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 006

Aula de Hoje:

Conjunto de Instruções

- **Instruções de controle**
- **Construções: condicionais e laços de repetição.**

Conjunto de Instruções

- **Desvios**
  - Alteram o fluxo de execução do programa.

Desvios

- **Todo programa executa instruções fora da sequência.**
- **Tipos de desvios (branches):**
  - Desvio condicional (conditional branches):
    - » branch if equal (`beq`): salta se igual.
    - » branch if not equal (`bne`): salta se não igual.
  - Desvio incondicional (unconditional branches):
    - » jump (`j`): salto com endereço/label.
    - » jump register (`jr`): salta para o endereço especificado em registrador.
    - » jump and link (`jal`): chamada de procedimento.

Instruções de desvio condicional

- **beq *(branch if equal)***

```
beq registrador1, registrador2, L1
```

  - se o valor do ***registrador1*** for ***igual*** ao do ***registrador2*** o programa será desviado para o label ***L1***.

- **bne *(branch if not equal)***

```
bne  registrador1, registrador2, L1
```

  - se o valor do ***registrador1*** não for igual ao do ***registrador2*** o programa será desviado para o label ***L1***.

Beq: exemplo

```
# MIPS assembly
        addi  $s0, $0, 4        # $s0 = 0 + 4 = 4
        addi  $s1, $0, 1        # $s1 = 0 + 1 = 1
        sll   $s1, $s1, 2       # $s1 = 1 << 2 = 4
        beq   $s0, $s1, target  # desvio é tomado
        addi  $s1, $s1, 1       # Não executado
        sub   $s1, $s1, $s0     # Não executado

target:                         # label
        add   $s1, $s1, $s0     # $s1 = 4 + 4 = 8
```

Bne: exemplo

```
# MIPS assembly
            addi  $s0, $0, 4        # $s0 = 0 + 4 = 4
            addi  $s1, $0, 1        # $s1 = 0 + 1 = 1
            sll   $s1, $s1, 2       # $s1 = 1 << 2 = 4
            bne   $s0, $s1, target   # branch not taken
            addi  $s1, $s1, 1       # $s1 = 4 + 1 = 5
            sub   $s1, $s1, $s0     # $s1 = 5 – 4 = 1

target:
            add   $s1, $s1, $s0     # $s1 = 1 + 4 = 5
```

Instruções de desvio condicional

Exemplo - Compilando um comando IF.
Seja o comando abaixo:

```
    if ( i == j ) go to L1;
    f = g + h;
L1: f = f - i;
```

Supondo que as 5 variáveis correspondam aos registradores $s0..$s4, respectivamente, como fica o código MIPS para o comando?

Solução

```
    beq   $s3,$s4,L1       #   vá para L1 se i = j
    add   $s0,$s1,$s2      #   f = g + h, executado se i != j
L1: sub   $s0,$s0,$s3      #   f = f – i, executado se i = j
```

Desvio incondicional (j)

```
# MIPS assembly
        addi $s0, $0, 4              # $s0 = 4
        addi $s1, $0, 1              # $s1 = 1
        j          target            # jump to target
        sra        $s1, $s1, 2       # não executado
        addi       $s1, $s1, 1       # não executado
        sub        $s1, $s1, $s0     # não executado

target: add $s1, $s1, $s0      # $s1 = 1 + 4 = 5
```

Instruções de desvio incondicional

- **j L1**
  - quando executado faz com que o programa seja desviado para L1

Exemplo – Compilando um comando if-then-else

Seja o comando abaixo:

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

Supondo:
i: $s3, **j:** $s4
f: $s0, **g:** $s1 e **h:** $s2

```
      bne $s3,$s4,Else  # vá para Else se i != j
      add $s0,$s1,$s2   #  f = g + h, se i != j
      j Exit            #  vá para Exit
Else: sub $s0,$s1,$s2   #  f = g – h, se i = j
Exit: nop
```

Desvio incondicional (jr)

```
# MIPS assembly
Endereço          Instrução
0x00002000        addi $s0, $0, 0x2010
0x00002004        jr   $s0
0x00002008        addi $s1, $0, 1
0x0000200C        sra  $s1, $s1, 2
0x00002010        lw   $s3, 44($s1)
```

Quais instruções serão executadas no trecho de código?

Instruções para teste de maior ou menor

- ***Set less than (slt)***
  - `slt  reg_temp, reg1, reg2`
    - se **reg1** é ***menor*** que **reg2**, **reg_temp** é setado, caso contrário é resetado.
    - Nos processadores MIPS o registrador $0 possui o valor zero ($zero).

Exemplo: Compilando o teste *Menor que*

Solução:

```
slt   $t0, $s0, $s1                 #  $t0 é setado se $s0 < $s1
bne   $t0, $zero, EnderecoAlvo      # vá para EnderecoAlvo, se $t0 != 0,
                                    #  ou seja $s0 < $s1
```

Instruções para teste de maior ou menor

- ***Set Greater Than (sgt)***
  - `sgt reg_temp, reg1, reg2`
    - se **reg1** é ***maior*** que **reg2**, **reg_temp** é setado, caso contrário é resetado.
    - Nos processadores MIPS o registrador $0 possui o valor zero ($zero).

Exemplo: Compilando o teste *Maior que*

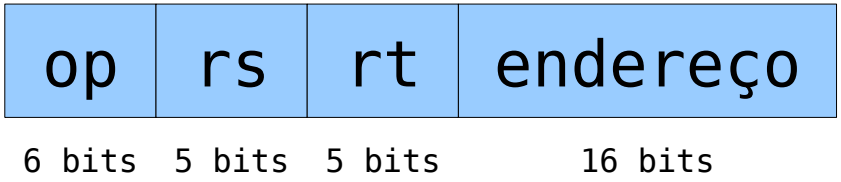
Solução:

```
sgt   $t0, $s0, $s1              #  $t0 é setado se $s0 > $s1
bne   $t0, $zero, EnderecoAlvo   # vá para EnderecoAlvo,
                                 # se $t0 != 0, ou seja a>b
```

Saltando mais longe

Dado o *branch* abaixo, rescrevê-lo de tal maneira a oferecer um deslocamento (offset) maior:

```
beq   $s0,$s1,L1
```



Solução

```
        bne $s0, $s1, L2
L2:     j L1
```

Salta para um jump que suporta um endereço maior

Construções de Alto Nível

- **if *statements***
- **if/else *statements***
- **while loops**
- **for loops**

if Statement

High-level code

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

f = f – i;
```

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j
    bne $s3, $s4, L1
    add $s0, $s1, $s2

L1: sub $s0, $s0, $s3
```

Note que em assembly o teste é o oposto (`i != j`) do teste em alto nível (`i == j`).

if / else Statement

High-level code

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

MIPS assembly code

```
# $s0 = f, $s1 = g, $s2 = h
# $s3 = i, $s4 = j

      bne $s3, $s4, L1
      add $s0, $s1, $s2
      j   done
L1:   sub $s0, $s0, $s3
done: nop
```

case/switch

Exemplo – Compilando o case/switch.

Seja a construção abaixo:

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

case/switch

Solução: supor que $t2 tenha 3 e f, g, h, i, j, k = $s0..$s5, respectivamente.

```
slt    $t3,$s5,$zero       # teste se k < 0
bne    $t3,$zero,Exit      # se k < 0 vá para Exit

slt    $t3,$s5,$t2         # teste se k < 3
beq    $t3,$zero,Exit      # se k>=2 vá para Exit

add    $t1,$s5,$s5         # $t1 = 2 * k
add    $t1,$t1,$t1         # $t1 = 4 * k

# assumindo que 3 palavras na memória, começando no endereço contido em $t4,
# tem endereçamento correspondente a L0, L1, L2

add    $t1,$t1,$t4         # $t1 = endereço de tabela[k]
lw     $t0,0($t1)          # $t0 = tabela[k]
jr     $t0                 # salto para endereço carregado em $t0: L0 ou L1 ou L2

L0:    add   $s0,$s3,$s4   # k = 0 -> f = i + j
       j     Exit

L1:    add   $s0,$s1,$s2   # k = 1 -> f = g + h
       j     Exit

L2:    add   $s0,$s3,$s2   # k = 2 -> f = i + h

Exit:  nop
```

Testa se *k* está dentro dos limites dos casos

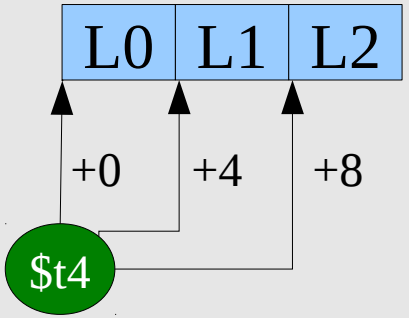
Calcula o deslocamento na tabela de saltos

*desloc = 4 * k*

*4*2 = 2+2+2+2 = 4+4*

addi $t1,$0, 4

mult $t1, $s5



For Loops

A forma geral de um laço *for* é:

```
for(inicialização; condição; loop){
  corpo do laço
}
```

- ***inicialização:*** executado antes do loop
- ***condição:*** testada no início de cada iteração
- ***loop:*** executa no fim de cada iteração
- ***corpo do laço:*** executado para cada vez que a condição é satisfeita

For Loops

Código de alto nível

```
//somar números de 0 a 9
int sum = 0;
int i;

for (i=0; i!=10; i = i+1) {
  sum = sum + i;
}
```

Código Assembly do MIPS

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        add  $s0, $0, $0
        addi $t0, $0, 10
for:    beq  $s0, $t0, end_for
        add  $s1, $s1, $s0
        addi $s0, $s0, 1
        j    for
end_for:
```

Loops (Laços)

- **Usando DO...WHILE**

  Exemplo:

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

`$s5` tem o endereço do início de `A`, que é o `A[0]`

```
Loop: g = g + A[i];
      i = i + j;
      if ( i != h ) go to Loop
```

- **Solução:**

Como os elementos de A, são inteiros (32 bits = 4 bytes) o deslocamento para acessar o i-ésimo elemento é dado por 4 * i. Se i=2, então o A[2] inicia no 8º byte a partir do endereço base.

```
Loop:   add $t1,$s3,$s3    # $t1 = 2 * i
        add $t1,$t1,$t1    # $t1 = 4 * i
        add $t1,$t1,$s5    # $t1 recebe endereço de A[i]
        lw  $t0,0($t1)     # $t0 recebe A[i]
        add $s1,$s1,$t0    # g = g + A[i]
        add $s3,$s3,$s4    # i = i + j
        bne $s3,$s2,Loop   # se i != h vá para Loop
```

Loops (Laços)

- **Usando while**
- **Exemplo:**

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

Solução: Para i, j e k correspondendo a $s3, $s4 e $s5, respectivamente, e o endereço base do *array* em $s6, temos:

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

For Loops: Usando slt

Código de Alto nível

```
// add the powers of 2 from 1
// to 100
int sum = 0;
int i;

for (i=1; i < 101; i = i*2) {
  sum = sum + i;
}
```

Código Assembly MIPS

```
# $s0 = i, $s1 = sum
        addi $s1, $0, 0
        addi $s0, $0, 1
        addi $t0, $0, 101
loop:   slt  $t1, $s0,$t0
        beq  $t1, $0, done
        add  $s1, $s1, $s0
        sll  $s0, $s0, 1
        j    loop
done:
```

$t1 = 1 if i < 101.

While Loops

High-level code

```
// determines the power
// of x such that 2^x = 128
int pow = 1;
int x   = 0;

while (pow != 128) {
  pow = pow * 2;
  x = x + 1;
}
```

MIPS assembly code

```
# $s0 = pow, $s1 = x

        addi $s0, $0, 1
        add  $s1, $0, $0
        addi $t0, $0, 128
while:  beq  $s0, $t0, done
        sll  $s0, $s0, 1
        addi $s1, $s1, 1
        j    while
done:
```

```
While (save[i] == k)
    i +=1;
```

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

Assumindo que o loop está alocado inicialmente na posição 80000 na memória, teremos a seguinte sequência de código em linguagem de máquina:

80000	**0**	**0**	**19**	**9**	**4**	**0**
80004	**0**	**9**	**22**	**9**	**0**	**32**
80008	**35**	**9**	**8**	**0**		
80012	**5**	**8**	**21**	**2**		
80016	**8**	**19**	**19**	**1**		
80020	**2**	**20000**				
80024	**.....**					

Operandos MIPS

Nome	Exemplo	Comentários
32 registradores	$s0, $s1, ..., $s7 $t0, $t1, ..., $t7	Locais rápidos para dados. No MIPS, os dados precisam estar em registradores para a realização de operações aritméticas. Os registradores $s0-$s7 são mapeados para 16-23; $t0-$t7 são mapeados para 8-15. O registrador MIPS $zero sempre é igual a 0.
2^{30} words na memória	Memória[0], Memória[4]..... Memória[4294967292]	Acessadas apenas por instruções de transferência de dados no MIPS. O MIPS utiliza endereços em bytes, de modo que os endereços em words seqüenciais diferem em 4 vezes. A memória contém estruturas de dados, arrays e spilled registers.

Assembly do MIPS

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	Add	add $s1,$s2,$s3	$s1 = $s2 + $s3	Três operandos; dados nos registradores
	subtract	sub $s1,$s2,$s3	$s1 = $s2- $s3	Três operandos; dados nos registradores
Transferência de dados	load word	lw $s1,100($s2)	$s1 = Memória[$s2 + 100]	Dados da memória para o registrador
	store word	sw $s1,100($s2)	Memória[$s2 + 100] = $s1	Dados do registrador para a memória
Lógica	And	and $s1,$s2,$s3	$s1 = $s2 & $s3	Três operadores em registrador; AND bit a bit
	Or	or $s1,$s2,$s3	$s1 = $s2 | $s3	Três operadores em registrador; OR bit a bit
	nor	nor $s1,$s2,$s3	$s1 = ~($s2 | $s3)	Três operadores em registrador; NOR bit a bit
	and immediate	andi $s1,$s2,100	$s1 = $s2 & 100	AND bit a bit entre registrador com constante
	or immediate	ori $s1,$s2,100	$s1 = $s2 | 100	OR bit a bit entre registrador com constante
	shift left logical	sll $s1,$s2,10	$s1 = $s2 << 10	Deslocamento à esquerda por constante
	shift right logical	srl $s1,$s2,10	$s1 = $s2 >> 10	Deslocamento à direita por constante
Desvio condicional	branch on equal	beq $s1,$s2,L	if ($s1 == $s2) go to L	Testa igualdade e desvia
	branch on not equal	bne $s1,$s2,L	if ($s1 != $s2) go to L	Testa desigualdade e desvia
	set on less than	slt $s1,$s2,$s3	if ($s2 < $s3) $s1 = 1; else $s1 = 0	Compara menor que; usado com beq, bne
	set on less than immediate	slt $s1,$s2,100	if ($s2 < 100) $s1 = 1; else $s1 = 0	Compara menor que imediato; usado com beq, bne
Desvio incondicional	jump	j L	go to L	Desvia para endereço de destino

FIGURA 2.12 Arquitetura MIPS revelada até a Seção 2.6. As partes destacadas mostram as estruturas MIPS introduzidas na Seção 2.6.

Instruções vistas até o momento

Linguagem de máquina do MIPS

Nome	Formato	Exemplo						Comentários
add	R	0	18	19	17	0	32	add $s1,$s2,$s3
sub	R	0	18	19	17	0	34	sub $s1,$s2,$s3
lw	I	35	18	17	100			lw $s1,100($s2)
sw	I	43	18	17	100			sw $s1,100($s2)
and	R	0	18	19	17	0	36	and $s1,$s2,$s3
or	R	0	18	19	17	0	37	or $s1,$s2,$s3
nor	R	0	18	19	17	0	39	nor $s1,$s2,$s3
andi	I	12	18	17	100			andi $s1,$s2,100
ori	I	13	18	17	100			ori $s1,$s2,100
sll	R	0	0	18	17	10	0	sll $s1,$s2,10
srl	R	0	0	18	17	10	2	srl $s1,$s2,10
beq	I	4	17	18	25			beq $s1,$s2,100
bne	I	5	17	18	25			bne $s1.$s2,100
slt	R	0	18	19	17	0	42	slt $s1,$s2,$s3
j	J	2	2500					j 10000 (ver Seção 2.9)
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções MIPS de 32 bits
Formato R	R	op	rs	rt	rd	shamt	funct	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato para transferências de dados e desvios

FIGURA 2.13 A linguagem de máquina do MIPS revelada até a Seção 2.6. As partes destacadas mostram as estruturas do MIPS introduzidas na Seção 2.6. O formato J, usado para instruções de jump, é explicado na Seção 2.9, que também explica os valores apropriados para os campos de endereço das instruções de desvio.

Resumo da Aula de Hoje

Tópicos mais importantes:

Linguagem Assembly

Microprocessador MIPS

Instruções de controle de fluxo de execução

- **Saltos**
- **Construções de linguagem de alto nível: condicionais e laços.**