Estudo e projeto do

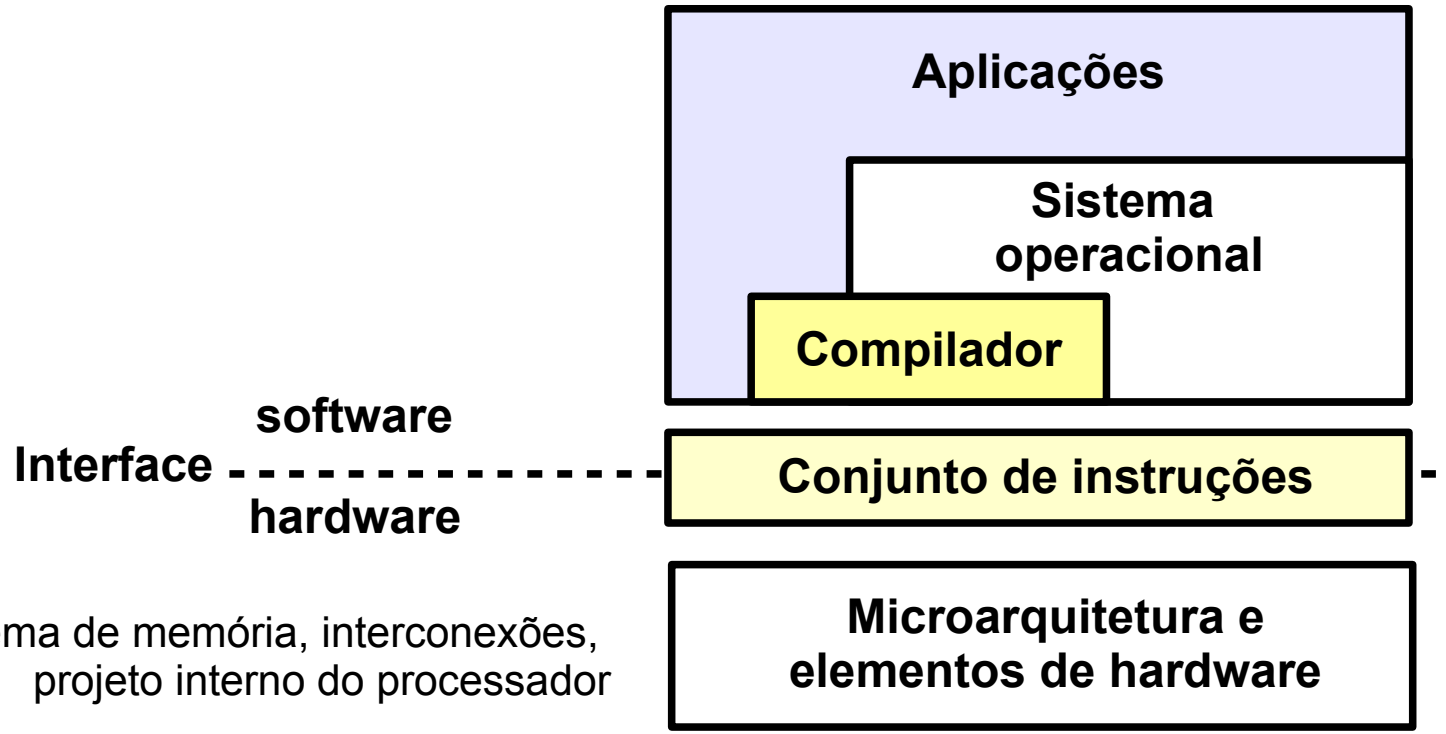
conjunto de instruções

(ISA - *Instruction Set Architecture*)

Marcos Ennes Barreto

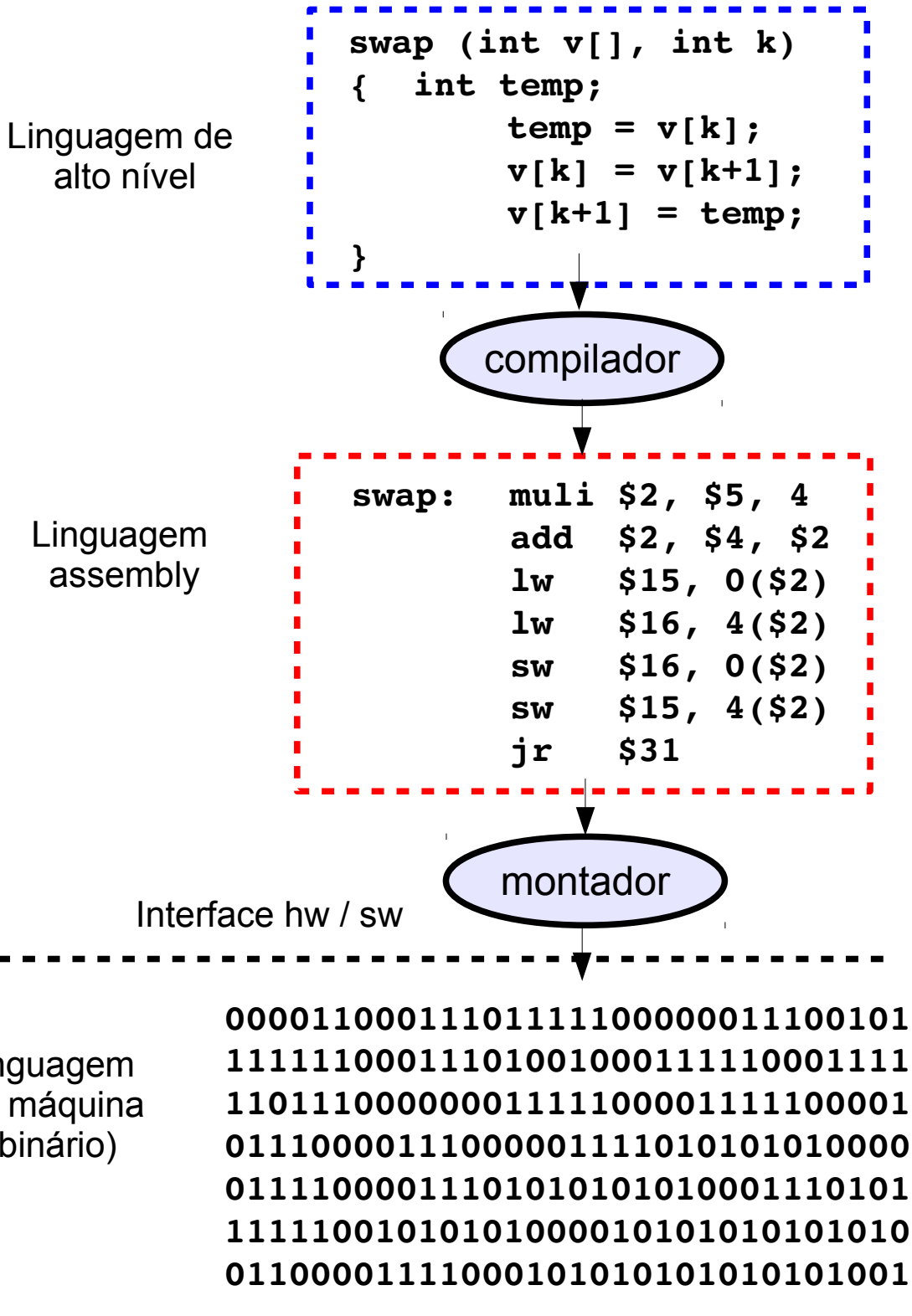
O que é o conjunto de instruções (ISA)?

- Parte da arquitetura visível ao programador e ao desenvolvedor de compiladores.
- Operações que (todos) os computadores oferecem e suportam de forma nativa.



Por que estudá-lo?

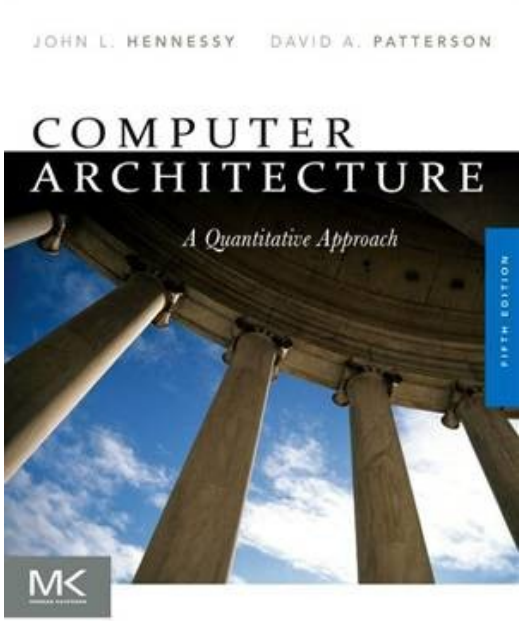
- Linguagens de alto nível escondem do programador os detalhes do hardware.
- Necessidade de entender a arquitetura do computador (estrutura e funcionamento do conjunto de instruções) para escrever programas mais eficientes ou para tirar proveito de especificidades do hardware não exploradas pelas linguagens de programação.



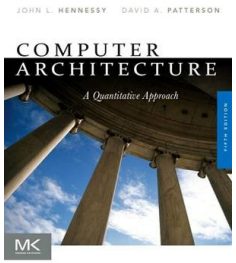
Referências iniciais

- HENNESSY, J. L.; PATTERSON, D. A. *Organização e projeto de computadores* – a interface hardware / software.
  4 ed. Rio de Janeiro: Campus, 2014.
  - *Capítulo 2*

- HENNESSY, J. L.; PATTERSON, D. A. Computer architecture – a quantitative approach.
  5 ed. Waltham: Morgan Kaufmann, 2012.
  - *Capítulo 1*
  - *Apêndices A e K*



Primeiros passos



Princípios gerais do projeto de ISAs
Apêndice A

- Contextualização
- Composição do conjunto de instruções
- Estudo de caso preliminar

Contextualização (1)

➔ Requisitos das categorias de arquiteturas

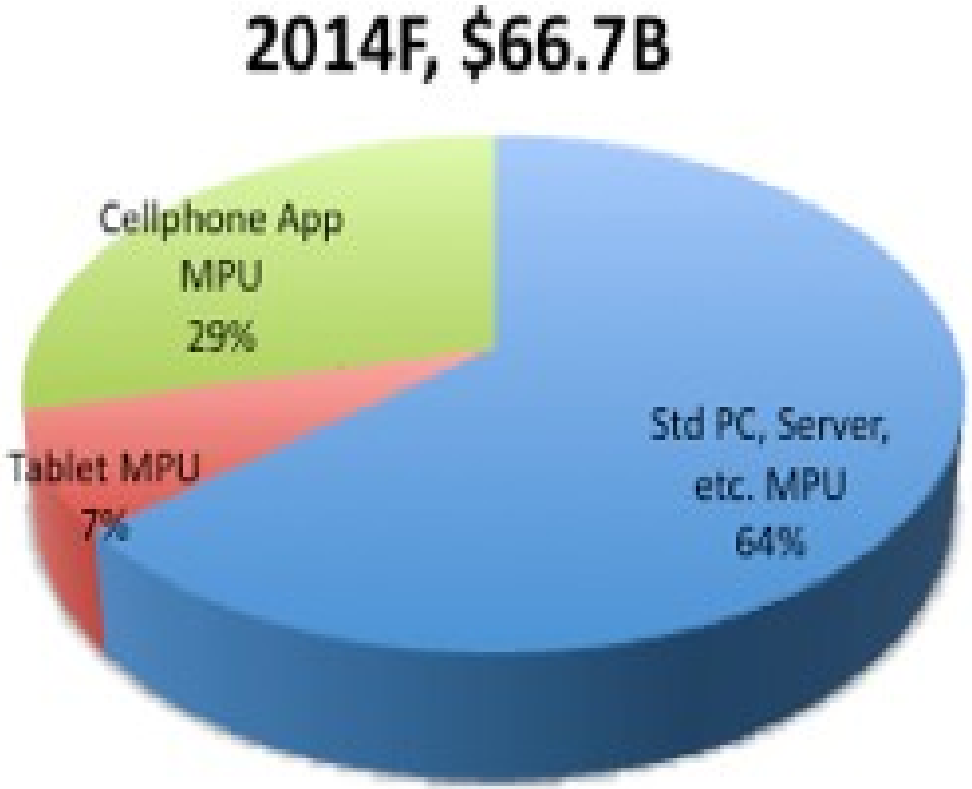
- Desktops
  - Foco no desempenho de programas (operações com inteiros e reais) e na manipulação de gráficos.
  - Tamanho do programa não importa.

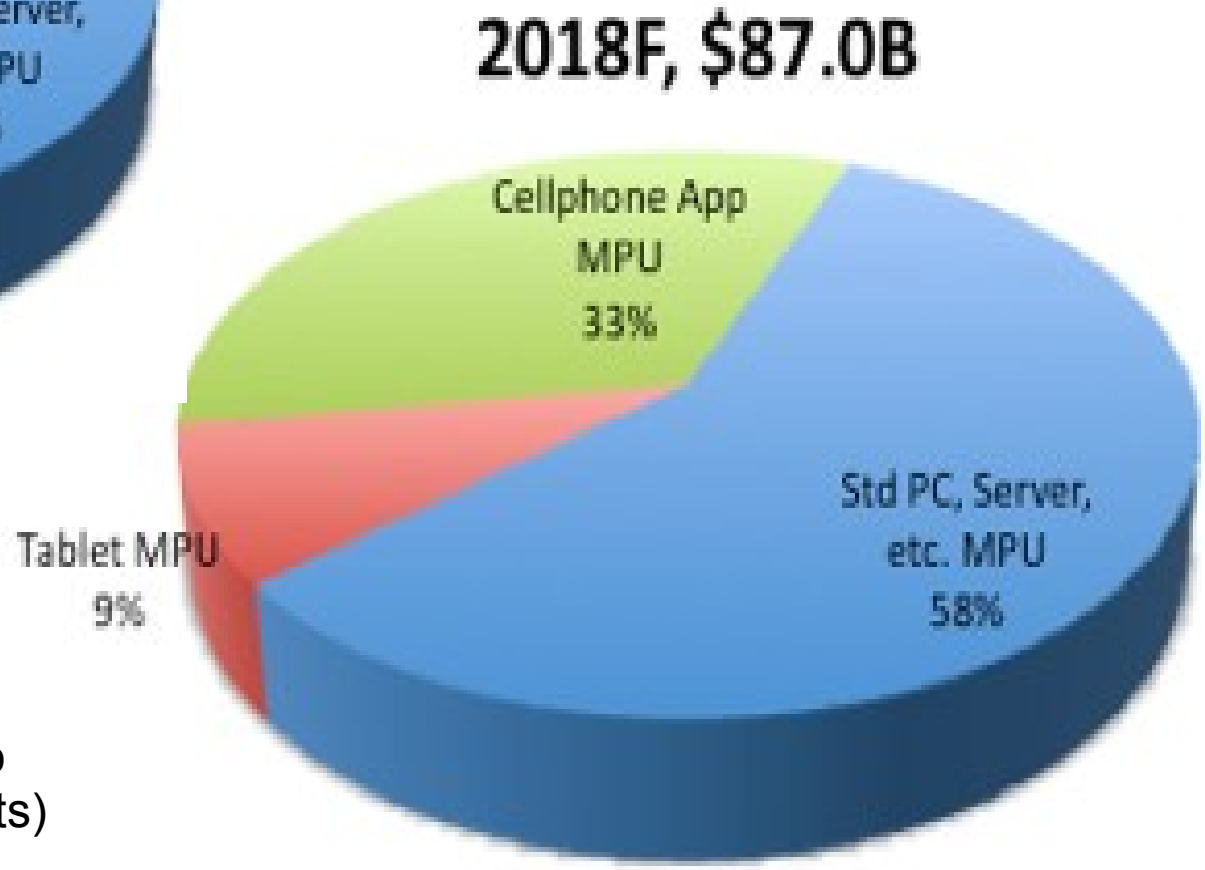
- Servidores e agregados (*clusters*)
  - Foco em disponibilidade, vazão (*throughput*), consumo de
  - energia e desempenho de operações com inteiros e strings.
  - Operações com reais podem não ser prioridade.

- Sistemas embarcados e dispositivos móveis
  - Prioridade para custo, energia e desempenho.
  - Tamanho do programa importa
    => menos memória = menor custo e menor consumo.
  - Operações com reais podem ser opcionais para reduzir o custo.

Contextualização (2)

- **Distribuição de mercado (*market share*)**





Venda de microprocessadores por tipo
Fonte: The McClean Report (IC Insights)

Contextualização (3)

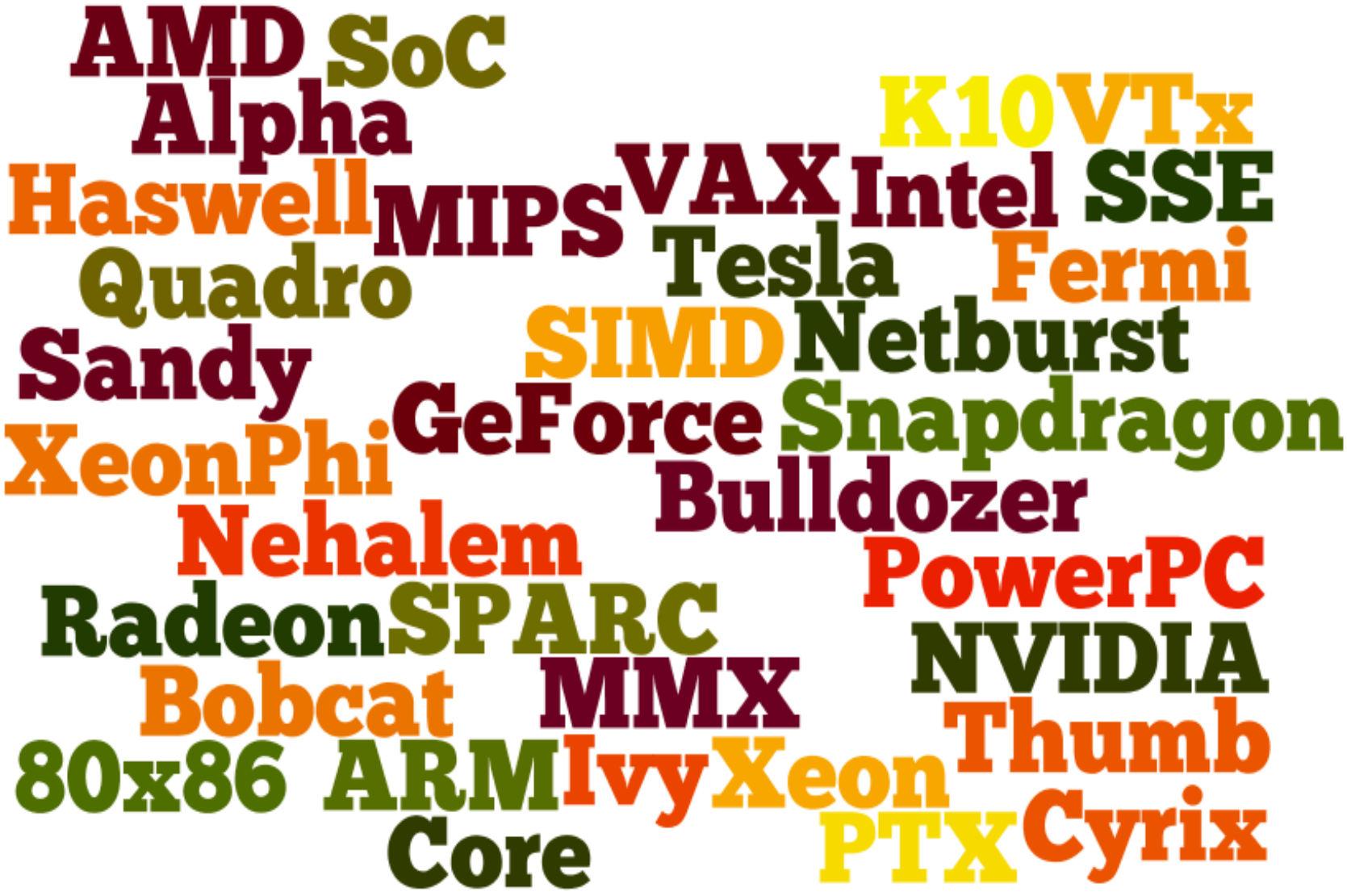
➔ Evolução dos conjuntos de instruções (ISAs)

- 80x86 (CISC):
  - ISA dominante no escopo de *desktops* e servidores *low-end*.
  - ISA com o maior número de extensões desde a sua criação.
  - Exemplos – Intel Core i3, i5, i7, Atom, Sandy e Ivy Bridge, Haswell.
  - Exemplos – AMD K5, K6, Athlon, Opteron, Phenom, FX.

- RISC:
  - ISA com diversas implementações em servidores *high-end* e em sistemas embarcados.
  - Exemplos: ARM, Thumb, MIPS, PowerPC, DEC Alpha.

Contextualização (4)

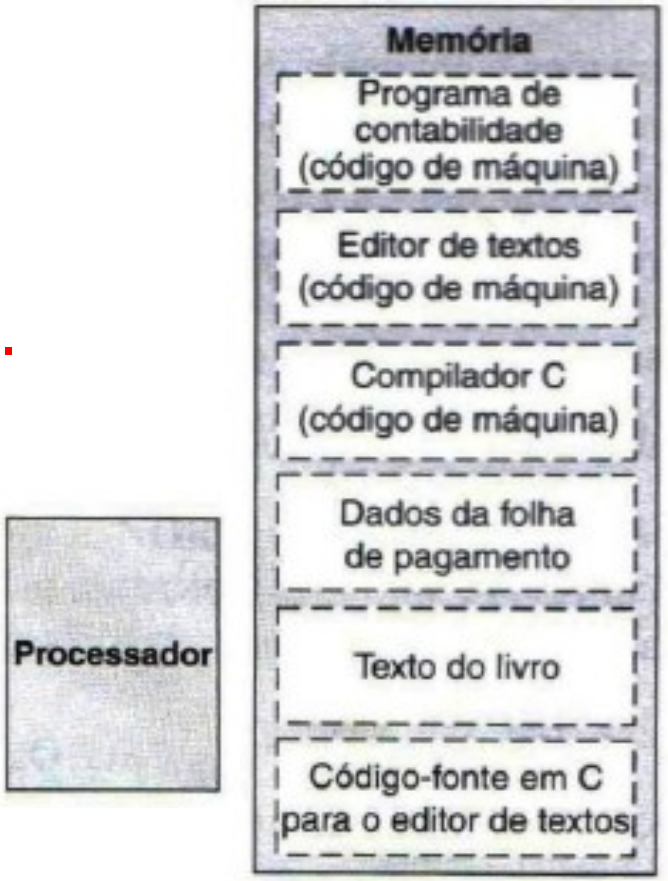
- **Ampla terminologia**



Contextualização (5)

→ Características gerais dos conjuntos de instruções

- O conjunto de instruções é semelhante para todas as arquiteturas.
  - Lembrar: operações que **(todos)** os computadores devem oferecer.
  - Hardware com os mesmos princípios e características.
    - Influência do modelo de von Neumann e do conceito de programa armazenado.
  - Projetistas têm objetivo comum:
    - *"Encontrar uma ISA que facilite o projeto do hardware ao mesmo tempo que maximiza o desempenho e minimiza o custo."*



Conceito de programa armazenado.

Contextualização (6)

➔ Diferenças entre os conjuntos de instruções

- Evolução tecnológica e questões de mercado.

- "**Filosofia**" de projeto adotada em cada fabricante:

  => **CISC (Complex Instruction Set Computer)**
  - Maior compatibilidade com diversos tipos de aplicações.
  - Maior facilidade para o desenvolvimento de compiladores e sistemas operacionais.

  => **RISC (Reduced Instruction Set Computer)**
  - Arquitetura mais simples para favorecer o desempenho.

Arquiteturas híbridas CISC / RISC

Processadores têm **conjuntos de instruções "externos"** que seguem a filosofia CISC e que são traduzidos **internamente** para um formato RISC.

Composição de um conjunto de instruções (1)

➔ Aspectos importantes para o compilador

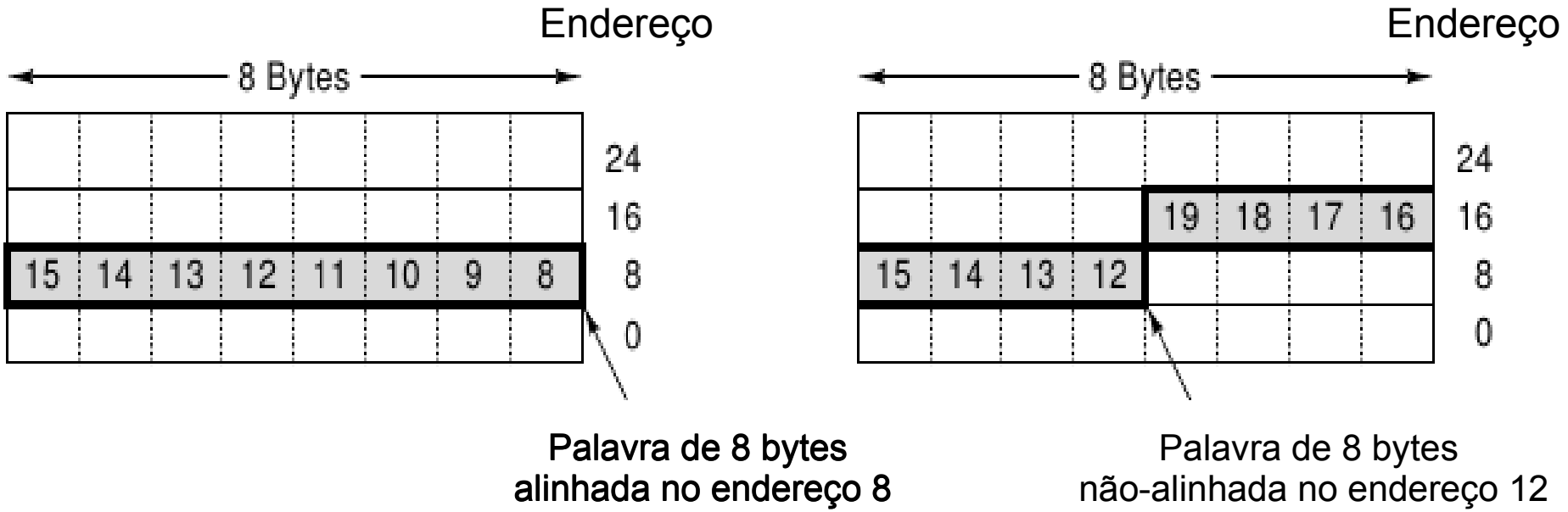
- Modelo de memória
- Registradores
- Tipos de dados
- Modos de execução
  - Modo kernel: todas as instruções são permitidas; usado para executar o sistema operacional.
  - Modo usuário: algumas instruções são proibidas; usado para executar aplicações dos usuários.

Composição de um conjunto de instruções (2)

➔ Aspectos importantes para o compilador

✗ Modelo de memória

- ✗ Memória dividida em células com endereços consecutivos.
- ✗ Células de memória com 8 bits (byte).
- ✗ Bytes agrupados em palavras (*words*) de 4 bytes (32 bits) ou 8 bytes (64 bits).
- ✗ Palavras devem ser alinhadas de acordo com os endereços da memória para aumentar o desempenho de acesso.



Composição de um conjunto de instruções (3)

→ Aspectos importantes para o compilador

- **Registradores**
  - Nem todos os registradores são visíveis no nível da ISA.
  - Registradores de propósito específico: contador de programa (PC), ponteiro de pilha (SP) etc.
  - Registradores de propósito geral (GPR): acesso rápido a dados usados com frequência => variáveis locais, resultados intermediários de cálculos etc.
  - Registradores do kernel: usados pelo SO para controlar dispositivos, cache, memória etc.
  - PSW (Program Status Word): bits de controle necessários ao funcionamento do processador.

Composição de um conjunto de instruções (4)

➔ Aspectos importantes para o compilador

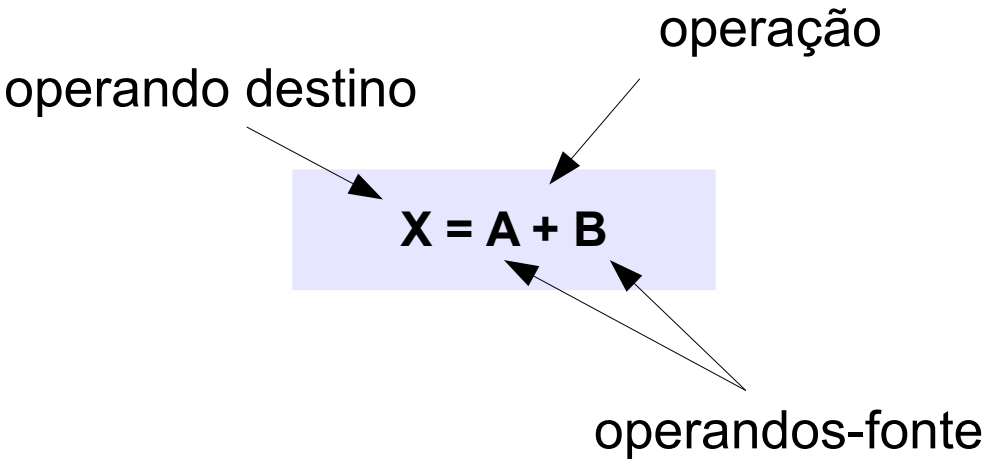
- **Tipos de dados**
  - Numéricos
    - Inteiros: 8, 16, 32 e 64 bits (contagem e identificação).
    - Reais (ponto flutuante): 32, 64 e 128 bits (cálculos e mensuração).
    - Pode-se empregar registradores diferentes para inteiros e reais.
  - Não-numéricos
    - Caracteres: ASCII (7 bits), Unicode (16 bits).
    - Strings
    - Booleanos: bytes 0 e 1
    - Bit maps: vetor de valores booleanos
    - Ponteiros: endereços de memória

Outros tipos de dados são implementados em software!

Composição de um conjunto de instruções (5)

➔ Principais decisões de projeto

- **Operandos**
  - Quantos?
  - Quais os tipos?
  - Qual a localização?
  - Como especificá-los?
- **Operações**
  - Quais?
- **Formato da instrução**
  - Tamanho?
  - Quantos formatos?



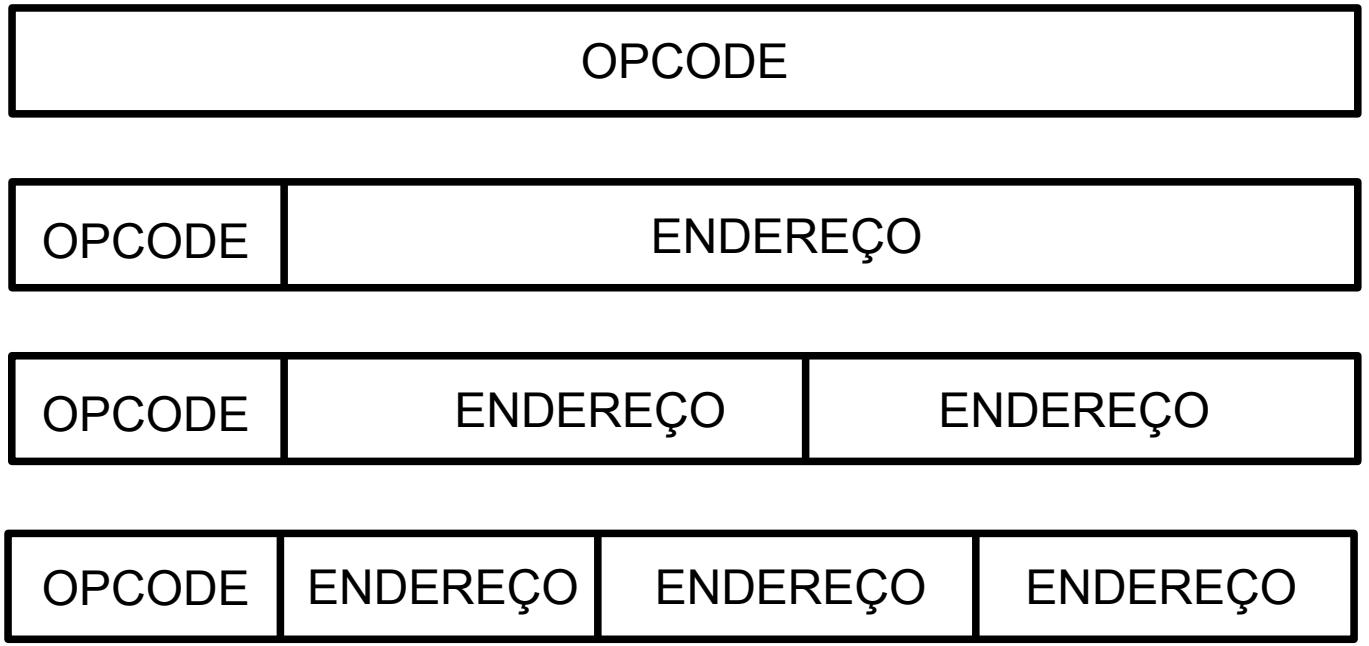
ADD R1, R2, R5

Como o processador interpreta a sequência 0001 1000 1101 1111 ?

Composição de um conjunto de instruções (6)

➔ Principais decisões de projeto

- **Quantos operandos?**
  - A maioria das instruções tem 3 operandos (ex. X = A + B).
  - A maioria das ISAs suporta de 0 a 3 operandos por instrução.



Composição de um conjunto de instruções (7)

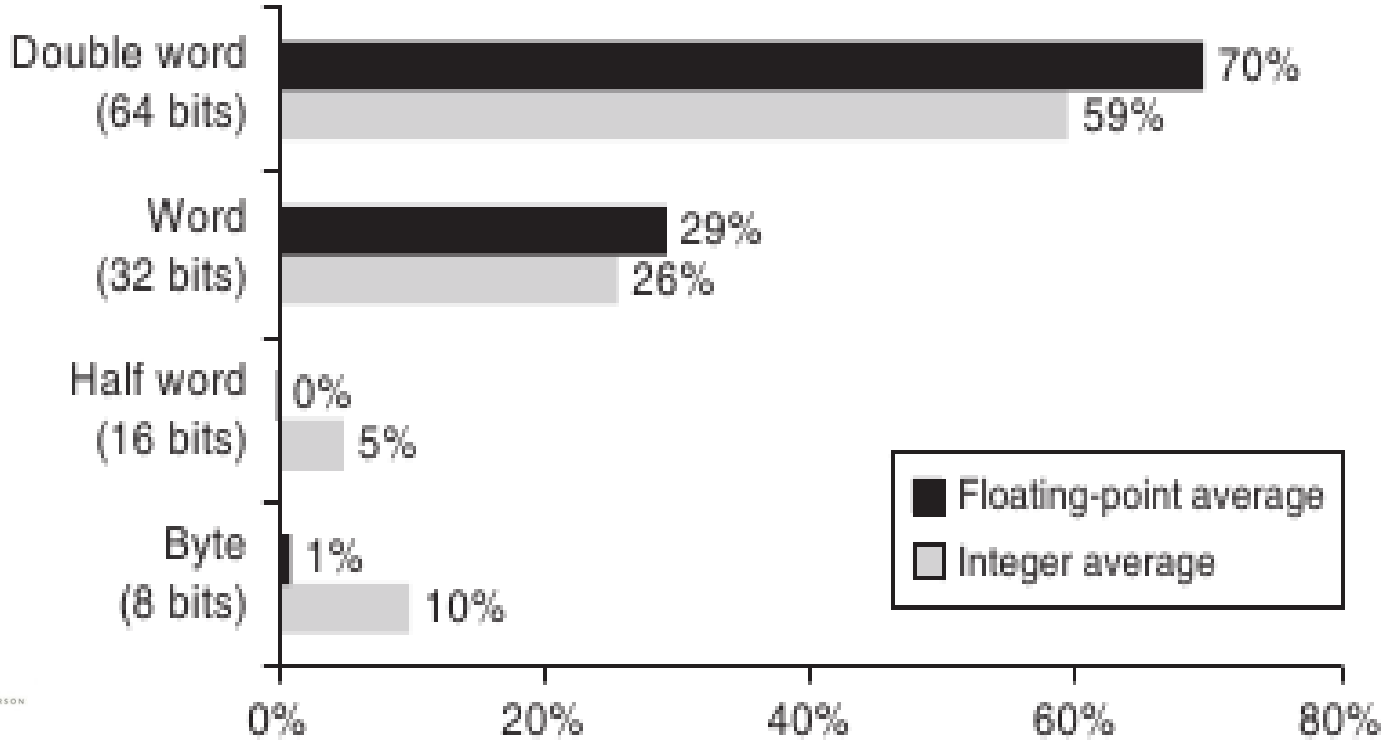
→ Principais decisões de projeto

- **Tipos dos operandos?**
  - Na maioria das ISAs (incluindo 80x86, ARM e MIPS):
    - 8 bits (*byte*): caracteres ASCII
    - 16 bits (*half word*): caracteres Unicode
    - 32 bits (*word*):
      - inteiros em complemento de 2
      - reais de precisão simples, formato IEEE 754
    - 64 bits (*long word*):
      - inteiros longos em complemento de 2
      - reais de precisão dupla, formato IEEE 754
    - 80 bits (na arquitetura 80x86):
      - reais de precisão dupla estendida

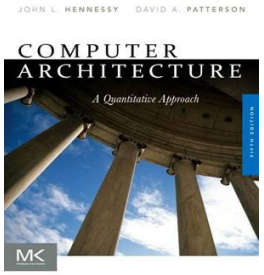
Composição de um conjunto de instruções (8)

➔ Principais decisões de projeto

- **Quais tipos são mais usados e devem ser suportados de forma mais eficiente?**



Distribuição de acesso a dados por tipo.



Composição de um conjunto de instruções (9)

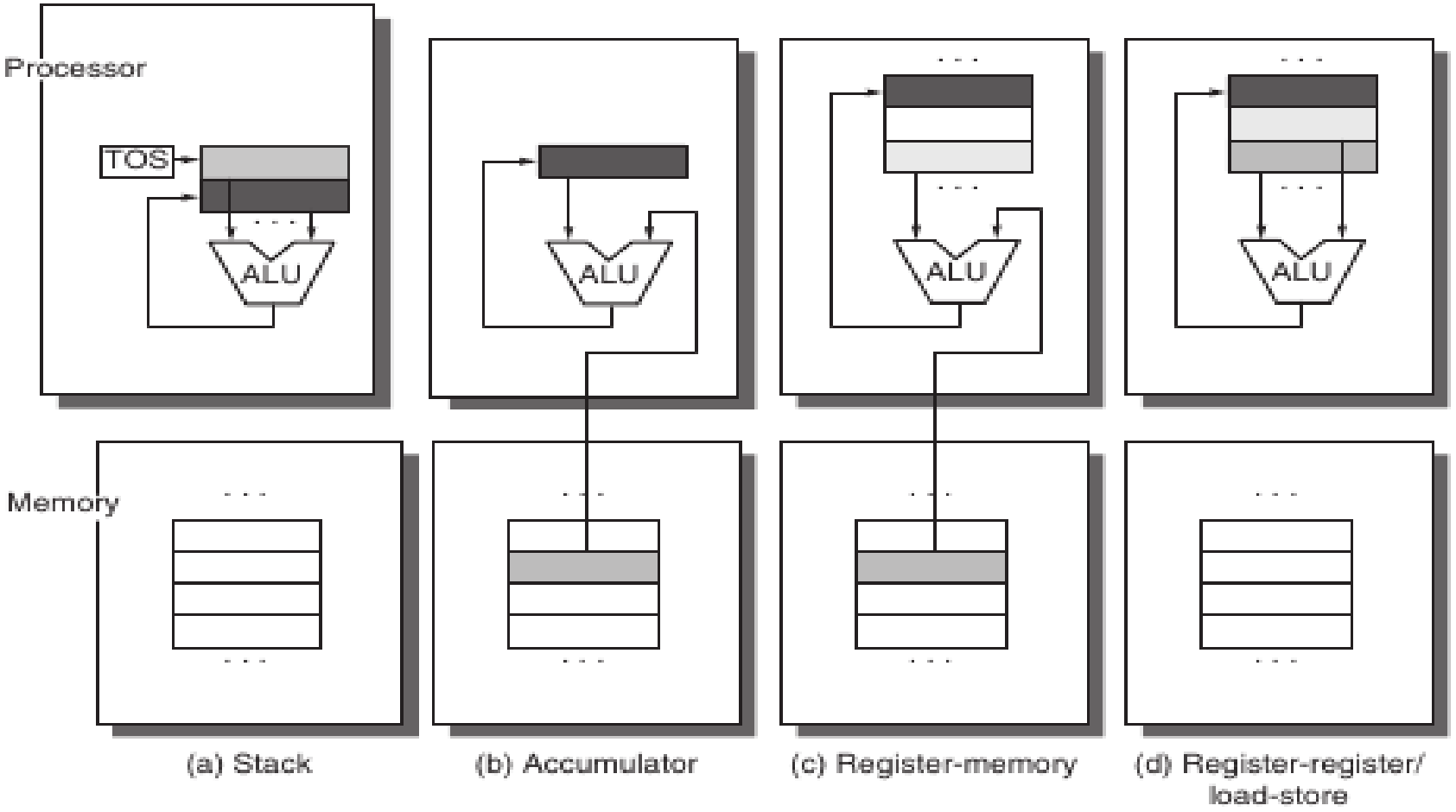
→ Principais decisões de projeto

- **Tipos dos operandos?**
  - Implícitos ou explícitos.
  - Classes básicas de ISAs, definidas em função do armazenamento interno ao processador:
    - Pilha (*stack*)
    - Acumulador
    - Registrador-memória
    - Registrador-registrador (*load/store*)

Composição de um conjunto de instruções (10)

➔ Principais decisões de projeto

- **Tipos dos operandos?** Implícitos ou explícitos



C = A + B

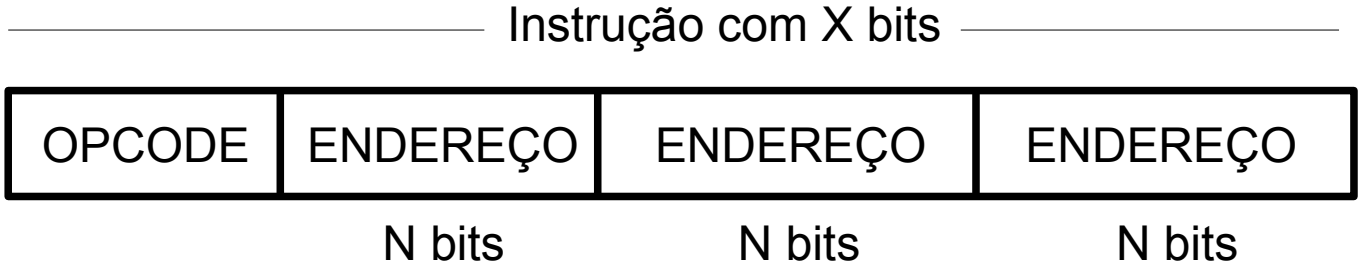
```
PUSH A
PUSH B
ADD
POP  C
```

```
LOAD  A
ADD   B
STORE C
```

```
LOAD  R1, A
ADD   R3, R1, B
STORE R3, C
```

```
LOAD  R1, A
LOAD  R2, B
ADD   R3, R1, R2
STORE R3, C
```

- **Localização dos operandos?**
  - Memória
    - 2^N endereços



  - Registradores
    - Fáceis de especificar
    - Acesso rápido

Name	Number	Use
$zero	0	The constant value 0
$at	1	Assembler temporary
$v0–$v1	2–3	Values for function results and expression evaluation
$a0–$a3	4–7	Arguments
$t0–$t7	8–15	Temporaries
$s0–$s7	16–23	Saved temporaries
$t8–$t9	24–25	Temporaries
$k0–$k1	26–27	Reserved for OS kernel
$gp	28	Global pointer
$sp	29	Stack pointer
$fp	30	Frame pointer
$ra	31	Return address

Exemplo: MIPS 32
- 32 registradores de 32 bits (ponto fixo)
- 32 registradores para ponto flutuante

Composição de um conjunto de instruções (12)

➔ Principais decisões de projeto

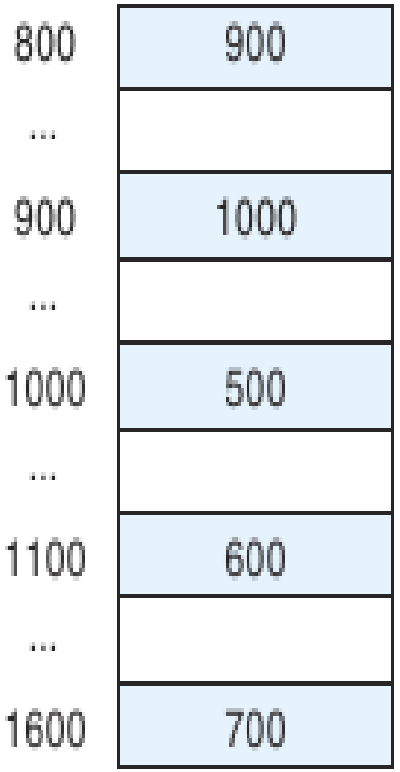
- **Como especificar a localização dos operandos?**
  - **Modos de endereçamento:** permitem especificar constantes, endereços de memória e registradores.

Modo	Localização do operando
Imediato	Valor do operando na instrução
Direto	Endereço efetivo do operando no campo de endereço da instrução
Indireto	Campo de endereço da instrução contém um endereço de memória que aponta para o endereço do operando
Registrador (direto)	Valor do operando em um registrador
Registrador indireto	Registrador contém o endereço do operando
Base + deslocamento (ou relativo ao PC)	Endereço efetivo do operando é obtido somando-se o valor do campo de endereço da instrução com um valor representando um deslocamento
Indexado	Endereçamento de arranjos (vetores e matrizes)
Pilha	Operandos implicitamente localizados na pilha

Composição de um conjunto de instruções (13)

➔ Principais decisões de projeto

- **Como especificar os endereços de memória?**
  - **Modos de endereçamento** - exemplos



Memória

R1 usado no modo indexado

`LOAD 800`

Imediato	800
Direto	900
Indireto	1000
Indexado	700

`LOAD R1`

Registrador direto	800
Registrador indireto	900

Composição de um conjunto de instruções (14)

➔ Principais decisões de projeto

- **Quais operações?**

Tipo de operação	Exemplos
Aritmética e lógica	Operações aritméticas e lógicas com inteiros: *add, subtract, multiply, divide, and, or, not.*
Movimentação de dados	Movimentação de dados entre memória e registradores: *load* e *store* (e variações).
Controle	Desvios condicionais e incondicionais (*branch, jump*), suporte a procedimentos (chamada e retorno).
Ponto flutuante	Operações aritméticas e lógicas com reais: *add, subtract, multiply, divide, compare*
Manipulação de string	Cópia, movimentação, busca, comparação
Suporte para E/S	E/S programada, baseada em interrupções ou DMA
Manipulação de bit	Deslocamento (*shift*) e rotação (*rotate*)
Suporte gráfico	Compressão e descompressão, operações pixel e vertex
Sistema	Chamada ao sistema operacional, suporte à virtualização, proteção, gerência de cache

Estudo de caso: ISA MIPS (1)

- Antes de calcular a expressão, deve-se associar registradores às variáveis,

f = (g + h) – (i + j)

```
la $s0, f // load address
la $s1, g
la $s2, h
la $s3, i
la $s4, j
```

- carregar o valor dos operandos nos registradores,

```
lw $s1, 0($s1) // load word
lw $s2, 0($s2)
lw $s3, 0($s3)
lw $s4, 0($s4)
```

- calcular a expressão e

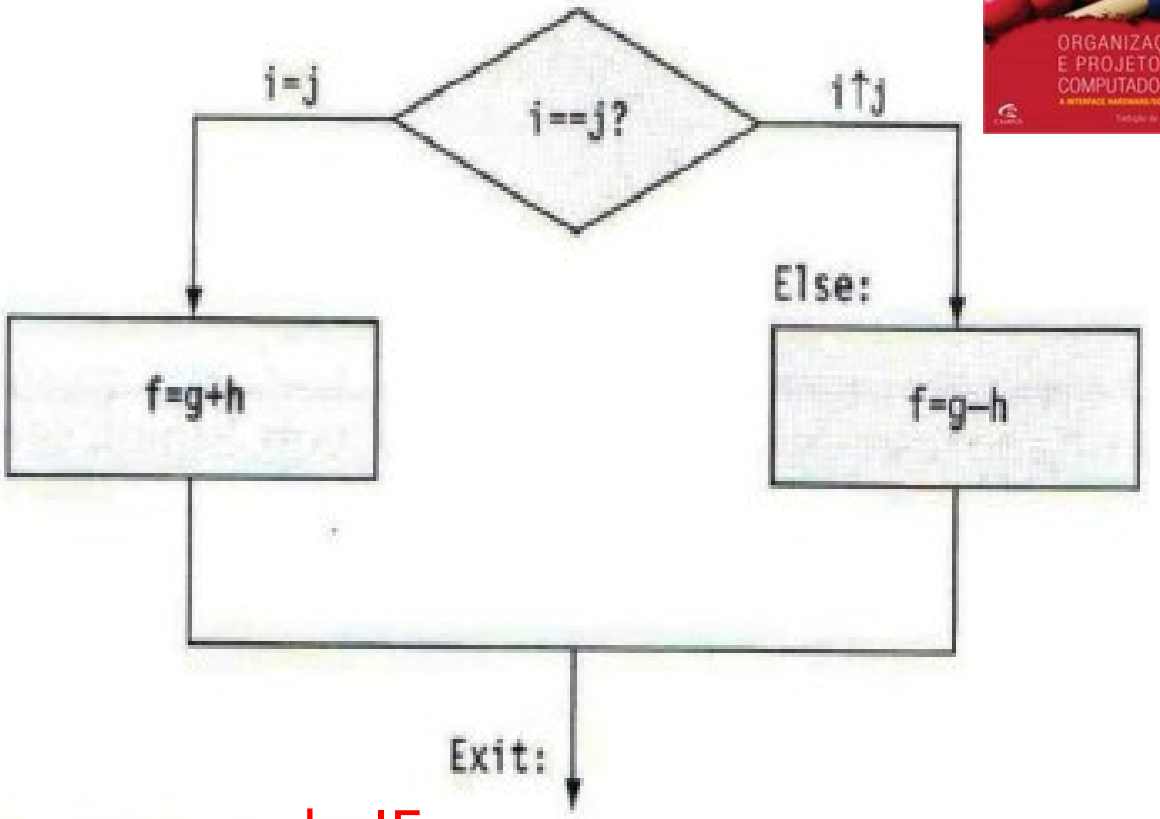
```
add $t0, $s1, $s2
add $t1, $s3, $s4
sub $t2, $t0, $t1
```

- gravar o resultado.

```
sw $t2, 0($s0)
// store word
```

Estudo de caso: ISA MIPS (2)


```
if (i == j)
    f = g + h
else f = g - h
```



- Supondo o mapeamento

```
la $s0, f
la $s1, g
la $s2, h
la $s3, i
la $s4, j
```

- Código para o comando IF

```
        bne $s3, $s4, ELSE   # vai para ELSE se i!=j
        add $s0, $s1, $s2    # f = g+h, se i==j
        j   EXIT             # vai para Exit (pula Else)
ELSE:   sub $s0, $s1, $s2    # f = g-h, se i!=j
EXIT: ...
```

```
// bne – branch if not equal
// j   – jump
```

Estudo de caso: ISA MIPS (3)


```
while (vetor[i] == k)
    i += 1;
```

- Supondo o mapeamento

```
la $s3, i
la $s5, k
la $s6, vetor
```

- Código para o comando WHILE

```
Loop:   sll $t1, $s3, 2     # $t1 = i * 4 (alinhamento)
        add $t1, $t1, $s6   # $t1 = endereço de vetor[i]
        lw  $t0, 0($t1)     # $t0 = valor de vetor[i]
        bne $t0, $s5, Exit  # termina laço se vetor[i]!=k
        addi $s3, $s3, 1    # i = i + 1
        j   Loop            # retorna para início do laço
Exit: ...

        // sll – shift left logical immediate
        // addi – add immediate
```

Composição de um conjunto de instruções (15)

➔ Principais decisões de projeto

- **Tamanho do formato binário da instrução?**
  - Variável, fixo ou híbrido

Operation and no. of operands	Address specifier 1	Address field 1	. . .	Address specifier *n*	Address field *n*

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

Operation	Address field 1	Address field 2	Address field 3

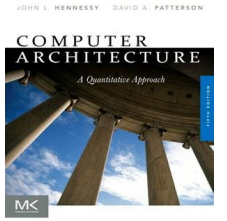
(b) Fixed (e.g., Alpha, ARM, MIPS, PowerPC, SPARC, SuperH)

Operation	Address specifier	**Address field**

Operation	Address specifier 1	Address specifier 2	Address field

Operation	Address specifier	Address field 1	Address field 2

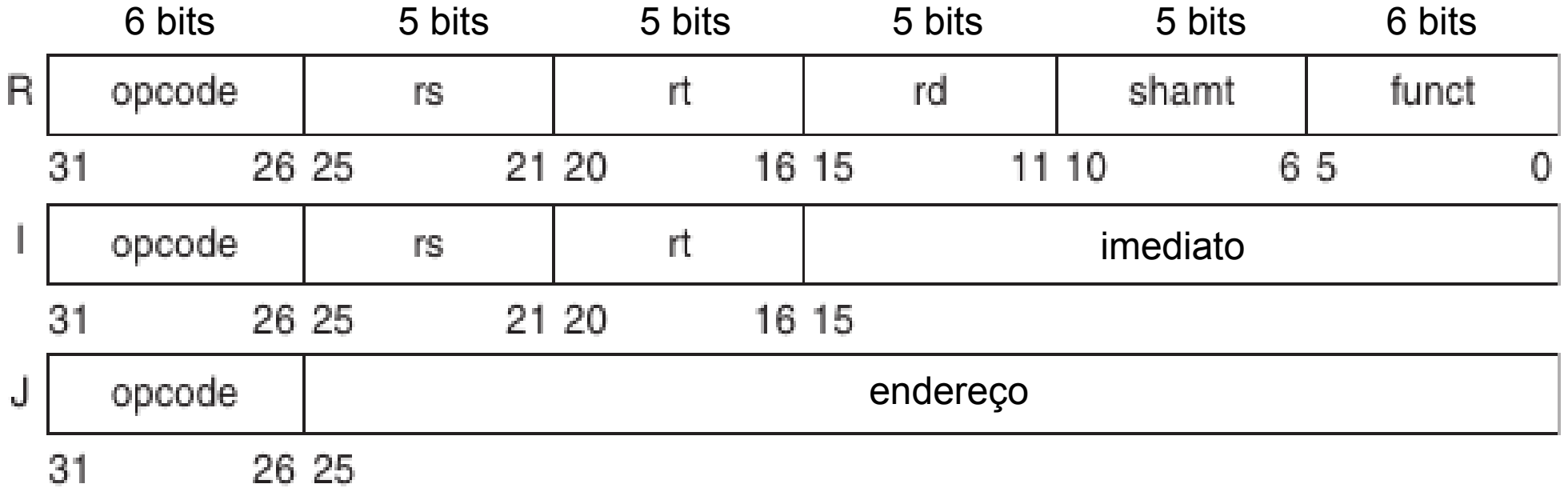
(c) Hybrid (e.g., IBM 360/370, MIPS16, Thumb, TI TMS320C54x)



Composição de um conjunto de instruções (16)

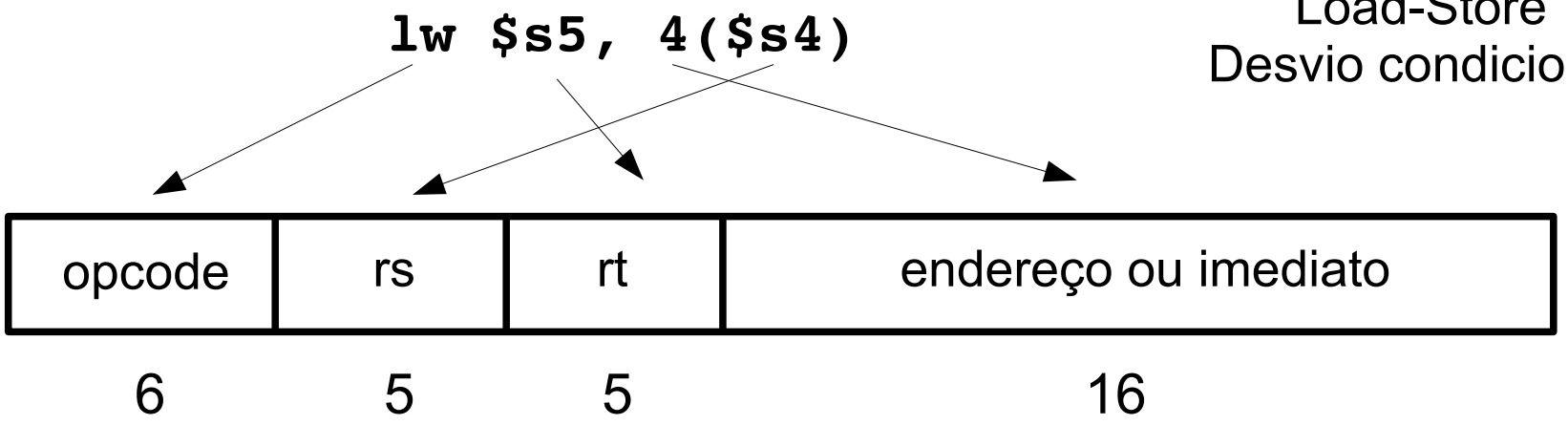
➔ Principais decisões de projeto

- **Formato binário (campos) da instrução?**
  - Exemplo: **MIPS 32**
    - Todas as instruções com 32 bits
    - O opcode informa ao processador qual é o formato

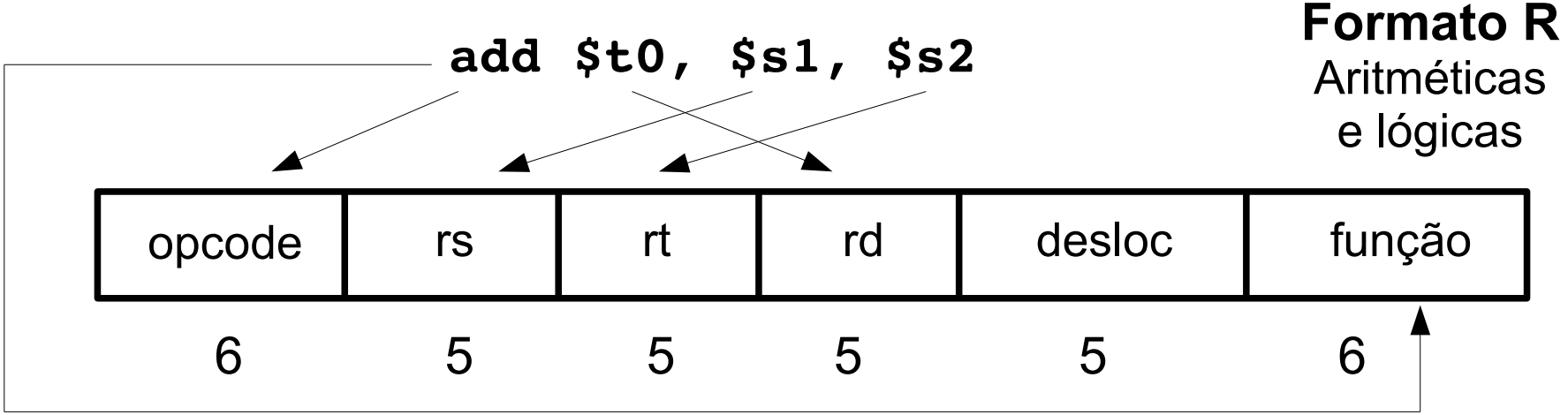


Estudo de caso: ISA MIPS (4)

Formato I
Load-Store
Desvio condicional



Formato R
Aritméticas
e lógicas



Estudo de caso: ISA MIPS (5)

```
Loop:   sll $t1, $s3, 2     # $t1 = i * 4 (alinhamento)
        add $t1, $t1, $s6   # $t1 = endereço de vetor[i]
        lw  $t0, 0($t1)     # $t0 = valor de vetor[i]
        bne $t0, $s5, Exit  # termina laço se vetor[i]!=k
        addi $s3, $s3, 1    # i = i + 1
        j   Loop            # retorna para início do laço
Exit: ...
```

8000	0	0	19	9	4	0	R
8004	0	9	22	9	0	32	R
8008	35	9	8	0			I
8012	5	8	21	2			I
8016	8	19	19	1			I
8020	2	2000					J
8024	.....						

ISA – Síntese (1)

➔ Como classificar os conjuntos de instrução?

- Pelo tipo de armazenamento interno ao processador.
- Pela quantidade de operandos explícitos na instrução.
- Pela localização dos operandos.
- Pelas operações providas no conjunto de instruções.
- Pelos tipos e tamanhos dos operandos.

Armazenamento interno	Operandos explícitos	Localização dos operandos	Acesso aos operandos por	Exemplos
Pilha	0 (operando implícito na pilha)	Pilha	Push / Pop para a pilha	B5500. HP3000/70
Acumulador	1	Acumulador	Load / Store para o acumulador	Motorola 6809
Registradores	2, 3	Registrador ou memória	Load / Store para registradores	360, VAX

ISA – Síntese (2)

➔ Resumo das características predominantes

· Armazenamento interno ao processador.
=> Arquitetura load-store baseada em GPR.

· Endereçamento de memória.
=> Modos mais usados: deslocamento, imediato e registrador indireto.

· Tipos e tamanhos dos operandos.
=> Inteiros (8, 16, 32 e 64 bits) e reais de 64 bits (formato IEEE).

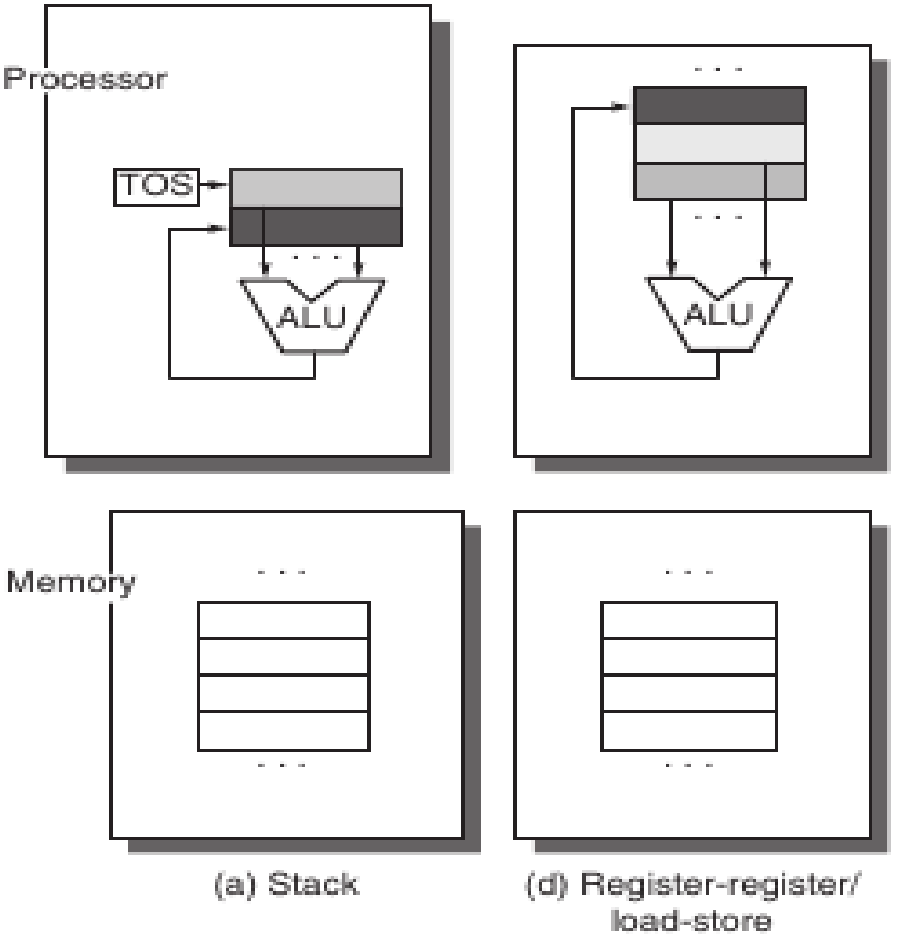
· Operações.
=> Suporte eficiente para instruções mais simples.
=> Ex.: add, sub, load, store, shift.

· Codificação (formato binário).
=> variável (tamanho do programa) X fixo (desempenho).

ISA – Síntese (3)

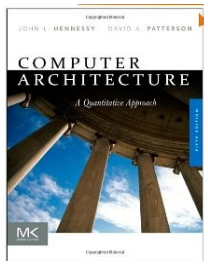
→ Métricas de comparação de ISAs

- Tamanho do código
  - quantos bytes no código executável?
- Densidade do código
  - quantas instruções em X bytes?
- Tráfego de memória
  - quantos referências à
  - memória no programa?
- Facilidade na escrita de compiladores
- Eficiência do código
  - restrições no acesso aos dados




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

Próximos passos



Princípios gerais do projeto de ISAs
Apêndice A

Estudo de caso: MIPS
Capítulo 2


```
swap(int v[], int k)
{int temp;
   temp = v[k];
   v[k] = v[k+1];
   v[k+1] = temp;
}
```

```
swap:
        muli $2, $5,4
        add  $2, $4,$2
        lw   $15, 0($2)
        lw   $16, 4($2)
        sw   $16, 0($2)
        sw   $15, 4($2)
        jr   $31
```

RISC

X

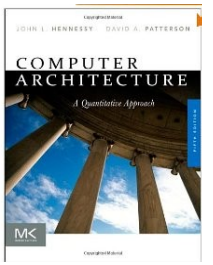
CISC

ISAs virtuais:
JVM, NVIDIA PTX

Java

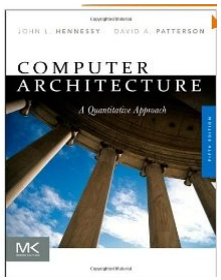
NVIDIA

Estudo de caso:
ISA 80x86
Apêndice K



Capítulo 13

Atividades



Apêndice A – exercícios A.8 (a, b, c) e A.9 (a)

Cap. 4 – exercícios com o simulador Neander
Cap. 11 – exercícios com o simulador Ramses

WEBER, R. F. Fundamentos de arquitetura de computadores. 4 ed. Porto Alegre: Bookman, 2012. *Capítulos 4 e 11* (simuladores Neander e Ramses)