Computer Architecture

Instruction Set Architecture

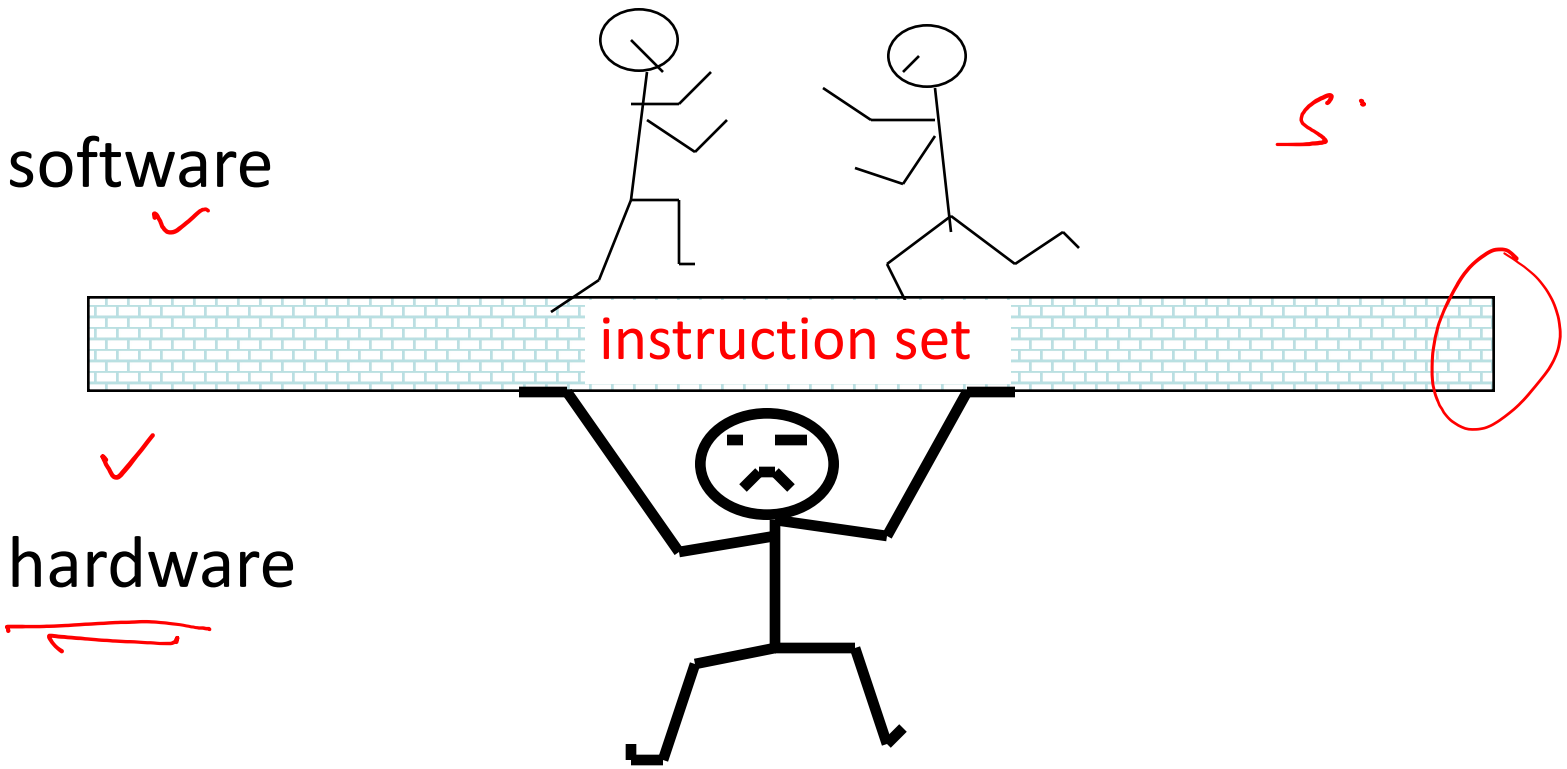
Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CS-683: Advanced Computer Architecture

Lecture 5 (09 August 2021)

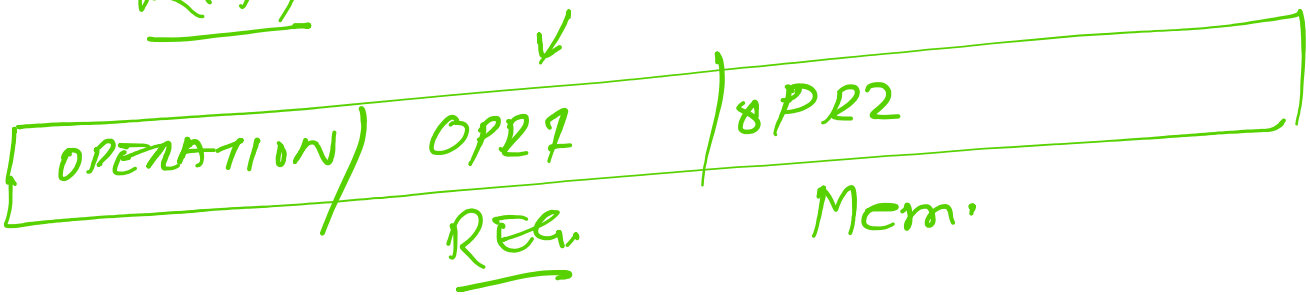
CADSL

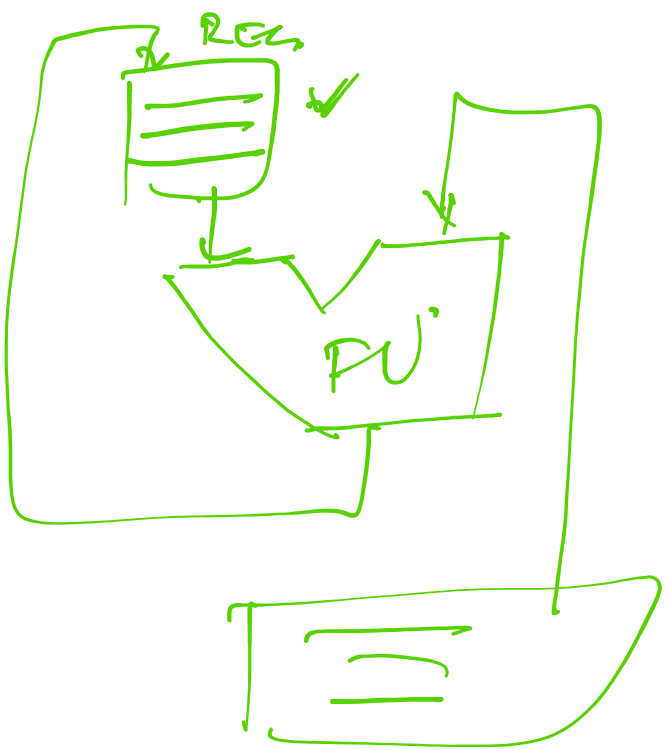
Instruction Set Architecture (ISA)

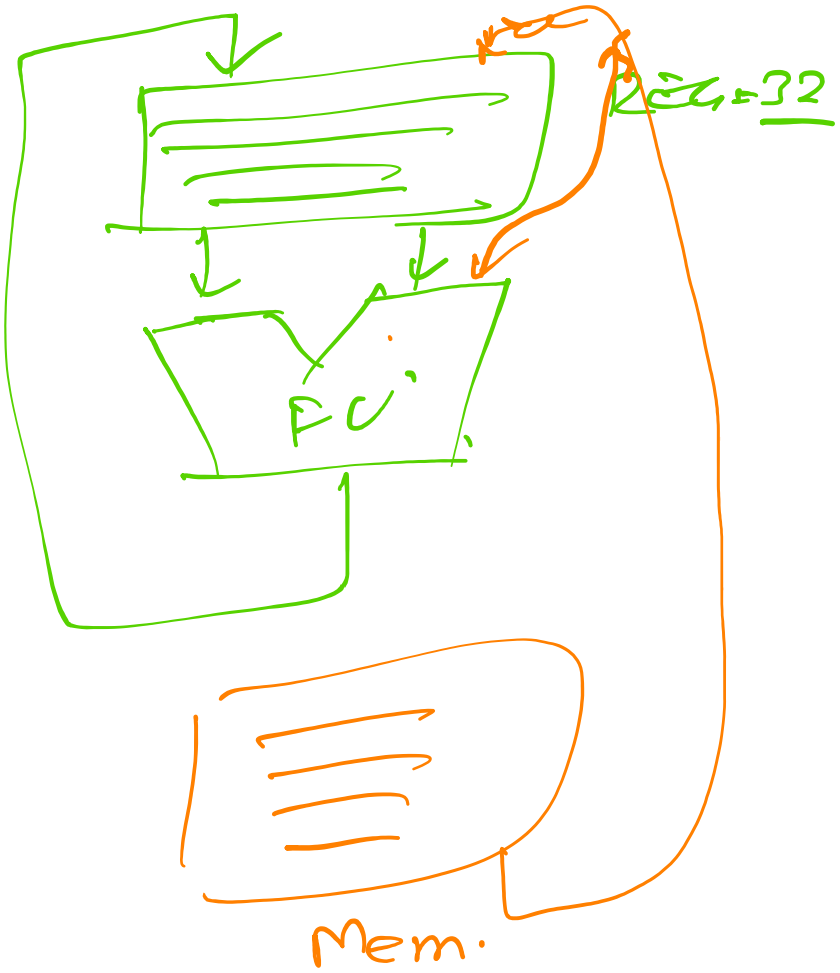


1. Scarcity of memory ⇒ min # bits to encode an instr
2. Slow memory
3. Ease of programming

R-M







Memory traffic

↓

Slow

32 ⇒

REG.

OPERATION	OPR1	OPR2
7	5	5

⇒ 17 bits

2 Add.

3 Address

OPERATION	OPR1	OPR2	DEST.
7	5	5	5

⇒ 22

Arithmetic / Logic / Control flow (branch / jumps)

Load / Store ⇒ allowed to communicate with memory

Load / Store Arch.

⇓

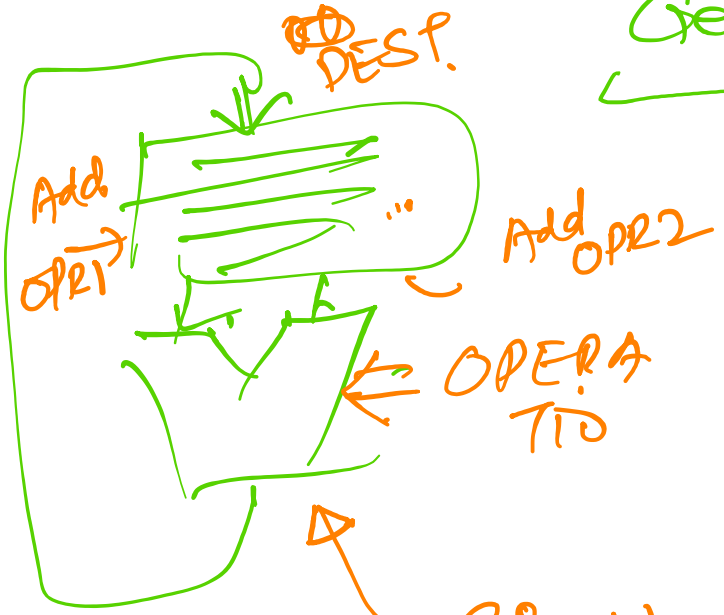
Very close to machine

⇓

Generating control signal for machine.

⇓

Code.



ISA → Compiler.

Turing

Simple instruction ⇒ RISC ⇒

MAC

* +

Load-Store Architectures

- Instruction set:

  add R1, R2, R3 sub R1, R2, R3 mul R1, R2, R3

  load R1, R4 store R1, R4

- Example: A*B - (A+C*B)

```
load R4, A
load R5, B
load R6, C
mul R7, R6, R5      /*    C*B          */
add R8, R7, R4      /*    A + C*B      */
mul R9, R4, R5      /*    A*B          */
sub R10, R9, R8     /*    A*B - (A+C*B) */
```

Load-Store: Pros and Cons

- Pros
  - Simple, fixed length instruction encoding
  - Instructions take similar number of cycles
  - Relatively easy to pipeline
- Cons
  - Higher instruction count
  - Not all instructions need three operands
  - Dependent on good compiler

Registers: Advantages and Disadvantages

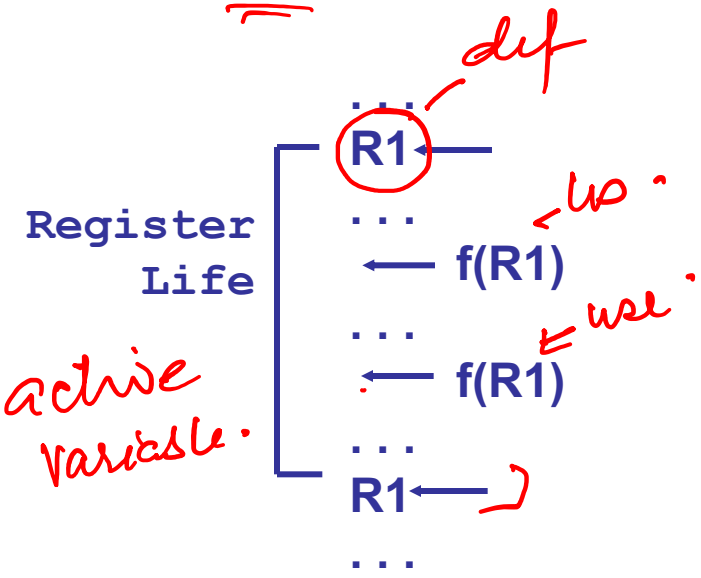
- Advantages
  - Faster than cache (no addressing mode or tags)
  - Deterministic (no misses)
  - Can replicate (multiple read ports)
  - Short identifier (typically 3 to 8 bits)
  - Reduce memory traffic
- Disadvantages
  - Need to save and restore on procedure calls and context switch
  - Can't take the address of a register (for pointers)
  - Fixed size (can't store strings or structures efficiently)
  - Compiler must manage

How Many Registers in RF

6 algorithms from CALGO(ACM) written in 4 languages; ALGOL,BASIC, BLISS,FORTRAN

Register Life

Life Length	No. of Lives	Fraction	Cum. Fraction
1~1	174,927	0.09	0.09
2~3	728,346	0.38	0.48
4~7	547,072	0.29	0.77
8~15	252,508	0.13	0.90
16~31	116,404	0.06	0.96
32~63	41,673	0.02	0.98
64~127	17,790	0.01	0.99
128~	15,603	0.01	1.00



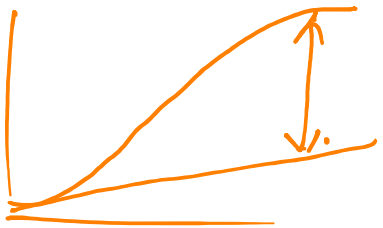
We need to try to keep the live registers in the RF

Avg number of simultaneous RL: 2 ~ 6
No program uses more than 15 registers simultaneously

Slow Memory

CISC / ⇒ RISC

Memory 200 – 1000 time slower.



⇒ Pack as much work as we can

INSERT A, X.

FFT.

Complex vs. Simple Instructions

- Complex instruction: An instruction does a lot of work, e.g. many operations
  - Insert in a doubly linked list
  - Compute FFT
  - String copy

- Simple instruction: An instruction does small amount of work, it is a primitive using which complex operations can be built
  - Add
  - XOR
  - Multiply

Complex vs. Simple Instructions

- Advantages of Complex instructions
  - + Denser encoding → smaller code size → better memory utilization, saves off-chip bandwidth, better cache hit rate (better packing of instructions)
  - + Simpler compiler: no need to optimize small instructions as much

- Disadvantages of Complex Instructions
  - - Larger chunks of work → compiler has less opportunity to optimize (limited in fine-grained optimizations it can do)
  - - More complex hardware → translation from a high level to control signals and optimization needs to be done by hardware

3 Ease of Programming

Reg.
↓
Scalar

for (i=0; i<1000; i++)
A(i) = A(i) + B(i)

which Address.
A(0)
A(1)
⋮
A(999)

A(0) = A(0) + B(0)
A(1) = A(1) + B(1)
⋮

OPERAND
↓
Register ✓
Memory → Direct
→ Indirect → Register ✓
→ Memory
Immediate

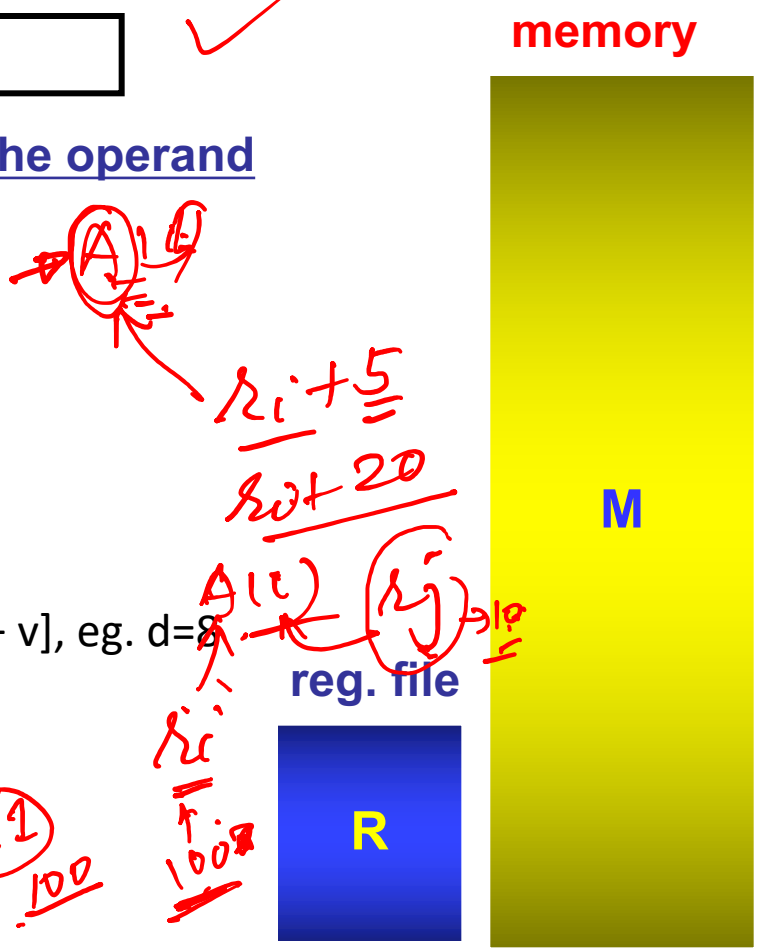
Add
a = b + 5

OPERATION	A/M	OPR1	OPR2

Kinds of Addressing Modes

OP	Ri	Rj	v

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], eg. d=8
• Autoincrement	M[[Ri]+1]
• Autodecrement	M[[Ri] - 1]
• Memory Indirect	M[M[Ri]]
• [Indirection Chains]	



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

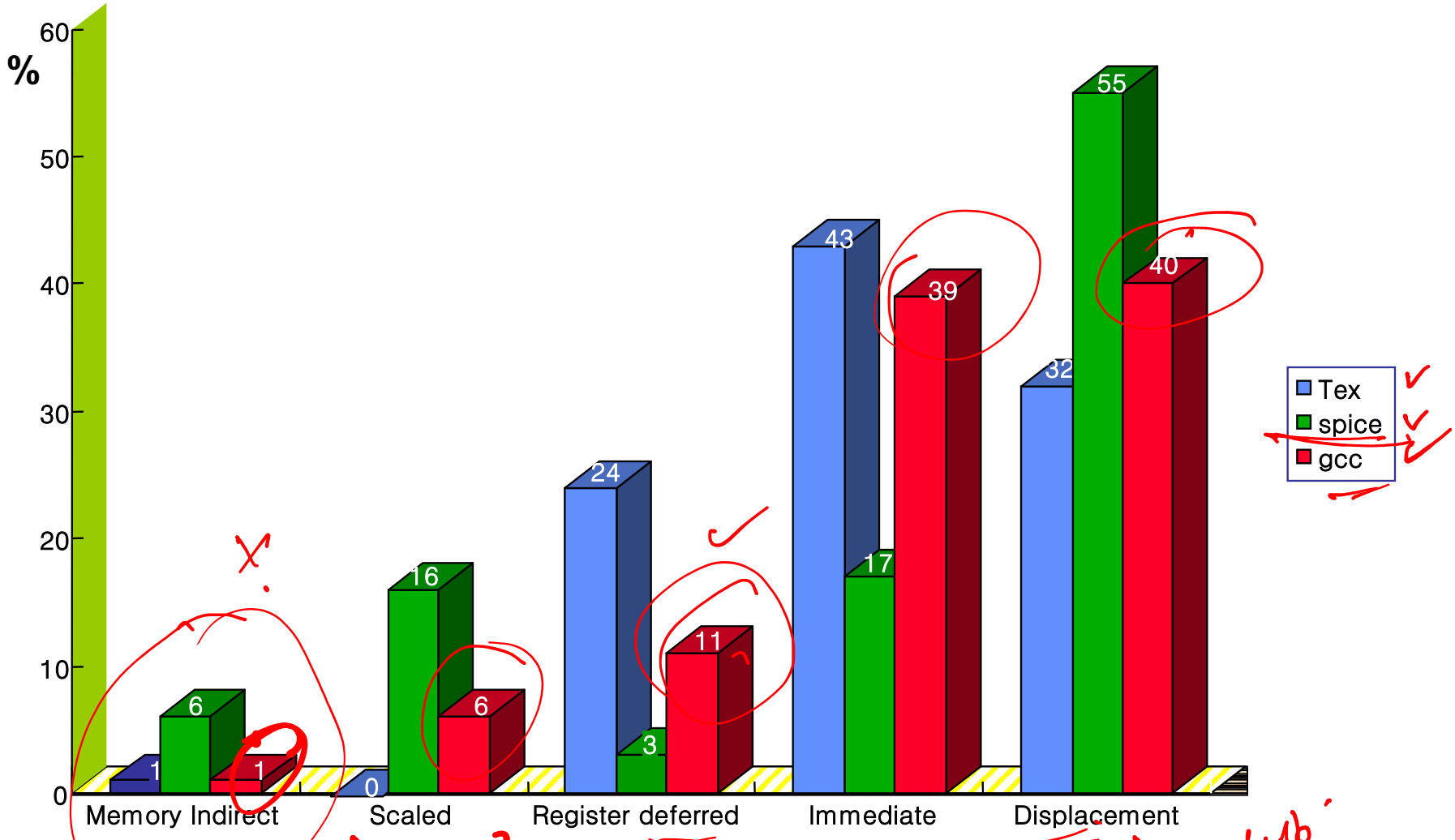
Majority of references to scalar
- 80% are local to a procedure
- References to arrays/structure require index or pointer

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

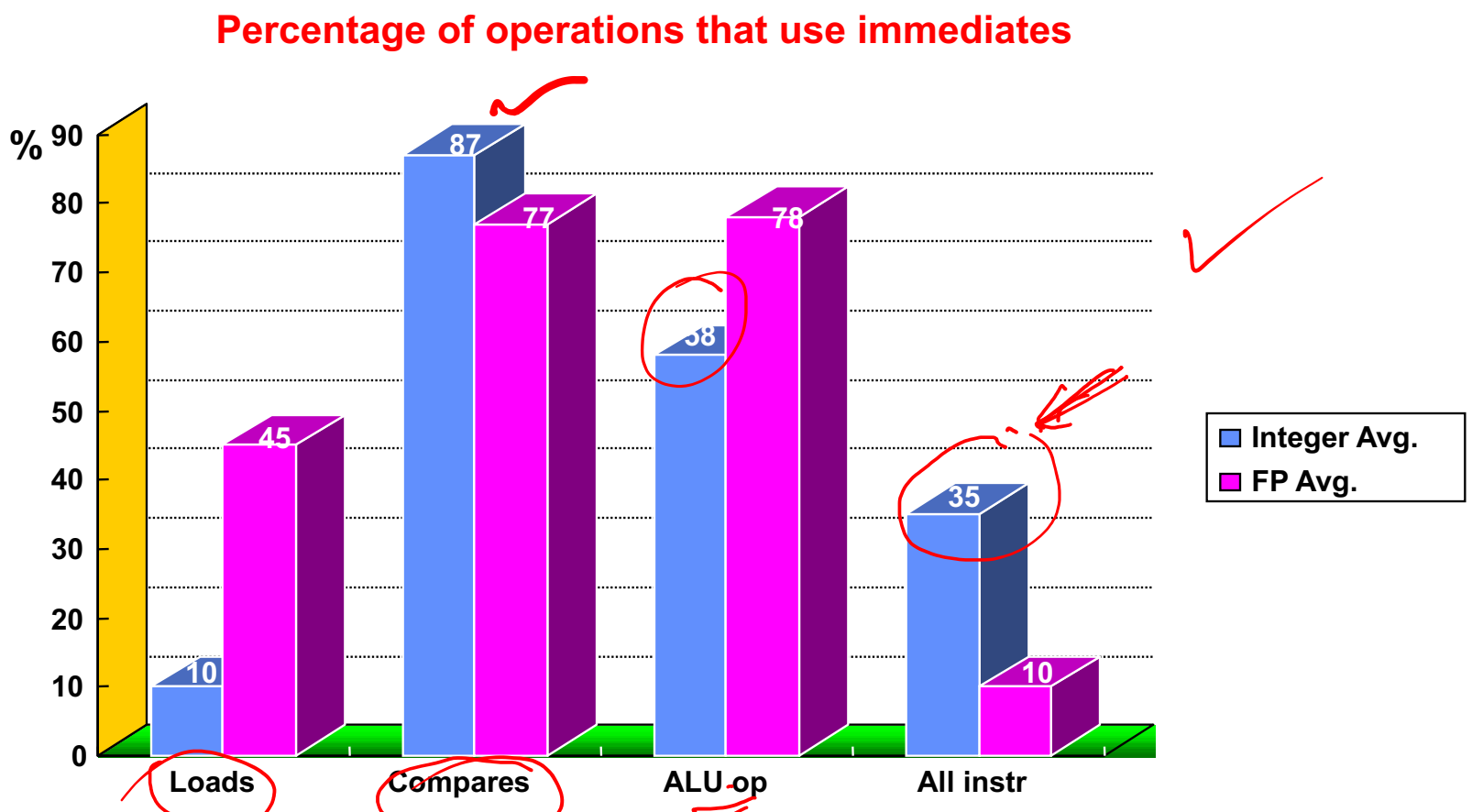
Memory Addressing Modes (VAX)



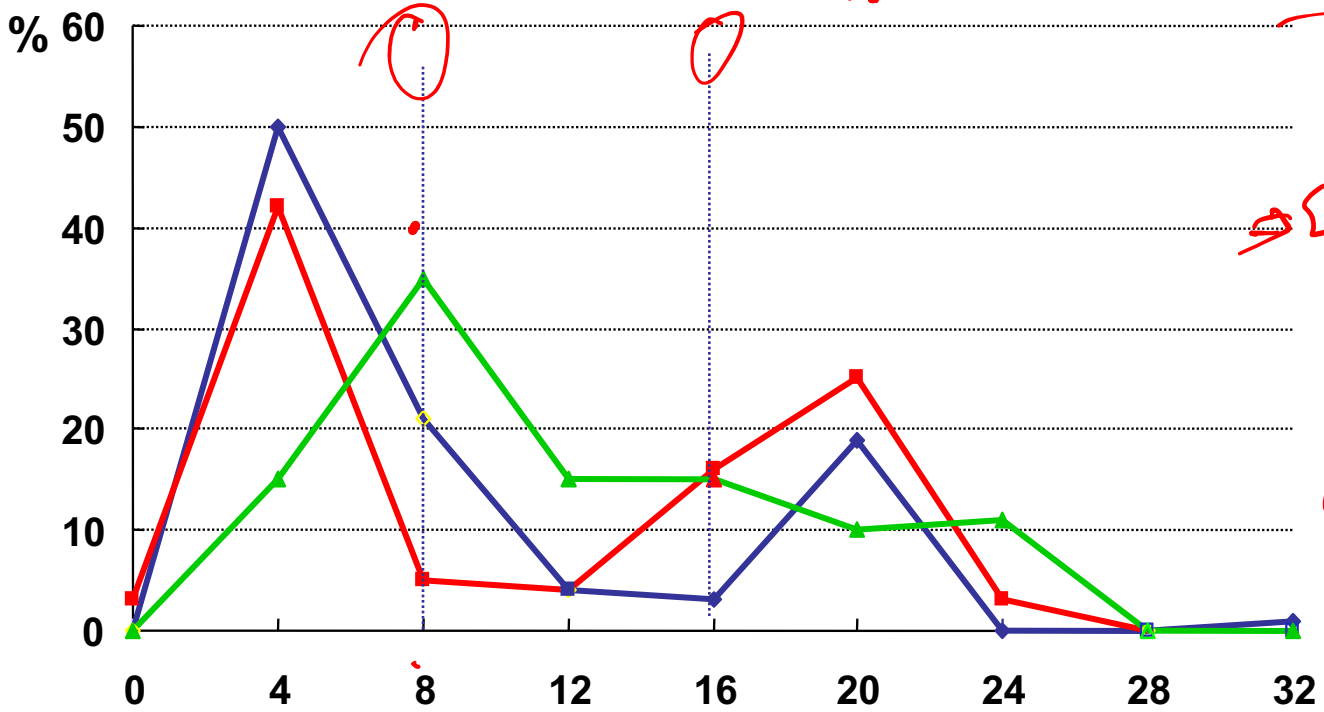
Operand Address bits: Displacement Values

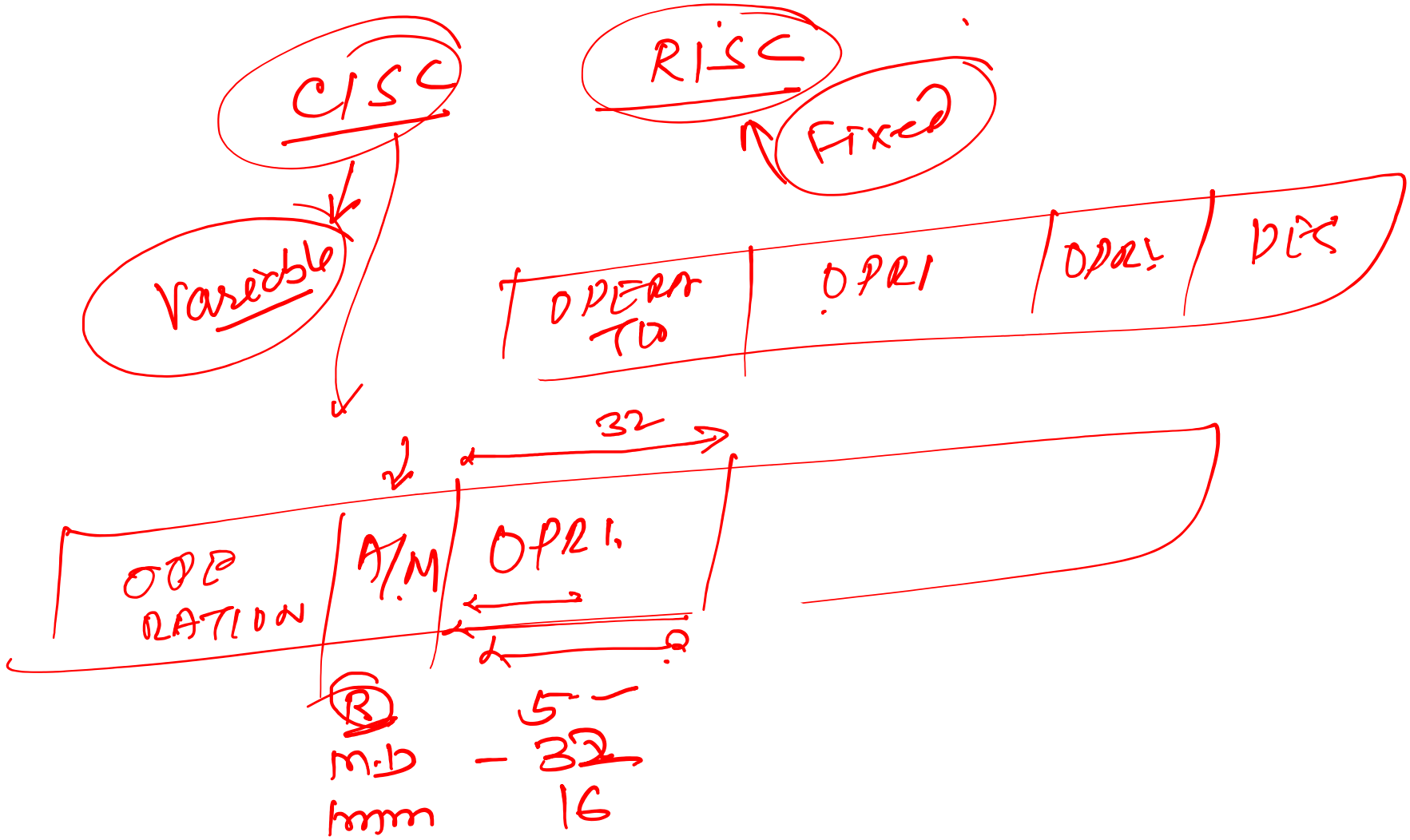
- This value is related to the operand address field when the address is represented by the displacement from the base address
- Wide distribution
- The vast majority --- positive
- A majority of the large displacements - negative

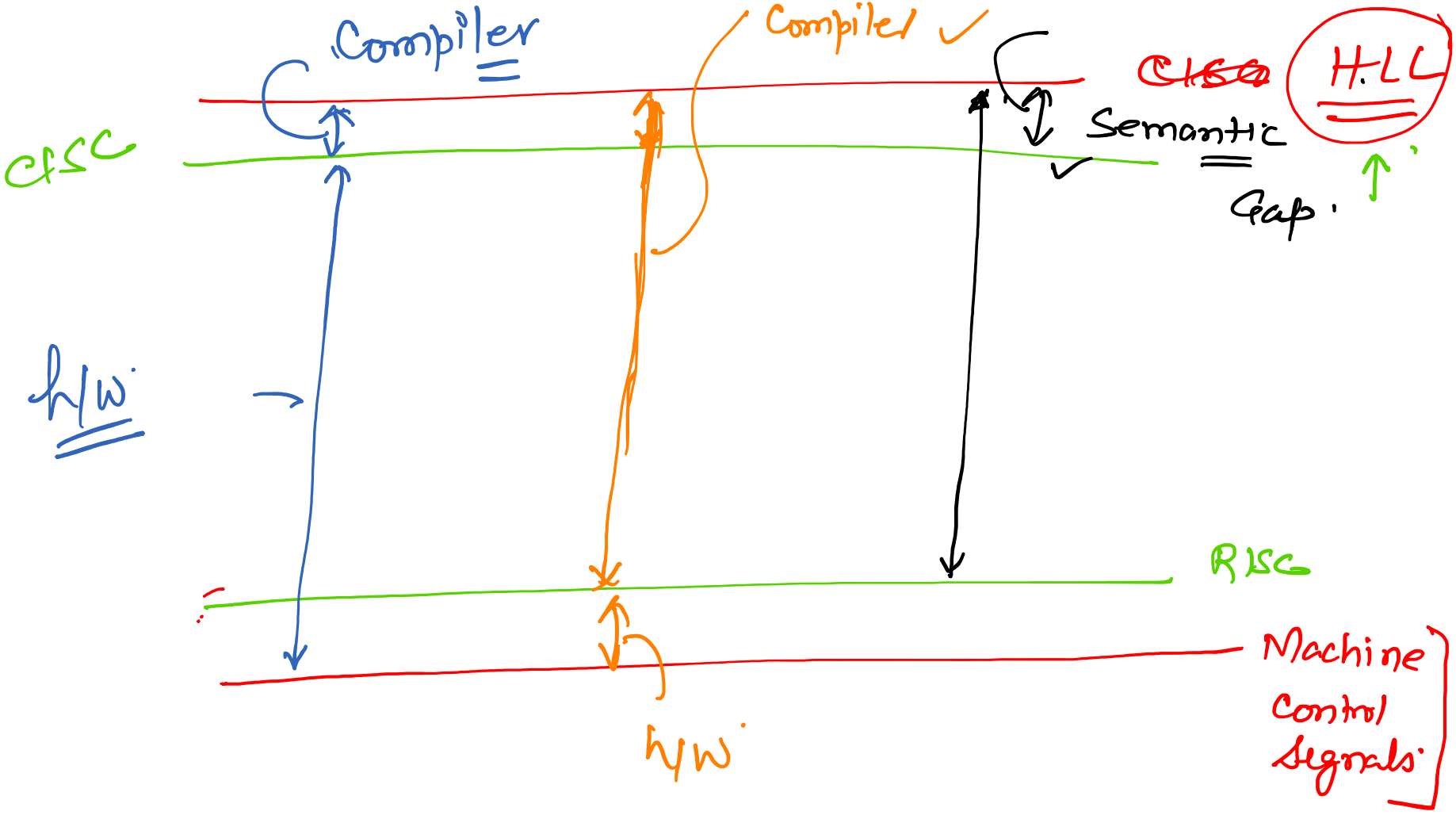
Operand Address bits: Immediate Addressing Mode

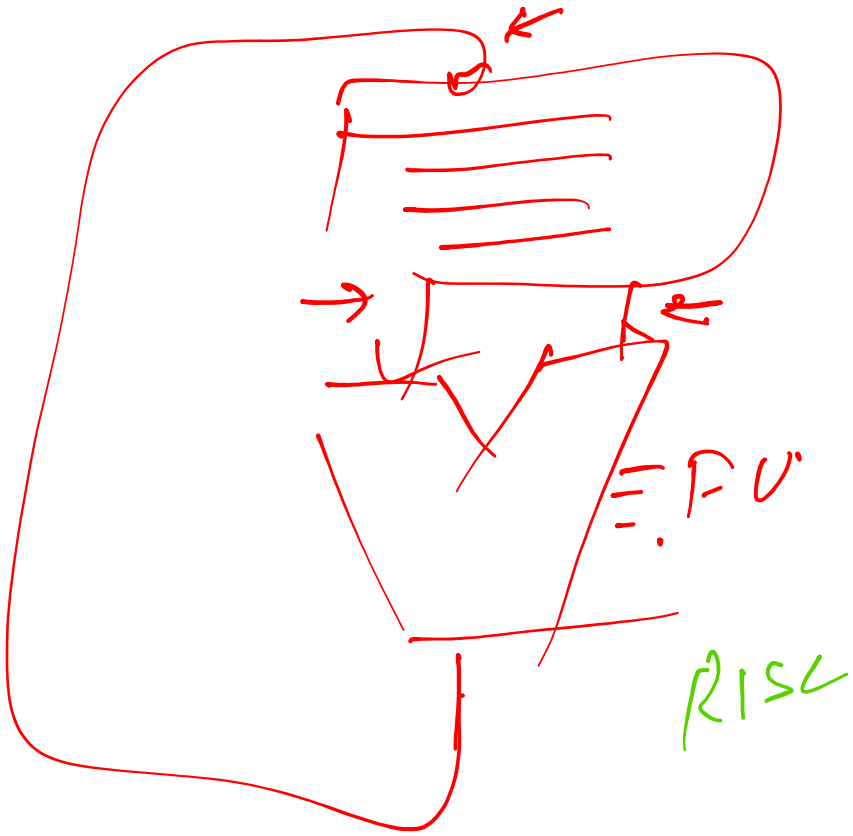


Operand Address bits: Immediate Addressing Mode

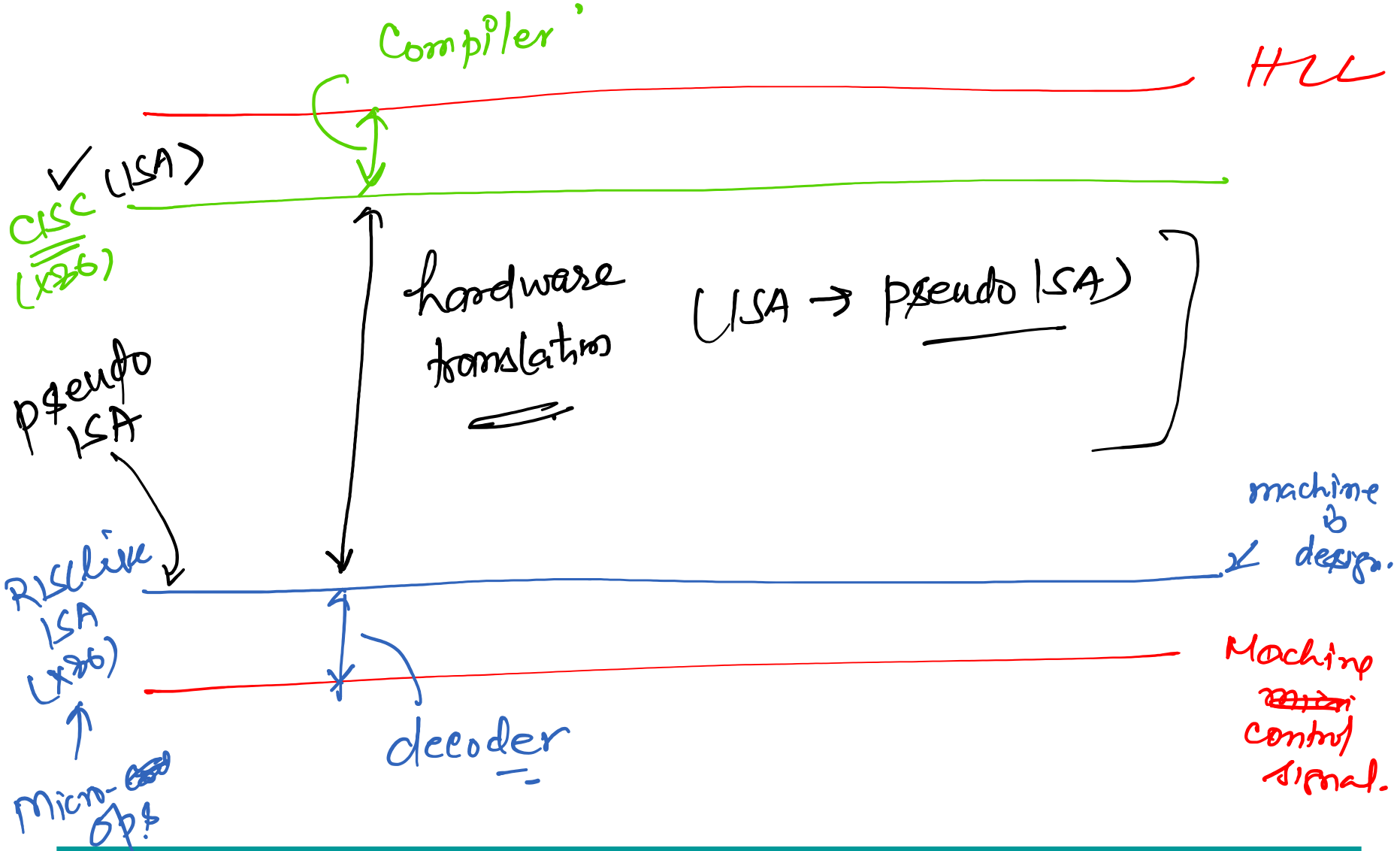



CISC
RISC
Fixed
Variable
OPERATION
OPR1
OPR2
DISP
32
OPE RATION
A/M
OPR1
R
5
M.D
32
16


Compiler
Compiled
CISC
H.L.L
Semantic
Gap.
h/w
RISC
Machine
Control
Signals
h/w


EFU
RISC

ISA Indirection



ISA-level Tradeoffs: Semantic Gap

- Where to place the ISA? Semantic gap
  - Closer to high-level language (HLL) → Small semantic gap, complex instructions
  - Closer to hardware control signals? → Large semantic gap, simple instructions

- RISC vs. CISC machines
  - RISC: Reduced instruction set computer
  - CISC: Complex instruction set computer
    - FFT, QUICKSORT, POLY, FP instructions?
    - VAX INDEX instruction (array access with bounds checking)

ISA-level Tradeoffs: Semantic Gap

- Simple compiler, complex hardware vs. complex compiler, simple hardware
  - Caveat: Translation (indirection) can change the tradeoff!

- Burden of backward compatibility

- Performance?
  - Optimization opportunity: Example of VAX INDEX instruction: who (compiler vs. hardware) puts more effort into optimization?
  - Instruction size, code size

Small versus Large Semantic Gap

- CISC vs. RISC
  - Complex instruction set computer → complex instructions
    - Initially motivated by "not good enough" code generation
  - Reduced instruction set computer → simple instructions
    - John Cocke, mid 1970s, IBM 801
      - Goal: enable better compiler control and optimization

Turing Award!

- RISC motivated by
  - Memory stalls (no work done in a complex instruction when there is a memory *stall*?)
  - Simplifying the hardware → lower cost, higher frequency
  - Enabling the compiler to optimize the code better
    - Find fine-grained parallelism to reduce *stalls*

How High or Low Can You Go?

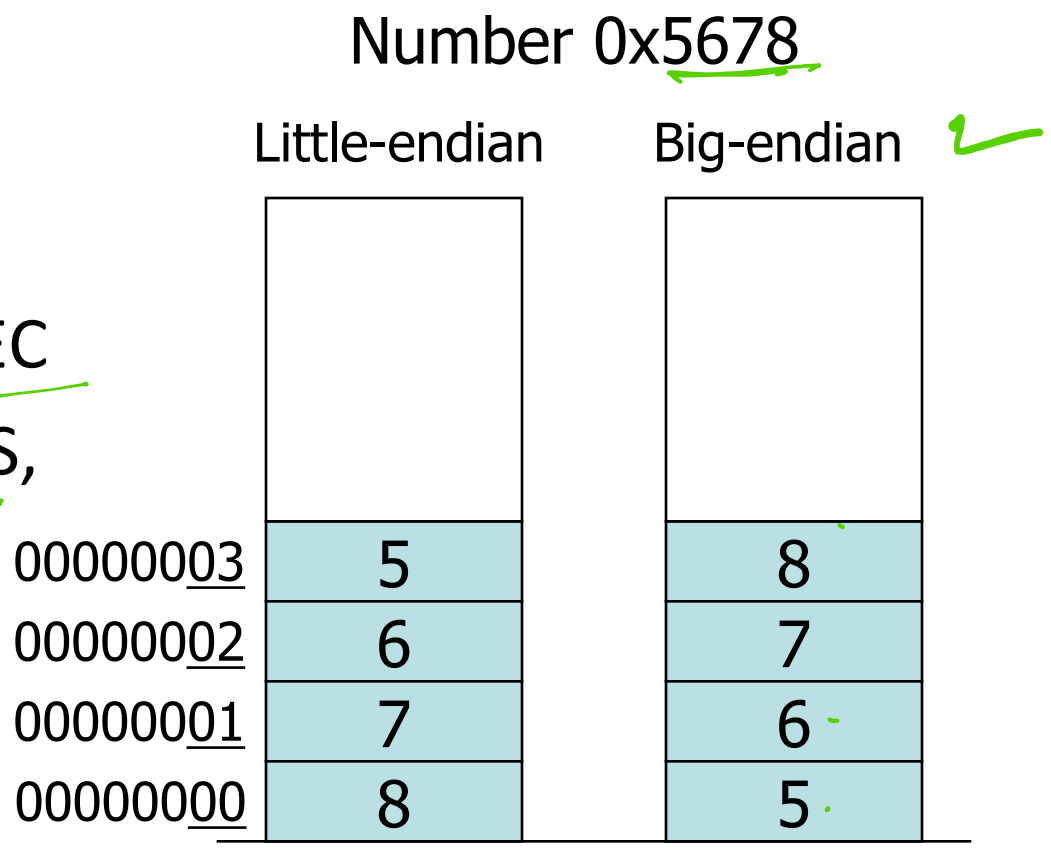
- Very large semantic gap
  - Each instruction specifies the complete set of control signals in the machine
  - Compiler generates control signals
  - Open microcode (John Cocke, 1970s)
    - Gave way to optimizing compilers
- Very small semantic gap
  - ISA is (almost) the same as high-level language
  - Java machines, LISP machines, object-oriented machines, capability-based machines

Memory Address

- Interpreting memory address
  - Big Endian
  - Little Endian
- Instruction misalignment
- Addressing mode

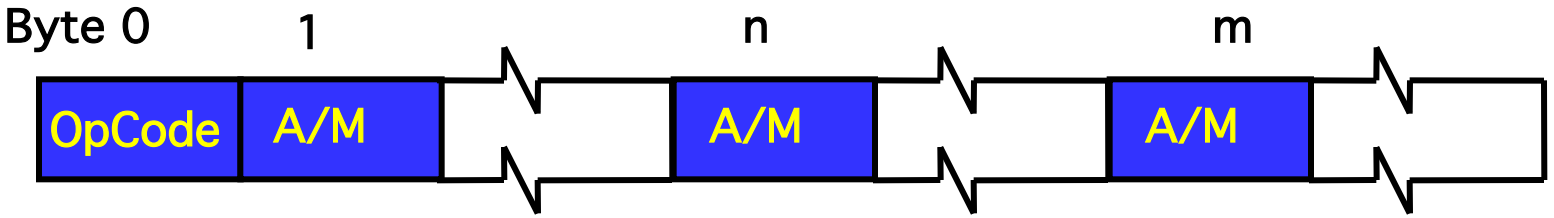
Little or Big: Where to Start?

- Byte ordering: Where is the first byte?
- Big-endian : IBM, SPARC, Mororola
- Little-endian: Intel, DEC
- Supporting both: MIPS, PowerPC, ARM



VAX-11

Variable format, 2- and 3-address instructions



- 32-bit word size, 16 GPR (4 reserved)
- Rich set of addressing modes (apply to any operand)
- Rich set of operations
  - bit field, stack, call, case, loop, string, poly, system
- Rich set of data types (B, W, L, Q, O, F, D, G, H)
- Condition codes

ISA-level Tradeoffs: Instruction Length

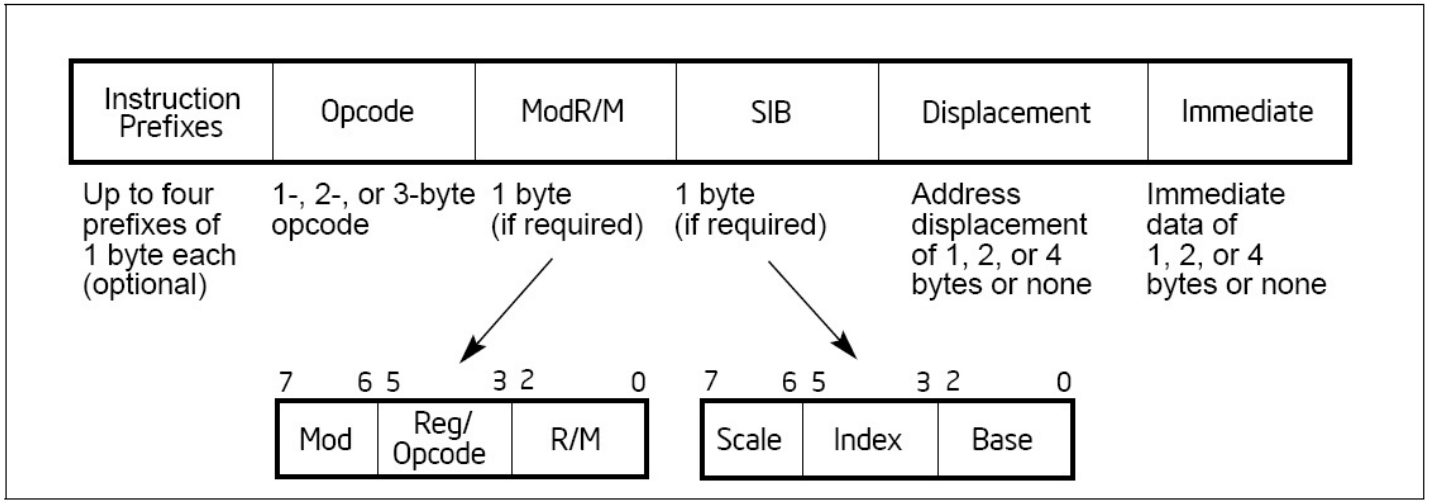
- Fixed length: Length of all instructions the same
  - + Easier to decode single instruction in hardware
  - + Easier to decode multiple instructions concurrently
  - -- Wasted bits in instructions
  - -- Harder-to-extend ISA (how to add new instructions?)
- Variable length: Length of instructions different (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
  - -- Harder to decode multiple instructions concurrently
- Tradeoffs
  - Code size (memory space, bandwidth, latency) vs. hardware complexity
  - ISA extensibility and expressiveness
  - Performance? Smaller code vs. imperfect decode

ISA-level Tradeoffs: Uniform Decode

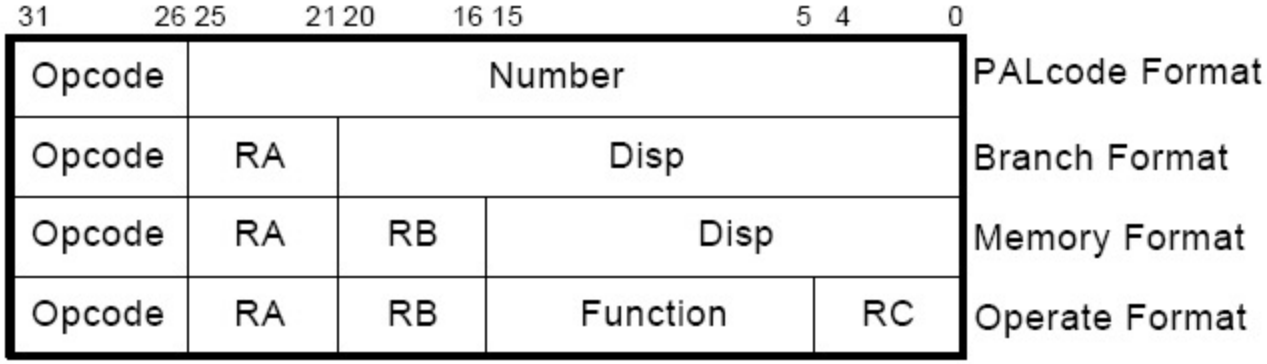
- Uniform decode: Same bits in each instruction correspond to the same meaning
  - ➢ Opcode is always in the same location
  - ➢ Ditto operand specifiers, immediate values, …
  - ➢ Many "RISC" ISAs: Alpha, MIPS, SPARC
  - + Easier decode, simpler hardware
  - + Enables parallelism: generate target address 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:

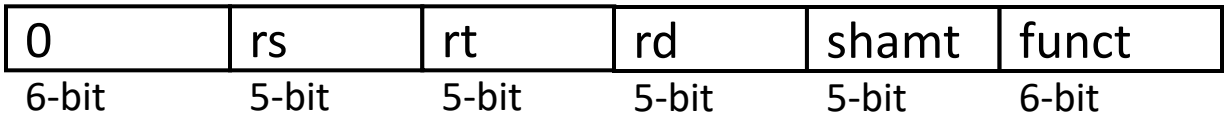


- Alpha:



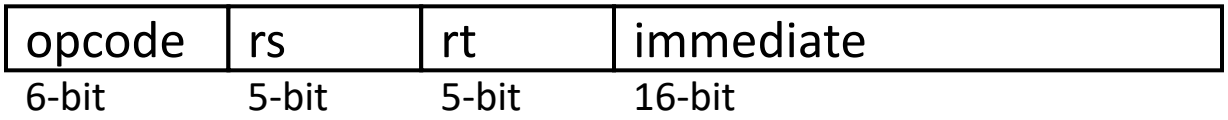
MIPS Instruction Format

- R-type, 3 register operands



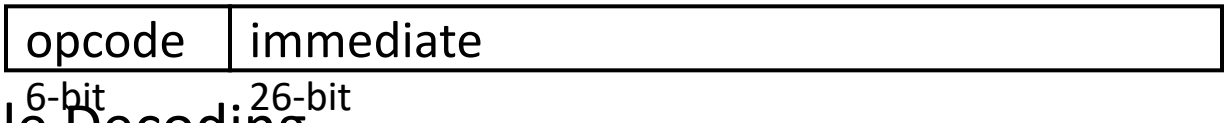
R-type

- 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

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

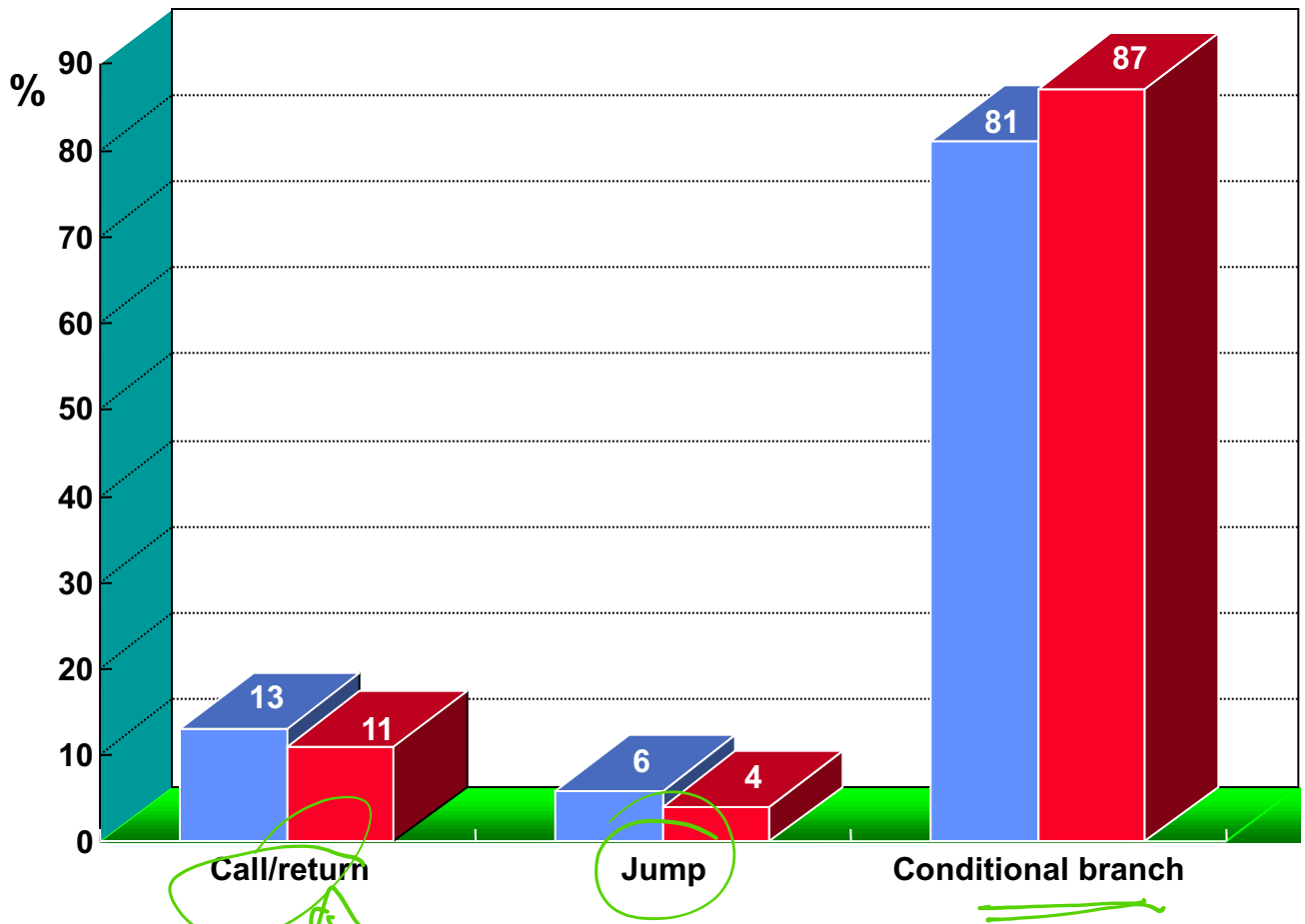
Majority of references to scalar
- 80% are local to a procedure
- References to arrays/structure require index or pointer

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

Operations in the Instr. Set

Rank	80x86 instructions	Integer 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 reg-reg	4%
9	call	1%
10	return	1%
Total		96%

Control Flow Instructions



Pipelined Arch

Thank You