15-740/18-740 Computer Architecture Lecture 2: ISA, Tradeoffs, Performance

Prof. Onur Mutlu
Carnegie Mellon University

Last Time …

- Brief intro to
  - On-chip networks
  - Memory controllers, DRAM bank organization
  - Static vs. dynamic scheduling
  - Cache coherence
- Moore's Law
- Levels of Transformation
- ISA vs. microarchitecture distinction

Required Readings

- Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.

- Wulf, "Compilers and Computer Architecture," IEEE Computer 1981.

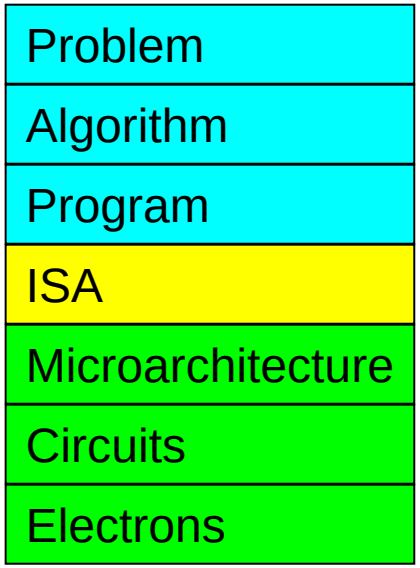
Papers for Review

- Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.

- Due September 17

Levels of Transformation

- ISA
  - Agreed upon interface between software and hardware
    - SW/compiler assumes, HW promises
  - What the software writer needs to know to write system/user programs
- Microarchitecture
  - Specific implementation of an ISA
  - Not visible to the software
- Microprocessor
  - **ISA, uarch**, circuits
  - "Architecture" = ISA + microarchitecture



ISA vs. Microarchitecture

- What is part of ISA vs. Uarch?
  - Gas pedal: interface for “acceleration”
  - Internals of the engine: implements “acceleration”
  - Add instruction vs. Adder implementation

- Implementation (uarch) can be various as long as it satisfies the specification (ISA)
  - Bit serial, ripple carry, carry lookahead adders
  - x86 ISA has many implementations: 286, 386, 486, Pentium, Pentium Pro, ...

- Uarch usually changes faster than ISA
  - Few ISAs (x86, SPARC, MIPS, Alpha) but many uarchs
  - *Why?*

ISA

- Instructions
  - Opcodes, Addressing Modes, Data Types
  - Instruction Types and Formats
  - Registers, Condition Codes
- Memory
  - Address space, Addressability, Alignment
  - Virtual memory management
- Call, Interrupt/Exception Handling
- Access Control, Priority/Privilege
- I/O
- Task Management
- Power and Thermal Management
- Multi-threading support, Multiprocessor support

Intel® 64 and IA-32 Architectures
Software Developer's Manual

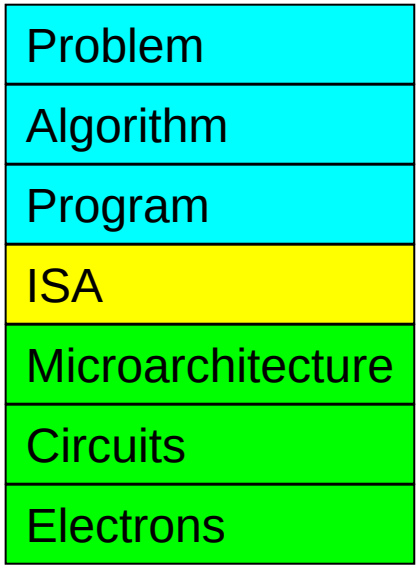
Volume 1:
Basic Architecture

Microarchitecture

- Implementation of the ISA under specific design constraints and goals
- Anything done in hardware without exposure to software
  - Pipelining
  - In-order versus out-of-order instruction execution
  - Memory access scheduling policy
  - Speculative execution
  - Superscalar processing (multiple instruction issue?)
  - Clock gating
  - Caching? Levels, size, associativity, replacement policy
  - Prefetching?
  - Voltage/frequency scaling?
  - Error correction?

Design Point

- A set of design considerations and their importance
  - leads to tradeoffs in both ISA and uarch
- Considerations
  - Cost
  - Performance
  - Maximum power consumption
  - Energy consumption (battery life)
  - Availability
  - Reliability and Correctness (or is it?)
  - Time to Market



- Design point determined by the “Problem” space (application space)

Tradeoffs: Soul of Computer Architecture

- ISA-level tradeoffs
- Uarch-level tradeoffs
- System and Task-level tradeoffs
  - How to divide the labor between hardware and software

ISA-level Tradeoffs: Semantic Gap

- Where to place the ISA? Semantic gap
  - Closer to high-level language (HLL) or closer to hardware control signals? → Complex vs. simple instructions
  - RISC vs. CISC vs. HLL machines
    - FFT, QUICKSORT, POLY, FP instructions?
    - VAX INDEX instruction (array access with bounds checking)
  - Tradeoffs:
    - Simple compiler, complex hardware vs. complex compiler, simple hardware
      - Caveat: Translation (indirection) can change the tradeoff!
    - Burden of backward compatibility
    - Performance?
      - Optimization opportunity: Example of VAX INDEX instruction: who (compiler vs. hardware) puts more effort into optimization?
      - Instruction size, code size

X86: Small Semantic Gap: String Operations

REP MOVS DEST SRC

```
IF AddressSize = 16
    THEN
        Use CX for CountReg;
    ELSE IF AddressSize = 64 and REX.W used
        THEN Use RCX for CountReg; FI;
    ELSE
        Use ECX for CountReg;
FI;
WHILE CountReg ≠ 0
    DO
        Service pending interrupts (if any);
        Execute associated string instruction;
        CountReg ← (CountReg – 1);
        IF CountReg = 0
            THEN exit WHILE loop; FI;
        IF (Repeat prefix is REPZ or REPE) and (ZF = 0)
        or (Repeat prefix is REPNZ or REPNE) and (ZF = 1)
            THEN exit WHILE loop; FI;
    OD;
```

```
DEST ← SRC;
IF (Byte move)
    THEN IF DF = 0
        THEN
            (R|E)SI ← (R|E)SI + 1;
            (R|E)DI ← (R|E)DI + 1;
        ELSE
            (R|E)SI ← (R|E)SI – 1;
            (R|E)DI ← (R|E)DI – 1;
        FI;
    ELSE IF (Word move)
        THEN IF DF = 0
            (R|E)SI ← (R|E)SI + 2;
            (R|E)DI ← (R|E)DI + 2;
            FI;
        ELSE
            (R|E)SI ← (R|E)SI – 2;
            (R|E)DI ← (R|E)DI – 2;
        FI;
    ELSE IF (Doubleword move)
        THEN IF DF = 0
            (R|E)SI ← (R|E)SI + 4;
            (R|E)DI ← (R|E)DI + 4;
            FI;
        ELSE
            (R|E)SI ← (R|E)SI – 4;
            (R|E)DI ← (R|E)DI – 4;
        FI;
    ELSE IF (Quadword move)
        THEN IF DF = 0
            (R|E)SI ← (R|E)SI + 8;
            (R|E)DI ← (R|E)DI + 8;
            FI;
        ELSE
            (R|E)SI ← (R|E)SI – 8;
            (R|E)DI ← (R|E)DI – 8;
        FI;
FI;
```

How many instructions does this take in Alpha?

Small Semantic Gap Examples in VAX

- FIND FIRST
  - Find the first set bit in a bit field
  - Helps OS resource allocation operations
- SAVE CONTEXT, LOAD CONTEXT
  - Special context switching instructions
- INSQUEUE, REMQUEUE
  - Operations on doubly linked list
- INDEX
  - Array access with bounds checking
- STRING Operations
  - Compare strings, find substrings, ...
- Cyclic Redundancy Check Instruction
- EDITPC
  - Implements editing functions to display fixed format output

- Digital Equipment Corp., "VAX11 780 Architecture Handbook," 1977-78.

Small versus Large Semantic Gap

- CISC vs. RISC
  - Complex instruction set computer → complex instructions
    - Initially motivated by "not good enough" code generation
  - Reduced instruction set computer → simple instructions
    - John Cocke, mid 1970s, IBM 801
      - Goal: enable better compiler control and optimization

- RISC motivated by
  - Memory stalls (no work done in a complex instruction when there is a memory stall?)
    - When is this correct?
  - Simplifying the hardware → lower cost, higher frequency
  - Enabling the compiler to optimize the code better
    - Find fine-grained parallelism to reduce stalls

Small versus Large Semantic Gap

- John Cocke's RISC (large semantic gap) concept:
  - Compiler generates control signals: open microcode

- Advantages of Small Semantic Gap (Complex instructions)

  + Denser encoding → smaller code size → saves off-chip bandwidth, better cache hit rate (better packing of instructions)

  + Simpler compiler

- Disadvantages

  - Larger chunks of work → compiler has less opportunity to optimize

  - More complex hardware → translation to control signals and optimization needs to be done by hardware

- Read Colwell et al., "Instruction Sets and Beyond: Computers, Complexity, and Controversy," IEEE Computer 1985.

ISA-level Tradeoffs: Instruction Length

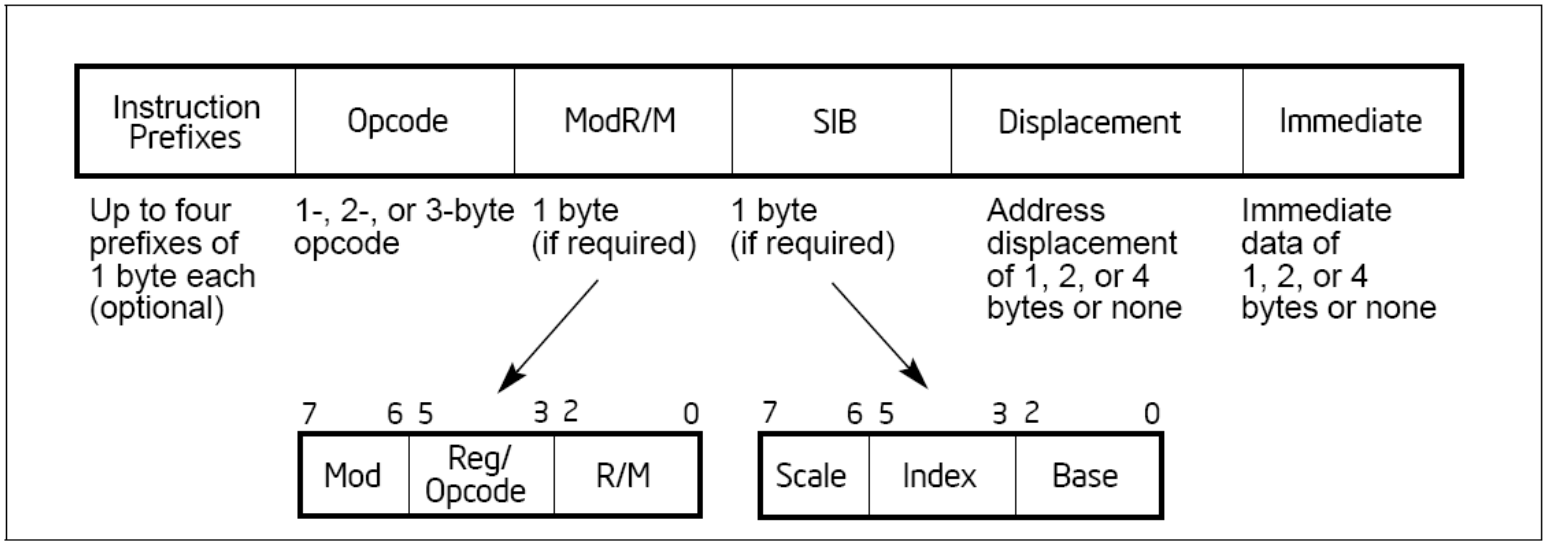
- Fixed length: Length of all instructions the same
  - + Easier to decode single instruction in hardware
  - + Easier to decode multiple instructions concurrently
  - -- Wasted bits in instructions (Why is this bad?)
  - -- Harder-to-extend ISA (how to add new instructions?)
- Variable length: Length of instructions different (determined by opcode and sub-opcode)
  - + Compact encoding (Why is this good?)
    - Intel 432: Huffman encoding (sort of). 6 to 321 bit instructions. How?
  - -- More logic to decode a single instruction
  - -- Harder to decode multiple instructions concurrently
- Tradeoffs
  - Code size (memory space, bandwidth, latency) vs. hardware complexity
  - ISA extensibility and expressiveness
  - Performance? Smaller code vs. imperfect decode

ISA-level Tradeoffs: Uniform Decode

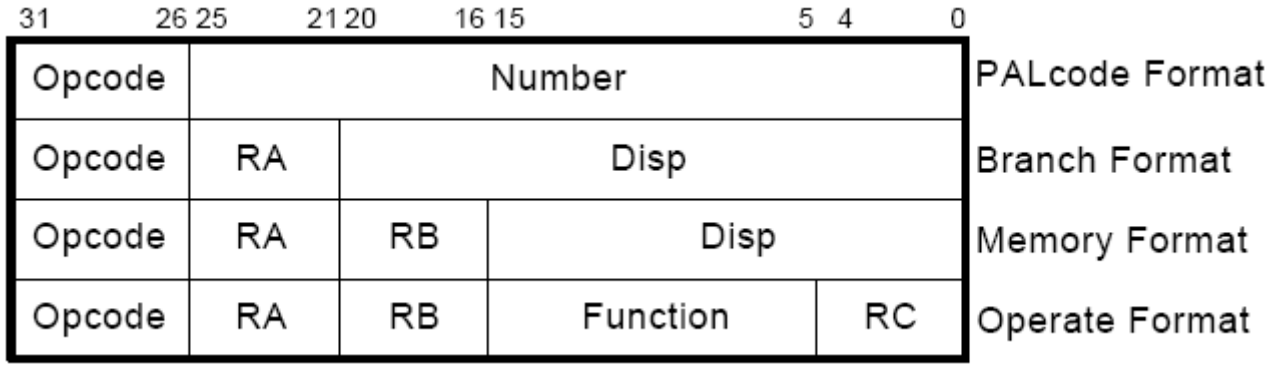
- Uniform decode: Same bits in each instruction correspond to the same meaning
  - Opcode is always in the same location
  - Ditto operand specifiers, immediate values, ...
  - Many "RISC" ISAs: Alpha, MIPS, SPARC
  - \+ Easier decode, simpler hardware
  - \+ Enables parallelism: generate target address before knowing the instruction is a branch
  - -- Restricts instruction format (fewer instructions?) or wastes space
- Non-uniform decode
  - E.g., opcode can be the 1st-7th byte in x86
  - \+ More compact and powerful instruction format
  - -- More complex decode logic (e.g., more adders to speculatively generate branch target)

x86 vs. Alpha Instruction Formats

- x86:



- Alpha:



ISA-level Tradeoffs: Number of Registers

- Affects:
  - Number of bits used for encoding register address
  - Number of values kept in fast storage (register file)
  - (uarch) Size, access time, power consumption of register file

- Large number of registers:

  + Enables better register allocation (and optimizations) by compiler → fewer saves/restores

  -- Larger instruction size

  -- Larger register file size

  -- (Superscalar processors) More complex dependency check logic