

# Infraestrutura de Hardware

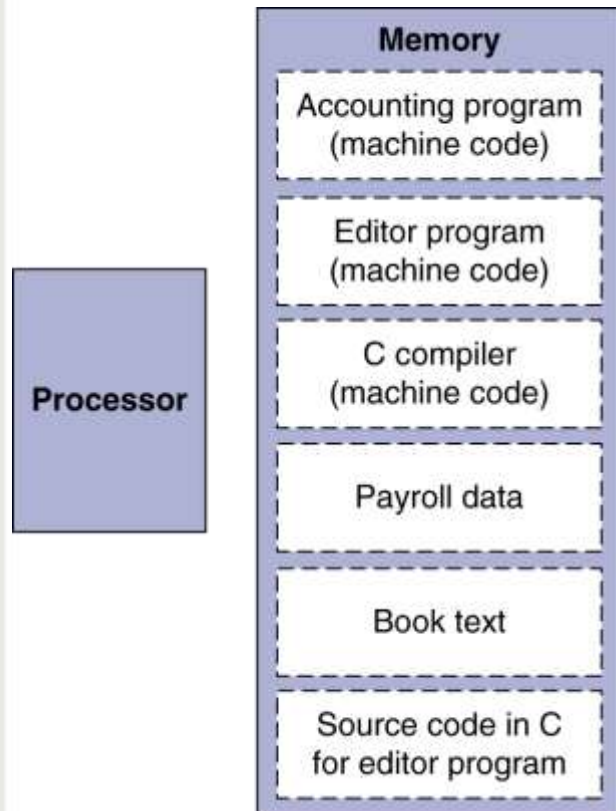
## Instruindo um Computador



UNIVERSIDADE  
FEDERAL  
DE PERNAMBUCO

[CIn.ufpe.br](http://CIn.ufpe.br)

# Computadores de Programas Armazenados

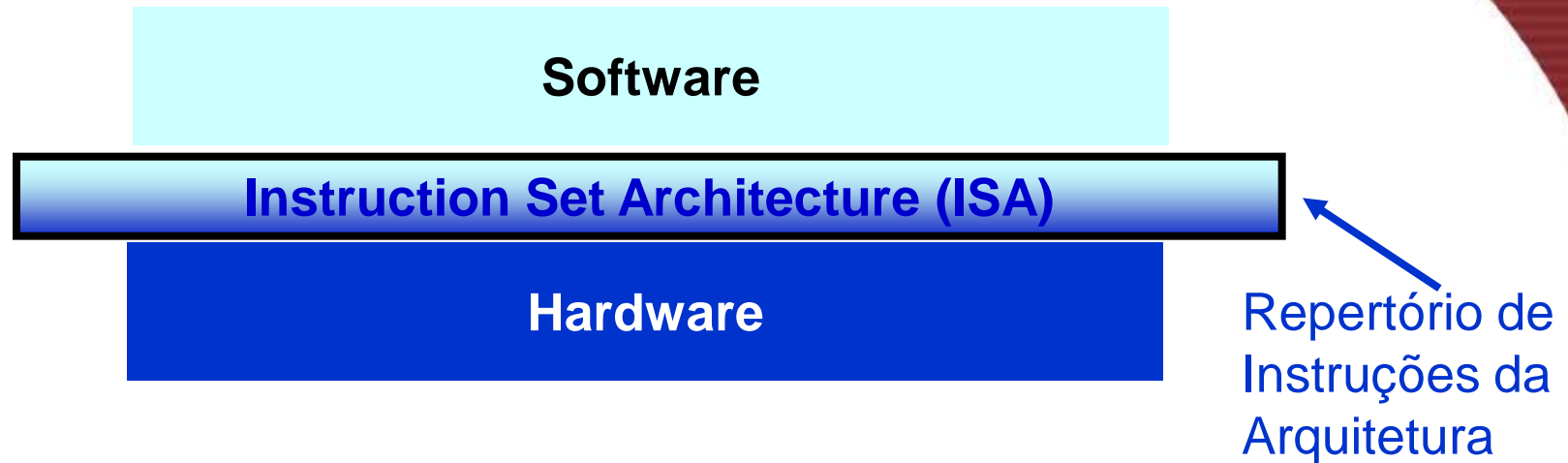


- Instruções e dados representados como números binários
- Instruções e dados armazenados em memória
- Programas podem interagir com outros programas  
Ex: compiladores, linkers, ...
- Compatibilidade de formatos binários permite que programas compilados funcionem em diferentes computadores  
ISAs padronizadas

# Perguntas que Devem ser Respondidas ao Final do Curso

- Como um programa escrito em uma linguagem de alto nível é entendido e executado pelo HW?
- **Qual é a interface entre SW e HW e como o SW instrui o HW a executar o que foi planejado?**
- O que determina o desempenho de um programa e como ele pode ser melhorado?
- Que técnicas um projetista de HW pode utilizar para melhorar o desempenho?

# Interface HW/SW: Repertório de Instruções da Arquitetura



- Última abstração do HW vista pelo SW
- Computadores diferentes podem ter diferentes ISAs
  - Mas com muitos aspectos em comum
- Visando portabilidade de código, indústria se alinha em torno de quantidade pequena de ISAs diferentes

# Evolução de ISAs

- Até metade da década de 60 computadores tinham ISAs com quantidade reduzida de instruções e instruções simples

Simplifica implementação

- Fim da década de 60 surge ISAs com grande número de instruções complexas

## **Complex Instruction Set Computer (CISC)**

Difícil implementação e existência de muitas instruções pouco usadas

- Começo da década de 80 ISAs com instruções simples voltam a ser comuns

## **Reduced Instruction Set Computer (RISC)**

# Exemplos de Processadores CISC e RISC

## ■ CISC

Intel x86, Pentium, AMDx86, AMD Athlon  
Muito utilizados em PCs



## ■ RISC

MIPS, SPARC, ARM, PowerPC  
Muito utilizados em sistemas embarcados



## ■ Tendência hoje é termos processadores híbridos

Ideias de RISC foram incorporados a CISC e vice-versa

# Repertório (ISA) do Processador MIPS

- Utilizado como exemplo nesta disciplina
- Desenvolvido no começo de 80, é um bom exemplo de uma arquitetura RISC
- Muito utilizado no mercado de sistemas embarcados

Aplicações em eletrônicos diversos, equipamento de rede/armazenamento, câmeras, impressoras, blue-rays, smart watches, etc

# Princípios de Projeto do MIPS (RISC)

- **Simplicidade é favorecida pela regularidade**

  - Instruções de tamanho fixo

  - Poucos formatos de instruções

  - Opcode sempre utiliza os primeiros 6 bits

- **Quanto menor, mais rápido**

  - Repertório de instruções limitados

  - Quantidade de registradores limitados

  - Número reduzido de modos de endereçamento

- **Torne rápido o caso mais comum**

  - Existência de instruções que contém operandos

- **Bom projeto requer boas escolhas (compromissos)**

  - Diferentes formatos de instruções complica decodificação,  
CONTUDO permite instruções de tamanho fixo



# Categoria e Formato de Instruções do MIPS

- Aritméticas
- Lógicas
- Transferência de dados
- Desvios de fluxo
- Gerenciamento de memória

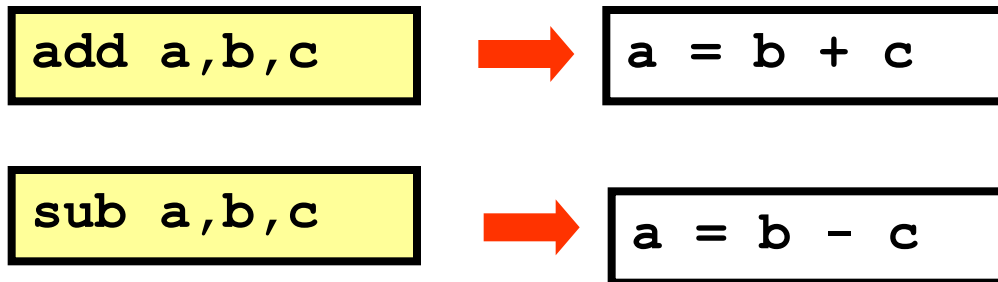
**3 Formatos de Instrução: todos com 32 bits**

op	rs	rt	rd	shamt	funct	R format
op	rs	rt	immediate			I format
op	jump target					J format

**Bom projeto requer boas escolhas (compromissos)**

# Operações Aritméticas

- Todas as instruções aritméticas no MIPS possuem 3 operandos:  
destino, fonte 1, fonte 2
- Cada instrução aritmética faz apenas uma operação



**A simplicidade é favorecida pela regularidade**

# Exemplo: Expressões Aritméticas

## Código C

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



## Pseudo-código Assembly MIPS

```
add t0,g,h #temp t0 = g + h  
add t1,i,j #temp t1 = i + j  
sub f,t0,t1 # f = t0 - t1
```

# Operandos nos Registradores

- Para melhorar desempenho, os operandos de uma instrução aritmética devem estar nos registradores

Acesso mais rápido em relação à memória

- Todos os registradores no MIPS possuem 32 bits

32 bits é uma **palavra** (word) no MIPS

- Número de registradores no MIPS é reduzido: **32**

Número grande de registradores pode penalizar desempenho

Impacto no tamanho da instrução

**Quanto menor, mais rápido**

# Registradores para Operandos no MIPS

- Nomes assembly dos registradores

**\$s0, \$s1...\$s7** – armazenam variáveis dos programas

**\$t0, \$t1...\$t9** – para valores temporários

## Código C

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



## Código Assembly MIPS

```
add $t0,$s1,$s2 # $s1 = g , $s2 = h  
add $t1,$s3,$s4 # $s3 = i , $s4 = j  
sub $s0,$t0,$t1 # $s0 = f = t0 - t1
```

# Representação das instruções

- Informação tem uma representação numérica na base 2

Codificação das instruções

Mapeamento de nomes de registradores para números

- \$s0 a \$s7 : 16 a 23
- \$t0 a \$t7 : 8 a 15
- \$t8-\$t9 : 24-25

# Formato da Instrução ADD e SUB

## Formato R de Instrução

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

- op      opcode da instrução
- rs      registrador que contém 1º operando fonte (source)
- rt      registrador que contém o 2º operando fonte
- rd      registrador destino que contém resultado
- shamt   shift amount , não utilizado para add e sub
- funct   função (function) que estende o opcode

# Representando ADD na Máquina

## ■ Número dos registradores

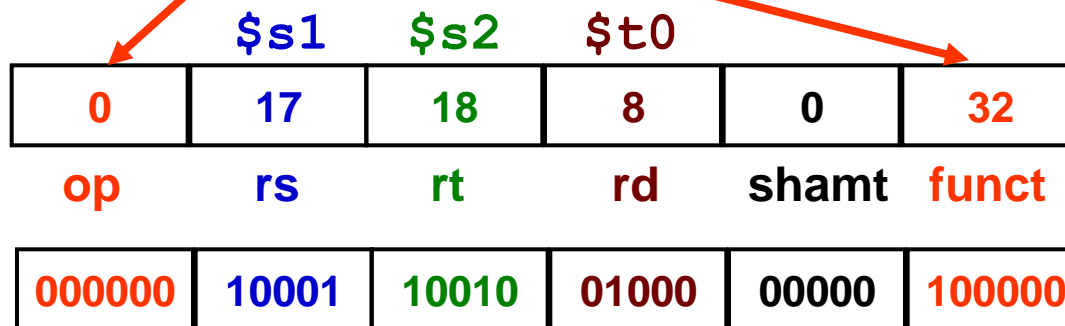
\$s0 - \$s7: 16 - 23

\$t0 - \$t7 : 8-15

\$t8-\$t9 : 24-25

## Código Assembly MIPS

add \$t0, \$s1, \$s2



Em binário



# Representando SUB na Máquina

- Número dos registradores

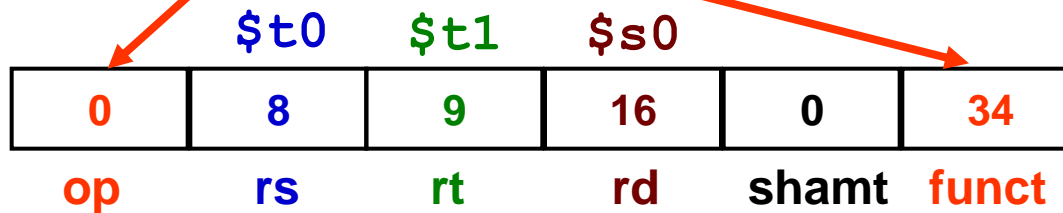
**\$s0 - \$s7: 16 - 23**

**\$t0 - \$t7 : 8-15**

**\$t8-\$t9 : 24-25**

## Código Assembly MIPS

**sub \$s0, \$t0, \$t1**



# Operações de Transferência de Dados (Memória)

- Operações de transferência de dados entre processador e memória no MIPS tem dois operandos:

registrador, endereço de memória

Endereço de memória formado por um registrador de base (contém endereço inicial) e um deslocamento(offset)

- Duas operações básicas: **lw** (load word), **sw** (store word)

```
lw r, offset(end_inicial)
```



Carrega conteúdo de **offset** + **end\_inicial** no registrador **r**

```
sw r, offset(end_inicial)
```



Armazena conteúdo de registrador **r** em **offset** + **end\_inicial**

# Operandos na Memória

- Memória muito utilizada para armazenar dados compostos  
Arrays, estruturas, dados dinâmicos
- Memória endereçada por byte (8 bits)
- **Contudo** memória é vista como uma sequência de palavras de 32 bits

**Endereços de palavras devem ser múltiplos de 4!**

- MIPS é Big Endian

Byte mais significativo tem menor endereço da palavra

Little Endian – byte mais significativo tem o maior endereço

# Endereços da Memória – Big Endian

- MIPS é Big Endian

0	0x12
1	0x34
2	0x68
3	0x00

$\$s0 = 0$   
`lw  $\$t0$ , 0( $\$s0$ )`



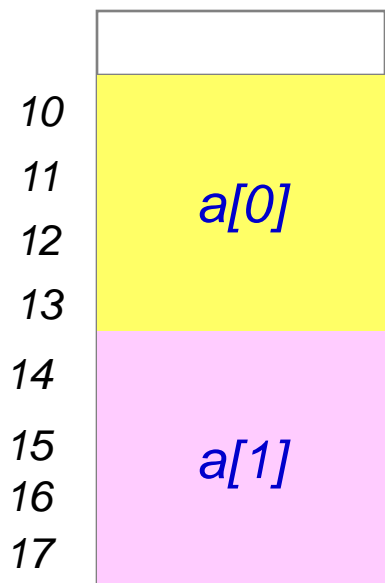
$\$t0 = 0x12346800$

# Endereços da Memória

## ■ MIPS

Inteiros de 32 bits

Array de inteiros chamado de *a*



**End** (*a*[0]) = 10

**End** (*a*[1]) = 14

$$\text{End}(\mathbf{a[i]}) = \text{End-inicial} + i \times 4$$

# Utilizando Arrays no MIPS (Memória)

- Registrador de base guarda endereço inicial do array  
Offset utilizado como índice do array

## Código C

```
a[12] = h + a[8];
```



h em \$s2,  
endereço base de  
a em \$s3

## Código Assembly MIPS

```
lw $t0, 32($s3)  
add $t0, $s2, $t0  
sw $t0, 48($s3)
```

Índice 8 requer offset de 32  
(4 x 8)

# Outro Exemplo de Arrays no MIPS

- Array com variável de indexação

## Código C

```
g = g + a[i];
```



## Código Assembly MIPS

```
add $t1, $s2, $s2  
add $t1, $t1, $t1  
add $t1, $t1, $s3  
lw  $t0, 0($t1)  
add $s1, $s1, $t0
```

g em \$s1, i em  
\$s2, endereço  
base de a em \$s3

End(a[i]) =  
base + i + i + i + i

# Registradores x Memória

- Acesso a registradores é mais rápido
- Utilização da memória requer *loads* e *stores*

Mais instruções a serem executadas

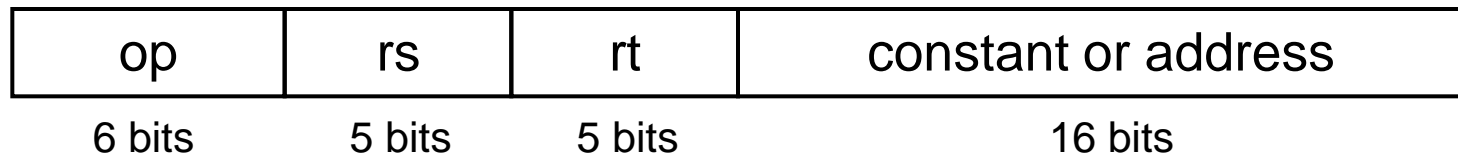
- Compilador deve maximizar a utilização de registradores

**Otimização de registradores é importante!**



# Formato da Instrução LW e SW

## Formato I de Instrução



- op      **op**code da instrução
- rs      **r**egistrador que neste caso contém endereço base
- rt      **r**egistrador fonte ou destino
- constant    **constante** que representa o offset

# Representando LW na Máquina

- Número dos registradores

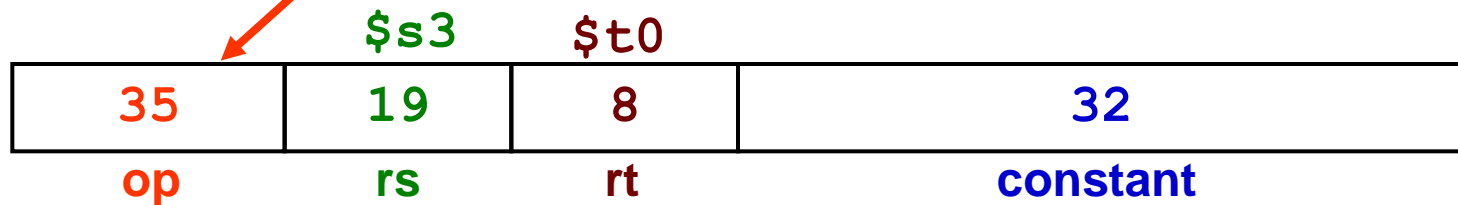
**\$s0 - \$s7: 16 - 23**

**\$t0 - \$t7 : 8-15**

**\$t8-\$t9 : 24-25**

## Código Assembly MIPS

**lw \$t0, 32(\$s3)**



# Representando SW na Máquina

- Número dos registradores

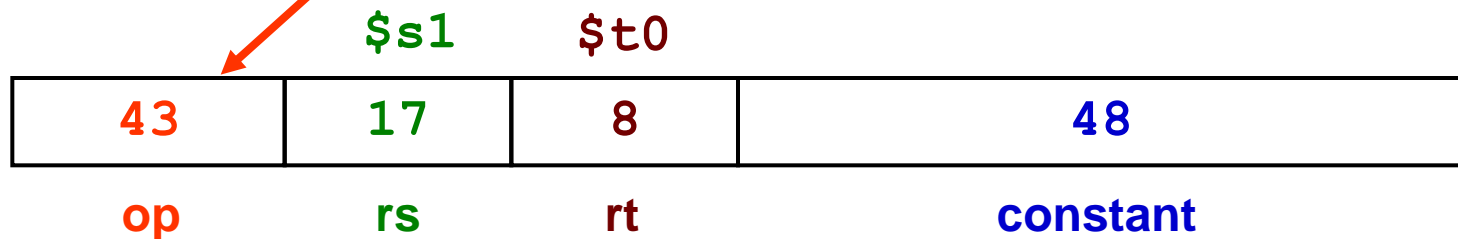
**\$s0 - \$s7: 16 - 23**

**\$t0 - \$t7 : 8-15**

**\$t8-\$t9 : 24-25**

## Código Assembly MIPS

**sw \$t0, 48(\$s1)**



# Operandos Imediatos

- Frequentemente utilizamos pequenas constantes em programas

Ex: **a = a + 1;**

- Possíveis soluções

Armazenar constantes na memória e depois carregá-las

Ter registradores que armazenam a mesma constante

- MIPS possui o registrador **\$zero** que armazena 0

**Ter instruções especiais que contêm constantes!**

# Instruções Imediatas

- MIPS oferece instruções onde uma constante está embutida na própria instrução
- Instruções imediatas contêm 3 operandos:  
destino, fonte, constante

`addi a,b,2` → `a = b + 2`

- Existe a adição imediata (addi), mas não existe a subtração imediata

Subtração : Soma com uma constante negativa

**Torne rápido o caso mais comum**

# Adição Imediata

## Código C

```
a = b + 8;  
a = a - 2;
```

a em \$s1, b em  
\$s2



## Código Assembly MIPS

```
addi $s1, $s2, 8  
addi $s1, $s1, -2
```

# Formato da Instrução ADDI

## Formato I de Instrução

op	rs	rt	constant
6 bits	5 bits	5 bits	16 bits

op      opcode da instrução  
rs      registrador fonte  
rt      registrador destino  
constant      constante embutida na instrução

# Representando ADDI na Máquina

- Número dos registradores

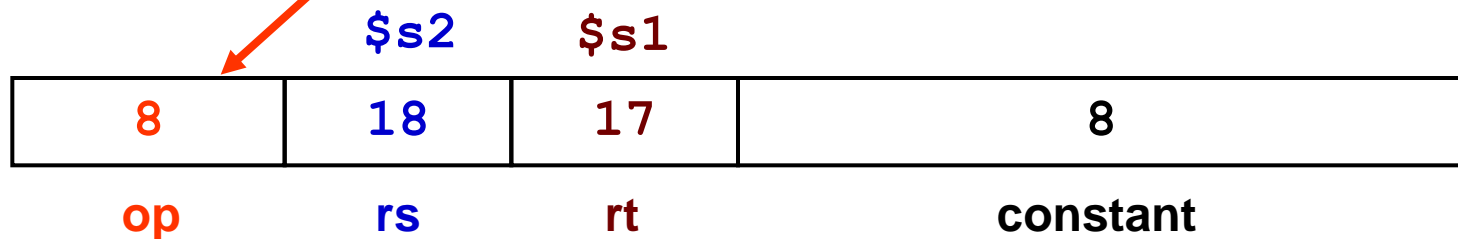
**\$s0 - \$s7: 16 - 23**

**\$t0 - \$t7 : 8-15**

**\$t8-\$t9 : 24-25**

## Código Assembly MIPS

```
addi $s1,$s2 8
```





# Operações Lógicas

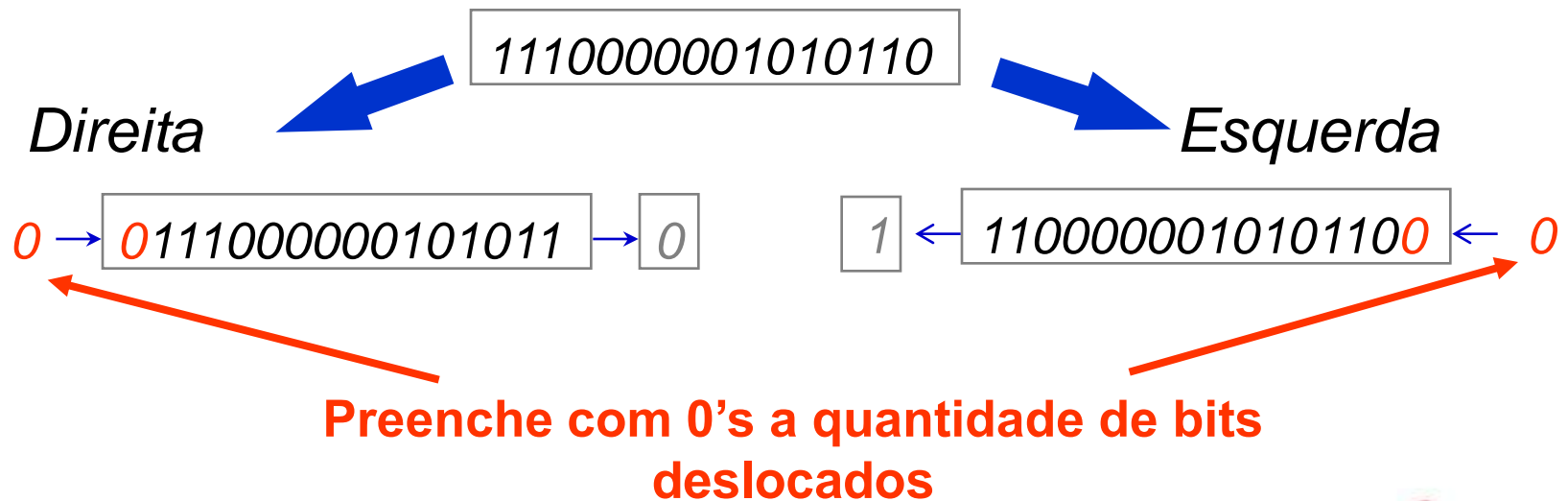
- Permitem manipulação bit a bit dos dados
- Úteis para extrair ou inserir um grupo de bits em uma palavra

Podem modificar o formato de um dado

Operação	C	Java	MIPS
Shift left	<<	<<	sll
Shift right	>>	>>>	srl
Bitwise AND	&	&	and, andi
Bitwise OR			or, ori
Bitwise NOT	~	~	nor

# Operações Lógicas de Deslocamento (Shift)

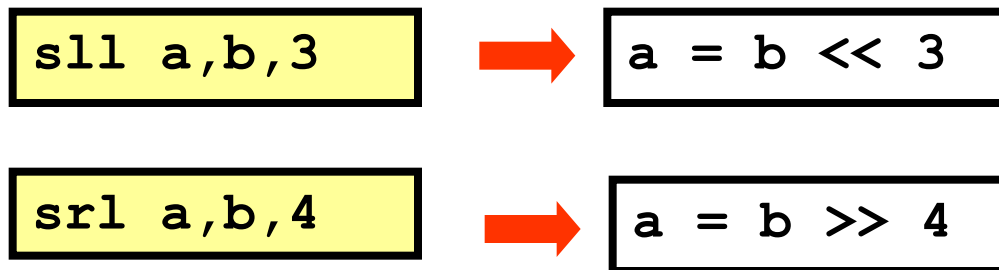
- Afeta a localização dos bits em um dado
- Permite o deslocamento para esquerda ou direita de bits de um dado
- Insere grupo de bits no dado



# Operações Lógicas de Deslocamento no MIPS

- Instruções de deslocamento no MIPS possuem 3 operandos:

destino, fonte, quantidade de bits deslocados



- Deslocamento para esquerda (direita) de  $i$  bits de um valor é equivalente a multiplicar (dividir) valor por  $2^i$

# Multiplicação com SLL

- Multiplicação do valor por 4 ( $2^2$ )

## Código C

```
g = g * 4;
```



## Código Assembly MIPS

```
sll $s1, $s1, 2
```

g em \$s1

g armazena valor 4  
( $100_2$ )

000000000000000100

g armazena valor 16  
( $10000_2$ )

00000000000010000

# Formato da Instrução SLL e SRL

## Formato R de Instrução

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

- op      opcode da instrução
- rs      não utilizado para **sll** e **srl**
- rt      registrador que contém operando fonte
- rd      registrador destino que contém resultado
- shamt   shift amount, quantidade de bits deslocados
- funct   função (function) que estende o opcode

# Representando SLL na Máquina

- Número dos registradores

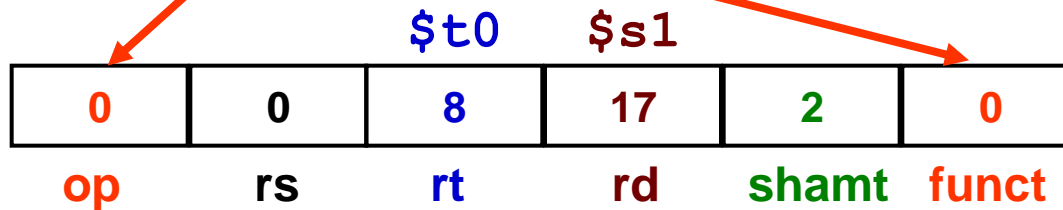
**\$s0 - \$s7: 16 - 23**

**\$t0 - \$t7 : 8-15**

**\$t8-\$t9 : 24-25**

## Código Assembly MIPS

**sll** \$s1, \$t0, 2



# Outras Operações Lógicas

- AND ,OR, NOT (MIPS implementa como *A NOR 0*)
- Úteis para extrair grupos de bits
  - “Máscara” para encontrar padrões de disposição de bits
- No MIPS, possui 3 operandos como (ADD) e tem formato R

and a,b,c



a = b & c

and \$t0,\$t1,\$t2

\$t2 0000 0000 0000 0000 0000 1101 1100 0000

\$t1 0000 0000 0000 0000 0011 1100 0000 0000

\$t0 0000 0000 0000 0000 0000 1100 0000 0000

# Operações de Controle de Fluxo

- Alterar a sequência de execução das instruções:

## *Ling. alto nível*

- *If ...then ...else*
- *case*
- *loop*
- *go to*



## *Linguagem máquina*

- *Desvio condicional a comparações entre variáveis e/ou valores*
- *Desvio incondicional*



# Desvios no MIPS

- Consegue-se através de dois tipos de instruções

***Branch*** (desvio condicional)

***Jump*** (desvio incondicional)

- Instruções do tipo *branch* tem 3 operandos

fonte 1, fonte 2, label de instrução

**beq** a, b, L1

se (a == b) desvie para L1

**bne** a, b, L2

se (a != b) desvie para L2

- Instruções do tipo *jump* tem um único operando, o label

**j** L3

desvie para L3

# Implementando Desvio Condicional - if

## Código C

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

g em \$s1, h em  
\$s2, i em \$s3



Assembler calcula  
endereço

## Código Assembly MIPS

```
bne $s3, $s2, Else
add $s3, $s1, $s2
j Exit
Else: sub $s3, $s1, $s2
Exit:...
```

# Implementando Loops - while

## Código C

```
while (save[i] == h)
    i += 1;
```

h em \$s2, i em  
\$s3 e endereço  
base de save  
em \$s4



## Código Assembly MIPS

```
Loop: sll $t1,$s3,2
      add $t1, $t1, $s4
      lw $t0,0($t1)
      bne $t0,$s2,Exit
      addi $s3, $s3,1
      j Loop
Exit:....
```

←  $\$t1 = i * 4$

←  $\text{End\_base} + \$t1$

# Mais Sobre Branchs no MIPS

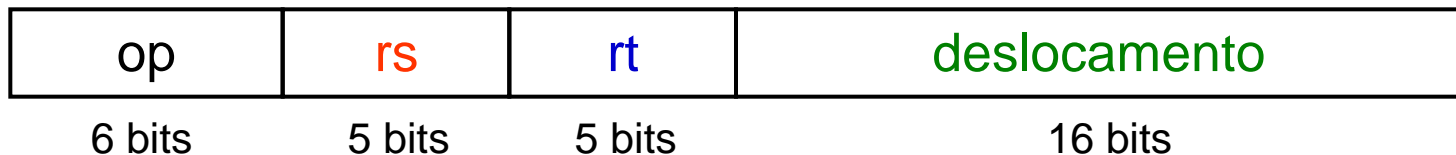
- O operando relativo ao label nas instruções de branch corresponde na verdade ao deslocamento em relação ao endereço da instrução contida no PC (Program Counter)

**PC já incrementado de 4!**

**$PC = PC + (\text{deslocamento} * 4)$  se  $reg1 == reg2$**

**beq *rs*, *rt*, deslocamento**

## Formato I de Instrução



# Representando BNE na Máquina

- Número dos registradores

**\$s0 - \$s7: 16 - 23**

**\$t0 - \$t7 : 8-15**

**\$t8-\$t9 : 24-25**

```
Loop: sll $t1,$s3,2  
      add $t1, $t1, $s4  
      lw $t0,0($t1)  
      bne $t0,$s2,Exit  
      addi $s3, $s3,1  
      j Loop  
Exit:...
```

	\$t0	\$s2	Exit
5	8	18	2
op	rs	rt	deslocamento

# Mais Sobre Jumps no MIPS

- O operando relativo ao label nas instruções de jumps corresponde na verdade ao endereço (número) da instrução a ser executada

**PC = endereço**

j endereço

## Formato J de Instrução



# Mais Operações Condicionais no MIPS

- Armazene 1 se condição é verdade, senão 0

## Set less than

```
slt rd,rs,rt
```

se ( $rs < rt$ )  $rd = 1$ , senão  $rd = 0$

- MIPS possui registrador que armazena valor 0

**\$zero**

- `slt` pode ser utilizada junto com `beq`, `bne`

```
slt $t0,$s1,$s2
```

se ( $\$s1 < \$s2$ ) desvie para L1

```
bne $t0,$zero, L1
```

# Mais Operações Condicionais no MIPS

- Nenhuma instrução de desvio para  $<$ ,  $>=$ , ... ?????

Combinar em uma só instrução branch e comparações

(  $>$ ,  $<$ ,  $>=$  ... ) , requer mais trabalho por instrução

Clock mais lento

Penaliza todas as instruções

Branch se  $==$  ou  $!=$  é o mais comum