

---

# Computer Architecture

## Chapter 2

### Instructions: Language of the Computer

## APPENDICES

### **A Assemblers, Linkers, and the SPIM Simulator A-2**

- A.1 Introduction A-3
- A.2 Assemblers A-10
- A.3 Linkers A-18
- A.4 Loading A-19
- A.5 Memory Usage A-20
- A.6 Procedure Call Convention A-22
- A.7 Exceptions and Interrupts A-33
- A.8 Input and Output A-38
- A.9 SPIM A-40
- A.10 MIPS R2000 Assembly Language A-45
- A.11 Concluding Remarks A-81
- A.12 Exercises A-82

### **B The Basics of Logic Design B-2**

- B.1 Introduction B-3
- B.2 Gates, Truth Tables, and Logic Equations B-4
- B.3 Combinational Logic B-9
- B.4 Using a Hardware Description Language B-20
- B.5 Constructing a Basic Arithmetic Logic Unit B-26
- B.6 Faster Addition: Carry Lookahead B-38
- B.7 Clocks B-48
- B.8 Memory Elements: Flip-Flops, Latches, and Registers B-50
- B.9 Memory Elements: SRAMs and DRAMs B-58
- B.10 Finite-State Machines B-67
- B.11 Timing Methodologies B-72
- B.12 Field Programmable Devices B-78
- B.13 Concluding Remarks B-79
- B.14 Exercises B-80

## ONLINE CONTENT



### **Graphics and Computing GPUs C-2**

- C.1 Introduction C-3
- C.2 GPU System Architectures C-7
- C.3 Programming GPUs C-12
- C.4 Multithreaded Multiprocessor Architecture C-25
- C.5 Parallel Memory System C-36
- C.6 Floating Point Arithmetic C-41
- C.7 Real Stuff: The NVIDIA GeForce 8800 C-46
- C.8 Real Stuff: Mapping Applications to GPUs C-55
- C.9 Fallacies and Pitfalls C-72
- C.10 Concluding Remarks C-76
- C.11 Historical Perspective and Further Reading C-77



### **Mapping Control to Hardware D-2**

- D.1 Introduction D-3
- D.2 Implementing Combinational Control Units D-4
- D.3 Implementing Finite-State Machine Control D-8
- D.4 Implementing the Next-State Function with a Sequencer D-22
- D.5 Translating a Microprogram to Hardware D-28
- D.6 Concluding Remarks D-32
- D.7 Exercises D-33



### **A Survey of RISC Architectures for Desktop, Server, and Embedded Computers E-2**

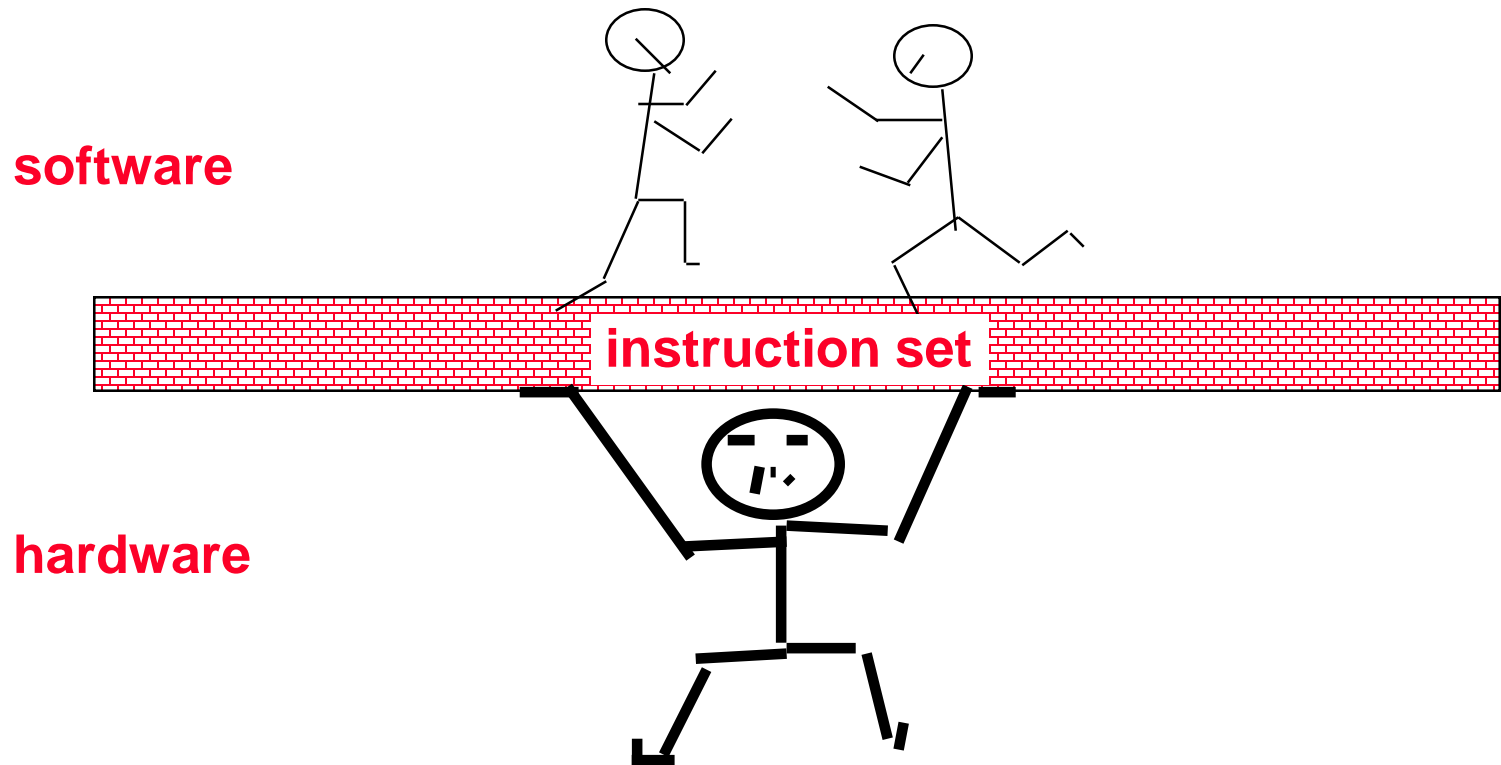
- E.1 Introduction E-3
- E.2 Addressing Modes and Instruction Formats E-5
- E.3 Instructions: The MIPS Core Subset E-9
- E.4 Instructions: Multimedia Extensions of the Desktop/Server RISCs E-16
- E.5 Instructions: Digital Signal-Processing Extensions of the Embedded RISCs E-19
- E.6 Instructions: Common Extensions to MIPS Core E-20
- E.7 Instructions Unique to MIPS-64 E-25
- E.8 Instructions Unique to Alpha E-27
- E.9 Instructions Unique to SPARC v9 E-29
- E.10 Instructions Unique to PowerPC E-32
- E.11 Instructions Unique to PA-RISC 2.0 E-34
- E.12 Instructions Unique to ARM E-36
- E.13 Instructions Unique to Thumb E-38
- E.14 Instructions Unique to SuperH E-39

# Instructions: Language of the Computer

---

- 2.1 Introduction 62
- 2.2 Operations of the Computer Hardware 63
- 2.3 Operands of the Computer Hardware 66
- 2.4 Signed and Unsigned Numbers 73
- 2.5 Representing Instructions in the Computer 80
- 2.6 Logical Operations 87
- 2.7 Instructions for Making Decisions 90
- 2.8 Supporting Procedures in Computer Hardware 96
- 2.9 Communicating with People 106
- 2.10 MIPS Addressing for 32-Bit Immediates and Addresses 111
- 2.11 Parallelism and Instructions: Synchronization 121
- 2.12 Translating and Starting a Program 123
- 2.13 A C Sort Example to Put It All Together 132
- 2.14 Arrays versus Pointers 141
- 2.15 Advanced Material: Compiling C and Interpreting Java 145
- 2.16 Real Stuff: ARMv7 (32-bit) Instructions 145
- 2.17 Real Stuff: x86 Instructions 149
- 2.18 Real Stuff: ARMv8 (64-bit) Instructions 158
- 2.19 Fallacies and Pitfalls 159
- 2.20 Concluding Remarks 161
- 2.21 Historical Perspective and Further Reading 163
- 2.22 Exercises 164

# The Instruction Set : a Critical Interface



*Instruction : words*

*Instructions Set : vocabulary*

# Functions of Instruction Set

---

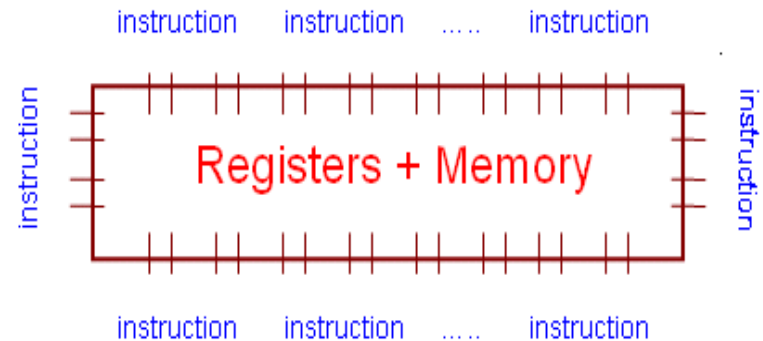
- ❑ Any Processor **Must Be Able to Do at Least** the Following Basic Functions
  - Arithmetic and Logic Operations
  - Data transfer to and from the memory
  - Conditional branches
    - Need a way to determine a condition
    - Need a target memory address to branch to if condition is met (or not met), 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  $\equiv$  **Registers & Memory**
- Operations  $\equiv$  **Instructions**
  - C code: `a = b + c`
  - MIPS 'code': `add a, b, c`

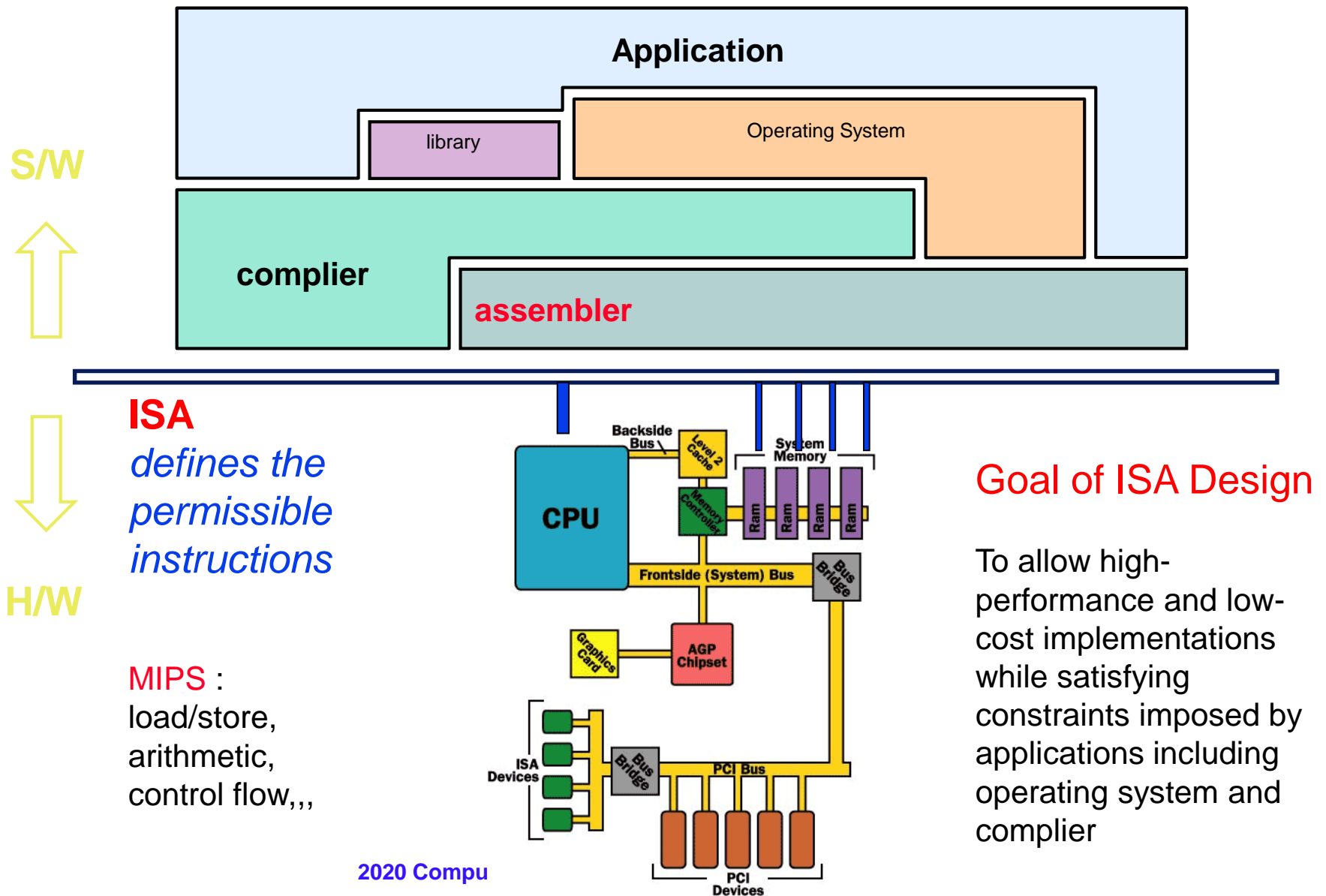
❖ *Operand vs. register*



## ❑ 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)



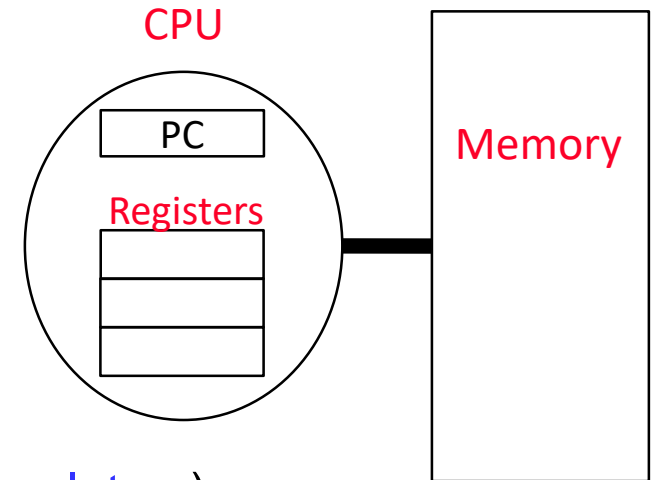
# Brief Historical Perspective on ISAs

## ❑ 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

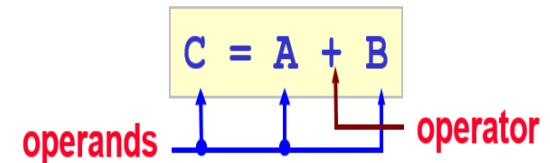


## ❑ Register-memory architectures

- One operand may be in memory (e.g. **accumulators**)
- E.g. x86 processors, Motorola 68000

## ❑ Register-register architectures (aka **load-store**)

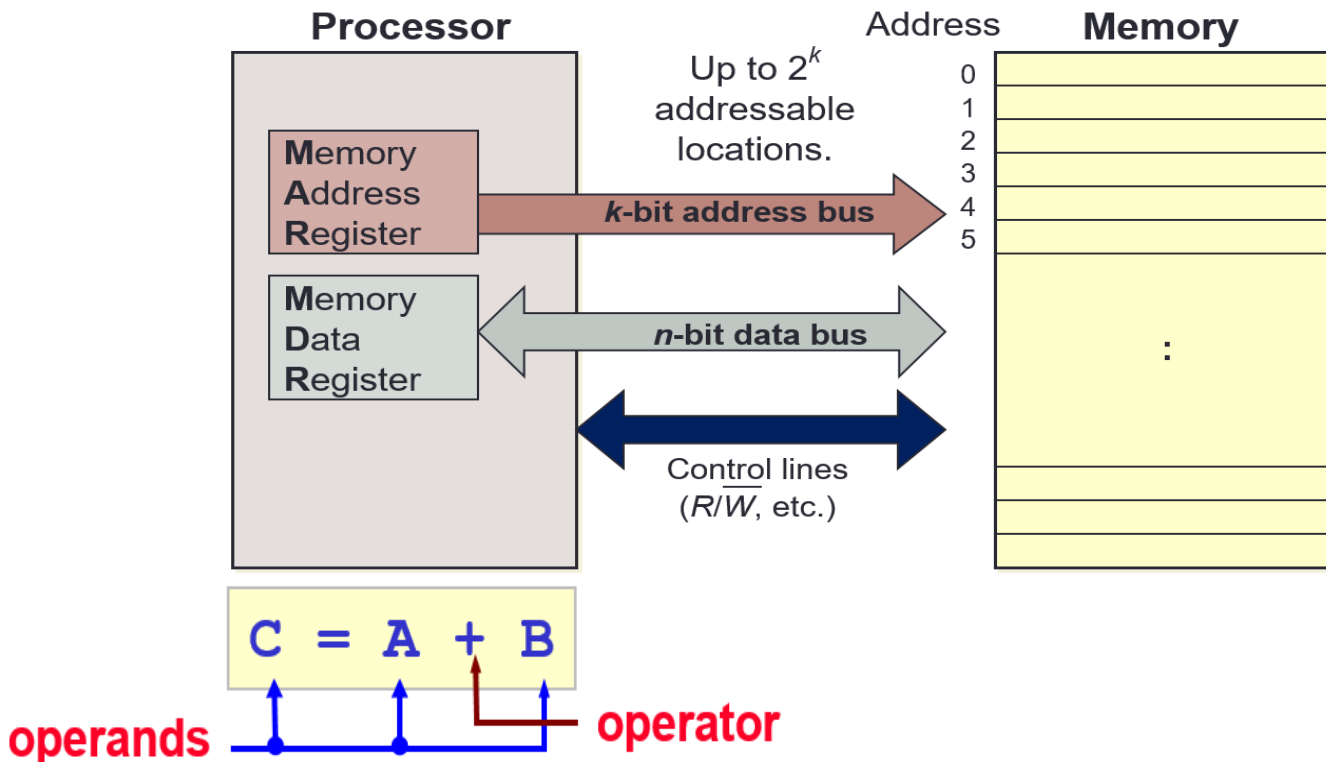
- All operands **must** be in registers
- E.g. MIPS, ARM





# 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

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

### Top 10 80X86 Instructions

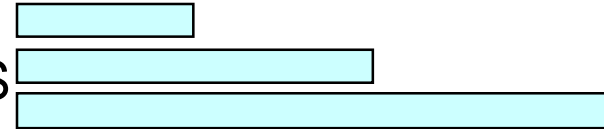
Rank	Instruction	Average % total executed
1	load	22%
2	conditional branch	20%
3	compare	16%
4	store	12%
5	add	8%
6	and	6%
7	sub	5%
8	move register- register	4%
9	call	1%
10	return	1%
	Total	96%

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

# Instruction Formats : Length, Operation

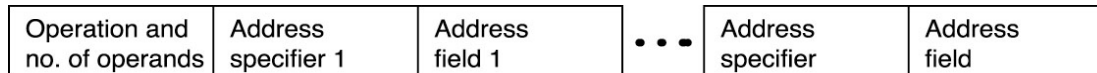
---

- **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).

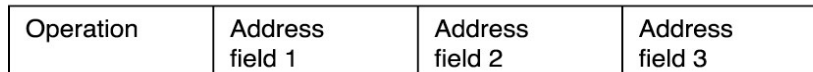


# Encoding the Instruction Set

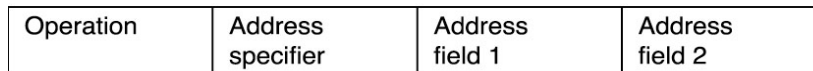
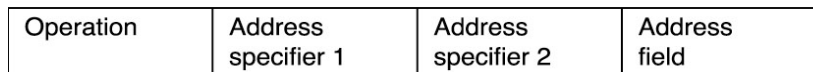
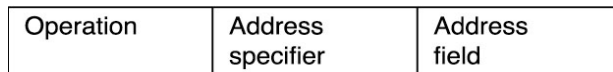
- How are instructions represented in binary format for execution by the processor?
- Things to be decided :
  - Number of registers
  - Number of addressing modes
  - Number of operands in an instruction
- Choices



(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

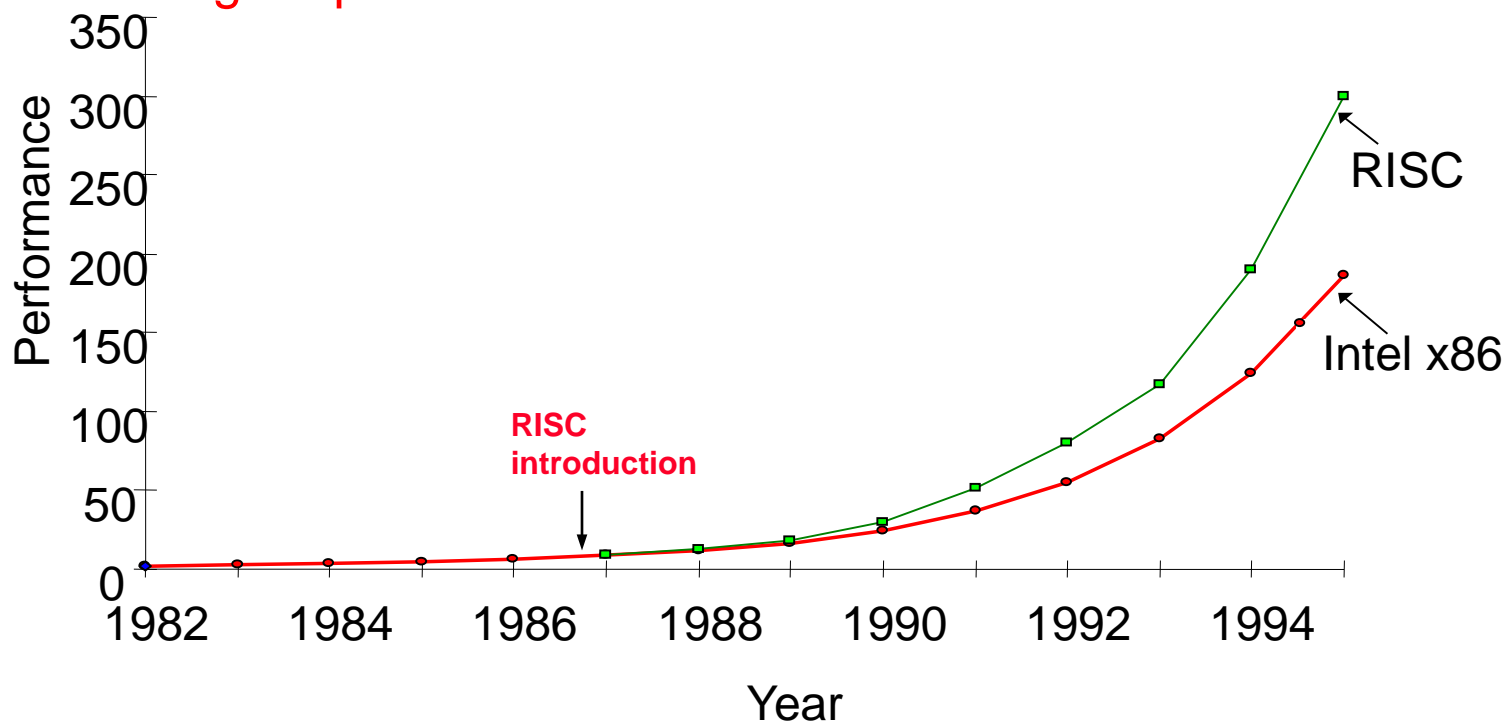
## ❑ Reduced Instruction Set Computer (RISC) : MIPS, ARM

- Keep the instruction set small and simple, makes it easier to build/optimize hardware
  - $\approx$  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 be 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.**



## <참고> Reduced Instruction Set Computer (RISC)

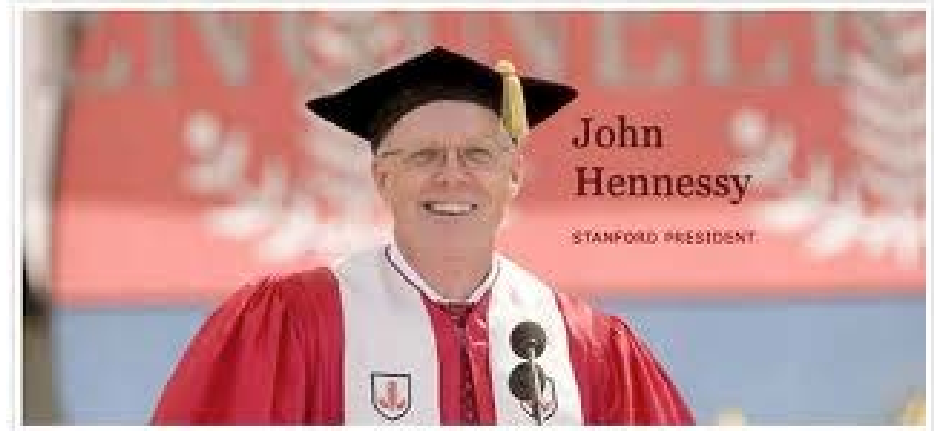
### ❑ Dave Patterson

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



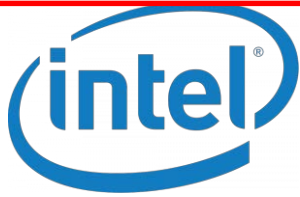
### ❑ John L. Hennessy

- MIPS, 1981
- Stanford
- Simple pipelining, keep full
- Influences: MIPS computer system, PlayStation, Nintendo





# Mainstream ISAs



## x86

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

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



## ARM architectures

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

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



## MIPS

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