From CISC to RISC

CISC Creates the Anti-CISC Revolution

Digital Equipment Company (DEC) introduces VAX (1977)

Commercially successful 32-bit CISC minicomputer

In 1970s and 1980s

CISC minicomputers became cheaper

Serious computers became available to small organizations

UNIX developed as minicomputer operating system

TCP/IP developed to support networks of minicomputers

Computer Science emerged as separate academic discipline

Students needed topics for final projects, theses, dissertations

Research results on CISC performance

Most machine instructions are never used

CISC implementations give up speed in favor of generality

CISC machines run slowly to support unnecessary features

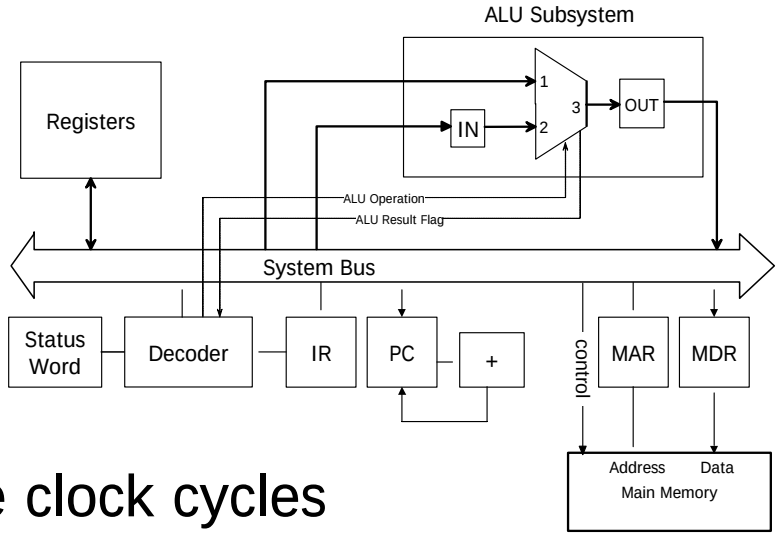
CISC Limitations

CISC instruction set requires microcode

- Many different instruction types
  - Each instruction requires different implementation
- Complex operations
  - Many instructions require complex decoding and sequencing

Central bus organization

- Atomic microcode operations
- System bus = bottleneck
  - Microcode operations — sequential
  - Machine instructions — sequential
- Machine instruction executes in multiple clock cycles



Memory access

- Operation complexity — non-uniform instruction length
- Instruction fetch — multiple clock cycles to load instruction

RISC "Philosophy"

Technological developments from 1975 to 1990

Price of RAM — from $5000 / MByte (1975) to $5 / MByte (1990)

Compilers — powerful and efficient with extensive optimization

Unix, C, and TCP/IP — practical portable code

Principal research result on CISC performance

~ 90% of run time = ~ 10% of VAX ISA

~ 90% of VAX instruction set < 10% of run time

Reduced Instruction Set Computer (RISC) — 1984

Apply Amdahl's "Law" to Instruction Set Architecture (ISA)

- Speed up operations accounting for most of run time
- Ignore performance degradation to other instructions

RISC ISA — keep most important instructions from CISC ISA

- Other CISC instructions implemented as multiple RISC instructions

Simple hardware implementation — faster execution

RISC Microprocessors

Simpler ISA

- Fewer machine instructions
- All instructions are same length

Simpler hardware design

- Allows lower CPI_i and higher clock speed
- No microcode — all instructions implemented in similar way
- No dedicated system bus
- CPU can process several instructions at once
- An instruction completes execution on almost every clock cycle

High level program compiled to RISC

- Larger IC_i — more machine instructions than compilation for CISC
- Run more quickly than same high level programs on CISC

All processors today use RISC technology

- Pure RISC (IBM Power, SPARC, MIPS, ARM, ...)
- RISC technology for CISC language Intel x86 (Pentium, Core, Xeon)
- Explicitly parallel RISC (Intel Itanium, IBM mainframes)

CISC vs. Pure RISC

	CISC	RISC
Instruction Types	300	50
Addressing Modes	15	5
Data Types	10	2
Procedure Handling	Automated	Coded
Implementations	Complex	Simple
Memory Organization	Complex	Simple

$$S = \frac{T}{T'} = \frac{CPI^{CISC} \times IC^{CISC} \times \tau^{CISC}}{CPI^{RISC} \times IC^{RISC} \times \tau^{RISC}} = \frac{CPI^{CISC}}{CPI^{RISC}} \times \frac{IC^{CISC}}{IC^{RISC}} \times \frac{\tau^{CISC}}{\tau^{RISC}}$$

$$= (\approx 6) \times \left(\approx \frac{1}{2} \right) \times \frac{\tau^{CISC}}{\tau^{RISC}}$$

$$\approx 3 \times \frac{\tau^{CISC}}{\tau^{RISC}}$$

Designing a RISC ISA

Considerations for a RISC ISA

Goals

- Simple — no instruction should require more steps than others
- Complete — able to perform any desired computation
- Orthogonal — only one way to encode any given computation

Choices

- Computation model
  - Register-register
  - Register-memory
- Range and type of operations
- Operands
  - Data types
  - Data sizes
- Addressing modes
  - Displacement sizes
- Branch types
  - Conditional
  - Unconditional
  - Procedural (call/return)
  - Branch offset (length of jump)

Instruction Types

Representative instruction distribution

Five programs from SPECint92 benchmark suite

Compile for x86 instruction set (ISA for Intel 386/486/Pentium)

Instruction	Relative Proportion of Total Run Time
Load	22%
Conditional branch	20%
Compare	16%
Store	12%
Add	8%
And	6%
Sub	5%
Move reg-reg	4%
Call	1%
Return	1%
Other	5%
Total	**100%**

First 10 instructions account for 95% of run time

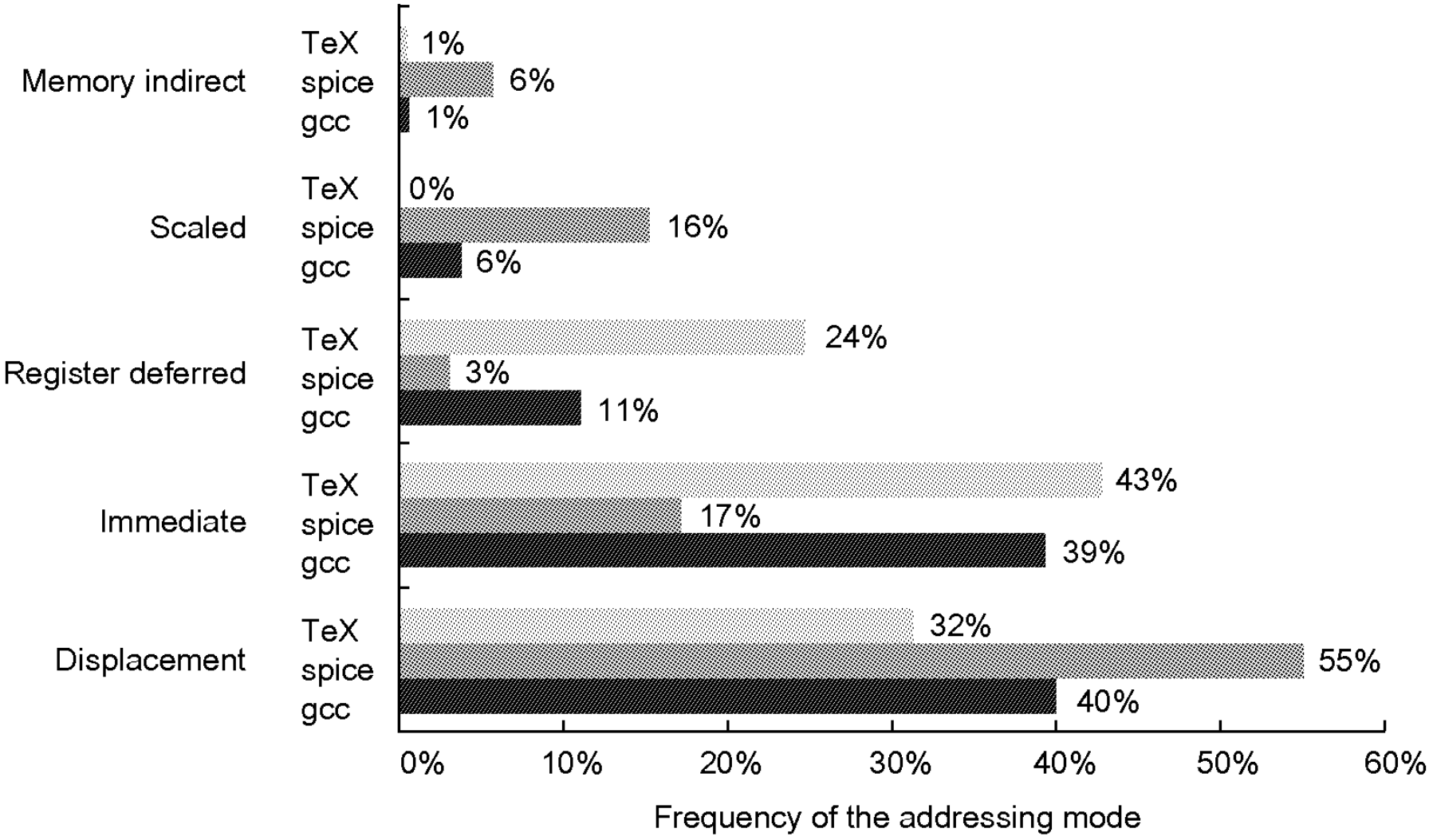
Amdahl's "Law"

Fast implementation of 95%
Other 5% will not seriously degrade performance

Must include unconditional branch for completeness

Ref: Hennessy / Patterson, figure 2.11

Addressing Modes Graph



Ref: Hennessy / Patterson, figure 2.6

Addressing Modes

Representative instruction distribution

Three programs from SPEC CINT92 and SPEC CFP95 benchmarks

Compile for VAX instruction set

Mode	tex	spice	gcc	Example of Mode
register deferred	24	3	11	`mem[R1]`
immediate	43	17	39	`#11223344`
displacement	32	55	40	`mem[R1 + disp]`
memory indirect	1	6	1	`mem[mem[R1]]`
scaled	0	16	6	`mem[R1 + R2 * d + disp]`
other	0	3	3	
total	**100**	**100**	**100**	
total (top 3)	**99**	**75**	**90**	

First three addressing modes

Account for more than 75% of all operand accesses

Ref: Hennessy / Patterson, figure 2.6

Instruction Length

Instructions should be of uniform length

Simplifies instruction **DECODING**

- No need to calculate instruction length
- Instruction fields are always in same place

Enables **INSTRUCTION FETCH** in 1 clock cycle

Practical instruction lengths

Most RISC machines for servers/workstations use 32-bit instructions

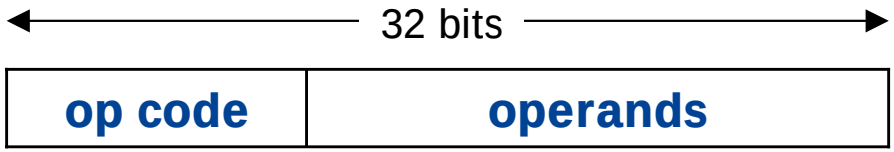
Special purpose RISC machines use longer instructions

Itanium and mainframes use 128-bit instructions

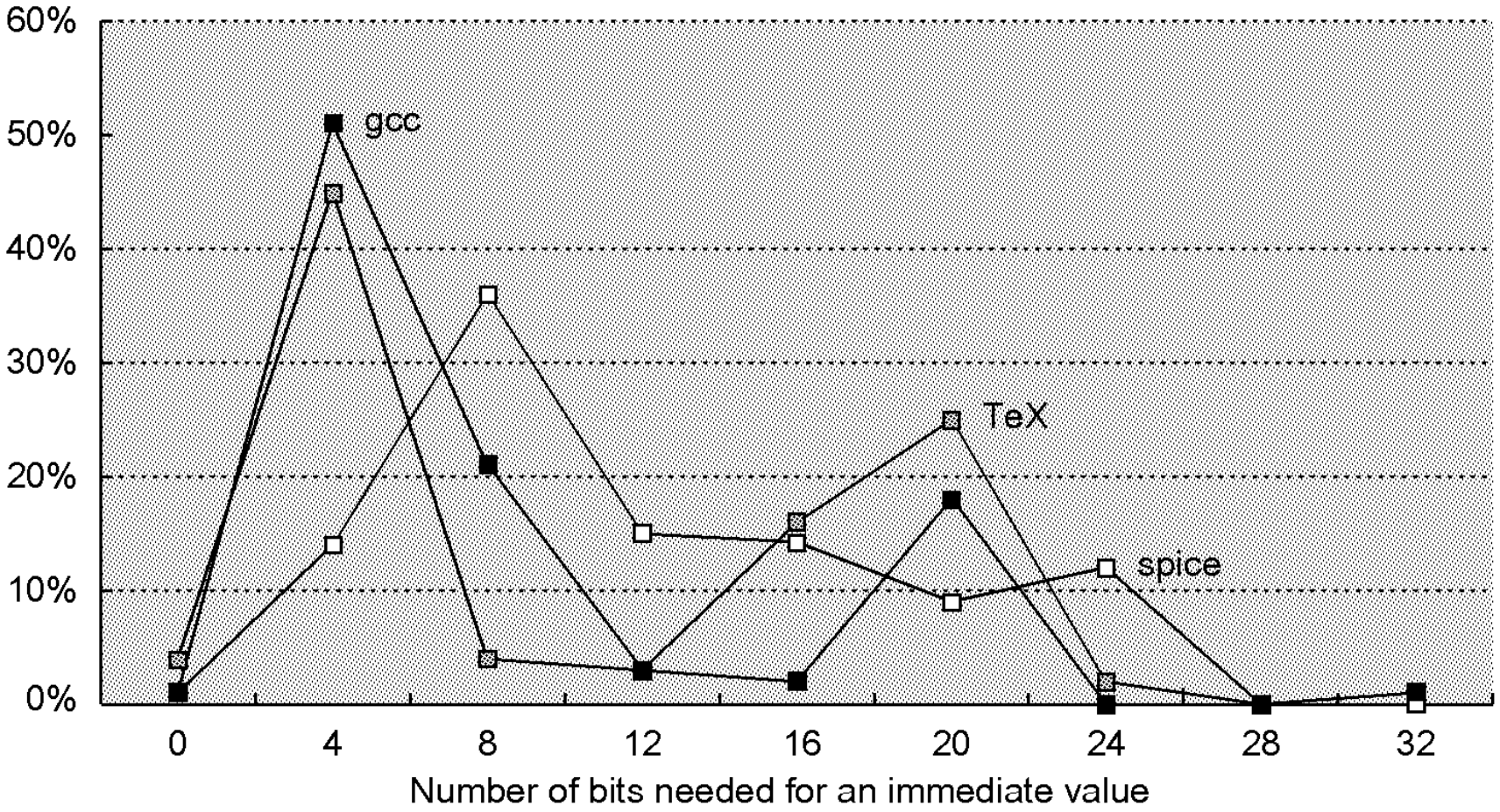
ISA defines 32-bit instructions

No single field can be 32 bits long

Includes address displacements, immediates, branch length



Length of Immediate Operand Graph



Ref: Hennessy / Patterson, figure 2.9

Length of Immediate Operand

Representative instruction distribution

Three programs from SPEC CINT92 and SPEC CFP95 benchmarks

Compile for VAX instruction set

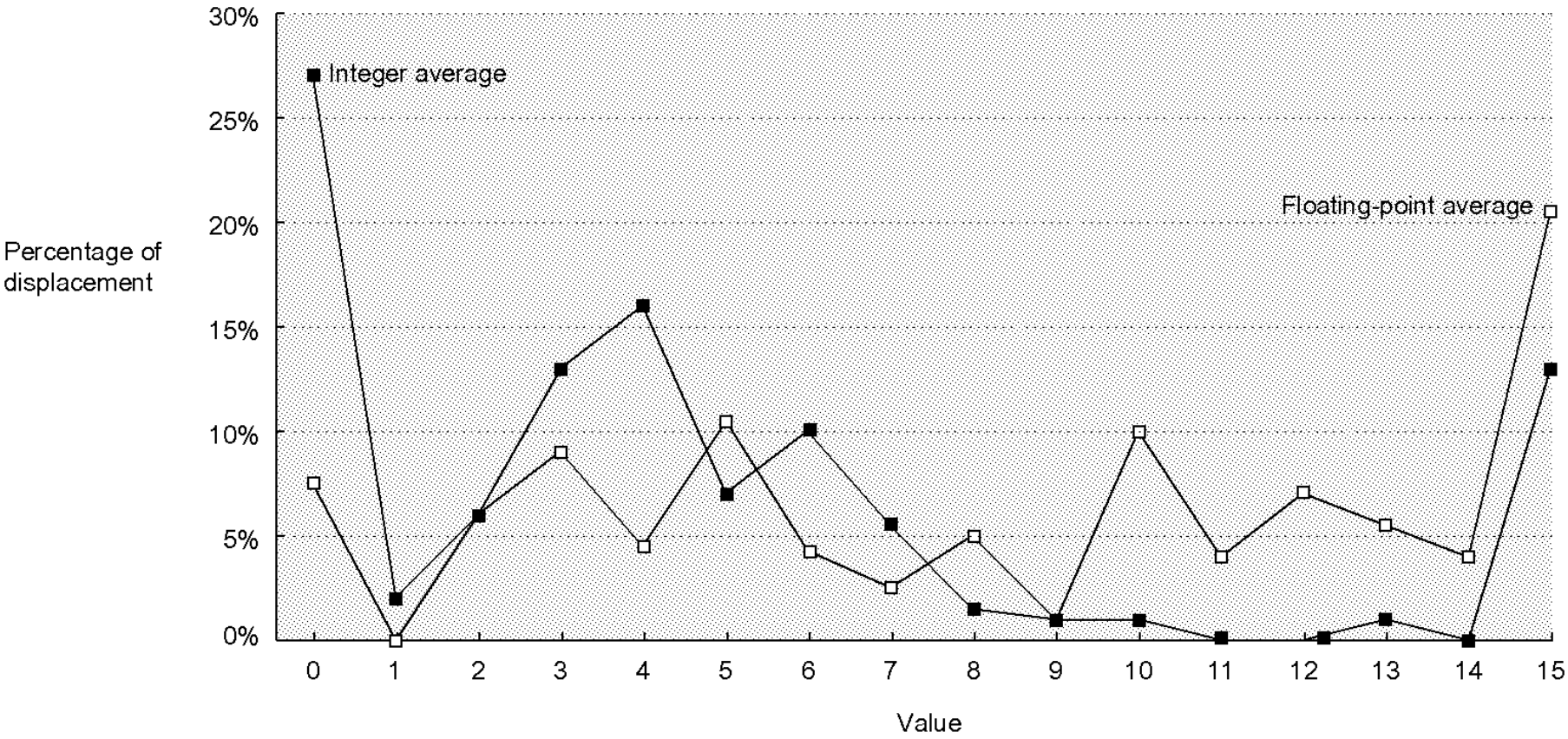
Immediate size	tex	spice	gcc
0	3	1	1
4	45	13	50
8	4	35	22
12	3	15	4
16	15	14	3
20	25	10	18
24	2	12	0
28	1	0	0
32	2	0	2
Total	**100**	**100**	**100**
Total to 16 bits	**70**	**78**	**80**

#1122

Allocating **16 bits** in 32-bit instruction for **immediate operands** covers more than 70% of cases

Ref: Hennessy / Patterson, figure 2.9

Displacement Length Graph



Ref: Hennessy / Patterson, figure 2.7

Displacement Length

Representative instruction distribution

Programs from SPEC CINT92 and SPEC CFP95 benchmarks

Compile for VAX instruction set

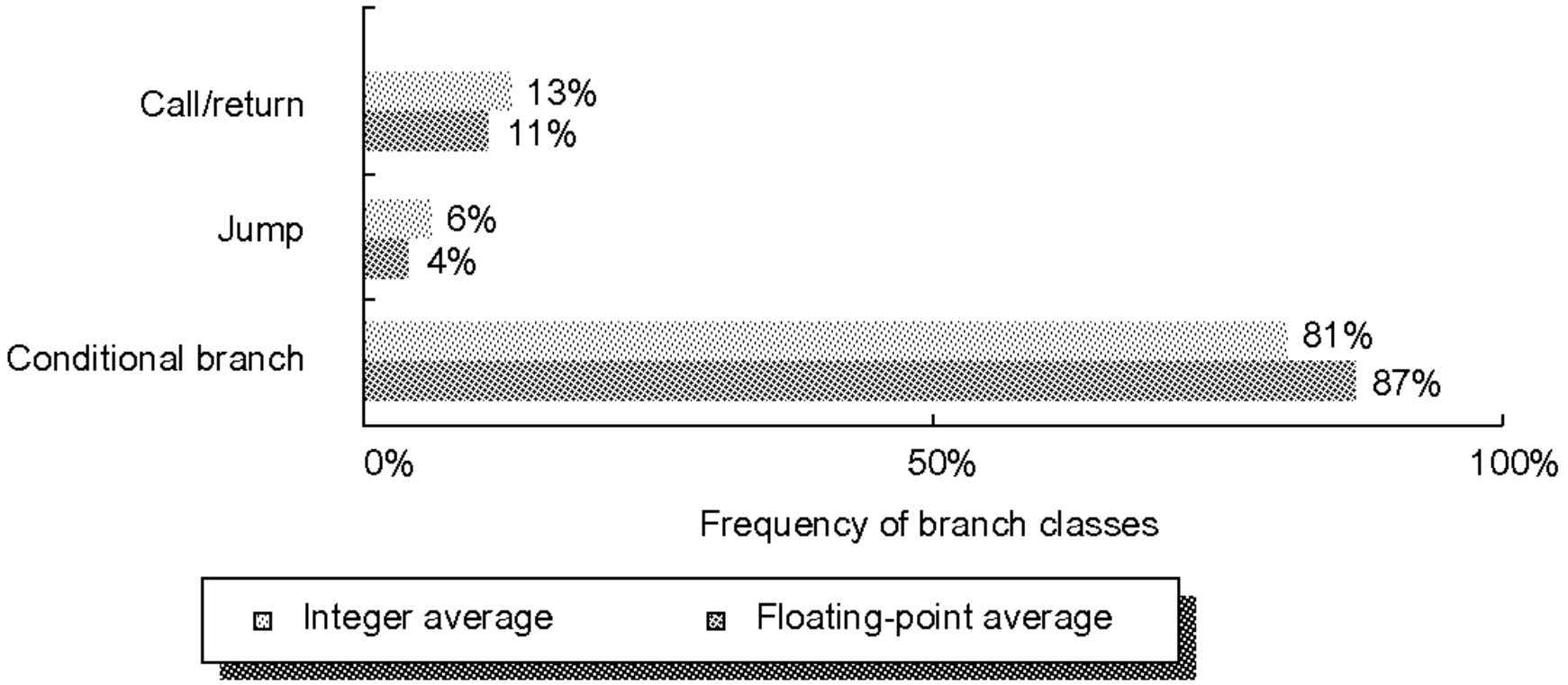
Bits in address displacement	int	FP
0	26	7
1	1	0
2	6	6
3	12	8
4	16	5
5	6	10
6	10	4
7	6	3
8	2	5
9	1	1
10	1	10
11	0	4
12	0	7
13	1	6
14	0	4
15	12	20
Total	**100**	**100**

Allocating **16 bits** for **address displacements** covers almost all cases

`mem[R1 + 1122]`

Ref: Hennessy / Patterson, figure 2.7

Branch Instructions Graph



Ref: Hennessy / Patterson, figure 2.12

Branch Instructions

Representative instruction distribution

Programs from SPEC CINT92 and SPEC CFP95 benchmarks

Compile for VAX instruction set

	Integer	FP
Call / Return	13	10
Unconditional Branch	6	4
Conditional branch	81	86
Total	100	100
Total of Conditional and Unconditional Branch	87	90

Conditional branch accounts for more than 80% of all branch instructions

Unconditional branch must be included for completeness

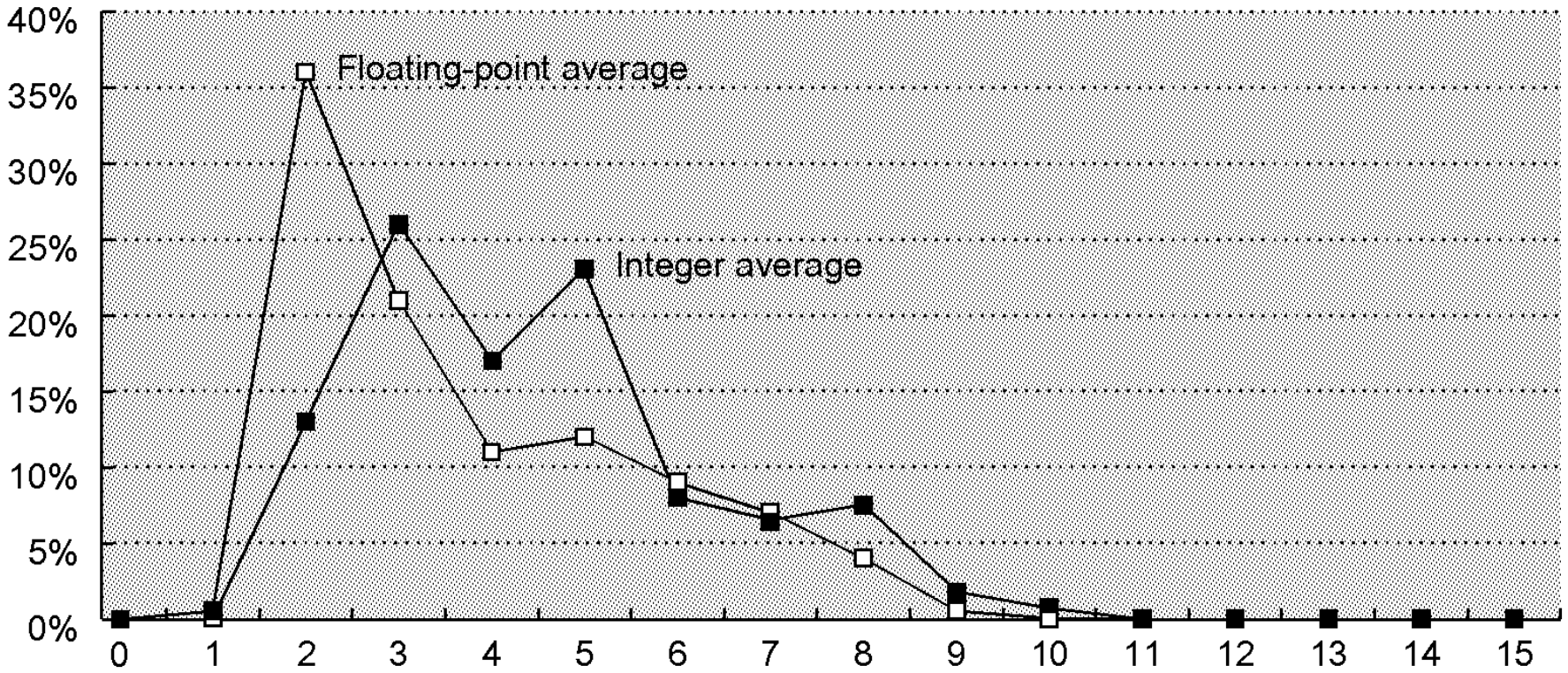
Call and return

Include many steps — saving registers and branching

Are difficult to implement

Ref: Hennessy / Patterson, figure 2.12

Branch Offset Graph



Ref: Hennessy / Patterson, figure 2.13

Branch Offset

Representative instruction distribution

Programs from SPEC CINT92 and SPEC CFP95 benchmarks

Compile for VAX instruction set

Offset bits for branch address	int	FP
0	0	0
1	1	0
2	13	36
3	26	21
4	16	11
5	24	12
6	6	9
7	5	6
8	6	4
9	2	1
10	1	0
11	0	0
12	0	0
13	0	0
14	0	0
15	0	0
Total	**100**	**100**

Allocating **16 bits** for **branch offsets** covers almost all cases

PC ← PC + 1122

Ref: Hennessy / Patterson, figure 2.13

Summary — RISC ISA By the Numbers

Instruction Types

10 instructions cover 95% of run time

Choose 30 – 50 most necessary / convenient instructions

Addressing Modes

Register

Immediate

Displacement

75% – 90% of run time addressing modes

Instruction Length

32-bit instructions

Branch Instructions

Conditional branch

Unconditional branch

75% – 90% of run time addressing modes

Length of immediate values

16-bit length for

Immediate operand	70% – 80% of run time immediates
Displacement	100% of run time address displacements
Branch offset	100% of run time branch offsets