Universidade Tecnológica Federal do Paraná – UTFPR
Bacharelado em Ciência da Computação

BCC33B – Arquitetura e Organização de Computadores

Prof. Rogério A. Gonçalves

rogerioag@utfpr.edu.br

Aula 004

Aula de Hoje:

***Assembly*: Formato Geral de Instruções**

Arquitetura do Microprocessador MIPS

Arquitetura interna do microprocessador

Registradores

Barramentos

Conjunto de Instruções

Formatos de Instruções

Tipos de Instruções

Modos de Endereçamento

Linguagem *Assembly*

Conjunto de Instruções

- **ISA – Instruction Set Architecture**
  - **O ISA é a porção da máquina visível ao:**
    - Programador (nível de montagem)
    - Projetistas de compiladores.

Linguagem *Assembly*

Linguagem de Alto Nível →

```
a = b + c
e = a - d
```

Linguagem de Montagem (*Assembly*) →

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

Linguagem de Máquina →

```
00000010001100100100000000100000
00000001001100110100100000100010
```

Formato da Instrução Assembly

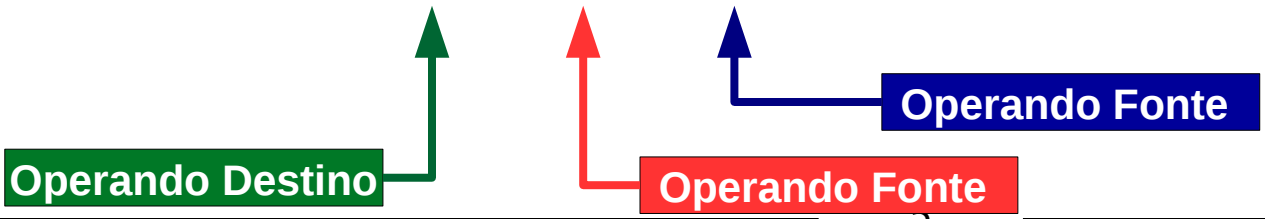
```
Rótulo (Label)   Mnemônico   Operandos        Comentário

LOOP:            add         $t1, $s1, $s2    # t1 = s1 + s2
                 sub         $t1, $t1, $s3    # t1 = t1 - s3
                 j           LOOP             # salta para LOOP
```

- Label: é uma referência simbólica a uma posição de memória

- Mnemônico: palavra abreviada que define a instrução

Operandos: indicam onde estão os dados a serem processados

Ex: `add $t1, $s1, $s2`



Instruções do MIPS

Operandos

- **Um computador necessita de localizações físicas de onde buscar os operandos binários.**
- **Um computer busca operandos de:**
  - **Registradores**
    - » add $t1, $s1, $s2
  - **Memória**
    - » lw $t2, 32($0) (instruções load e store, por exemplo)
  - **Constantes (também denominados de *imediatos*)**
    - » addi $t1, $s1, **7**

Instruções do MIPS

Operandos

- Trabalhar diretamente com a Memória é mais lento.
- A ideia é trabalhar sobre registradores.
- Muitas arquiteturas possuem um conjunto pequeno de registradores (rápidos).
- MIPS tem trinta e dois registradores de 32-bit.
- MIPS é uma arquitetura de 32-bit devido seus operandos serem dados de 32-bit.

(Uma versão MIPS de 64-bit também existe)

Tipos de Dados

- **Tipos de Dados e Constantes**
  - Byte: 8 bits
  - Halfword (meia-palavra): 2 bytes: 16 bits
  - Word (palavra): 4 bytes : 32 bits
  - Caracteres ocupam 1 byte para serem armazenados
  - Um inteiro requer 1 palavra (32 bits).
- **Constantes e Literais**
  - Números diretamente: 4.
  - Caracteres entre aspas simples: 'b'.
  - Strings (cadeias de caracteres) entre aspas duplas: “Uma cadeia”.

Registradores

- **Existem 32 registradores de propósito geral**
  - **São precedidos por $ nas instruções em *Assembly***
- ***Dois formatos de representação:***
  - *Usando o número de representação do registrador. Ex.: $0 até $31*
  - *Usando o nome equivalente. Ex.: $t1, $s2, $sp*
- ***Convenção do uso:***
  - ***$t0 a $t9 = ($8 a $15, $24, $25);*** *são registradores de uso geral. Não precisam ser preservados nas chamadas de procedimentos.*
  - ***$s0 - $s7 = ($16 a $23);*** *registradores de uso geral. Devem ser preservados nas chamadas de procedimentos.*
  - ***$sp = $29;*** *stack pointer (ponteiro da pilha)*
  - ***$fp = $30;*** *frame pointer (início da pilha)*
  - ***$ra = $31;*** *return address: endereço de retorno da chamada de subrotina.*
  - ***$a0 a $a3 = ($4 a $7);*** *são usados para argumentos para subrotina.*
  - ***$v0, $v1 = ($2, $3);*** *são usados para retorno de valores de uma subrotina.*

Registradores

Nome	Número	Uso
$0 ou $zero	0	Valor constante 0
$at	1	assembler temporary
$v0-$v1	2-3	Valores de retorno de procedimentos.
$a0-$a3	4-7	Passagem de parâmetro (argumentos) para procedimentos.
$t0-$t7	8-15	Temporários.
$s0-$s7	16-23	Variáveis Salvas.
$t8-$t9	24-25	Mais temporários.
$k0-$k1	26-27	Temporários para uso pelo Sistema Operacional
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address: endereço de retorno do procedimento.

Registradores

Nome	Número do registrador	Uso	Preservado na chamada?
$zero	0	o valor constante 0	n.a.
$v0-$v1	2-3	valores para resultados e avaliação de expressões	não
$a0-$a3	4-7	Argumentos	não
$t0-$t7	8-15	Temporários	não
$s0-$s7	16-23	Valores salvos	sim
$t8-$t9	24-25	Mais temporários	não
$gp	28	Ponteiro global	sim
$sp	29	stack pointer	sim
$fp	30	frame pointer	sim
$ra	31	Endereço de retorno	sim

FIGURA 2.18 Convenções de registradores MIPS. O registrador 1, chamado $at, é reservado para o montador (ver Seção 2.10), e os registradores 26-27, chamados $k0–$k1, são reservados para o sistema operacional.

Registradores

- **Registradores especiais Lo e Hi**
  - são usados para armazenar o resultado de **divisões** e **multiplicações**
  - **Não são diretamente endereçáveis**
  - **Acesso por meio de instruções especiais:**
    - **`mfhi`** (“move from Hi”)
    - **`mflo`** (“move from Lo”)

Conjunto de Instruções

Classificação das Instruções

As instruções são classificadas por funções (tipos):

1) **Instruções de Transferência:**
   - Movem dados entre registradores, memória e E/S.
2) **Instruções Lógicas e Aritméticas:**
   - Executam operações lógicas e aritméticas com os dados (operandos) contidos nos registradores.
3) **Instruções de Desvio:**
   - Fazem com que o microprocessador altere o modo normal de execução sequencial, podendo desviar para uma nova instrução endereçada pela instrução de desvio.
4) **Instruções de Controle (especiais):**
   - **Instruções para controlar a pilha e instruções de controle da máquina (estado do processador e modo de operação do processador).**

Instruções MIPS

- **Assembly:**
  - **Linguagem de mnemônicos ou linguagem de montagem.**
- **Linguagem de Máquina**
  - **Computadores só "conhecem" 0's e 1's**
    - **Linguagem de Máquina**: representação binária das instruções
  - **Instruções de 32 bits (32-bit)**
    - » Simplicidade em favor da regularidade: dados e instruções de 32-bit
  - **Três formatos de instruções :**
    - » R-Type: Operandos registradores (register operands)
    - » I-Type: Operandos imediatos (immediate operand)
    - » J-Type: Saltos, desvios, *branches, jumps*.

Instruções MIPS

- **R-type: *Register-type***
  - **3 operandos são registradores:**
    - » **`rs`, `rt`**: source registers
    - » **`rd`**: destination register
  - **Outros campos:**
    - » **`op`**: opcode ou código da *operação.*
    - » **`funct`**: função
      - juntos, o opcode e a função informam a operação a ser executada
    - » `shamt`: a quantidade de *shift para instruções de deslocamento*

R-Type

Instruções MIPS

Assembly Code

```
add $s0, $s1, $s2
sub $t0, $t3, $t5
```

Field Values

op	rs	rt	rd	shamt	funct
0	17	18	16	0	32
0	11	13	8	0	34
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Machine Code

op	rs	rt	rd	shamt	funct	
000000	10001	10010	10000	00000	100000	(0x02328020)
000000	01011	01101	01000	00000	100010	(0x016D4022)
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Nota: a ordem dos registradores no código assembly:

`add rd, rs, rt`

Instruções MIPS

- **I-Type: Immediate-Type**
  - **3 operands:**
    - » **`rs`**, **`rt`**: register operands
    - » **`imm`**: 16-bit em complemento de dois immediate
  - **Outros campos:**
    - » **`op`**: opcode

I-Type

Instruções MIPS

- **Exemplo I-Type:**

Assembly Code

```
addi $s0, $s1, 5
addi $t0, $s3, -12
lw   $t2, 32($0)
sw   $s1,  4($t1)
```

Field Values

op	rs	rt	imm
8	17	16	5
8	19	8	-12
35	0	10	32
43	9	17	4
6 bits	5 bits	5 bits	16 bits

Nota: a ordem dos registradores no código assembly:

```
addi rt, rs, imm
lw   rt, imm(rs)
sw   rt, imm(rs)
```

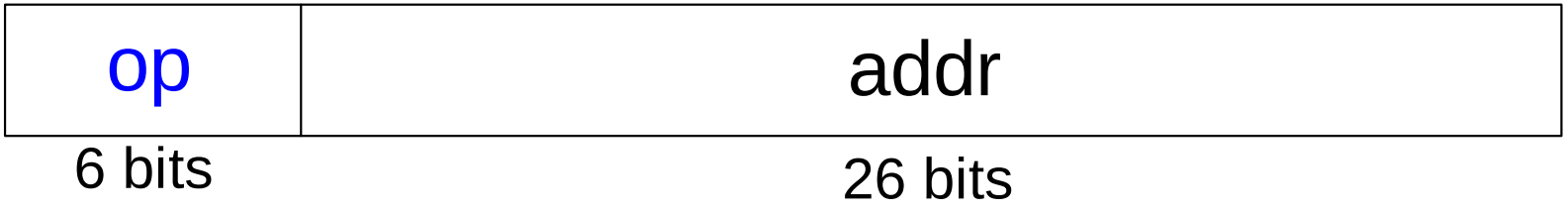
Machine Code

op	rs	rt	imm	
001000	10001	10000	0000 0000 0000 0101	(0x22300005)
001000	10011	01000	1111 1111 1111 0100	(0x2268FFF4)
100011	00000	01010	0000 0000 0010 0000	(0x8C0A0020)
101011	01001	10001	0000 0000 0000 0100	(0xAD310004)
6 bits	5 bits	5 bits	16 bits	

Instruções MIPS

- **J-Type: Jump-Type**
  - **26-bit address operand (`addr`)**
  - **Usado nas instruções jump (`j`)**

J-Type



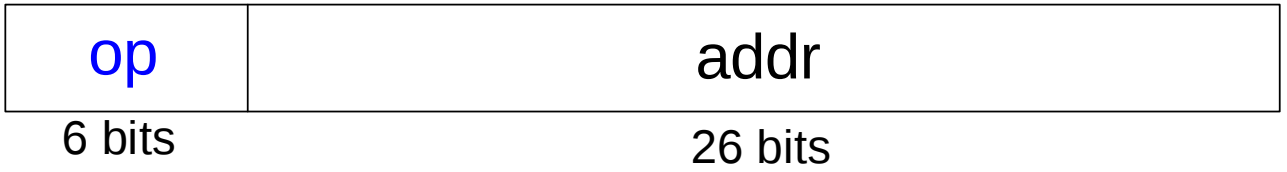
Instruções MIPS

- **Resumo dos Formatos das Instruções**

R-Type

I-Type

J-Type



Instruções Aritméticas

- **Adição**

Código de Alto Nível

```
a = b + c;
```

Assembly MIPS

```
# $s0 = a, $s1 = b, $s2 = c
add $s0, $s1, $s2
```

add: mnemônico, indica qual a operação a ser executada

$s1, $s2: operandos fonte

$s0 : operando destino, onde será armazenado o resultado

Instruções Aritméticas

- **Subtração**

Código de Alto Nível

```
a = b - c;
```

Assembly MIPS

```
# $t0 = a, $t1 = b, $t2 = c
sub $t0, $t1, $t2
```

sub : mnemônico, indica qual a operação a ser executada

$t1, $t2: operandos fonte

$t0 : operando destino, onde será armazenado o resultado

Instruções Aritméticas

Essas instruções podem ser combinadas para a implementação de códigos mais complexos:

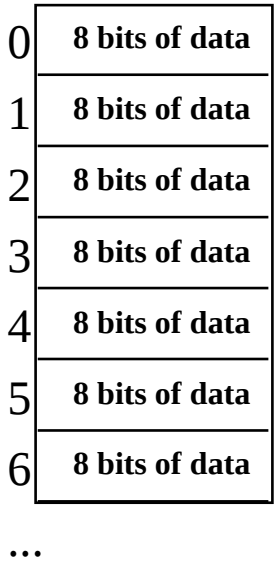
Código de Alto Nível	Assembly MIPS
`a = b + c - d;`	`# $s0 = a, $s1 = b, $s2 = c, $s3 = d`
	`add $t0, $s1, $s2  # t = b + c`
	`sub $s0, $t0, $s3  # a = t - d`

```
// single line comment
/* multiple line
    comment */
```

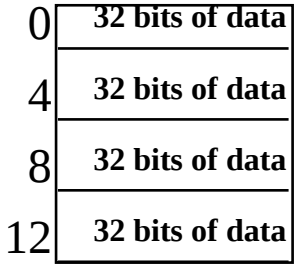
Acesso à Memória

Acesso à memória alinhado a:

- **Byte – dados**

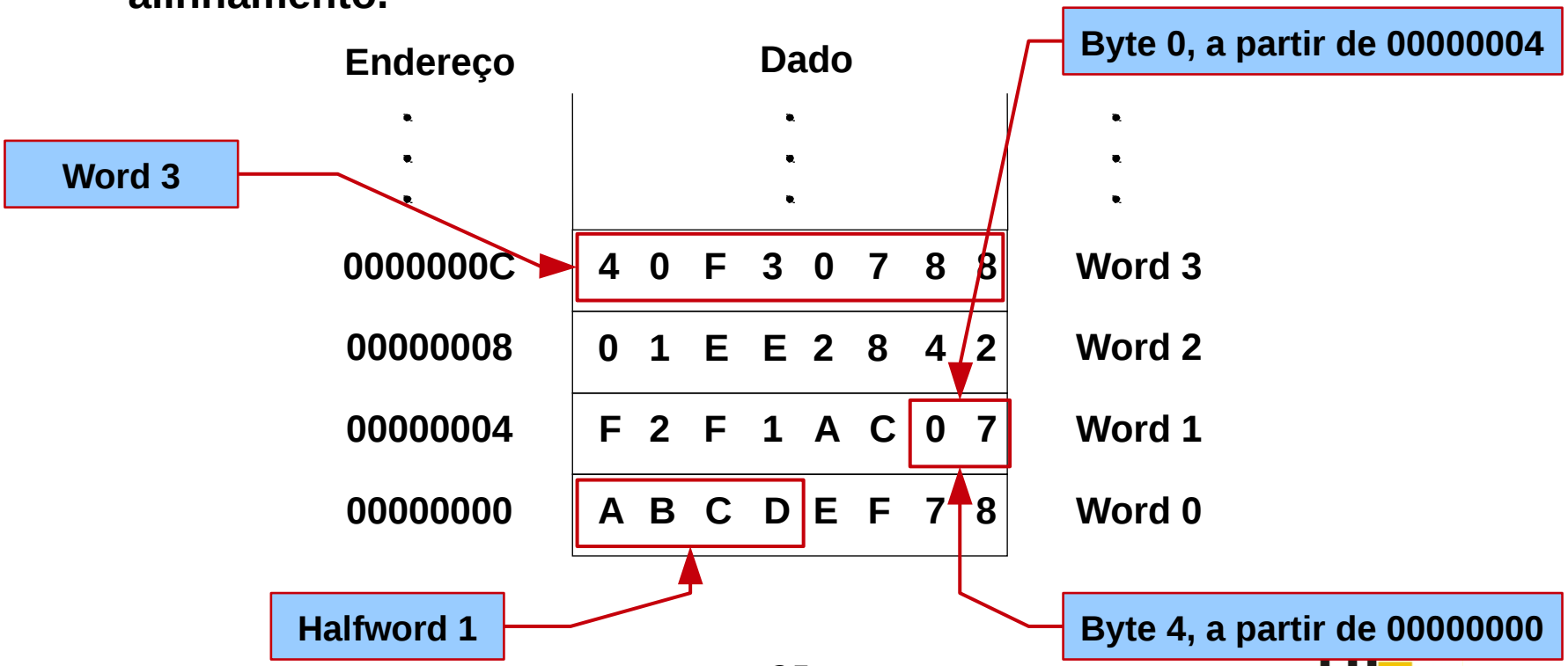


- **Word – instruções**



Memória

- **Operandos em Memória**
  - Memória é byte endereçável
  - É possível acessar palavras, meias-palavras e bytes fazendo o devido deslocamento a partir do endereço base e considerando o alinhamento.



Instruções Load/Store

- **Lendo uma palavra (word) da memória**
  - **Instrução *load word*: lw**
  - **Opcode: 100011 (35), I-type**

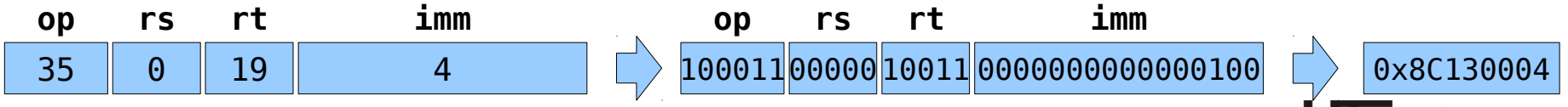
Deslocamento é em bytes a partir do endereço base

Código *Assembly*

```
# $0: 00000000 (constante zero)
lw $s3, 4($0)  # read memory word 1 into $s3 (1x4)
```

$s3: F2F1AC07

Endereço	Dado	
. . .	. . .	. . .
0000000C	4 0 F 3 0 7 8 8	Word 3
00000008	0 1 E E 2 8 4 2	Word 2
00000004	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

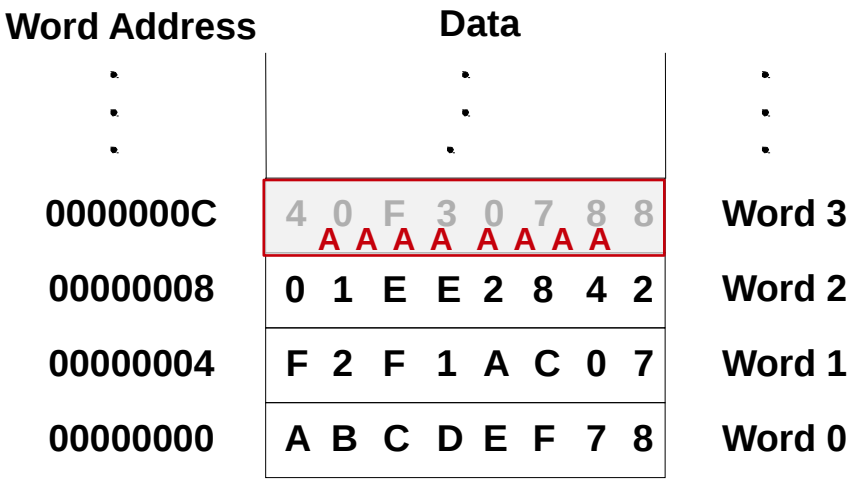


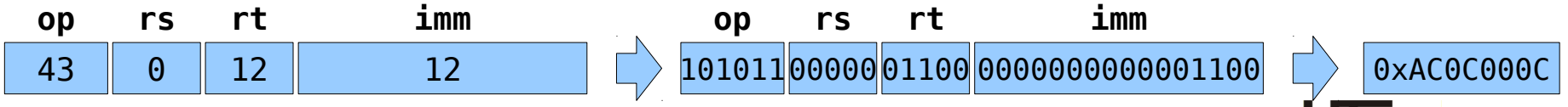
Instruções Load/Store

- **Escrevendo uma palavra (*word)* na memória**
  - **Instrução *Store Word*: sw**
  - **Opcode: 101011 (43), I-type**

Código Assembly `$t4: AAAAAAAA`

```
sw $t4, 0xC($0)  # write $t4 to memory word 3 (3x4)
```



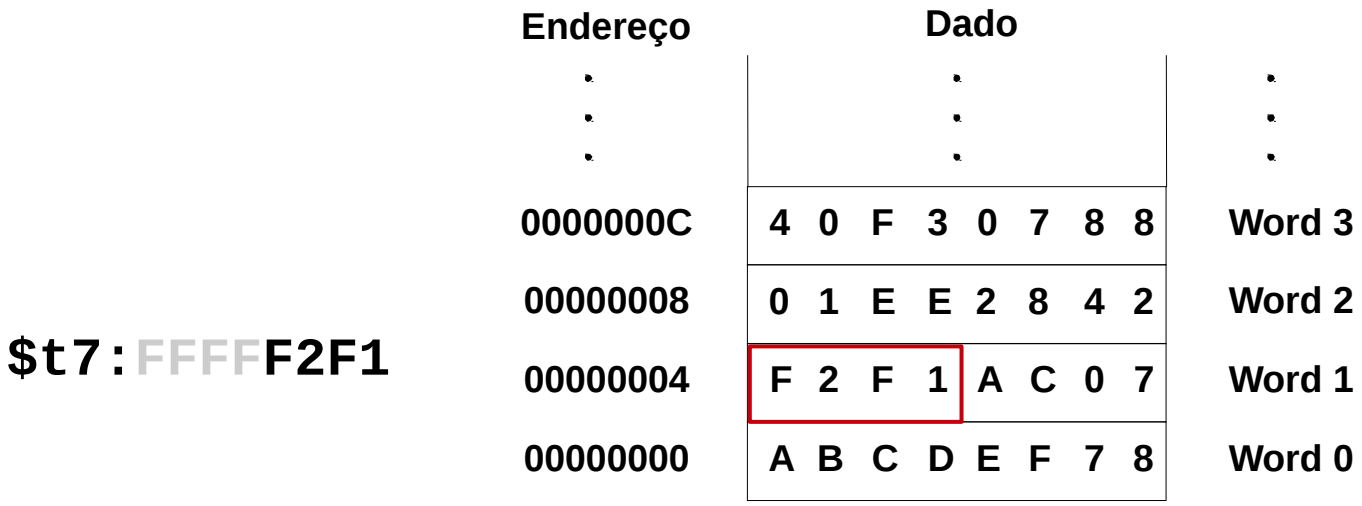


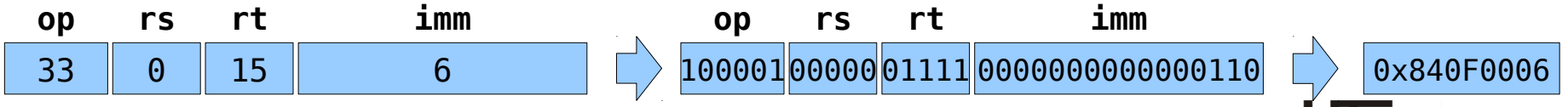
Instruções Load/Store

- **Lendo uma meia-palavra (halfword) da memória**
  - **Instrução *load half*: lh**
  - **Opcode: 100001 (33), I-type**

Código *Assembly*

```
lh $t7, 6($0)  # read memory half word 3 into $t7 (3x2)
```





Instruções Load/Store

- **Escrevendo uma meia-palavra (halfword) na memória**
  - **Instrução *Store half*: sh**
  - **Opcode: 101001 (41), I-type.**

Código Assembly $t4: ABCDEFEF

```
sh $t4, 14($0)  # write $t4 to memory halfword 7 (7x2)
```

Word Address	Data	
0000000C	4 0 F 3 **E F E F** 0 7 8 8	Word 3
00000008	0 1 E E 2 8 4 2	Word 2
00000004	F 2 F 1 A C 0 7	Word 1
00000000	A B C D E F 7 8	Word 0

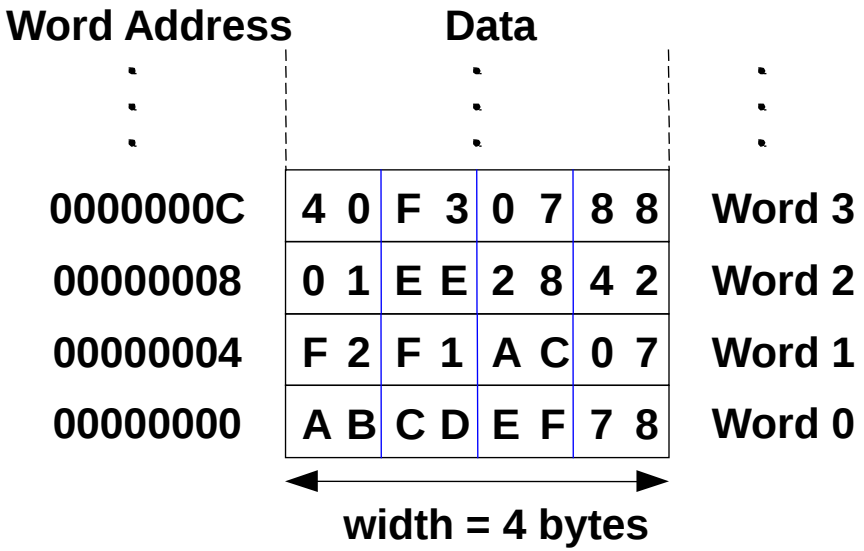
op	rs	rt	imm
41	0	12	14

op	rs	rt	imm
101001	00000	01100	0000000000001110

0xA40C000E

Instruções Load/Store

- **Operandos em Memória**
  - **Memória como *byte* endereçável**
- **Lendo um único byte**
  - » Load e store um único bytes: load byte (`lb`) e store byte (`sb`)
  - » Cada word de 32-bit tem 4 bytes, assim o endereço deve ser incrementado de 4



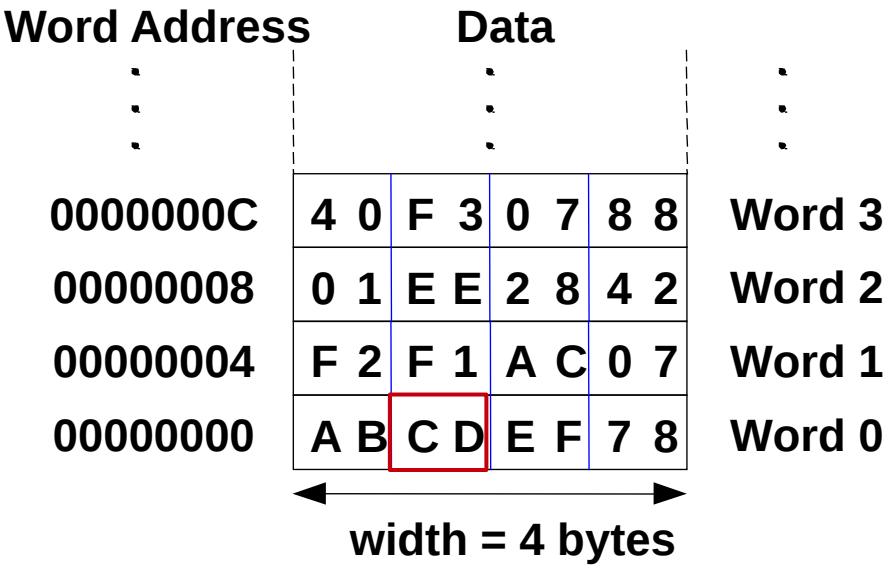
Instruções Load/Store

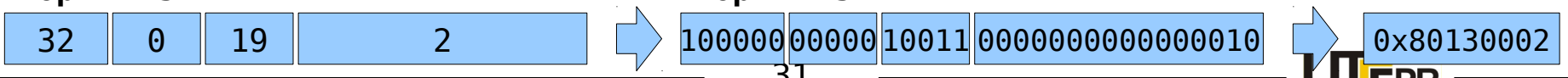
- **Lendo um byte da Memória**
  - Instrução *Load Byte:* lb
  - Opcode: 100000 (32), I-type

Código Assembly MIPS

```
lb $s3, 2($0)  # read memory byte 2 of word 0 into $s3
```

$s3: FFFFFFCD



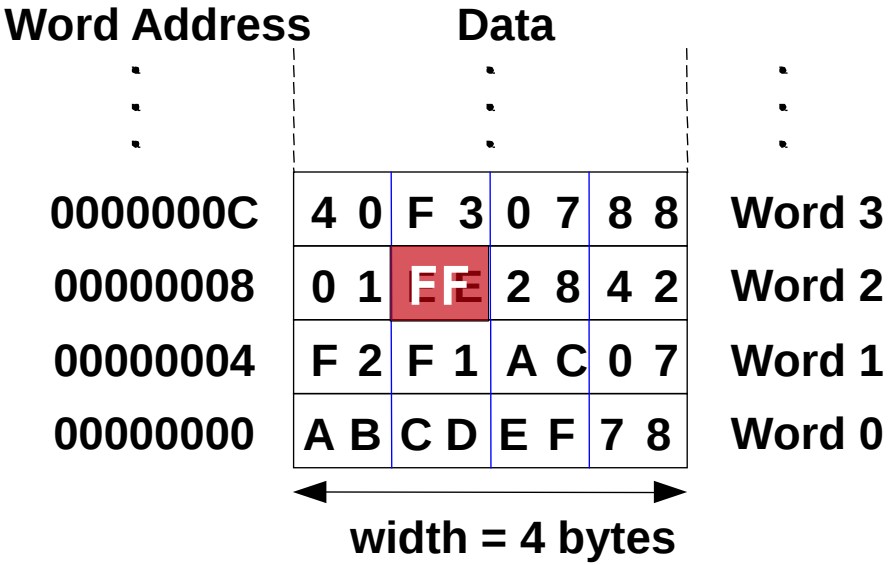


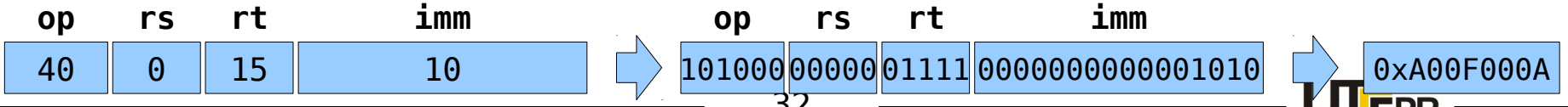
Instruções Load/Store

- **Escrevendo um byte na Memória**
  - **Instrução Store *Byte:* sb**
  - **Opcode: 101000 (40), I-type**

Código Assembly MIPS **$t7: FFFFFFFF**

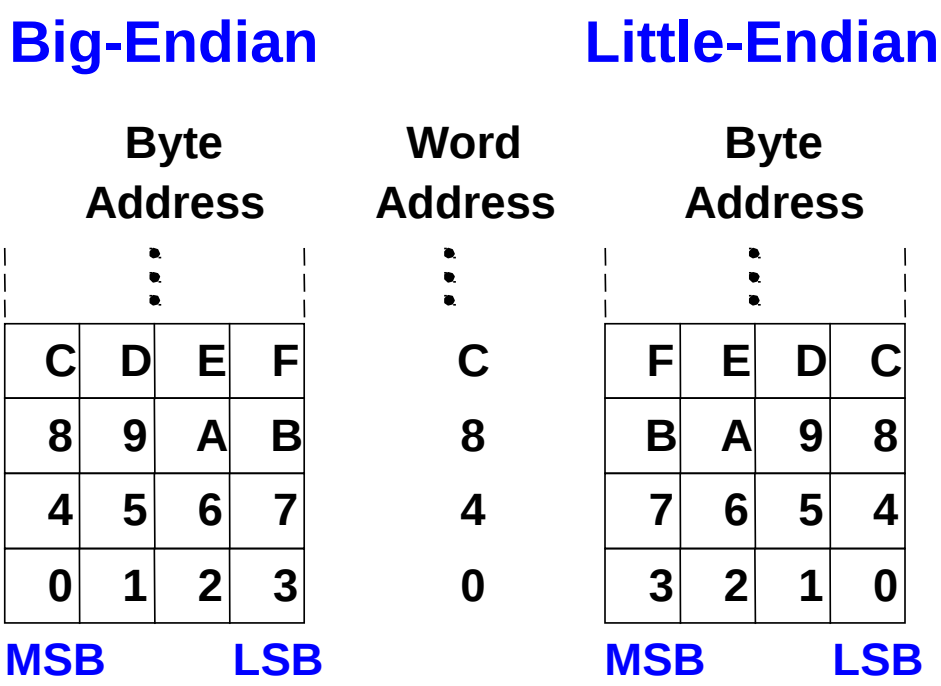
```
sb $t7, 10($0) # write $t7 into memory byte 2 of word 2
```





Instruções MIPS

- **Big-Endian e Little-Endian**
  - Como são numerados os bytes na word



Instruções MIPS

- **Big- e Little-Endian Exemplos:**

- **Suponha que inicialmente `$t0` contém 0x23456789. Após o seguinte trecho de programa ser executado em um sistema big-endian, qual o valor de `$s0`. E em um sistema little-endian?**

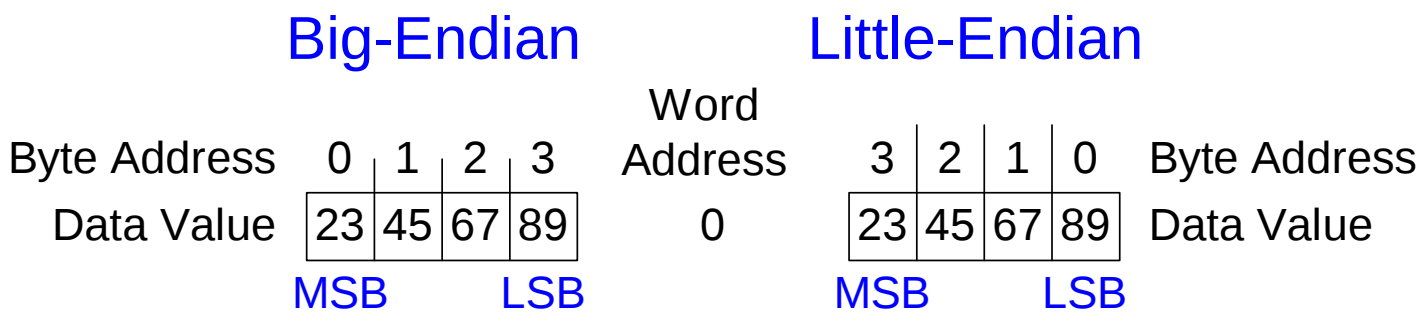
```
sw $t0, 0($0)
lb $s0, 1($0)
```

Instruções MIPS

- **Big- e Little-Endian Exemplos:**
  - **Suponha que inicialmente `$t0` contém 0x23456789. Após o seguinte trecho de programa ser executado em um sistema big-endian, qual o valor de `$s0`. E em um sistema little-endian?**

```
sw $t0, 0($0)
lb $s0, 1($0)
```

  - **Big-endian:** 0x00000045
  - **Little-endian:** 0x00000067



Instruções MIPS

- **Operandos: Constantes/Imediatos**
  - **Um imediato é um número de 16-bit em complemento de dois.**

High-level code

```
a = a + 4;
b = a – 12;
```

MIPS assembly code

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

Acesso a elementos de um *Array*

Supondo a existência de um *array* A de inteiros (inteiro: 32 bits)

Código C: A[300] = h + A[300];

Código MIPS:

```
lw   $t0, 1200($t1)
add  $t0, $s2, $t0
sw   $t0, 1200($t1)
```

op	rs	rt	rd	address/shamt	address/funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

Modos de Endereçamento

Especificam os operandos usados numa operação. O processador MIPS tem os modos de endereçamento:

1) **Endereçamento de Registrador (Register):**
   - Operando é um registrador
   - a instrução especifica diretamente o registrador a ser acessado.
2) **Endereçamento Imediato:**
   - a instrução não especifica o endereço do operando, mas o próprio operando, que está disponível imediatamente
3) **Endereçamento por base ou deslocamento *(Base Addressing)*:**
   - o operando é uma localização de memória cujo endereço é a soma de um registrador e uma constante na instrução
   - Um registrador com o endereço de base é utilizado.
4) **Endereçamento relativo ao PC *(PC-Relative)*:**
   - O endereço é a soma de PC e uma constante da instrução
5) **Endereçamento pseudodireto *(Pseudo Direct)*:**
   - onde o endereço de desvio (26 bits) é concatenado com os 4 bits mais significativos do PC

Modos de Endereçamento

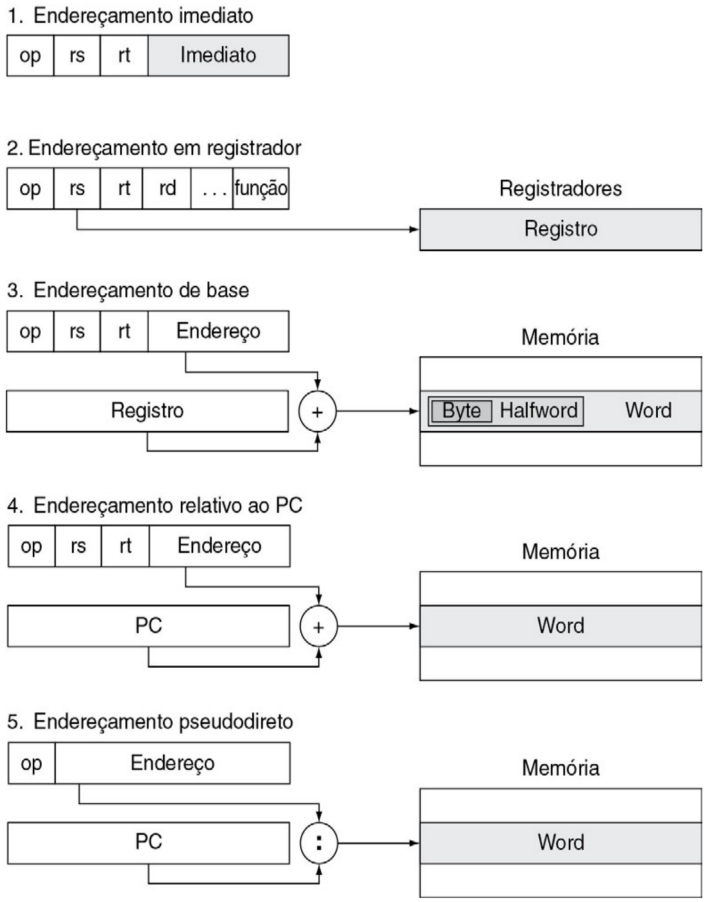
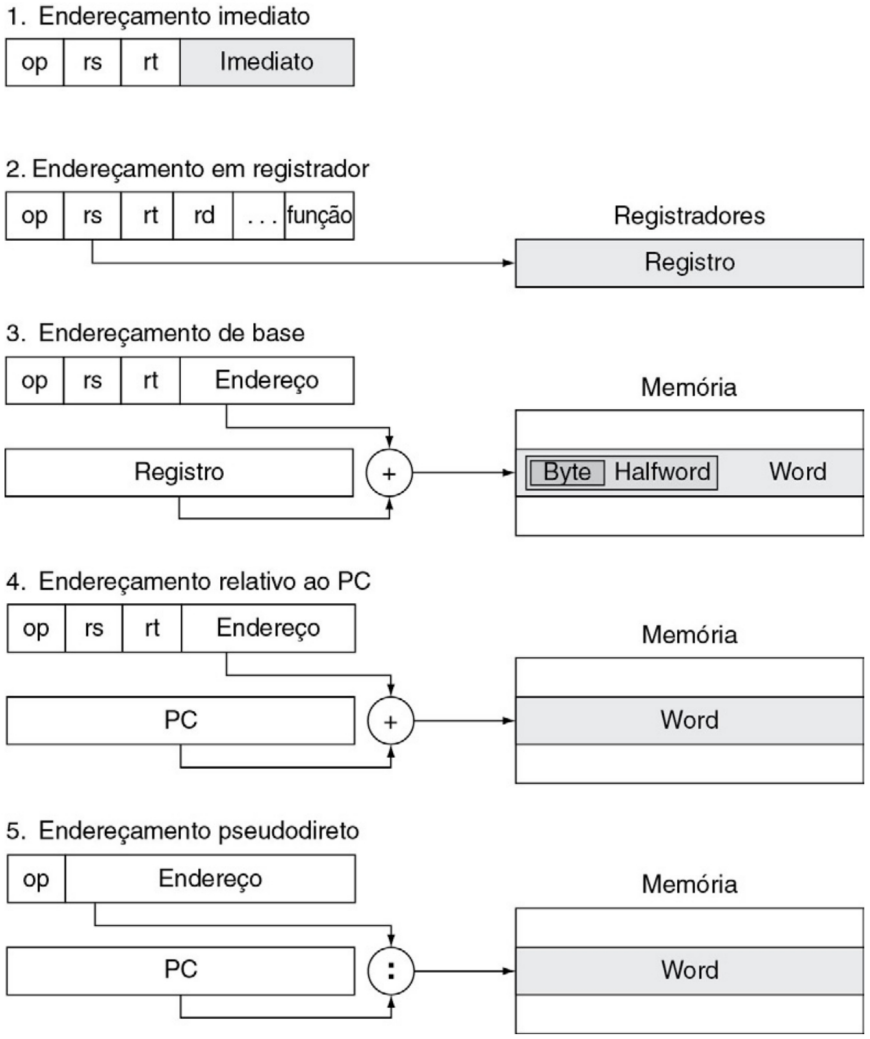


FIGURA 2.18 Ilustração dos cinco modos de endereçamento do MIPS. Os operandos estão sombreados na figura. O operando do modo 3 está na memória, enquanto o operando para o modo 2 é um registrador. Observe que as versões de load e store acessam bytes, halfwords ou palavras. Para o modo 1, o operando é formado pelos 16 bits da própria instrução. Os modos 4 e 5 endereçam as instruções na memória, com o modo 4 adicionando um endereço de 16 bits deslocado à esquerda em 2 bits ao PC, e o modo 5 concatenando um endereço de 26 bits deslocado à esquerda em 2 bits com os 4 bits superiores do PC.

Modos de Endereçamento



Modos de Endereçamento

Endereçamento de Registrador

- Os operandos estão somente Registradores
  - Exemplo: `add $s0, $t2, $t3`
  - Exemplo: `sub $t8, $s1, $0`

Endereçamento Imediato

- Imediato de 16-bit é usado como operando
  - Exemplo: `addi $s4, $t5, -73`
  - Exemplo: `ori  $t3, $t7, 0xFF`

Modos de Endereçamento

Endereçamento por Base ou deslocamento (Base Addressing):

- **O endereço do operando é:**
  - **`base address + sign-extended immediate`**
  - **Exemplo:** **`lw  $s4, 72($0)`**
    - Address = `$0 + 72`
  - **Exemplo:** **`sw  $t2, -25($t1)`**
    - Address = `$t1 - 25`

Modos de Endereçamento

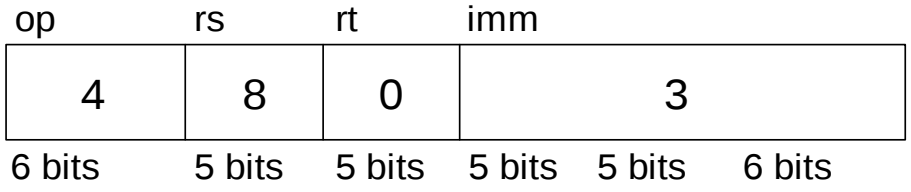
Endereçamento PC-Relative

```
0x10              beq   $t0, $0, else
0x14              addi  $v0, $0, 1
0x18              addi  $sp, $sp, i
0x1C              jr    $ra
0x20      else:   addi  $a0, $a0, -1
0x24              jal   factorial
```

Assembly Code

```
beq $t0, $0, else
(beq $t0, $0, 3)
```

Field Values



Modos de Endereçamento

Endereçamento *Pseudo-direct*

```
0x0040005C          jal   sum
...
0x004000A0  sum:    add   $v0, $a0, $a1
```

JTA 0000 0000 0100 0000 0000 0000 1010 0000 (0x004000A0)

26-bit addr 0000 0000 0100 0000 0000 0000 1010 0000 (0x0100028)

0 1 0 0 0 2 8

Field Values

op	imm
3	0x0100028
6 bits	26 bits

Machine Code

op	addr	
000011	00 0001 0000 0000 0000 0010 1000	(0x0C100028)
6 bits	26 bits	

Programa Armazenado

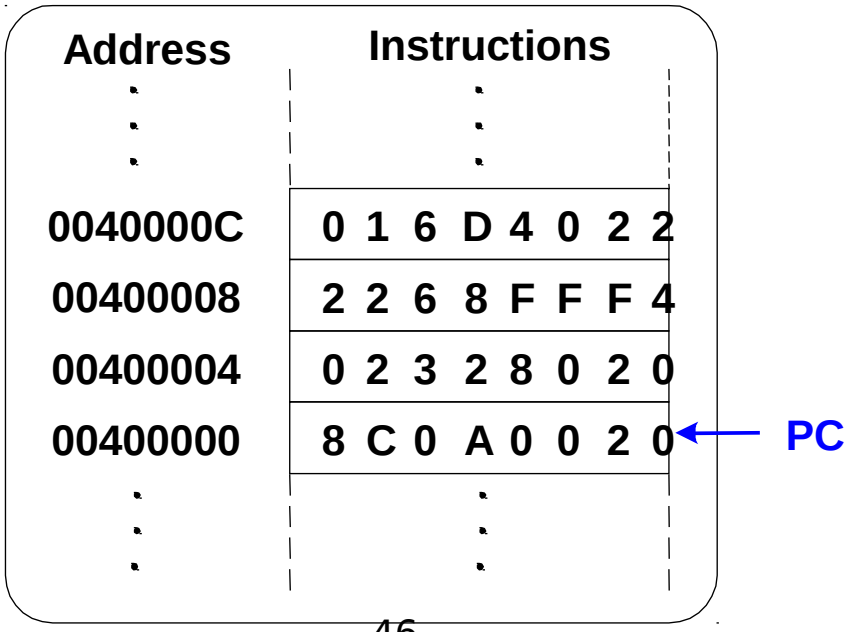
- **Instruções e dados de 32-bit armazenados na memória**
- **Seqüência de instruções: é a única diferença entre dois programas**
- **Execução de um novo programa:**
  - **Simplesmente armazene o novo programa na memória**
- **Execução do programa pelo hardware do processador:**
  - **Busca as instruções da memória em seqüência (*fetches* → reads)**
  - **Decodifica e Executa a operação especificada.**
- **Um *program counter* (PC) indica a instrução corrente (ou a próxima instrução).**
- **no MIPS, programas tipicamente iniciam no endereço de memória 0x00400000.**

Programa Armazenado

- **Exemplo:**

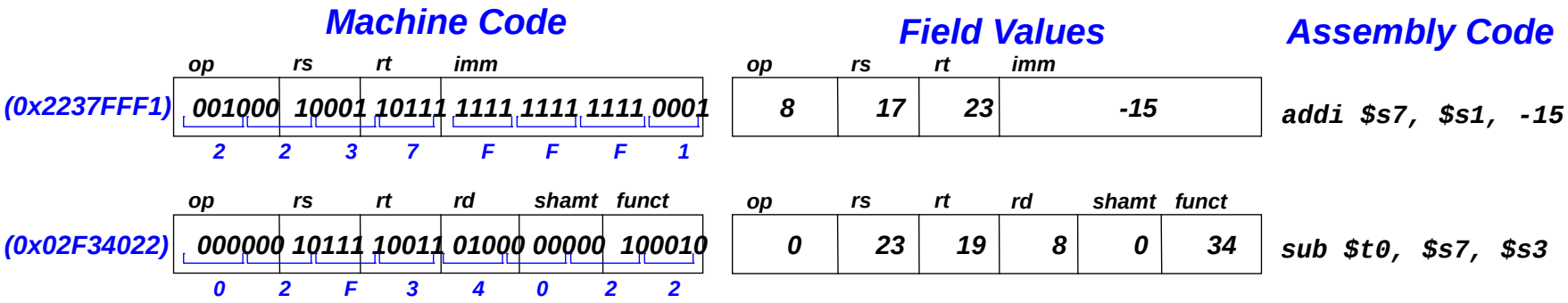
Assembly Code	Machine Code
`lw $t2, 32($0)`	`0x8C0A0020`
`add $s0, $s1, $s2`	`0x02328020`
`addi $t0, $s3, -12`	`0x2268FFF4`
`sub $t0, $t3, $t5`	`0x016D4022`

Stored Program



Interpretando o Código de Máquina

```
Inicia-se com o opcode
Opcode informa como fazer o parse dos bits remanecentes
Se opcode é todo 0's
  R-type instruction
  Os bits Function informam qual instrução é
Caso contrário
  Opcode informa qual é a instrução
```



Exercícios

Implemente os programas em A*ssembly* do MIPS para resolver as expressões:

a) a = b + c

b) f = (g + h) – (i + j)

Exercícios

Suponha que *h* seja associado com o registrador $s2 e o endereço base do *array A* armazenado em $s3. Qual o código MIPS para o comando A[12] = h + A[8];?

Exercícios

Suponha que *h* seja associado com o registrador $s2 e o endereço base do *array A* armazenado em $s3. Qual o código MIPS para o comando A[12] = h + A[8];?

Solução

```
lw    $t0,32($s3)  #  $t0 recebe A[8]
add   $t0,$s2,$t0  #  $t0 recebe h + A[8]
sw    $t0,48($s3)  # armazena o resultado em A[12]
```

Exercícios

Qual o código MIPS para carregar uma constante de 32 bits no registrador $s0 ?

0000 0000 0011 1101 0000 1001 0000 0000

Exercícios

Qual o código MIPS para carregar uma constante de 32 bits no registrador $s0 ?

0000 0000 0011 1101 0000 1001 0000 0000

Solução

```
lui   $s0,61          # 61₁₀ = 0000 0000 0011 1101₂
ori   $s0,$s0,2304    # 2304₁₀ = 0000 1001 0000 0000₂

0000 0000 0011 1101 0000 1001 0000 0000₂
```

Leitura Recomendada

Capítulo 2

PATTERSON, David A.; HENNESSY, John L. ***Organização e projeto de computadores: a interface hardware/software.*** Rio de Janeiro, RJ: Elsevier, 2005. 484 p. ISBN 9788535215212.

Resumo da Aula de Hoje

Tópicos mais importantes:

Linguagem Assembly

Microprocessador MIPS

Registradores

Formatos de Instruções

Tipos de Instruções

Modos de Endereçamento