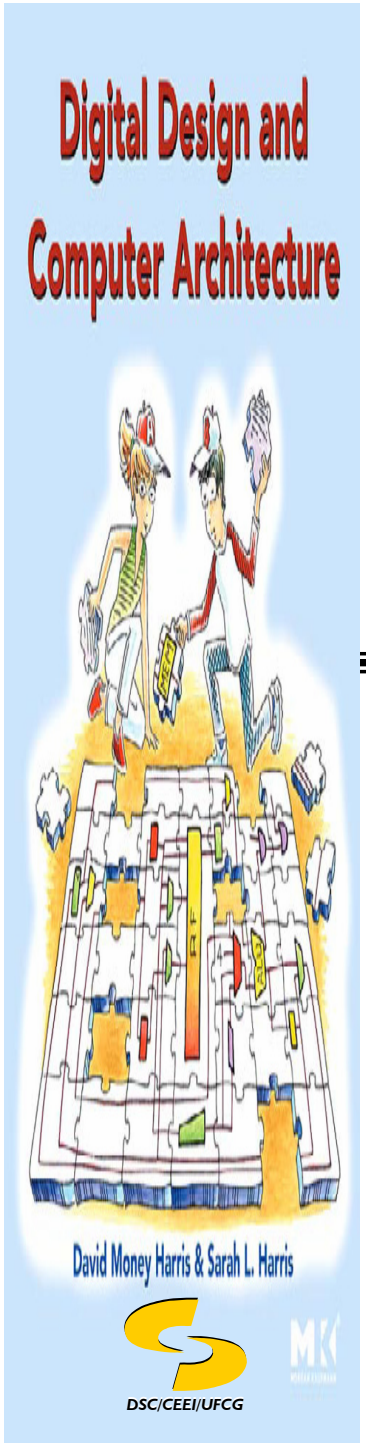


Universidade Federal de Campina Grande

Departamento de Sistemas e Computação

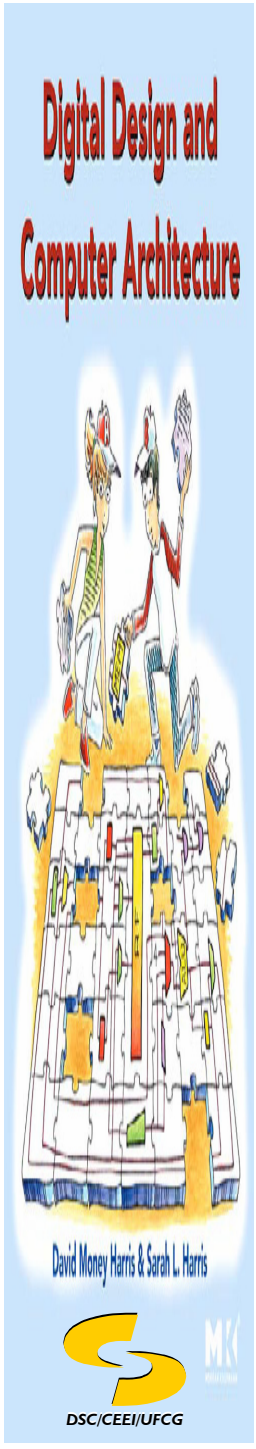
Curso de Bacharelado em Ciência da Computação

Organização e Arquitetura de Computadores I

Organização e Arquitetura Básicas de Computadores (Parte I)

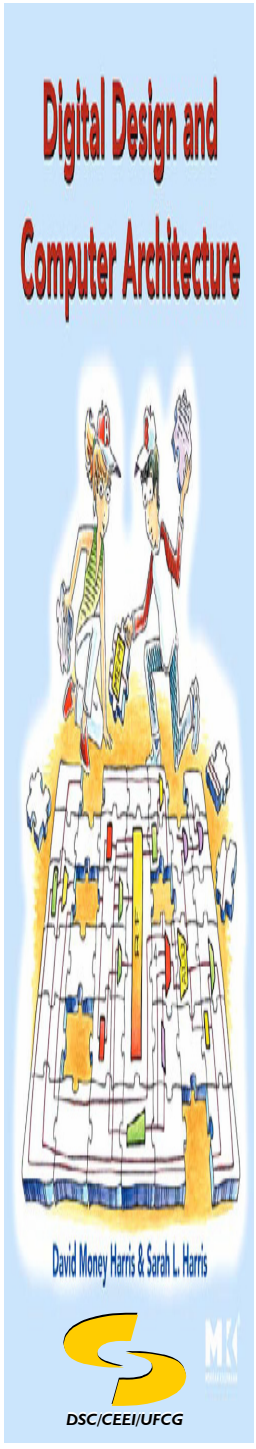
Profª Joseana Macêdo Fechine Régis de Araújo
joseana@computacao.ufcg.edu.br

Carga Horária: 60 horas



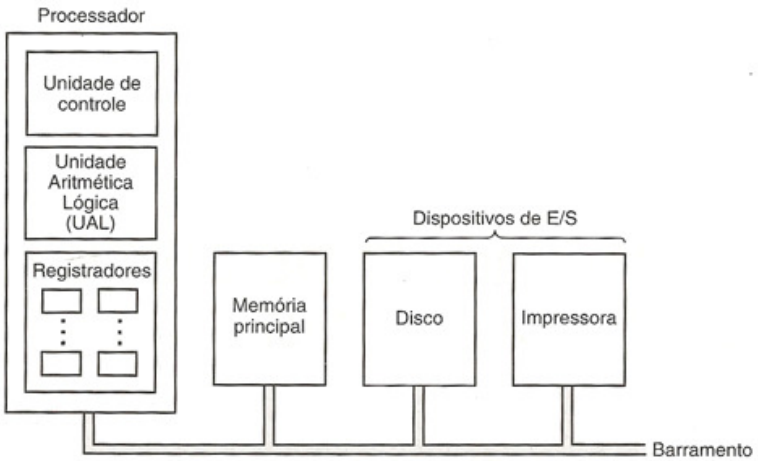
Tópicos

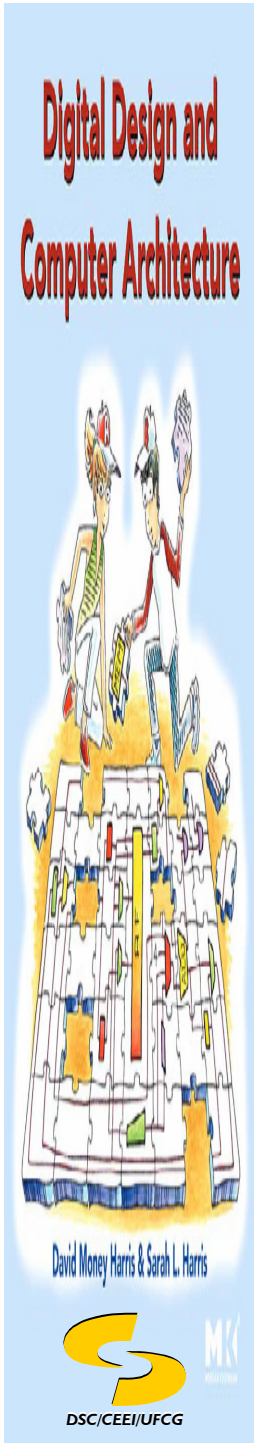
- Organização e Arquitetura Básicas de Computadores
  - Conceitos Básicos (Processadores)



Organização e Arquitetura Básicas de Computadores

- **Composição básica de um Computador digital**
  - **Processador (CPU)**
  - Memória
  - Dispositivos de entrada e saída interligados

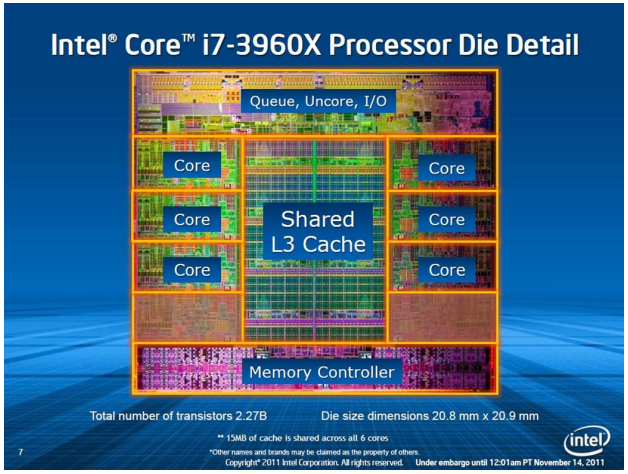


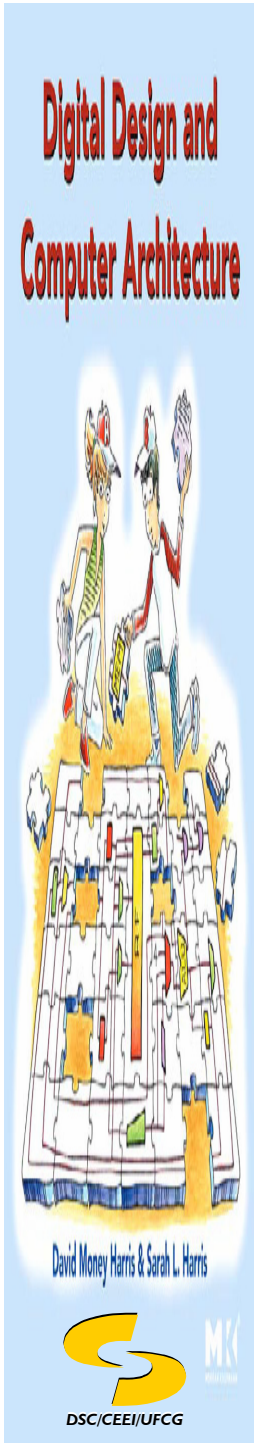


Organização e Arquitetura Básicas de Computadores

CPU - Componentes Fundamentais

- Unidade de Controle
- Unidade Aritmética e Lógica
- Registradores
- Sistemas de Comunicação (Barramentos)

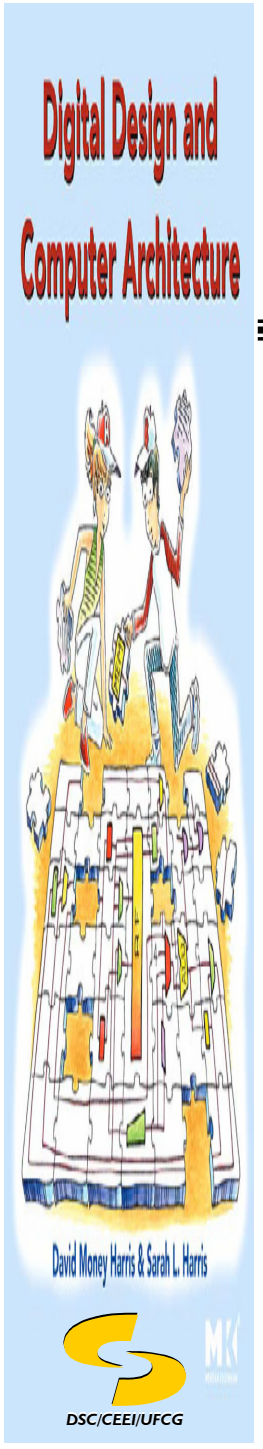




Organização e Arquitetura Básicas de Computadores

UC - Unidade de Controle

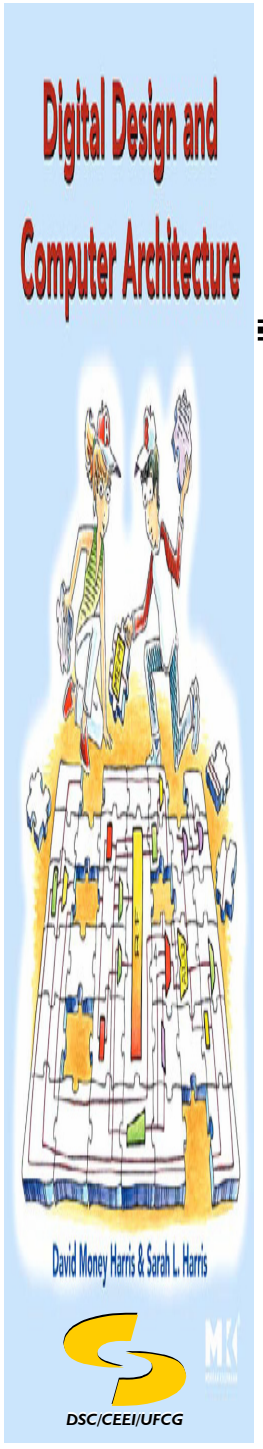
- **Funções**: busca, interpretação e controle de execução das instruções, e o controle dos demais componentes do computador.
- Envia ordens de cálculo para a UAL, que indica os valores a processar, e os coloca nos registradores para esse efeito.
- A partir da UC a informação é transferida para as outras partes que constituem o computador, como a memória, os sistemas de E/S, etc..



Organização e Arquitetura Básicas de Computadores

UAL - Unidade Aritmética e Lógica (ULA)

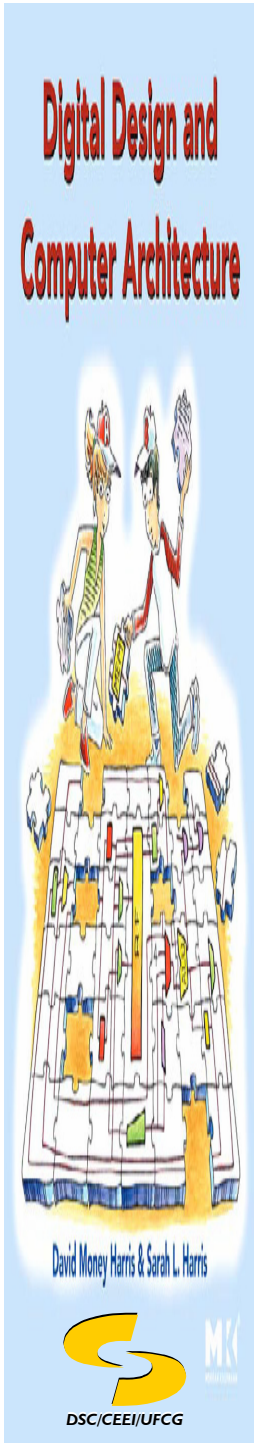
- **Função**: a execução efetiva das instruções.
- Aglomerado de circuitos lógicos e componentes eletrônicos simples que, integrados, realizam as operações aritméticas e lógicas (soma, subtração, multiplicação, divisão, AND, OR, XOR, complemento, deslocamento, incremento e decremento).
- Processadores modernos utilizam mais de uma UAL.



Organização e Arquitetura Básicas de Computadores

Registradores

- **Função**: armazenamento de dados e resultados que serão usados pela UAL.
- Servem de memória auxiliar básica para a UAL.
- Classificação (atual): registradores de uso geral e registradores de uso específico.
- Em geral, os registradores de dados da UCP têm uma largura (quantidade de bits que podem armazenar) igual ao tamanho estabelecido pelo fabricante para a **palavra do referido processador**.
- A quantidade e o emprego dos registradores variam bastante de modelo para modelo de UCP.

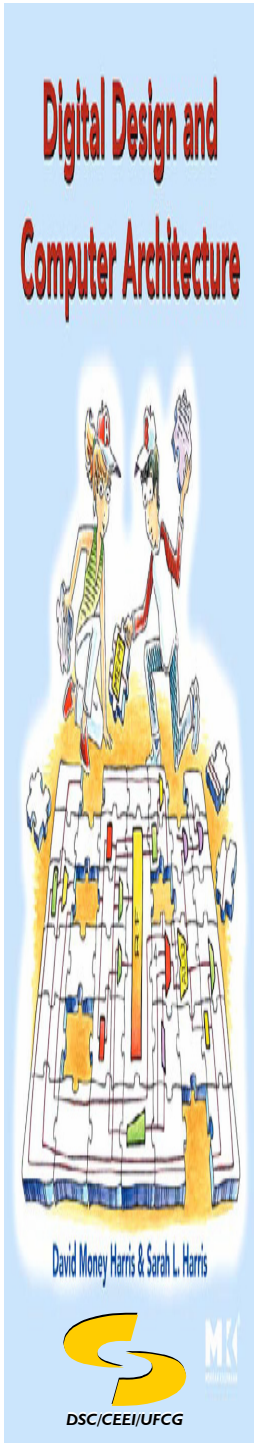


Organização e Arquitetura Básicas de Computadores

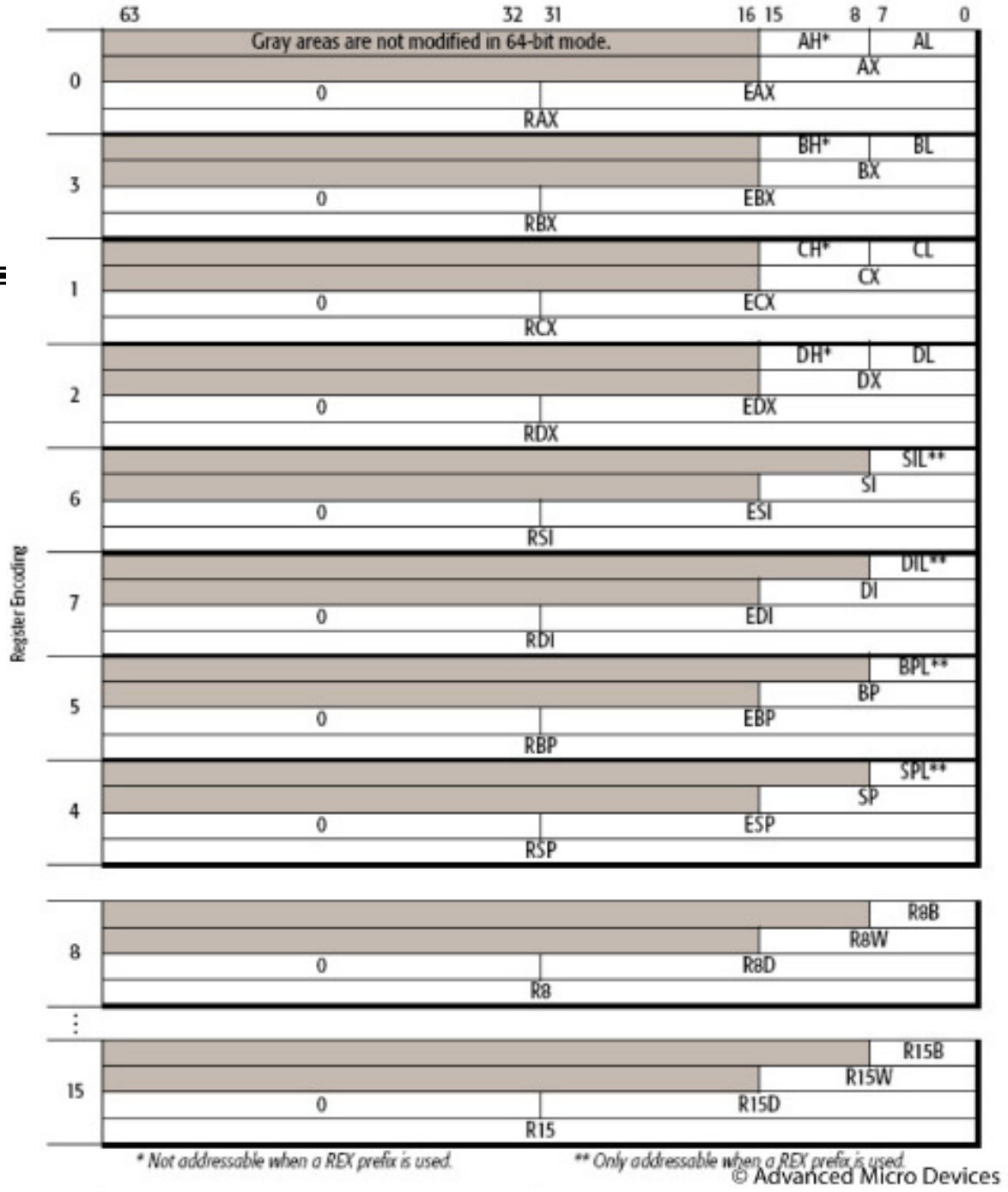
Registradores

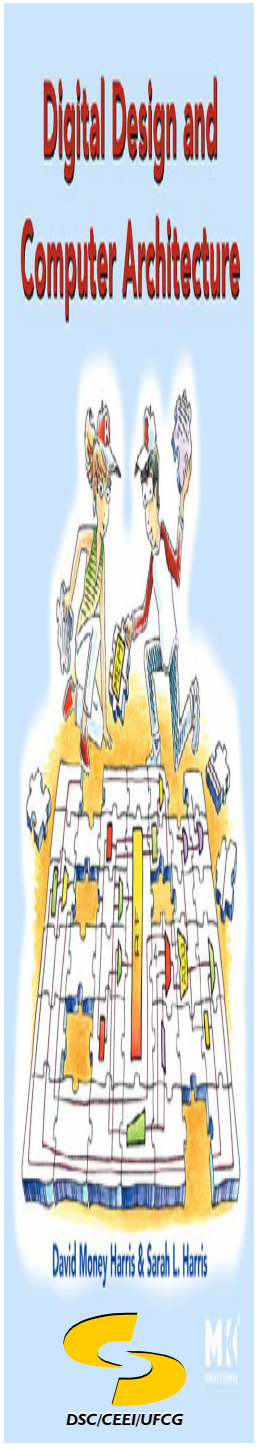
- Exemplos:
  - *Program Counter* (PC): armazena o endereço da próxima instrução
  - *Registrador de Instruções* (IR): armazena instrução que está sendo executada.
  - Registradores de uso geral, registradores de segmentos, registrador FLAGS (PSW - *Program Status Word*), ...

Mais informações:
http://www.numaboa.com.br/informatica/oiciliS/assembler/referencias/arquitetura.php



Registradores (Intel):

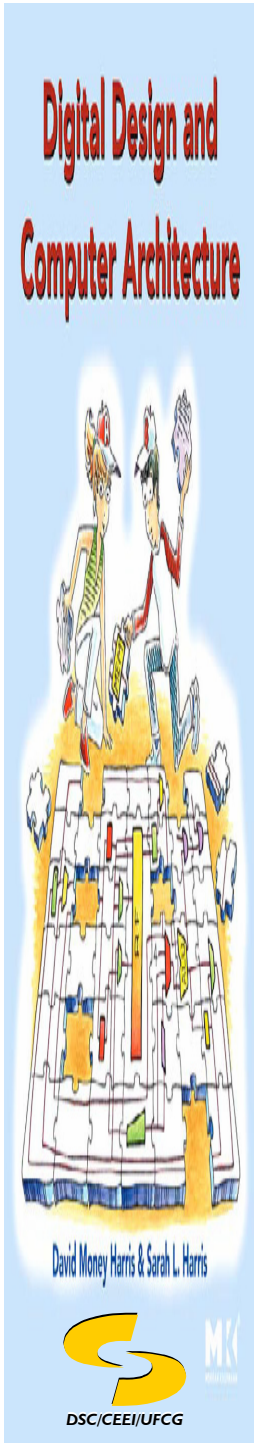




Organização e Arquitetura Básicas de Computadores

Registradores (processadores de 64 bits, intel)

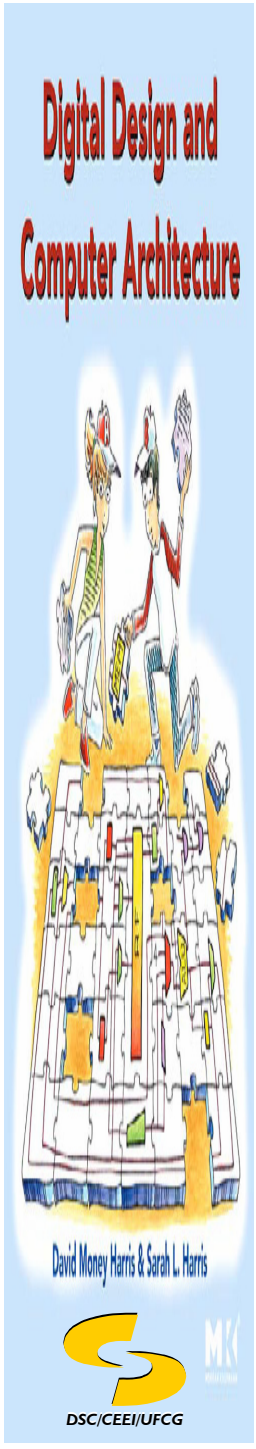
- Os registradores do x86 foram estendidos para 64 bits e receberam o prefixo “R”.
- Registrador de 64 bits correspondente ao EAX -> RAX.
- Foram acrescentados mais 8 registradores de uso geral: R8-R15.
- Foram acrescentados 8 registradores XMM: XMM8-XMM15
- O apontador de instrução, EIP, foi ampliado também, e agora se chama RIP.
- Registrador de FLAGS aumentou, embora não foram acrescentadas novas flags.



Organização e Arquitetura Básicas de Computadores

Barramentos

- Conjunto de fios paralelos que permite a transmissão de dados, endereços, sinais de controle e instruções
- Tipos: barramentos internos e externos ao processador

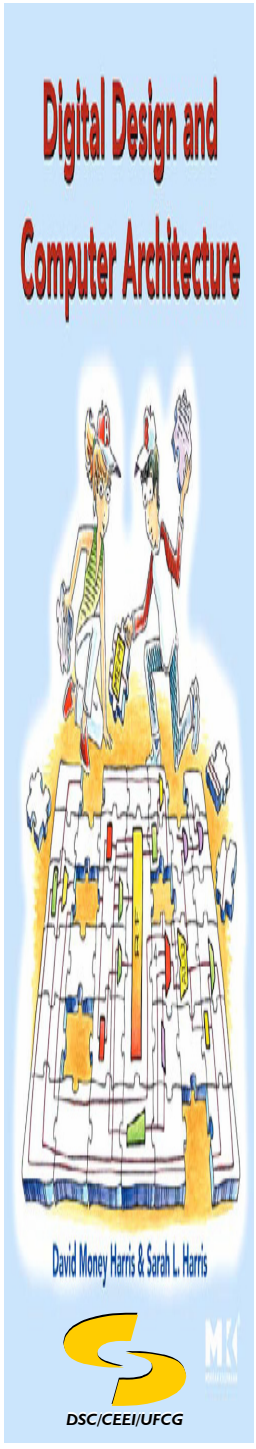


Organização e Arquitetura Básicas de Computadores

Arquitetura de um computador simples

- **Composição**
  - Caminho de Dados (*Datapath*)
  - Unidade de Controle para controlar as operações do *Datapath*

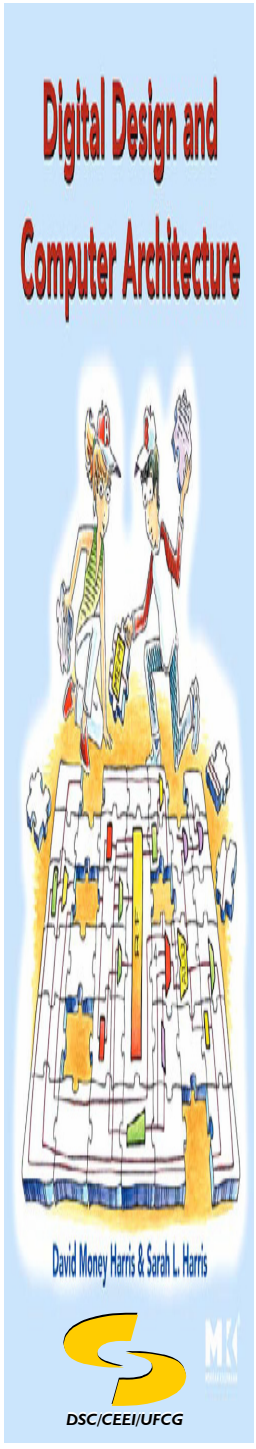
- **Especificação de um *Datapath***
  - Um conjunto de registradores
  - As micro-operações
    - ULA
    - *Shifter*
  - Um interface de controle



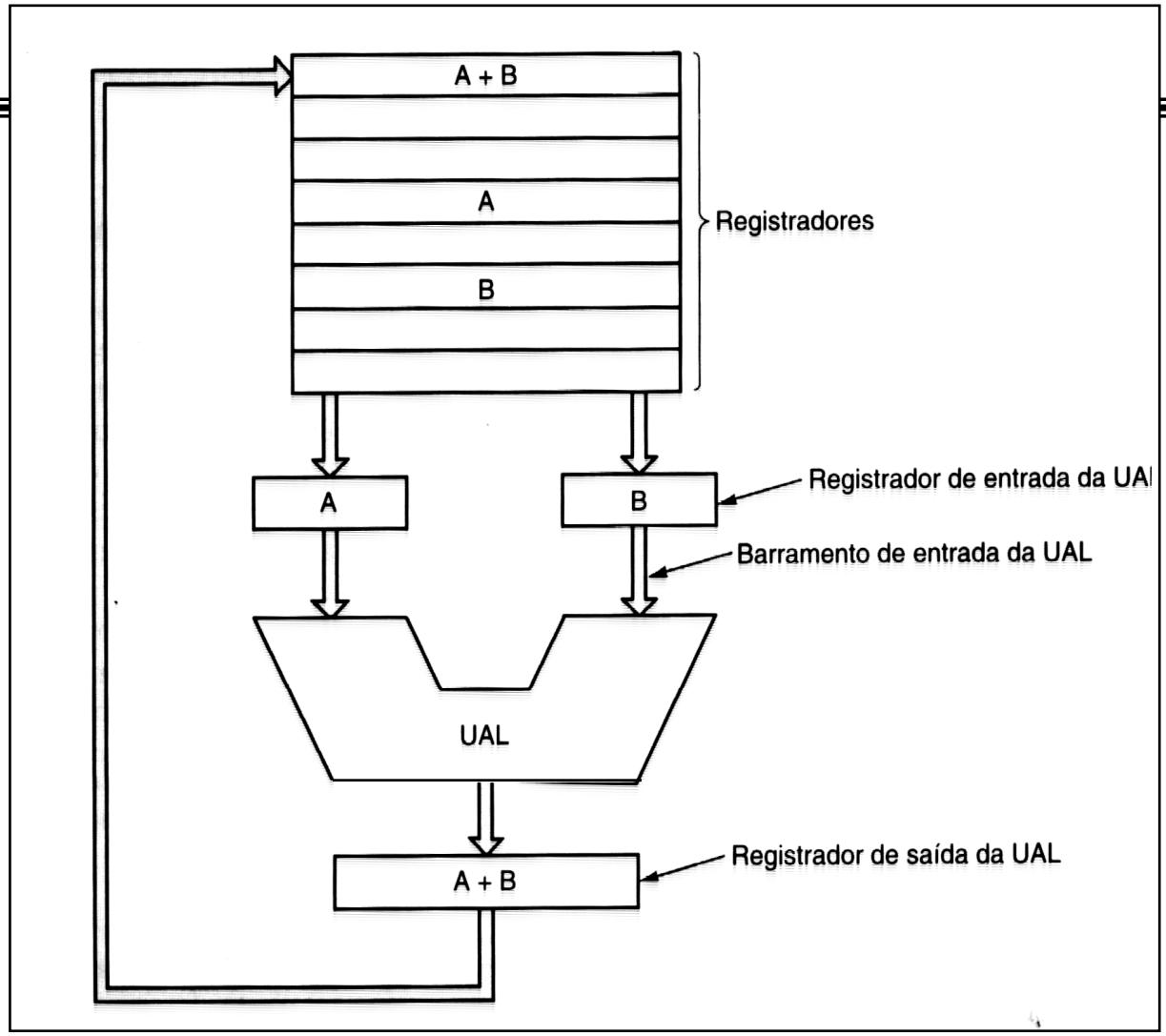
Organização e Arquitetura Básicas de Computadores

Caminho de dados

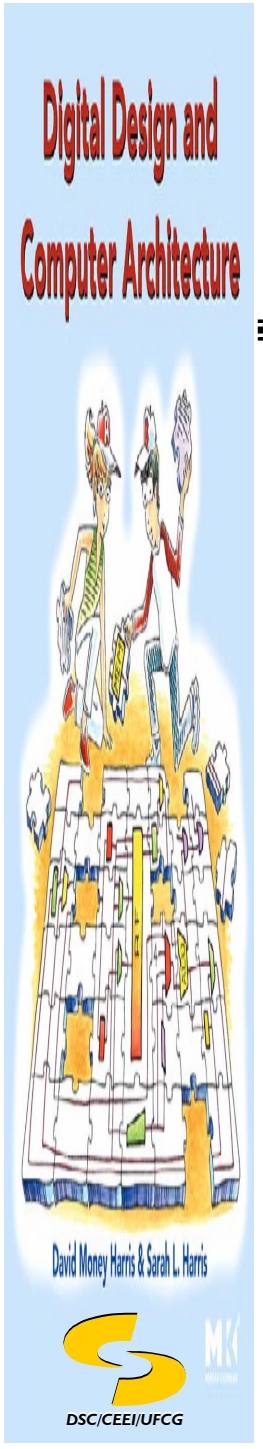
- Parte constituída dos registradores, UAL e barramentos
- Os registradores alimentam as duas entradas (A e B) da UAL
- A saída da UAL é conectada a um dos registradores



Organização e Arquitetura Básicas de Computadores



Caminho de dados de uma Máquina típica de Von Neumann.

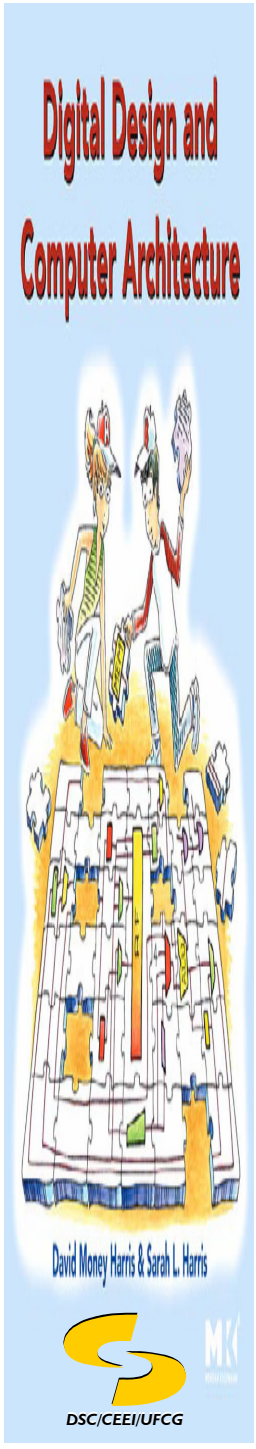


Organização e Arquitetura Básicas de Computadores

Caminho de dados

- **Importante**: A velocidade do ciclo do caminho de dados determina, em última análise, a velocidade do computador.

- **Observação**: "Palavras" – são as unidades de dados movidas entre a memória e os registradores. A referência a uma palavra deve ser feita por meio de um número inteiro.

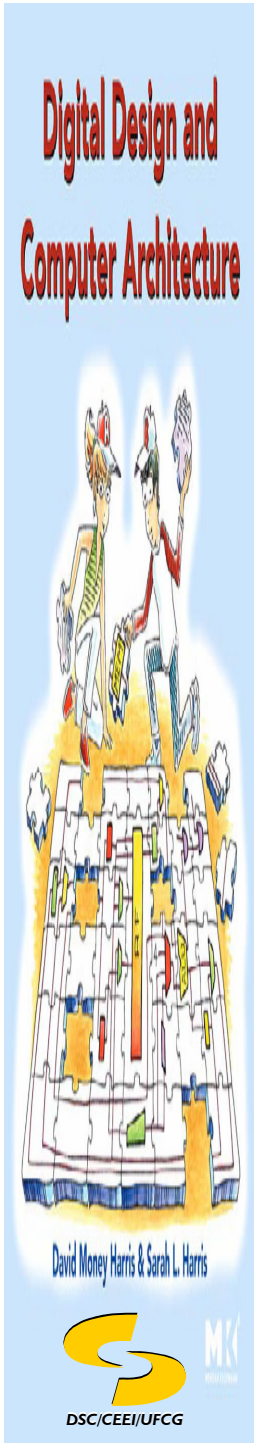


Organização e Arquitetura Básicas de Computadores

Arquitetura de um computador simples

- Independente da classe da instrução, as duas primeiras etapas para sua execução são as mesmas:
  - Enviar o PC para a memória e buscar a instrução;
  - Ler um ou dois registradores (usando o campo da instrução, para selecionar os registradores a serem lidos).

- Após a utilização da ULA, os passos são diferentes para as diferentes classes.

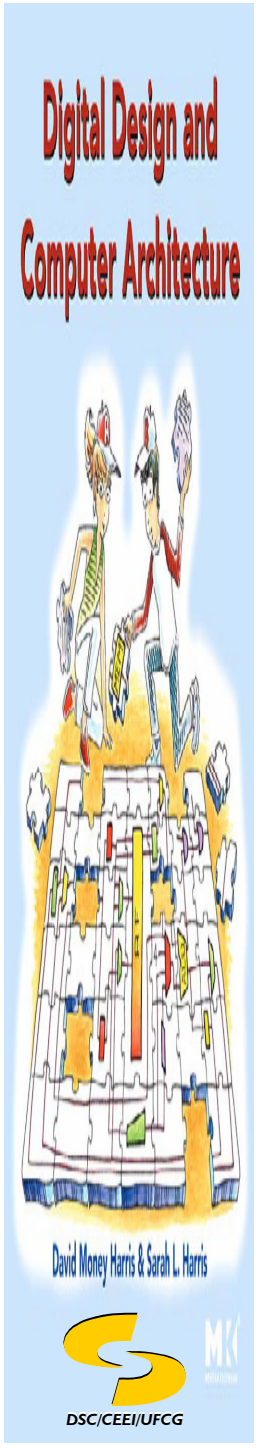


Computador Simples - Exemplo

MIPS

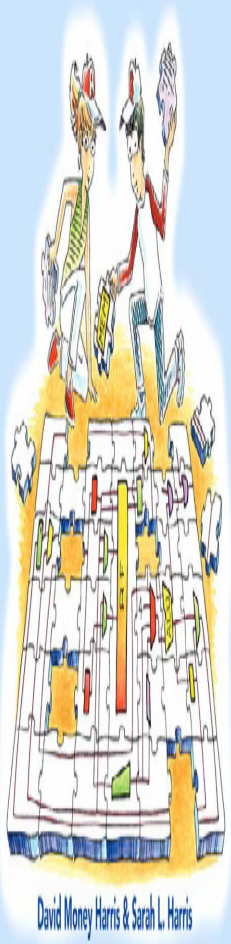
(*Microprocessor without Interlocking Pipeline Stages*)

- Arquitetura tipo RISC
- Versões de 32 e 64 bits
- Quase 100 milhões de processadores MIPS fabricados em 2009
- Usada pela NEC, Nintendo, Cisco, Silicon Graphics, Sony, impressoras HP e Fuji, etc



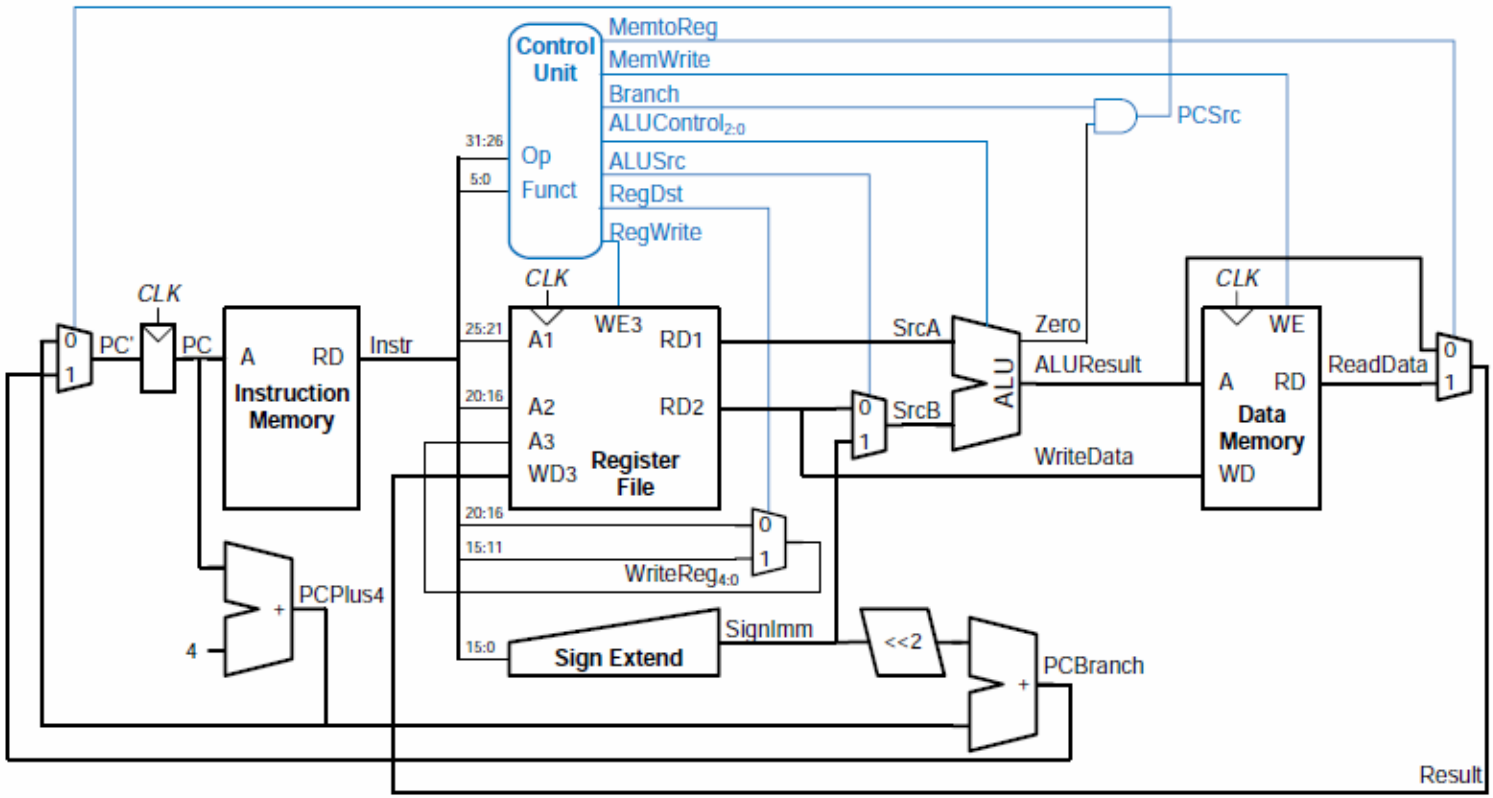
Exemplo - MIPS

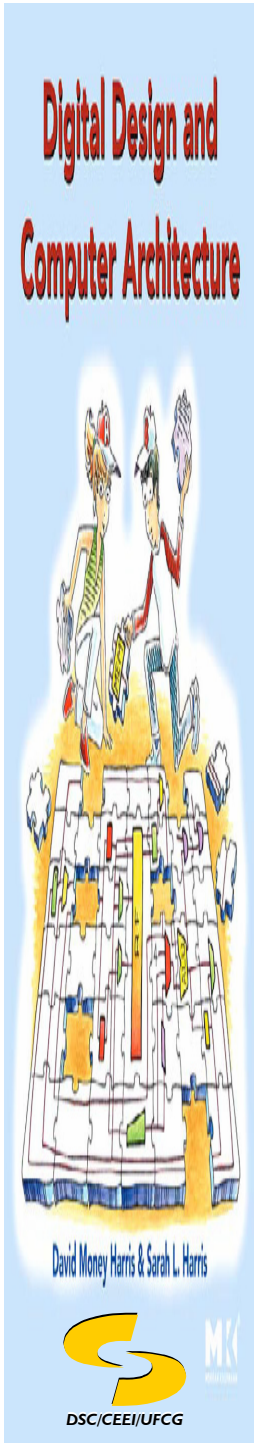
- **Componentes básicos**
  - Barramento
  - Unidade de controle
  - Banco de registradores
  - Unidade lógica e aritmética (ALU)
  - Contador de programa (PC)
  - Memória
  - Registrador de instruções (IR)



Exemplo - MIPS

Composição básica



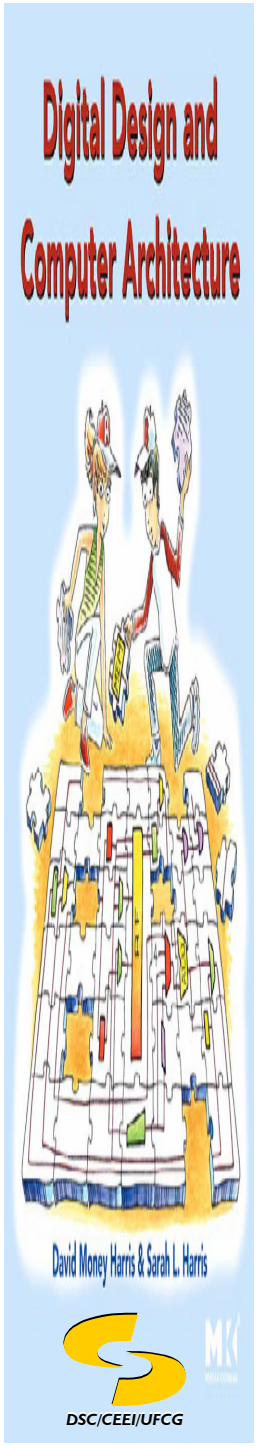


Exemplo - MIPS

Registradores

Name	Number	Use
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2–3	procedure return values
$a0-$a3	4–7	procedure arguments
$t0-$t7	8–15	temporary variables
$s0-$s7	16–23	saved variables
$t8-$t9	24–25	temporary variables
$k0-$k1	26–27	operating system (OS) temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address

Registrador	Número	Uso
$zero	0	Valor constante igual a zero
$v0-$v1	2-3	Retorno de funções
$a0-$a3	4-7	Argumentos de funcoes
$t0-$t7	8-15	Temporários, não precisam ser salvos
$s0-$s7	16-23	Salvos por uma função chamada
$t8-$t9	24-25	Mais temporários
$gp	28	Apontador global
$sp	29	Apontador de pilha
$fp	30	Apontador de quadro
$ra	31	Endereço de retorno

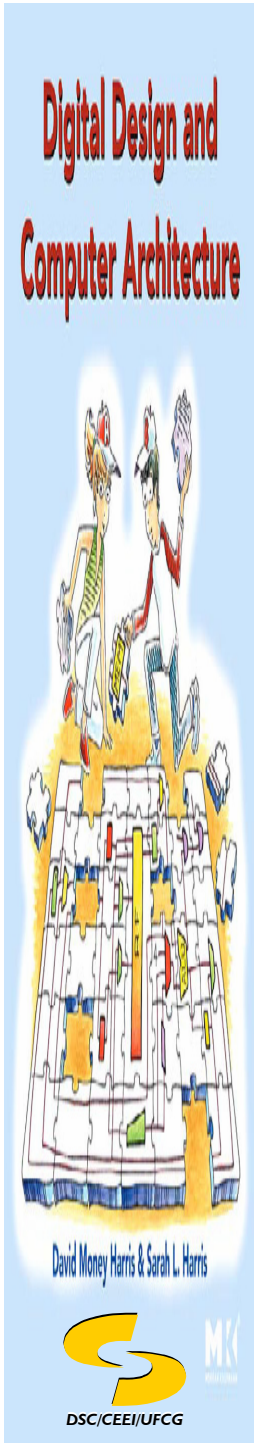


Exemplo - MIPS

- **Instruções são codificadas em bits**
  00000010010100111000100000100000

- **Programas são armazenados na memória**
  - **Instruções são lidas e escritas da memória assim como os dados que serão transformados (processados)**

- **Ciclo de busca e execução da instrução (simplificado)**
  1. **Instruções são lidas da memória e carregadas no registrador RI**
  2. **Os bits da instrução guardada no registrador RI são decodificados e controlam as ações subsequentes**
  3. **A instrução é executada e o endereço para a leitura da próxima instrução é calculado**



Exemplo - MIPS

- Todas as instruções aritméticas e lógicas possuem três operandos
  - A ordem dos operandos é fixa (destino primeiro)

```
[label:] Op-Code [operando], [operando], [operando] [#comentário]
```

- Linguagem de Montagem (Assembly)

- Sintaxe de instruções assembly:
  1. Label: opcional, identifica bloco do programa
  2. Código de operação: indicado por um Mnemônico
  3. Operandos: Registradores ou memória
  4. Comentários: opcional, tudo que vem depois do #

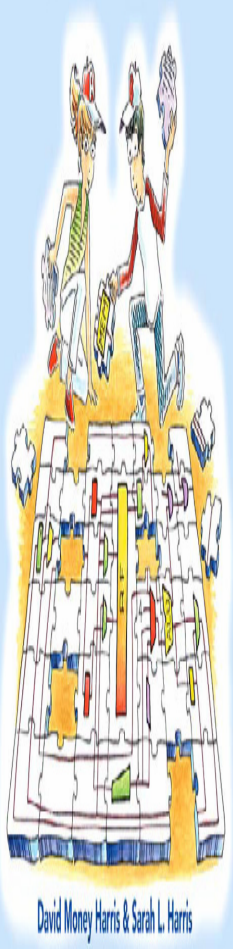
Exemplo - MIPS

1º Princípio de projeto MIPS

- Simplicidade favorece regularidade
  - Mais que três operandos por instrução exigiria um projeto de hardware mais complicado

2º Princípio de projeto MIPS

- Menor significa mais rápido
  - Uma quantidade maior que 32 registradores exigiria:
    - Um ciclo de clock maior
    - Formato de instruções maior, para comportar mais bits de endereçamento



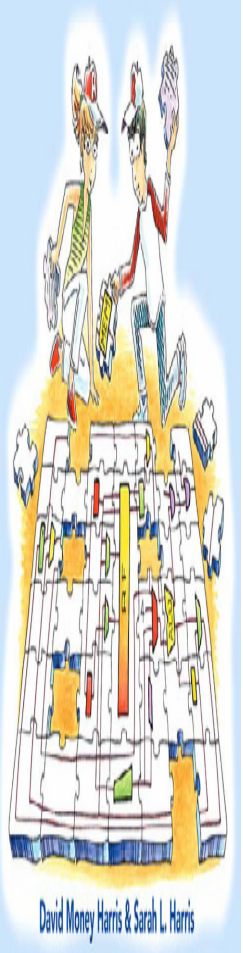
Exemplo - MIPS

Instruções MIPS (Armazenamento na Memória)

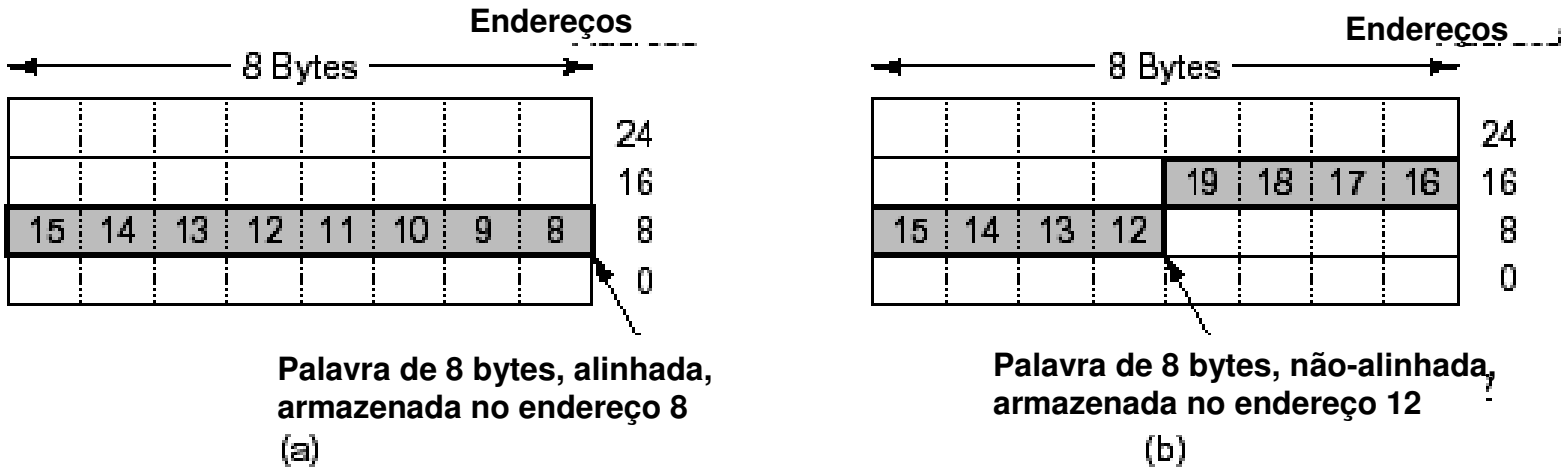
- MIPS exige que todas as palavras comecem em endereços que são múltiplos de 4 Bytes
  - **Alinhamento**: objetos devem estar em um endereço que é um múltiplo do seu tamanho

- Dois sistemas para numeração dos Bytes dentro uma palavra
  - Big endian
  - Little endian

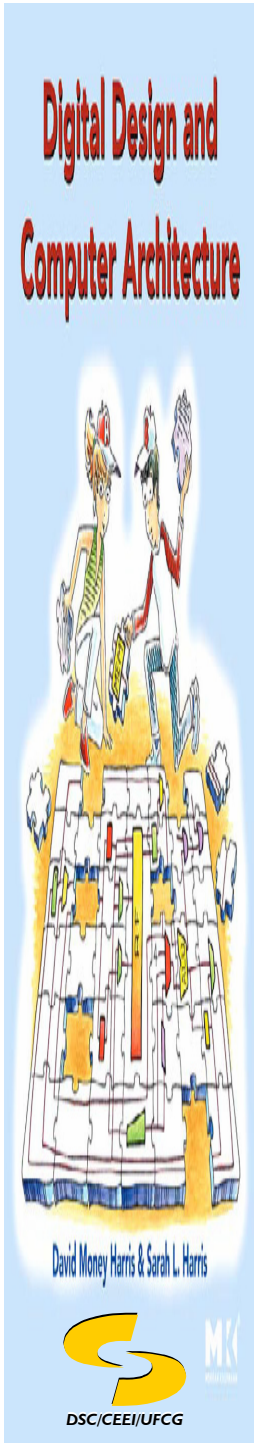
Os termos *big endian* (maior valor-big-em primeiro lugar-menor endereço) e *little endian* (menor valor-little-em primeiro lugar) foram inseridos no jargão da computação por um artigo publicado em 1981, citando o problema e relacionando-o a um episódio mencionado no livro As Viagens de Gulliver – povo que foi à guerra para decidir qual a melhor maneira de quebrar ovos, se pelo maior (*big*) lado ou se pelo menor (*little*) lado.



Exemplo - MIPS



Uma palavra de 8 bytes em uma memória little-endian. (a) Alinhada (b) Não Alinhada. Algumas máquinas exigem que as palavras da memória sejam alinhadas.



Exemplo - MIPS

Instruções MIPS (Armazenamento na Memória)

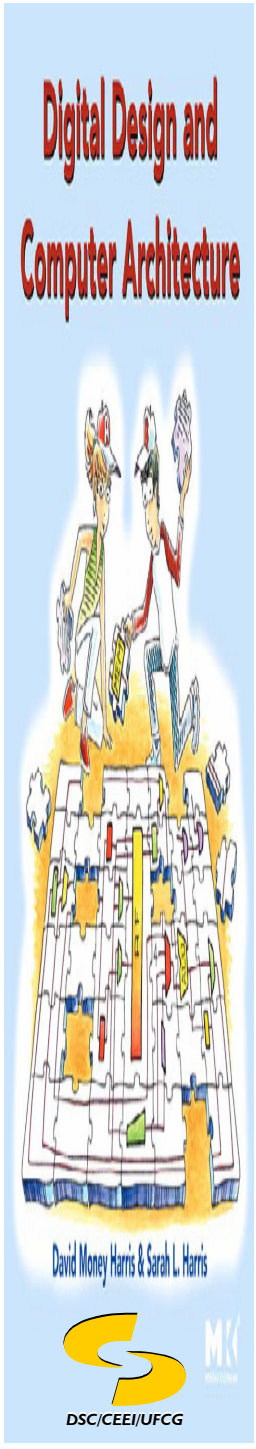
Valor em hexadecimal: 6151CE94

40	61
41	51
42	CE
43	94

(a) *Big endian*

40	94
41	CE
42	51
43	61

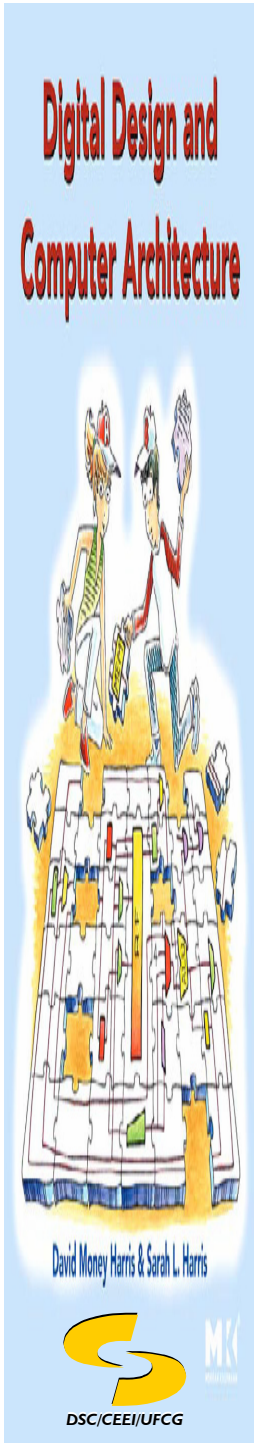
(b) *Little endian*



Exemplo - MIPS

Tipos de Dados

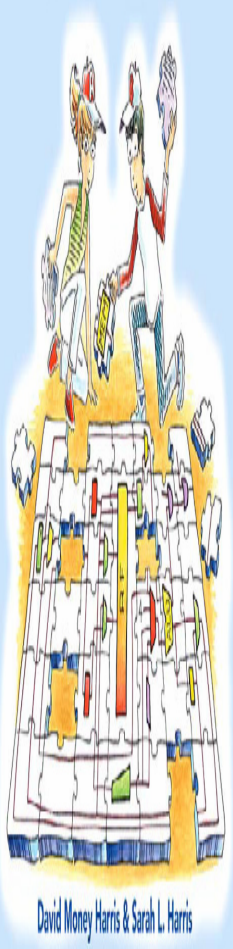
- **Numéricos**
  - Inteiros
  - Ponto Flutuante
- **Não Numéricos**
  - Textos ou alfanuméricos



Exemplo - MIPS

Instruções MIPS

- Instruções do programa assembly devem ser traduzidas em números binários para que a máquina as execute.

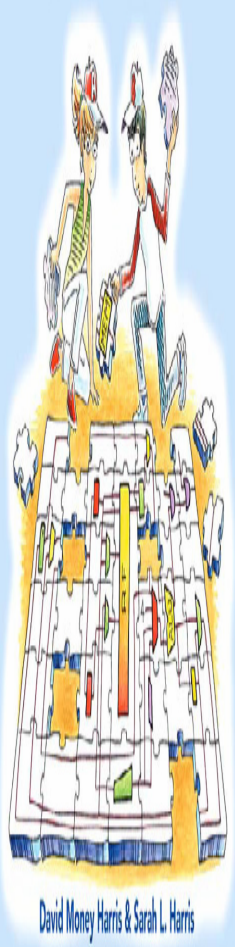


Exemplo - MIPS

Instruções MIPS

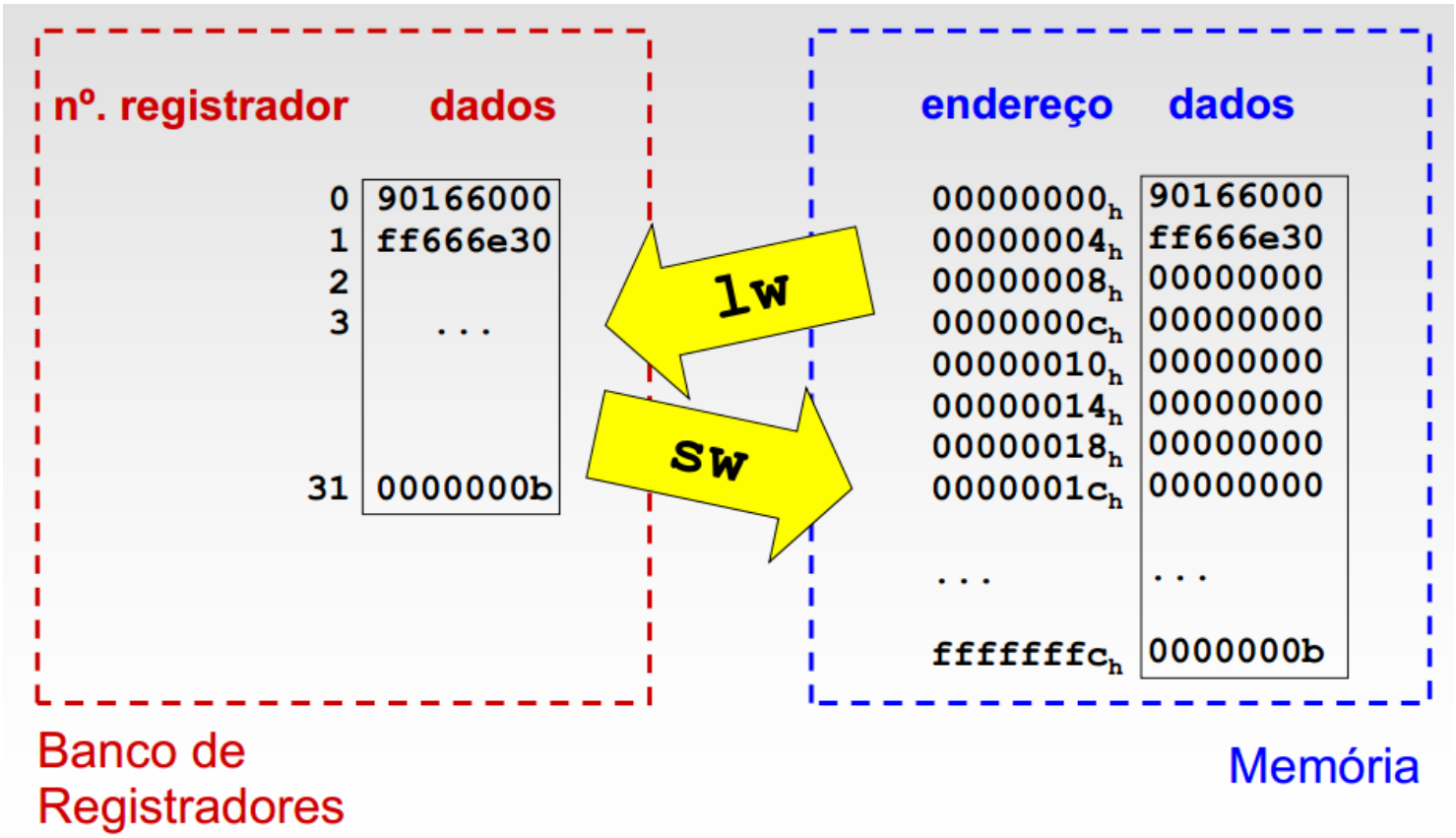
- Transferência de Dados
- Lógicas e Aritméticas
- Controle
- Suporte a procedimentos

Operações lógicas e aritméticas só ocorrem entre Registradores. Portanto, instruções para transferir dados entre a memória e os registradores são necessárias, antes da execução de tais operações.



Exemplo - MIPS

Instruções MIPS (Transferência de Dados)

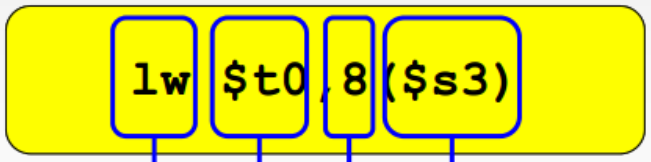


Exemplo - MIPS

Instruções MIPS (Transferência de Dados)

Copiar dados de → para	Instrução
Memória → Registrador	load word (lw)
Registrador → Memória	store word (sw)

Formato:

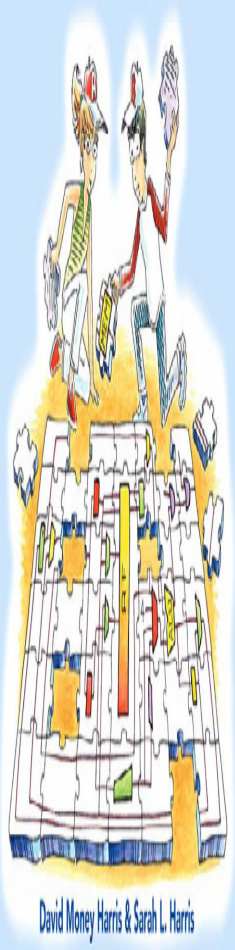


instrução

registrador de destino

offset (em Bytes)

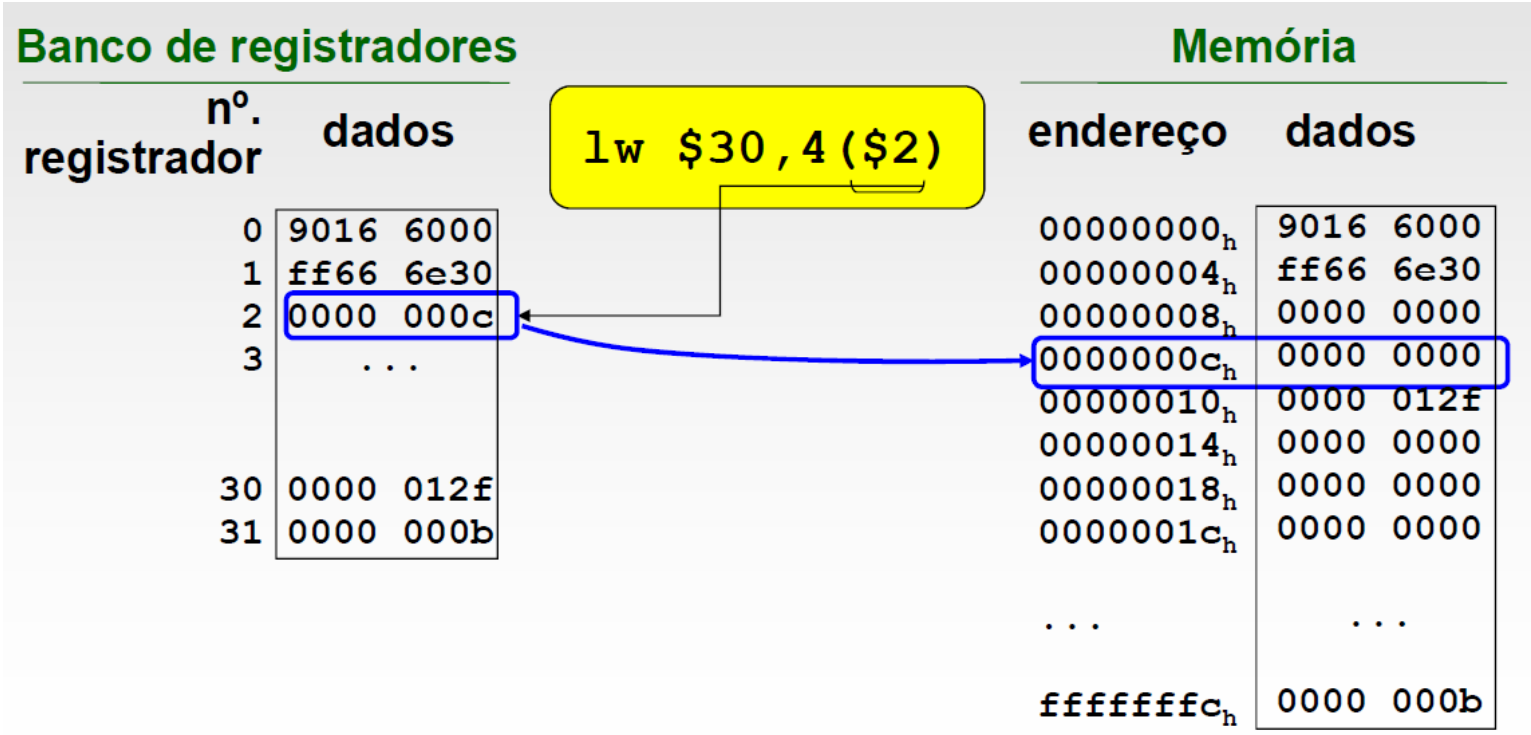
registrador base, que armazena o endereço base

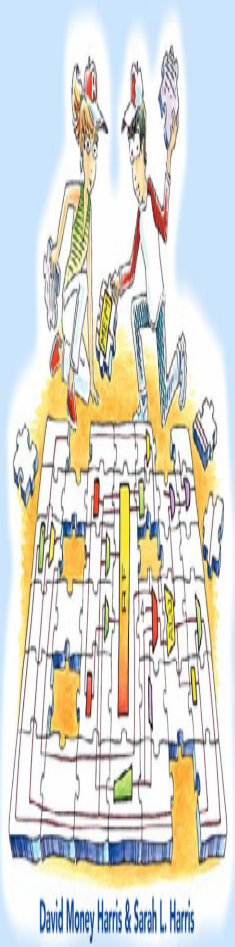


Exemplo - MIPS

Instruções MIPS (Transferência de Dados)

Load word (lw)

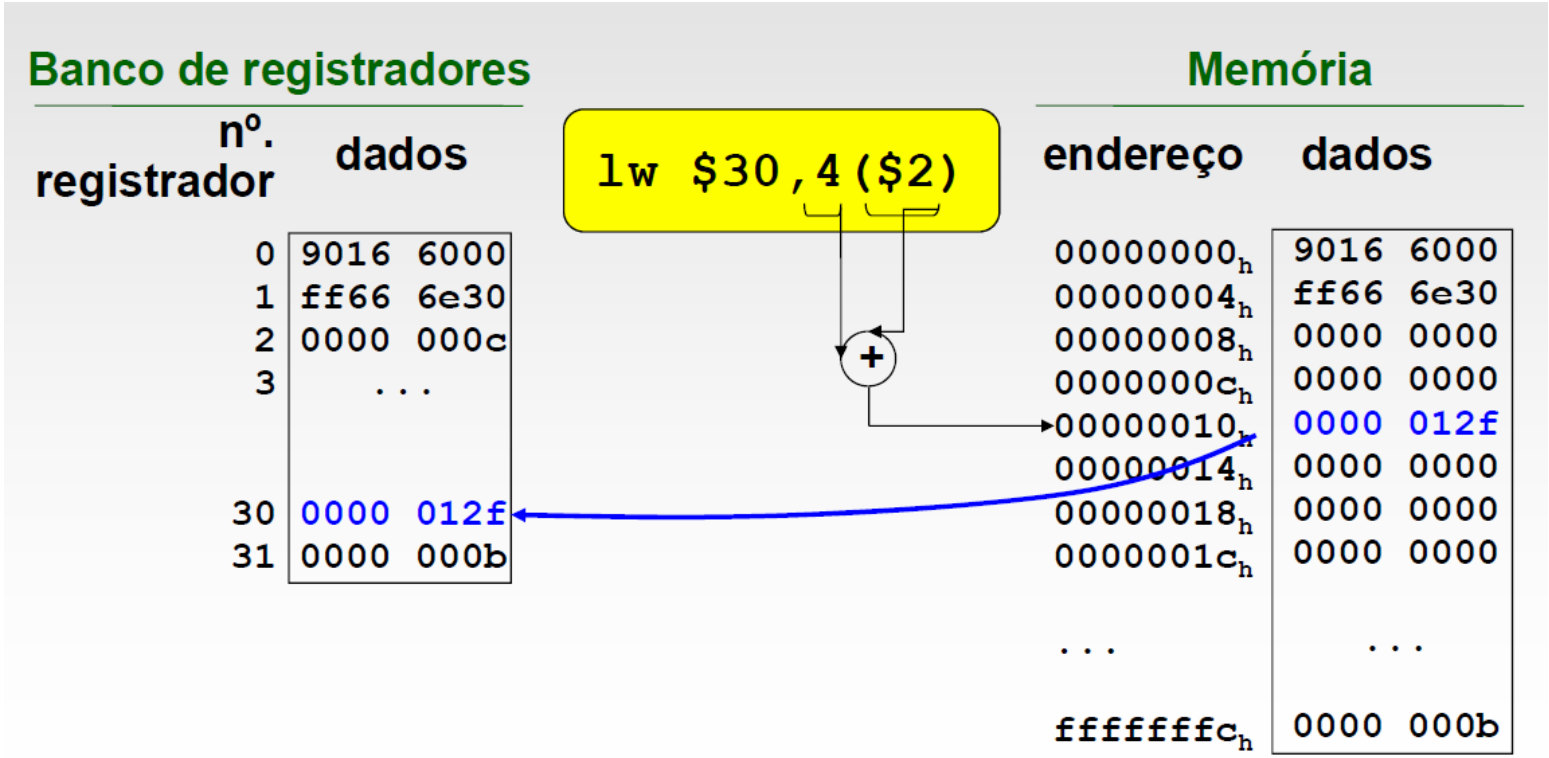


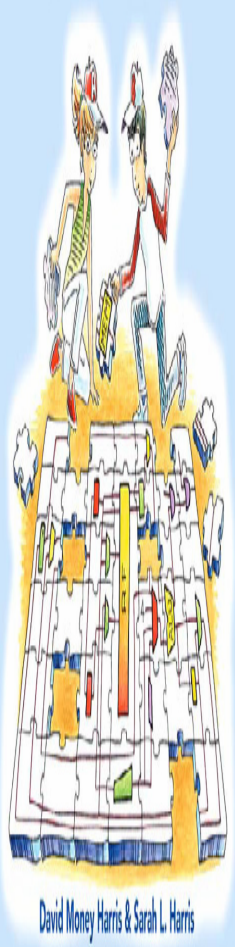


Exemplo - MIPS

Instruções MIPS (Transferência de Dados)

Load word (lw)

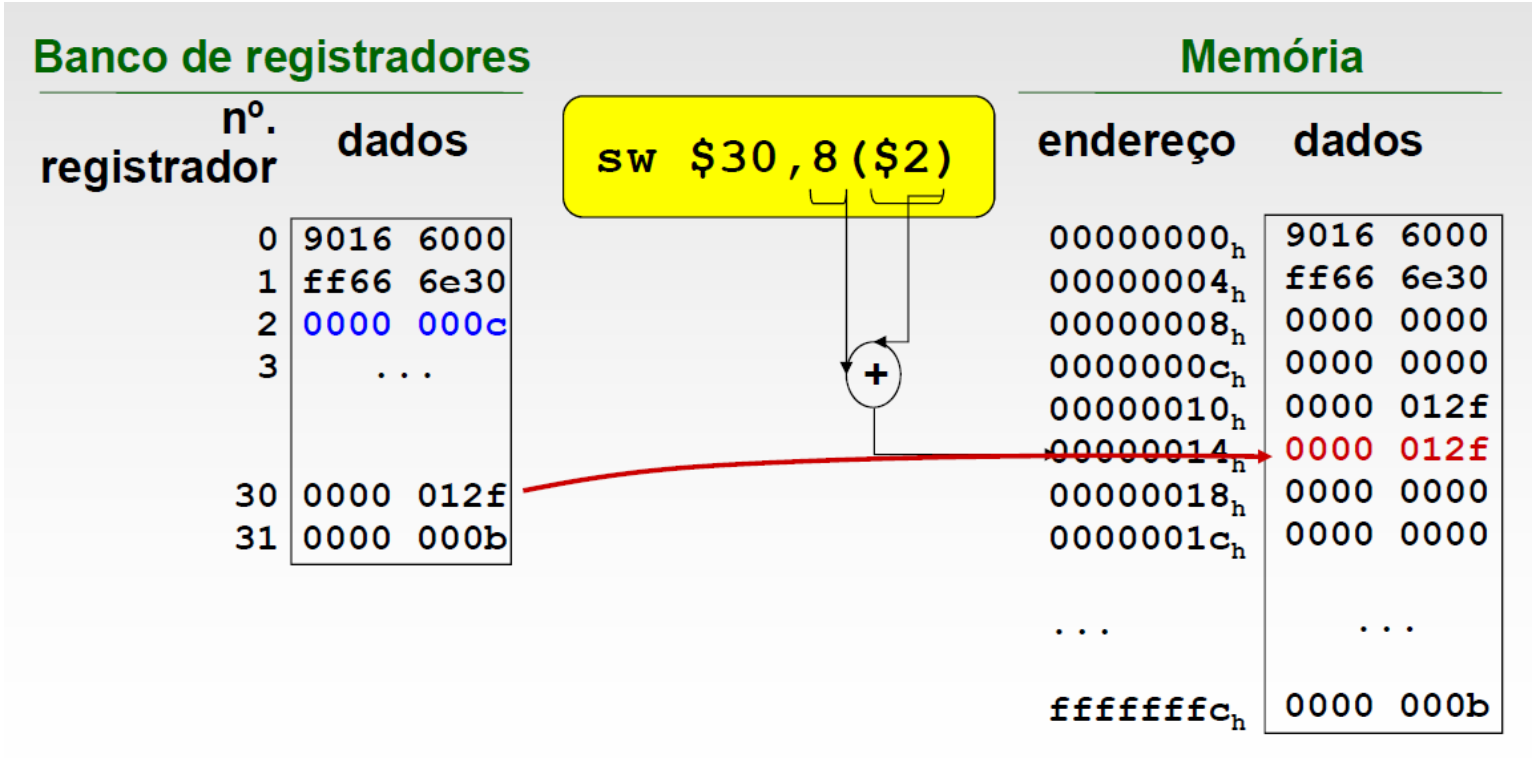


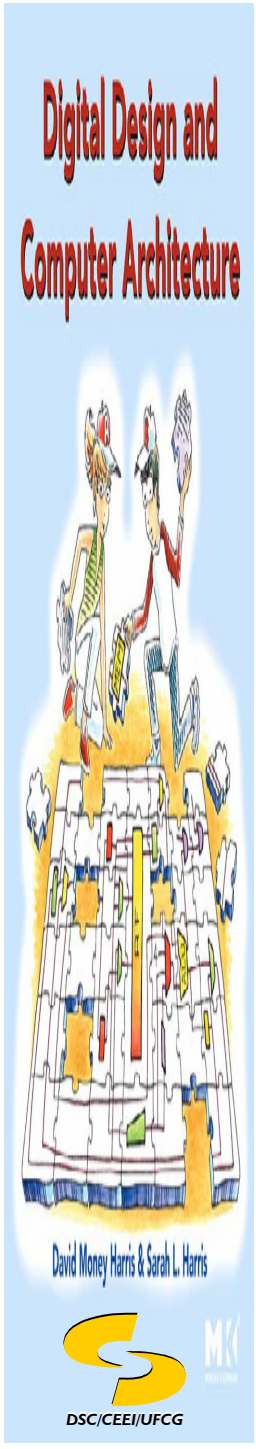


Exemplo - MIPS

Instruções MIPS (Transferência de Dados)

Store word (sw)

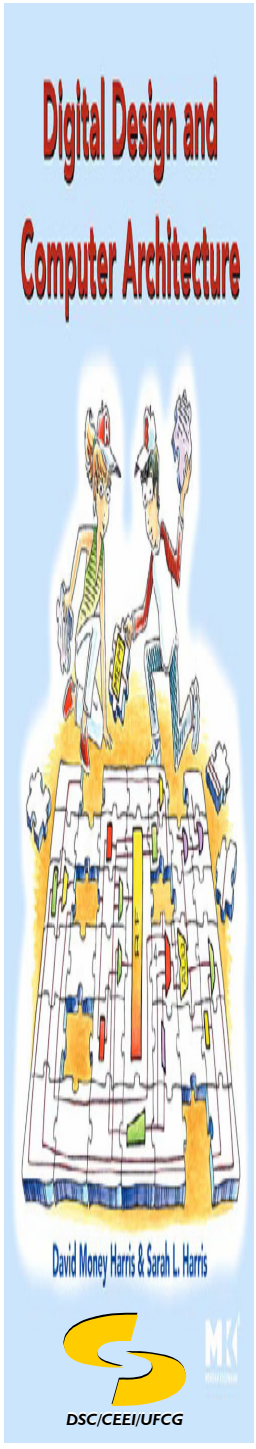




Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

- Cada instrução e cada registrador devem ser mapeados segundo um código e dispostos segundo um dos seguintes formatos:
  - Formato registrador (R)
  - Formato imediato (I)
  - Formato de jump (J)



Exemplo - MIPS

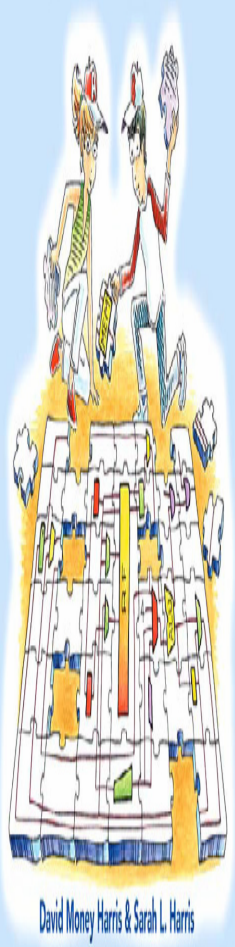
Instruções MIPS (Representação das Instruções)

- **Formato registrador (R)**
  - **Op-code: sempre zero para o formato R**
  - Rs: registrador do primeiro operando de origem
  - Rt: registrador do segundo operando de origem
  - Rd: registrador que recebe o resultado da operação (destino)
  - Shamt: quantidade de deslocamento (shift amount).
  - Function code: especifica qual a operação a ser executada

Op-Code	Rs	Rt	Rd	Shamt	Function Code
000000	sssss	ttttt	ddddd	00000	ffffff

R-type

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits



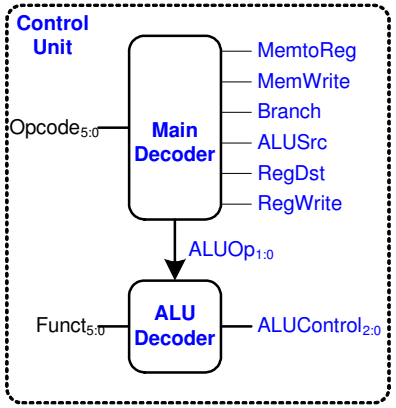
David Money Harris & Sarah L. Harris

Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

- **Formato registrador (R)**
  - **Op-code: sempre zero para o formato R**

Op-Code	Rs	Rt	Rd	Shamt	Function Code
000000	sssss	ttttt	ddddd	00000	ffffff



Instruction	$Op_{5:0}$	RegWrite	RegDst	AluSrc	Branch	MemWrite	MemtoReg	$ALUOp_{1:0}$	Jump
R-type	000000	1	1	0	0	0	0	10	0
lw	100011	1	0	1	0	0	1	00	0
sw	101011	0	X	1	0	1	X	00	0
beq	000100	0	X	0	1	0	X	01	0
addi	001000	1	0	1	0	0	0	00	0
j	000100	0	X	X	X	0	X	XX	1

Decoder

$ALUOp_{1:0}$	Meaning
00	Add
01	Subtract
10	Look at Funct
11	Not Used

$ALUOp_{1:0}$	Funct	$ALUControl_{2:0}$
00	X	010 (Add)
X1	X	110 (Subtract)
1X	100000 (add)	010 (Add)
1X	100010 (sub)	110 (Subtract)
1X	100100 (and)	000 (And)
1X	100101 (or)	001 (Or)
1X	101010 (slt)	111 (SLT)

Exemplo - MIPS

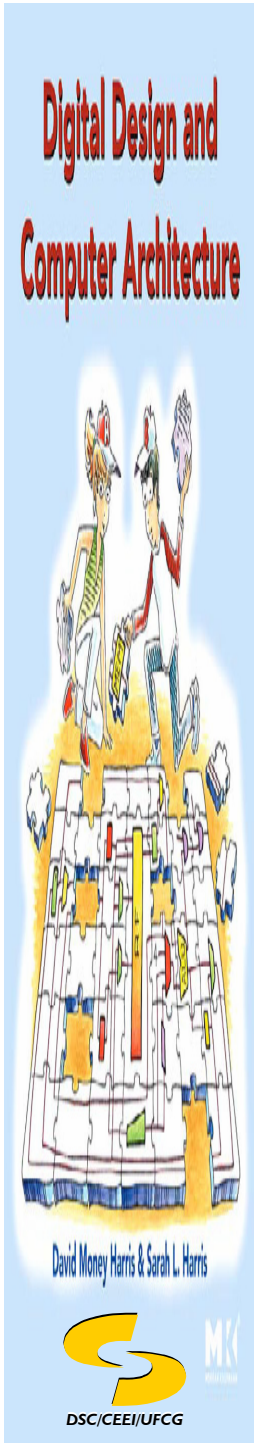
Instruções MIPS (Representação das Instruções)

- **Formato registrador (R)**

Assembly Code	Field Values						Machine Code						
	op	rs	rt	rd	shamt	funct	op	rs	rt	rd	shamt	funct	
`add $s0, $s1, $s2`	0	17	18	16	0	32	000000	10001	10010	10000	00000	100000	(0x02328020)
`sub $t0, $t3, $t5`	0	11	13	8	0	34	000000	01011	01101	01000	00000	100010	(0x016D4022)
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Figure 6.6 Machine code for R-type instructions

Name	Number	Use
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2–3	procedure return values
$a0-$a3	4–7	procedure arguments
$t0-$t7	8–15	temporary variables
$s0-$s7	16–23	saved variables
$t8-$t9	24–25	temporary variables
$k0-$k1	26–27	operating system (OS) temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address



Exemplo - MIPS

Exemplos de Instruções:

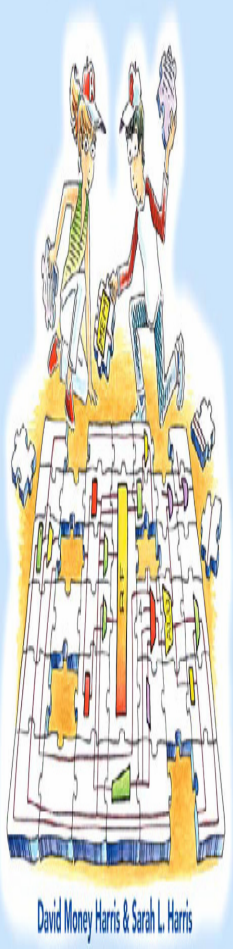
Source Registers

$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code	Result								
`and $s3, $s1, $s2`	$s3	0100	0110	1010	0001	0000	0000	0000	0000
`or  $s4, $s1, $s2`	$s4	1111	1111	1111	1111	1111	0000	1011	0111
`xor $s5, $s1, $s2`	$s5	1011	1001	0101	1110	1111	0000	1011	0111
`nor $s6, $s1, $s2`	$s6	0000	0000	0000	0000	0000	1111	0100	1000

Figure 6.14 Logical operations

Name	Number	Use
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2–3	procedure return values
$a0-$a3	4–7	procedure arguments
$t0-$t7	8–15	temporary variables
$s0-$s7	16–23	saved variables
$t8-$t9	24–25	temporary variables
$k0-$k1	26–27	operating system (OS) temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address

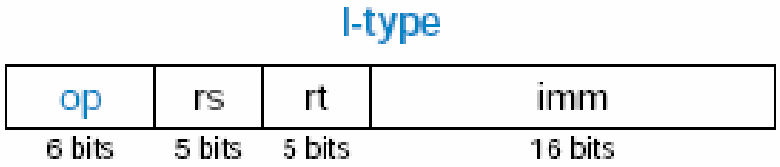


Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

- **Formato imediato (I)**
  - Op-code: especifica qual operação a ser executada
  - Rs: registrador do operando de origem
  - Rt: registrador que recebe o resultado da operação (destino)
  - Immediate: endereço de memória ou constante numérica

Op-Code	Rs	Rt	Immediate
ffffff	sssss	ttttt	iiiiiiiiiiiiiiii



Exemplo - MIPS

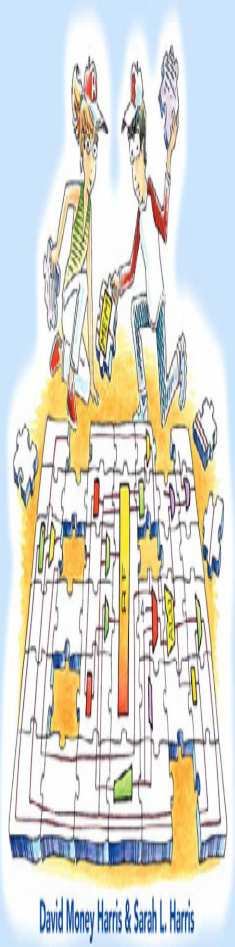
Instruções MIPS (Representação das Instruções)

- **Formato imediato (I)**

Assembly Code	op	rs	rt	imm	op	rs	rt	imm	Machine Code
addi $s0, $s1, 5	8	17	16	5	001000	10001	10000	0000 0000 0000 0101	(0x22300005)
addi $t0, $s3, -12	8	19	8	–12	001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
lw $t2, 32($0)	35	0	10	32	100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
sw $s1, 4($t1)	43	9	17	4	101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
	6 bits	5 bits	5 bits	16 bits	6 bits	5 bits	5 bits	16 bits	

Figure 6.9 Machine code for I-type instructions

Name	Number	Use
$0	0	the constant value 0
$at	1	assembler temporary
$v0-$v1	2–3	procedure return values
$a0-$a3	4–7	procedure arguments
$t0-$t7	8–15	temporary variables
$s0-$s7	16–23	saved variables
$t8-$t9	24–25	temporary variables
$k0-$k1	26–27	operating system (OS) temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	procedure return address



Exemplo - MIPS

Instruções MIPS (Programa Armazenado)

Assembly Code	Machine Code
`lw $t2, 32($0)`	0x8C0A0020
`add $s0, $s1, $s2`	0x02328020
`addi $t0, $s3, -12`	0x2268FFF4
`sub $t0, $t3, $t5`	0x016D4022

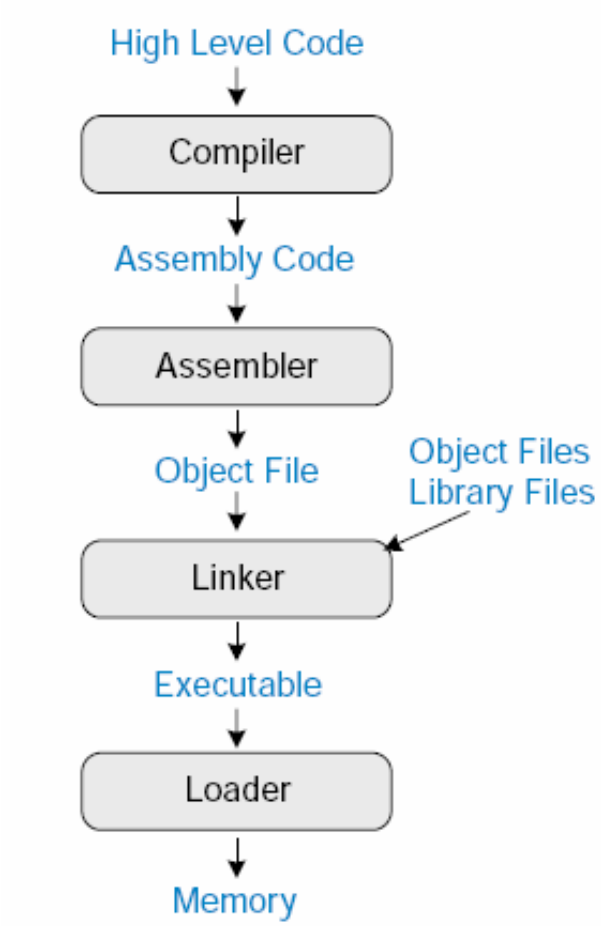
Stored Program

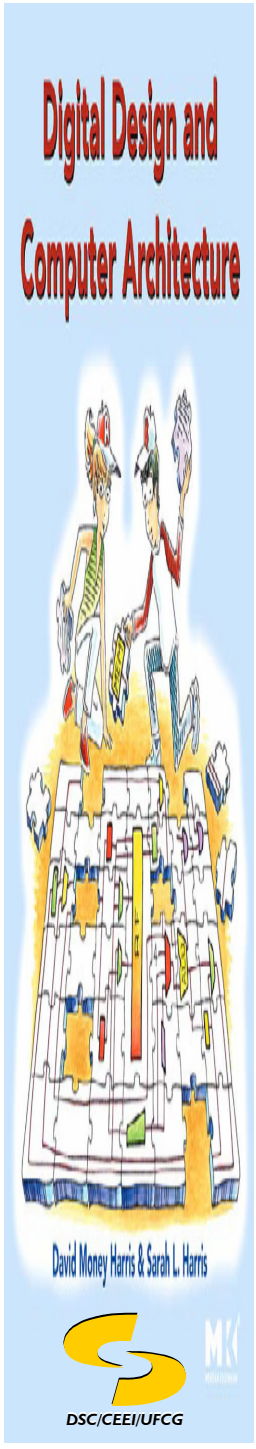
Address	Instructions
0040000C	0 1 6 D 4 0 2 2
00400008	2 2 6 8 F F F 4
00400004	0 2 3 2 8 0 2 0
00400000	8 C 0 A 0 0 2 0 ← PC

Main Memory

Exemplo - MIPS

Passos para execução de um Programa





Exemplo - MIPS

Exemplos de Instruções:

Code Example 6.1 ADDITION

High-Level Code	MIPS Assembly Code
`a = b + c;`	`add a, b, c`

Code Example 6.2 SUBTRACTION

High-Level Code	MIPS Assembly Code
`a = b - c;`	`sub a, b, c`

David Money Harris & Sarah L. Harris

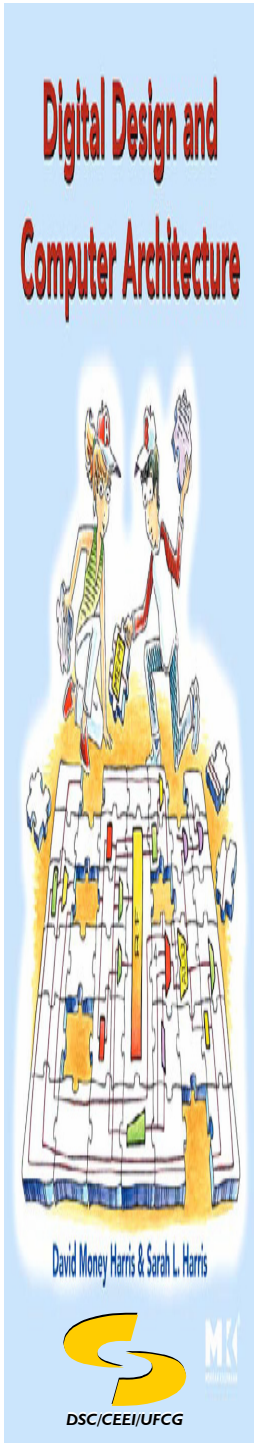
Exemplo - MIPS

Exemplos de Instruções:

Code Example 6.3 MORE COMPLEX CODE

High-Level Code	MIPS Assembly Code

`a = b + c - d;  // single-line comment`
`/* multiple-line`
`comment */`
	`sub t, c, d  # t = c - d`
	`add a, b, t  # a = b + t`



Exemplo - MIPS

Exemplos de Instruções:

Code Example 6.4 REGISTER OPERANDS

High-Level Code	MIPS Assembly Code
`a = b + c;`	`# $s0 = a, $s1 = b, $s2 = c` `add $s0, $s1, $s2  # a = b + c`

Code Example 6.5 TEMPORARY REGISTERS

High-Level Code	MIPS Assembly Code

`a = b + c - d;`	`# $s0 = a, $s1 = b, $s2 = c, $s3 = d`
	`sub $t0, $s2, $s3  # t = c - d`
	`add $s0, $s1, $t0  # a = b + t`

David Money Harris & Sarah L. Harris

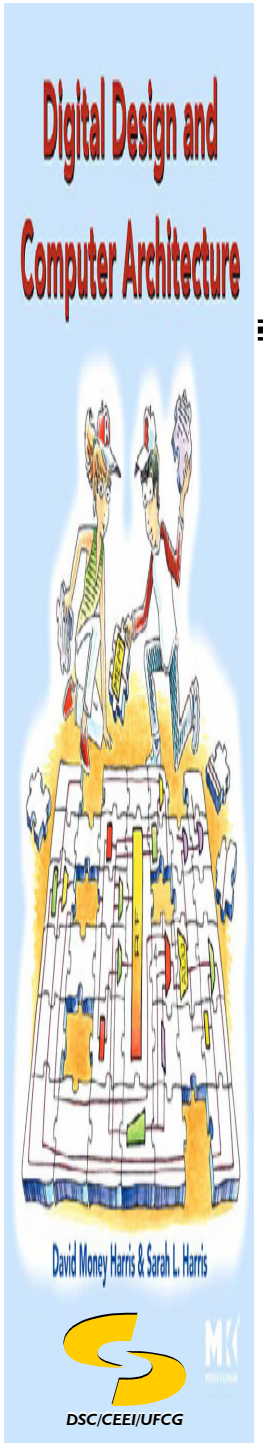
Exemplo - MIPS

Exemplos de Instruções:

Code Example 6.9 IMMEDIATE OPERANDS

High-Level Code	MIPS Assembly Code

`a = a + 4;`
`b = a - 12;`
	`# $s0 = a, $s1 = b`
	`addi $s0, $s0, 4 # a = a + 4`
	`addi $s1, $s0, -12 # b = a - 12`

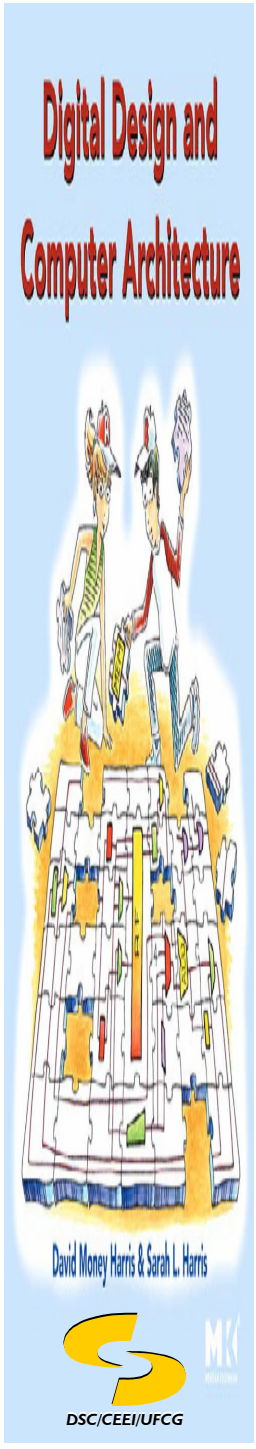


Exemplo - MIPS

Exercício: Traduza o código abaixo, em linguagem de alto nível, para linguagem Assembly.

```
a = b - c;
f = (g + h) - (i + j);
```

As variáveis *a* - *c* estão armazenadas nos registradores *$s0* - *$s2* e as variáveis *f* - *j* estão armazenadas em *$s3* - *$s7*.



Exemplo - MIPS

Solução:

```
a = b - c;
f = (g + h) - (i + j);
```

Solution: The program uses four assembly language instructions.

```
# MIPS assembly code
# $s0 = a, $s1 = b, $s2 = c, $s3 = f, $s4 = g, $s5 = h,
# $s6 = i, $s7 = j
  sub $s0, $s1, $s2   # a = b - c
  add $t0, $s4, $s5   # $t0 = g + h
  add $t1, $s6, $s7   # $t1 = i + j
  sub $s3, $t0, $t1   # f = (g + h) - (i + j)
```

Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

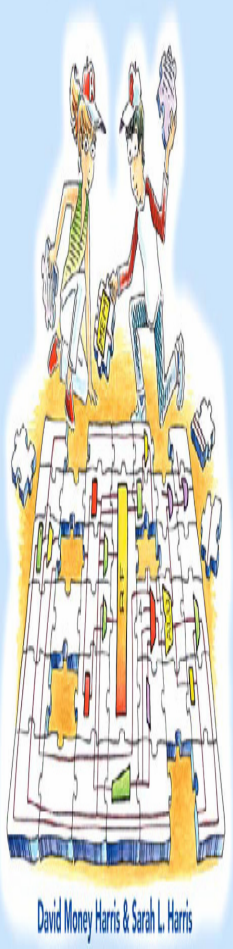
- Instruções para tomada de decisão
  - Alteram o fluxo de controle do programa
  - Alteram a "próxima" instrução a ser executada
- Instruções de controle:
  - Salto condicional
  - Salto incondicional

Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

- **Instruções MIPS para salto condicional:**
  - Branch on equal (beq)
  - Branch on not equal (bne)
  - Set on less than (slt)
  - Set on less than immediate (slti)

- **Instruções MIPS para salto incondicional:**
  - jump (j)



Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

- **Formato jump (J)**
  - Op-code: especifica qual operação a ser executada
  - Target: local da memória a saltar, onde estão as próximas instruções a serem executadas.

Op-Code	Target
00001f	tttttttttttttttttttttttttt

J-type

op	addr
6 bits	26 bits

Exemplo - MIPS

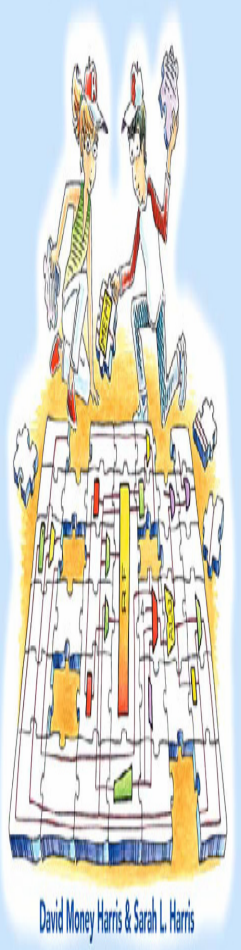
Instruções MIPS (Representação das Instruções)

- Formato jump (J)

Code Example 6.14 UNCONDITIONAL BRANCHING USING j

MIPS Assembly Code

```
  addi   $s0, $0, 4     # $s0 = 4
  addi   $s1, $0, 1     # $s1 = 1
  j      target         # jump to target
  addi   $s1, $s1, 1    # not executed
  sub    $s1, $s1, $s0  # not executed
target:
  add    $s1, $s1, $s0  # $s1 = 1 + 4 = 5
```



Exemplo - MIPS

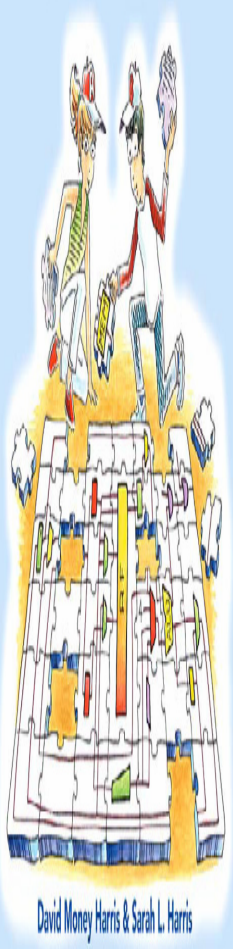
Instruções MIPS (Representação das Instruções)

- Formato jump (J)

Code Example 6.15 UNCONDITIONAL BRANCHING USING `jr`

MIPS Assembly Code

```
0x00002000   addi   $s0, $0, 0x2010  # $s0 = 0x2010
0x00002004   jr     $s0              # jump to 0x00002010
0x00002008   addi   $s1, $0, 1       # not executed
0x0000200c   sra    $s1, $s1, 2      # not executed
0x00002010   lw     $s3, 44 ($s1)    # executed after jr instruction
```



Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

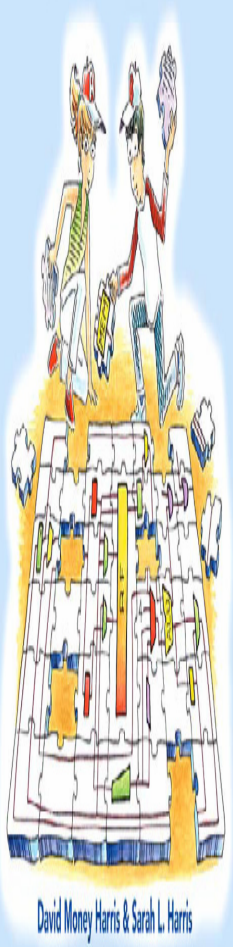
- **Branching**

Code Example 6.12 CONDITIONAL BRANCHING USING `beq`

MIPS Assembly Code

```
  addi  $s0, $0, 4        # $s0 = 0 + 4 = 4
  addi  $s1, $0, 1        # $s1 = 0 + 1 = 1
  sll   $s1, $s1, 2       # $s1 = 1 << 2 = 4
  beq   $s0, $s1, target  # $s0 == $s1, so branch is taken
  addi  $s1, $s1, 1       # not executed
  sub   $s1, $s1, $s0     # not executed

target:
  add   $s1, $s1, $s0     # $s1 = 4 + 4 = 8
```



Exemplo - MIPS

Code Example 6.16 `if` STATEMENT

High-Level Code

```
if (i == j)
  f = g + h;

f = f - i;
```

MIPS Assembly Code

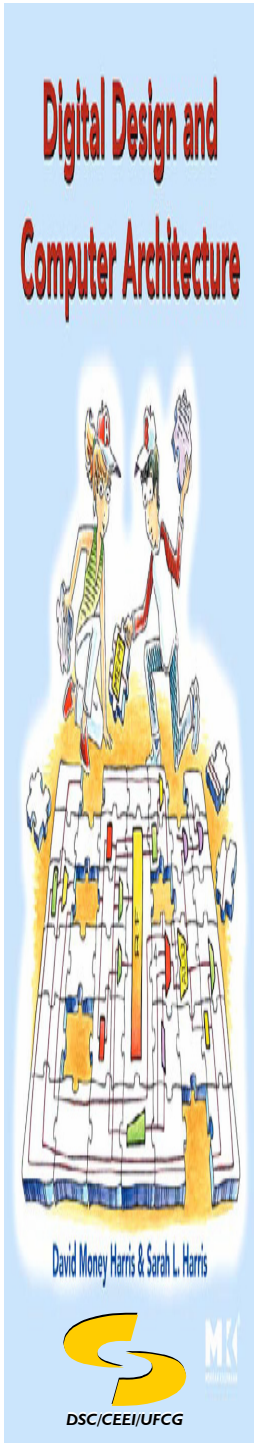
```
# $s0 = f, $s1 = g, $s2 = h, $s3 = i, $s4 = j
  bne $s3, $s4, L1     # if i ! = j, skip if block
  add $s0, $s1, $s2    # if block: f = g + h
L1:
  sub $s0, $s0, $s3    # f = f - i
```

Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

Suporte a Procedimentos

- Procedimentos: Conjunto de instruções com função definida
  - Realizam uma série de operações como base em valores de parâmetros
  - Podem retornar valores computados

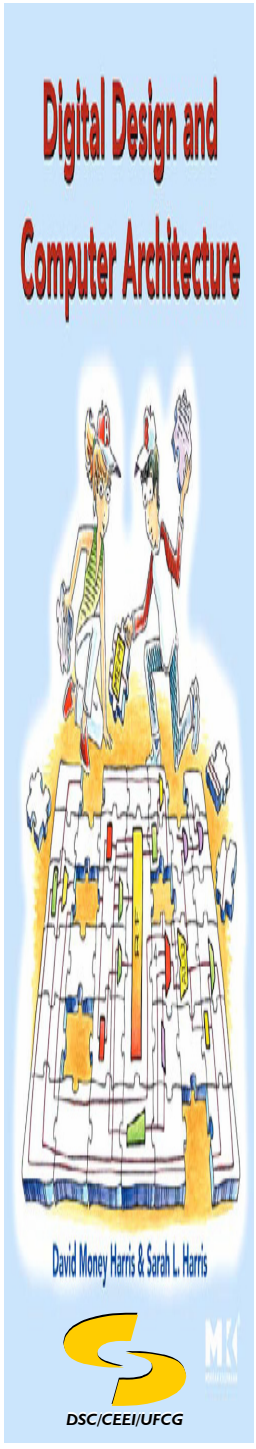


Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

Suporte a Procedimentos

- **Motivos para o uso de procedimentos:**
  - Tornar o programa mais fácil de ser entendido
  - Permitir a reutilização do código do procedimento
  - Permitir que o programador se concentre em uma parte do código (os parâmetros funcionam como barreira)



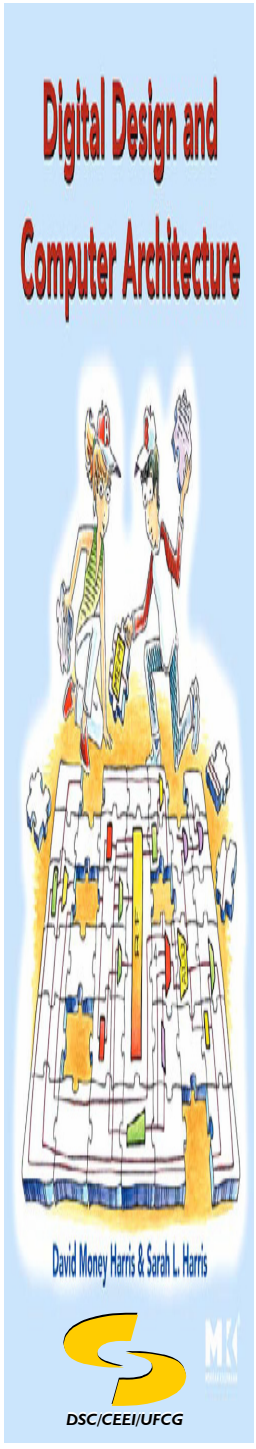
Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

Suporte a Procedimentos

Passos para a execução:

1. O programa coloca os parâmetros em um lugar onde o procedimento chamado possa acessá-los;
2. O programa transfere o controle para o procedimento
3. O procedimento acessa os valores necessários à realização de sua tarefa;
4. O procedimento executa sua tarefa, gerando valores;
5. O procedimento (chamado) coloca os valores gerados em um lugar onde o programa (chamador) pode acessá-los;
6. O procedimento transfere o controle de volta para o ponto do programa que o chamou.

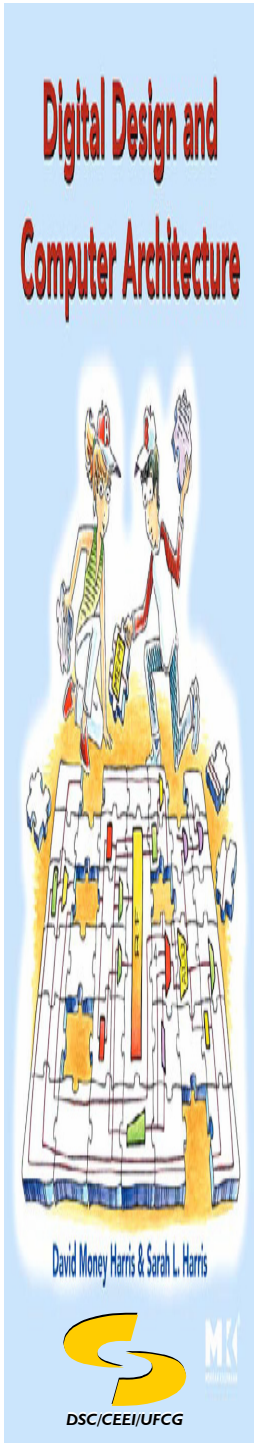


Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

Suporte a Procedimentos

- MIPS aloca alguns registradores para implementar o uso de procedimentos:
  - $a0-$a3
    - Quatro registradores usados como argumentos para passagem de parâmetros
  - $v0-$v1
    - Dois registradores usados para retornar valores
  - $ra
    - Registrador que guarda o endereço de retorno (*return address*) para o ponto do programa que chamou o procedimento



Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

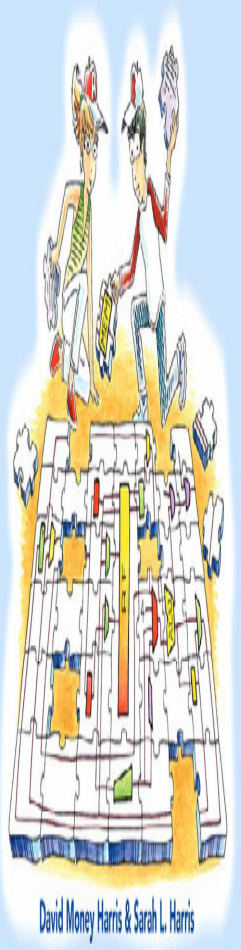
Suporte a Procedimentos

- **Jump and link (jal)**
  - Salta para o endereço especificado, salvando o endereço da próxima instrução em **$ra**

```
jal   label  #desvia para o endereço indicado
             #por label.  $ra ← PC + 4
```

- **Jump register (jr)**
  - Desvio incondicional para endereço guardado em **$ra**

```
jr  $ra      #desvia para o endereço da
             #memória guardado em $ra
```

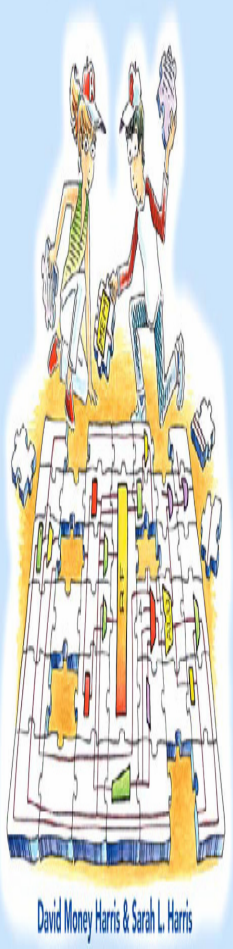


Exemplo - MIPS

Instruções MIPS (Representação das Instruções)

Suporte a Procedimentos

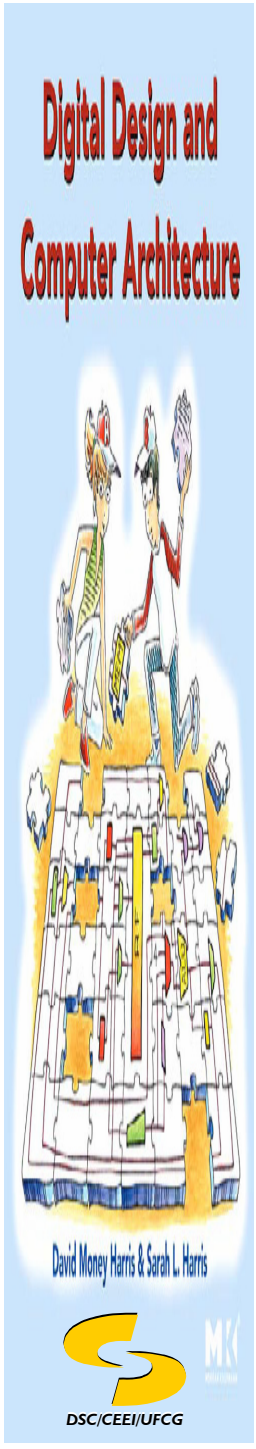
- **Passos no MIPS para execução de procedimentos**
  1. Chamador coloca os valores dos parâmetros em $a0-$a3
  2. Chamador chama **jal X** para saltar ao procedimento X (chamado)
  3. Chamado realiza suas tarefas
  4. Chamado coloca os resultados em $v0-$v1
  5. Chamado retorna o controle ao chamador usando jr $ra



Exemplo - MIPS

Suporte a Procedimentos

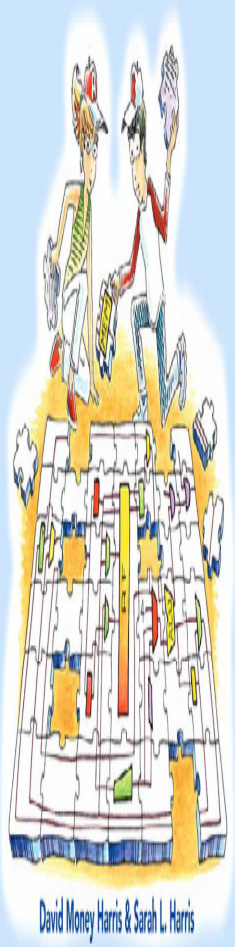
- **Preservação de Contexto**
  - Qualquer registrador usado pelo chamador deve ter seu conteúdo restaurado para o valor que tinha antes da chamada
  - Conteúdo dos registradores é salvo na memória. Depois da execução do procedimento, estes registradores devem ter seus valores restaurados
  - Se as chamadas forem recursivas, é conveniente o uso de uma pilha



Exemplo - MIPS

Suporte a Procedimentos

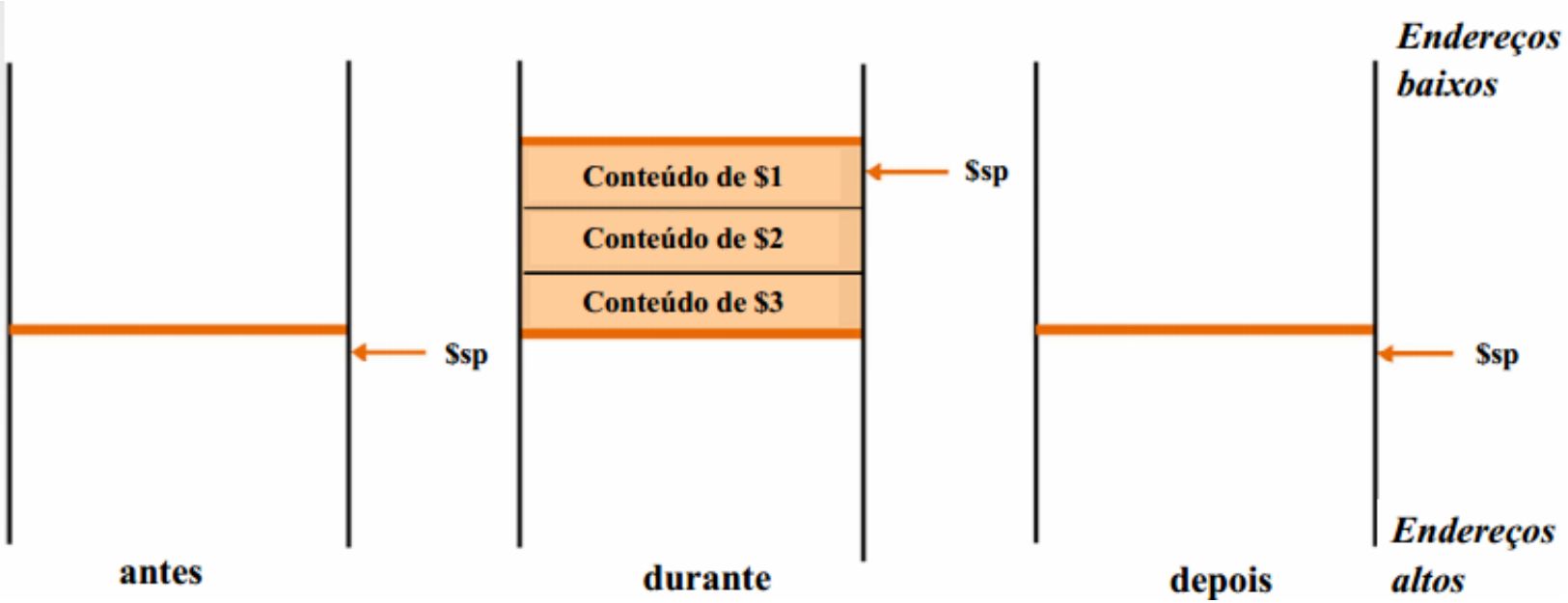
- **Apontador de Pilha (Stack Pointer), $sp**
  - Registrador usado para guardar o endereço do topo da pilha da chamada de procedimentos
- **Indica:**
  - a posição de memória que contêm os valores dos registradores salvos na memória pela última chamada
  - a posição a partir da qual a próxima chamada de procedimento pode salvar seus registradores
- **A pilha cresce do endereço mais alto para o mais baixo**

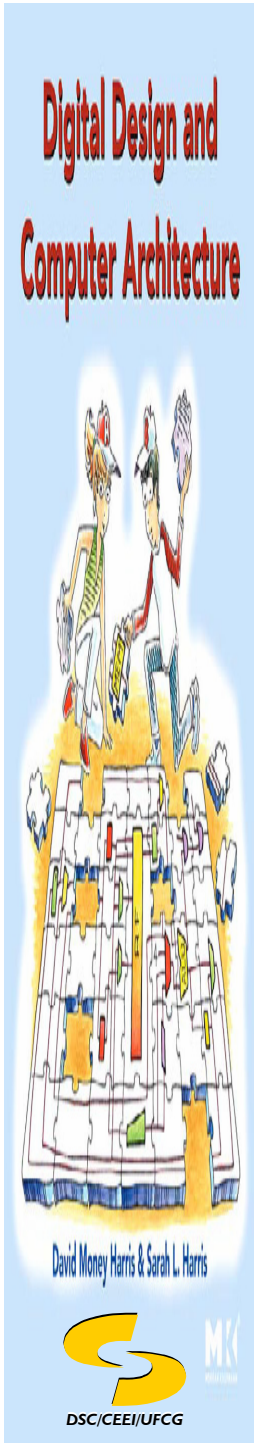


Exemplo - MIPS

Suporte a Procedimentos

- Valor do stack pointer **($sp)** em momentos diferentes da chamada de procedimento:



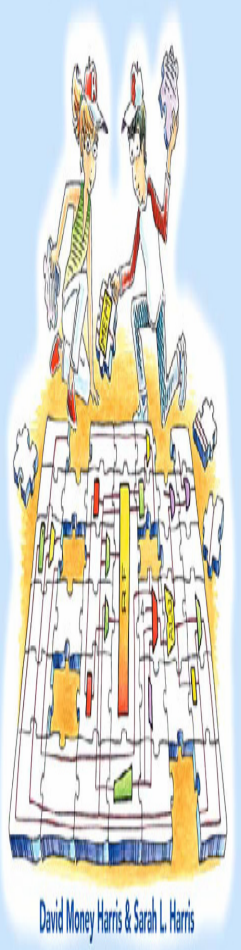


Exemplo - MIPS

Suporte a Procedimentos – Pilha – Exemplo

```
int leaf_example(int g, int h, int i, int j)
{
    int f;
    f = (g + h) - (i + j);
    return f;
}
```

- $a0 corresponde a g , $a1 corresponde a h
- $a2 corresponde a i , $a3 corresponde a j
- $s0 corresponde a f
- Variável local (deve ser salva, pois será modificada pelo procedimento)
- Valor de retorno deve ser colocado em $v0



Exemplo - MIPS

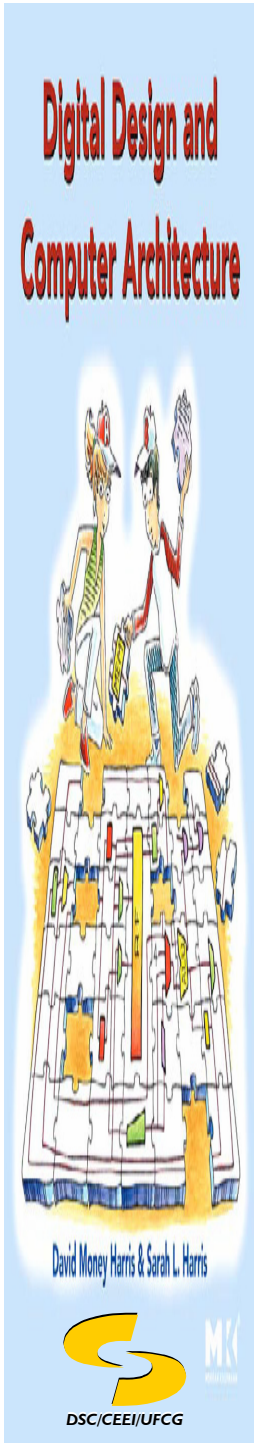
Suporte a Procedimentos – Pilha – Exemplo

```
leaf_example:                # label para chamada
   addi  $sp, $sp, -4        # avança o stack pointer
   sw    $s0, 0($sp)         # empilha o valor de $s0

   add   $t0, $a0, $a1       # $t0 ← g + h
   add   $t1, $a2, $a3       # $t1 ← i + j
   sub   $s0, $t0, $t1       # f ← $t0 - $t1

   add   $v0, $s0, $zero     # coloca resultado em $v0

   lw    $s0, 0($sp)         # restaura $s0
   addi  $sp,$sp,4           # desempilha o topo
   jr    $ra                 # volta para o chamador
```



Exemplo - MIPS

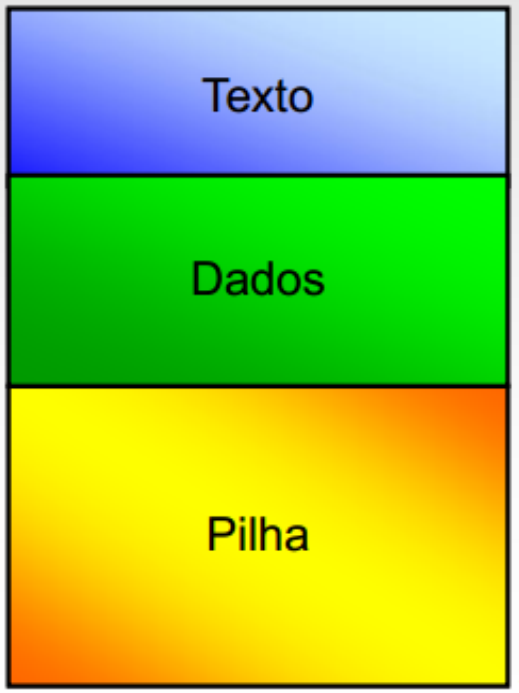
Suporte a Procedimentos - Pilha

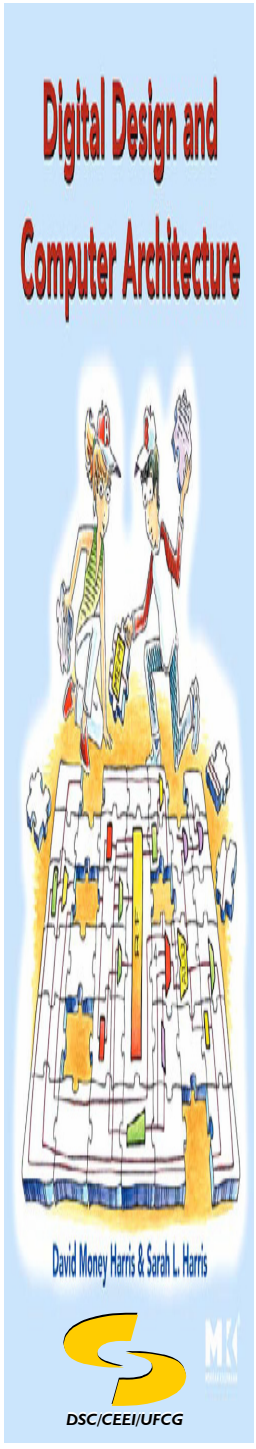
Para todo programa, o sistema operacional aloca três segmentos de memória:

Segmento de texto: armazena o código de máquina

Segmento de dados: alocado para constantes e variáveis globais

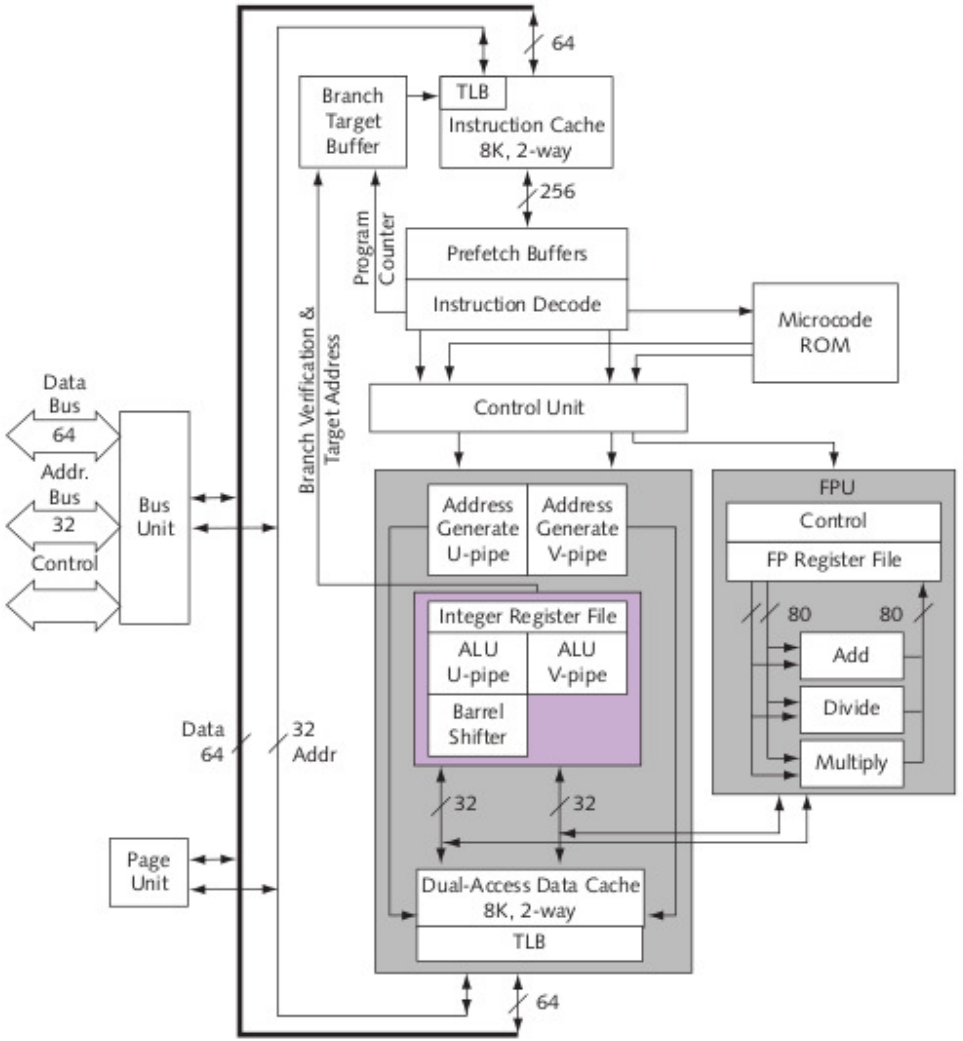
Segmento de pilha: local onde são passados parâmetros, alocado espaço para variáveis locais e armazenados endereços de retorno para chamadas de funções aninhadas/recursivas

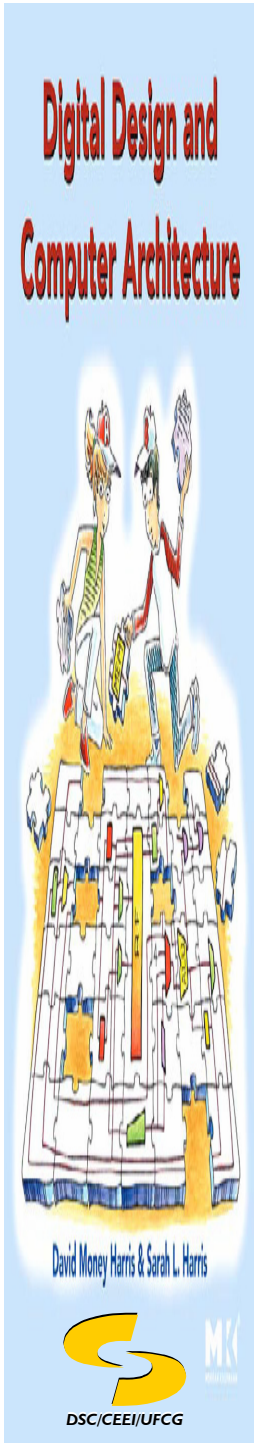




Organização e Arquitetura Básicas de Computadores

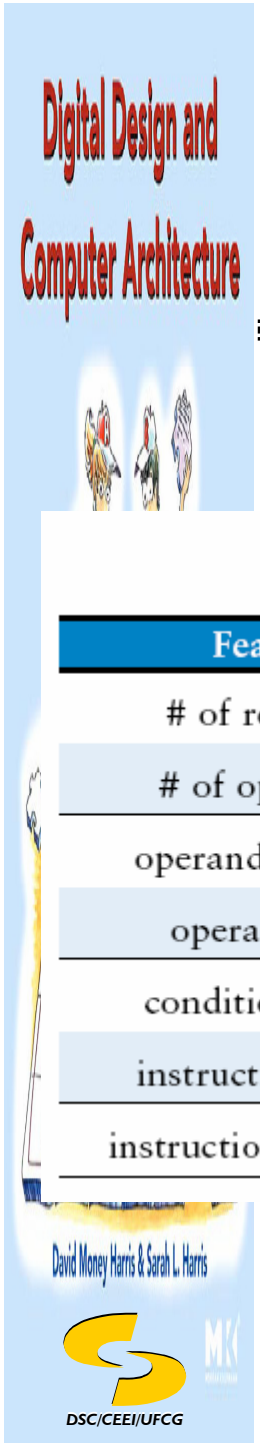
Arquitetura de um computador com processador Intel





Arquitetura IA-32

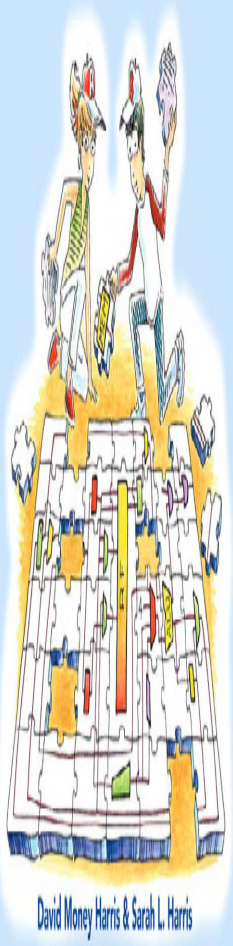
- **Tipos de instruções típicas:**
  - LOAD, STORE - realizam o movimento de dados e instruções entre memória e registradores
  - MOVE - realizam cópia de valores entre registros
  - ADD, SUB, MULT,... - realizam operações aritméticas
  - AND, OR, XOR, ... - realizam operações lógicas
  - EQ, NEQ, LEQ,... - realizam operações de comparação
  - GOTO - operação de desvio



Arquitetura MIPS X IA-32

Table 6.9 Major differences between MIPS and IA-32

Feature	MIPS	IA-32
# of registers	32 general purpose	8, some restrictions on purpose
# of operands	3 (2 source, 1 destination)	2 (1 source, 1 source/destination)
operand location	registers or immediates	registers, immediates, or memory
operand size	32 bits	8, 16, or 32 bits
condition codes	no	yes
instruction types	simple	simple and complicated
instruction encoding	fixed, 4 bytes	variable, 1–15 bytes

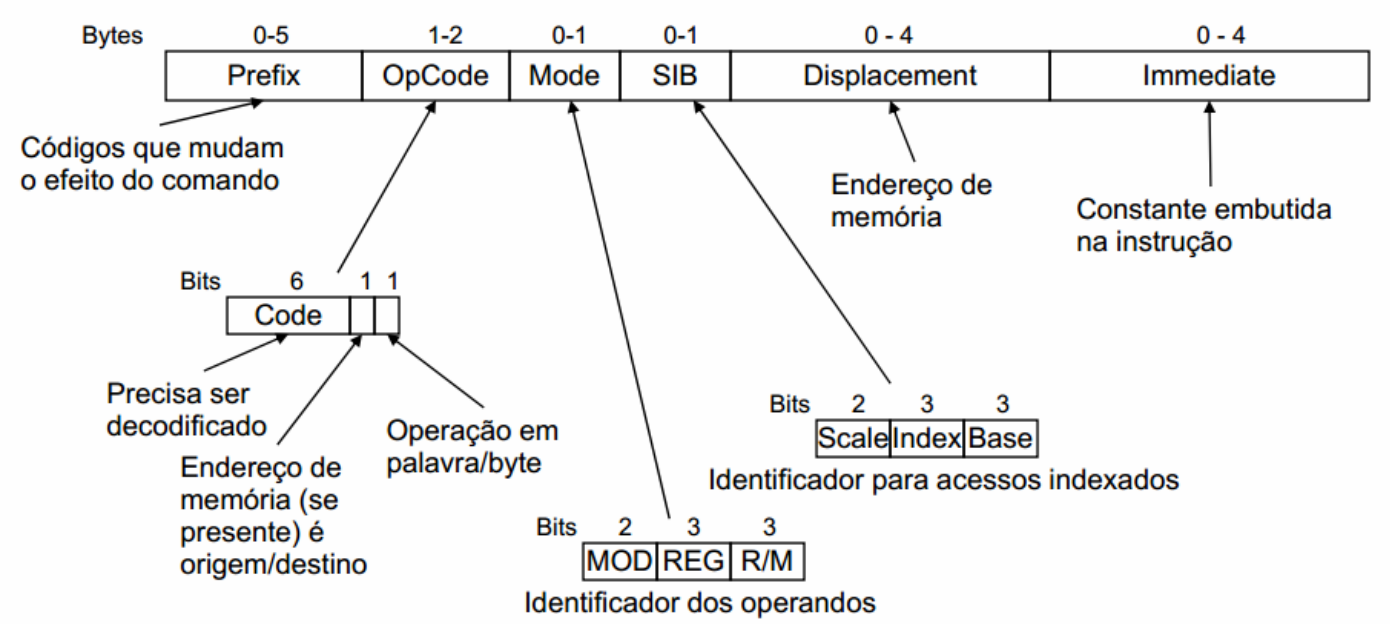


Arquitetura IA-32

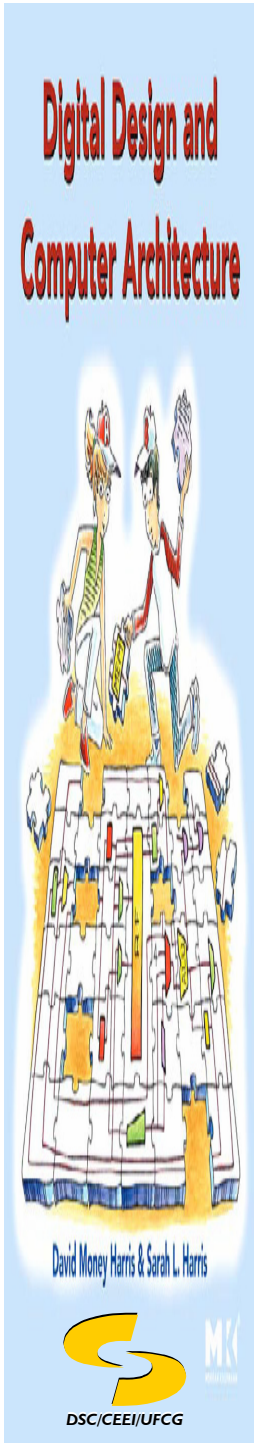
Table 6.10 Operand locations

Source/ Destination	Source	Example	Meaning
register	register	`add EAX, EBX`	`EAX <- EAX + EBX`
register	immediate	`add EAX, 42`	`EAX <- EAX + 42`
register	memory	`add EAX, [20]`	`EAX <- EAX + Mem[20]`
memory	register	`add [20], EAX`	`Mem[20] <- Mem[20] + EAX`
memory	immediate	`add [20], 42`	`Mem[20] <- Mem[20] + 42`

Arquitetura IA-32



Uma enorme fração de todos os transistores do Pentium 4 é dedicada a decompor instruções CISC, distinguir o que pode ser feito em paralelo, resolver conflitos, fazer previsões, sanar as consequências de previsões incorretas e outros controles, sobrando uma quantidade surpreendentemente pequena deles para executar o trabalho real que o usuário solicitou”

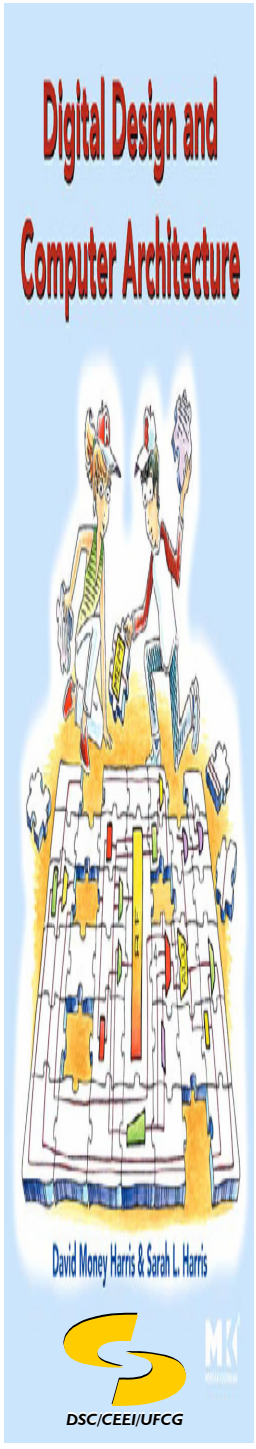


Arquitetura IA-64

- **Arquitetura IA-64 (Intel + HP)**
  - Máquina completa de 64 bits: ruptura em relação ao Pentium
  - Máquina RISC de última geração, com paralelismo, do tipo carregue/armazene, e três endereços
  - Primeira implementação: série Itanium (Itanium 2)

Arquitetura IA-64

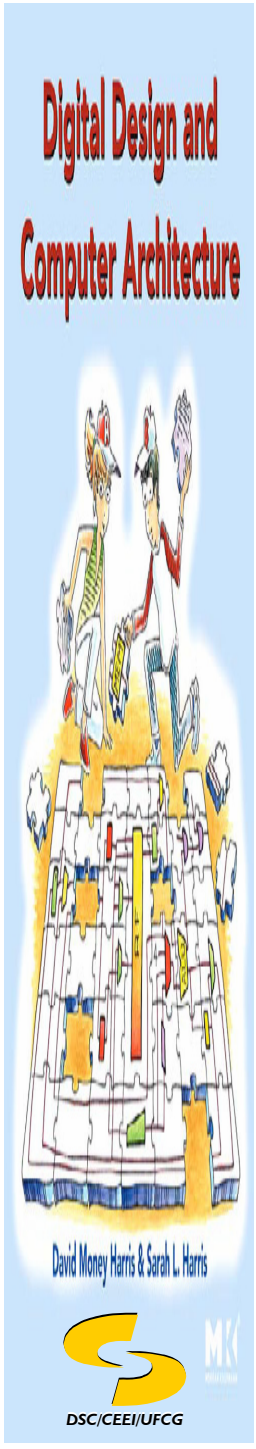
- **Computação por instrução explicitamente paralela**
- **Trabalho de reordenar instruções, alocar registradores, escalonar unidades funcionais é passado todo para o compilador**
- **Redução de referências à memória**
  - 128 registradores de uso geral de 64 bits
  - 128 registradores de ponto flutuante
  - 64 registradores de bits (predicação)
  - Janelas de registradores com tamanho variável em função da necessidade dos procedimentos
    - Cada procedimento tem acesso a 32 registradores estáticos + um número variável de registradores de alocação dinâmica
  - Vários outros registradores



Nível ISA

Importante:

- Instruções podem ter tamanhos diversos (complica o projeto mas tem-se economia de memória) ou
- Instruções podem ser todas de tamanhos iguais (simplifica o projeto, mas desperdiça espaço. **Por que?**)



Nível ISA

Importante:

1. O tamanho ideal de uma instrução deve considerar, além do preço da memória, o tempo de decodificação e de execução de uma instrução.
2. Como os processadores modernos são capazes de executar várias instruções no mesmo ciclo de clock, torna-se imperativo um mecanismo de busca de várias instruções em cada ciclo de clock (memórias cache).
3. Quando uma instrução tem endereços, o tamanho do endereço deve ser compatível com o tamanho máximo da memória do computador. Mas, memórias maiores requerem endereços mais longos resultando em instruções maiores.