Computer Architecture

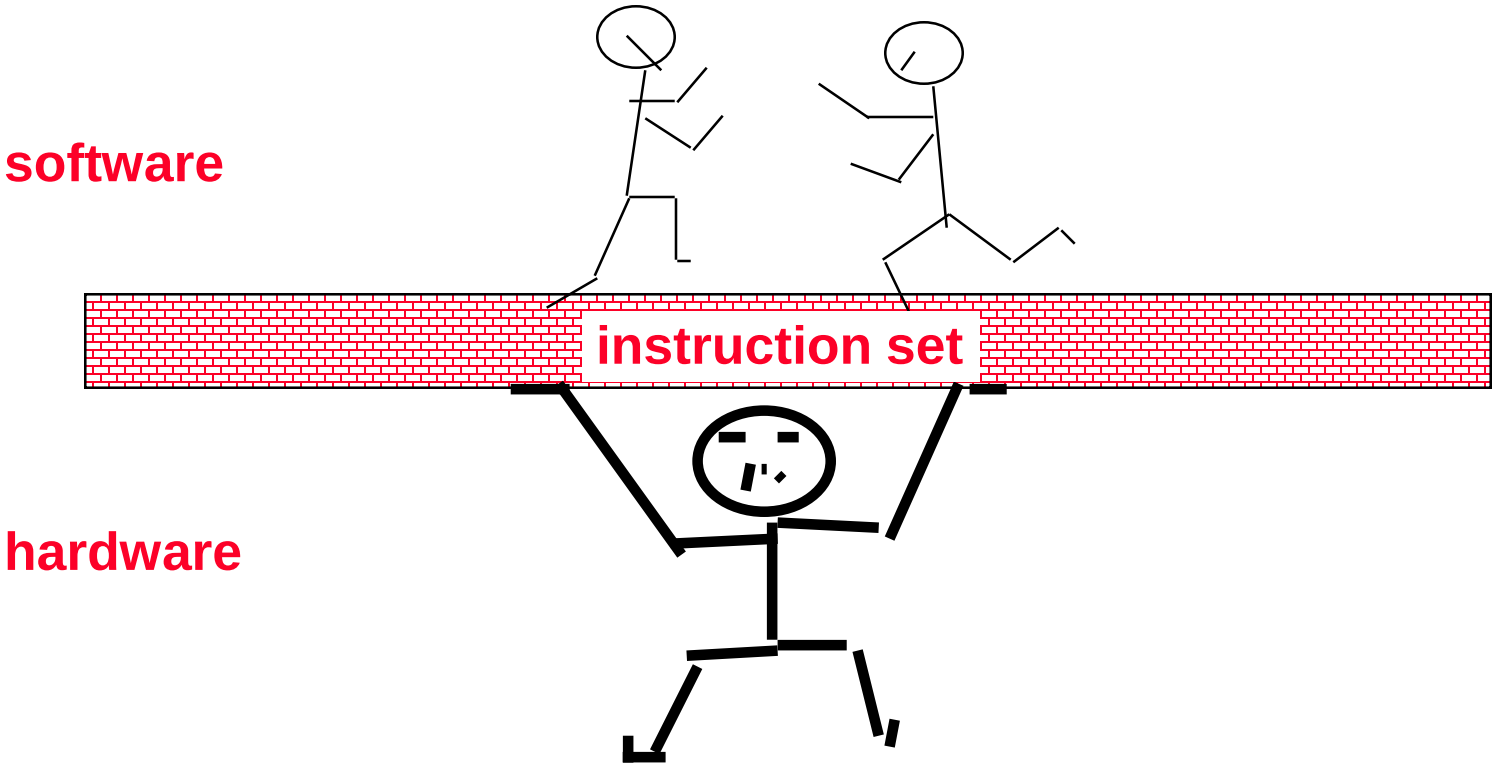
Chapter 2
Instructions: Language of the Computer

APPENDICES

ONLINE CONTENT

Instructions: Language of the Computer

The Instruction Set : a Critical Interface



Instruction : words

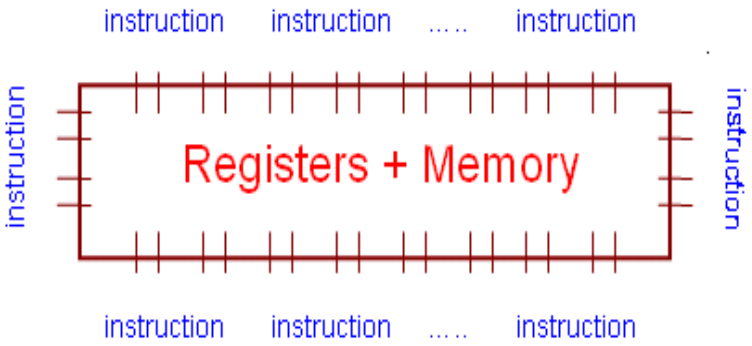
Instructions Set : vocabulary

Functions of Instruction Set

- Any Processor Must Be Able to Do at Least the Following Basic Functions
  - Arithmetic and Logic Operations
  - Data transfer to and from the memory
  - Conditional branches
    - Need a way to determine a condition
    - Need a target memory address to branch to if condition is met (or not met), go to next instruction otherwise
  - Jump and subroutine linkage (procedure call)
    - Need a large range of target memory address to branch
    - Procedure call needs a return address
- Additional functions: Examples
  - Move data between registers, to I/O, or to co-processor
  - Exception and Interrupt Instructions

Instruction Sets Classification

- An Abstract Data Type
  - Objects ≡ Registers & Memory
  - Operations ≡ Instructions
    - C code: `a = b + c`
    - MIPS 'code': `add a, b, c`
  - *Operand vs. register*



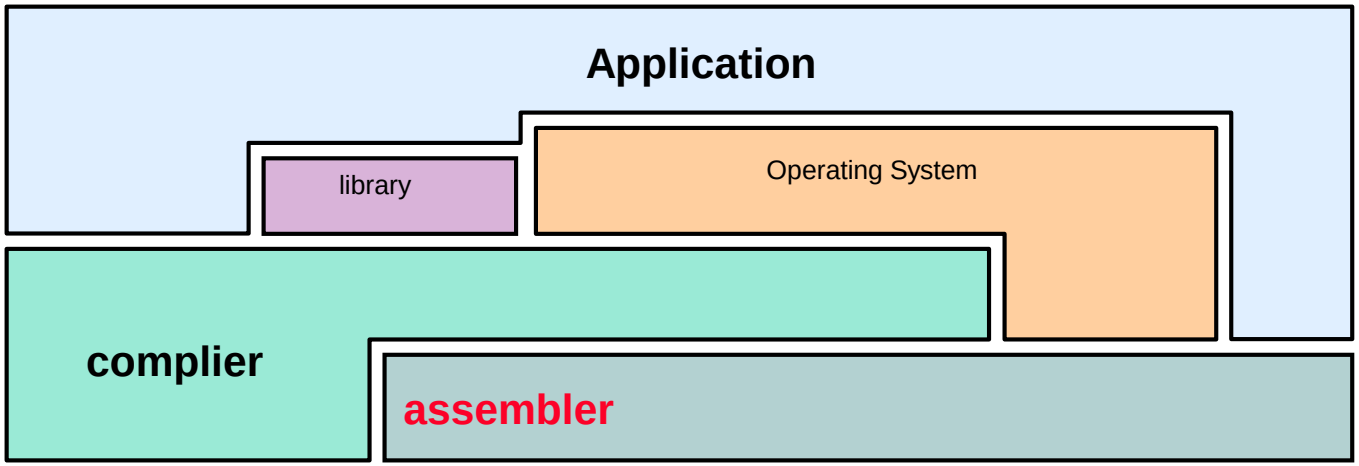
- Classifying Instruction Sets
  - Machine : number/kind of registers

    0 (stack machine) 1 (accumulator machine)

    small (2-6) general registers (e.g. 16, 32 or more)
  - # of addresses per instruction
    - 0 (stack machine)
    - 1 (accumulator machine)
    - 2 (general registers)
    - 3 (general registers)

Instruction Set Architecture (ISA)

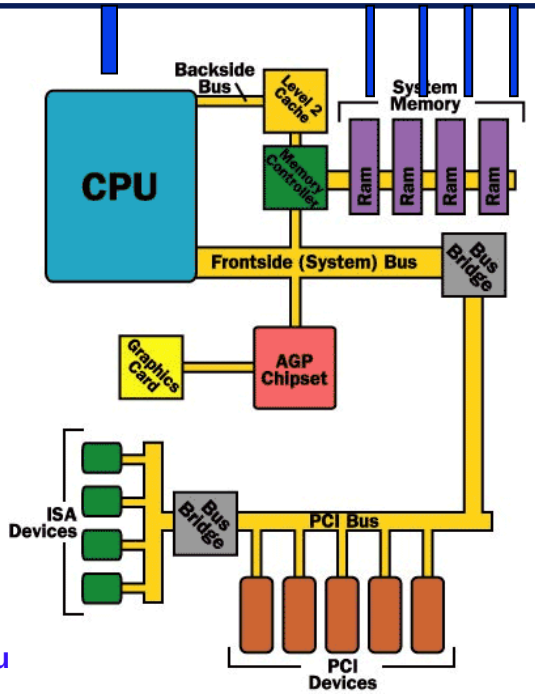
S/W



H/W

ISA
defines the permissible instructions

MIPS :
load/store,
arithmetic,
control flow,,,

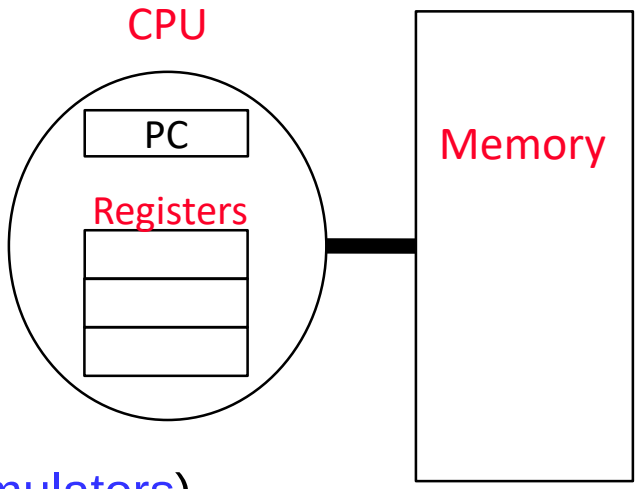


Goal of ISA Design

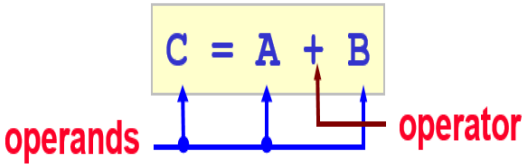
To allow high-performance and low-cost implementations while satisfying constraints imposed by applications including operating system and complier

Brief Historical Perspective on ISAs

- The ISA defines:
  - The system's state (*e.g.* registers, memory, program counter)
  - The instructions the CPU can execute
  - The effect that each of these instructions will have on the system state

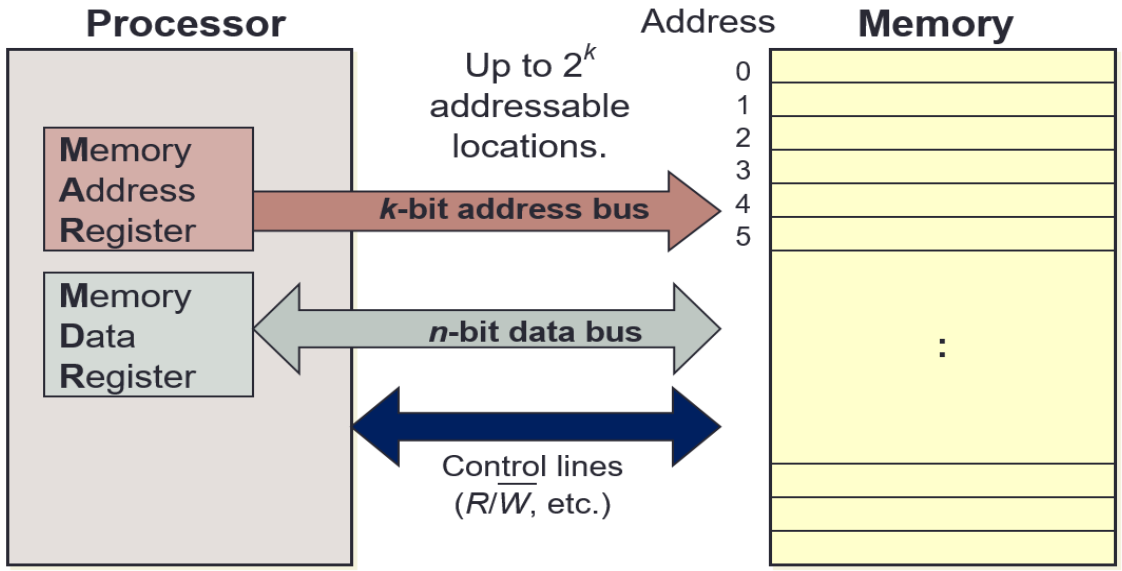


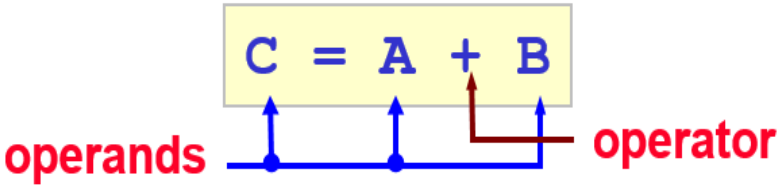
- General-purpose registers
  - Registers can be used for any purpose
  - E.g. MIPS, ARM, x86
- *Register-memory* architectures
  - One operand may be in memory (e.g. accumulators)
  - E.g. x86 processors, Motorola 68000
- *Register-register* architectures (aka load-store)
  - All operands ***must*** be in registers
  - E.g. MIPS, ARM



Memory Addressing Mode

- Memory Address and Content
  - Given *k*-bit address, the address space is of size 2^k
  - Each memory transfer consists of one word of ***n*** bits





- Addressing Mode:
  - Ways to specify an operand in an assembly language

Operations in Instructions Set

- Standard Operations

Data Movement	load (from memory) store (to memory) memory-to-memory move register-to-register move input (from I/O device) output (to I/O device) push, pop (to/from stack)
Arithmetic	integer (binary + decimal) or FP add, subtract, multiply, divide
Shift	shift left/right, rotate left/right
Logical	not, and, or, set, clear
Control flow	Jump (unconditional), Branch (conditional)
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronization	test & set (atomic r-m-w)
String	search, move, compare
Graphics	pixel and vertex operations, compression/decompression

Instruction Usage

- Designed versus actually used operations

Typical Instructions Provided by CISC

Data Movement	Load (from Memory) Store (to Memory) Memory-to-Memory Move Register-to-Register Move Input (from I/O Device) Output (to I/O Device) Push, Pop (to/from Stack)
Arithmetic	Integer (binary + decimal) or FP Add, Subtract, Multiply, Divide
Logical	not, and, or, set, clear
Shift	Shift Left/Right, Rotate Left/Right
Control (Jump/Branch)	Unconditional, Conditional
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronization	test & set (atomic read-modify-write)
String	search, translate

Top 10 80X86 Instructions

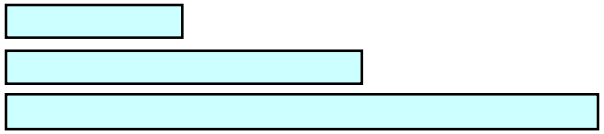
Rank	Instruction	Average % total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register-register	4%
9	call	1%
10	return	1%
	Total	96%

Simple instructions dominate instruction frequency and most of the instructions in CISC are not used.

Make these instructions fast!
Amdahl's law – make the common cases fast!

Instruction Formats : Length, Operation

- Variable-length instructions
  - 80x86: 1 ~ 17 bytes, Digital VAX: 1 ~ 54 bytes
  - require multi-step fetch and decode
  - more flexible (but complex) and compact IS



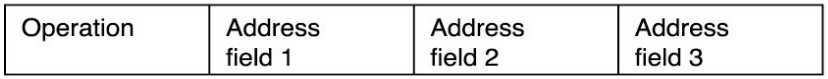
- Fixed-length instructions
  - Used in most RISC (Reduced Instruction Set Computers)
  - MIPS, PowerPC: Instructions are 4 bytes long.
  - Allow for easy fetch and decode.
  - Simplify pipelining and parallelism.
- Hybrid instructions : a mix of variable- and fixed-length instructions

- Opcode
  - unique code to specify the desired operation
- The type and size of the operands
  - Character (8 bits), half-word (eg: 16 bits), word (eg: 32 bits), single-precision floating point (eg: 1 word), double-precision floating point (eg: 2 words).

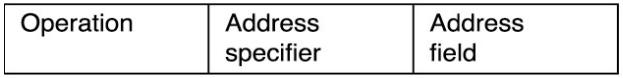
Encoding the Instruction Set

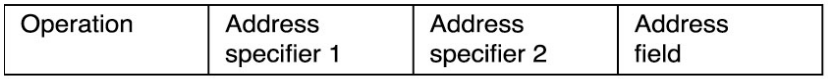
- How are instructions represented in binary format for execution by the processor?
- Things to be decided :
  - Number of registers
  - Number of addressing modes
  - Number of operands in an instruction
- Choices

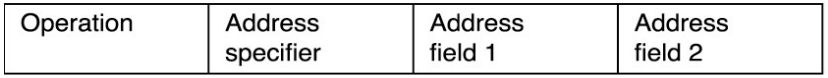
(a) Variable (e.g., VAX, Intel 80x86)



(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)







(c) Hybrid (e.g., IBM 360/70, MIPS16, Thumb, TI TMS320C54x)

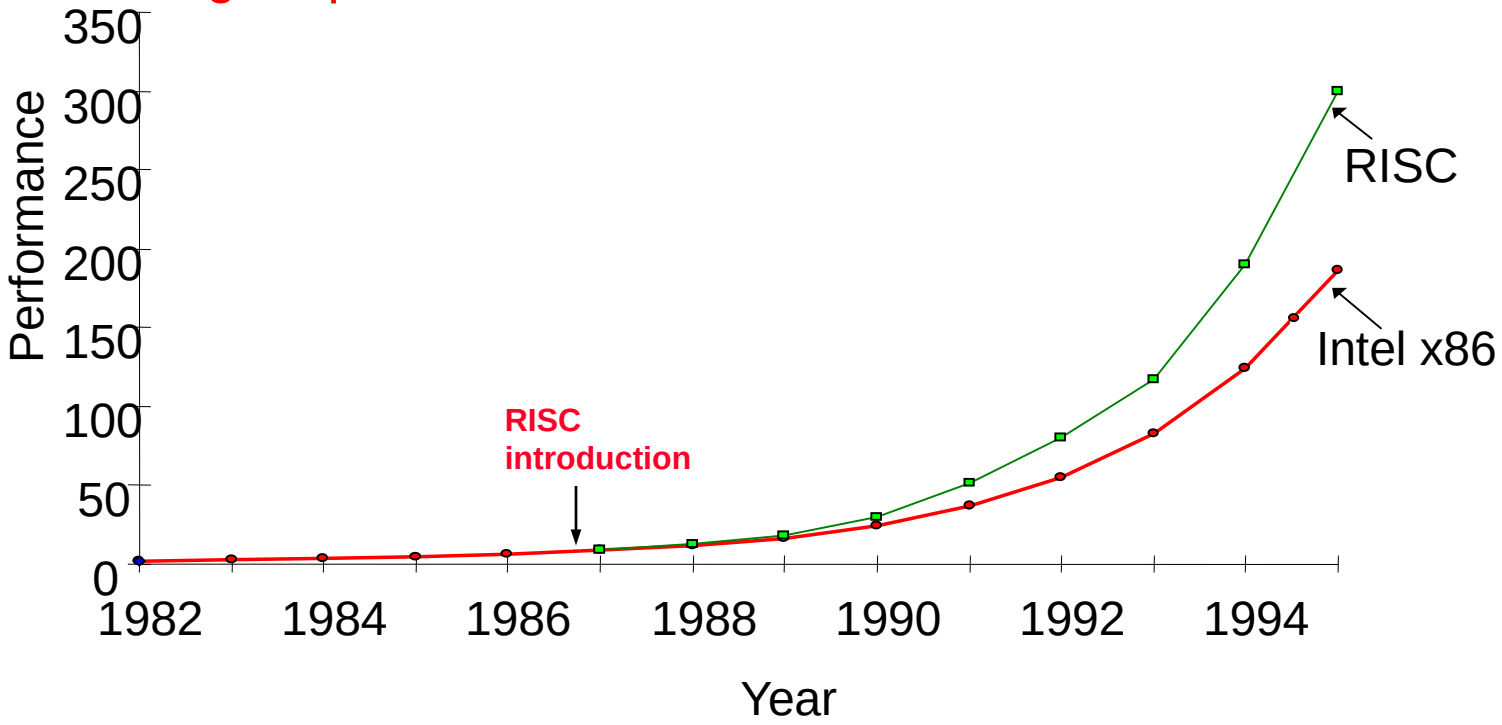
RISC vs. CISC

- Complex Instruction Set Computer (CISC) : x86-32(IA32)
  - > 1000 instructions, 1 to 15 bytes each
  - Single instruction performs complex operation
  - 10s of addressing modes
    - e.g. Mem[segment + reg + reg*scale + offset]
- Desktops/Servers

- Reduced Instruction Set Computer (RISC) : MIPS, ARM
  - Keep the instruction set small and simple, makes it easier to build/optimize hardware
    - ≈ 200 instructions, 32 bits each, 3 formats
    - all operands in registers
- Energy efficiency, Embedded Systems, Phones/Tablets

<참고> What is RISC and Why?

- RISC is an architecture design concept based on the principle that simpler hardware runs faster (e.g. MIPS). It uses smaller and regular instruction set to achieve performance, while relying on compiler technology to achieve functions used to done by complex instructions.
- Opposite to RISC is Complex Instruction Set Computer (CISC) (e.g. Intel x86). CISC believes complex instructions implemented in hardware can achieve higher performance.

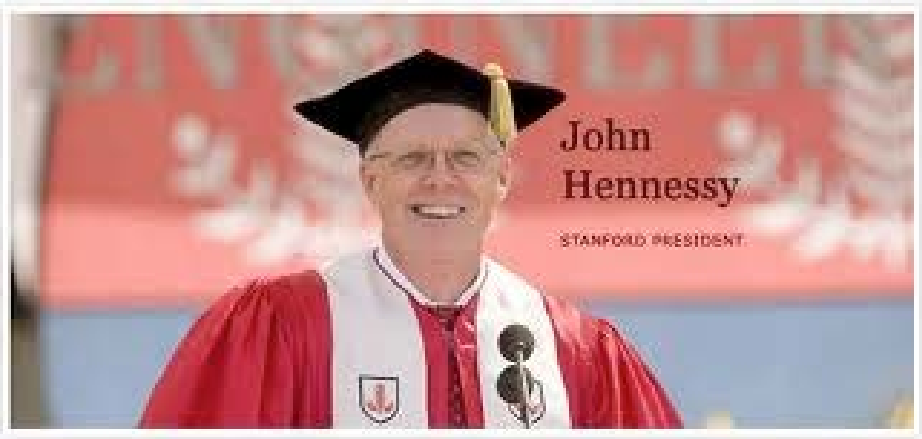


Courtesy D. Patterson

<참고> Reduced Instruction Set Computer (RISC)

- Dave Patterson
  - RISC Project, 1982
  - UC Berkeley
  - RISC-I: ½ transistors & 3x faster
  - Influences: Sun SPARC, namesake of industry

- John L. Hennessy
  - MIPS, 1981
  - Stanford
  - Simple pipelining, keep full
  - Influences: MIPS computer system, PlayStation, Nintendo



Mainstream ISAs

x86

Designer	Intel, AMD
Bits	16-bit, 32-bit and 64-bit
Introduced	1978 (16-bit), 1985 (32-bit), 2003 (64-bit)
Design	CISC
Type	Register-memory
Encoding	Variable (1 to 15 bytes)
Endianness	Little

Macbooks & PCs, Servers
(Core i3, i5, i7, M)
x86-64 Instruction Set

ARM architectures

Designer	ARM Holdings
Bits	32-bit, 64-bit
Introduced	1985; 31 years ago
Design	RISC
Type	Register-Register
Encoding	AArch64/A64 and AArch32/A32 use 32-bit instructions, T32 (Thumb-2) uses mixed 16- and 32-bit instructions. ARMv7 user-space compatibility[1]
Endianness	Bi (little as default)

Smartphone-like devices
(iPhone, iPad, Raspberry Pi)
ARM Instruction Set

MIPS

Designer	MIPS Technologies, Inc.
Bits	64-bit (32→64)
Introduced	1981; 35 years ago
Design	RISC
Type	Register-Register
Encoding	Fixed
Endianness	Bi

Digital home & networking equipment
(Blu-ray, PlayStation 2)
MIPS Instruction Set

* RISC-V: The *Free and Open RISC* Instruction Set Architecture https://riscv.org/

Summary

- Many possible implementations of the same ISA
  - x86 implementations: *8086 (c. 1978), 80186, 286, 386, 486, Pentium, Pentium Pro, Pentium-4, Core i7, AMD Athlon, AMD Opteron, Transmeta Crusoe, SoftPC*
  - MIPS implementations: *R2000, R4000, R10000, ...*
  - JVM: *HotSpot, PicoJava, ARM Jazelle, ...*
  - *RISC-V: RV32I, RV32E, RV64I, RV128I, …*
    - *Open-Source*

- ISA classes : Stack, Accumulator, and General purpose register

- Most current systems use general-purpose register(GPR) based ISA

Operations and Operands

- Examples pp. 65 ~ 69

a = b + c;

add a, b, c