The course so far...

- Quantifying Performance (Chap 1)
  - How to gauge performance... (inside cover)
- Memory hierarchy Design (Chap 5)
  - Where the majority of time is spent...
  - Application 1 of what we learned... Imbench
  - Application 2 of what we learned... TBA

Where we’re headed...

- Getting to know MIPS64 (inside back cover)
  - Chap 2
- How pipelining works...
  - Appendix A
- ILP
  - Chap 3
  - Parts of Chap 4
  - Maybe parts of Chaps 6/8

Classifying ISA by Internal Storage Classification

- Implicit operands
  - Stack architecture
    - Push A, Push B, Add, Pop C
  - Accumulator architecture
    - Load A, Add B, Store C
- Explicit operands
  - Register-memory architecture (GPR)
    - Load R1, A; Load R2, B; Store C, R1
  - Register-register architecture (load store/GPR)
    - Load R1, A; Load R2, B; Add R3, R1, R2; Store C, R3

\[ C = A + B \]
GPR Architectures

- Register to register (0,3 → MIPS)
  - Simpler ISA encoding, predictable instruction latency
  - Higher instruction count, wasteful encoding
- Register to memory (1,2 → x86)
  - Operands access memory, good encode density
  - Can result in complex instruction set encoding, instruction latency varies
- Memory to memory (2,2 → VAX)
  - No wasted registers, variable instruction length, instruction latency varies, memory access inefficient

*(m,n) pair is (# mem operands, # total operands)*

Memory Addressing

What object is accessed as a function of address and length?

- Byte addressable memory
  - Byte = 8 bits, word(0,4) = 32 bits, dword(0) = 64 bits
  - Accesses larger than a byte must be aligned
  - Byte ordering must be considered
- Typical operand types supported
  - Character: 1 byte
  - Integers: byte, half word, word
  - Single precision fp: 1 word
  - Double precision fp: 2 words

Little vs. Big Endian Byte Ordering

For byte aligned addressing, an address gives the location data begins

Example:
Store dword value 0x1234567F @ dword aligned address 0x00001FA0
Read dword one byte at a time from 0x00001FA0

Little Endian:
Address of a multiple-byte data type is
of its least significant byte

Big Endian:
Address of a multiple-byte data type is
of its most significant byte

More on alignment...

- How data is stored in memory is encoded in instruction (ISA dependent)
- MIPS architecture requires byte aligned addressing, allows byte, word, dword loads and stores to memory (no partial ALU ops)
- Shifting can be used to easily align addresses to word, dword, etc. boundaries

Addressing Modes

- Describes how architectures specify the address of an object they will access (effective address)
- Reduced instruction counts vs added complexity
- Most common addressing modes
  - Register: Add R4, R3
  - Register Indirect: Add R4, (R3)
  - Immediate: Add R4, #3
  - Size of immediate field? 8 - 16 bits
  - Displacement: Add R4, 100(R1)
    - Size of displacement address? 12 - 16 bits
    - Memory Indirect: Add R4, (R1)
      - Scaled: Add R1,100(R2)[R3]
Instruction set operations

- ALU: add, sub, and
- Data transfer: load/store
- Control: branch, jump, calls and returns, traps
- System: system calls
- Floating Point: fadd, fmultiply
- Decimal: decimal add, multiply
- String: move, compare, search
- Graphics: pixel ops, compression

*Most widely used instructions are also the simplest operations.

Control Flow

- Conditional: branch
  - PC relative (8 bits)
  - Comparisons
    - Compare and branch (special cases like jle)
    - Condition register (more complex comparisons)
- Unconditional: jump
  - Indirect addressing - target not known at compile time
  - Dynamic shared libs, virtual functions, etc.
  - Architecture support for saving registers
  - Caller vs. callee saving

Instruction Encoding

- All instructions must be encoded into a binary representation for execution by CPU
- Operation specified in opcode (add, sub, etc.)
- Conflicts: (Registers + addr modes greatly influence encoding)
  - Desire high number of registers and addressing modes
  - Both increase complexity, instruction size, and program length
  - Desire fast and simple decoding of instructions

Variable vs. fixed length encoding

- Variable length encoding (x86)
  - 1-17 bytes
  - add EAX, 1000(EBX)
  - 1 byte for op code (32-bit int add + 2 operands)
  - 1 byte for operands (source/dest reg + addr mode + base reg)
  - 4 bytes for address field (disp 1000)
- Fixed length encoding (MIPS)
  - Same number of operands
  - Field use determined by instruction op code
  - Addressing modes specified in op code
- Hybrid (MIPS16)
  - Finite number of formats specified in opcode
  - Number of fields dependent on op code (variable)

Multi-pass compilers

- Regular, orthogonal ops, data types and addr modes
- Design primitives, not high-level features
- Simplify trade-offs among alternatives

Arch design to aid compiler design
Summary of quantitative design decisions regarding ISA

- GPR load/store architecture for simplicity
- Support displacement, immediate, indirect addressing modes
- 8-, 16-, 32-, and 64-bit int and 64-bit fp types
- Support load, store, add, sub, mv reg to reg, shift
- Support cmpe, cmpne, cmpl, branch (8-bit PC), jump, call, return
- Fixed instruction set encoding (perf over code size)
- At least 16 GPRs, separate fp regs

Who uses MIPS?