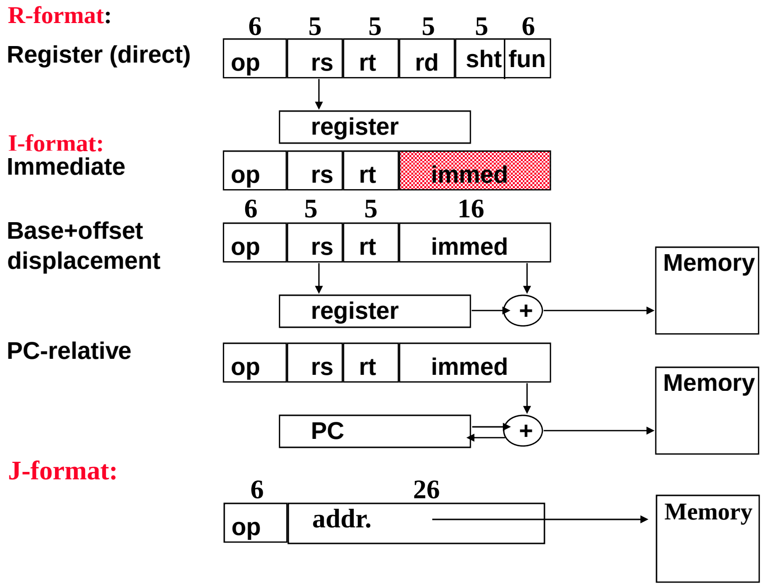
CS4200/5200
Computer Architecture I

Instruction Set Principles

Dr. Xiaobo Zhou
Department of Computer Science

Review: MIPS Addressing Modes/Instruction Formats



R-format:
Register (direct)

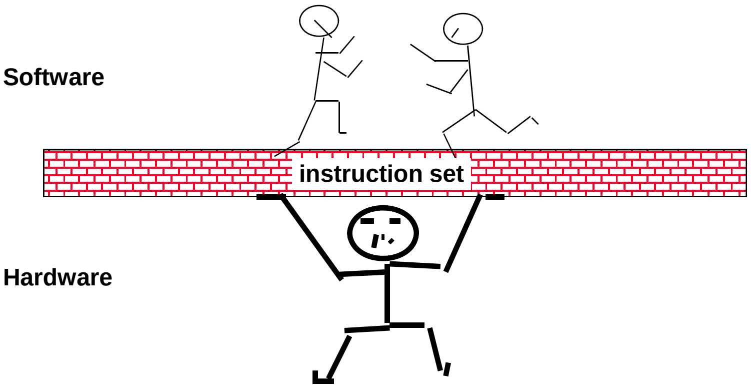
I-format:
Immediate

Base+offset displacement

PC-relative

J-format:

Review: Instruction Set Design



Software

instruction set

Hardware

An instruction is a binary code, which specifies a basic operation (e.g. add, subtract, and, or) for the computer
- **Operation Code: defines the operation type**
- **Operands: operation source and destination**

Basic Issues in Instruction Set Design

--- What operations (and how many) should be provided

LD/ST/INC/BRN sufficient to encode any computation
But not useful because programs too long!

--- How (and how many) operands are specified

Most operations are dyadic (eg, A <- B + C)
Some are monadic (eg, A <- ~B)

--- How to encode these into consistent instruction formats

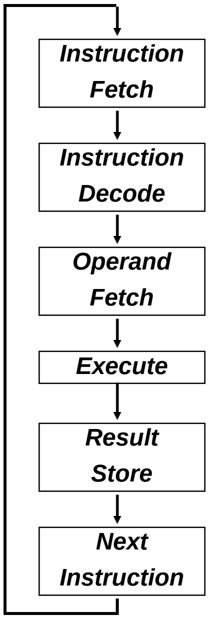
Instructions should be multiples of basic data/address widths

Typical instruction set:

° 32 bit word
° basic operand addresses are 32 bits long
° basic operands, like integers, are 32 bits long
° in general case, instruction could reference 3 operands (A := B + C)

challenge: encode operations in a small number of bits!

Execution Cycle



Obtain instruction from program storage

Determine required actions and instruction size

Locate and obtain operand data

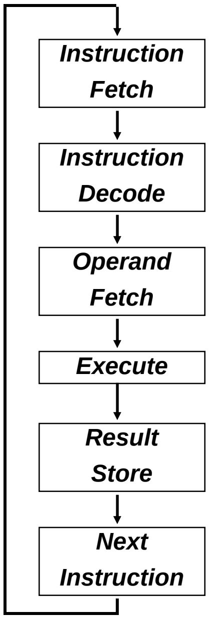
Compute result value or status

Deposit results in storage for later use

Parallelism is important!

Determine successor instruction

What Must be Specified?



° Instruction Format or Encoding
   – how is it decoded?

° Location of operands and result
   – where other than memory?
   – how many explicit operands?
   – how are memory operands located?
   – which can or cannot be in memory?

° Data type and Size

° Operations
   – what are supported

° Successor instruction
   – jumps, conditions, branches

- fetch-decode-execute is implicit!

Basic ISA Classes

Accumulator: (earliest machines)

1 address	load/store A	acc←/→ mem[A]
1 address	add A	acc ← acc + mem[A]

Stack: (HP calculator, Java virtual machines)

0 address	add	tos ← tos + next

Register (register-Memory): (e.g. Intel 80x86, Motorola 68xxx)

2 address	add A B	EA(A) ← EA(A) + EA(B)
3 address	add A B C	EA(A) ← EA(B) + EA(C)

Register (Load/Store): (e.g. SPARC, MIPS, PowerPC)

3 address	add Ra Rb Rc	Ra ← Rb + Rc
	load Ra Rb	Ra ← mem[Rb]
	store Ra Rb	mem[Rb] ← Ra

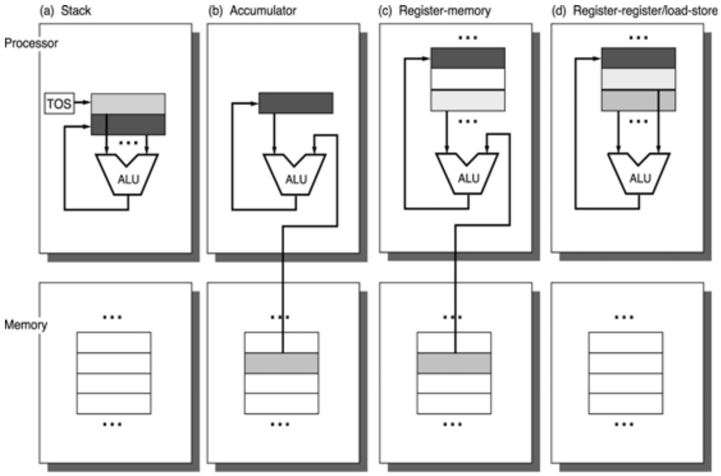
Memory – to - memory: no more shipping today

Classifying ISAs

1. Dimension 1: Where other than memory?
   - Accumulator
   - Stack
   - A set of registers

2. Naming Implicitly or explicitly?
   - Implicitly:
     - Accumulator
     - Stack; operands identified by TOS
   - Explicitly
     - General-purpose register architectures
       - Either registers or memory locations

Operand locations for the ISAs



lighter shade: inputs; dark shade: result

Comparing Instructions

Comparing Number of Instructions

° Code sequence for C = A + B for four classes of instruction sets:

Stack	Accumulator	Register (register-memory)	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

Add R3, R1, B
Store R3, C

S[tos - 4] = S[tos] op S[tos - 4];
tos = tos – 4;

Number of Instructions? Cycles per instruction?

General Purpose Registers Dominate

- Since 1975 all machines use general purpose registers
  (Java Virtual Machine adopts Stack architecture)
- Advantages of registers
  - registers are faster than memory
  - registers are easier for a compiler to use
    - e.g., (A*B) – (C*D) – (E*F) can do multiplies in any order
      vs. stack (S[tos-4] = S[tos] op S[tos-4]; tos = tos – 4)
  - registers can hold variables
    - memory traffic is reduced, so program is sped up
      (since registers are faster than memory)
    - code density improves (since register named with fewer bits than memory location)
  - registers are efficient in pipelining
- how many registers are sufficient?
  - Compilers reserve some for expression evaluation, parameter passing, and the remainder to hold variables.

Examples of Register Usage

Number of memory addresses per typical ALU instruction	Maximum number of operands per typical ALU instruction	Type of architecture	Examples
0	3	L-S	Alpha, ARM, SPARC, MIPS, Power PC. TM32
1	2/3	R-M	IBM 360/370, Intel 80x86, Motorola 68000
2	2	M-M	VAX (also has 3-operand formats)
3	3	M-M	VAX (also has 2-operand formats)

Typical combinations of memory operands and total operands per instruction

Example:

In VAX: ADDL (R9), (R10), (R11)

mem[R9] <-- mem[R10] + mem[R11]

VAX: richest of addressing modes
fewest restrictions on memory addressing

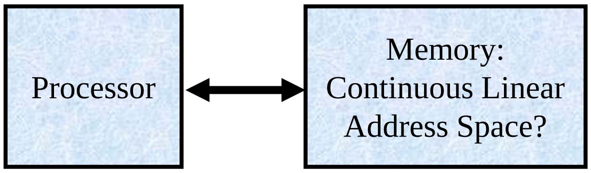
In MIPS:

```
lw   R1, (R10);         load a word
lw   R2, (R11)
add  R3, R1, R2;        R3 <-- R1+R2
sw  R3, (R9);           store a word
```

Pros and Cons of Number Memory Operands/Operands

- Register–register: 0 memory operands/instr, 3 (register) operands/instr
  - + Simple, fixed-length instruction encoding. Simple code generation model. Instructions take similar numbers of clocks to execute
  - – Higher instruction count than architectures with memory references in instructions. Some instructions are short and bit encoding may be wasteful.
- Register–memory (1,2)
  - + Data can be accessed without loading first. Instruction format tends to be easy to encode and yields good density.
  - – Operands are not equivalent since a source operand in a binary operation is destroyed. Encoding a register number and a memory address in each instruction may restrict the number of registers. Clocks per instruction varies by operand location.
- Memory–memory (2,2) or (3,3)
  - + Most compact. Doesn’t waste registers for temporaries.
  - – Large variation in instruction size, especially for three-operand instructions. Also, large variation in work per instruction. Memory accesses create memory bottleneck.

Memory Addressing



Since 1980 almost every machine uses addresses to level of 8-bits (byte)

2 questions for design of ISA:

- **Since could read a 32-bit word as four loads of bytes from sequential byte addresses or as one load word from a single byte address,**

  How do byte addresses map onto words?

  Can a word be placed on any byte boundary?

Alignment Restriction

Alignment issue:
access objects larger than a byte must be aligned

- **Alignment: require that objects fall on address that is multiple of their size**
- **An access to an object of size *s* bytes at byte address *A* is aligned if *A* Mod *s* = 0**
- **Alignment leads to faster data transfers**
- **Example:**

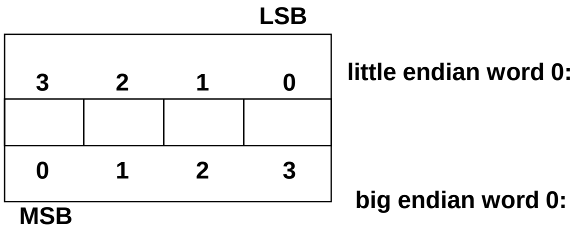
Addressing Objects

Big Endian: address of most significant (MSB) = word address

IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA

Little Endian: address of least significant (LSB) = word address

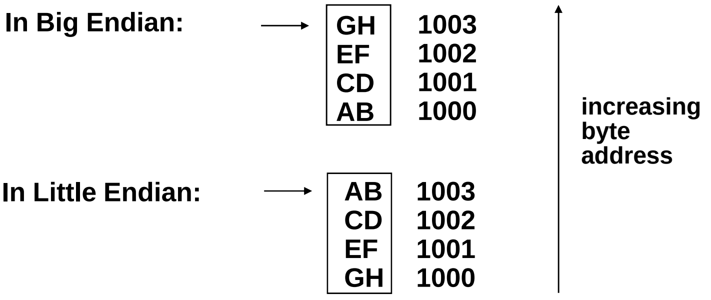
Intel 80x86, DEC Vax



BIG Endian versus Little Endian

Example 1: Memory layout of a number #ABCDEFGH:

SW $4(#ABCDEFGH), 1000($0)



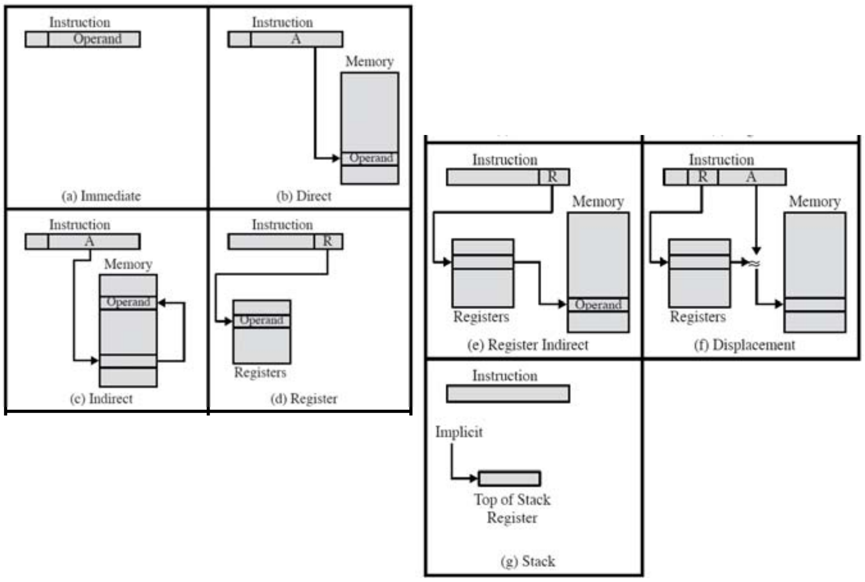
Example 2: Memory layout of a number #FF00H

How about load?

Addressing Modes

Addressing mode	Example	Meaning
Register	Add R4,R3	R4← R4+R3
Immediate	Add R4,#3	R4 ← R4+3
Displacement	Add R4,100(R1)	R4 ← R4+Mem[100+R1]
Register indirect	Add R4,(R1)	R4 ← R4+Mem[R1]
Indexed	Add R3,(R1+R2)	R3 ← R3+Mem[R1+R2]
Direct or absolute	Add R1,(1001)	R1 ← R1+Mem[1001]
Memory indirect	Add R1,@(R3)	R1 ← R1+Mem[Mem[R3]]
Auto-increment	Add R1,(R2)+	R1 ← R1+Mem[R2]; R2 ← R2+d
Auto-decrement	Add R1,–(R2)	R2 ← R2–d; R1 ← R1+Mem[R2]
Scaled	Add R1,100(R2)[R3]	R1 ← R1+Mem[100+R2+R3*d]

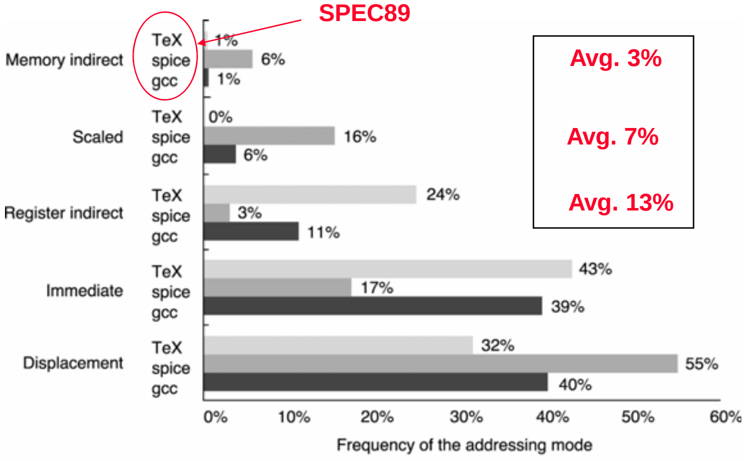
Addressing Mode Illustrations



Addressing Mode:

- Addressing modes have the ability to significantly reduce instruction counts
- They also add to the complexity of building a machine, and may increase the CPI of computers
- The usage of various addressing modes is important in helping the architect choose what to include

Addressing Mode Usage (VAX)



SPEC89

Avg. 3%

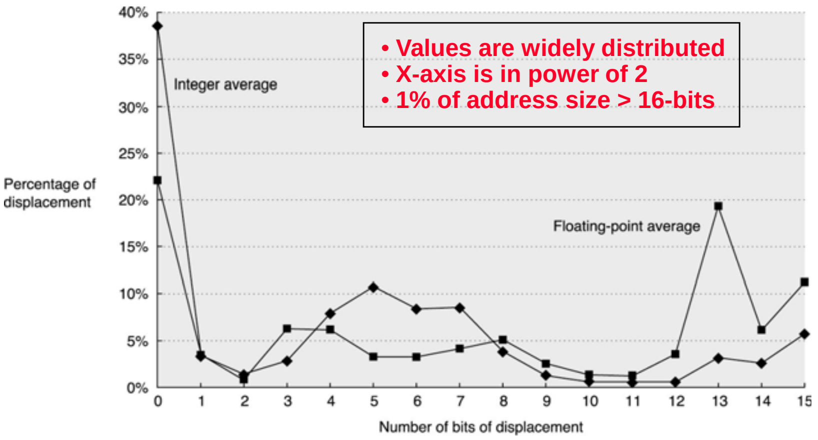
Avg. 7%

Avg. 13%

What it tells?

- important addressing modes: Displacement, Immediate, Register Indirect

Displacement Address Size (Alpha)



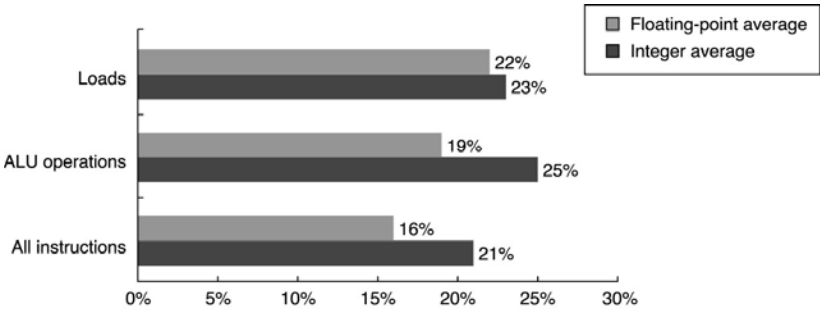
- Values are widely distributed
- X-axis is in power of 2
- 1% of address size > 16-bits

What it tells?

- displacement size should be 12-16 bits, capturing 75%-99%

Why important? directly affects the instruction length

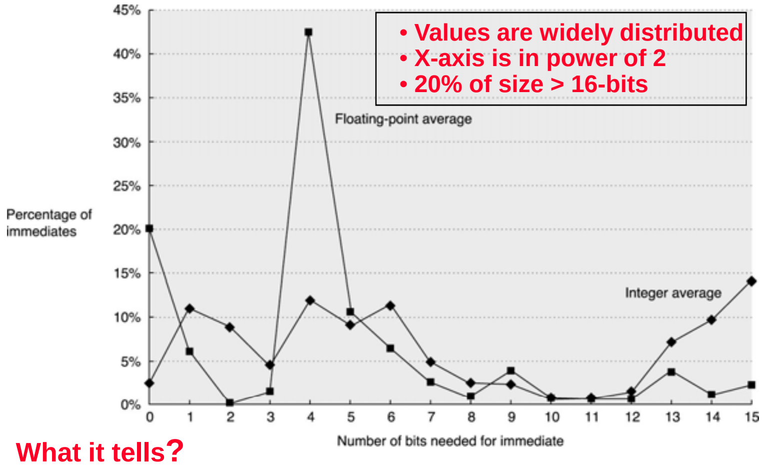
Immediate Frequency (Alpha)



What it tells?

- About one quarter of data transfers and ALU operations have an immediate Operand.

The distribution of Immediate Size (Alpha)



- Values are widely distributed
- X-axis is in power of 2
- 20% of size > 16-bits

What it tells?

- immediate size should be 8-16 bits, capturing 50%-80%

© 2003 Elsevier Science (USA). All rights reserved.

Why important? affects the instruction length too

Summary of Addressing Modes

- **Data Addressing modes that are important:**
  Displacement, Immediate, Register Indirect
- **Displacement size should be 12 to 16 bits**
- **Immediate size should be 8 to 16 bits**

Data Types and Sizes

Bit: 0, 1

Bit String: sequence of bits of a particular length
4 bits is a nibble
8 bits is a byte
16 bits is a half-word
32 bits is a word

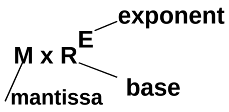
Character:
ASCII 7 bit code
EBCDIC 8 bit code (IBM)
UNICODE 16 bit code (Java)

Decimal:
digits 0-9 encoded as 0000b thru 1001b
two decimal digits packed per 8 bit byte

Integers:

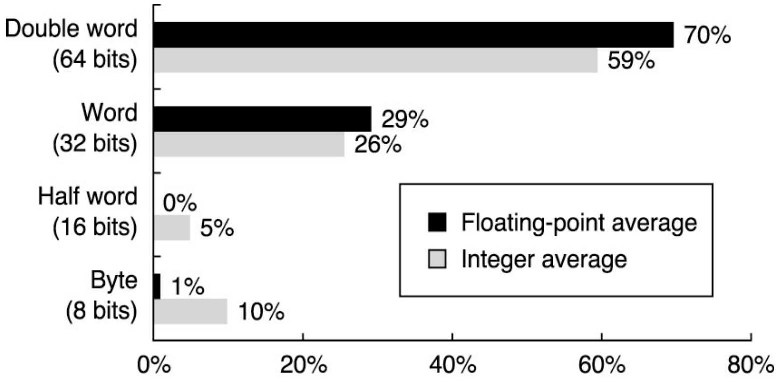
Sign & Magnitude:	**0X vs. 1X**	**Positive #'s same in all**
1's Complement:	**0X vs. 1(~X)**	**First 2 have two zeros**
2's Complement:	**0X vs. (1's comp) + 1**	**Last one usually chosen**

Floating Point:
Single Precision
Double Precision
Extended Precision


$M \times R^E$
exponent
base
mantissa

How many +/- sign's?
Where is decimal pt?
How are +/- exponents represented?

Operand Size Usage (on 64-bit addresses)



•Support these data sizes and types:
8-bit, 16-bit, 32-bit integers and
32-bit and 64-bit IEEE 754 floating point numbers

Typical Operations

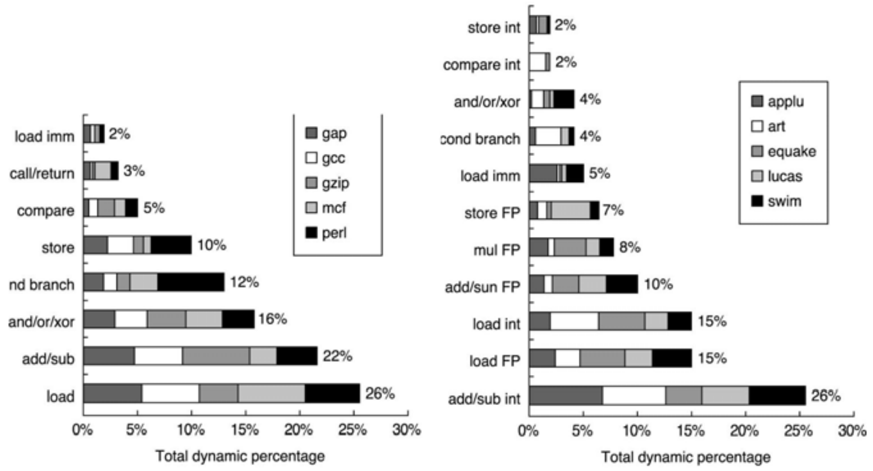
Data Movement	Load (from memory) Store (to memory) memory-to-memory move register-to-register move input (from I/O device) output (to I/O device) push, pop (to/from stack)
Arithmetic	integer (binary + decimal) or FP Add, Subtract, Multiply, Divide
Logical	not, and, or, set, clear
Shift	shift left/right, rotate left/right
Control (Jump/Branch)	unconditional, conditional
Subroutine Linkage	call, return
Interrupt	trap, return
Synchronization	test & set (atomic r-m-w)
String	move, compare, search, translate

Top 10 80x86 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%

° Simple instructions dominate instruction frequency

Most Popular MIPS Instructions



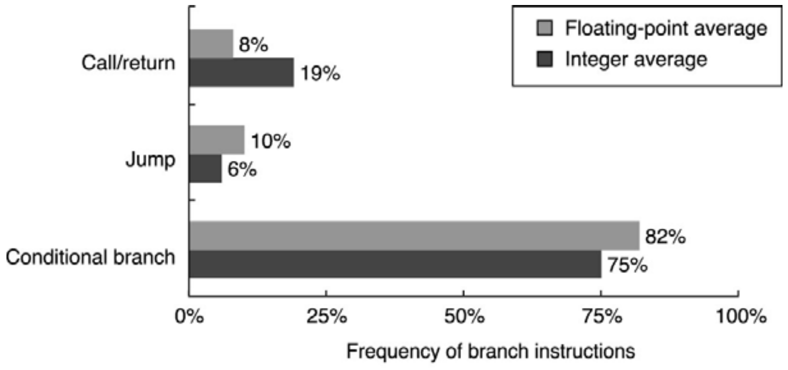
Left: SPECint2000 (96%) **Right: SPECfp2000 (97%)**

Operation Summary

- **Support these simple instructions, since they will dominate the number of instructions executed:**

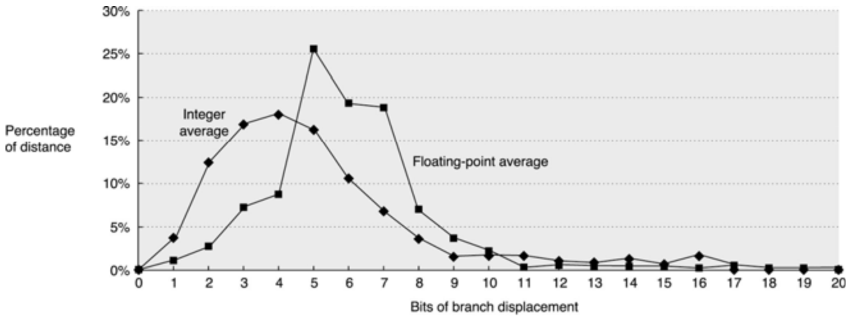
load,
store,
add,
subtract,
move register-register,
and,
shift,
compare equal, compare not equal,
branch (with a PC-relative address at least 8-bits long),
jump,
call,
return;

Instructions for Control Flow (Alpha)



•Conditional branches; jumps; procedure call/return

Branch Distances (Alpha)

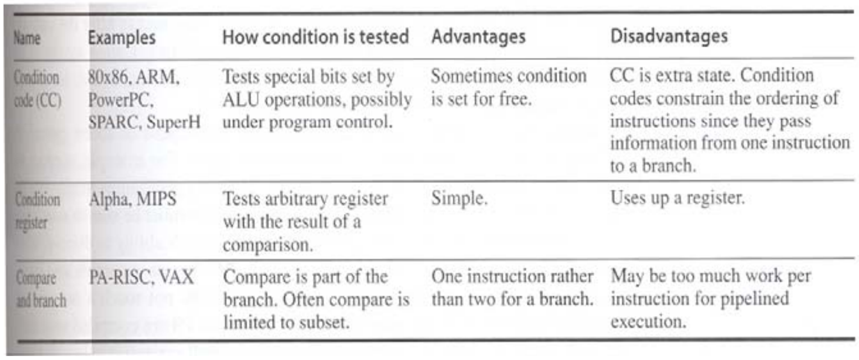


- Branch distances in terms of number of instructions Between the target and the branch instruction.

What it tells?

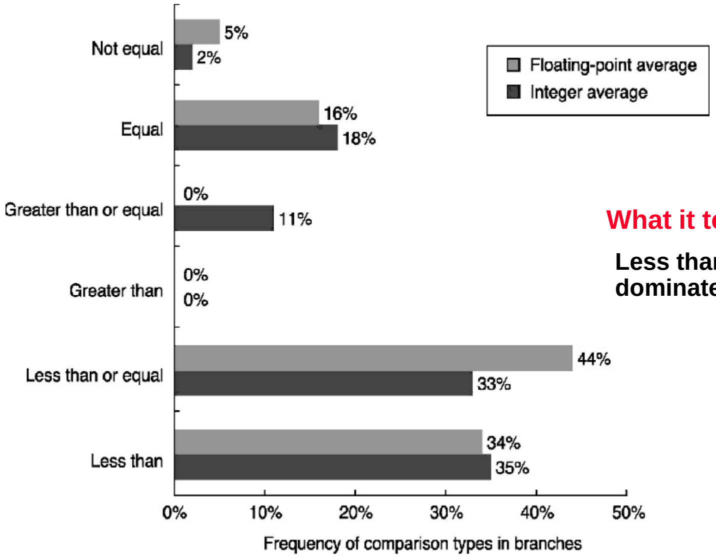
- Most branches are to targets that can be encoded in 4-8 bits; short displacement fields often suffice for branches; Important if coding density is the issue!

Conditional Branch Options



Name	Examples	How condition is tested	Advantages	Disadvantages
Condition code (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Tests special bits set by ALU operations, possibly under program control.	Sometimes condition is set for free.	CC is extra state. Condition codes constrain the ordering of instructions since they pass information from one instruction to a branch.
Condition register	Alpha, MIPS	Tests arbitrary register with the result of a comparison.	Simple.	Uses up a register.
Compare and branch	PA-RISC, VAX	Compare is part of the branch. Often compare is limited to subset.	One instruction rather than two for a branch.	May be too much work per instruction for pipelined execution.

Frequency of Compares (Alpha)



What it tells?

Less than (or equal) dominates

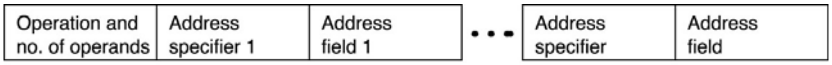
Encoding an Instruction Set

- Balance several competing forces
  - The desire to have as many registers and addressing modes
  - The impact of the size of the registers and addressing mode fields on the average instruction size and hence the average program size
  - A desire to have instructions encoded into lengths that will be easy to handle in a pipelined implementation; many desktop and server architects have chosen to use a fixed-length instruction to gain implementation benefits while sacrificing average code size

Example: add EAX, 1000 (EBX)

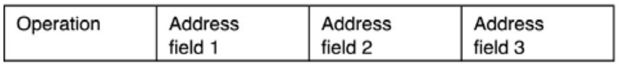
1 + 1 + 4 = 6 bytes (in 80x86 32-bit mode)

Generic Examples of Instruction Formats



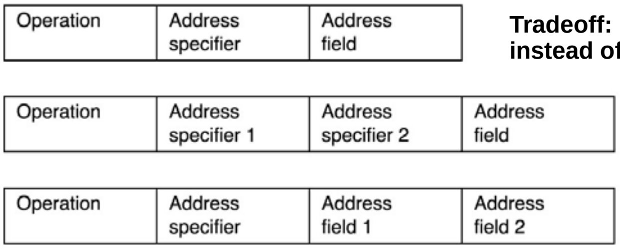
(a) Variable (e.g., VAX, Intel 80x86)

low avg. code size -- best when many addressing modes but poor perf.



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

easy decoding for compiler, easy pipelining but wasted bits in instr.



Tradeoff: multiple of bytes, instead of arbitrary bit length

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

Instruction Formats and Reduced Code Size

- If code size is most important, use variable length instructions
- If performance is most important, use fixed length instructions – ease of decoding
- Recent embedded machines (ARM, MIPS) added optional mode to execute subset of 16-bit wide instructions (Thumb, MIPS16), which both claim a code size reduction of up to 40%
- Some architectures actually exploring on-the-fly *hardware decompression* for more density (IBM).

Instruction Set Architectures

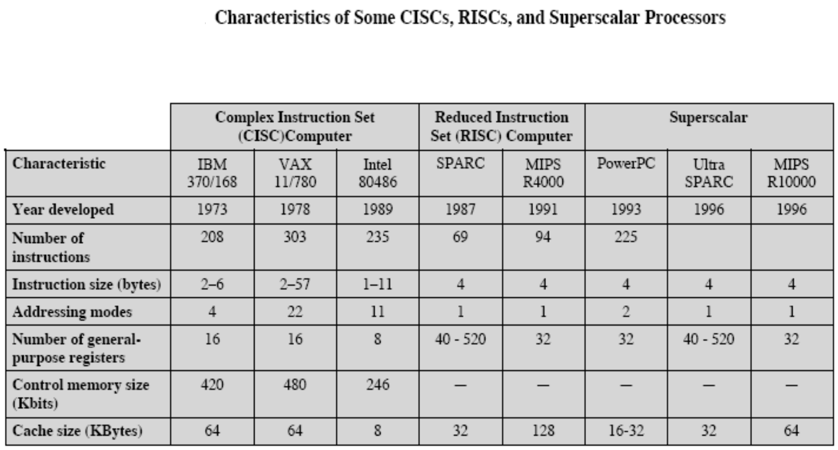
- Class of ISA
- Memory addressing
- Addressing modes
- Types and sizes of operands
- Operations
- Control flow instructions
- Encoding an ISA

Reduced Instruction Set Computer (RISC)

Key elements

- A large number of general-purpose registers, and/or the use of compiler technology to optimize register usage
- A limited and simple instruction set
- An emphasis on optimizing the instruction set
  - A single instruction size (typically 4 bytes)
  - Register-to-register operations
    - No operations that combines load/store with arithmetic
  - A small number of data addressing modes
  - Simple addressing modes
    - No indirect addressing
  - Simple instruction formats

CISC and RISC Characteristics



Characteristics of Some CISCs, RISCs, and Superscalar Processors

	Complex Instruction Set (CISC)Computer			Reduced Instruction Set (RISC) Computer		Superscalar		
Characteristic	IBM 370/168	VAX 11/780	Intel 80486	SPARC	MIPS R4000	PowerPC	Ultra SPARC	MIPS R10000
Year developed	1973	1978	1989	1987	1991	1993	1996	1996
Number of instructions	208	303	235	69	94	225		
Instruction size (bytes)	2–6	2–57	1–11	4	4	4	4	4
Addressing modes	4	22	11	1	1	2	1	1
Number of general-purpose registers	16	16	8	40 - 520	32	32	40 - 520	32
Control memory size (Kbits)	420	480	246	—	—	—	—	—
Cache size (KBytes)	64	64	8	32	128	16-32	32	64

CISC Program size vs. RISC program size

- Is it certain that a CISC program will be smaller than a corresponding RISC program?
  - in many cases, a CISC program, expressed in symbolic machine language, may be shorter (fewer instructions)
  - but the number of bits of memory occupied may not be noticeably smaller

Machine Examples: Address & Registers

Intel 8086	2^{20} x 8 bit bytes AX, BX, CX, DX SP, BP, SI, DI CS, SS, DS IP, Flags	acc, index, count stack, string code,stack,data segment
VAX 11	2^{32} x 8 bit bytes 16 x 32 bit GPRs	r15-- program counter r14-- stack pointer r13-- frame pointer r12-- argument ptr
MC 68000	2^{24} x 8 bit bytes 8 x 32 bit GPRs 7 x 32 bit addr reg 1 x 32 bit SP 1 x 32 bit PC	
MIPS	2^{32} x 8 bit bytes 32 x 32 bit GPRs 32 x 32 bit FPRs HI, LO, PC	

Concluding Remarks

- Changes in 1990s
  - Address size doubles
    - Instruction sets: 32-bit addresses → 64-bit addresses
    - Registers: 32-bit → 64-bit
  - Optimization of cache performance
    - Pre-fetch instructions were added (Memory Hierarchy)
  - Support for Multimedia
    - Instruction sets extended for MM and DSP applications
- Trends in ISA design
  - Long instruction words
    - More instruction-level parallelism (Pipelining)
  - Blending general-purpose and DSP architectures
  - 80x86 emulation
    - Given the popularity of software for 80x86 architecture, see if changes to the instruction sets can improve performance, cost or power when emulating the 80x86 architecture

Lecture Summary: ISA

- Use general purpose registers with a load-store architecture;
- Support these addressing modes: displacement (with an address offset size of 12 to 16 bits), immediate (size 8 to 16 bits), and register deferred;
- Support these simple instructions, since they will dominate the number of instructions executed: load, store, add, subtract, move register-register, and, shift, compare equal, compare not equal, branch (with a PC-relative address at least 8-bits long), jump, call, and return;
- Support these data sizes and types: 8-bit, 16-bit, 32-bit integers and 64-bit IEEE 754 floating point numbers;
- Use fixed instruction encoding if interested in performance and use variable instruction encoding if interested in code size;
- Provide at least 16 general purpose registers plus separate floating-point registers, be sure all addressing modes apply to all data transfer instructions, and aim for a minimalist instruction set.

Reading

- **Reading:**
  CO4: Chapter 2 (MIPS)
  CA 5: Appendix A (ISA)

- **Preview:**
  CO4: Chapter 4 (The Processor)

Links to Information Assurance related Websites

- **National Security Agency:** http://www.nsa.gov/
- **NIST, Computer Security Division, Computer Security Resource Center:** http://csrc.nist.gov/
- **Common Criteria for Information Technology Security Evaluation:** http://www.commoncriteriaportal.org/
- **U.S. Department of Homeland Security:** http://www.dhs.gov/
- **ITU (International Telecommunication Union:** http://www.itu.int/
- **Internet Society (ISOC):** http://www.isoc.org/
- **The Internet Engineering Task Force (IETF):** http://www.ietf.org/
- **Internet Architecture Board (IAB):** http://www.iab.org/
- **International Organization for Standardization (ISO):** http://www.iso.org
- **IEEE Computer Society:** http://www.computer.org
- **Association for Computing Machinery (ACM):** http://www.acm.org/
- **USENIX: The Advanced Computing Systems Association:** http://www.usenix.org/

Homework 2, due 1 week later

- **Re-do (optional, no extra credits) all examples in MIPS ISA (refer to CO 4, Chapter 2)**
- **Homework; see course Web site**
- **Reading assignment; see course Web site**