Instruções

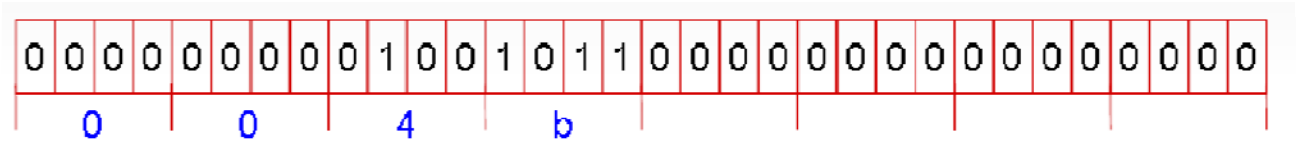
MIPS: Instruções

- Transferência de Dados
  - Como carregar constantes de 32 bits?
    - Utilizar a instrução lui (*load upper immediate*) para os MSBs
    - Utilizar a instrução ori (*or immediate*) para os LSBs

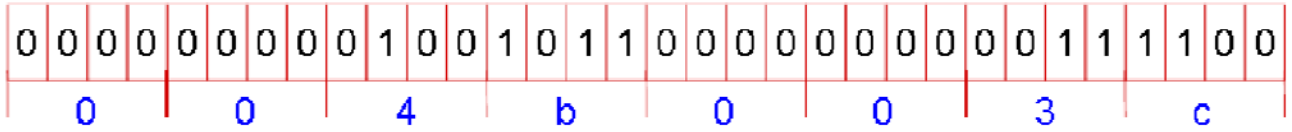
MIPS: Instruções

- Transferência de Dados
  - Ex: $t0 0x004b003c

lui $t0, 0x004b



ori $t0,$t0, 0x003c



MIPS: Instruções

- Desvio condicional (branch)
  - Em linguagens de programação de alto nível o comando de tomada de decisão é geralmente implementado através do comando if
  - O MIPS possui duas principas instruções de desvio:
    - beq r1, r2, L1 (branch if equal)
      - se os valores de r1 e r2 são iguais, desvia para a linha identificada pelo label L1
    - bne r1, r2, L1 (branch if not equal)
      - se os valores de r1 e r2 são diferentes, desvia para a linha identificada pelo label L1

if cond then goto label

MIPS: Instruções

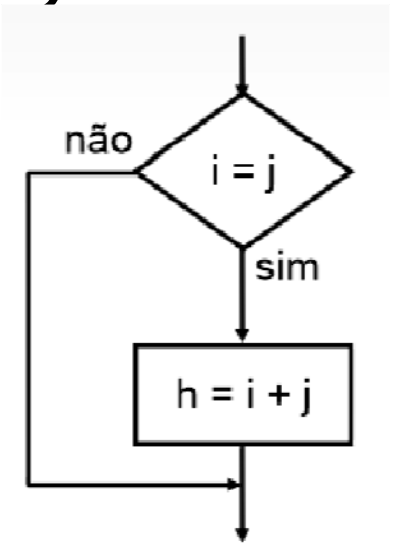
- Desvio condicional (branch)

```
beqz $s0, label          #if $s0==0 goto label

bnez $s0, label          #if $s0!=0 goto label

bge  $s0, $s1, label     #if $s0>=$s1 goto label

ble  $s0, $s1, label     #if $s0<=$s1 goto label

blt  $s0, $s1, label     #if $s0<$s1 goto label

beq  $s0, $s1, label     #if $s0==$s1 goto label

bne  $s0, $s1, label     #if $s0!=$s1 goto label

bgez $s0, label          #if $s0>=0 goto label
```

MIPS: Instruções

- Desvio condicional (branch)

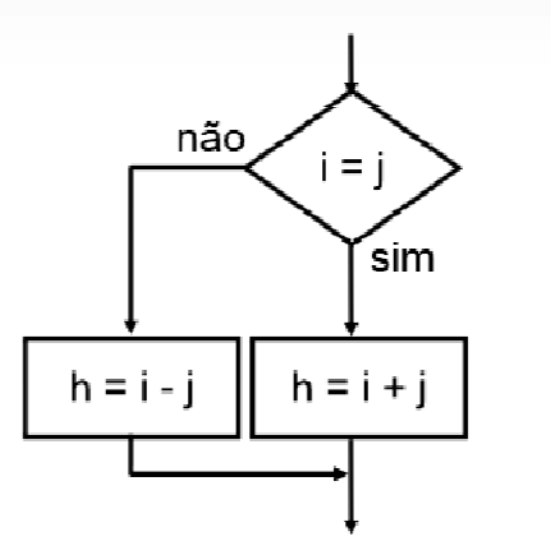
```
        bne $8, $9, exit
        add $10, $8, $9
exit: ...
```



MIPS: Instruções

- Desvio condicional (branch)

```
        bne  $8, $9, else
        add  $10, $8, $9
        j exit
else: sub  $10, $8, $9
exit : ...
```



MIPS: Instruções

- Instrução de Comparação
  - Permite testar se uma variável é menor do que outra
  - Compara o conteúdo de dois registradores e atribui o valor 1 a um terceiro registrador se primeiro < segundo. Caso contrário, é atribuído o valor 0 ao terceiro registrador.
  - slt (set on less than)

```
slt $t0, $s3, $s4        # $t0 recebe 1 se $s3 < $s4
                         # $t0 recebe 0 se $s3 ≥ $s4
```

MIPS: Instruções

- Instruções de Controle
  - Com instruções simples como Branch e Jump é possível construir estruturas como:
    - Loops (For, While, Repeat Until)
    - If-Then-Else
    - Chamadas de funções

MIPS: Instruções

- Instruções de Controle

```
if ($t0==$t1)
then
    $t2= 25
else
    $t2= 77
$t3 = $t3 + $t2
```

```
        beq $t0, $t1, blockA
        j blocoB
blocoA: li $t2, 25
        j exit
blocoB: li $t2, 77
exit:   addu $t3, $t3, $t2
```

MIPS: Instruções

- Instruções de Controle

repeat … until $t0>$t1
  t0= t0 + 1

loop1:
    addi $t0, $t0, 1
    ble $t0, $t1, loop1   # if $t0<=$t1 goto loop1

MIPS: Instruções

- Instruções de Controle

```
switch (k) {
    case 0: f = i + j; break;
    case 1: f = g + h; break;
    case 2: f = g - h; break;
    case 3: f = i - j;
}
```

```
        slt $t3, $s5, $zero
        bne $t3, $zero, Exit
        slt $t3, $s5, $t2
        beq $t3, $zero, Exit
        add $t1, $s5, $s5
        add $t1, $t1, $t1
        add $t1, $t1, $t4
        lw $t0, 0 ($t1)
        jr $t0
L0:     add $s0, $s3, $s4
        J Exit
L1:     add $s0, $s1, $s2
        j Exit
L2:     sub $s0, $s1, $s2
        j Exit
L3:     sub $s0, $s3, $s4
Exit:
```

Sub-rotina

MIPS: Sub-rotina

- Passos pra a execução
  1. Passagem de parâmetros
  2. Transferência do controle de execução
  3. Aquisição dos recursos de armazenamento necessários para o procedimento
  4. Realização da tarefa desejada.
  5. Armazenamento do resultado em local acessível ao programa que chamou a sub-rotina.
  6. Retorno à execução do programa original.

MIPS: Sub-rotina

- Convenção
  - $a0 - $a3: quatro registradores para a passagem de parâmetros.
  - $v0 - $v1: dois registradores para retorno de resultados.
  - $ra: registrador de endereço de retorno.
  - $t0 – t9: 10 registradores temporários)

- Instruções adicionais
  - jal ROTINA
    - Desvia a execução para o endereço indicado por ROTINA e automaticamente salva o endereço de retorno (PC + 4) em $ra.
  - jr $ra
    - Desvia para a posição de memória indicada

MIPS: Sub-rotina

- Convenção
  - Se forem necessários mais que quatro argumentos e/ou dois resultados:
    - $t0-$t9
      - Dez registradores temporários que NÃO SÃO preservados pelo chamador
    - $s0-$s7
      - Oito registradores que DEVEM ser preservados se utilizados no procedimento chamado

MIPS: Sub-rotina

- Convenção
  - Registradores "caller-saved": $t0-$t9
    - Chamador é responsável por salvá-los em memória caso seja necessário usá-los novamente depois que um procedimento é chamado
  - Registradores "callee-saved": $s0-$s7
    - Chamado é responsável por salvá-los em memória antes de utilizá-los e restaurá-los antes de devolver o controle ao chamador

MIPS: Sub-rotina

```
        li $a0,10
        li $a1,21
        li $a3,31
        jal silly       #Now the result of the function is in $v0.
        li $v0,10       #syscall code 10 is for exit.
        Syscall         #exit the program gracefully
silly: add $t0,$a0,$a1
        sub $v0,$a3,$t0
        jr $ra
```

Pilha

MIPS: Pilha

- Preservação de Contexto
  - Quando as chamadas a procedimentos forem recursivas ou aninhadas, é conveniente o uso de uma pilha
  - Qualquer registrador usado pelo chamador deve ter seu conteúdo restaurado para o valor original antes da chamada
  - O conteúdo dos registradores é salvo na memória. Depois da execução do procedimento, estes registradores devem ter seus valores restaurados

MIPS: Pilha

- Preservação de Contexto
  - $sp (stack pointer) - registrador utilizado para guardar o endereço do topo da pilha da chamada de procedimentos:
    - a posição de memória que contêm os valores dos registradores salvos na memória pela última chamada
    - a posição a partir da qual a próxima chamada de procedimento pode salvar seus registradores

MIPS: Pilha

- Manipulando a Pilha
  - MIPS não possui instruções específicas para manipular a pilha
  - As instruções lw e sw devem ser utilizadas para essa função
    - Ex.: inserir o conteúdo do registrador $s0 na pilha

```
addi  $sp, $sp, -4   # avança o stack pointer
sw    $s0, 0($sp)    # empilha o valor de $s0
```

    - Ex.: remover o topo da pilha e armazenar em $s0

```
lw    $s0, 0($sp)   # restaura $s0
addi  $sp,$sp,4     # desempilha o topo
```

Chamdas de Sistema

MIPS: Chamadas de Sistema

- Syscall
  - Instrução especial que suspende a execução do programa do usuário e transfere o controle para o sistema operacional
  - O sistema operacional acessa o conteúdo do registrador $V0 para determinar qual tarefa esta sendo solicitada e retorna o controle ao programa do usuário quando a tarefa é realizada
  - Possibilita chamar serviços fornecidos pelo sistema operacional como a leitura do teclado, a exibição de dados na tela, operações de acesso a disco, rede, etc...

MIPS: Chamadas de Sistema

- Syscall

Função	$v0	Arguments / Result
Print Integer	1	$a0 = integer value to print
Print Float	2	$f12 = float value to print
Print Double	3	$f12 = double value to print
Print String	4	$a0 = address of null-terminated string
Read Integer	5	$v0 = integer read
Read Float	6	$f0 = float read
Read Double	7	$f0 = double read
Read String	8	$a0 = address of input buffer $a1 = maximum number of characters to read
Exit Program	10	
Print Char	11	$a0 = character to print
Read Char	12	$a0 = character read

MIPS: Chamadas de Sistema

- Syscall

a) Carregar argumentos das funções em registradores pré-definidos
b) Carregar código das funções no registrador $v0
c) Executar a função syscall

```
a) li  $a0, 10      # argumento: $a0=10
b) li  $v0, 1       # código da função "print integer"
c) syscall          # exibe na tela conteúdo de $a0 (10)
```

MIPS: Chamadas de Sistema

- Syscall
  - Finalizar a execução do programa

```
li $v0, 10          # syscall code 10 is for exit.
syscall             # make the syscall.
```

  - Leitura de inteiros

```
li   $v0, 5         # load syscall read_int into $v0
syscall             # make the syscall
move $t0, $v0       # move the number read into $t0
```