints -- 64-bit word holds 4 16-bit values
- Arithmetic operations take place on smaller values in parallel
- Need operations for packing/unpacking

Cray-Style Vector

- Bank of N registers each with M words (usually FP)
- Operations between corresponding registers (vector-vector)
- Scalar-vector, vector reduce to scalar
- Gather/scatter, population count

M

00	01	02	03
10	11	12	13
20	21	23	23
30	31	32	33
40	41	42	43
50	51	52	53
60	61	62	63
70	71	72	73

N

Operations

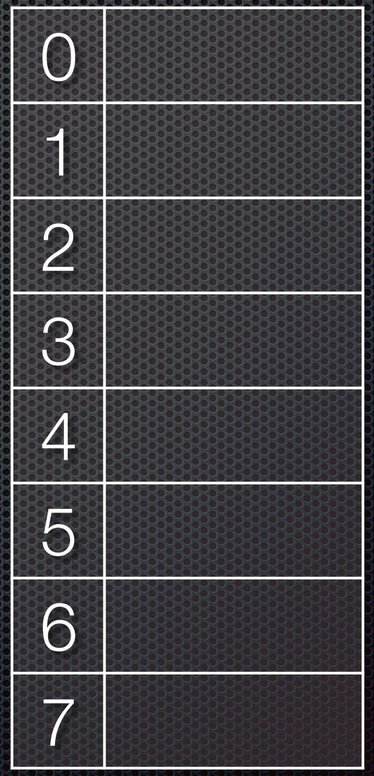
- What ops go with each data type?
- Integers: Arithmetic, logic, comparisons
- Floats: Arithmetic, comparisons
- Chars: Pack, unpack, shift, mask, compare
- Vectors: Gather, scatter, arith., pop. count

Basics

- Word size (32 bits)
- Data types
- Registers
- Execution model

Registers

- General vs. special purpose
  - Mixed
- Bank size considerations
  - How many are useful?
  - Space in operand fields
- Bank per type vs. unified
- Same or different size?



Registers

- General vs. special purpose
  - Mixed
- Bank size considerations
  - How many are useful?
  - Space in operand fields
- Bank per type vs. unified

0	0
1	
2	
3	
4	
5	SP
6	Return
7	PC

Basic Computational Needs Defined

- Word size
- Data types
- Operations
- Working registers
- Now ready to address execution paradigm

Basics

- Word size (32 bits)
- Data types
- Registers
- Execution model

Execution Model

- Number of addresses (operands) per instruction
- Fetch/next instruction
- Memory organization
- Control

Addresses (Operands)

- Three per instruction is flexible and efficient in time, but requires more space within the instruction
  - A = B op C
- Two per instruction makes one do double-duty. More compact instructions
  - A = A op B

Less Common

- One address requires an implicit second operand (a unique register, typically called the accumulator). Compact instructions, but more of them
- Zero address uses a stack. Tiny instructions, and a challenging programming model

Fetch

- Single instruction per word, autoincrement PC
- Multiple instructions per word, autoinc PC
- Multiple words per instruction, PC advances by variable amount
- Micro-operations in long (or multiple) word, including next PC in each

Memory Organization

- Harvard architecture -- separate instruction and data memory
- Princeton (von Neumann) architecture -- combined
- Word/byte address
- Data path width
- Address space