What is Instruction Set Architecture

- Instruction set architecture (ISA)
  - Abstract model of a computer
  - Describes *what* it does, rather than *how* it does
  - Describes low-level programmer’s view of a computer
  - Defines types of operations a computer carries out

- Attributes visible to the programmers
  - What are they?
What is Instruction Set Architecture

- We are interested in
  - Instructions: format and types
  - Operations:
    - How many ops?
    - What can they do?
    - How complex are they?
  - Data types
  - Addressing modes: the way instructions access data
  - Memory and registers: resources used by instructions
- MIPS microprocessor is used to study aspects of ISA

Instruction

- An instruction is an op-code followed by address(es)
  - Address means any address in system
  - Registers, memory locations, I/O ports, etc.
- Format
  - Layout of bits in an instruction
  - Usually more than one instruction format in an instruction set
- Length of instruction
- Length of op-code field(s)
- Style
  - Number of addresses
Instruction Types

- In general
  - Data transfer
  - Arithmetic
  - Logical
  - Shift
  - Flow control
  - I/O
  - :
  - etc.

Instruction Length

- Affected by and affects:
  - Memory size
  - Memory organization
  - Bus structure
  - CPU complexity
  - CPU speed
- Tradeoff between powerful instruction repertoire and saving space
Allocation of Bits

- Number of addressing modes
- Number of operands
- Register versus memory
- Number of register sets
- Address range

How Complex are Instructions

- Classification of processors
  - CISC (Complex instruction set computer)
    - Large instruction set
    - Variable-length instructions
    - Operations directly on contents of memory locations allowed
    - Powerful instructions, but difficult to optimize code
    - e.g., Motorola 68000, Intel x86, etc.
  - RISC (Reduced instruction set computer)
    - Smaller instruction set
    - Fixed-length instructions
    - Load/store processor – operations on contents of registers
    - Primitives, easy to optimize code, but longer program, but can be pipelined
      - e.g., PowerPC, Sun SPARC, MIPS, etc.
  - More on CISC vs. RISC later ...
Addressing Modes

- Where to find the data items, i.e., operands
- Fundamental addressing modes
  - Absolute: operand specifies memory location of data
  - Immediate: operand is actual value
  - Indirect: operand provides pointer (reference) of data
  - Register direct – operand specifies register number
  - Displacement/indexed
  - Auto-increment/auto-decrement
    - etc.

Registers

- Temporary storage, sketch pads in CPU
- Part of memory system, like main memory
- Register vs. main memory
  - Location – inside CPU
  - Faster access
  - Fewer registers
  - Shorter address

- Example
  - _____ bits needed to specify one of eight registers
  - 32 bits needed to select one out of _____ memory locations
Registers

- Design issues
  - Number and size
  - General purpose vs. specialized
  - User visible
  - Control and status

General-Purpose Registers

- General Purpose vs. Specialized
- General purpose
  - Increase flexibility and programmer options
  - Increase instruction size & complexity
- Specialized (w/ implicit spec. in opcode)
  - Smaller (faster) instructions, as it saves bits
  - Less flexibility
- Which is better?
  - No final and best answer
  - Trend toward specialized
General-Purpose Registers

- How many?
  - Usually between 8 and 32
- Fewer ⇒ more memory references
- More does not reduce memory references and takes up processor real estate
- RISC (e.g., Sparc):
  - Advantages of using hundreds of registers

General-Purpose Registers

- How big?
- Large enough to hold full address
- Large enough to hold full word
- Possible to combine two data registers
  - e.g., PDP-11: two registers to hold one long integer
**Last but not the least - Endian**

- Byte order problem
- What order do we read numbers that occupy more than one byte
  - e.g. (numbers in hex to make it easy to read)
    
    $12345678_{16}$ can be stored in 4 8-bit memory locations as follows

    | Address Value (1) | Address Value (2) |
    |------------------|------------------|
    | 184 12           | 184 78           |
    | 185 34           | 185 56           |
    | 186 56           | 186 34           |
    | 187 78           | 187 12           |

Is it a problem? Why?

**Endian**

- The problem is called Endian
- The system on the left has the most significant byte in the lowest address
  
  ⇒ This is called big-endian (*big* end first)
- The system on the right has the least significant byte in the lowest address
  
  ⇒ This is called little-endian
- Big-endian examples: M68000, Internet
- Little-endian example: Intel x86
- Is it a *problem*? Why?