Universidade Federal da Bahia
Instituto de Matemática
Departamento de Ciência da Computação

MATA48

Arquitetura de Computadores

Aula 2

Projeto do conjunto de instruções

Prof. Marcos E Barreto

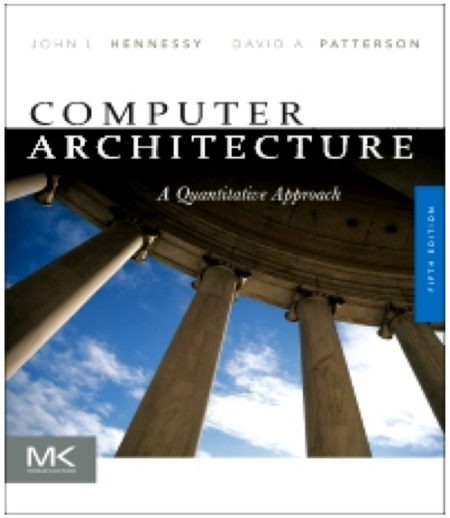
(2018.2)

Tópicos

- Contextualização
- Tipos de conjuntos de instruções
- Modos de endereçamento
- Tipos e tamanho de operandos

- Referência:

Hennessy, J.; Patterson. D.; Computer architecture – a quantitative approach. 5 ed., 2012. Apêndice A.



Contextualização

- Conjunto de instruções (*instruction set architecture*) = porção da arquitetura visível ao programador e ao compilador.

- Diferentes alternativas para o projeto do ISA, de acordo com as categorias de sistemas.
  - Sistemas desktop
    - Foco no desempenho de programas com dados inteiros e reais. Tamanho do código não importa.
  - Servidores
    - Foco no processamento de dados inteiros e strings, em aplicações Web, banco de dados, serviços de arquivos etc.
  - Dispositivos móveis
    - Foco em redução de custo e de consumo de energia. Programas pequenos consomem menos memória; instruções de ponto flutuante são opcionais para tornar o projeto mais barato.

Contextualização (2)

- Diferenças entre arquiteturas RISC e CISC
- **RISC** (*reduced instruction set architecture*)
  - Usada com sucesso nas três categorias.
  - Ex.: MIPS
- **CISC** (*complex instruction set architecture*)
  - Compatibilidade binária com software PC.
  - Grande quantidade de transistores => Lei de Moore.
  - Ex.: 80x86 => ISA externo (visível ao programador) é traduzido para um ISA RISC usado internamente pelo processador.

Contextualização (3)

- Tipos de operadores e instruções associadas

Operator type	Examples
Arithmetic and logical	Integer arithmetic and logical operations: add, subtract, and, or, multiply, divide
Data transfer	Loads-stores (move instructions on computers with memory addressing)
Control	Branch, jump, procedure call and return, traps
System	Operating system call, virtual memory management instructions
Floating point	Floating-point operations: add, multiply, divide, compare
Decimal	Decimal add, decimal multiply, decimal-to-character conversions
String	String move, string compare, string search
Graphics	Pixel and vertex operations, compression/decompression operations

Figure A.12 Categories of instruction operators and examples of each. All comput-

Contextualização (4)

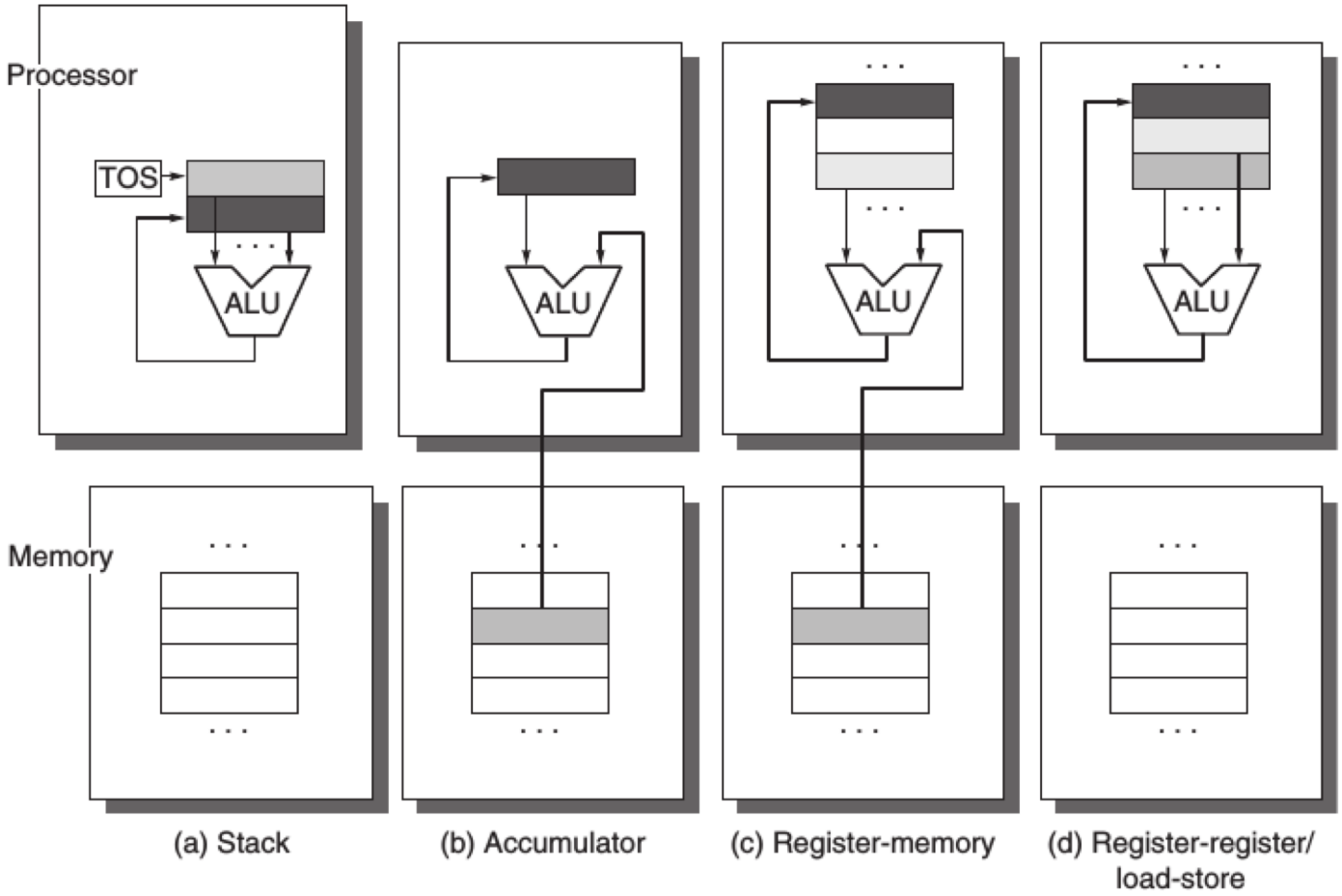
- Distribuição de frequência entre os diferentes tipos de instruções.

Rank	80x86 instruction	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 register-register	4%
9	call	1%
10	return	1%
Total		**96%**

Figure A.13 The top 10 instructions for the 80x86. Simple instructions dominate this

Tipos de conjunto de instruções

- Dependem do armazenamento interno ao processador.
- Alternativas:
  - Pilha (*stack*)
  - Acumulador
  - Registradores de propósito geral (GPR):
    - Registrador-memória
    - Load-store



C = A + B

PUSH A	LOAD A	LOAD R1, A	LOAD R1, A
PUSH B	ADD B	ADD R3, R1, B	LOAD R2, B
ADD	STORE C	STORE R3, C	ADD R3, R1, R2
POP C			STORE R3, C

Tipos de conjunto de instruções (3)

- GPR x registradores de uso específico
- Quantos registradores de uso geral (GPR) devem existir na arquitetura?
  - Dependem da eficiência do compilador

Tipos de conjunto de instruções (4)

- Natureza dos operandos em operações típicas da ULA
- 1º) Quando uma instrução da ULA tem 2 ou 3 operandos?
- 2º) Quantos operandos podem ser endereços de memória?

0	3	Load - Store	Alpha, ARM, MIPS, PowerPC, SPARC, SuperH, TM32
1	2	Registrador - memória	IBM 360/370, Intel 80x86, Motorola 68000, TI TMS320C54
2	2	Memória - memória	VAX
3	3	Memória - memória	VAX

Exemplo

Comentário
$Y \leftarrow A - B$
$T \leftarrow D \times E$
$T \leftarrow T + C$
$Y \leftarrow Y \div T$

Instruções com três endereços

$$Y = \frac{A - B}{C + (D * E)}$$

Comentário
$Y \leftarrow A$
$Y \leftarrow Y - B$
$T \leftarrow D$
$T \leftarrow T \times E$
$T \leftarrow T + C$
$Y \leftarrow Y \div T$

) Instruções de dois endereços

ACUMULADOR

Instrução	Comentário
LOAD D	$AC \leftarrow D$
MPY E	$AC \leftarrow AC \times E$
ADD C	$AC \leftarrow AC + C$
STOR Y	$Y \leftarrow AC$
LOAD A	$AC \leftarrow A$
SUB B	$AC \leftarrow AC - B$
DIV Y	$AC \leftarrow AC \div T$
STOR Y	$Y \leftarrow AC$

(c) Instruções de um endereço

Endereçamento de memória

- Como os endereços de memória são especificados e interpretados?
- Diferentes tipos de acesso: byte (8 bits), half words (16 bits), words (32 bits), double words (64 bits).
- Dois tipos de ordenação de bytes

Little Endian

7	6	5	4	3	2	1	0

Byte “x.....x000” na posição do bit menos significante da double word

Big Endian

0	1	2	3	4	5	6	7

Byte “x.....x000” na posição do bit mais significante da double word

Endereçamento de memória (2)

- Acesso a objetos com mais de um byte devem ser alinhados.
- Ex.: acesso ao objeto de tamanho *s* no endereço *A* é alinhado se **A mod s = 0**.

Alinhamento de memória

	Value of 3 low-order bits of byte address							
Width of object	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**
1 byte (byte)	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned	Aligned
2 bytes (half word)	Aligned		Aligned		Aligned		Aligned	
2 bytes (half word)		Misaligned		Misaligned		Misaligned		Misaligned
4 bytes (word)	Aligned				Aligned			
4 bytes (word)		Misaligned				Misaligned		
4 bytes (word)			Misaligned				Misaligned	
4 bytes (word)				Misaligned				Misaligned
8 bytes (double word)	Aligned							
8 bytes (double word)		Misaligned						
8 bytes (double word)			Misaligned					
8 bytes (double word)				Misaligned				
8 bytes (double word)					Misaligned			
8 bytes (double word)						Misaligned		
8 bytes (double word)							Misaligned	
8 bytes (double word)								Misaligned

Figure A.5 Aligned and misaligned addresses of byte, half-word, word, and double-word objects for byte-addressed computers. For each misaligned example some objects require two memory accesses to complete. Every aligned object can always complete in one memory access, as long as the memory is as wide as the object. The figure shows the memory organized as 8 bytes wide. The byte offsets that label the columns specify the low-order 3 bits of the address.

Modos de endereçamento

- Como a arquitetura especifica o endereço de um objeto a ser acessado?
- Os modos de endereçamento usam constantes, registradores e endereços de memória.

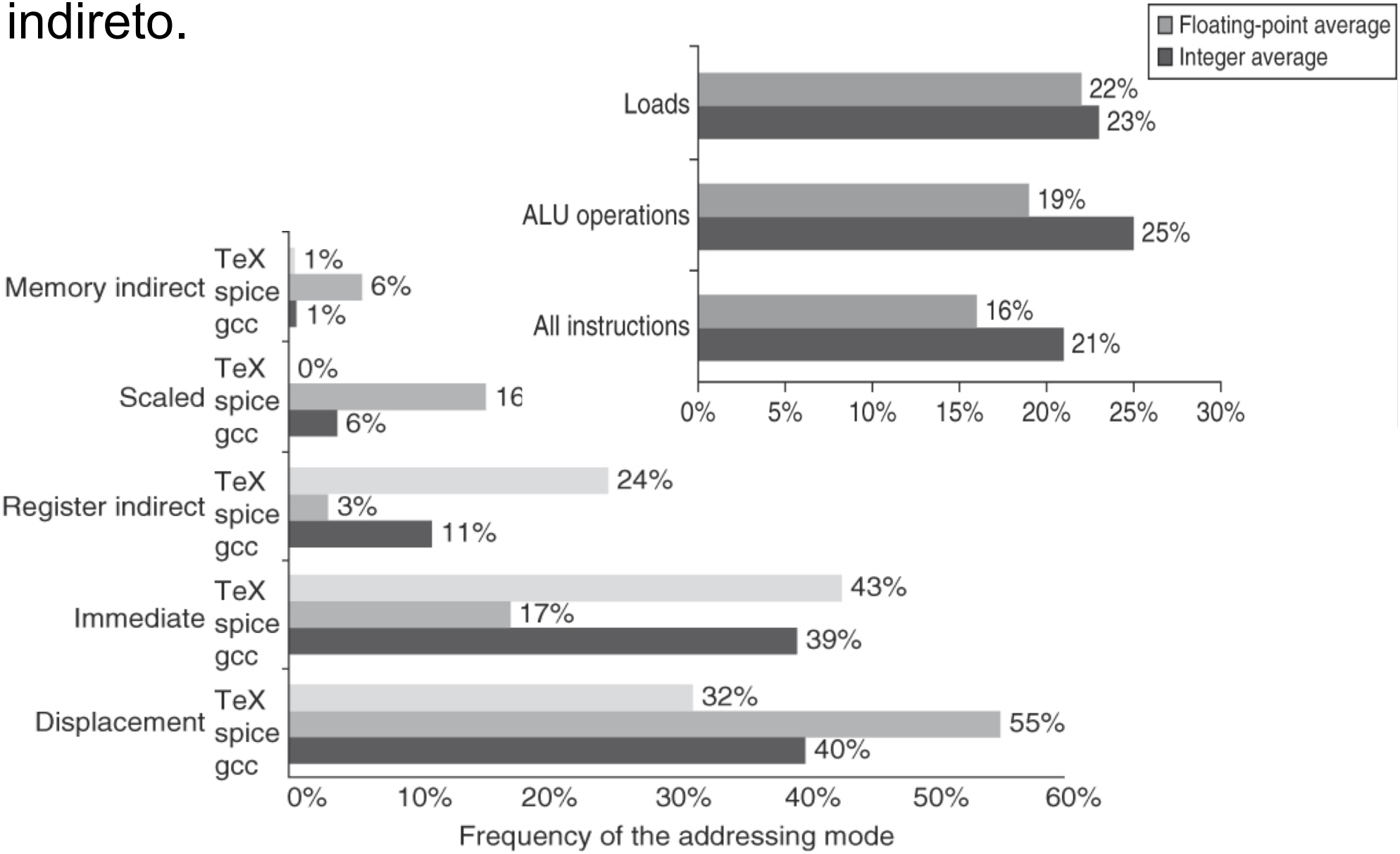
Modos de endereçamento

Addressing mode	Example instruction	Meaning	When used
Register	Add R4,R3	Regs[R4] ← Regs[R4] + Regs[R3]	When a value is in a register.
Immediate	Add R4,#3	Regs[R4] ← Regs[R4] + 3	For constants.
Displacement	Add R4,100(R1)	Regs[R4] ← Regs[R4] + Mem[100 + Regs[R1]]	Accessing local variables (+ simulates register indirect, direct addressing modes).
Register indirect	Add R4,(R1)	Regs[R4] ← Regs[R4] + Mem[Regs[R1]]	Accessing using a pointer or a computed address.
Indexed	Add R3,(R1 + R2)	Regs[R3] ← Regs[R3] + Mem[Regs[R1] + Regs[R2]]	Sometimes useful in array addressing: R1 = base of array; R2 = index amount.
Direct or absolute	Add R1,(1001)	Regs[R1] ← Regs[R1] + Mem[1001]	Sometimes useful for accessing static data; address constant may need to be large.
Memory indirect	Add R1,@(R3)	Regs[R1] ← Regs[R1] + Mem[Mem[Regs[R3]]]	If R3 is the address of a pointer *p*, then mode yields **p*.
Autoincrement	Add R1,(R2)+	Regs[R1] ← Regs[R1] + Mem[Regs[R2]] Regs[R2] ← Regs[R2] + *d*	Useful for stepping through arrays within a loop. R2 points to start of array; each reference increments R2 by size of an element, *d*.
Autodecrement	Add R1, −(R2)	Regs[R2] ← Regs[R2] − *d* Regs[R1] ← Regs[R1] + Mem[Regs[R2]]	Same use as autoincrement. Autodecrement/-increment can also act as push/pop to implement a stack.
Scaled	Add R1,100(R2)[R3]	Regs[R1] ← Regs[R1] + Mem[100 + Regs[R2] + Regs[R3] * *d*]	Used to index arrays. May be applied to any indexed addressing mode in some computers.

Figure A.6 Selection of addressing modes with examples, meaning, and usage. In autoincrement/-decrement

Modos de endereçamento (3)

- Mais tradicionais: imediato, deslocamento e registrador indireto.



Tipos e tamanho dos operandos

- Como especificar o tipo do operando?
  - Codificado junto ao código da operação (*opcode*).
  - Uso de um campo (*tag*) interpretado pelo hardware.
- Categorias:
  - Desktop e servidores
    - Tipo do operando define seu tamanho efetivo.
    - Caractere (8 bits), half words (16 bits), words (32 bits), real de precisão simples (32 bits), real de precisão dupla (64 bits).
    - Representação: inteiros (complemento de 2), caracteres (ASCII ou Unicode 16 bits – Java), reais (IEEE 754).

Exemplo de acessos aos operandos

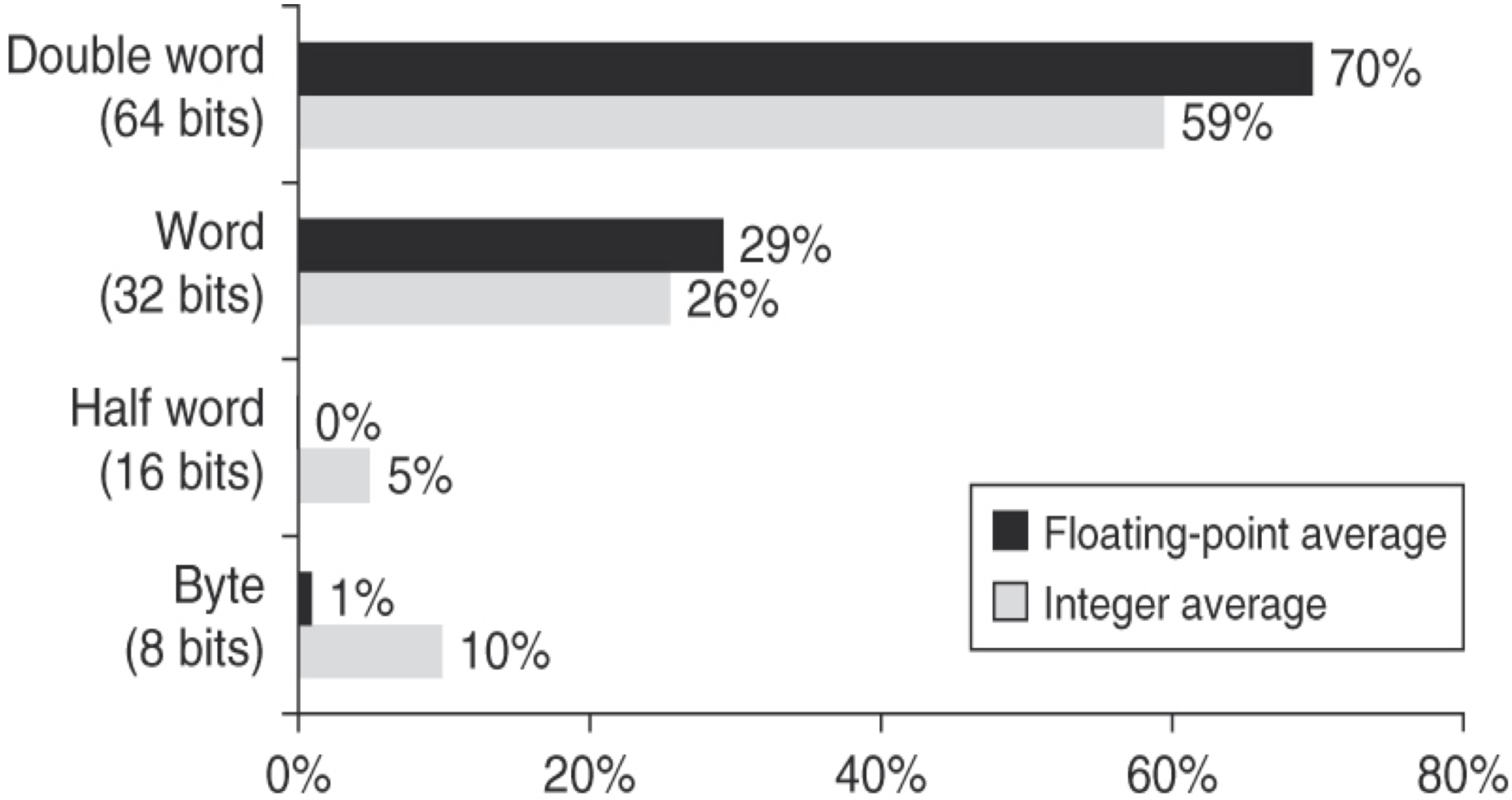


Figure A.11 Distribution of data accesses by size for the SPEC benchmark programs. The double-word data type is used for double-precision floating point in floating-point programs and for addresses, since the computer uses 64-bit addresses. On a 32-bit address computer the 64-bit addresses would be replaced by 32-bit addresses, and so almost all double-word accesses in integer programs would become single-word accesses.