Architecture

- *Architecture*, as applied to computer systems, refers to a formal specification of an interface in the system, including the logical behavior of resources managed via the interface.

- *Implementation* describes the actual embodiment of an architecture.

- *Abstraction levels* correspond to implementation layers, whether in hardware or software, each associated with its own interface or architecture.

- We are assuming the standard von Neumann architecture.

Important Interfaces

- Instruction Set Architecture (ISA)
  - Interface 4: User ISA.
  - Interface 3: System ISA.
- Application Binary Interface (ABI)
  - Interface 4: User ISA.
  - Interface 2: System calls.
- Application Programming Interface (API)
  - Interface 4: User ISA.
  - Interface 1: HLL library calls.

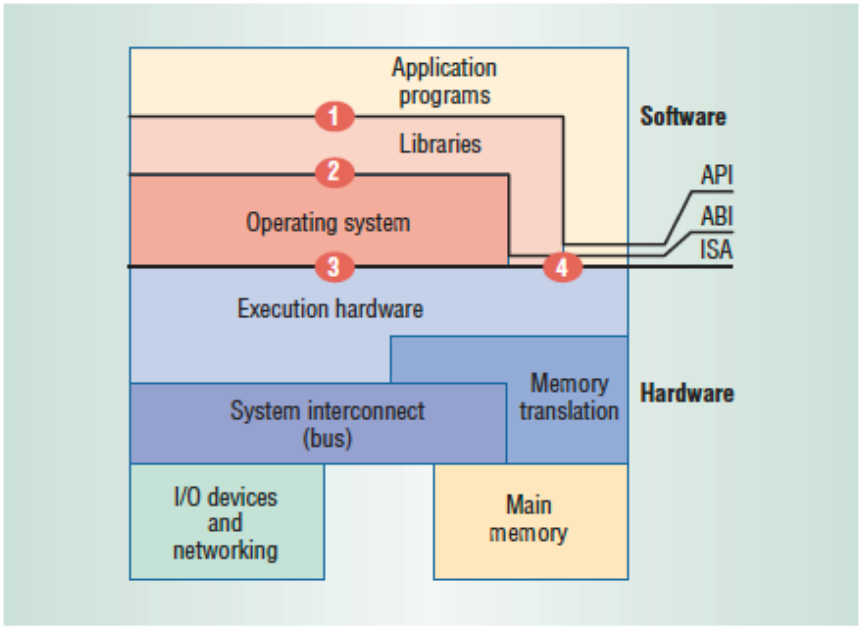

Figure 2. Computer system architecture. Key implementation layers communicate vertically via the instruction set architecture (ISA), application binary interface (ABI), and application programming interface (API).

From James E. Smith and Ravi Nair, "The Architecture of Virtual Machines", *Computer*, May 2005, 32-38. IEEE Computer Society.

Interface 4: The User ISA

The formal contract between execution hardware and software.

$\mathbb{A} = (\Sigma, \mathcal{I})$

Interface 4: The User ISA

The formal contract between execution hardware and software.

$\mathbb{A} = (\Sigma, \mathcal{I})$

- Σ: Universe of machine states.
  - $\sigma = (\sigma_M, \sigma_P) \in \Sigma$: An individual state.
  - σ_M: Memory state (value of each addressable memory unit in the user-accessible virtual address space).
  - σ_P: Processor state.

Interface 4: The User ISA

The formal contract between execution hardware and software.

$\mathbb{A} = (\Sigma, \mathcal{I})$

- Σ: Universe of machine states.
  - $\sigma = (\sigma_M, \sigma_P) \in \Sigma$: An individual state.
  - σ_M: Memory state (value of each addressable memory unit in the user-accessible virtual address space).
  - σ_P: Processor state.

- $\mathcal{I}$: Set of (user-level) machine instructions.
  - $\iota \in \mathcal{I}$: An individual instruction that transforms an input state to an output state. $\iota: \Sigma \to \Sigma$.

User-Level Processor State

- Essential
  - Program Counter (PC): $\mathcal{P}$.
- Typical
  - A small amount of fast storage close to processor (aka “registers”) whose units are accessed by name.
- Optional
  - Condition flags.

ISA Taxonomy

Based on the maximum number of operands *explicitly* specified in instructions.

ISA Taxonomy

Based on the maximum number of operands *explicitly* specified in instructions.

- 0-operand machine, aka “stack machine”
  - Operands come implicitly from the top items in a stack of values.
  - Java® Virtual Machine.

ISA Taxonomy

Based on the maximum number of operands *explicitly* specified in instructions.

- 0-operand machine, aka “stack machine”
  - Operands come implicitly from the top items in a stack of values.
  - Java® Virtual Machine.
- 1-operand machine, aka “accumulator machine”
  - Single implicit accumulator register that is both the left operand and the result. The right operand is specified explicitly.
  - Early machines, small microcontrollers (MOS 6502).

ISA Taxonomy

Based on the maximum number of operands *explicitly* specified in instructions.

- 0-operand machine, aka “stack machine”
  - Operands come implicitly from the top items in a stack of values.
  - Java® Virtual Machine.
- 1-operand machine, aka “accumulator machine”
  - Single implicit accumulator register that is both the left operand and the result. The right operand is specified explicitly.
  - Early machines, small microcontrollers (MOS 6502).
- 2-operand machine
  - Two named operands, one of which also serves as the result.
  - x86-64.

ISA Taxonomy

Based on the maximum number of operands *explicitly* specified in instructions.

- 0-operand machine, aka “stack machine”
  - Operands come implicitly from the top items in a stack of values.
  - Java® Virtual Machine.
- 1-operand machine, aka “accumulator machine”
  - Single implicit accumulator register that is both the left operand and the result. The right operand is specified explicitly.
  - Early machines, small microcontrollers (MOS 6502).
- 2-operand machine
  - Two named operands, one of which also serves as the result.
  - x86-64.
- 3-operand machine
  - Three named operands, two for inputs, one for result.
  - ARMv8.