Arquitectura dos Computadores

Aula 4 – Desigualdades

Professor: Dr. Christophe Soares

Recordando

A memória é endereçada ao *byte*, mas o *lw* e o *sw* acedem à memória uma *word* de cada vez.

O apontador (usado pelo *lw* e o sw*)* é apenas um endereço de memória, portanto podemos usá-lo em somas e subtrações.

Uma instrução de decisão permite decidir o que executar em tempo real e não durante a escrita do programa.

Recordando

As decisões em C são efetuadas usando *if*, *while*, *do while*, ou *for*.

No MIPS as instrução para decisões são os saltos condicionais beq e bne e o salto incondicional j

Desigualdades no MIPS (1/6)

Até agora só testamos igualdades (== e != em C)

Nos programas também temos que testar <, >, ≤ e ≥

Instrução MIPS para desigualdade:

- **"Set on Less Than" : slt reg1, reg2, reg3**

```
if (reg2 < reg3)
    reg1 = 1;
else reg1 = 0;
```

- **Em assembler, "*set*" quer dizer "*set to 1*", "*reset*" quer dizer "*set to 0*"**

Desigualdades no MIPS (2/6)

Como se usa?

```
if (g < h) goto Less;
```

$s0 – g
$s1 – h

Resposta em assembler do MIPS

```
        slt $t0,$s0,$s1     # $t0 = 1 if g<h
        bne $t0,$0,Less     # goto Less
        ( ... )             # if $t0≠0
                            # (if (g<h))
Less:
```

- **salta se $t0 != 0, logo se g < h**
- **O registo $0 contém sempre o valor 0, frequentemente usado com *bne* e o *beq* depois de uma instrução *slt***
- ***slt* & *bne* permitem implementar a desigualdade <**

Desigualdades no MIPS (3/6)

Sabemos implementar o <, mas como iremos implementar >, ≤ e ≥ ?

Podíamos adicionar mais 3 instruções, mas seria contrário ao objetivo do MIPS : *"Simpler is Better"*

Será que conseguimos implementar ≤, > e ≥ usando < (i.e., o *slt*) e saltos condicionais?

Desigualdades no MIPS (4/6)

Agora com o >:

```
if (g > h) goto GTER;  // (h<g)
```

$s0 – g
$s1 – h

Resposta em assembler do MIPS

```
        slt $t0,$s1,$s0     #$t0=1 if $s1<$s0 (h<g)
                            #(= g>h)
        bne $t0,$0,GTER     #goto GTER if $t0≠0
        ( ... )
GTER:
```

Desigualdades no MIPS (5/6)

Vamos tentar com o ≤:

```
if (g ⩽ h) goto LorEQ; // !(g>h) ⇒ !(h<g)
```

$s0 – g
$s1 – h

Resposta em assembler do MIPS

```
        slt $t0,$s1,$s0     #$t0=1 if $s1<$s0 (h<g)
                            #$t0=0 if h⩾g (= g⩽h)
        beq $t0,$0,LoEQ     #goto LorEQ if $t0==0
        ( ... )
LorEQ:
```

Desigualdades no MIPS (6/6)

Finalmente com o ≥:

```
if (g ⩾ h) goto GorEQ; // !(g<h)
```

$s0 – g
$s1 – h

Resposta em assembler do MIPS

```
        slt $t0,$s0,$s1     #$t0=1 if $s0<$s1 (g<h)
                            #$t0=0 if g⩾h
        beq $t0,$0,GorEQ    #goto GorEQ if $t0==0
        ( ... )
GorEQ:
```

Constantes em desigualdades

Também existe uma versão do slt para fazer testes com constantes: slti (e.g., usado em ciclos)

C

```
if (g ⩾ 1) goto Loop // !(g<1)
```

$s0 – g

MIPS

```
Loop:
        ( ... )
        slti $t0,$s0,1      # $t0 = 1 if
                            # $s0<1 (g<1)
        beq  $t0,$0,Loop    # goto Loop
                            # if $t0==0
                            # (if (g⩾1))
```

E para números sem sinal?

Também existem instruções de desigualdade sem sinal : sltu, sltiu

Colocam o resultado a 0 ou 1, com base em comparações sem sinal

Qual o valor de $t0, $t1?

($s0 = FFFF FFA_{hex}, $s1 = 0000 FFA_{hex})

```
slt  $t0, $s0, $s1
sltu  $t1, $s0, $s1
```

MIPS: Com sinal vs. sem sinal

Atenção! O significado do "u" depende da instrução em causa

- **Estender ou não o sinal**

  lb
  lbu

- **Com/Sem overflow**

  add, addi, sub, mult, div
  addu, addiu, subu, multu, divu

- **Comparar números com/sem Sinal**
  slt, slti
  sltu, sltiu

Exercício 1

```
Loop:   addi $s0,$s0,-1     # i = i - 1
        slti $t0,$s1,2      # $t0 = (j < 2)
        beq  $t0,$0 ,Loop   # goto Loop if $t0 == 0
        slt  $t0,$s1,$s0    # $t0 = (j < i)
        bne  $t0,$0 ,Loop   # goto Loop if $t0 ≠ 0
```

$s0 – i
$s1 – j

Qual o código C que preenche correctamente o espaço?

```
do {i--;} while(__);
```

1:	j	<	2	&&	j	<	i
2:	j	≥	2	&&	j	<	i
3:	j	<	2	&&	j	≥	i
4:	j	≥	2	&&	j	≥	i
5:	j	>	2	&&	j	<	i
6:	j	<	2	||	j	<	i
7:	j	≥	2	||	j	<	i
8:	j	<	2	||	j	≥	i
9:	j	≥	2	||	j	≥	i
10:	j	>	2	||	j	<	i

Exercício 2 (1/2)

Escreva o código MIPS do seguinte programa em C:

```
main() {
    int i=0, j=20, s=0;
    while(i≠j) {
        s=s+i;
        i++;
    }
    for(i=0;i<j;i++) s=s*2;
}
```

$s0 – i
$s1 – j
$s2 – s

Exercício 2 (2/2)

Código do MIPS:

```
        .text
        add $s0, $0, $0     # i=0
        addi $s1, $0, 20    # j=20
        add $s2, $0, $0     # s=0
Loop1:  beq $s0, $s1, End1  # if (i==j) goto End1
        add $s2, $s2, $s0   # s+=i
        addi $s0,  $s0, 1   # i++
        j Loop1
End1:   add $s0, $0, $0     # i=0
Loop2:  slt $t0, $s0, $s1   # $t0=1 if i<j or $t0=0 if i ≥ j
        beq $t0, $0, End2   # if $t0==0 goto End2
        sll $s2, $s2, 1     # s*=2
        addi $s0, $s0, 1    # i++
        j Loop2
End2:   li $v0, 1
        move $a0, $s2
        syscall
        li $v0, 10
        syscall
```

Código em C:

```
main() {
    int i=0, j=20, s=0;
    while(i≠j) {
        s=s+i;
        i++;
    }
    for(i=0;i<j;i++) s=s*2;
}
```

$s0 – i

$s1 – j

$s2 – s

Pseudo-instruções

Branch on less than:

```
blt $s0, $s1, Label # if s0<s1 goto Label
```

Branch on less than unsigned:

```
bltu $s0, $s1, Label # if s0<s1 goto Label (s0 & s1 unsigned)
```

Branch on greater than:

```
bgt $s0, $s1, Label # if s0>s1 goto Label
```

Branch on greater than unsigned:

```
bgtu $s0, $s1, Label # if s0>s1 goto Label (s0 & s1 unsigned)
```

Pseudo-instruções

Branch on less than equal:

```
ble $s0, $s1, Label # if s0 ⩽ s1 goto Label
```

Branch on less than equal unsigned:

```
bleu $s0, $s1, Label # if s0 ⩽ s1 goto Label (s0 & s1 unsigned)
```

Branch on greater than equal:

```
bge $s0, $s1, Label # if s0 ⩾ s1 goto Label
```

Branch on greater than equal unsigned:

```
bgeu $s0, $s1, Label # if s0 ⩾ s1 goto Label (s0 & s1 unsigned)
```

Conclusão...

- **Desigualdades**
  - **<**
  - **>**
  - **<=**
  - **>=**
- **Pseudo-instruções podem ser usadas enquanto não necessitarmos de escrever código máquina (cf. enunciado)!**