

Instruction Set Principles and Examples

Review of Instruction Set Architecture Principles and Examples
Topics

- Classifying Instruction Set Architectures
- Memory Addressing
- Operations in the Instruction Set
- The DLX Architecture
- The Role of Compiler

Classifying Instruction Set Architectures

Top-level Taxonomy Based on CPU Internal Stores

- Stack Architecture: The internal store is organized as a stack.
- Accumulator Architecture: A register is reserved as accumulator.
- General Purpose Register (GPR) Architectures: Registers are addressed explicitly.
 - Register-Memory Machines
 - Load-Store (Register-Register) Machines

Modern architectures tend to be GPR register-register machines.

- Registers are fast and can be used for variables.
- Instruction take similar numbers of clocks to execute, easing the design of pipelining.

GPR Architecture Breakdown

- the number of operands: 2 or 3
- the number of memory operands: from 0 to 2 or 3.

No. of Memory Accesses	No. of Operands	Examples
0	3	SPARC, MIPS, PowerPC, ALPHA
1	2	Intel 80x86, Motorola 68000
2	2	VAX (two-operand format)
3	3	VAX (three-operand format)

Example (Exercise 2.3): Compare different architectures in terms of memory memory bandwidth (code + data). Assume

- Opcode is 1 byte.
- All memory addresses are 2 bytes.
- All data operands are 4 bytes.
- All instructions are an integral number of bytes in length.

Memory Addressing

Interpreting Memory Addresses

- Big and little ending for the objects with multiple bytes (i.e. 4-byte word)
 - big ending: the byte at $xx..xx00$ is the most significant byte.
 - little ending: the byte at $xx..xx00$ is the least significant byte.
- Alignment: The address A of objects with s bytes must satisfy

$$A \bmod s = 0$$

to avoid cross-boundary memory references.

Addressing Modes – how the address of an object is specified

- Register Mode : Use register name to address the object in registers.
 - one half of the operands in SPEC89 generated for VAX
- Immediate Mode: Objects are part of the instruction.
- Memory Mode: the way to address the objects in memory

Addressing Modes	Notation	Meaning
Displacement	100(R1)	Mem[100+Regs[R1]]
Register deferred(indirect)	(R1)	Mem[Regs[R1]]
Indexed	(R1+R2)	Mem[Regs[R1]+Regs[R2]]
Direct(Absolute)	(1001)	Mem[1001]
Memory indirect(deferred)	@(R3)	Mem[Mem[Regs[R3]]]
Autoincrement	(R3)+	Mem[Regs[R3]] Regs[R3] = Regs[R3] + s
Autodecrement	(R3)-	Mem[Regs[R3]] Regs[R3] = Regs[R3] - s
Scaled	100(R2)[R3]	Mem[100+Regs[R2]+Regs[R3]*s]

- Breakdown of another half for the memory and immediate modes

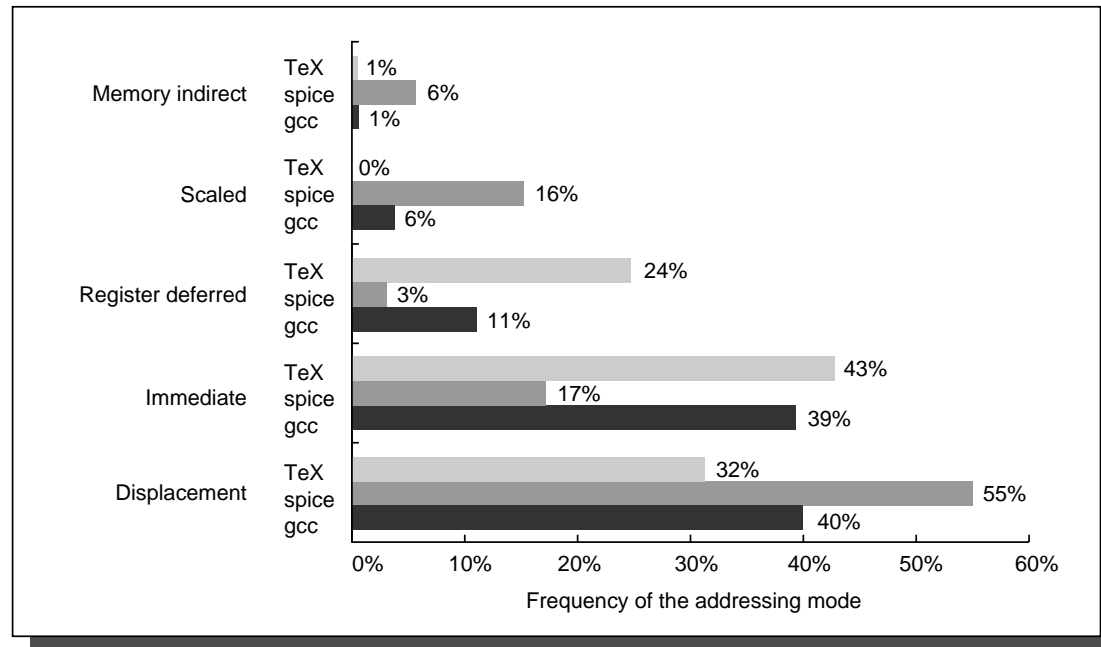


FIGURE 2.6 Summary of use of memory addressing modes (including immediates).

- Displacement dominates the memory addressing mode (32% – 55%).
- Immediate tails displacement (17% – 43%).

How many bits are sufficient for displacements?

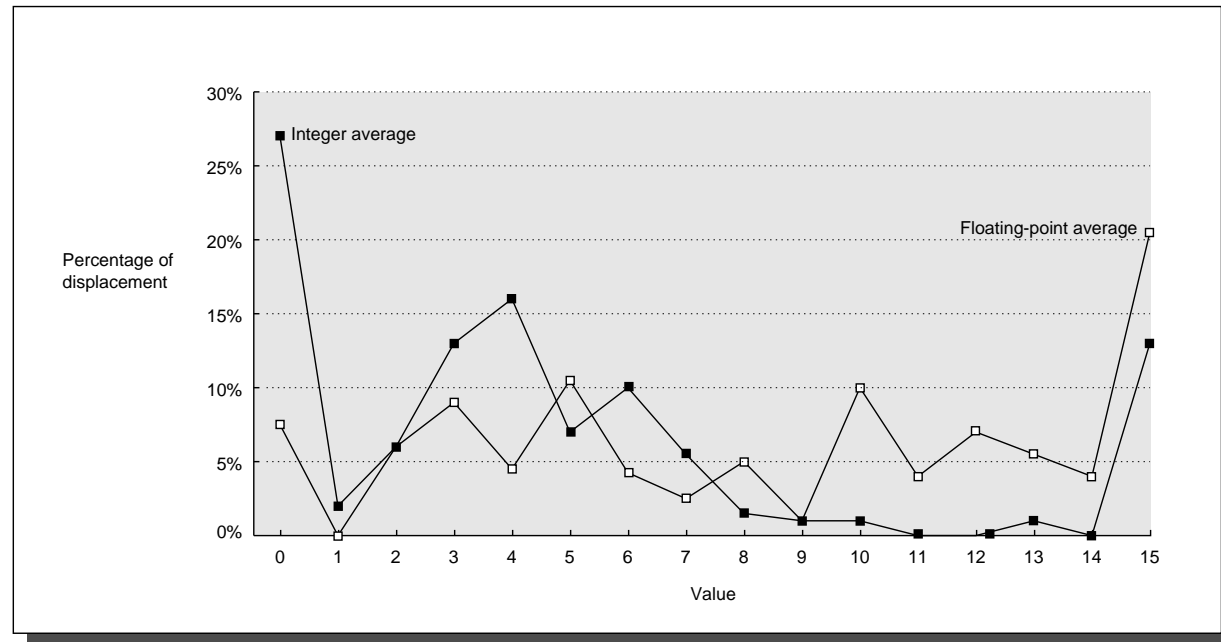


FIGURE 2.7 Displacement values are widely distributed.

- 12 bits would capture about 75%.
- 16 bits would capture about 99%.

Immediate Addressing Mode

- Percentage of operations that use immediate

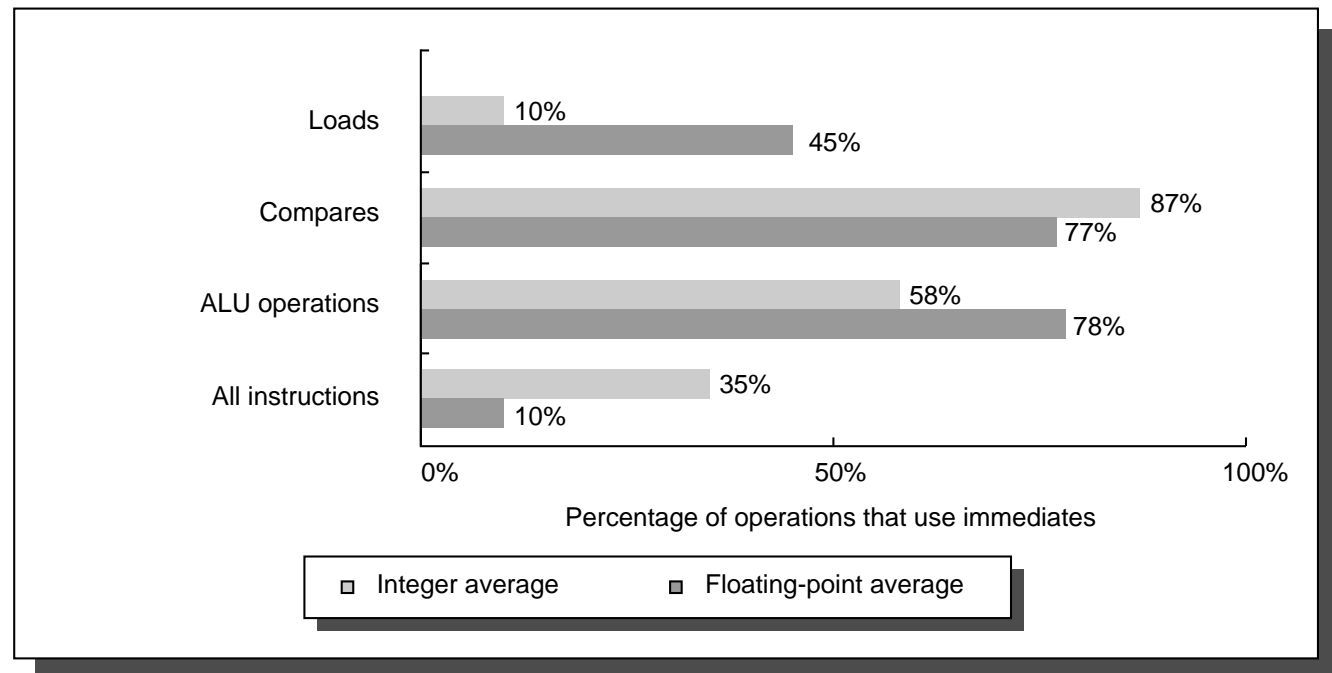


FIGURE 2.8 We see that for integer ALU operations about one-half to three-quarters of the operations have an immediate operand, while for integer compares 75% to 85% of the occurrences use an immediate operand.

– Compare dominates (77% – 87%)

- How many bits are sufficient for immediate?

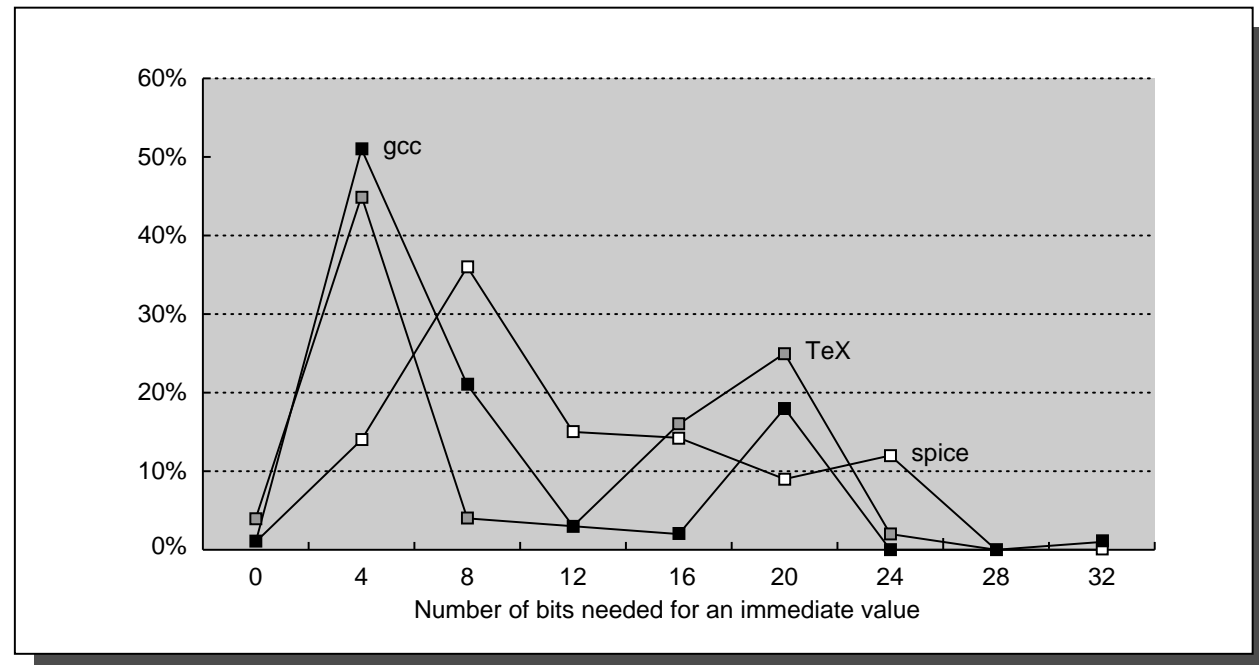


FIGURE 2.9 The distribution of immediate values is shown.

- 8 bits would capture about 50% to 75%.
- 16 bits would capture about 75% to 80%.

- Summary
 - New architecture should support at least the following addressing modes
 - * displacement
 - * immediate
 - * register deferred
 - The size of displacement should be at least 12 to 16 bits.
 - The size of immediate should be least 8 to 16 bits.

Operations in the Instruction Set

What are the common cases in the instruction set?

- Most widely executed instructions are the simple operations.
- IC percentage of five SPECint92 for 80x86
 - 96% are 10 simple operations as follows

operation	percentage
load	%22
conditional branch	%20
compare	%16
store	%12
add	%8
and	%6
sub	%5
move reg-reg	%4
call	%1
return	%1

New architecture should contain the these simple operations and make them fast.

Instructions for Control Flow

- Breakdown of control flow instructions

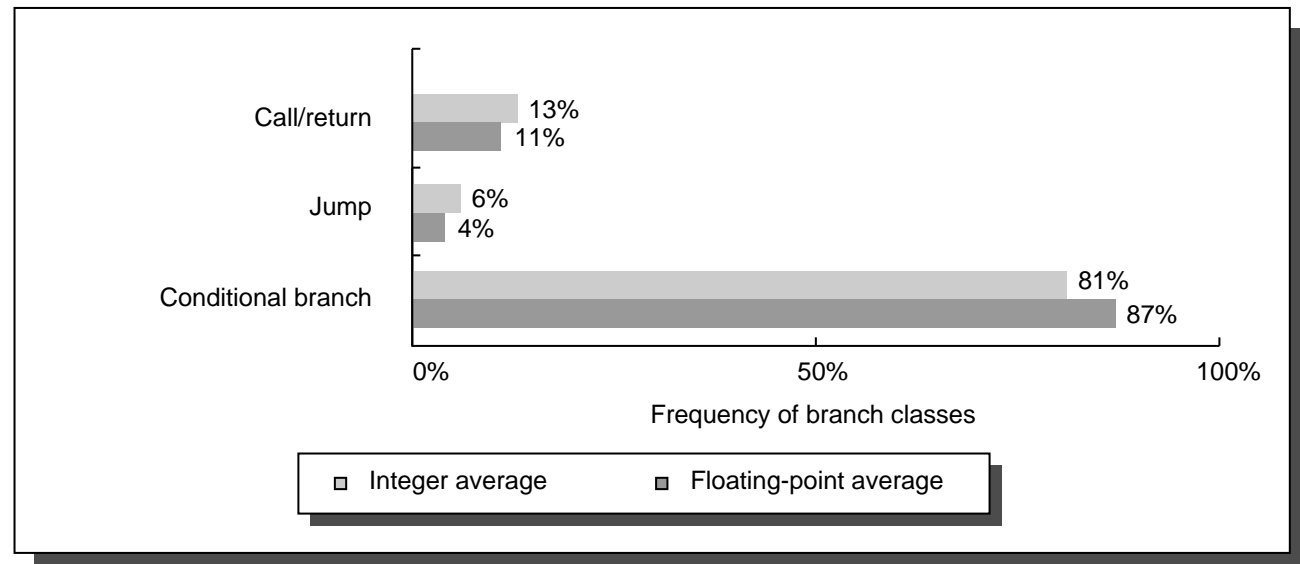


FIGURE 2.12 Breakdown of control flow instructions into three classes: calls or returns, jumps, and conditional branches.

- Target address
 - PC-relative for static addresses known at compile-time
 - register-differed or others for dynamic target addresses
 - distribution of displacement in PC-relative target addresses (DLX)

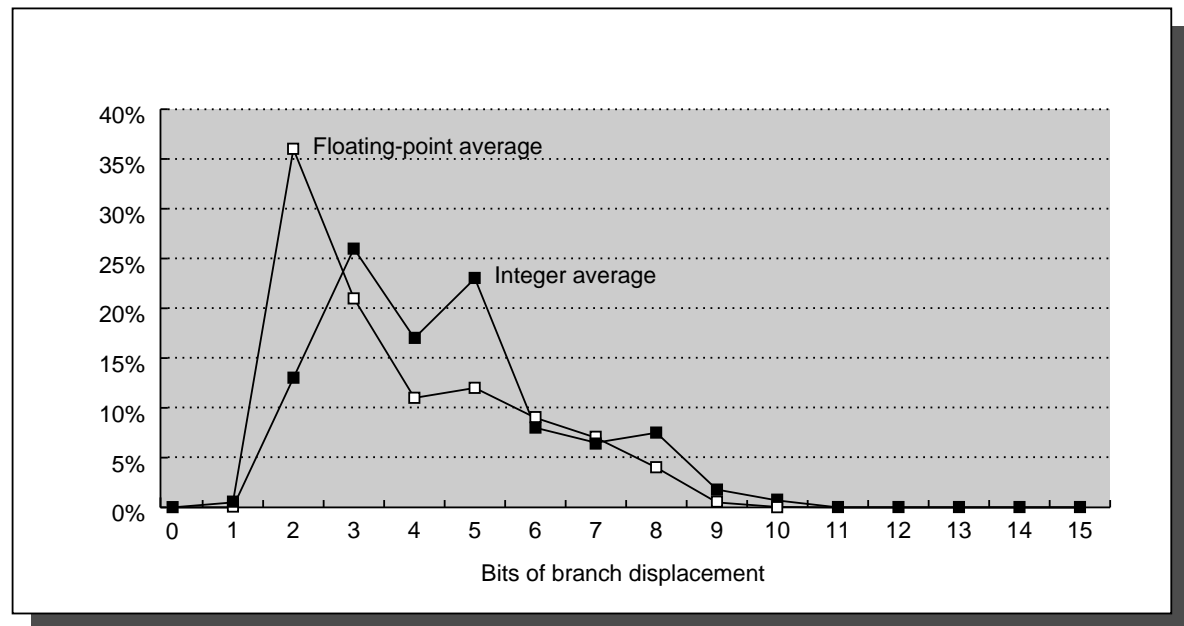


FIGURE 2.13 Branch distances in terms of number of instructions between the target and the branch instruction.

8 bits captures 95% of the displacement

Example (Exercise 2.1(a))

- A load-store machine with variable instruction lengths.
- Instruction Mix is shown Figure 2.26.
- Branch target and memory references have 0, 8 and 16 bits offsets.
- Accumulative percentage of offsets bits for branch targets and memory references is shown in Figure 2.32.
- What is the average lengths of an instruction?

Type and Size of Operands

- Modern architectures use opcode to designate type of operands (rather than tag as part of data)
 - characters – ASCII
 - integers – two's complement
 - floating-point numbers – IEEE standard 754
- Distribution of size of operands in memory

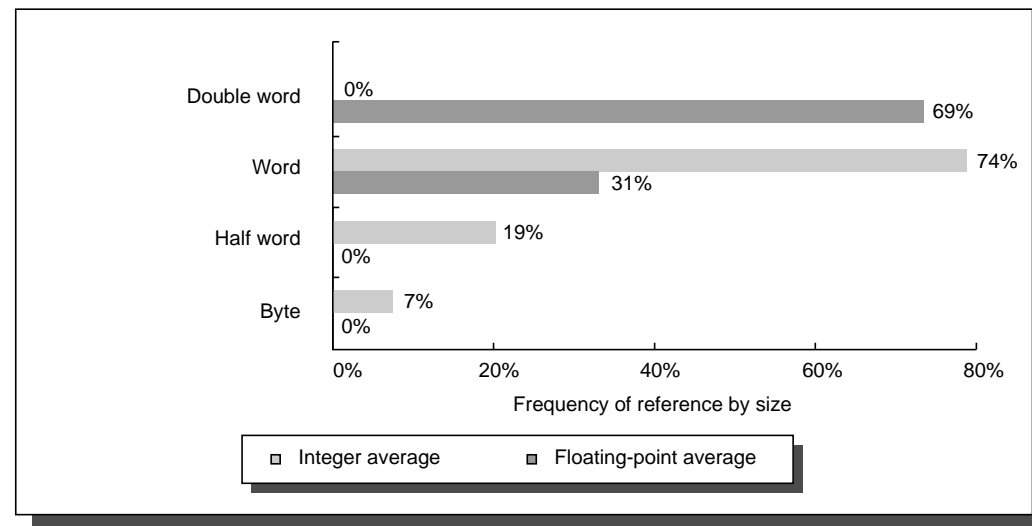


FIGURE 2.16 Distribution of data accesses by size for the benchmark programs.

Encoding an Instruction Set

- Fundamental trade-off: size of program vs. ease of decoding in CPU
- Three competing forces
 - desire to accommodate large number of addressing mode and registers
 - desire to reduce the size of instructions and programs
 - desire to ease the decoding and implementation of instructions
- Variable instruction length (VAX)
 - independence between addressing mode and opcode (operation)
 - use address specifier: (mode, reg)
- Fixed instruction length (DLX, MIPS)
 - small number of addressing modes
 - use opcode to imply the addressing mode

- Hybrid approach (80x86)

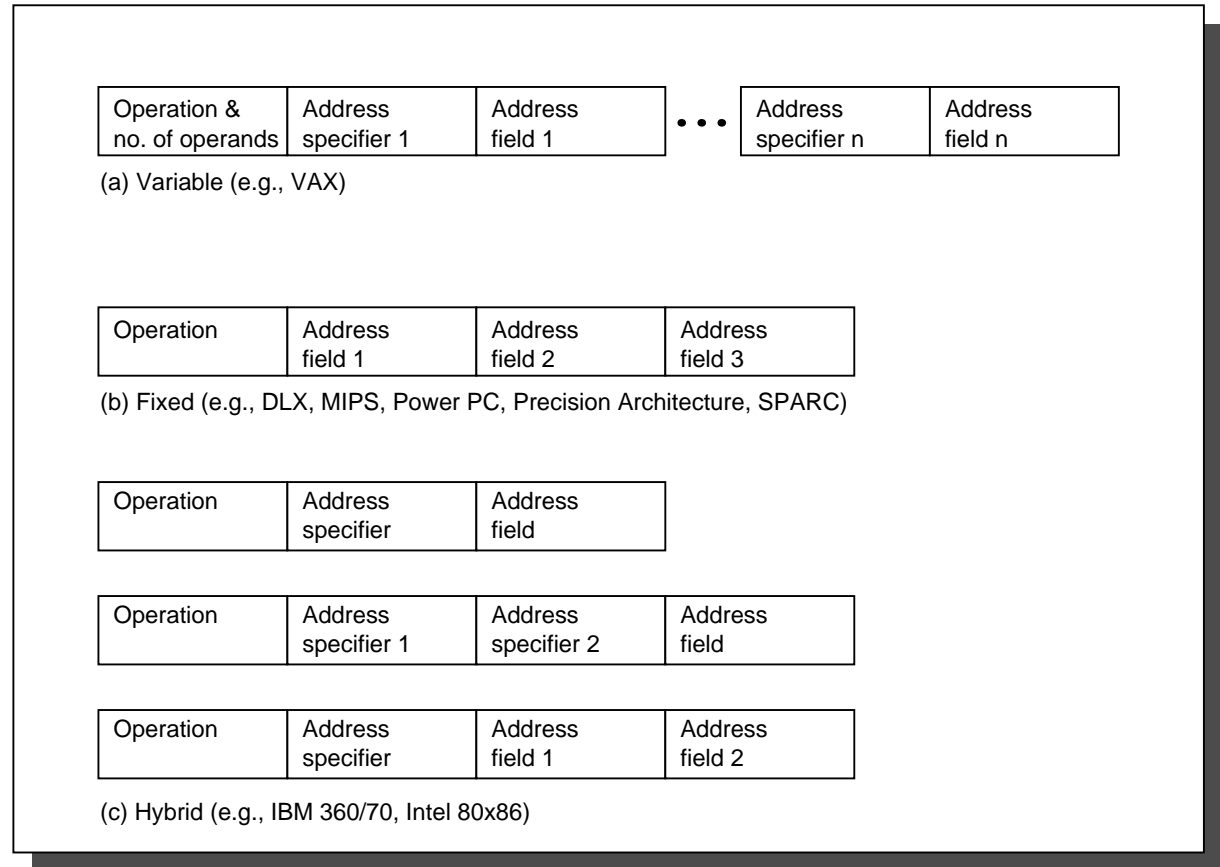


FIGURE 2.17 Three basic variations in instruction encoding.

The DLX Architecture

Characteristics of DLX Architecture

- GPR load-store architecture
 - 32 32-bit integer registers (R0, R1, ..., R31)
 - 32 32-bit floating-point registers (F0, F1, ..., F31)
- simple addressing modes
 - immediate (16-bit)
 - displacement (16-bit)
 - register deferred: implemented by displacement of 0
 - absolute (16-bit): implemented by a 16-bit displacement with R0

- simple data types
 - byte (8-bit)
 - half word integer (16-bit)
 - word integer (32-bit)
 - single precision floating point (32-bit)
 - double precision floating point (64-bit)
- fixed instruction length with three types
 - I-type for register-immediate and branch
 - R-type for register-register
 - J-type for Jump/call

DLX Instruction Format

I - type instruction



Encodes: Loads and stores of bytes, words, half words
 All immediates ($rd \leftarrow rs1 \text{ op immediate}$)

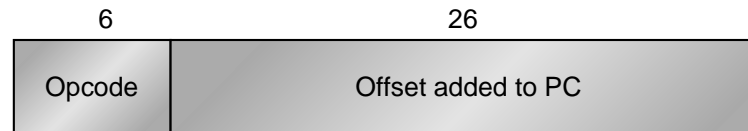
Conditional branch instructions (rs1 is register, rd unused)
 Jump register, jump and link register
 ($rd = 0, rs = \text{destination}, \text{immediate} = 0$)

R - type instruction



Register-register ALU operations: $rd \leftarrow rs1 \text{ func } rs2$
 Function encodes the data path operation: Add, Sub, . . .
 Read/write special registers and moves

J - type instruction



Jump and jump and link
 Trap and return from exception

- I-Type Instruction (rs1: 6..10, rd: 11..15, immediate: 16..31)
 - load and store: memory address is `immediate + Regs[rs1]`.
 - Immediate ALU: `Regs[rd] ← Regs[rs1] op immediate`.
 - Branch(BEQZ): Condition is `Regs[rs1] == 0`; target is `PC + immediate`.
 - Jump Register: Target is `Regs[rs1]`; (rd and immediate are unused).
- R-Type Instruction (rs1: 6..10, rs2: 11..15, rd: 16..20 func: 21..31)
 - reg-reg ALU: `Regs[rd] ← Regs[rs1] op Regs[rs2]` .
- J-Type Instruction (offset: 6..31)
 - Jump: Target is `PC + offset`.

DLX Instructions

- Data Transfer

Example	Name	Meaning
LW, R1, 30(R2)	load word	$\text{Regs}[R1] \leftarrow_{32} \text{Mem}[30+\text{Regs}[R2]]$
LB, R1, 30(R2)	load byte	$\text{Regs}[R1] \leftarrow_{32} (\text{Mem}[30 + \text{Regs}[R2]]_0)^{24} \text{## Mem}[30+\text{Regs}[R2]]$
LBU, R1, 30(R2)	load byte	$\text{Regs}[R1] \leftarrow_{32} 0^{24} \text{## Mem}[30+\text{Regs}[R2]]$
LH, R1, 30(R2)	load half word	$\text{Regs}[R1] \leftarrow_{32} (\text{Mem}[30 + \text{Regs}[R2]]_0)^{16} \text{## Mem}[30+\text{Regs}[R2]]$ $\text{## Mem}[31+\text{Regs}[R2]]$
LF, F0, 30(R2)	load float	$\text{Regs}[F0] \leftarrow_{32} \text{Mem}[30+\text{Regs}[R2]]$
LD, F0, 30(R2)	load double	$\text{Regs}[F0] \text{## Regs}[F1] \leftarrow_{64} \text{Mem}[30+\text{Regs}[R2]]$
SD, F0, 30(R2)	store double	$\text{Mem}[30+\text{Regs}[R2]] \leftarrow_{32} \text{Regs}[F0]$ $\text{Mem}[34+\text{Regs}[R2]] \leftarrow_{32} \text{Regs}[F1]$
SH, R3, 30(R2)	store half	$\text{Mem}[30+\text{Regs}[R2]] \leftarrow_{16} \text{Regs}[R3]_{16..31}$

- Arithmetic/Logical

Example	Name	Meaning
ADD R1, R2, R3	Add	$\text{Regs}[R1] \leftarrow_{32} \text{Regs}[R2] + \text{Regs}[R3]$
ADDI R1, R2, #3	Add immediate	$\text{Regs}[R1] \leftarrow_{32} \text{Regs}[R2] + 3$
LHI R1, #3	Load high immediate	$\text{Regs}[R1] \leftarrow_{32} 3\#\#0^{16}$

- Control

Example	Name	Meaning
J name	Jump	$\text{PC} \leftarrow \text{name};$ $\text{PC} + 4 - 2^{25} \leq \text{name} \leq \text{PC} + 4 + 2^{25}$
JAL name	Jump and link	$\text{Regs}[R31] \leftarrow \text{PC}+4; \text{PC} \leftarrow \text{name};$ $\text{PC} + 4 - 2^{25} \leq \text{name} \leq \text{PC} + 4 + 2^{25}$
JR R2	Jump register	$\text{PC} \leftarrow \text{Regs}[R2];$
JALR R2	Jump and link register	$\text{Regs}[R31] \leftarrow \text{PC}+4; \text{PC} \leftarrow \text{Regs}[R2];$
BEQZ R2, name	Branch equal zero	if ($\text{Regs}[R4] == 0$) $\text{PC} \leftarrow \text{name};$ $\text{PC} + 4 - 2^{25} \leq \text{name} \leq \text{PC} + 4 + 2^{25}$
BNEZ R2, name	Branch not equal zero	if ($\text{Regs}[R4] \neq 0$) $\text{PC} \leftarrow \text{name};$ $\text{PC} + 4 - 2^{25} \leq \text{name} \leq \text{PC} + 4 + 2^{25}$

- Floating Point

Example (Exercise 2.6) – DLX code for code fragment

```
for (i=0; i<100; i++)  
    A[i] = B[i] + C;
```

assuming

- Variables **A**, **B**, **C** and **i** are in addresses 0, 2000, 1500 and 2000, respectively. (Therefore, you can use immediate mode to address them – less than 16 bits.)
- All data and addresses are kept in memory except when they are operated on.
- Values of registers are lost between iterations of the loop.

Instruction Count Breakdowns

- five programs from SPECint92 for DLX

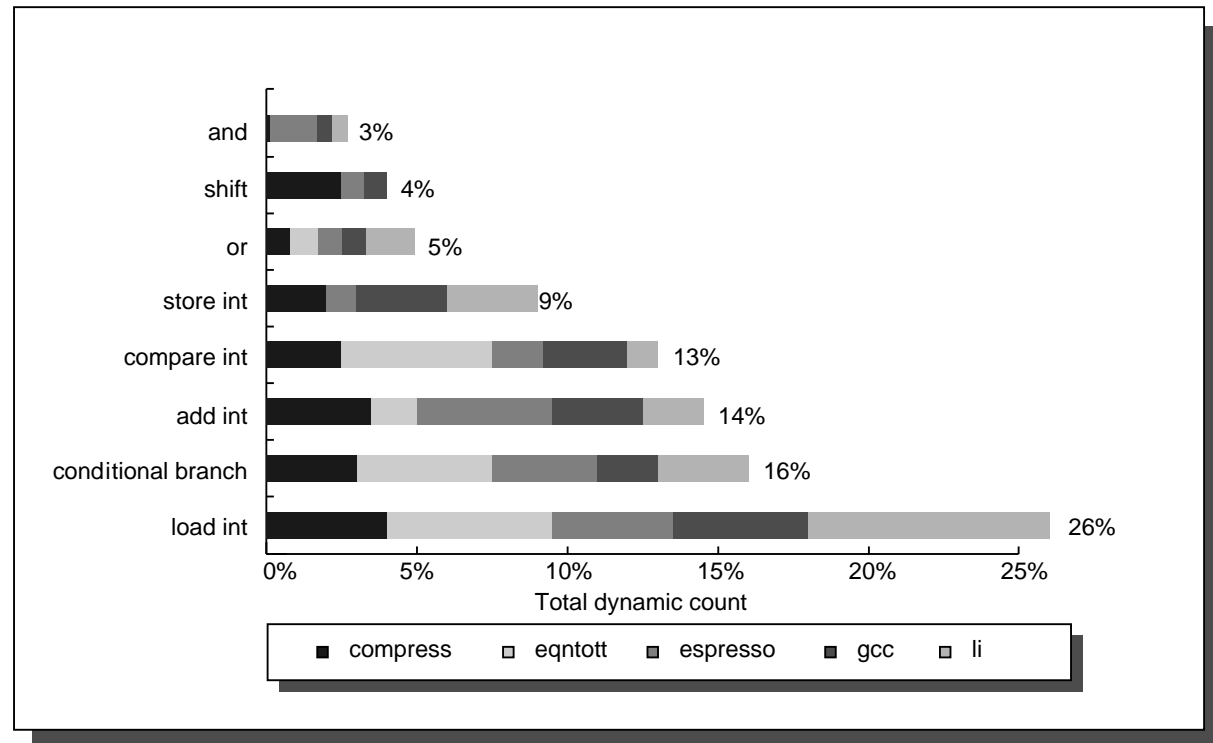


FIGURE 2.28 Graphical display of instructions executed of the five programs from SPECint92 in Figure 2.26.

- five programs from SPECfp92 for DLX

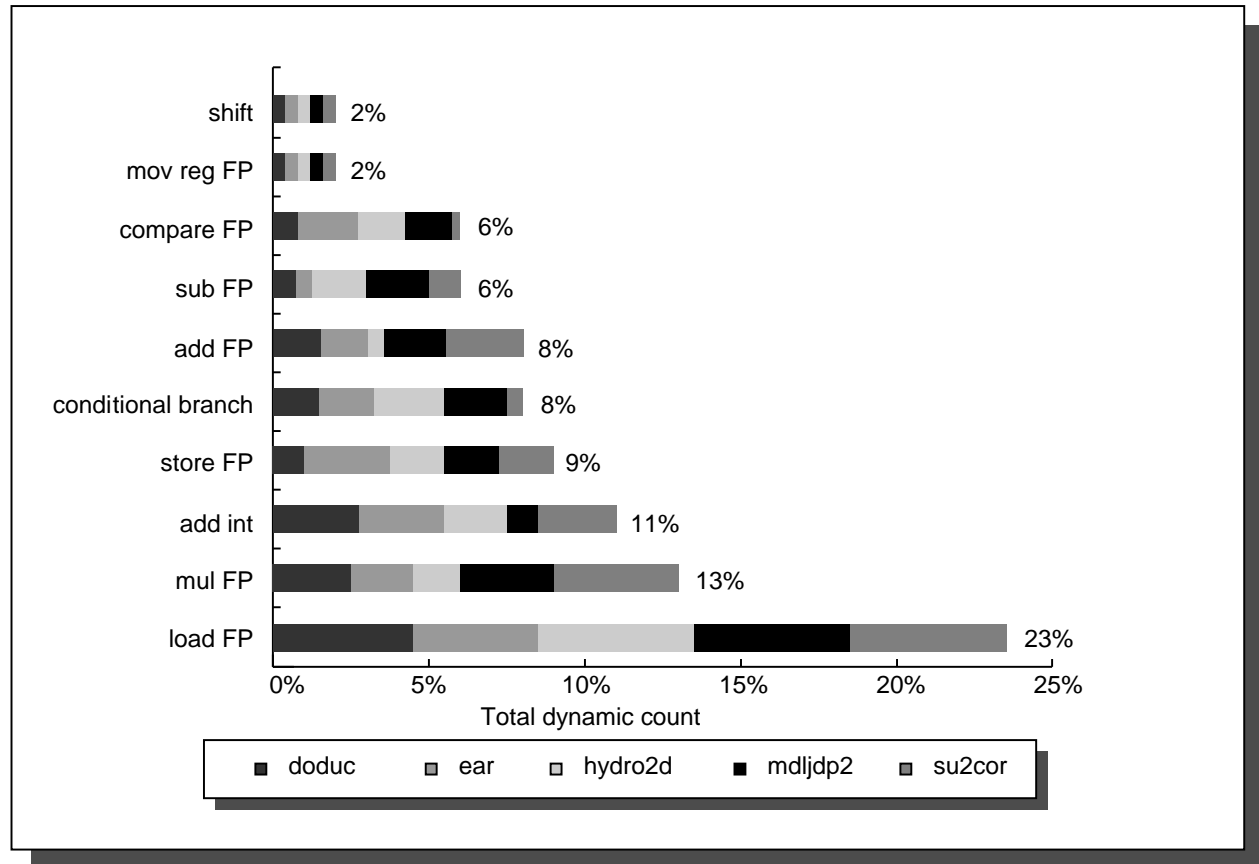


FIGURE 2.29 Graphical display of instructions executed of the five programs from SPECfp92 in Figure 2.27.

CPI ratio between MIPS and VAX

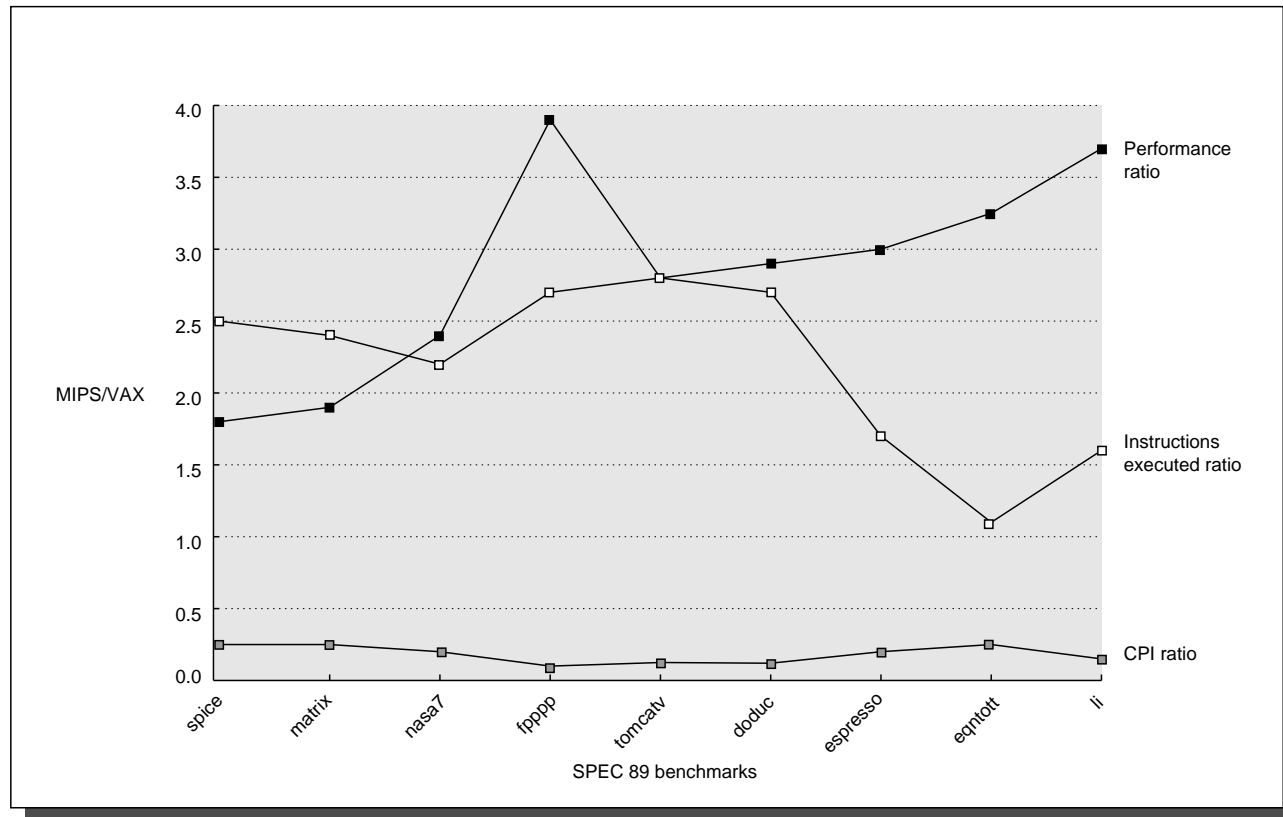


FIGURE 2.30 Ratio of MIPS M2000 to VAX 8700 in instructions executed and performance in clock cycles using SPEC89 programs.

Why does MIPS beat VAX?

- $IC_{MIPS} \approx 2 \times IC_{VAX}$
- $CPI_{MIPS} \approx CPI_{VAX}/6$
- MIPS is about 3 times as fast as VAX given the same clock cycle times.

The Role of Compiler

Why is compiler important to computer designers?

- Instruction set architecture is the target of compiler.
 - Instruction set architecture should allow compilers to generate efficient code.
- Compiler optimization affects the performance of the code generated.
 - Instruction set architecture should allow to simplify compilers. efficient code.
 - Computer designers should know the limitation of compiler optimization.

Structure of Optimizing Compilers

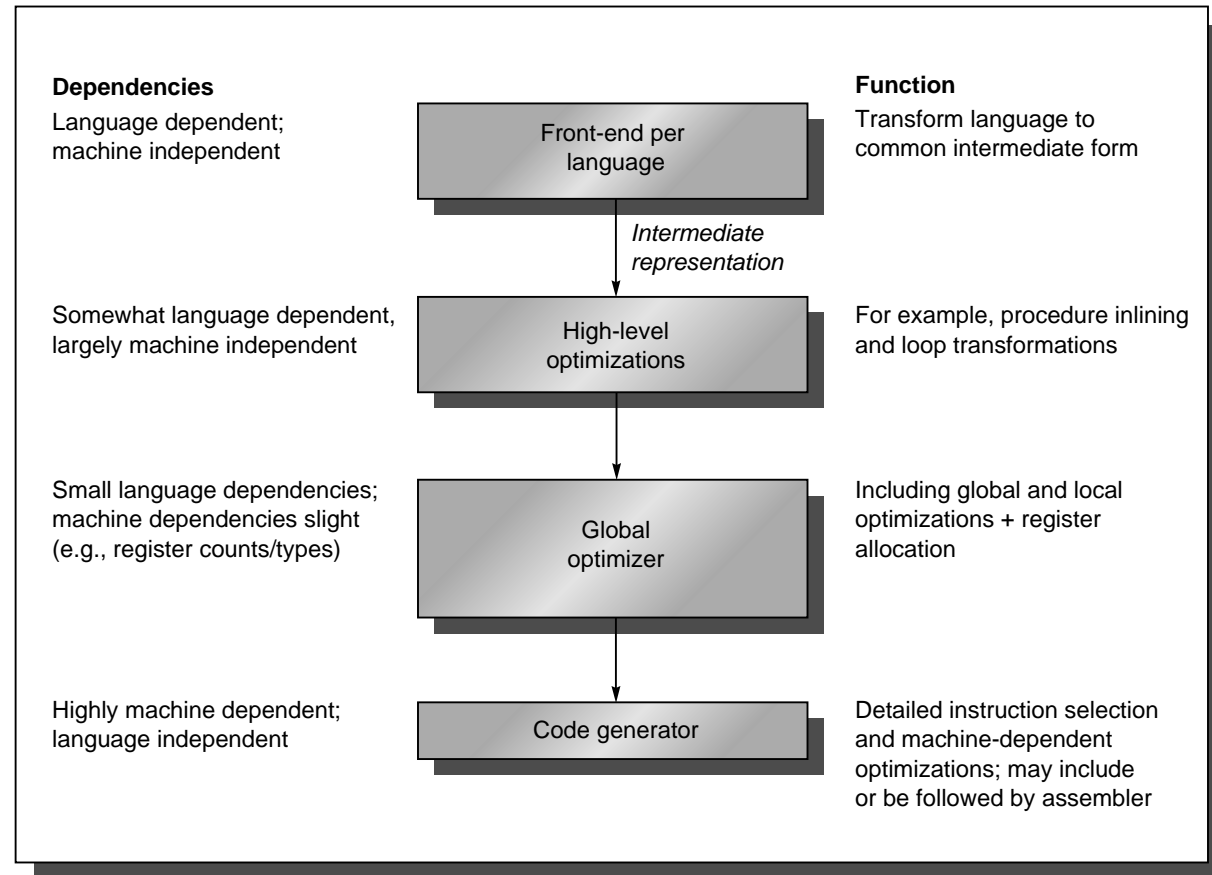


FIGURE 2.18 Current compilers typically consist of two to four passes, with more highly optimizing compilers having more passes.

Compiler Optimizations

- High-level
 - procedure integration: procedure in-lining
- Local
 - Common expression elimination (18%)
 - Constant propagation (22%)
 - Stack height reduction
- Global
 - Global common expression elimination (13%)
 - Copy propagation (11%)
 - Code motion (16%)
 - Induction variable elimination (2%)
- Machine-Dependent
 - Strength reduction
 - Pipeline scheduling
 - Branch offset optimization

Limitation of Compiler Optimization

- Phase-ordering problem
- Alias prevents allocating registers to variables

Impact of Compilers Technology on the Architect's Decisions

- Graph coloring for register allocation is more efficient with more than 16 registers.
- Alias limits the use of large number of registers

How can computer designers help compiler writers?

- Regularity: Orthogonality among addressing mode, data types and operations.
- Simplicity: Provide primitives not solutions.
- Simplify Trade-offs the compiler has to make.
- Provide instructions that bound the quantities known at compiler-time.