ISA: operações lógicas, operações relacionais e suporte à tomada de decisões

ISA: operações lógicas

- **Operações sobre campos de bits**
  - **Exemplo:**
    - » **Examinar os caracteres dentro de uma palavra**
  - **Simplificam**
    - » **Empacotamento/desempacotamento de bits**

operação	operador C	operador Java	MIPS
shift left	<<	<<	sll
shift right	>>	>>>	srl
AND bit-a-bit	&	&	and, andi
OR bit-a-bit	|	|	or, ori
NOT bit-a-bit	~	~	nor

ISA: operações lógicas

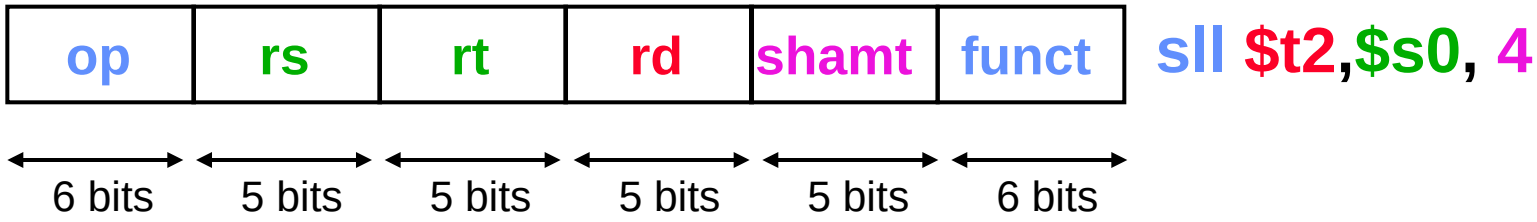
- **Deslocamentos (sll, srl)**

  sll $t2, $s0, 4 # $s0 << 4 bits

  $s0: 0000 0000 0000 0000 0000 0000 0000 1001 = 9_{dec}

  $t2: 0000 0000 0000 0000 0000 0000 1001 0000 = 144_{dec}

  - **shamt: shift amount (formato R)**
  - **Deslocar de i-bits para a esquerda = multiplicar por 2^i**



ISA: operações lógicas

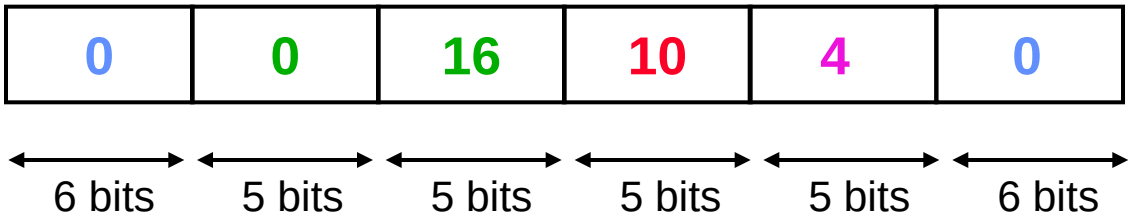
- **Deslocamentos (sll, srl)**

  sll $t2, $s0, 4 # $s0 << 4 bits

  $s0: 0000 0000 0000 0000 0000 0000 0000 1001 = 9_{dec}

  $t2: 0000 0000 0000 0000 0000 0000 1001 0000 = 144_{dec}

  - **shamt: shift amount (formato R)**
  - **Deslocar de i-bits para a esquerda = multiplicar por 2^i**



sll $t2,$s0, 4

ISA: operações lógicas

- **AND**

  and $t0, $t1, $t2 # $t0 = $t1 & $t2

  $t2: 0000 0000 0000 0000 0000 1101 0000 0000

  $t1: 0000 0000 0000 0000 0011 1100 0000 0000

  $t0: 0000 0000 0000 0000 0000 1100 0000 0000

  – **Usada para aplicar padrão de bits que força o resultado para 0**

    » **Máscara**

ISA: operações lógicas

- **OR**

  or $t0, $t1, $t2 # $t0 = $t1 | $t2

  $t2: 0000 0000 0000 0000 0000 1101 0000 0000

  $t1: 0000 0000 0000 0000 0011 1100 0000 0000

  $t0: 0000 0000 0000 0000 0011 1101 0000 0000

- **NOT (NOR)**

  nor $t0, $t1, $t3 # $t0 = ~ ($t1 | $t3)

  $t3: 0000 0000 0000 0000 0000 0000 0000 0000

  $t1: 0000 0000 0000 0000 0011 1100 0000 0000

  $t0: 1111 1111 1111 1111 1100 0011 1111 1111

ISA: suporte para decisões

- **Computador ≠ calculadora**
  - **Capacidade de tomar decisões**
  - **Teste de condição**
    - » **Descrita por operadores relacionais ou lógicos**
- **Linguagens de programação**
  - **if-then-else, while, for, repeat ... until**
- **Suporte no ISA**
  - **Desvio condicional (“branch”)**
  - **Desvio incondicional (“jump”)**

Desvios no MIPS

- **Endereço-alvo do desvio**
  - **Instrução a ser executada quando desvio tomado**
    - » **Representação em linguagem de montagem: “labels”**
- **Desvios condicionais**
  - **beq reg1, reg2, L1**
    - » **se reg1 == reg2, desvie para o endereço L1**
  - **bne reg1, reg2, L1**
    - » **se reg1 != reg2, desvie para o endereço L1**
- **Desvio incondicional**
  - **j L1**
    - » **Desvie para o endereço L1**

Compilando “if-then-else”

- **Em linguagem de alto nível**

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

- **Alocação:**

  (f, g, h, i, j) → ($s0, $s1, $s2, $s3, $s4)

- **Em linguagem de montagem**

```
                bne $s3,$s4,Else
                add $s0,$s1,$s2
                j Exit
Else:           sub $s0,$s1,$s2
Exit:           ...
```

Interface HW/SW

- **Desvios e “labels”**
  - **Não precisam ser escritos explicitamente em linguagens de alto nível**
  - **Codificação resulta mais rápida em alto nível**
- **Compiladores**
  - **Inserem desvios e “labels”**
    - » **Inexistentes no código em alto nível**

Compilando “while”

- **Em linguagem de alto nível**

  while (save[i] == k) i += 1;

- **Alocação:**

  (i, k) → ($s3, $s5); $s6: base do arranjo “save”

- **Em linguagem de montagem**

```
        ...
Loop:   sll $t1, $s3, 2      # $t1 = 4*i
        add $t1, $t1, $s6    # $t1 = endereço de save[i]
        lw $t0, 0($t1)       # t0 = save[i]
        bne $t0, $s5, Exit   # vá p/ Exit se save[i] ≠ k
        addi $s3, $s3, 1     # i = i+1
        j Loop
Exit:   ...
```

Bloco básico

- **Seqüência de instruções tal que:**
  - **Não contém desvios**
    - » **Exceto talvez no final**
  - **Não contém “labels”**
    - » **Exceto talvez no início**
- **Fundamental para compilação**
  - **Primeiras fases: divisão em blocos básicos**
- **Fundamental para otimização**
  - **Define o escopo da otimização**
    - » **Local e global**

Instruções para comparação

- **Testes mais populares**
  - **Igualdade e desigualdade**
    - » **Embutidos em desvios condicionais**
  - **Teste "menor que"**
    - » **Instrução de comparação: slt**

- **Comparação entre duas variáveis**

```
slt  $t0, $s3, $s4  # $t0 = 1 se $s3 < $s4
```

- **Comparação entre variável e constante**

```
slti  $t0, $s3, 10  # $t0 = 1 se $s3 < 10
```

Interface HW/SW

- **Para implementar relacionais MIPS só usa:**
  - **beq**
  - **bne**
  - **slt**
  - **slti**
  - **registrador $zero**
- **Comparação e desvio em instrução única**
  - **Relacionais = , ≠**
- **Comparação e desvio em duas instruções**
  - **Relacionais <, > , ≤ , ≥**

Combinando comparações e desvios

- **Exemplo: blt $s0, $s1, Less (pseudo)**
  Se $s0 < $s1, desvie p/ Less;

```
slt   $t0, $s0, $s1      # $t0 = 1, se $s0 < $s1
bne $t0, $zero, Less  # vá para "Less", se $t0 ≠0 ($s0 < $s1)
```

Combinando operações lógicas e desvios

- **Máscaras**
  - **Exemplo “se bit 1 de $t0=0 vá para L”**
- **Código**

```
andi $t0, $t0, 2
beq $t0, $zero, L
```

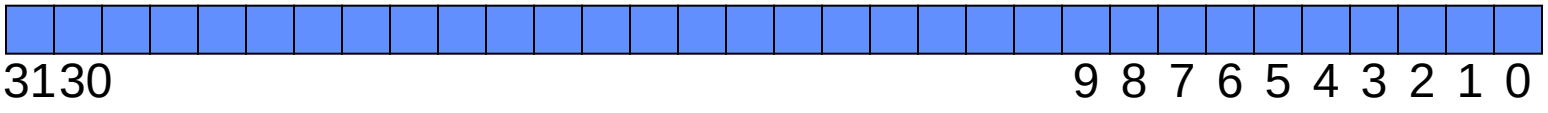
t0	0101 … 00?0
2	0000 … 0010
t0 and 2	0000 … 00?0

Revisando alternativas de representação numérica

- **Como representar**
  - **Números inteiros “unsigned”?**
    - » **Dígitos binários representam a magnitude**
  - **Números inteiros positivos ou negativos ?**
    - » **Um dígito binário tem acepção de sinal**
    - » **Representação em complemento de dois**
  - **Número fracionários ou reais ?**
    - » **Ponto flutuante (mantissa e expoente)**
    - » **Fora do âmbito desta disciplina**

Inteiros sem sinal

- **Representação binária simples**



$$2^{n-1} \times \text{bit}_{n-1} + 2^{n-2} \times \text{bit}_{n-2} \ldots 2^{2} \times \text{bit}_{2} + 2^{1} \times \text{bit}_{1} + 2^{0} \times \text{bit}_{0}$$

- **Faixa: [0, 2^n-1]**
  - **Se número ≥ 2^n ⇒ transbordo ou "overflow"**
- **Uso**
  - **Endereços de memória**
  - **Inteiros sem sinal (unsigned int)**

Inteiros com sinal

- **Complemento de dois**

$$-2^{n-1} \times \text{bit}_{n-1} + 2^{n-2} \times \text{bit}_{n-2} \ldots 2^{2} \times \text{bit}_{2} + 2^{1} \times \text{bit}_{1} + 2^{0} \times \text{bit}_{0}$$

- **Faixa: [- 2^{n-1} ; + 2^{n-1})**
  - **Se número $\geq$ $+2^{n-1}$ $\Rightarrow$ transbordo (“overflow”)**
  - **Se número $<$ -2^{n-1} $\Rightarrow$ transbordo (“overflow”)**
- **Uso**
  - **Variáveis inteiras**
  - **Constantes inteiras**

MIPS: faixa de representação

- **Números com sinal (32 bits):**

```
0000 0000 0000 0000 0000 0000 0000 0000 =   0dec
0000 0000 0000 0000 0000 0000 0000 0001 = + 1dec
0000 0000 0000 0000 0000 0000 0000 0010 = + 2dec
   ...
0111 1111 1111 1111 1111 1111 1111 1110 = + 2.147.483.646dec
0111 1111 1111 1111 1111 1111 1111 1111 = + 2.147.483.647dec   Max int
1000 0000 0000 0000 0000 0000 0000 0000 = - 2.147.483.648dec   Min int
1000 0000 0000 0000 0000 0000 0000 0001 = - 2.147.483.647dec
1000 0000 0000 0000 0000 0000 0000 0010 = - 2.147.483.646dec
   ...
1111 1111 1111 1111 1111 1111 1111 1101 = - 3dec
1111 1111 1111 1111 1111 1111 1111 1110 = - 2dec
1111 1111 1111 1111 1111 1111 1111 1111 = - 1dec
```

ISA: tipos de comparações

- **Comparação com sinal × sem sinal**
  - **(slt, slti) × (sltu, sltiu)**

  - **$s0: 1111 1111 1111 1111 1111 1111 1111 1111**
  - **$s1: 0000 0000 0000 0000 0000 0000 0000 0001**

  - **slt $t0, $s0, $s1 # $t0=1, pois -1 < 1**

  - **sltu $t1, $s0, $s1 # $t1=0, pois 4.294.967.295 > 1**

Atalho para checar limites de arranjo

- **Tratar números sinalizados como não-sinalizados**
  - **Alternativa de baixo custo para testar $0 \leq x < y$**
  - **Limites de indexação de um arranjo com y elementos**
- **Idéia-chave:**
  - **Inteiros negativos**
    - » **Em notação complemento de dois**
    - » **MSB é bit de sinal**
  - **Parecem números grandes**
    - » **Em notação não-sinalizada**
    - » **MSB contribui com um grande valor para a magnitude**
- **Consequência**
  - **Comparação não-sinalizada: sltu $t0, x, y**
  - **Equivale a duas comparações: $0 \leq x < y$**

Atalho para checar limites de arranjo

- **Exemplo:**
  - **Desvie para IndexOutofBounds se x < 0 ou x ≥ y**
  - **Suponha que (x, y) → ($s1, $t2)**

```
sltu $t0, $s1, $t2          # $t0 = 1, se 0 ≤ x < y
                            # $t0 = 0, se x < 0 ou x ≥ y
beq $t0, $zero, IndexOutofBounds
```

Suporte para “Case/Switch”

- **Implementação 1**
  - **Estruturas if-then-else aninhadas**
  - **Ineficiente**
- **Implementação 2**
  - **“Jump address table”**
  - **Indexar a tabela e desviar para a opção apropriada**
  - **Suporte:**
    - » **Desvio incondicional cujo endereço-alvo é especificado em registrador**

    jr $s3

Interface HW/SW

- **Linguagens de alto nível**
  - **C, C++, Java, etc.**
  - **Há várias construções condicionais e laços em linguagens de alto nível**
    - » **If-then-else, case/switch**
    - » **While, repeat ... until, for**
- **Seu suporte no ISA são os desvios**
  - **Condicionais: beq, bne**
  - **Incondicionais: j, jr**