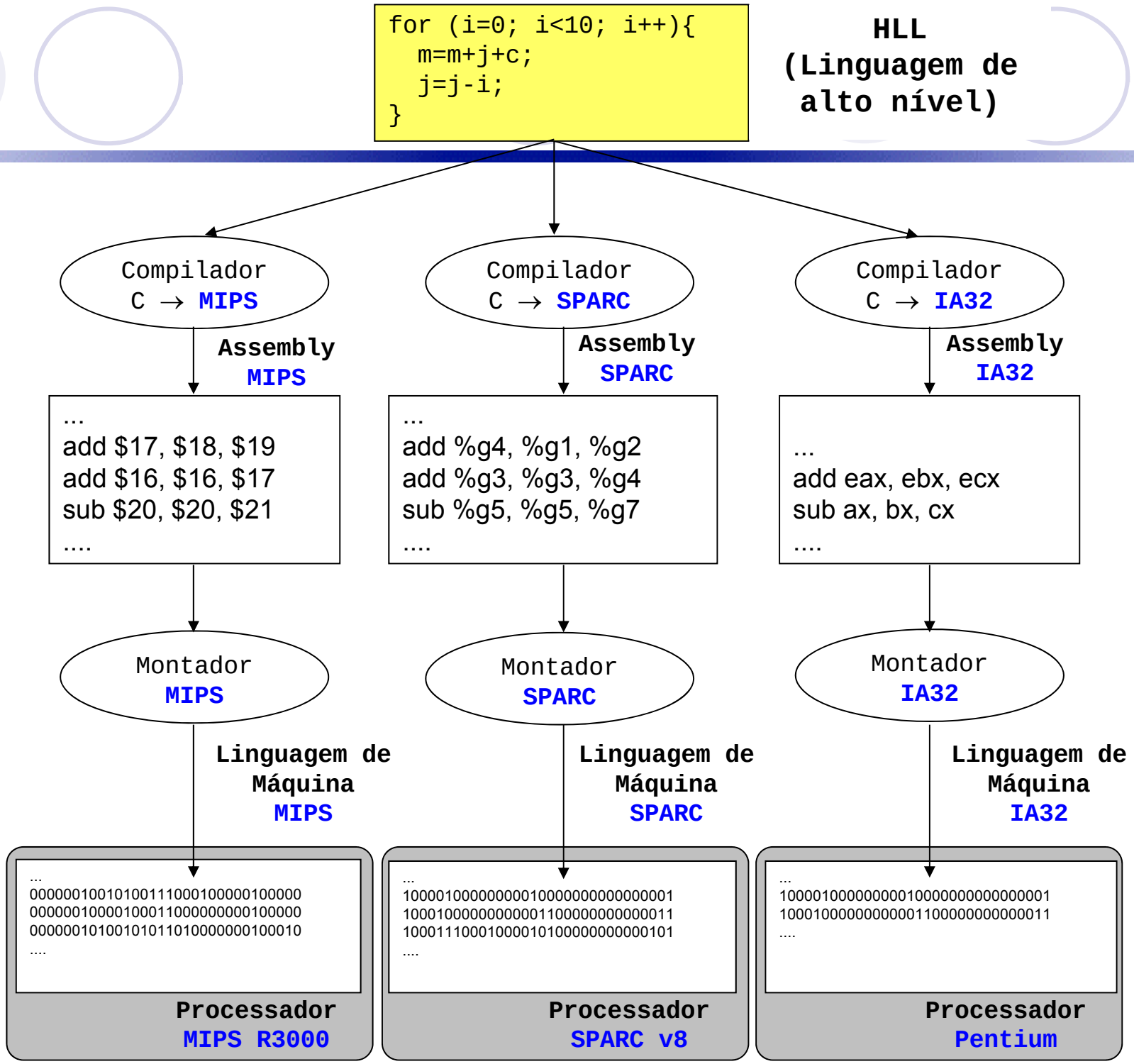
Sistemas Processadores e Periféricos
Aula 1 - Revisão

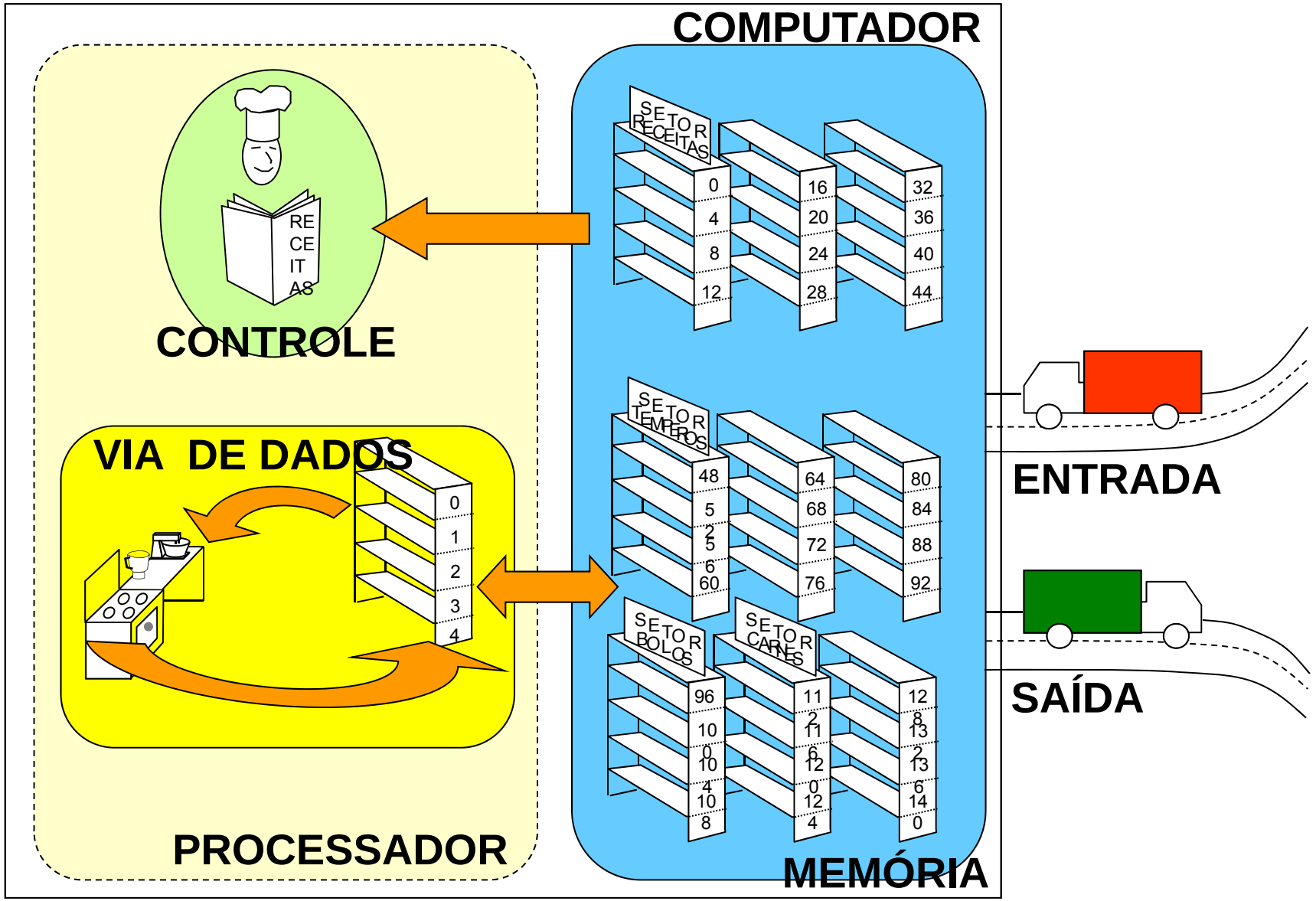
Prof. Frank Sill Torres
DELT – Escola de Engenharia
UFMG

Adaptado a partir dos Slides de Organização de Computadores 2006/02 do professor

Leandro Galvão DCC/UFAM - galvao@dcc.ufam.edu.br pelo Prof. Ricardo de Oliveira Duarte


for (i=0; i<10; i++){
m=m+j+c;
j=j-i;
}
HLL
(Linguagem de
alto nível)
Compilador
C → MIPS
Compilador
C → SPARC
Compilador
C → IA32
Assembly
MIPS
Assembly
SPARC
Assembly
IA32
...
add $17, $18, $19
add $16, $16, $17
sub $20, $20, $21
....
...
add %g4, %g1, %g2
add %g3, %g3, %g4
sub %g5, %g5, %g7
....
...
add eax, ebx, ecx
sub ax, bx, cx
....
Montador
MIPS
Montador
SPARC
Montador
IA32
Linguagem de
Máquina
MIPS
Linguagem de
Máquina
SPARC
Linguagem de
Máquina
IA32
...
00000010010100111000100000100000
00000010000100011000000000100000
00000010100101011010000000100010
....
...
10000100000000001000000000000001
10001000000000001100000000000011
10001110001000010100000000000101
....
...
10000100000000001000000000000001
10001000000000001100000000000011
....
Processador
MIPS R3000
Processador
SPARC v8
Processador
Pentium

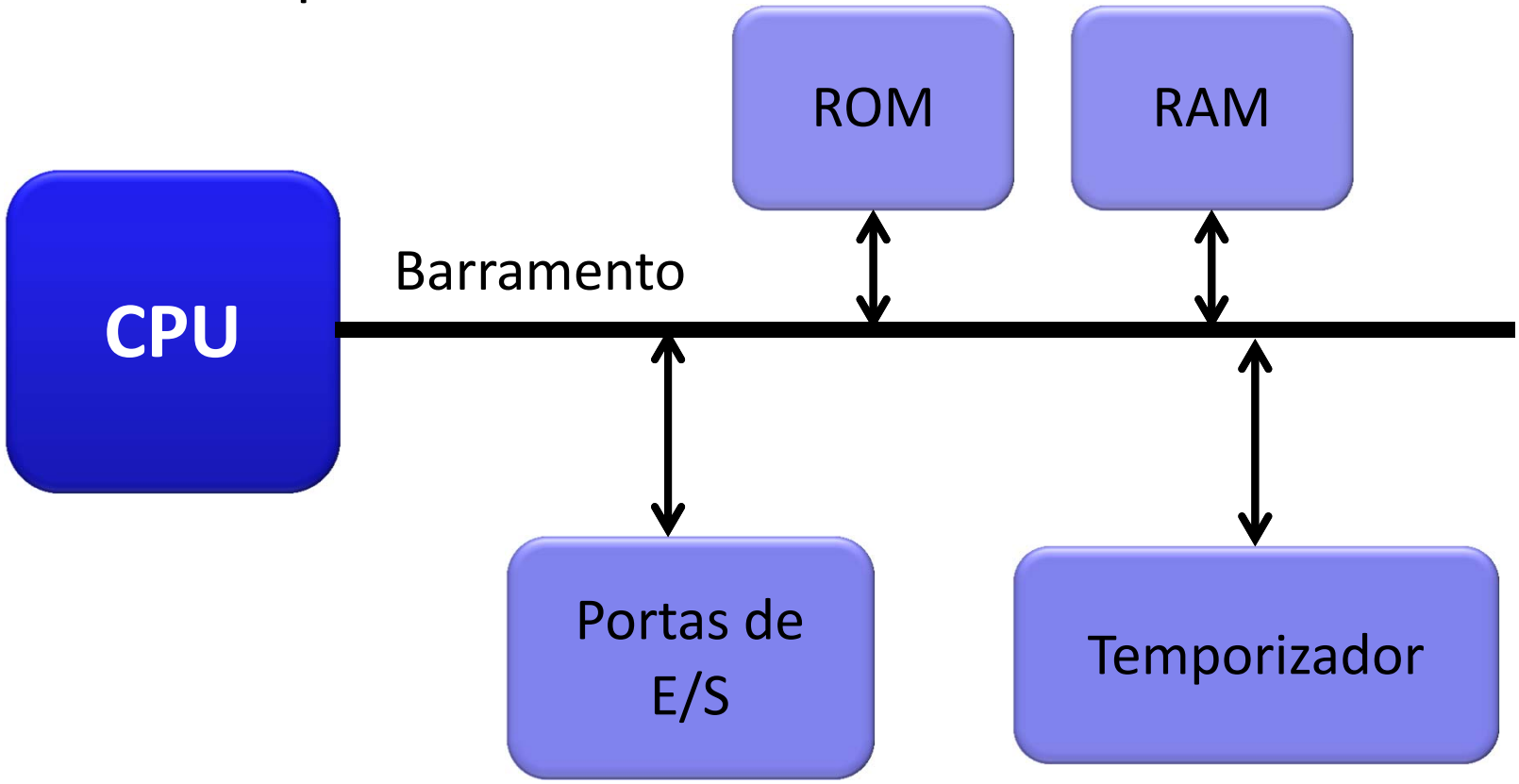
Revisão - Organização de um Computador



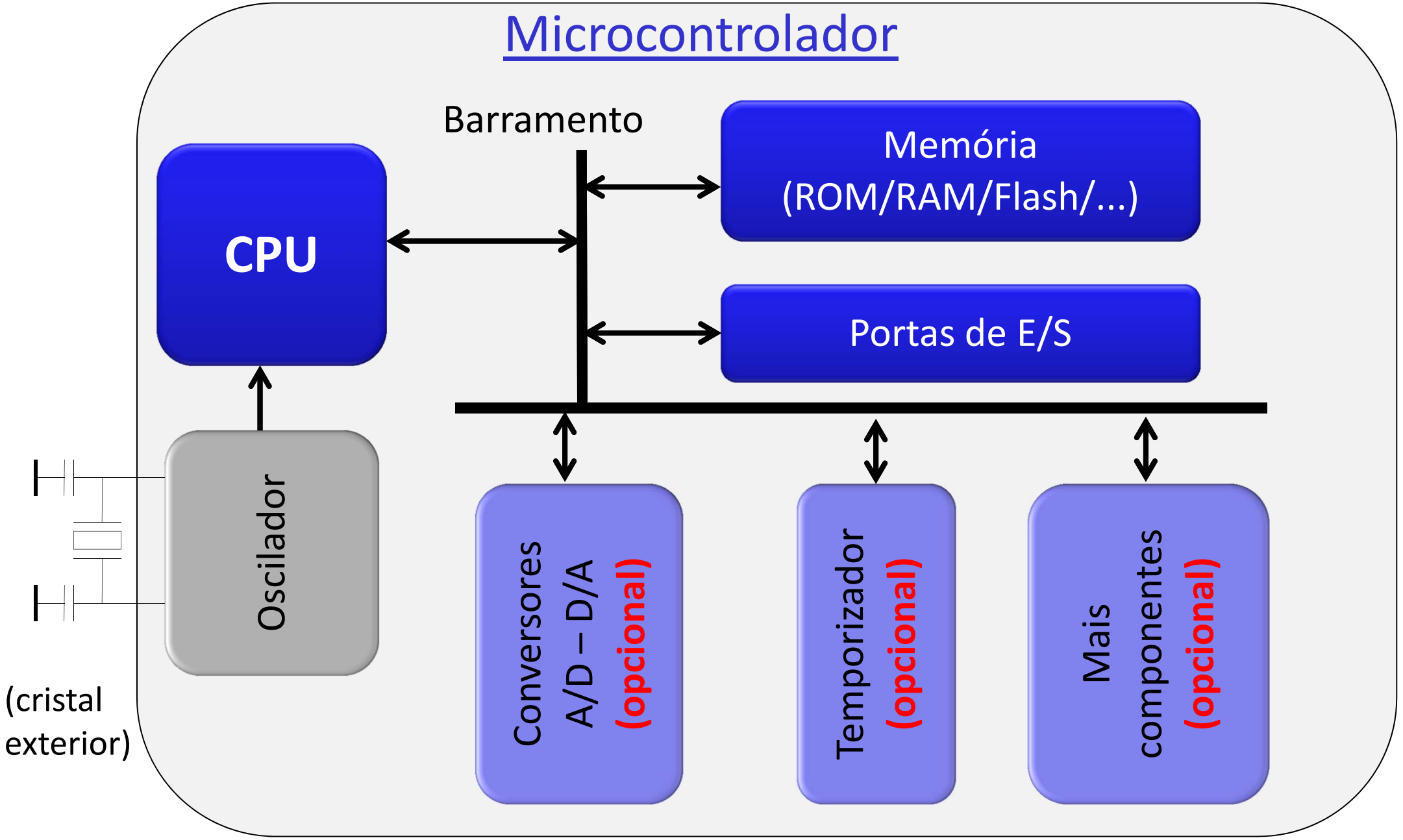
Revisão - Características de Microprocessadores

- Outros periféricos necessários para funcionamento (como memória RAM/ROM, Temporizadores, ...)

→ conectados pelo barramento



Revisão - Arquitetura de um Microcontrolador



Revisão - CPU vs. MCU

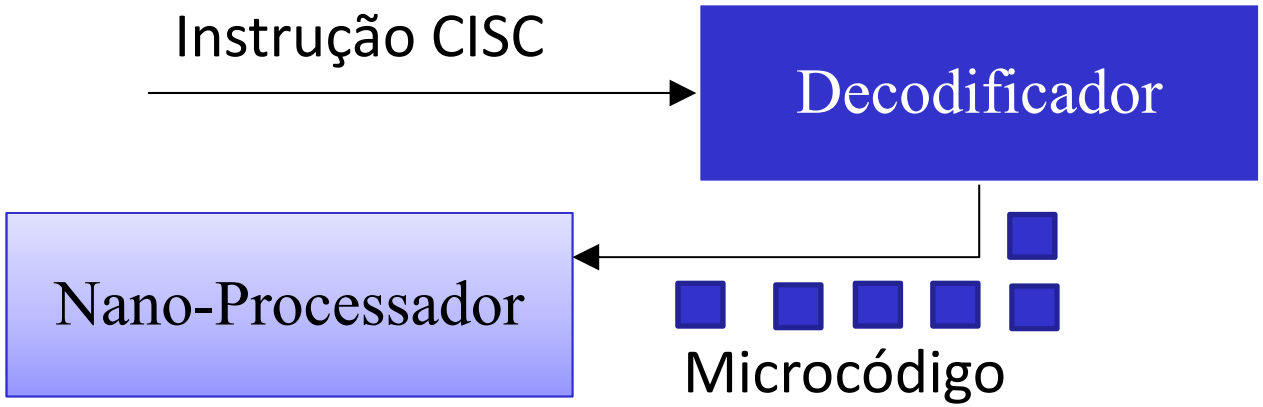
Microprocessadores (CPU)

- Outros periféricos necessários para funcionar (RAM, ROM, E/S, ...)
- Poderosa
- Expansivo
- Versatilidade
- Consumo / preço alto (normalmente)
- Uso geral

Microcontroladores (MCU)

- Tem CPU, RAM, ROM, E/S, no mesmo chip
- CPU menos poderosa
- Tratamento de poucos formatos de dados (tipicamente 8 ou 16 bits)
- Consumo / preço baixo (normalmente)
- Uso específico

Revisão - CISC - Microprogramação



- Cada instrução CISC separado em: instrução de máquina, tipo de endereçamento e endereços, registradores
- Seguinte: Envio de instruções pequenas (microcódigo) para Nano-processador (processador no processador)
- Execução de uma instrução CISC demora vários ciclos de clock

Revisão - RISC vs. CISC

- Desempenho (D)

$$\frac{tempo}{programa} = \frac{tempo}{ciclo} \times \frac{ciclos}{instrução} \times \frac{instruções}{programa}$$

- CISC:
  - Redução do número de instruções por programa
  - Aumento do número de ciclos por instrução
- RISC:
  - Redução do número de ciclos por instrução
  - Aumento de número de instruções por programa

Sistemas Processadores e Periféricos
Conjunto de Instruções MIPS

Prof. Frank Sill Torres
DELT – Escola de Engenharia
UFMG

Adaptado a partir dos Slides de Organização de Computadores 2006/02 do professor

Leandro Galvão DCC/UFAM - galvao@dcc.ufam.edu.br pelo Prof. Ricardo de Oliveira Duarte

Arquitetura MIPS

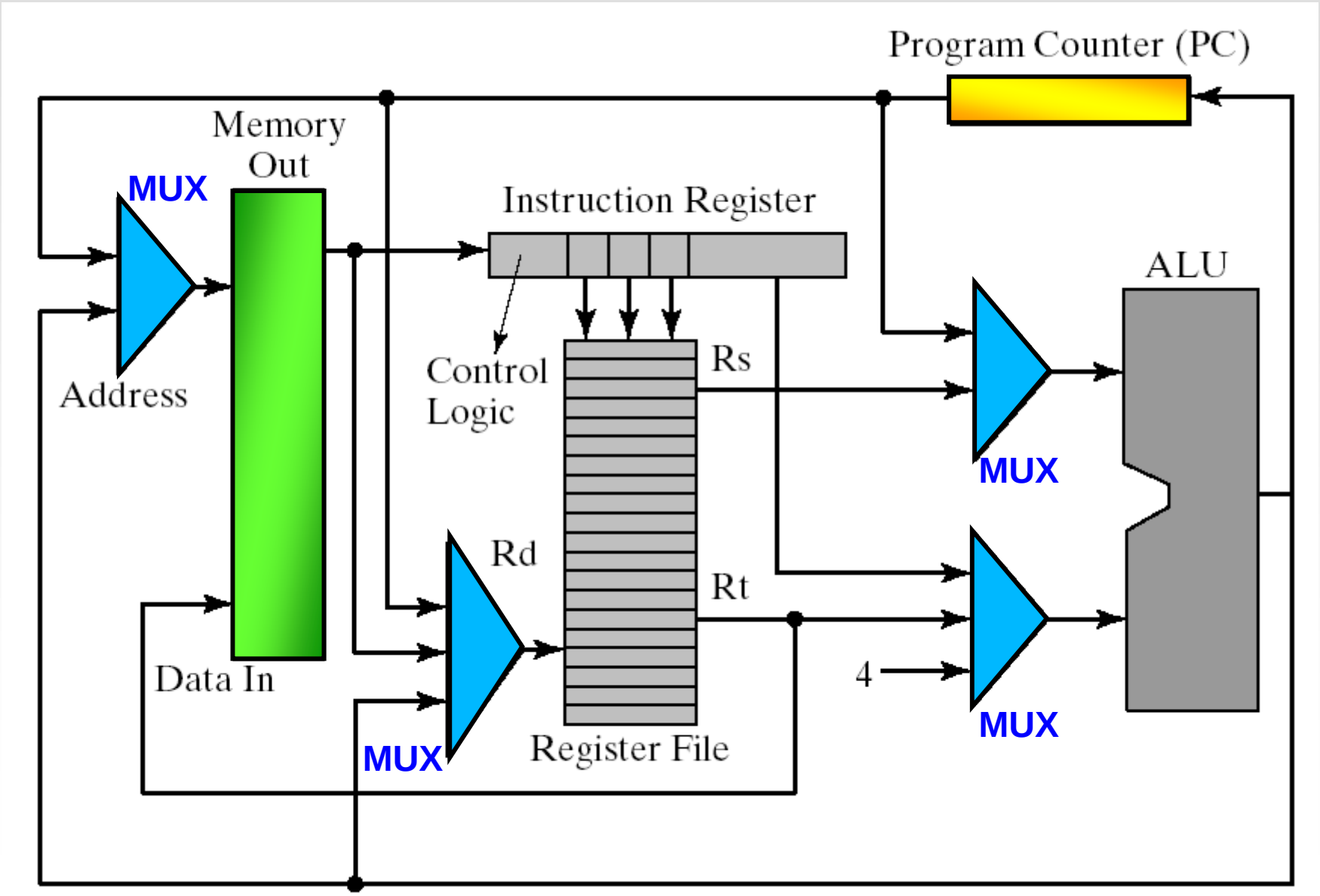
Arquitetura MIPS

- MIPS (**M**icroprocessor without **I**nterlocking **P**ipeline **S**tages)
- Arquitetura tipo RISC
- Versões de 32 e 64 bit (nós usamos: 32 bit)
- Quase 100 milhões de processadores MIPS fabricados em 2009
- Usada pela NEC, Nintendo, Cisco, Silicon Graphics, Sony, impressoras HP e Fuji, etc.

Arquitetura MIPS

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

Arquitetura MIPS
:: Diagrama simplificado



Arquitetura MIPS
:: Barramento

- A interconexão dos componentes básicos (exceto unidade de controle) se dá por meio do barramento (bus)
- Barramento é um conjunto de condutores por meio dos quais os bits são transmitidos
- Maior parte dos barramentos do MIPS tem 32 bits de largura

Arquitetura MIPS
:: Barramento

- Na maior parte das situações, é necessário enviar informações de mais de uma fonte para um só destino (Exemplo: dos 32 registradores para a ALU)

- Tal tarefa é implementada no hardware pelos multiplexadores (MUX)

Arquitetura MIPS
:: Unidade de controle

- Implementado no nível de hardware para buscar uma sequência de sinais de controle na memória (programa) e executá-las

- Envia sinais de controle para orientar os MUX sobre qual barramento de entrada deve ser selecionado para saída

- As trilhas de controle não estão mostradas no diagrama simplificado (folha 13)

Arquitetura MIPS
:: Banco de registradores

- A arquitetura MIPS possui um banco de 32 registradores
- Cada um deles comportam valores de 32 bits
- O nome de todos os registradores começa com o símbolo do cifrão: **$**

Arquitetura MIPS
:: Banco de registradores

- Adota-se uma convenção que especifica quais deles devem ser utilizados em certas circunstâncias

- Os registradores PC (contador de programas) e IR (registrador de instrução) não fazem parte do banco de registradores

Register	Number	Usage
zero	0	Constant 0
at	1	Reserved for the assembler
v0	2	Used for return values from function calls
v1	3	
a0	4	Used to pass arguments to functions
a1	5	
a2	6	
a3	7	
t0	8	Temporary (Caller-saved, need not be saved by called functions)
t1	9	
t2	10	
t3	11	
t4	12	
t5	13	
t6	14	
t7	15	
s0	16	Saved temporary (Callee-saved, called function must save and restore)
s1	17	
s2	18	
s3	19	
s4	20	
s5	21	
s6	22	
s7	23	
t8	24	Temporary (Caller-saved, need not be saved by called function)
t9	25	
k0	26	Reserved for OS kernel
k1	27	
gp	28	Pointer to global area
sp	29	Stack pointer
fp	30	Frame pointer
ra	31	Return address for function calls

Convenção de uso dos registradores

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

Arquitetura MIPS
:: Unidade lógica e aritmética (ALU)

- Responsável pela execução (em binário) das seguintes operações:
  - Aritméticas
  - Lógicas (AND, OR, NOT, XOR)

- A operação a ser realizada depende do código de operação buscado na memória principal

Arquitetura MIPS
:: Contador de programa (PC)

- É um registrador que é inicializado pelo sistema operacional com o endereço da primeira instrução do programa armazenado na memória

- Todas as instruções têm 32 bits de tamanho

- Após uma instrução ser buscada na memória e armazenada no IR, o PC é incrementado em quatro Bytes, de forma que o CPU terá o endereço da próxima instrução a ser executada

Arquitetura MIPS
:: Memória

- Pode ser vista como um grande arranjo de células:
  - onde informações são armazenadas (store)
  - de onde informações são buscadas (load)
- Tais operações são realizadas sobre uma palavra por vez
- A arquitetura MIPS define palavras de 4 Bytes de tamanho
- A menor unidade de endereçamento é um Byte

Arquitetura MIPS
:: Registrador de instrução (IR)

- É um registrador de 32 bits que possui uma cópia da mais recente instrução buscada na memória

- A depender do código contido no IR, a unidade de controle determinará qual a operação a ser realizada

Conceito de Programa Armazenado

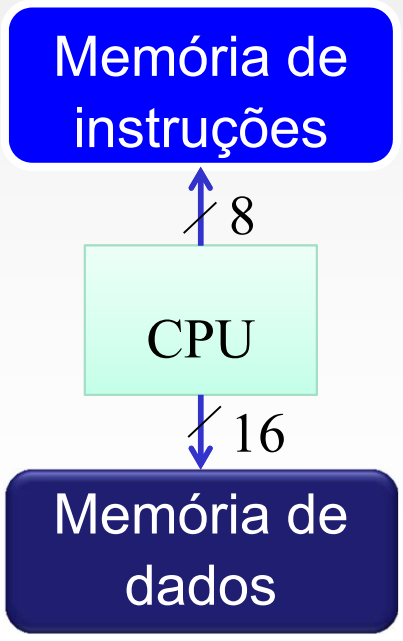
- John Von Neumann em 1942 (também: “Arquitetura-von-Neumann”)
- Instruções são codificadas em bits

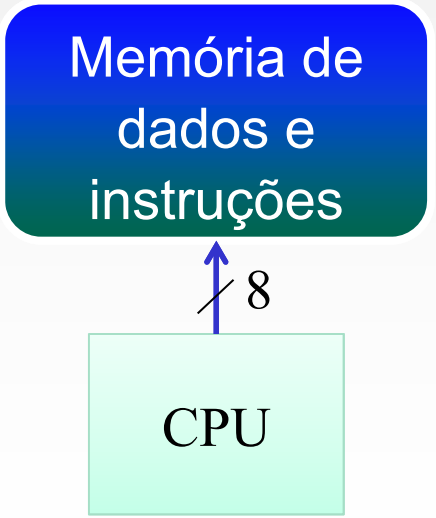
  add $17, $18, $19 → 00000010010100111000100000100000
- Programas são armazenados na memória
- Instruções são lidas e escritas da memória assim como os dados que serão transformados (processados)
- Ciclo de busca e execução da instrução (simplificado)

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

Conceito de Programa Armazenado

- "Arquitetura Harvard"
  - Instruções e dados em memórias separadas
  - Acesso de instruções independente do acesso de dados → acesso concorrente (ao mesmo tempo)
- "Harvard" vs. "Programa armazenado"





Instruções Básicas

Linguagem de Montagem (Assembly)

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

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

- 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 #

Linguagem de Montagem (Assembly) :: Exemplo

- Some b com c e coloque o resultado em a
  - Instrução de linguagem de montagem:

```
add $s0, $s1, $s2
```

  - Equivalente ao comando em linguagem C:

```
a = b + c
```

Linguagem de Montagem (Assembly)

- Em assembly, estamos manipulando registradores do MIPS
- Em código C (sem compilação), estamos manipulando posições da memória
- A associação entre posições da memória e registradores é realizada pelo compilador C

1º. Princípio de projeto MIPS

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

Código C:

```
A = B + C + D;
E = F - A
```

Código MIPS:

```
add $t0, $s1, $s2   #t0 = s1 + s2
add $t0, $t0, $s3   #t0 = t0 + s3
sub $s0, $s4, $t0   #s0 = s4 – t0
```

2º. Princípio de projeto MIPS

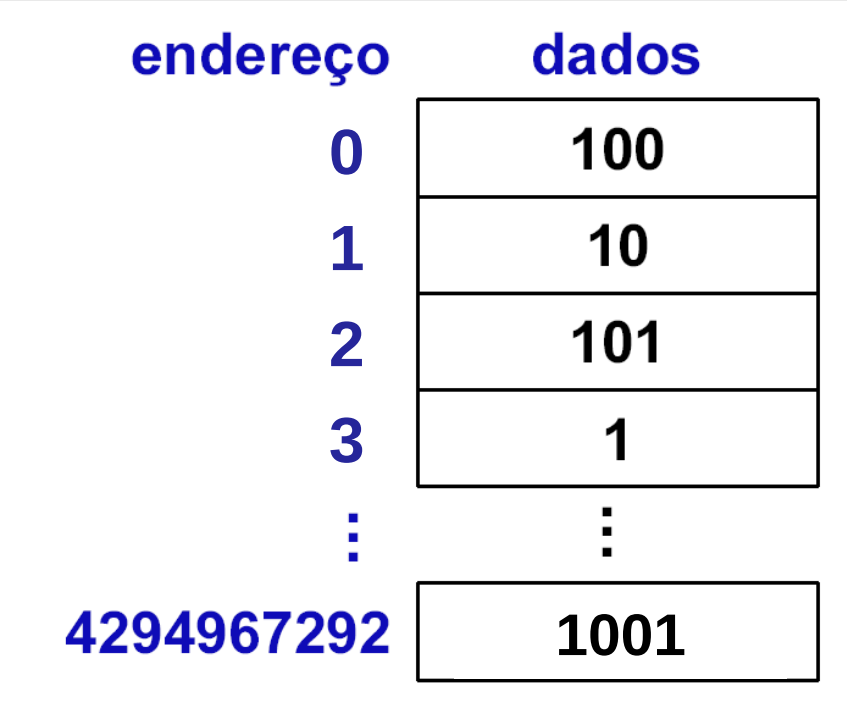
- **Menor significa mais rápido**
  - Uma quantidade maior que 32 registradores exigiria:
    - Um ciclo de clock maior
    - Formato de instruções maior, para comportar mais bits de endereçamento

Instruções MIPS
:: Armazenamento na memória

- 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
- Arquitetura MIPS endereça Bytes individuais
- Assim, endereços de palavras consecutivas diferem em 4 Bytes

Instruções MIPS
:: Armazenamento na memória

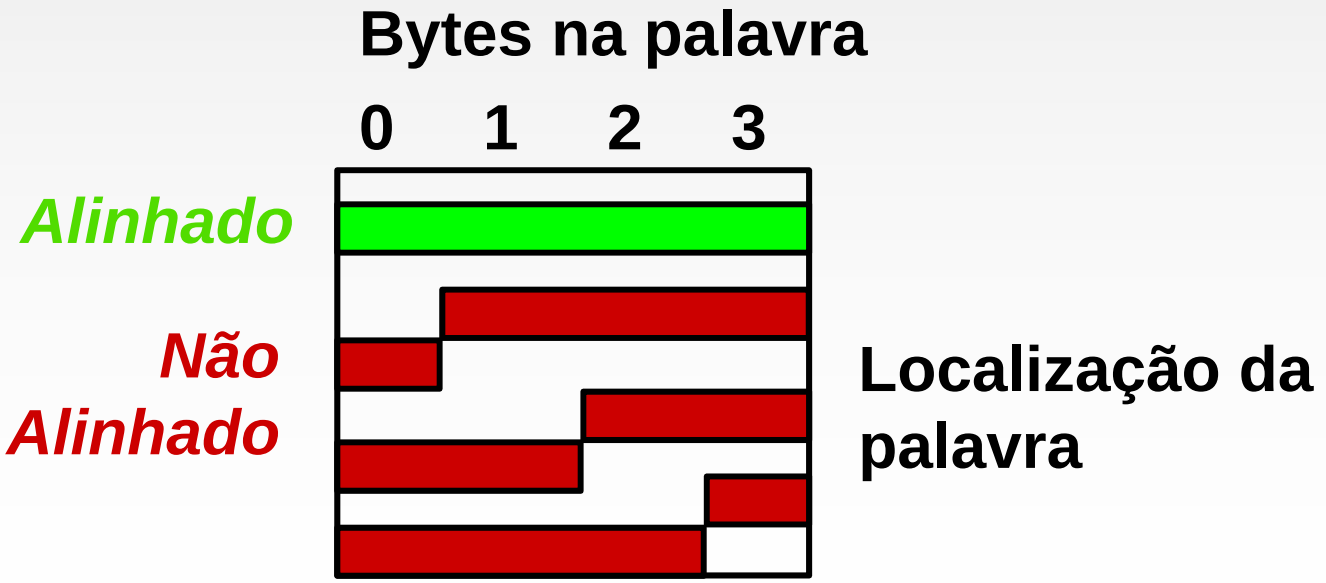
- O espaço de endereçamento de memória do MIPS é de 2^{32} Bytes (de 32 bits):



Instruções MIPS
:: Armazenamento na memória

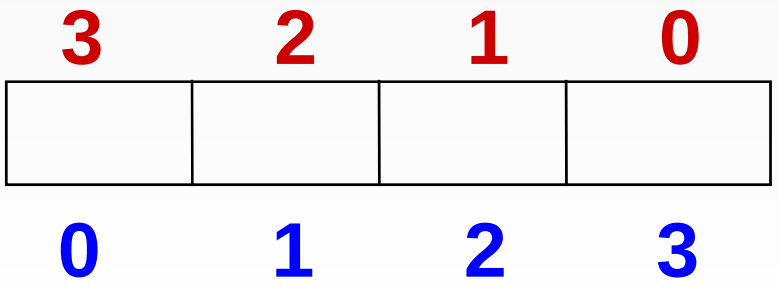
- MIPS exige que todas as palavras comecem em endereços que são múltiplos de 4 Bytes

- Alinhamento: objetos devem estar em um endereço que é um múltiplo do seu tamanho



Instruções MIPS
:: Armazenamento na memória

- Dois sistemas para numeração dos Bytes dentro uma palavra
- Big endian - Byte mais à esquerda marca endereço da palavra
- Little endian – Byte mais à direita marca endereço da palavra



Instruções MIPS
:: Armazenamento na memória

- Big endian - Byte mais à esquerda marca endereço da palavra
- Little endian – Byte mais à direita marca endereço da palavra

- Exemplo: palavra = **$6151CE94_h$**, endereço **= $0F40_h$**

1) Big Endian:

0F40	61	51	CE	94
0F44	00	00	00	00

2) Little Endian:

0F40	94	CE	51	61
0F44	00	00	00	00

Instruções MIPS
:: Armazenamento na memória

- Big Endian:
  - IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- Little Endian:
  - Intel 80x86, DEC Vax, DEC Alpha (Windows NT)
- Nomeação baseado em um artigo publicado em 1981: citação do 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

Instruções MIPS
:: Armazenamento na memória

Big-Endian **vs.** **Little-Endian**

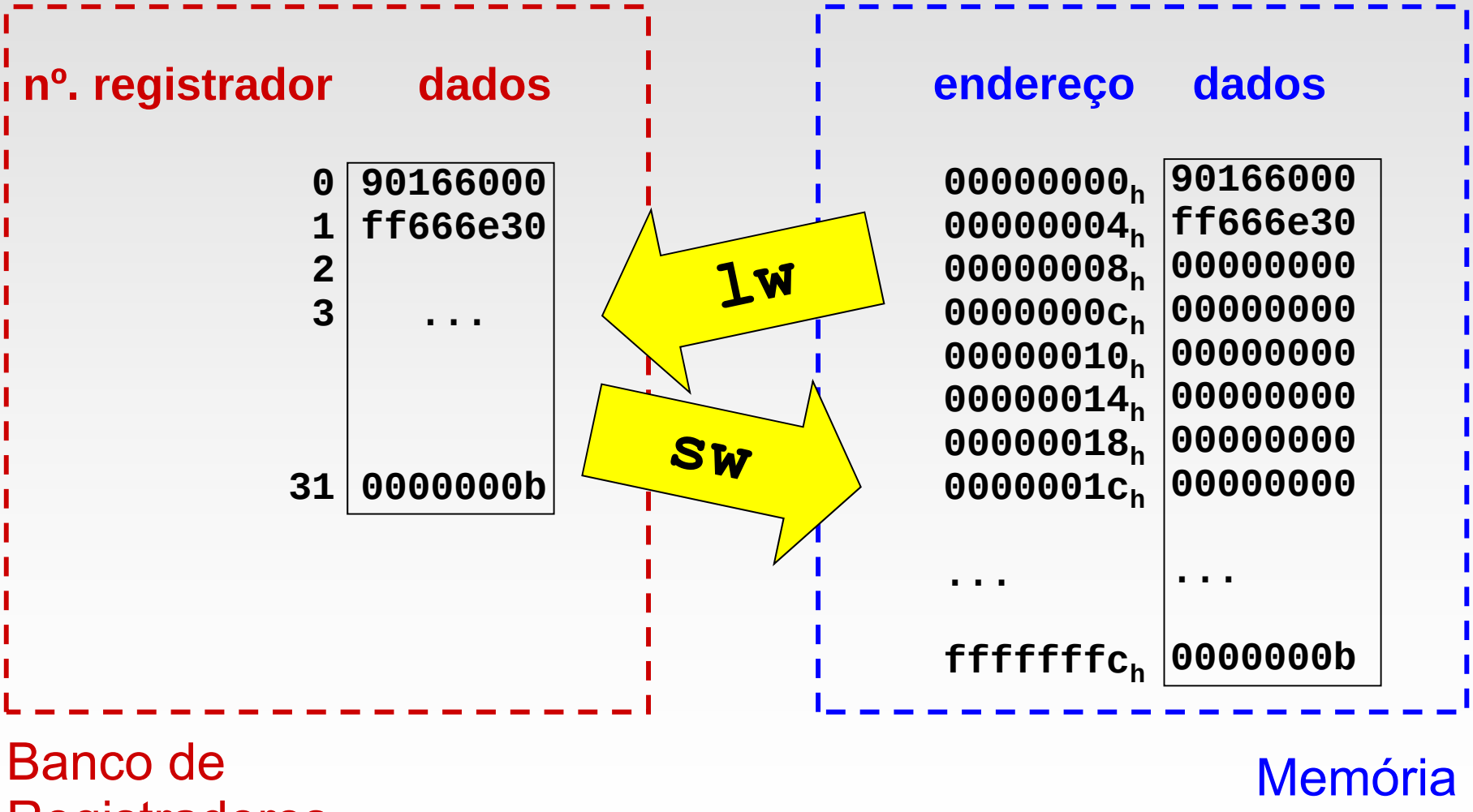
- mais fácil de determinar o sinal do número
- mais fácil de comparar dois números
- mais fácil de fazer a divisão
- mais fácil de imprimir

- adição e multiplicação dos números de multi-precisão é mais fácil

Instruções MIPS

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

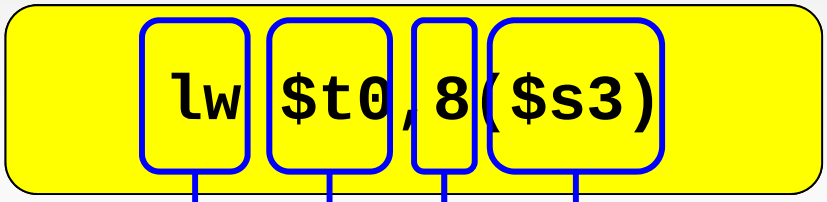
Instruções MIPS
:: Instruções de transferência de dados



Instruções MIPS
:: Instruções de transferência de dados

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

Formato:



instrução

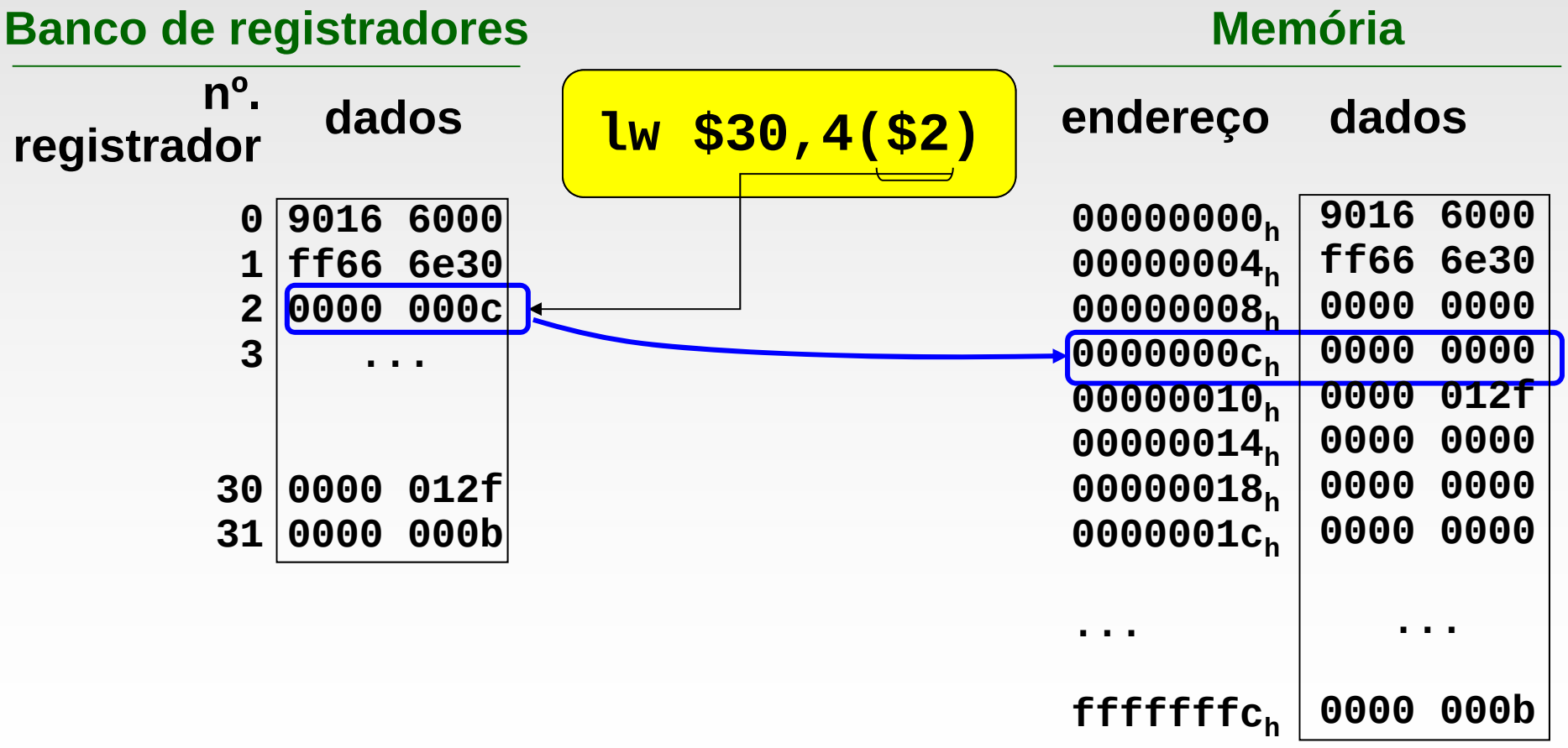
registrador de destino

offset (em Bytes)

registrador base, que armazena o endereço base

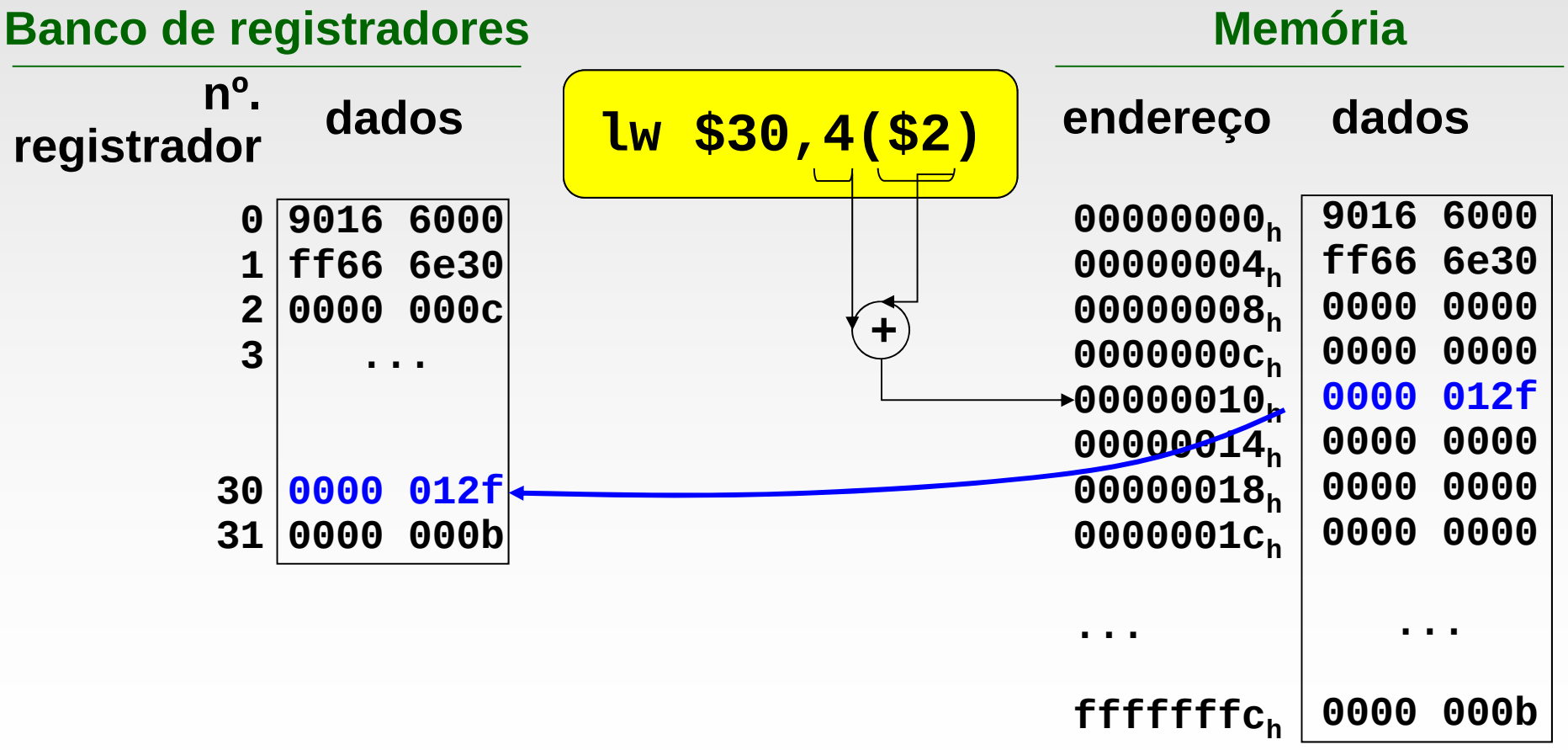
Instruções MIPS
:: Instruções de transferência de dados

- **Load word (lw)**



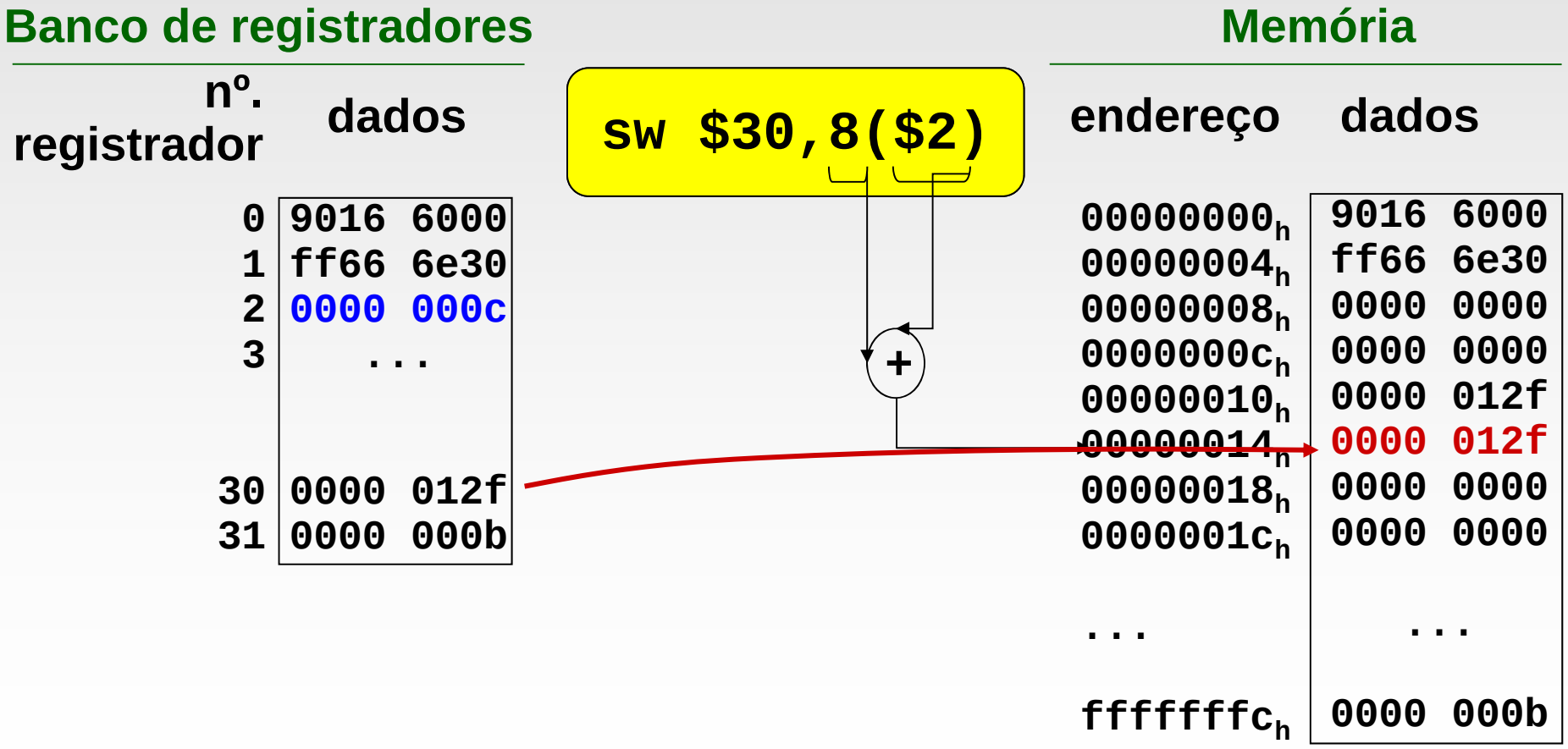
Instruções MIPS
:: Instruções de transferência de dados

- **Load word (lw)**



Instruções MIPS
:: Instruções de transferência de dados

- **Store word (sw)**



Instruções MIPS
:: Instruções de transferência de dados

- **Exemplo 1**
  - Suponha que o valor da variável h esteja armazenado em **$s2** e que o endereço base da matriz A esteja armazenado em **$s3**.
  - Qual o código assembly para: A[12] = h + A[8]?

```
lw  $t0, ?($s3)     #   $t0 ← A[8]
add $t0, $s2, $t0   #   $t0 ← $s2 + A[8]
sw  $t0, 48($s3)    # A[12] ← h + A[8]
```

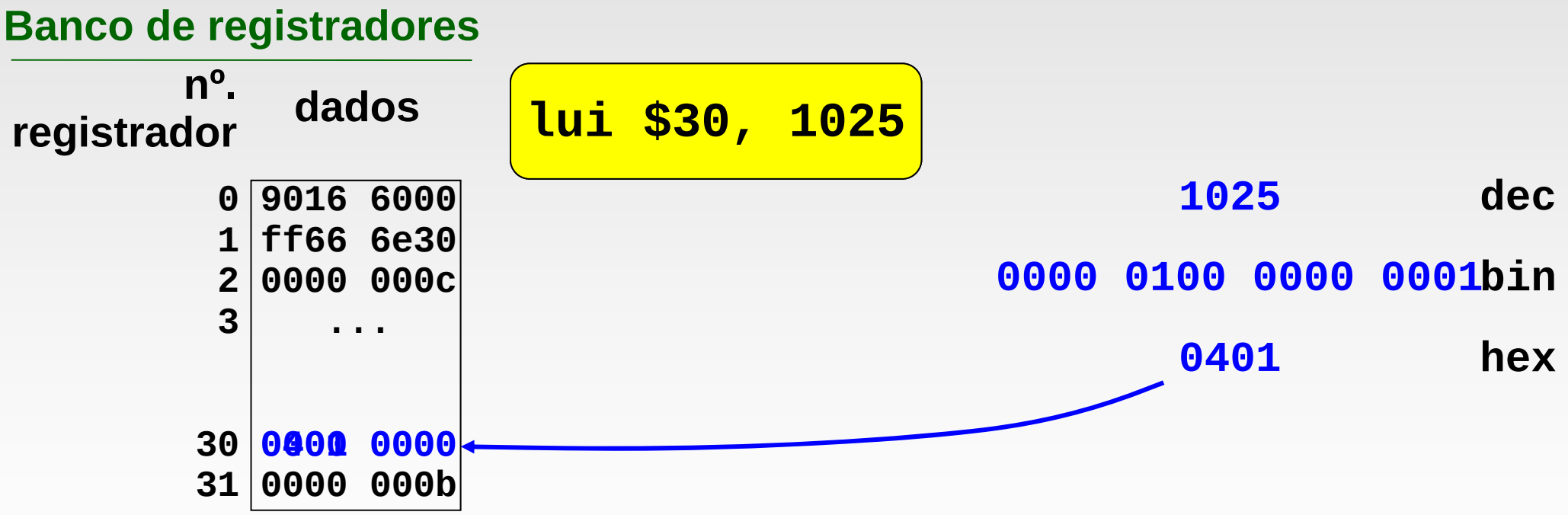
Instruções MIPS
:: Instruções de transferência de dados

- **Exemplo 2**
  - Suponha que o endereço base da matriz B esteja armazenado em **$s4**.
  - Qual o código assembly para trocar os valores do B[10] e do B[11]?

```
lw  $t0, 40($s4)     #   $t0 ← B[10]
lw  $t1, 44($s4)     #   $t1 ← B[11]
sw  $t0, 44($s4)     # B[11] ← B[10]
sw  $t1, 40($s4)     # B[10] ← B[11]
```

Instruções MIPS
:: Instruções de transferência de dados

- **Load upper immediate (lui)**



Representando instruções no computador

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

- Dessa forma, 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)

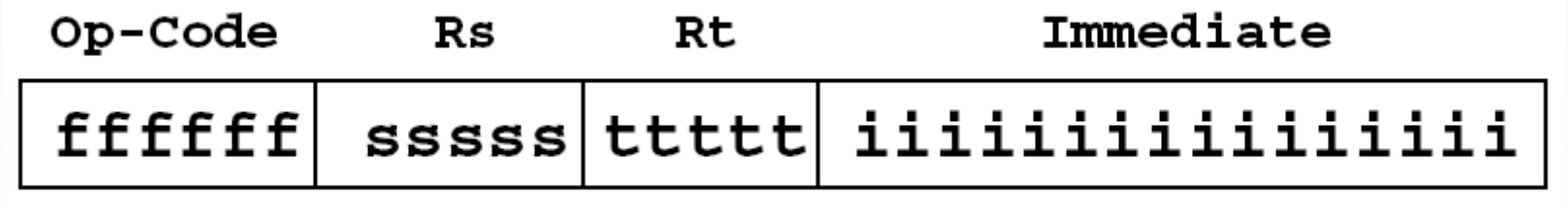
Representando instruções no computador

- **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). Será abordado mais tarde.
  - Function code: especifica qual a operação a ser executada

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

Representando instruções no computador

- **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



Representando instruções no computador

- **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.

Representando instruções no computador

- **Exemplos**

```
add $t0, $s1, $s2
```

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
add	$s1	$s2	$t0	--	add

Instrução (decimal):

0	17	18	8	0	$(20)_h$
add	$s1	$s2	$t0	--	add

Instrução (binário):

000000	10001	10010	01000	00000	100000

Representando instruções no computador

- **Exemplos**

```
lw $t0, 32($s2)
```

	op	rs	rt	immediate
	6 bits	5 bits	5 bits	16 bits
	lw	$s2	$t0	offset
Instrução (decimal):	$(23)_h$	18	8	32
	lw	$s2	$t0	offset
Instrução (binário):	100011	10010	01000	0000000000100000

Representando instruções no computador

- **Exemplos**

```
sw $t0, 32($s2)
```

	op	rs	rt	immediate
	6 bits	5 bits	5 bits	16 bits
	sw	$s2	$t0	offset
Instrução (decimal):	$(2B)_h$	18	8	32
	sw	$s2	$t0	offset
Instrução (binário):	101011	10010	01000	0000000000100000

Representando instruções no computador

- **Exemplos**

```
lui $t0, 1023
```

op	rs	rt	immediate
6 bits	5 bits	5 bits	16 bits
lui	--	$t0	immediate

Instrução (decimal):

$(F)_h$	0	8	1023
lui	--	$t0	immediate

Instrução (binário):

001111	00000	01000	0000011111111111

Constantes :: Instruções Imediatas

- Constantes são utilizadas com frequência em operações
- Nesses caso, incluir constantes em uma instrução agiliza as operações, em vez de carregar as constantes a partir da memória
- Tais instruções são conhecidas como instruções imediatas

Constantes :: Instruções Imediatas

- Exemplo: Adição imediata (addi)

```
addi  $s3, $s2, 4     # $s3 ← $s2 + 4
```

Instrução (decimal):

op	rs	rt	immediate
8	18	19	4
addi	$s2	$s3	constante

Instrução (binário):

001000	10010	10011	0000000000000100
addi	$s2	$s3	constante

3º. Princípio de projeto MIPS

- **Agilize os casos mais comuns**
  - As instruções imediatas ilustram este princípio, consequente direto da Lei de Amdahl

Lei de Amdahl

"O maior aumento de desempenho possível introduzindo melhorias numa determinada característica é limitado pela percentagem em que essa característica é utilizada"

$$\textit{Tempo de execução (Tex) após melhoria} = \frac{T_{ex}\ \textit{afetado pela melhoria}}{\textit{Quantidade da melhoria}} + T_{ex}\ \textit{não-afetado}$$

$$T_{novo} = \frac{F \cdot T_{velho}}{S} + (1 - F) \cdot T_{velho} = T_{velho}\left[\frac{F}{S} + (1 - F)\right]$$

$$Melhoria_{global} = \frac{T_{velho}}{T_{novo}} = \frac{1}{\frac{F}{S} + (1 - F)}$$

Lei de Amdahl

- Exemplo 1: Considere um programa com T_{ex} = 100 seg, sendo 20% adições. Qual o ganho se a adição for 4 vezes mais rápida?

$$T_{ex\_novo1} = \frac{20}{4} + 80 = 85s \qquad ganho = \frac{100}{85} = 1,18$$

- E se for o resto 2 vezes mais rápida?

$$T_{ex\_novo2} = 20 + \frac{80}{2} = 60s \qquad ganho = \frac{100}{60} = 1,67$$

Lei de Amdahl

- Exemplo 2: Considere que a instrução *lw* precisa de 9 ciclos de clock, a instrução *add* precisa de 1 ciclo e a instrução *addi* precisa de 2 ciclos. De qual fator a instrução *lw* precisa ser melhor para que uma adição de uma constante usando *add* seja mais rápida do que com *addi*?

```
lw  $t0, 10($s4)
add $s1, $s1, $t0
```

```
addi  $s1, $s1, 4
```

$$T_{lw,add,velho} = 9 + 1 = 10,\ T_{addi} = 2$$

$$\frac{T_{lw,add,velho}}{T_{lw,add,novo}} = \frac{1}{\frac{9/10}{S} + \left(1 - 9/10\right)} > \frac{T_{lw,add,velho}}{T_{addi}} = 5$$

$$S > 9$$

➔ A instrução *lw* precisa ser 9 vezes mais rápida

Constantes grandes

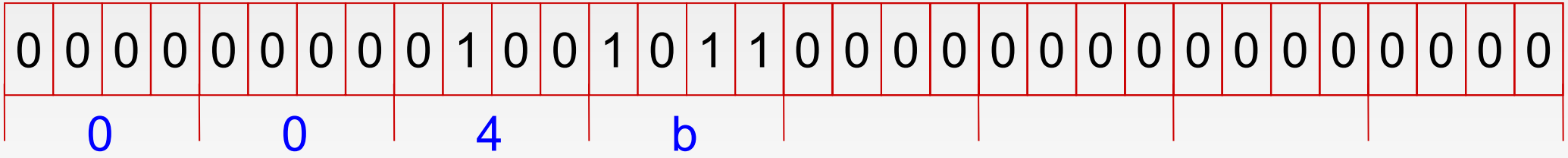
- Como carregar constantes de 32 bits em um registrador?

- Duas instruções são necessárias:
  - Instrução **`lui`** (load upper immediate) para carregar bits mais significativos
  - Instrução **`ori`** (or immediate) para carregar bits menos significativos

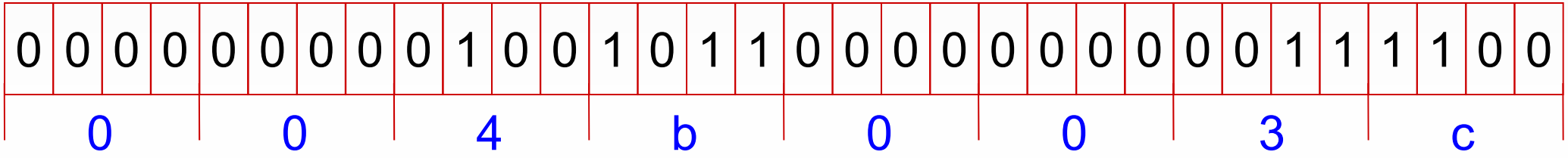
Constantes grandes

- Exemplo: deseja-se carregar o registrador $t0 com o endereço de memória 0x004b003c

```
lui $t0, 0x004b
```




```
ori $t0,$t0, 0x003c
```



O que vocês aprenderam hoje?

- Arquitetura MIPS
  - Barramento
  - Banco de Registradores
  - ....
- Instruções Básicas
- Princípios de projeto de MIPS
- Armazenamento na memória
- Representação de instruções

Questões

Converta o código em C seguinte para o código *assembly* usando o conjunto de instruções MIPS.

// Suponha que os valores das variáveis *x*, *y* e *z* estejam armazenados em $s2, $s3
// e $s4 e que o endereço base da matriz B esteja armazenada em $s6

```
main() {
    int x, y, z;
    int B[20];

    B[0] = x;
    y = B[2] + B[3];
}
```

```
sw $s2, 0($s6)      // B[0] = x
lw $t0, 8($s6)      // t0 = B[2]
lw $t1, 12($s6)     // t1 = B[3]
add $s3,$t0,$t1     // y = t0+t1
```

Questões

Converta o código assembly para o código de máquina usando estas tabelas

Nome	$zero	$v0–$v1	$a0–$a3	$t0–$t7	$s0–$s7	$t8–$t9
Número	0	2 - 3	4 - 7	8 - 15	16 - 23	24 - 25

Nome	Sintaxe	Função	Formato / Opcode / funct		
Adição	add $s1, $2, $3	$1 = $2 + $3 (signed)	R	0	20_{hex}
Carrega word	lw $t,C($s)	$t = Memória[$s + C]	I	23_{hex}	-
Armazena word	sw $t,C($s)	Memória[$s + C] = $t	I	$2B_{hex}$	-

sw $s2, 0($s6)	101011 10110 10010 00000 00000 000000
lw $t0, 8($s6)	100011 10110 01000 00000 00000 001000
lw $t1, 12($s6)	100011 10110 01001 00000 00000 001100
add $s2,$t0,$t1	000000 01000 01001 10010 00000 100100