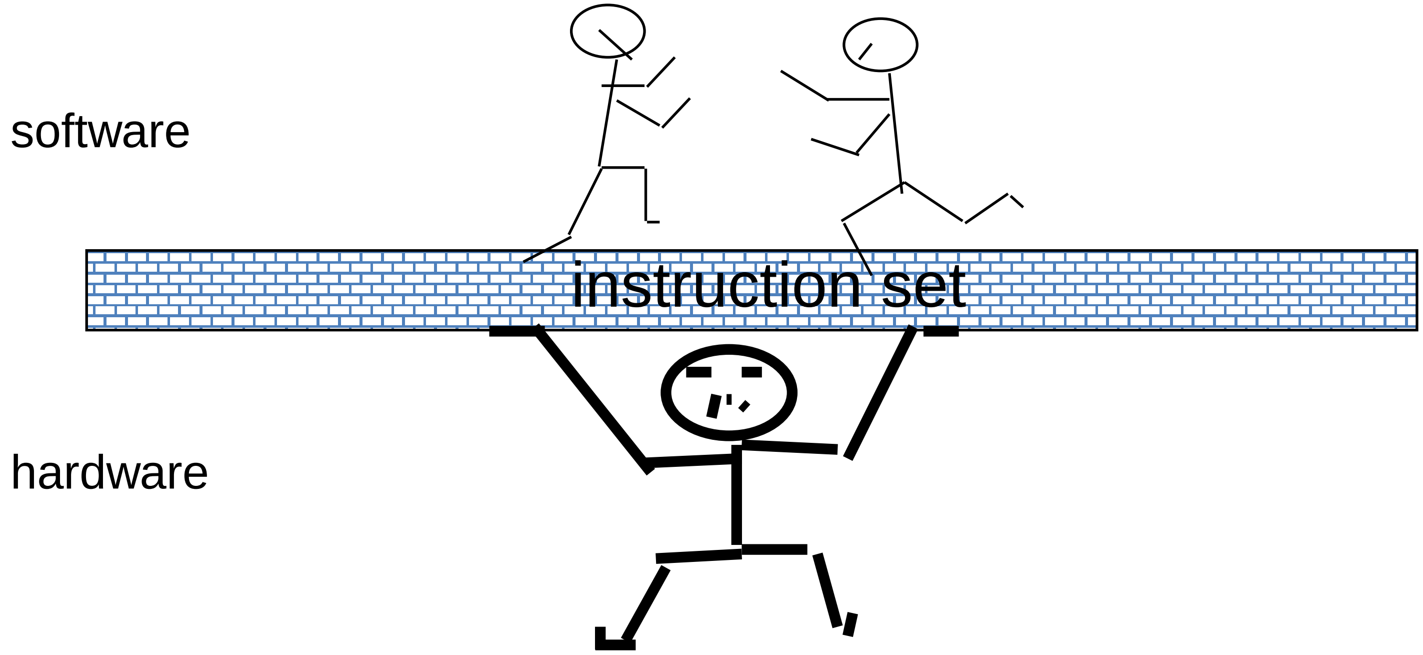
# COMP4611: Design and Analysis of Computer Architectures

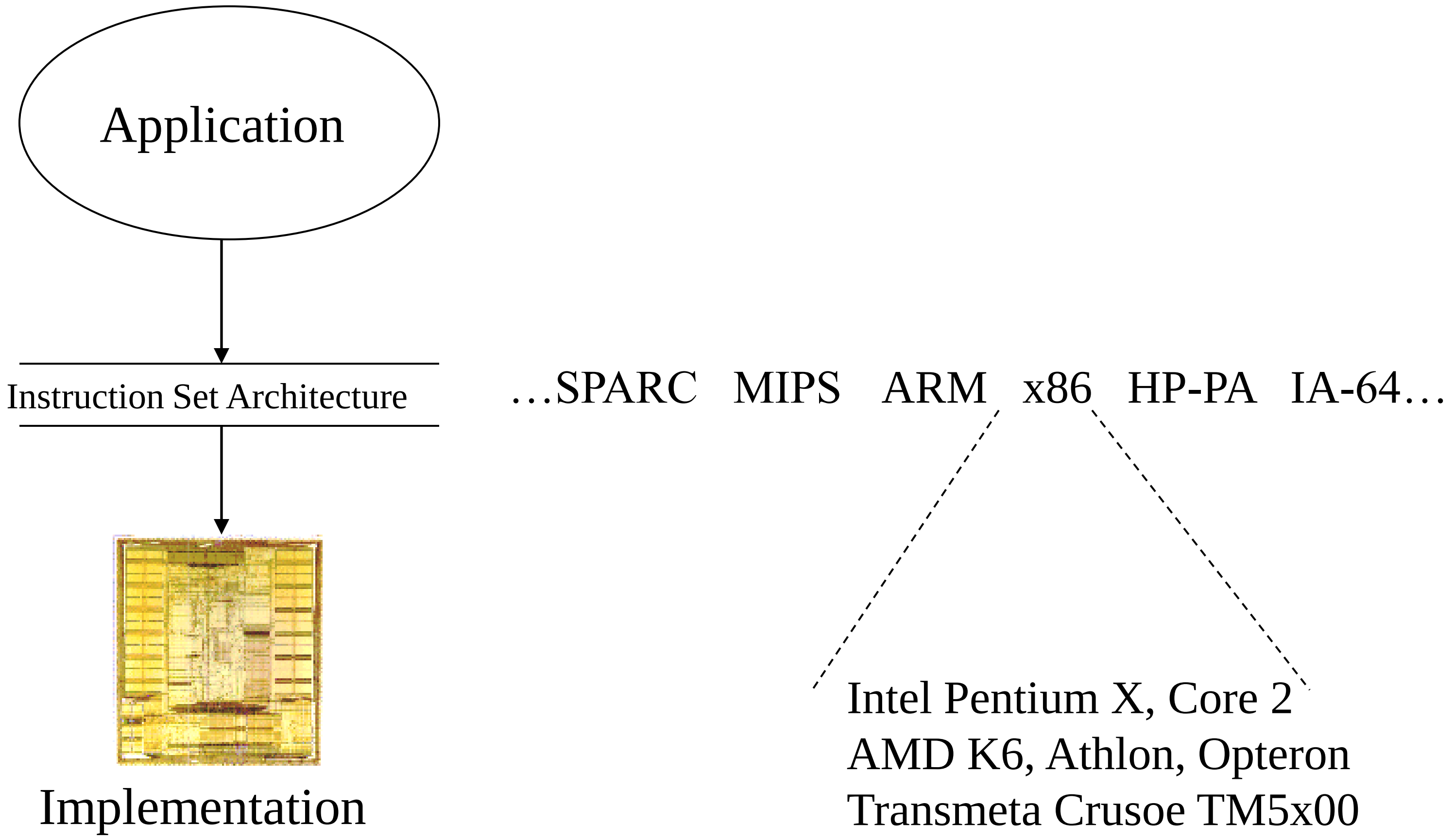
## Instruction Set Architecture

## Instruction Set Architecture (ISA)

- In order to use the hardware of a computer, we must *speak* its language.
- The words of a computer language are called *instructions*, and its vocabulary is called an *instruction set*.
- Instruction set of a computer: the portion of the computer visible to the assembly level programmer or to the compiler writer.



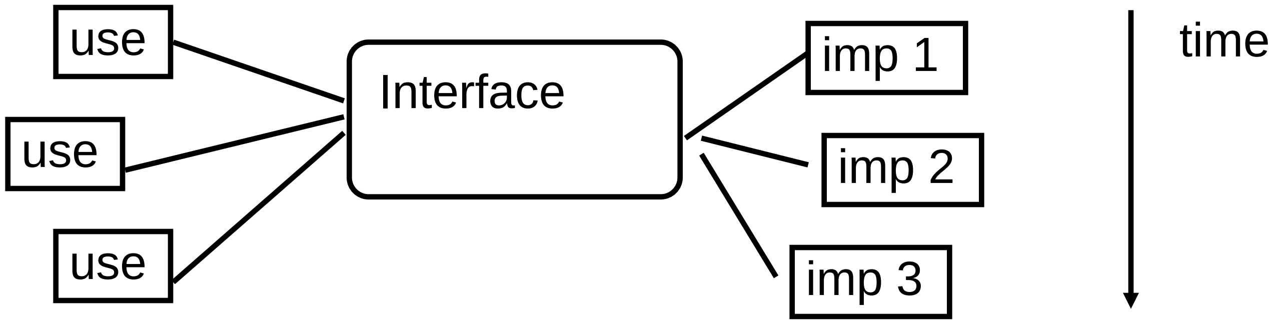
# Instruction Set Architecture



# Interface Design

A good interface

- lasts through many implementations (portability, compatibility)
- is used in many different ways (generality)
- provides *convenient* functionality to higher levels
- permits an *efficient* implementation at lower levels



## Instruction Set Architecture

- Strong influence on cost/performance
  - Remember the Execution Time equation?

$$\text{CPU time} = \frac{\text{Seconds}}{\text{Program}} = \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Seconds}}{\text{Cycle}}$$

  - ISA has a huge impact on programs and their performance!

- New ISAs are rare, but new (extended) versions are not
  - 16-bit, 32-bit and 64-bit X86 versions
- Longevity is a strong function of marketing prowess

# Instruction Set

- What is an instruction set?
  - Specification of a computer model and a set of instructions running on it
  - Each instruction is directly executed by the computer hardware (usually the CPU).
- How is it represented?
  - By a binary format, typically bits, bytes, words.
  - Word size is typically 16, 32, 64 bits today.
  - Length options:
    - Fixed – each instruction encoded in same size
    - Variable – half-word, whole word, multiple word instructions are possible

## Instruction Set Encoding

Considerations affecting instruction set encoding:

- To support registers and addressing modes provided
- The impact of the size of the register and addressing mode fields on the average instruction size and on the average program.
- To encode instructions into lengths that will be easy to handle in the implementation.
  - Usually, the instruction length should be a multiple of bytes.

## *Instruction Format*

- Fixed
  - **Operation, address specifier 1, address specifier 2, address specifier 3.**
  - **MIPS, SPARC, Power PC.**
- Variable
  - **Operation & # of operands, address specifier1, …, specifier n**.
  - **VAX**
- Hybrid
  - Intel x86
  - **operation, address specifier, address field.**
  - **Operation, address specifier 1, address specifier 2, address field.**
  - **Operation, address field, address specifier 1, address specifier 2.**
- **Summary:**
  - **If code size is most important, use variable format.**
  - **If performance is most important, use fixed format.**

# Three Examples of Instruction Set Encoding

| | | | | | |
|---|---|---|---|---|---|
| Operations & no of operands | Address specifier 1 | Address field 1 | ● ● ● | Address specifier n | Address field n |

**Variable: VAX (1-53 bytes)**

| | | | |
|---|---|---|---|
| Operation | Address field 1 | Address field 2 | Address field3 |

**Fixed: DLX, MIPS, PowerPC, SPARC**

| | | |
|---|---|---|
| Operation | Address Specifier | Address field |

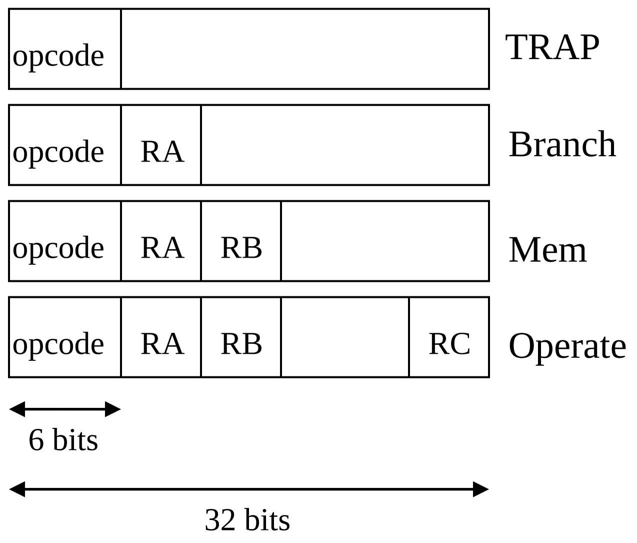
| | | | |
|---|---|---|---|
| Operation | Address Specifier 1 | Address Specifier 2 | Address field |

| | | | |
|---|---|---|---|
| Operation | Address Specifier | Address field 1 | Address field 2 |

**Hybrid : IBM 360/370, Intel 80x86**

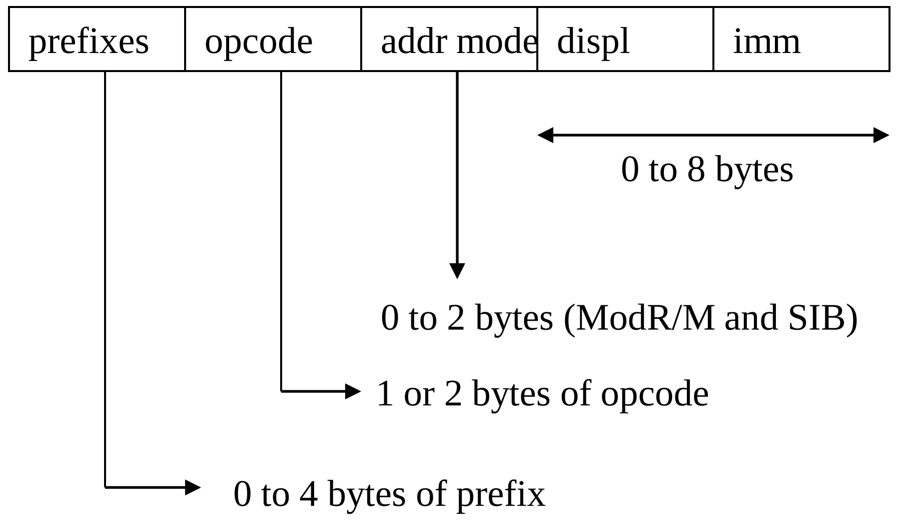
# Instruction Formats

Alpha (fixed length)



VAX-11: 1—53 bytes

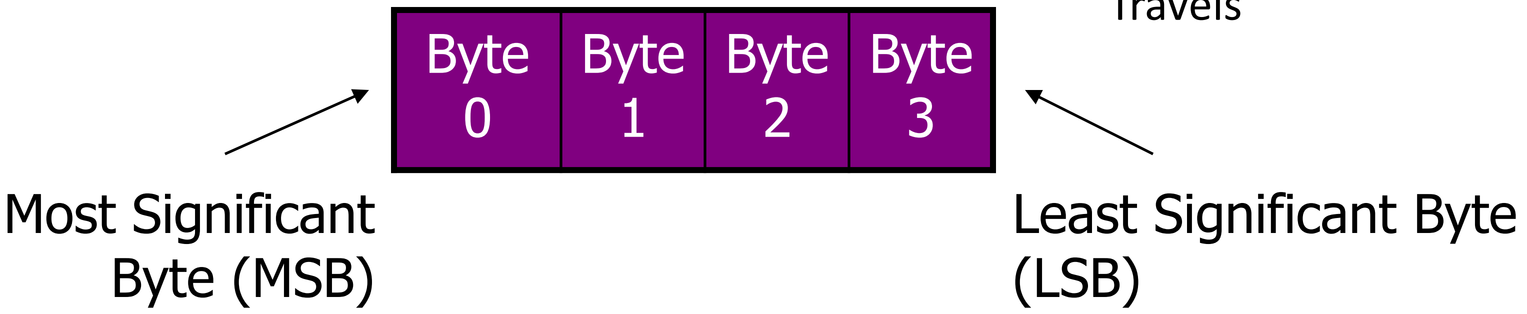
x86 (variable length)



# Instruction Format

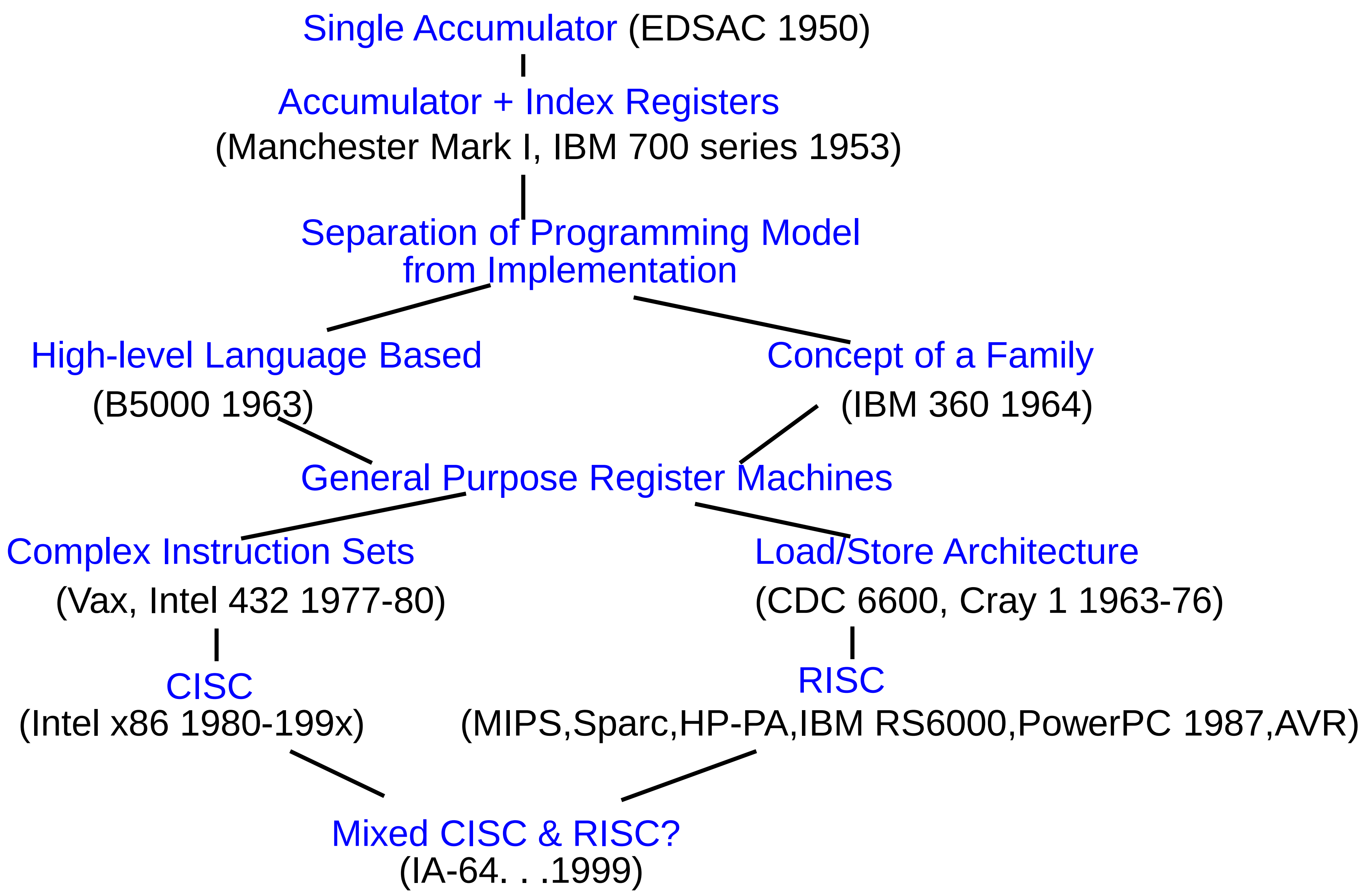
- Endian-ness: byte ordering in multibyte data
- Little Endian vs. Big Endian

To know how important this issue is, read to "Gulliver's Travels"



| Memory Address | +0 | +1 | +2 | +3 | |
|---|---|---|---|---|---|
| Big Endian | Byte 0 | Byte 1 | Byte 2 | Byte 3 | MSB in the lowest (first) memory address |
| Little Endian | Byte 3 | Byte 2 | Byte 1 | Byte 0 | LSB in the lowest (first) memory address |

# Evolution of Instruction Sets


Single Accumulator (EDSAC 1950)
Accumulator + Index Registers
(Manchester Mark I, IBM 700 series 1953)
Separation of Programming Model from Implementation
High-level Language Based
(B5000 1963)
Concept of a Family
(IBM 360 1964)
General Purpose Register Machines
Complex Instruction Sets
(Vax, Intel 432 1977-80)
Load/Store Architecture
(CDC 6600, Cray 1 1963-76)
CISC
(Intel x86 1980-199x)
RISC
(MIPS,Sparc,HP-PA,IBM RS6000,PowerPC 1987,AVR)
Mixed CISC & RISC?
(IA-64. . .1999)

# Basic ISA Classes

- **Accumulator:**
  - 1 Address: add A (acc $\leftarrow$ acc + Mem[A]).
- **Stack:**
  - 0 address: add (tos $\leftarrow$ tos + second of stack).
- **General Purpose Register (GPR):**
  - A fast register file (collection of registers)
  - Operands of arithmetic operations are explicitly specified
    - Operands can be memory locations or registers
  - 2 addresses: add A, B EA(A) $\leftarrow$EA(A)+EA(B)
  - 3 addresses: add A, B, C EA(A) $\leftarrow$EA(B)+EA(C)

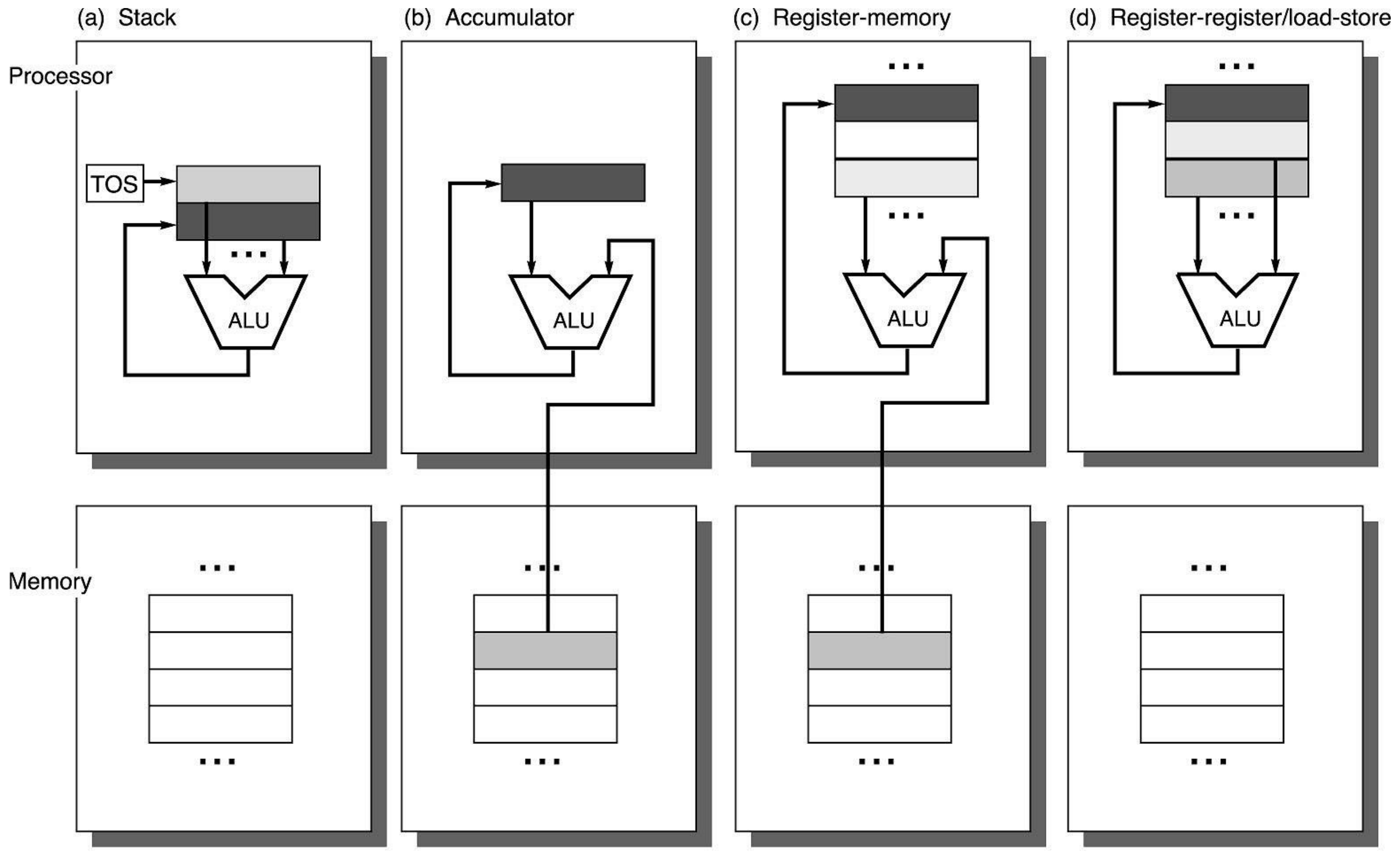
# Basic ISA Classes

- Classes of GPR computers
  - Register-register, register-memory, memory-memory
- Load/Store (register-register):
  - ALU operations : No memory reference.
  - 3 addresses: add R1, R2, R3  $R1 \leftarrow R2 + R3$

    load R1, R2  $R1 \leftarrow Mem[R2]$

    store R1, R2  $Mem[R1] \leftarrow R2$
- Comparison: Bytes per Instruction? Number of Instructions? Cycles per instruction?

# Operand Locations in Four ISA Classes



## Comparison of ISA Classes

- Code Sequence for C = A+B
  - A, B and C are variables stored in memory

| Stack | Accumulator | Register (register-Mem) | Register (load/store) |
|---|---|---|---|
| Push A | Load A | Load R1, A | Load R1, A |
| Push B | Add B | Add R1, B | Load R2, B |
| Add | Store C | Store C, R1 | Add R3, R1, R2 |
| Pop C | | | Store C, R3 |

- Memory efficiency? How long is an instruction? How many instructions? Code density?

**What's the minimum number of instructions we need for a computer?**

## What may an ISA want to provide?

If you are the architect for a new ISA, how many/what instructions/addressing modes/features would you add to the ISA?

- Computers should use general purpose registers
- Computer should use a load-store architecture
- The number of instructions is not a limitation any more

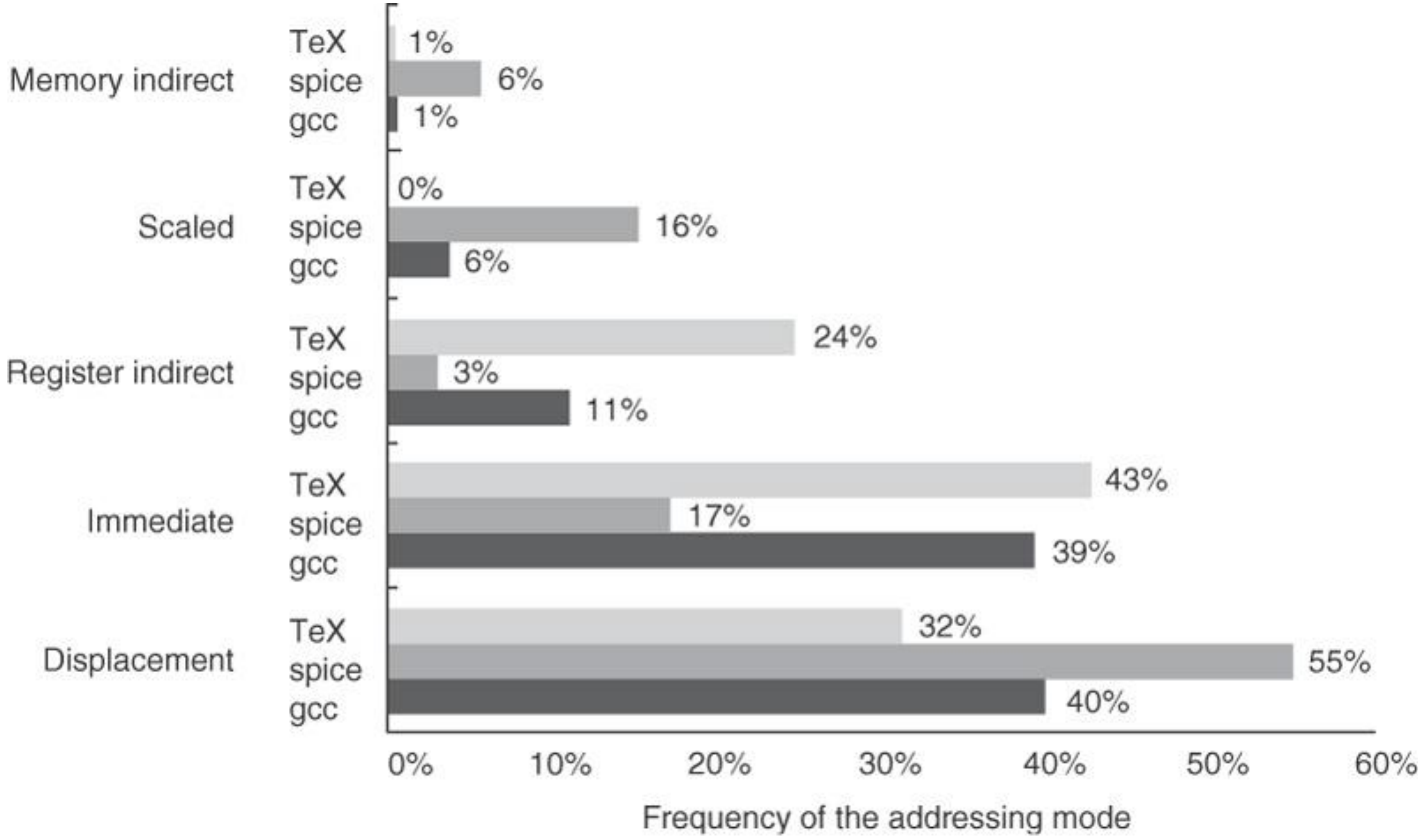
## Addressing Modes

- An important aspect of ISA design
  - Has major impact on both the HW complexity and the instruction count
  - HW complexity affects the CPI and the cycle time
- Basically a set of mappings
  - From address specified to address used
  - Address used = effective address
  - Effective address may go to memory or to register file
  - Effective address generation is an important focus

## Typical Memory Addressing Modes

| Addressing Mode | Sample Instruction | Meaning |
|---|---|---|
| Register | Add R4, R3 | Regs[R4] ←Regs[R4] + Regs[R3] |
| Immediate | Add R4, #3 | Regs[R4] ←Regs[R4] + 3 |
| Displacement | Add R4, 10 (R1) | Regs[R4] ←Regs[R4]+Mem[10+Regs[R1]] |
| Indirect | Add R4, (R1) | Regs[R4] ←Regs[R4]+ Mem[Regs[R1]] |
| Indexed | Add R3, (R1 + R2) | Regs [R3] ←Regs[R3]+Mem[Regs[R1]+Regs[R2]] |
| Absolute | Add R1, (1001) | Regs[R1] ←Regs[R1] + Mem[1001] |
| Memory indirect | Add R1, @ (R3) | Regs[R1] ←Regs[R1] + Mem[Mem[Regs[R3]]] |
| Autoincrement | Add R1, (R2) + | Regs[R1] ←Regs[R1] + Mem[Regs[R2]] |
| | | Regs[R2] ←Regs[R2] + d |
| Autodecrement | Add R1, - (R2) | Regs [R2] ←Regs[R2] -d |
| | | Regs{R1] ←Regs[Regs[R1] +Mem[Regs[R2]] |
| Scaled | Add R1, 100 (R2) [R3] | Regs[R1] ←Regs[R1] + |
| | | Mem[100+Regs[R2]+Regs[R3]*d] |

## Utilization of Memory Addressing Modes



# Addressing Modes Usage Example

For 3 programs running on VAX ignoring direct register mode:

| | |
|---|---|
| Displacement | 42% avg, 32% to 55% |
| Immediate: | 33% avg, 17% to 43% |
| Register deferred (indirect): | 13% avg, 3% to 24% |
| Scaled: | 7% avg, 0% to 16% |
| Memory indirect: | 3% avg, 1% to 6% |
| Misc: | 2% avg, 0% to 3% |

75%

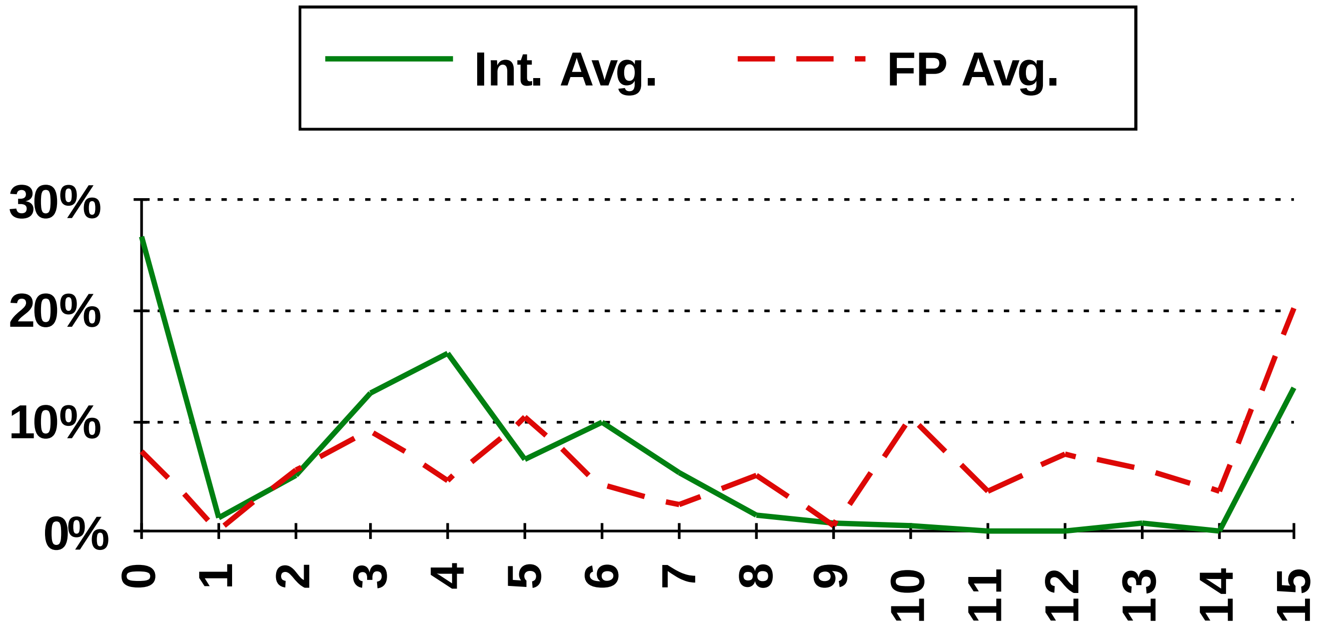
88%

75% displacement & immediate
88% displacement, immediate & register indirect.

Observation: In addition to Register direct, Displacement, Immediate, Register Indirect addressing modes are important.

# Displacement Address Size Example

Avg. of 5 SPECint programs vs. avg. 5 SPECfp programs

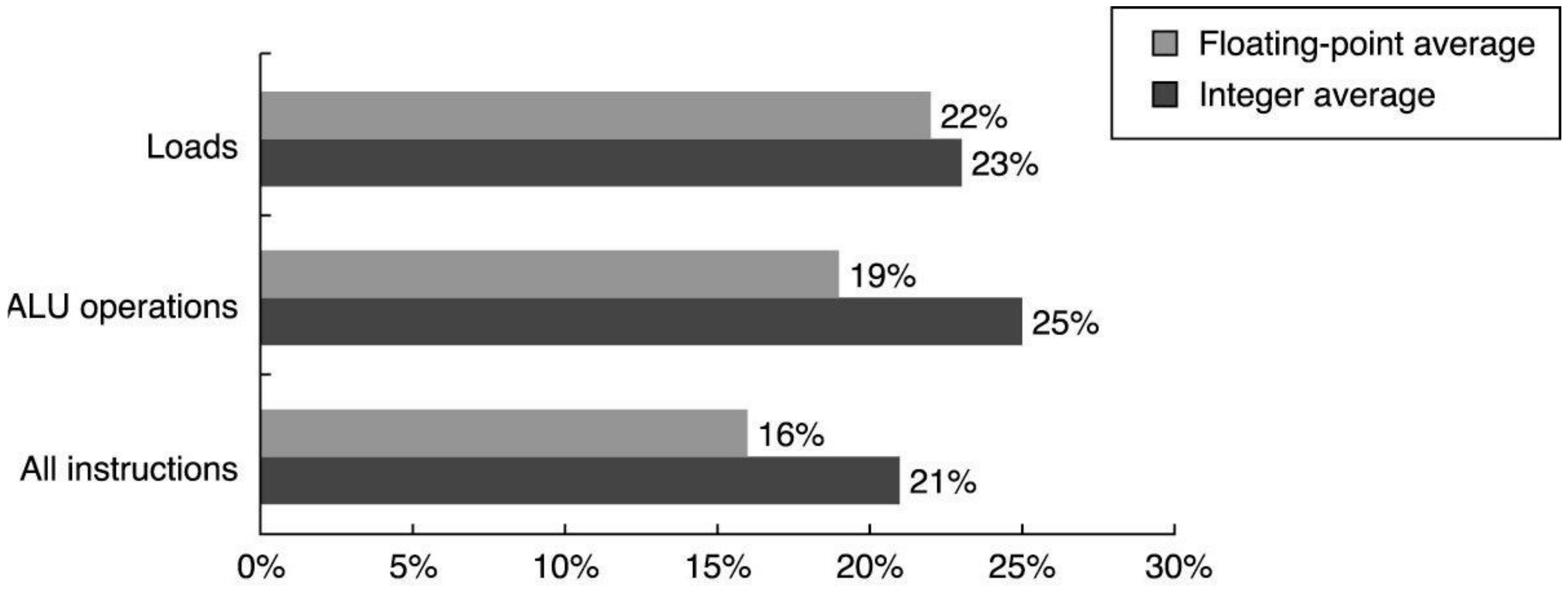


Displacement Address Bits Needed

1% of addresses > 16-bits

12 - 16 bits of displacement needed

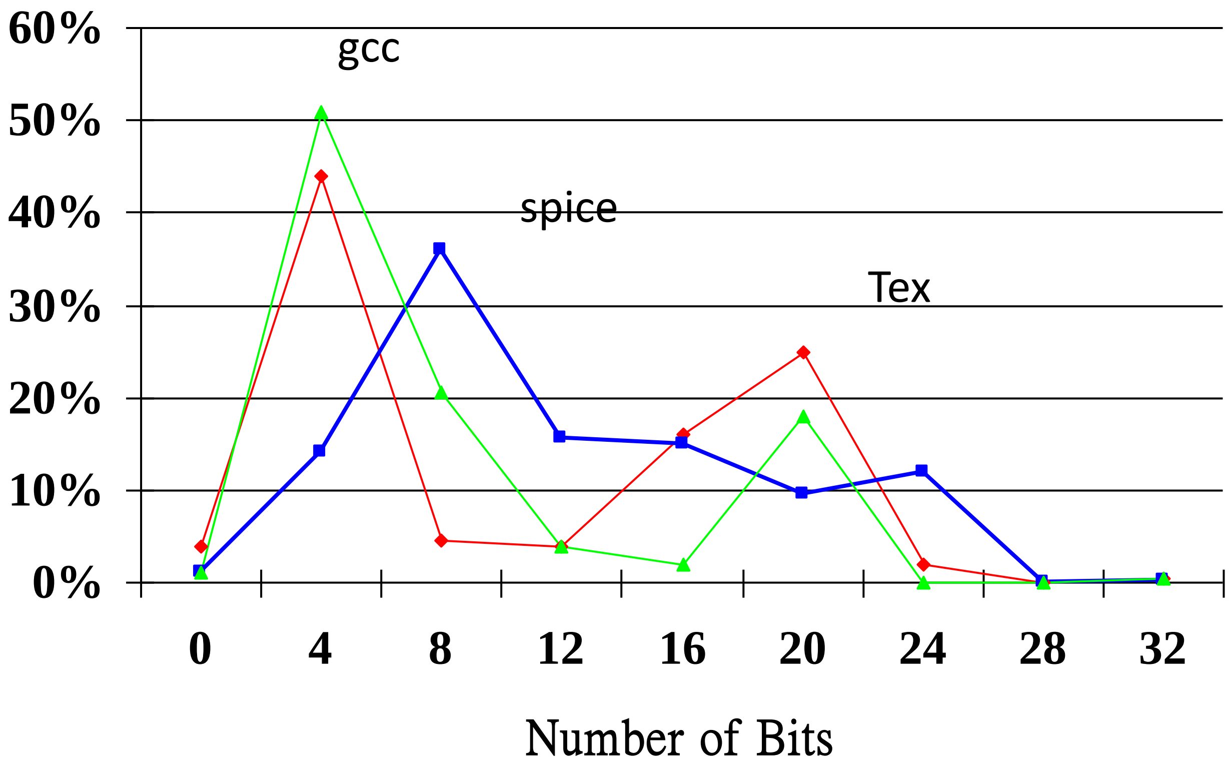
## Immediate Addressing Mode



About one quarter of data transfers and ALU operations have an immediate operand for SPEC CPU2000 programs.

# Immediate Addressing Mode

- 50% to 60% fit within 8 bits
- 75% to 80% fit within 16 bits



## Addressing Mode Summary

- Important data addressing modes
  - Displacement
  - Immediate
  - Register Indirect
- Displacement size should be 12 to 16 bits.
- Immediate size should be 8 to 16 bits.

## Instruction Operations

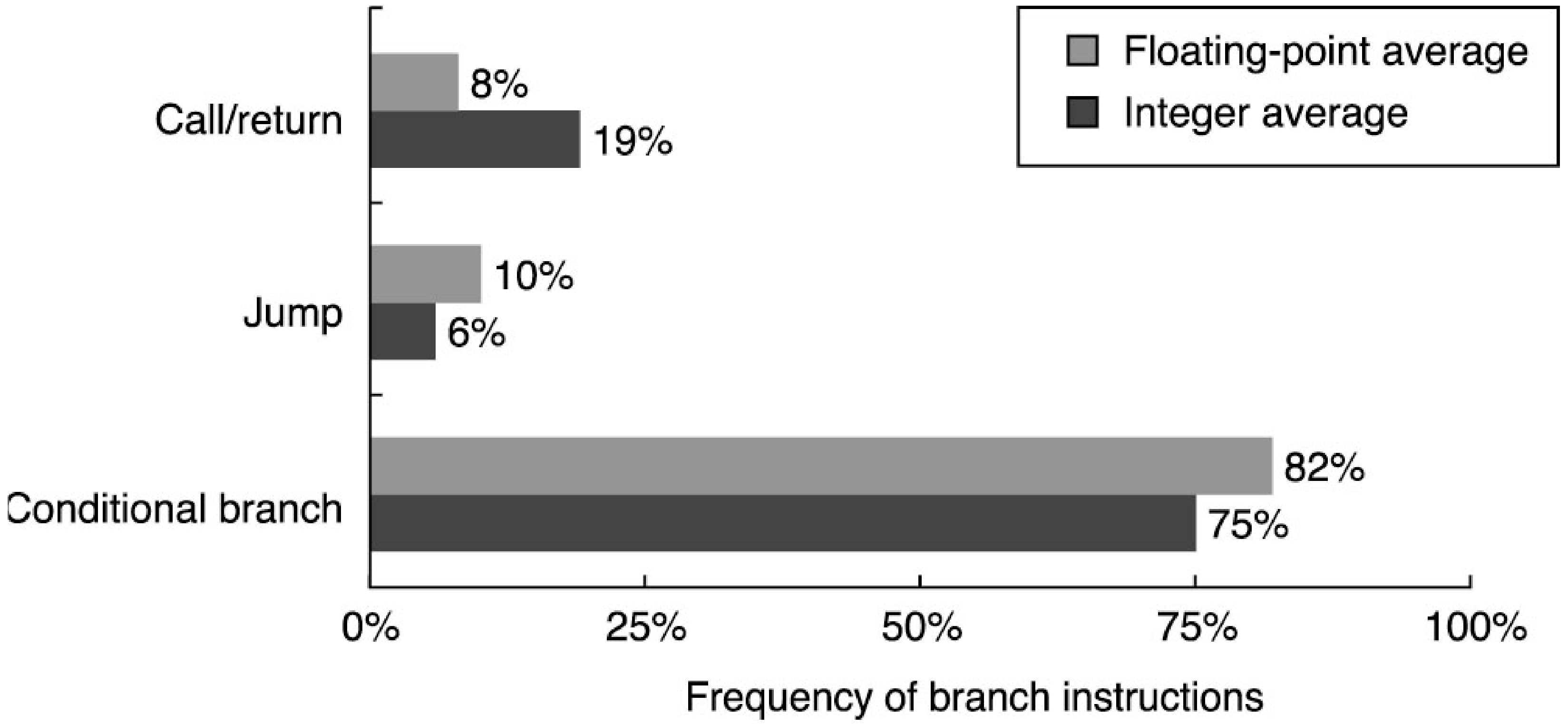
- Arithmetic and Logical:
  - **add, subtract, and , or, etc.**
- Data transfer:
  - **Load, Store, etc.**
- Control
  - **Jump, branch, call, return, trap, etc.**
- Synchronization:
  - **Test & Set.**
- String:
  - **string move, compare, search.**

## Instruction Usage Example: Top 10 Intel X86 Instructions

| Rank | instruction | Integer Average Percent 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% |

*Observation: Simple instructions dominate instruction usage frequency.*

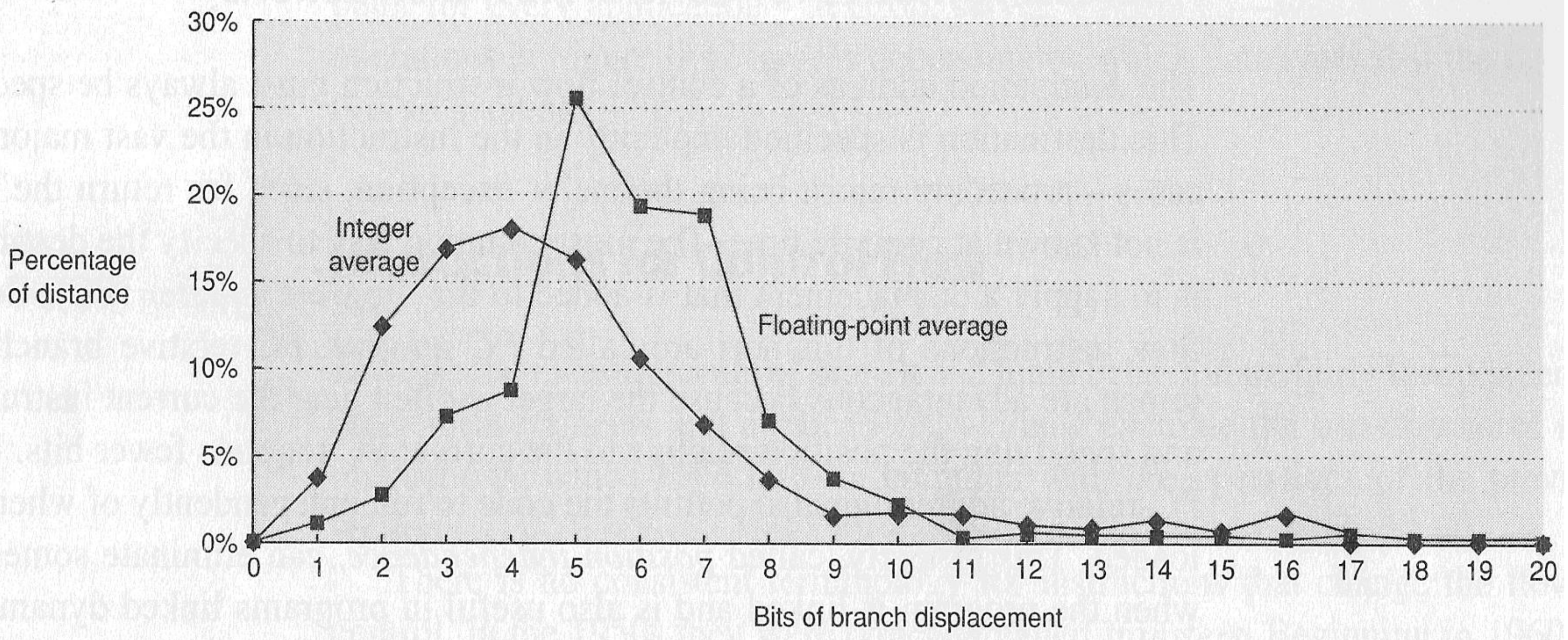
## Instructions for Control Flow



Breakdown of control flow instructions into three classes: calls or returns, jumps and conditional branches for SPEC CPU2000 programs.

# Conditional Branch Distance

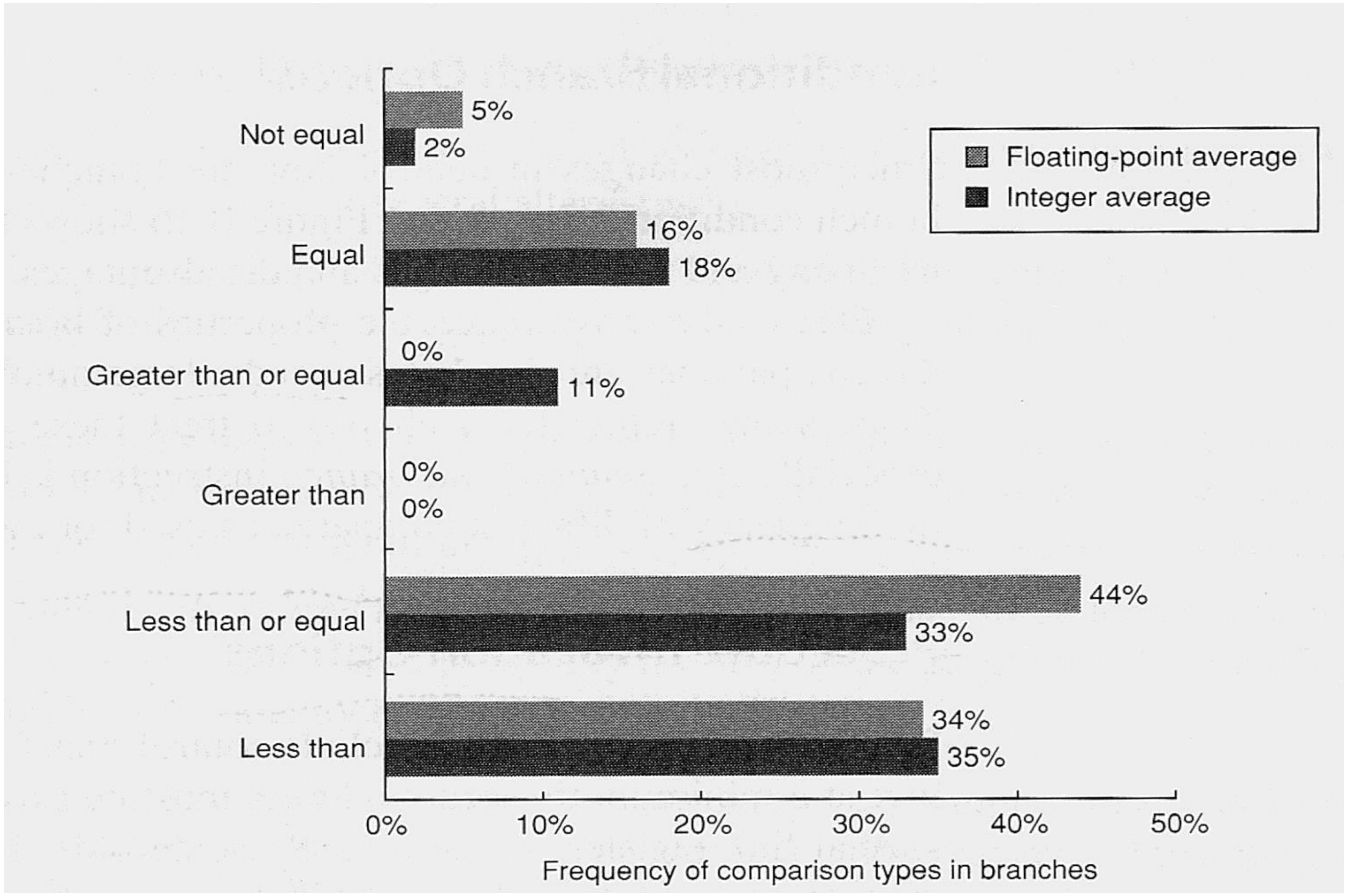
- Short displacement fields often sufficient for branch



- PC-relative, since most branches from current PC address
  - At least 8 bits.

## Conditional Branch Addressing

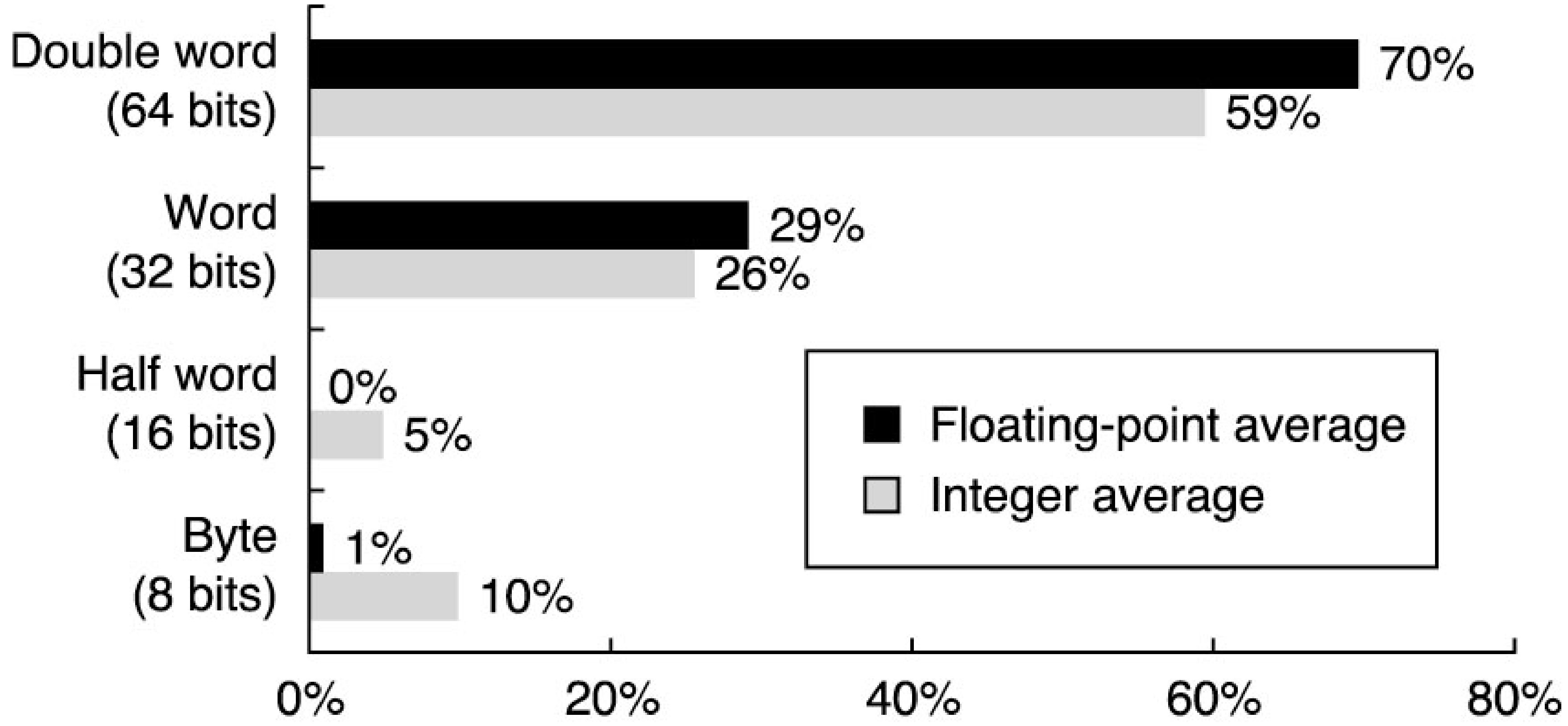
- Compare Equal/Not Equal/Less than (or Equal) are important



## Data Type and Size of Operands

- Byte, half word (16 bits), word (32 bits), double word (64 bits).
- Arithmetic:
  - Decimal: 4bit per digit.
  - Integers: 2's complement
    - 8 bits (C char), 16 bits (C int16_t, sometimes int, short), 32 bits (C int32_t, sometimes long), 64 bits (C int64_t, sometimes long long)
  - Floating-point: IEEE standard
    - Single precision: 32 bits (C float)
    - Double precision: 64 bits (C double)
    - Quadruple precision: 128 bits
    - Extended precision: e.g., 80 bits

# Type and Size of Operands



Distribution of data accesses by size for SPEC CPU2000 benchmark programs

## What should an ISA provide?

Making system design choices using a **quantitative approach**:

- Use general purpose registers with a load-store architecture.
- Support these addressing modes: displacement, immediate, register indirect.
- Support these simple instructions: load, store, add, subtract, move register, shift, compare equal, compare not equal, branch, jump, call, return.
- Support these data size: 8-,16-,32-bit integer, IEEE FP standard.
- Provide at least 16 general purpose registers plus separate FP registers and aim for a minimal instruction set.

## 64-Bit Processors

## 32-bit Computing

- In computer architecture, a word is defined as a unit of data that can be addressed and moved between the computer processor and the storage area.
- In 32-bit computing a word is 32 bits.
- In some cases, the defined bit-length of a word is equivalent to the width of the computer's data bus (and registers) so that a word can be moved in a single operation from the storage to the processor registers

## 32-bit Computing

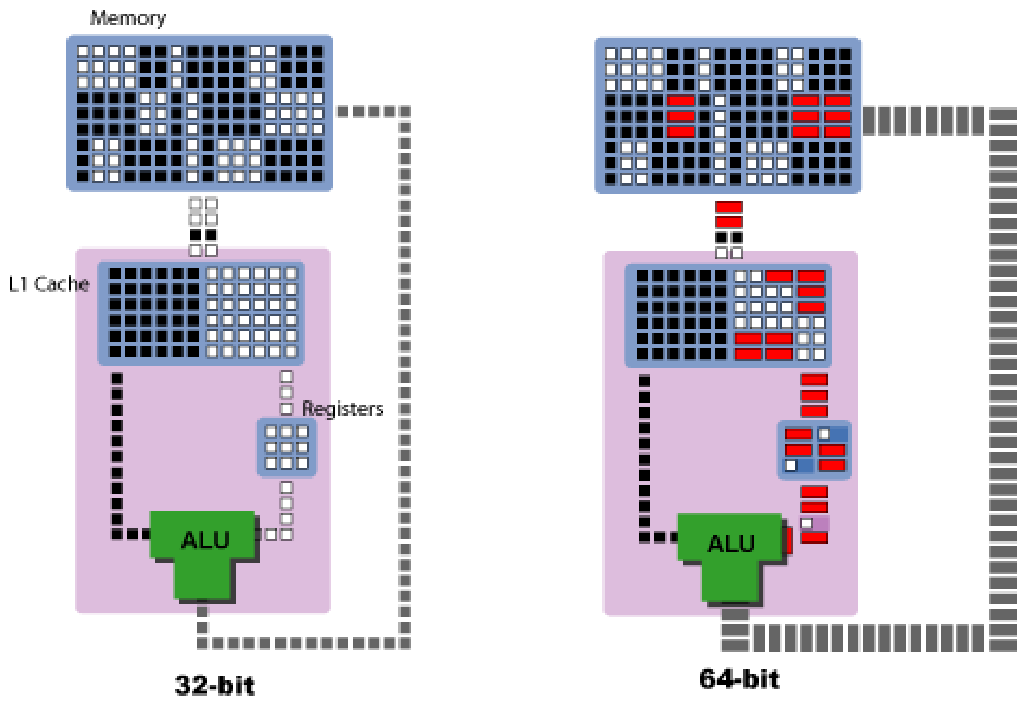
- In a 32-bit microprocessor;
  - There are 32-bit general-purpose registers in the processor.
  - There are $2^{32}$ = 4GB memory to be addressed.

## 64-bit Computing

- The best and simple definition is enhancing the processing word in the architecture to 64 bits.
- The addressable memory increases from 4 GB to up to $2^{64}$ = 18 billion GB (18 exabytes)
  - Many implementations support less than this
- Size of registers extended to 64 bits
- Integer and address data up to 64 bits in length can now be operated on
- $2^{64} = \underline{1.8 \times 10^{19}}$ integers can be represented with 64 bits vs. $\underline{4.3 \times 10^{9}}$ with 32 bits
  - Dynamic range has increased by a factor of 4.3 billion!

# 64-bit Computing

- Stepping up from 32 to 64 bits does not mean doubling performance
- Certain applications will benefit, others may not



## What Applications Can Benefit Most From 64-bit?

- Large databases
- Business and scientific simulation and modeling programs
- Highly graphics-intensive software (CAD, 3-D games)
- Cryptography
- Etc.