Instruction Set Architecture

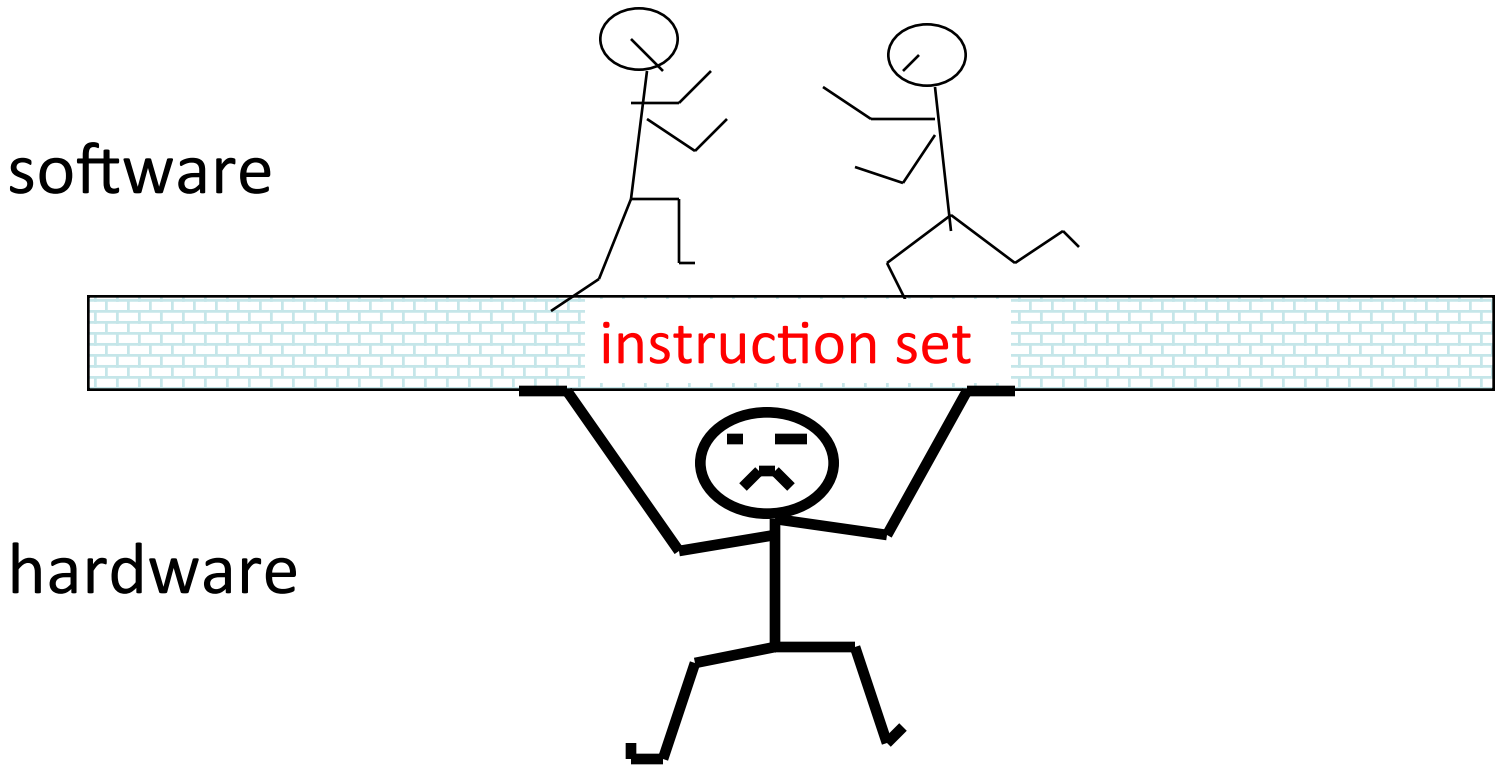
Virendra Singh
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

EE-309: Microprocessors

Lecture 19 (31 Aug 2015)

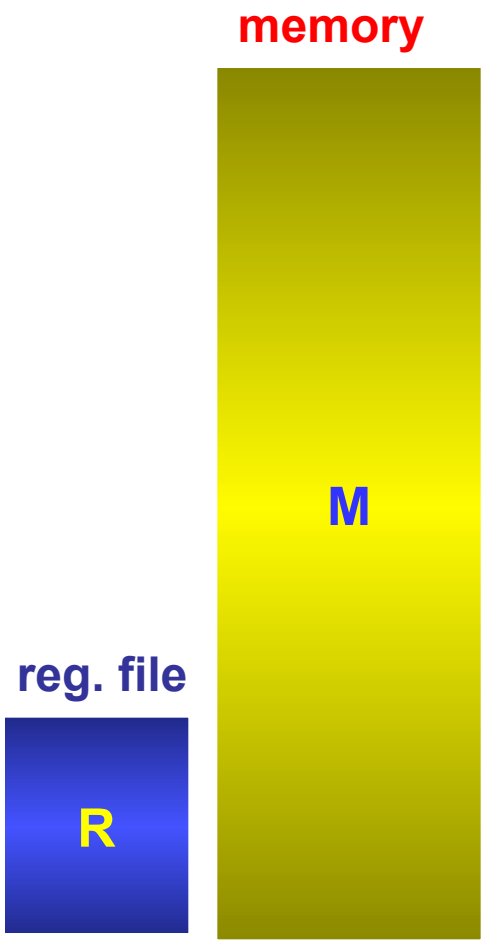
CADSL

Instruction Set Architecture (ISA)



Kinds of Addressing Modes

Addressing Mode	value in [ ] is the operand
• Register direct	[Ri]
• Immediate (literal)	v
• Direct (absolute)	M[v]
• Register indirect	M[[Ri]]
• Base+Displacement	M[[Ri] + v]
• Base+Index	M[[Ri] + [Rj]]
• Scaled Index	M[[Ri] + [Rj]*d + v], e.g. d=8
• Autoincrement	M[[Ri]+1]
• Autodecrement	M[[Ri] - 1]
• Memory Indirect	M[M[Ri]]



ISA-level Tradeoffs: Instruction Length

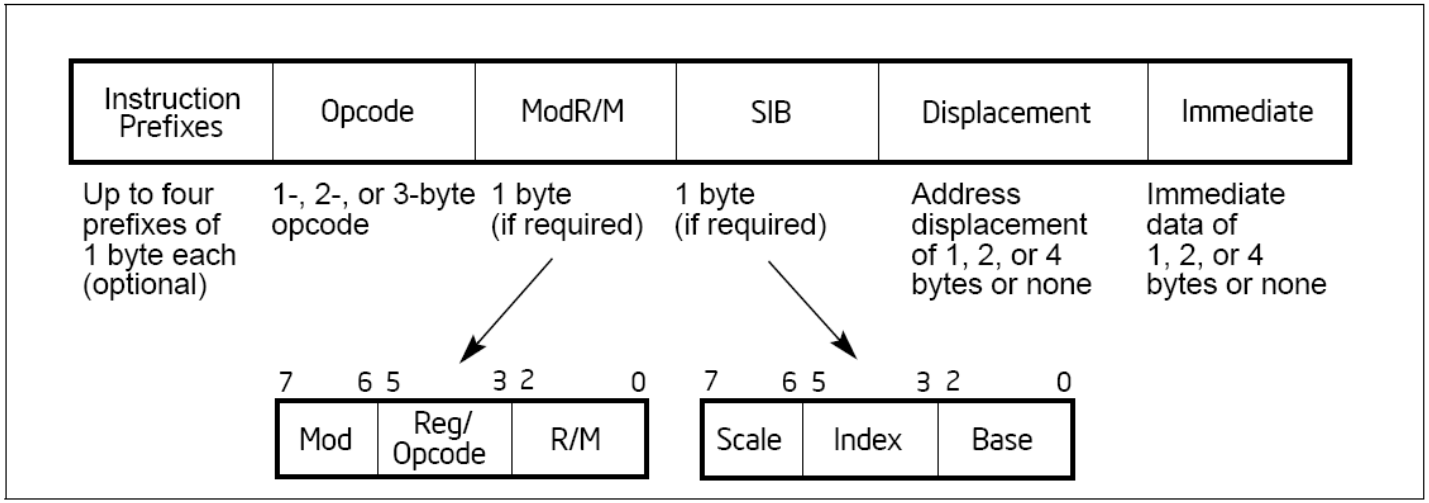
- Fixed length: Length of all instructions the same
  - + Easier to decode single instruction in hardware
  - -- Wasted bits in instructions
  - -- Harder-to-extend ISA (how to add new instructions?)
- Variable length: Length of instructions different (determined by opcode and sub-opcode)
  - + Compact encoding
    - Intel 432: Huffman encoding (sort of). 6 to 321 bit instructions. How?
  - -- More logic to decode a single instruction
- Tradeoffs
  - Code size (memory space, bandwidth, latency) vs. hardware complexity
  - ISA extensibility and expressiveness
  - Performance? Smaller code vs. complex decode

ISA-level Tradeoffs: Uniform Decode

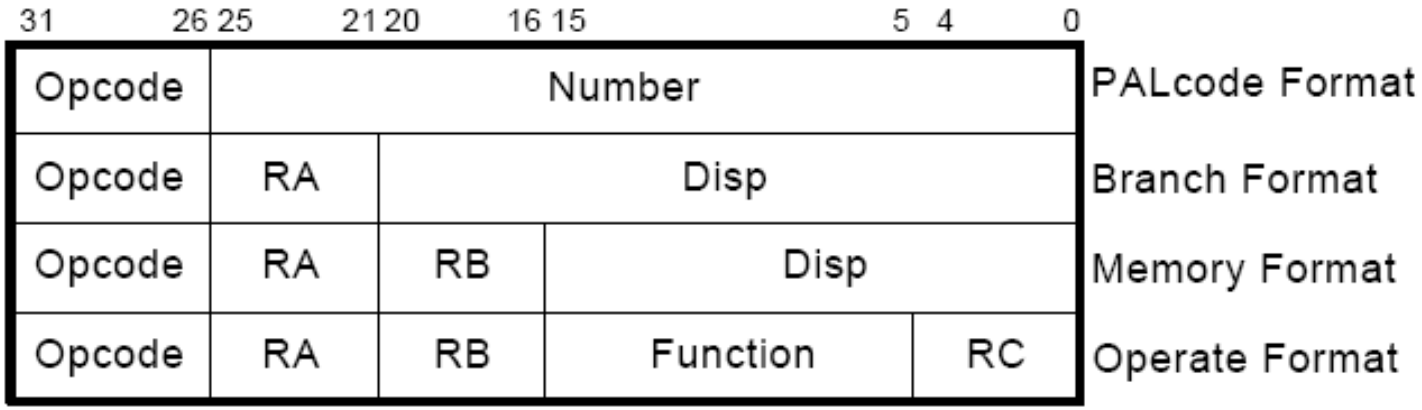
- Uniform decode: Same bits in each instruction correspond to the same meaning
  - ➢ Opcode is always in the same location
  - ➢ Ditto operand specifiers, immediate values, …
  - ➢ Many "RISC" ISAs: Alpha, MIPS, SPARC
  - \+ Easier decode, simpler hardware
  - \+ Enables parallelism: generate target address before knowing the instruction is a branch
  - -- Restricts instruction format (fewer instructions?) or wastes space
- Non-uniform decode
  - ➢ e.g., opcode can be the 1st-7th byte in x86
  - \+ More compact and powerful instruction format
  - -- More complex decode logic

x86 vs. Alpha Instruction Formats

- x86:

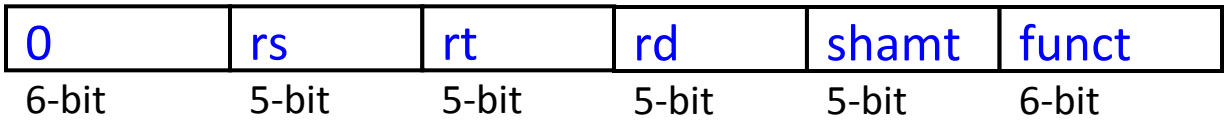


- Alpha:



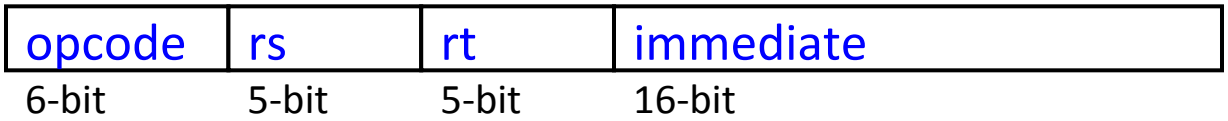
MIPS Instruction Format

- R-type, 3 register operands



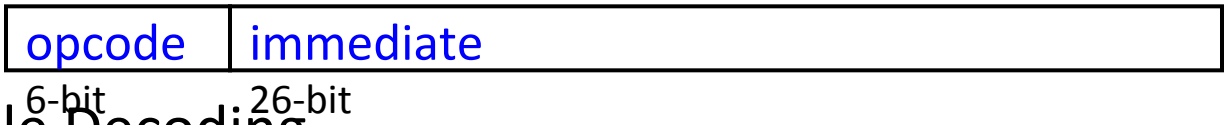
R-type

- I-type, 2 register operands and 16-bit immediate operand



I-type

- J-type, 26-bit immediate operand



J-type

- Simple Decoding
  - 4 bytes per instruction, regardless of format
  - must be 4-byte aligned (2 lsb of PC must be 2b'00)
  - format and fields easy to extract in hardware

A Note on Length and Uniformity

- Uniform decode usually goes with fixed length

- In a variable length ISA, uniform decode can be a property of instructions of the same length
  - It is hard to think of it as a property of instructions of different lengths

Instruction Execution Characteristics:

Type of Operations

Relative Dynamic Frequencies of statements in HLL programs

What type of statements is most frequent?

- Assignment statements dominate
  - Functional instructions and Transfer instructions
  - Movements of data must be made simple, thus fast

- Conditional Statements(if and loop together)
  - Instructions with Control function
  - Sequence control mechanism is important

Instruction Execution Characteristics: Time Consumed by Statements

Time Consumed ⟷ Number of Machine Instructions

	Dynamic Occur		Machine Instr Wt		Memory Ref Wt	
	PASCAL	C	PASCAL	C	PASCAL	C
Assignment	45	38	13	13	14	15
Loop	5	3	42	32	33	26
Call/Return	15	12	31	33	44	45
If	29	43	11	21	7	13
goto	-	3	-	-	-	-
others	6	1	3	1	2	1

Machine instruction weighted
= [Average No. of machine Instr. / Statements] x [Frequency of Occurrences]

Memory reference weighted
= [Average No. of memory references / Statement] x [Frequency of Occurrences]

Most time consuming statement is procedure CALL/RETURN

Instruction Execution Characteristics: Type of Operands

Dynamic Frequencies of Occurrences

	PASCAL	C	Average
Integer Constant	16	23	20
Scalar Variable	58	53	55
Array/Structure	26	24	25

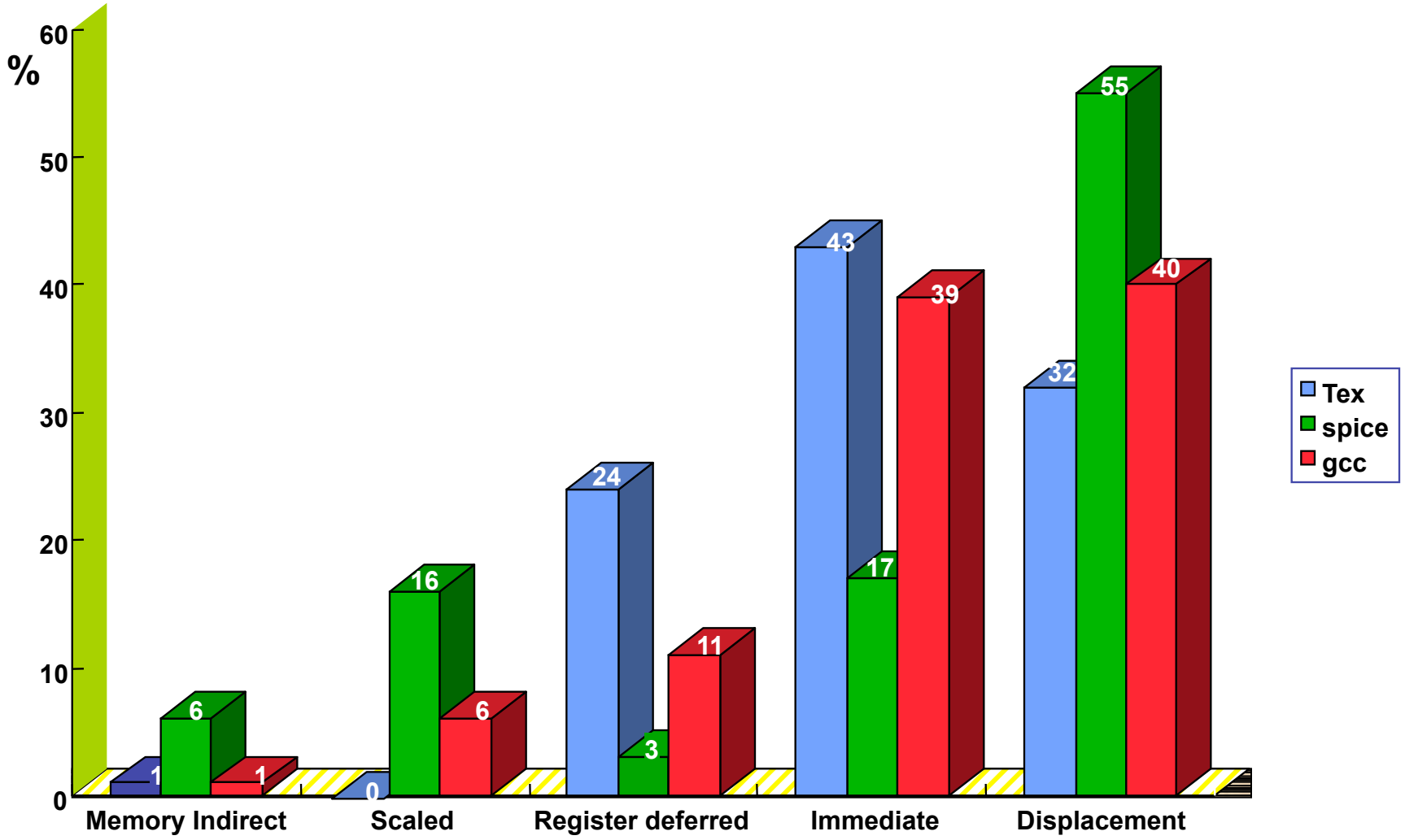
Majority of references to scalar
- 80% are local to a procedure
- References to arrays/structure require index or pointer

Locations of operands(Average per instruction)
- 0.5 operands in memory
- 1.4 operands in registers

Addressing Modes (VAX)

- Register
- Immediate
- Register Indirect
- Displacement
- Indexed
- Direct Absolute
- Memory Indirect
- Auto Increment
- Auto decrement

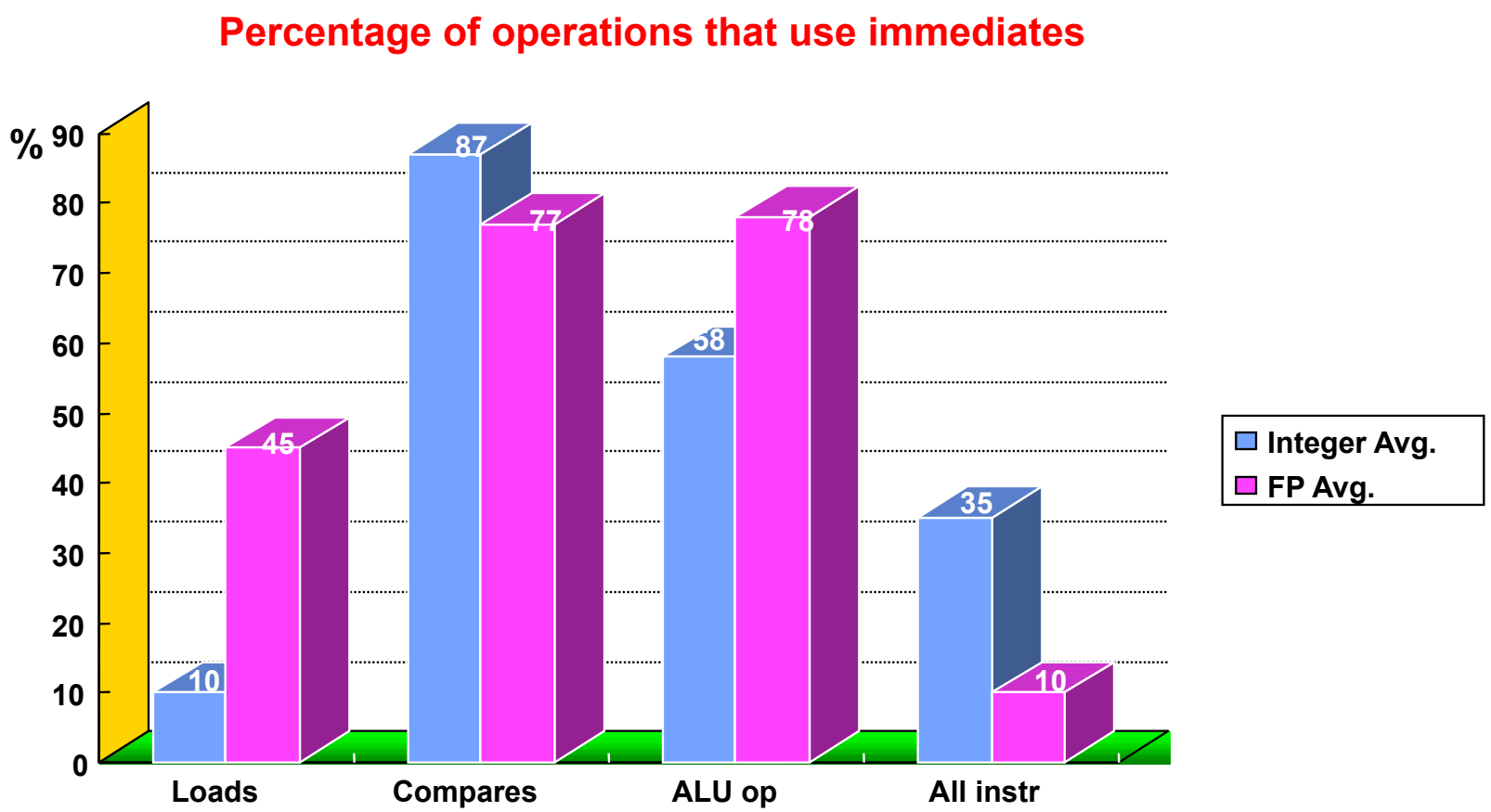
Memory Addressing Modes (VAX)



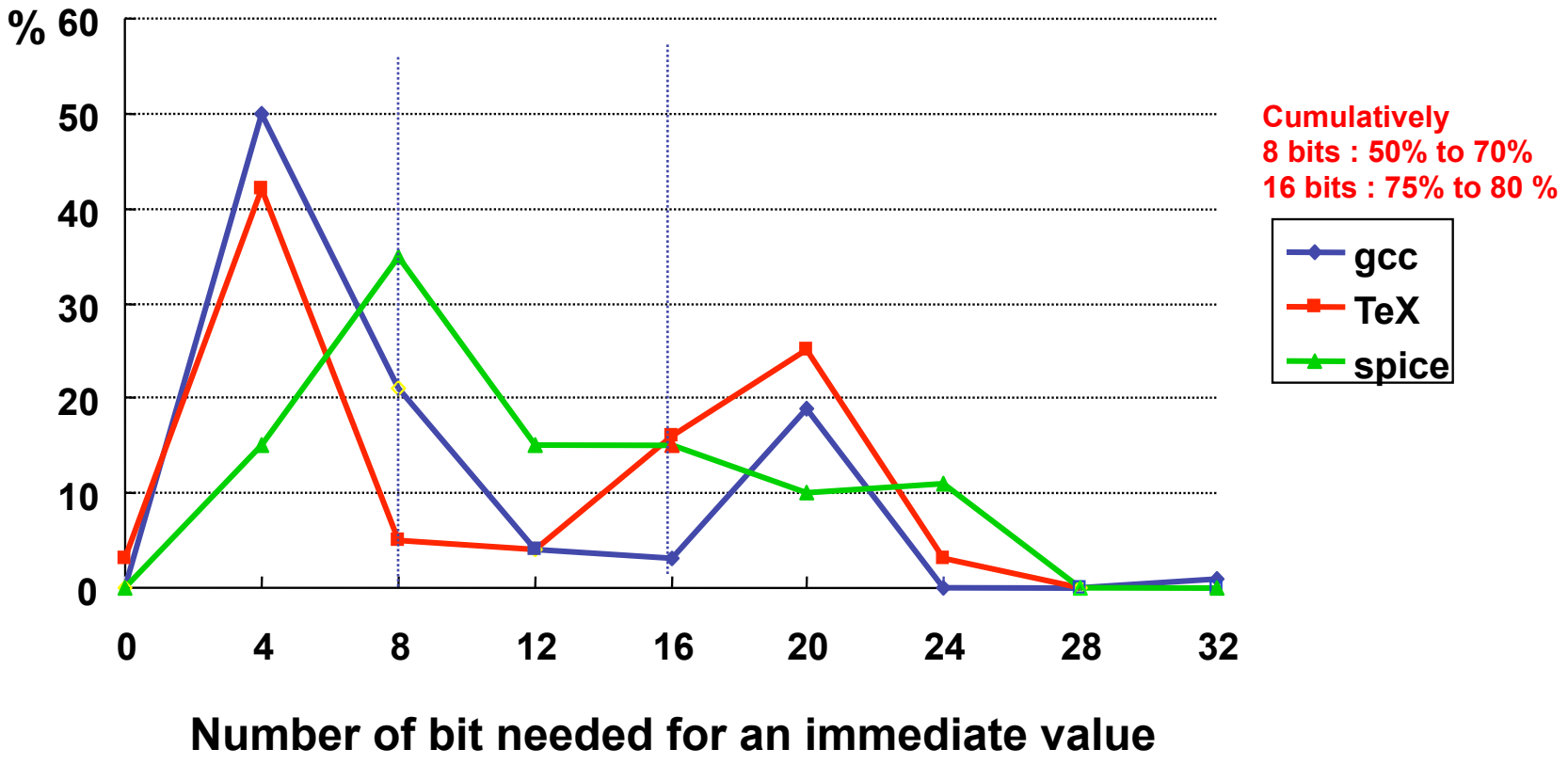
Operand Address bits: Displacement Values

- This value is related to the operand address field when the address is represented by the displacement from the base address
- Wide distribution
- The vast majority --- positive
- A majority of the large displacements - negative

Operand Address bits: Immediate Addressing Mode



Operand Address bits: Immediate Addressing Mode



PERFORMANCE

What is Performance for us?

- For computer architects
  - CPU time = time spent running a program
- Intuitively, bigger should be faster, so:
  - Performance = 1/X time, where X is response, CPU execution, etc.
- Elapsed time = CPU time + I/O wait
- We will concentrate on CPU time

Improve Performance

- Improve (a) response time or (b) throughput?
  - Faster CPU
    - Helps both (a) and (b)
  - Add more CPUs
    - Helps (b) and perhaps (a) due to less queueing

Performance Comparison

- Machine A is n times faster than machine B iff perf(A)/perf(B) = time(B)/time(A) = n
- Machine A is x% faster than machine B iff
  - perf(A)/perf(B) = time(B)/time(A) = 1 + x/100
- e.g. time(A) = 10s, time(B) = 15s
  - 15/10 = 1.5 => A is 1.5 times faster than B
  - 15/10 = 1.5 => A is 50% faster than B

Iron Law

$$\text{Processor Performance} = \frac{\text{Time}}{\text{Program}}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Cycles}}{\text{Instruction}} \times \frac{\text{Time}}{\text{Cycle}}$$

(code size) (CPI) (cycle time)

Architecture --> Implementation --> Realization

Compiler Designer Processor Designer Chip Designer

Our Goal

- Minimize time which is the product, NOT isolated terms
- Common error to miss terms while devising optimizations
  - e.g. ISA change to decrease instruction count
  - BUT leads to CPU organization which makes clock slower
- Bottom line: terms are inter-related

Other Metrics

- MIPS and MFLOPS
- MIPS = instruction count/(execution time x 10^6)
  = clock rate/(CPI x 10^6)
- But MIPS has serious shortcomings

Problems with MIPS

- Ignores program
- Usually used to quote peak performance
  - Ideal conditions → guaranteed not to exceed!
- When is MIPS ok?
  - Same compiler, same ISA
    - e.g. same binary running on AMD Phenom, Intel Core i7
  - Instr/program is constant and can be ignored

Other Metrics

- MFLOPS = FP ops in program/(execution time x 10^6)
- Assuming FP ops independent of compiler and ISA
  - Often safe for numeric codes: matrix size determines # of FP ops/program
  - However, not always safe:
    - Missing instructions (e.g. FP divide)
    - Optimizing compilers
- Relative MIPS and normalized MFLOPS

Iron Law Example

- Machine A: clock 1ns, CPI 2.0, for program x
- Machine B: clock 2ns, CPI 1.2, for program x
- Which is faster and how much?

  Time/Program = instr/program x cycles/instr x sec/cycle

  Time(A) = N x 2.0 x 1 = 2N

  Time(B) = N x 1.2 x 2 = 2.4N

  Compare: Time(B)/Time(A) = 2.4N/2N = 1.2
- So, Machine A is 20% faster than Machine B for this program

Iron Law Example

Keep clock(A) @ 1ns and clock(B) @2ns

For equal performance, if CPI(B)=1.2, what is CPI(A)?

Time(B)/Time(A) = 1 = (Nx2x1.2)/(Nx1xCPI(A))

CPI(A) = 2.4

Iron Law Example

- Keep CPI(A)=2.0 and CPI(B)=1.2
- For equal performance, if clock(B)=2ns, what is clock(A)?

Time(B)/Time(A) = 1 = (N x 2.0 x clock(A))/(N x 1.2 x 2)

clock(A) = 1.2ns

Thank You