Arquitetura de Computadores I

Cap. 02 – Instruções Básicas: A Linguagem da Máquina

(Aula 03)

Plano de aula

- Exercícios
- Loops

Exercícios

- Considerando que as variáveis a, b, c e d estão relacionadas aos registradores $s1 a $s4 ($s6 possui 4), e que o endereço base do vetor V está em $s5, traduza o seguintes trechos de código em linguagem C para a linguagem de montagem do MIPS.

a)
```
if (v[0] != a) {
    v[0] = b;
   }else {
    v[0] = c;
   }
```

b)
```
if (v[b] != v[b+1]) {
    v[b] = v[b+1];
   }
```

c)
```
if (a == b) {
    if (c != (a+d)) {
        a = a + b;
    }else {
        a = a - b;
    }
    c = c + c;
   } else {
    d = d + d;
   }
```

Loops

- Loops podem ser construídos com as mesmas instruções que constroem IF.
  - Ex. Suponha que `A` seja um vetor de 100 elementos e que o compilador associe as variáveis `g, h, i e j` aos registradores $s1, $s2, $s3 e $s4 respectivamente. Vamos admitir que o endereço base do vetor `A` esteja em $s5.

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

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

Loop

- Blocos básicos
  - São sequências de instruções sem desvio, exceto possivelmente no final, e sem labels-alvo de desvios, exceto possivelmente no início.

Loop

- Exemplo 2:

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

Loop:   add $t1, $s3, $s3    #Reg. Temp. $t1 recebe 2 * i
    add $t1, $t1, $t1     #Reg. Temp. $t1 recebe 4 * i
    add $t1, $t1, $s6     #$t1, recebe endereço de save[i]
    lw $t0, 0($t1)        #Reg. Temp. $t0 recebe save[i]
    bne $t0, $s5, Exit    #desvia para Exit se save[i] != k
    add $s3, $s3, $s4     #i recebe i + j
    j Loop             #desvia para Loop
Exit:
```

Outros operadores de comparação

Ok! Mas como realizar comparações do tipo >, <, >= e <= ?

– Tais comparações, no código do MIPS, são realizadas com uma instrução que compara os valores de dois registradores e atribui 1 a um terceiro registrador se:
  - Se o valor do primeiro registrador fonte é menor que o valor do segundo
  - Caso contrário, ela atribui 0 ao terceiro registrador

– Esta instrução no MIPS é chamada "*set on less than*" ou *slt*.
  - Exemplo

```
Slt $t0, $s3, $s4   # $t0 recebe 1 se $s3 < $s4
```

Outros operadores de comparação

- Além disso, existe um registrador chamado `$zero`, que é “read only” (somente leitura) e mantem o valor 0.
- Portanto, o uso combinado de `slt, bne, beq` e o registrador `$zero` é possível construir qualquer operador de comparação
- Ex.
  - Qual o código MIPS para testar se uma variável a, correspondente ao registrador $s0, é menor que outra variável b (registrador $s1) e desviar para o label `Less` se a condição for verdadeira?

```
slt $t0, $s0, $s1  #reg. $t0 recebe 1 se $s0 < $s1 (a < b)
bne $t0, $zero, Less   #desvia para Less se $t0 != 0.
```

Outros operadores de comparação

- Construa o código MIPS para cada comparação abaixo.

- Se $s1 == $s2 vá para L
- Se $s1 != $s2 vá para L
- Se $s1 > $s2 vá para L
- Se $s1 >= $s2 vá para L
- Se $s1 < $s2 vá para L
- Se $s1 <= $s2 vá para L

Outros operadores de comparação

- Se $s1 == $s2 vá para L

```
beq $s1, $s2, L
```

- Se $s1 != $s2 vá para L

```
bne $s1, $s2, L
```

- Se $s1 > $s2 vá para L

```
slt $t0, $s2, $s1
bne $t0, $zero, L
```

- Se $s1 >= $s2 vá para L
  - Equivale a saber se !($s1 < $s2)

```
slt $t0, $s1, $s2
beq $t0, $zero, L
```

- Se $s1 < $s2 vá para L

```
slt $t0, $S1, $s2
bne $t0, $zero, L
```

- Se $s1 <= $s2 vá para L
  - Equivale a saber se !($s2 < $s1)

```
slt $t0, $s2, $s1
beq $t0, $zero, L
```

Atividade

- Traduza o código C em linguagem de montagem do MIPS. Considere que as variáveis a, b, c e d estão relacionadas aos registradores $s1, $s2, $s3 e $s4.

a)
```
if (a < b) {
    if (c > d) {
        a = a – a;
    }
}
b = b + b;
```

b)
```
while (a <= b)
    a = a + c;
```

c)
```
if (a <= b) {
    if (c >= d) {
        a = a – a;
    }else {
        c = d;
    }
}
```

Outro desvio incondicional

- A instrução `jr` (`jump register`) realiza um “salto” para o endereço especificado por um registrador qualquer.
  - Registradores podem armazenar endereços de instruções?
    - Ora, cada registrador pode armazenar uma palavra de dado e os endereços (tanto de instruções quanto de dados) no MIPS cabem em uma palavra. Portanto, a resposta é sim.
  - Ex.

```
jr $s0  #salta para a instrução especificada pelo reg. $s0
```

Loops

Operandos do MIPS

Nome	Exemplo	Comentários
32 registradores	`$s0, $s1, …, $s7` `$t0, $t1, …, $t7, $zero`	Posições de acesso rápido para armazenamento de dados. No MIPS, os dados devem estar em registradores para que as operações aritméticas possam ser realizadas. Os registradores `$s0-$s7` são mapeados nos registradores reais de número 16-23 e os registradores `$t0-$t7` nos de número 8-15. **O registrador `$zero` do MIPS sempre tem o valor 0 armazenado nele.**
2^{30} palavras de memória	Memória[0], Memória[4], …, Memória[4294967292]	No MIPS, estas posições só são acessadas por instruções de transferência de dados. O MIPS endereça byte, de modo que endereços de palavras consecutivas diferem de 4 unidades. A memória armazena estruturas de dados, como arrays, além dos registradores "derramados".

Linguagem de montagem do MIPS

Categoria	Instrução	Exemplo	Significado	Comentários
Aritmética	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Três operandos; dados em registradores
	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Três operandos; dados em registradores
Transferência de dados	load word	`lw $s1, 100($s2)`	`$s1 = Memória[$s2 + 100]`	Dados transferidos da memória para registradores
	store word	`sw $s1, 100($s2)`	`Memória[$s2 + 100] = $s1`	Dados transferidos de registradores para a memória
Desvio condicional	**branch on equal**	`beq $s1, $s2, L`	`se ($s1 == $s2) desvia para L`	**Testar a igualdade e desviar se verdadeira**
	branch on not equal	`bne $s1, $s2, L`	`se ($s1 != $s2) desvia para L`	**Testar a desigualdade e desviar se verdadeira**
	set on less than	`slt $s1, $s2, $s3`	`se($s2 < $s3) $s1 = 1; senão $s1 = 0`	**Compare se menor ou igual; usada junto a `beq` e a `bne`**
Desvio incondicional	**jump**	`j 2500`	`desvia para 10000`	**Desviar para o endereço-alvo**
	jump register	`jr st1`	`desvia para $t1`	**Para comandos *switch***

Loops

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)
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 (veja Seção 3.8)**
jr	**R**	**0**	**9**	**0**	**0**	**0**	**8**	**jr $t1**
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções do MIPS têm 32 bits
Formato R	R	op	rs	rt	rd	shamt	funct	Formato das instruções aritméticas
Forrmato I	I	op	rs	rt	endereço			Formato das **instruções** de transferência de dados

Figura 3.9 Arquitetura do MIPS introduzida na Seção 3.5. As partes em negrito mostram as estruturas da linguagem de montagem do MIPS introduzidas na Seção 3.5. O formato J, usado nas instruções de desvio incondicional, é explicado na Seção 3.8. A Seção 3.8 também explica os valores do campo de endereço das instruções de desvio condicional.

Reflexão

"A única coisa mais cara do que a educação é a ignorância."

Benjamin Franklin
inventor norte americano

Referências

- Hennessy, J. L., Patterson, D. A. *Organização e projeto de computadores: A Interface hardware/software*. 2 ed, Rio de Janeiro, LTC, 2000.