Arquitetura de Computadores

Linguagem de máquina (Continuação)

Prof. Tiago Gonçalves Botelho

Instruções Básicas: A Linguagem da Máquina

- **Objetivos:** estudar as instruções de uma máquina real por meio de códigos escritos em linguagem de montagem e de alto nível.

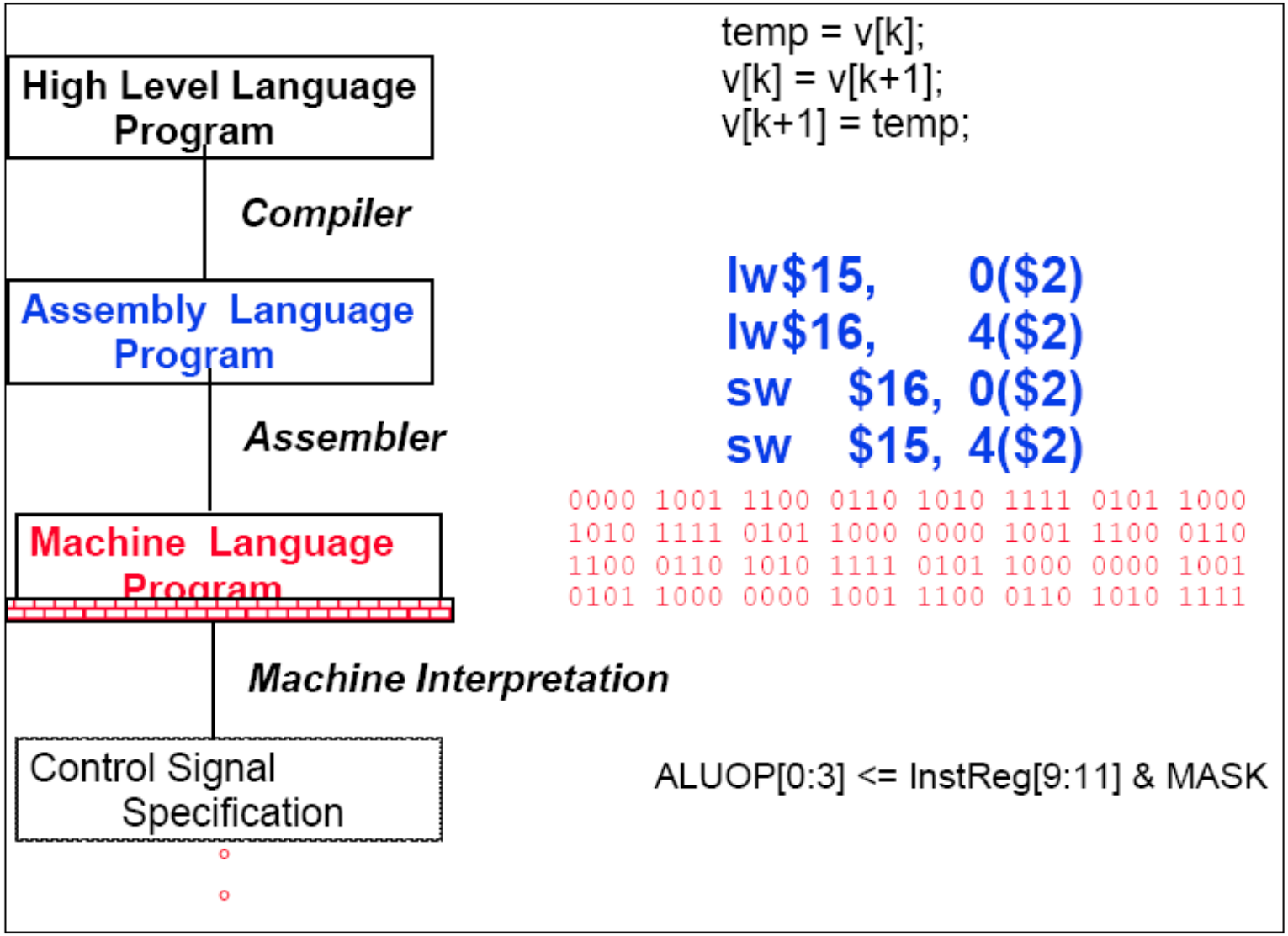
- **Metodologia:** estudar a linguagem de máquina passo a passo até chegar a aspectos mais elaborados.

- **Justificativa:** construir um conjunto de instruções que facilite o projeto tanto de hardware quanto de software e que maximize a performance e minimize os custos.

Instruções Básicas: A Linguagem da Máquina

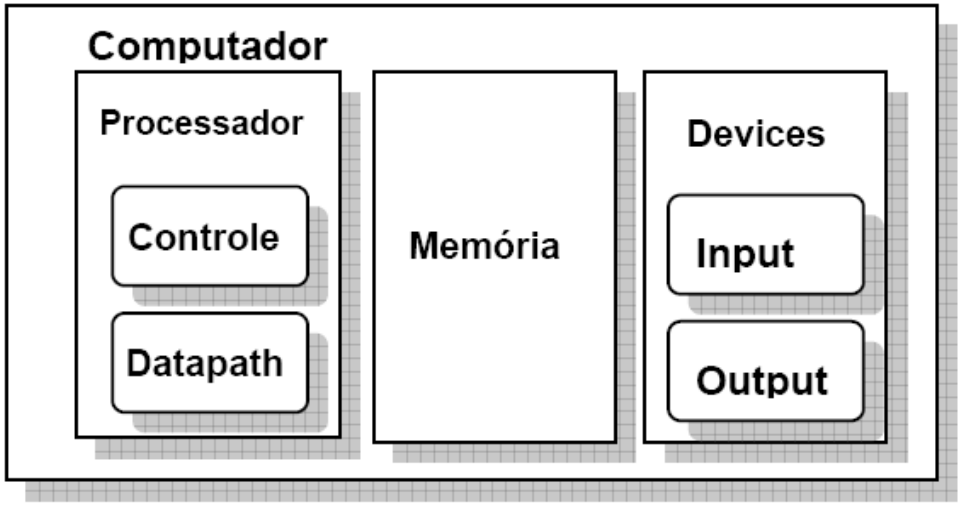
- Arquitetura do Computador = Conjunto de Instruções (ISA) + Organização da Máquina

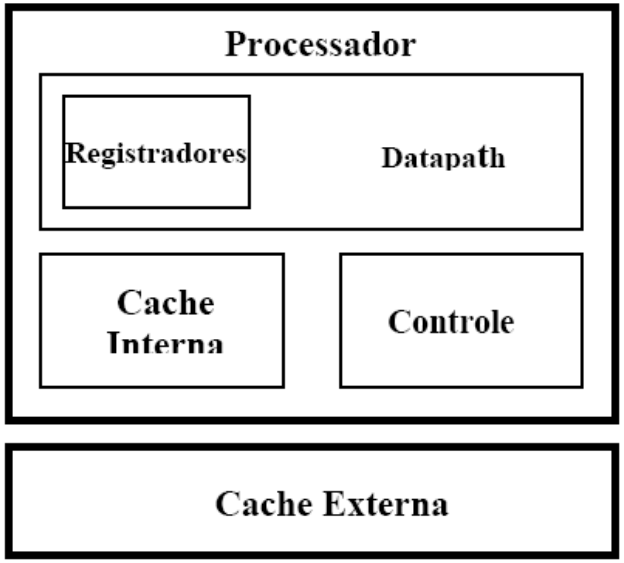
Instruções Básicas: A Linguagem da Máquina



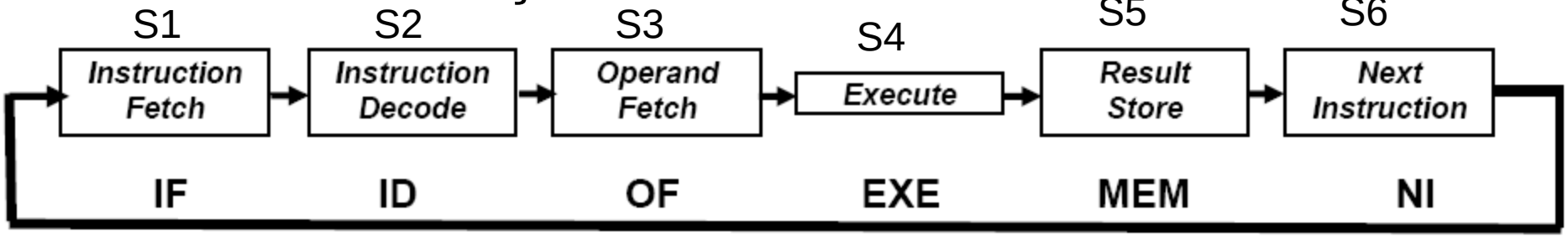
Instruções Básicas: A Linguagem da Máquina

- Organização da máquina





- Ciclo de execução



Instruções Básicas: Processador MIPS

- MIPS Comp. Systems, Inc. → Stanford, 1984

Microprocessor without Interlocked Pipeline Stages/

Microprocessador sem estágios inter-bloqueados de pipeline)

- 1992 → comprada pela Silicon Graphics Inc.
- 1998 → MIPS Technologies Inc. em 1998
- Atualmente produz outros dispositivos eletrônicos
- Características do processador MIPS:
  - Implementa um pequeno conjunto de instruções simples
  - Cada instrução é executada em um único ciclo de clock
  - Utiliza a técnica de pipeline
  - Utiliza 32 registradores de 32 bits

Instruções Básicas: MIPS - ISA

- 111 instruções, representadas em 32 bits:
  - 21 Instruções aritméticas (+, -, *, /, %)
  - 8 Instruções lógicas (&, |, ~)
  - 8 Instruções de manipulação
  - 12 Instruções de comparação (>, <, =, >=, <=, ¬)
  - 25 Instruções branch/jump
  - 15 Instruções de load
  - 10 Instruções de store
  - 8 Instruções de move
  - 4 Instruções diversas

Instruções Básicas: MIPS - Instruções Aritméticas

- Soma: (três operandos)

```
add a,b,c  # A soma do conteúdo dos registradores b e c é
           # colocada no registrador a
```

- Subtração: (três operandos)

```
sub a,b,c # A subtração do conteúdo dos registradores b e c
          # é colocada no registrador a
```

- Exemplos

Seja o seguinte segmento de código escrito em C que contém as seguintes variáveis a, b, c, d e e:

a = b + c ➔ `add a, b, c`

d = a – e ➔ `sub d, a, e`

Instruções Básicas: MIPS - Instruções Aritméticas

- Exemplos

Seja o seguinte segmento de código escrito em C que contém as seguintes variáveis a, b, c, d e e. Como codificar em MIPS?

a = b + c + d + e

```
add a, b, c #a=b+c
add a, a, d #a=b+c+d
add a, a, e #a=b+c+d+e
```

Instruções Básicas: MIPS - Instruções Aritméticas

- Seja o código em C:

  f = (g + h) – (i + j)

- Desenvolver código equivalente em MIPS:
  - a)Código utilizando o menor número de variáveis
  - b)Refazer de forma a preservar valor inicial de variáveis

Instruções Básicas: MIPS - Instruções Aritméticas

- Resposta usando registradores temporários:

```
add t0, g, h  # t0 é uma variável temporária.
add t1, i, j  # t1 é uma variável temporária.
sub f, t0,t1  # f recebe (g + h) – (i + j)= (t0 – t1)
```

- O MIPS possui 32 registradores de 32 bits que são numerados da seguinte forma:
  - Registradores de **variáveis**: **`$s0,$s1,$s2,...`**
  - Registradores **temporários**: **`$t0,$t1,$t2,...`**

Instruções Básicas: MIPS - Instruções Aritméticas

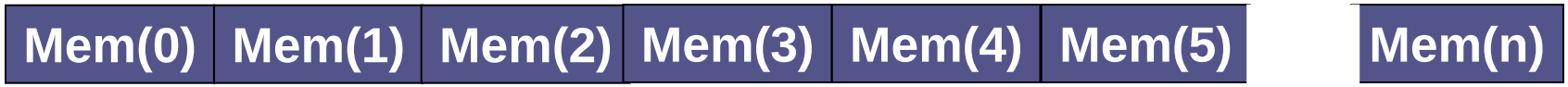
- Operações aritméticas somente entre registradores
- Necessita de instruções que transfiram dados entre a memória e os registradores
- Por que não utilizar mais do que 32 registradores?

- Usando o exemplo anterior:

```
add $t0,$s1,$s2  # $t0 contém g + h
add $t1,$s3,$s4  # $t1 contém i + j
sub $s0,$t0,$t1  # f recebe $t0 - $t1,
                 # ou (g + h) – (i + j)
```

Instruções Básicas: MIPS - Instruções Aritméticas

- A memória → um vetor unidirecional acessado por meio de endereços



- Endereço de memória → formado por uma constante associada ao conteúdo de um registrador:

  Constante($Registrador)

- A função de transferência de dados da memória para o registrador é conhecida como **load**

- MIPS → **load word** ou simplesmente **lw**

Instruções Básicas: MIPS - Instruções Aritméticas

- A função de transferência de dados do registrador para a memória é conhecida como **store**
- MIPS → **store word** ou simplesmente **sw**
- **Formatos:**

lw $t0, Constante($S0)

sw $S1, Constante($S2)

Instruções Básicas: MIPS - Instruções Aritméticas

- Suponha que A seja um vetor de 100 palavras, e que o compilador tenha associado as variáveis g e h aos registradores $s1 e $s2. O endereço inicial do vetor, ou endereço base, está armazenado em $s3. Escreva o código C na linguagem do MIPS.

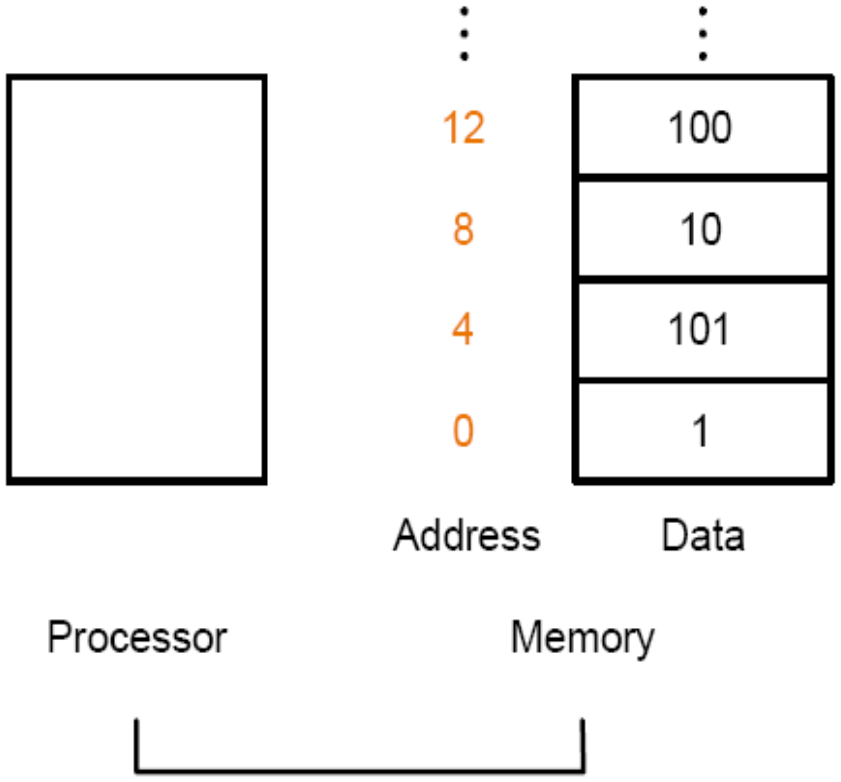
`g = h + A[8]`

O valor de A[8] está na memória e deve ser transferido para um registrador:

```
lw  $t0, 8($s3)      # registrador $t0 recebe A[8]
add $s1, $s2, $t0 # g recebe h + A[8]
```

Instruções Básicas: MIPS - Instruções Aritméticas

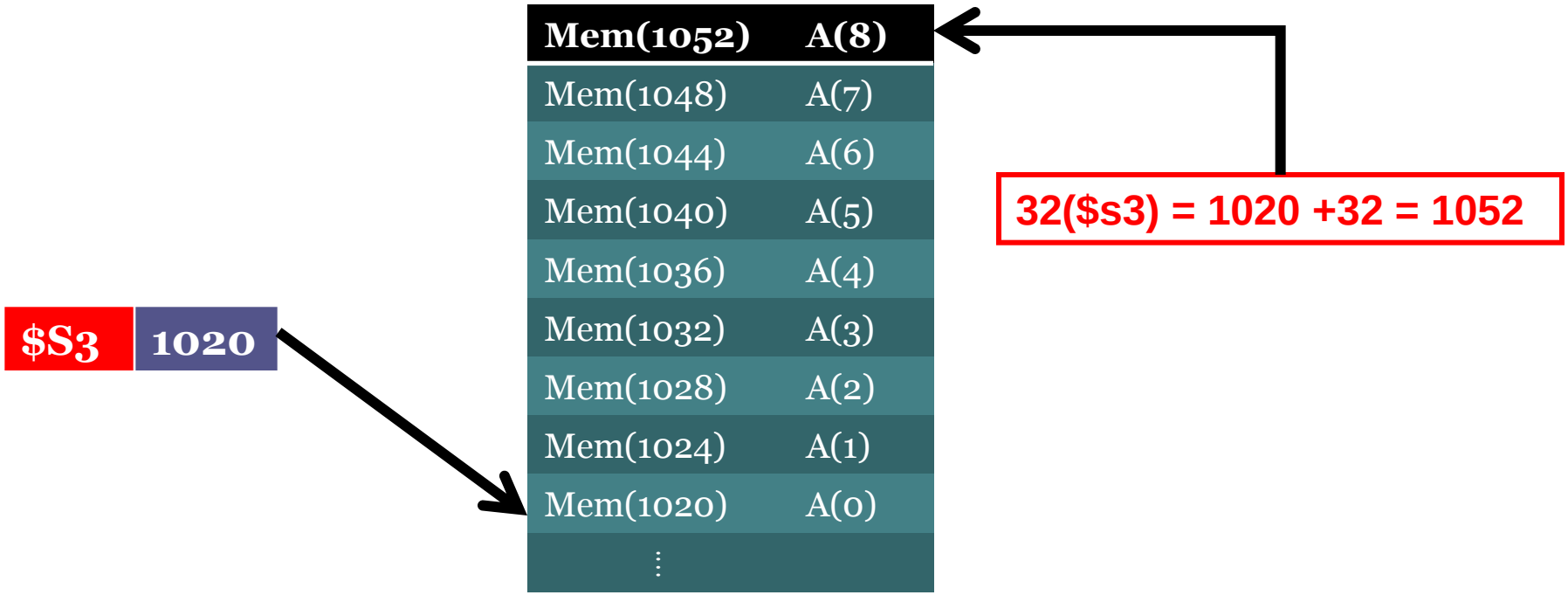
- Constante da instrução → deslocamento
- O registrador cujo valor armazenado é somado a essa constante é chamado de registrador base
- MIPS → armazena palavras de 32 bits (4 bytes)
- Endereçamento de memória é sempre múltiplo de 4



Instruções Básicas: MIPS - Instruções Aritméticas

- Para obter o correto deslocamento no registrador base $s3, deveríamos usar 8X4 = 32, dessa forma seria deslocado para elemento A[8]:

```
lw  $t0, 32($s3)   # registrador $t0 recebe A[8]
add $s1, $s2, $t0 # g recebe h + A[8]
```



Instruções Básicas: MIPS - Instruções Aritméticas

- Suponha que h está associado ao registrador $s2 e que o endereço base do vetor A esteja armazenado em $s3. Escreva o código de montagem para:

A[12] = h + A[8]

Instruções Básicas: MIPS - Instruções Aritméticas

- Resultado para a expressão em C (A[12] = h + A[8]):

```
lw  $t0, 32($s3)    # registrador temporário $t0 recebe A[8]
add $t0, $s2, $t0   # registrador temporário $t0 recebe h + A[8]
sw  $t0, 48($s3)    # h + A[8] é armazenado de volta para a
                    # memória em A[12]
```

Resumo das instruções

Categoria	Instrução	Exemplo	Significado	Comentário
Aritmética	add	add $s1, $s2, $s3	$s1= $s2 + $s3	3 operandos; dados em registradores
	sub	sub $s1, $s2, $s3	$s1= $s2 - $s3	3 operandos; dados em registradores
Transferência de dados	load word	lw $s1, 100($s2)	$s1= Memória[$s2 +100]	Dados transferidos da memória para registradores
	store word	sw $s1, 100($s2)	Memória[$s2 + 100] = $s1	Dados transferidos do registrador para a memória

- Instruções com constantes: addi $s1, $s1, 23

Instrução do Tipo R

- **Tipo R – registrador:**

Op	Rs	Rt	Rd	Shamt	Funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
31 26	25 21	20 16	15 11	10 6	5 0

- **Op** – operação básica a ser realizada pela instrução→ *Opcode*.
- **Rs** – registrador que contém o primeiro operando fonte.
- **Rt** - registrador que contém o segundo operando fonte.
- **Rd** - registrador que guarda o resultado da operação→ registrador-destino.
- **Shamt** – quantidade de bits deslocados.
- **Funct** – especifica uma variação da operação apontada no campo **op**, conhecida como código de função.

Exemplo: add Rd, Rs, Rt

Instrução do Tipo R

- **Tipo R – registrador**

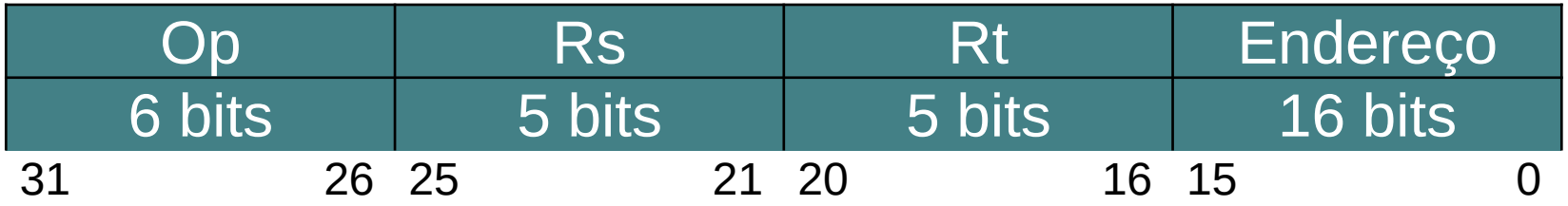
Op	Rs	Rt	Rd	Shamt	Funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits
31 26	25 21	20 16	15 11	10 6	5 0

- **Exemplo:** `add $t0, $S0, $S3`

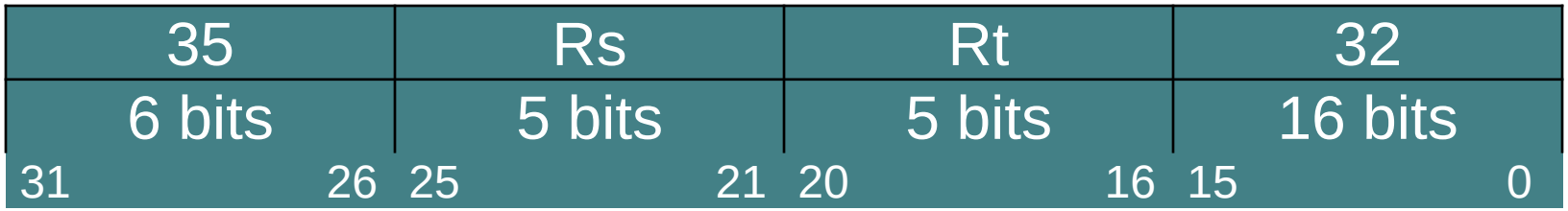
Op= 0	Rs=16=S_0	Rt=19=S_3	Rd=8=t_0	Shamt=0	Funct=32
000000	10000	10011	01000	00000	100000
31 26	25 21	20 16	15 11	10 6	5 0

Instrução do Tipo I

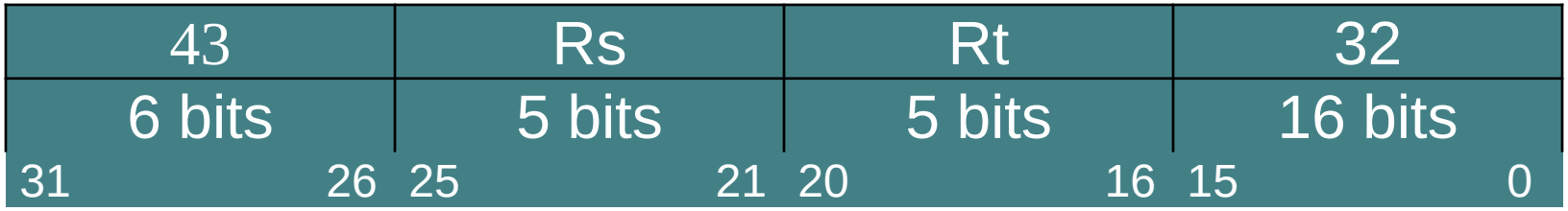
- **Tipo I – Imediato:**



Exemplo: lw Rt, 32(Rs)



Exemplo: sw Rt, 32(Rs)



Codificação de instruções MIPS

Instrução	Formato	op	rs	rt	rd	Shamt	Funct	Endereço
add	R	0	reg	reg	reg	0	32	n.a.
sub	R	0	reg	reg	reg	0	34	n.a.
add immediate (addi)	I	8	reg	reg	n.a.	n.a.	n.a.	constante
lw (load)	I	35	reg	reg	n.a.	n.a.	n.a.	endereço
sw (store)	I	43	reg	reg	n.a.	n.a.	n.a.	endereço

Tradução completa de instrução

- Linguagem de alto nível: A[300] = h + A[300]
  - Considerar A[ ] em $t1, h em $s2
- Linguagem assembly:

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

- Linguagem de máquina (ainda em decimal):

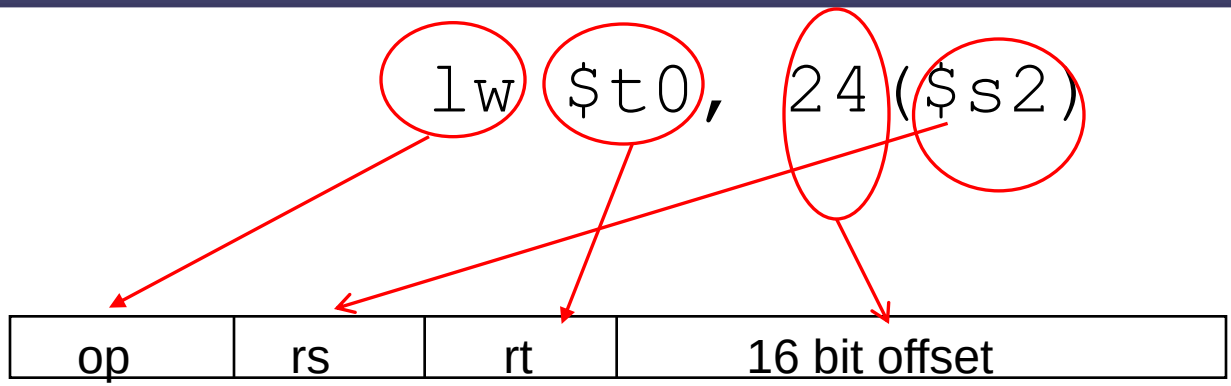
op	rs	rt	rd	Endereço/shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

Tradução completa de instrução

- Linguagem de máquina. Resultado final em binário:

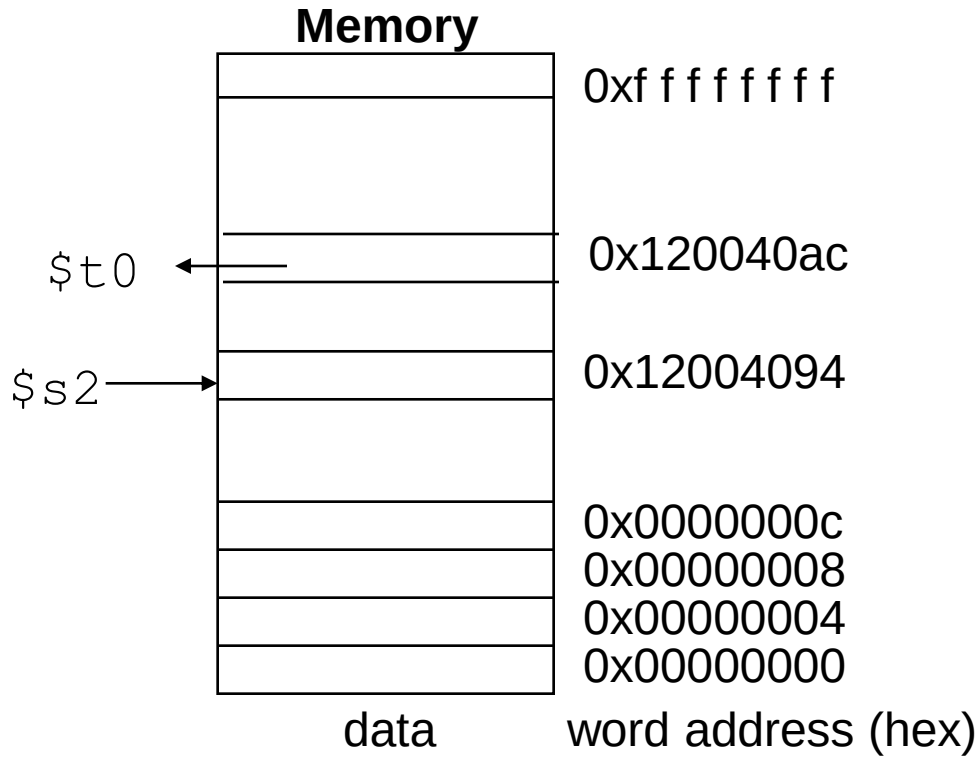
op	Rs	rt	rd	Endereço/shamt	funct
100011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
101011	01001	01000	0000 0100 1011 0000		

Formato das Instruções: Load/Store



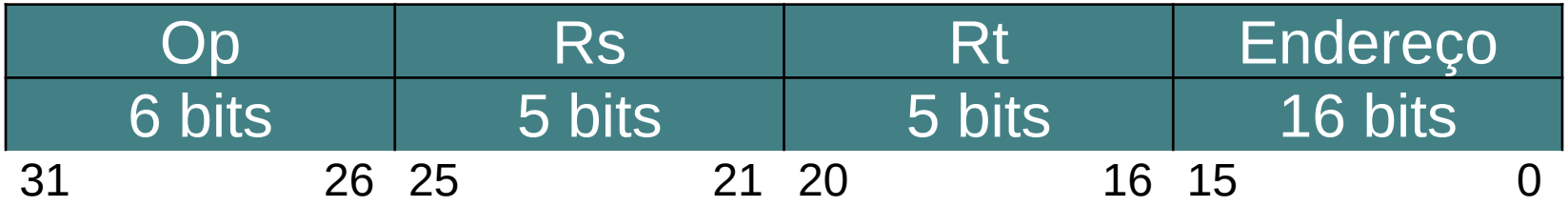
24_{10} + $s2 =

. . . 0001 1000
+ . . . 1001 0100
. . . 1010 1100 =
0x120040ac



Instrução do Tipo Desvio (Branch)

- **Tipo I – Imediato:**



- Exemplo 1: Branch equal:

```
beq Rs, Rt, endereço
beq Registrador1, Registrador2, L1
```

- Exemplo 2: Branch not equal:

```
bne Rs, Rt, endereço
bne Registrador1, Registrador2, X20
```

Instrução do Tipo Desvio (Branch)

- **Instruções Branch em Assembly**

Comando	Exemplo	Comentários
blt	blt Rs1, Rs2, Label	#vá para Label Se Rs1 < Rs2
bgt	bgt Rs1, Rs2, Label	#vá para Label Se Rs1 > Rs2
ble	ble Rs1, Rs2, Label	#vá para Label Se Rs1 <= Rs2
bge	bge Rs1, Rs2, Label	#vá para Label Se Rs1 >= Rs2

Instrução Desvio Incondicional (Jump)

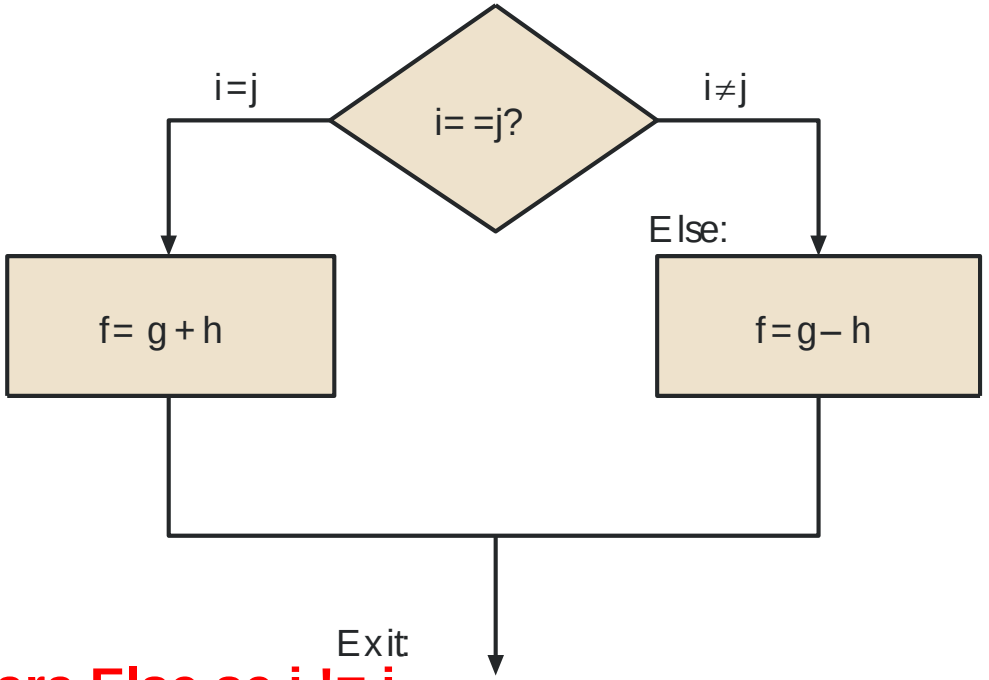
- **Tipo J – Jump:**

- Exemplo1: J Endereço

Instrução do Tipo Desvio (Branch)

- Exemplo: Seja o código C abaixo:

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




```
bne $s3, $s4, Else;          # desvia para Else se i != j
add $s0, $s1, $s2;       # f = g + h
j Exit                            # desvia para a saída
Else: sub $s0, $s1, $s2;  # f = g - h , salta se i == j
                                  # e Else label com o
Exit:                  # endereço de desvio
```

Instrução do Tipo Desvio (Branch)

- Altere o exemplo anterior para trabalhar com beq:

```
beq $s3, $s4, IF;          # desvia para IF se i == j
sub $s0, $s1, $s2;         # f = g - h
j Exit                        # desvia para a saída
IF: add $s0, $s1, $s2;   # f = g + h , salta se i != j
                              # e IF label com o
Exit:                   # endereço de desvio
```

Instrução do Tipo Desvio (Branch)

- Exercício: Desenvolva um código utilizando o MIPS que receba a nota de um aluno e armazene 1 caso o aluno seja aprovado e 0 caso seja reprovado. Considere nota >=6 para aprovação.

Loop While

Exemplo: Seja o código em linguagem C: Suponha que i e k correspondam aos registradores $s3 e $s5, a base do array save esteja em $s6 e j em $s4.

```
while (save [ i ] == k)
        i = i + j;
```

```
Loop:  add  $t1, $s3, $s3   # t1 ← 2* i
       add  $t1, $t1, $t1    # t1← 4 * i
       add  $t1, $t1, $s6    # t1← endereço de Save[ i ]
       lw    $t0, 0( $t1)    # t0← Save[ i ]

        bne $t0, $s5, Exit   # desvia para Exit se save[ i ] != k
        add $s3, $s3, $s4        # i ← i + j
       j Loop                # desvia para Loop
Exit:
```

Loops: a utilização de vetores com índice variável

Exemplo: Seja o loop programado na linguagem C:

```
do{
        g = g + A[ i ];
        i = i + j;
} while(i!=h)
```

Suponha que A seja um vetor de 100 posições e para g, h, i e j são associados os registradores: $s1 → $s4. O vetor base está armazenado em $s5.

Desenvolver o código em assembly do MIPS

Loops: a utilização de vetores com índice variável


```
Loop: add  $t1, $s3, $s3    # t1 ← 2 * i
      add  $t1, $t1, $t1    # t1← 4 * i
      add  $t1, $t1, $s5    # t1← endereço de A[ i ]
      lw   $t0, 0( $t1)     # t0← A[ i ]

      add  $s1, $s1, $t0    # g ← g + A[ i ]
      add  $s3, $s3, $s4    # i ← i + j

      bne $s3, $s2, Loop    # desvia para o Loop se i != h
Exit:
```

Compilação do teste “Menor do que”

- A instrução ***set on less than*** ou ***slt*** é usada para testar dois valores
- Esta instrução compara os valores de dois registradores e atribui o valor 1 a um terceiro se o valor do primeiro registrador for menor que o segundo; caso contrário atribui zero
- Para criar facilidades de programação um registrador denominado de **$zero** é mapeado no registrador 0

```
slt  $t0, $s3, $s4 # Se $s3 < $s4 → $t0 = 1
```

Compilação do teste “Menor do que”

Exemplo: Teste se uma variável a associada ao registrador $s0 é menor que b (reg. $s1) e desvie para Less se a condição for verdadeira.

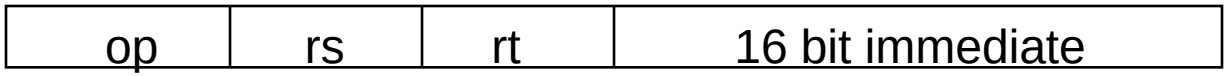
```
slt $t0, $s0, $s1       # Se a < b → $t0 = 1

bne $t0, $zero, Less    # desvia para Less se $t0 != 0
                        # isto se a < b
```

Operandos Imediatos

- Códigos tipicamente utilizam pequenas constantes
- Qual a abordagem para implementar constantes?
  - Carregar as constantes na memória e depois carregá-las em registrador
  - Criar registradores específicos para armazenar a constante (tal como o $zero)
  - Ter instruções especiais que contenham constantes !

```
addi $t1, $t1, 4  #$t1 = $t1 + 4
slti $t0, $s2,15 #$t0 = 1 if $s2<15
```



Formato I

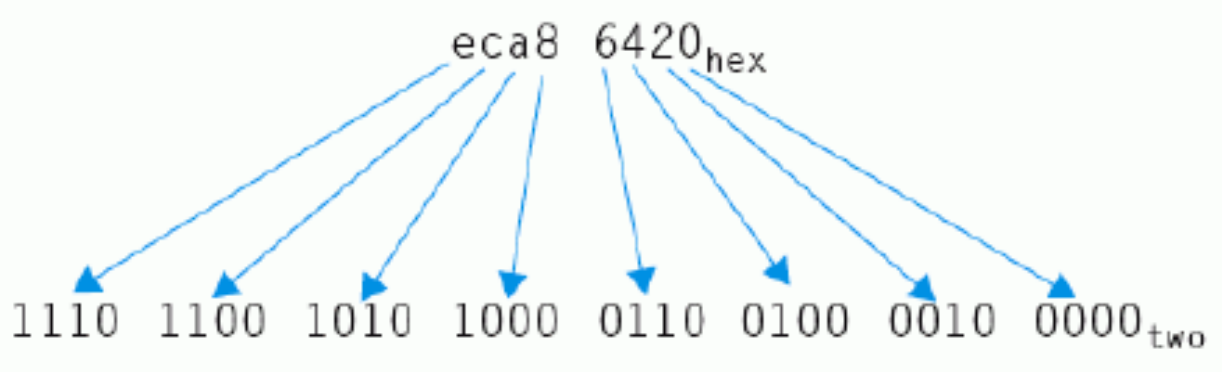
- A constante inserida dentro da própria instrução!
  - Imediato é limitado: $+2^{15}-1$ até -2^{15}

Transformação de Números

Binário para Hexadecimal

Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary	Hexadecimal	Binary
0_{hex}	0000_{two}	4_{hex}	0100_{two}	8_{hex}	1000_{two}	c_{hex}	1100_{two}
1_{hex}	0001_{two}	5_{hex}	0101_{two}	9_{hex}	1001_{two}	d_{hex}	1101_{two}
2_{hex}	0010_{two}	6_{hex}	0110_{two}	a_{hex}	1010_{two}	e_{hex}	1110_{two}
3_{hex}	0011_{two}	7_{hex}	0111_{two}	b_{hex}	1011_{two}	f_{hex}	1111_{two}

Hexadecimal para Binário



Representação de Binário sem Sinal

Hex (8+2 dígitos)	Binary (32 dígitos)	Decimal (Até 10)
0x00000000	0…0000	0
0x00000001	0…0001	1
0x00000002	0…0010	2
0x00000003	0…0011	3
0x00000004	0…0100	4
0x00000005	0…0101	5
0x00000006	0…0110	6
0x00000007	0…0111	7
0x00000008	0…1000	8
0x00000009	0…1001	9
	…	…
0xFFFFFFFC	1…1100	$2^{32} - 4$
0xFFFFFFFD	1…1101	$2^{32} - 3$
0xFFFFFFFE	1…1110	$2^{32} - 2$
0xFFFFFFFF	1…1111	$2^{32} - 1$

2^{31}	2^{30}	2^{29}	. . .	2^3	2^2	2^1	2^0	bit weight
31	30	29	. . .	3	2	1	0	bit position
1	**1**	**1**	**. . .**	**1**	**1**	**1**	**1**	**bit**

1 0 0 0 . . . 0 0 0 0 - 1

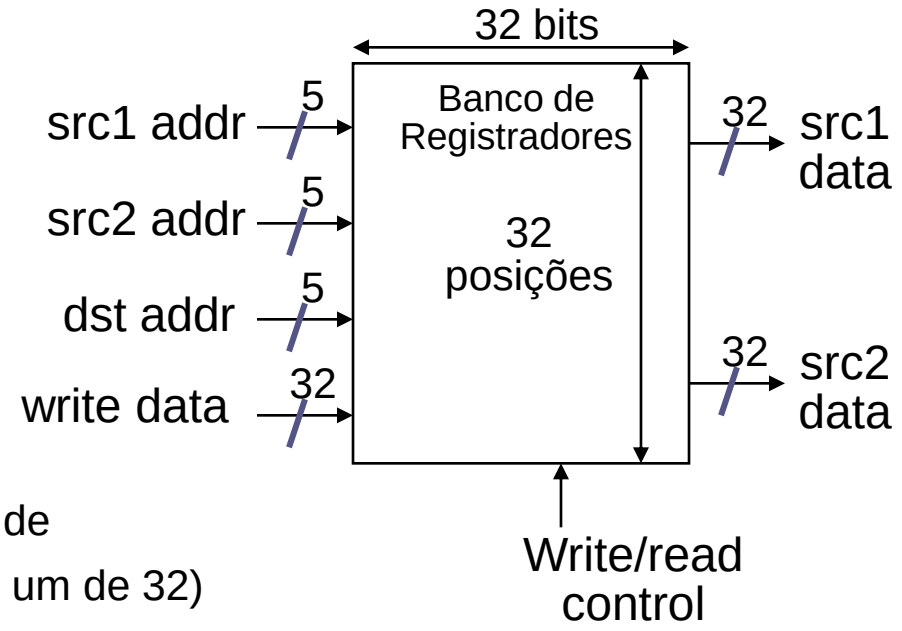
$2^{32} - 1$

MIPS Register Convention

Nome	Número do Registrador	Uso	Preservado na chamada?
$zero	0	constante 0 (hardware)	n.a.
$at	1	reservado p/ assembler	n.a.
$v0 - $v1	2-3	valores para resultados	não
$a0 - $a3	4-7	argumentos	sim
$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 retorno (hardware)	sim

MIPS: Banco de Registradores

- Posssui 32 registradores de 32 bits
  - Duas portas de leitura e
  - Uma porta de escrita



- Registradores são
- Mais rápidos do que memórias
- Mas se o banco de registradores for grande será mais lento (por exemplo, um banco de 64 registradores poderá ser 50% mais lento do que um de 32)
- Porta de Read/write impactam a velocidade quadraticamente
- Mais fácil de compilar e usar:
  - (A*B) – (C*D) – (E*F)
- Mantém as variáveis de forma que:
  - melhora a densidade do código (registradores são chamados com menos bits do que a memória)
  - Todas as máquinas (desde 1975) têm usado registradores de propósito geral

Referências

- Patterson, David A.; Hennessy, John; Organização e projeto de computadores: a interface hardware/software; 3ª ed.; Elsevier, 2005.
- Prof. Luiz Henrique Andrade Correia; Notas de aula.