Universidade Federal de Pelotas
Instituto de Física e Matemática
Departamento de Informática
Bacharelado em Ciência da Computação

Arquitetura e Organização de Computadores I

Aula 27

Arquitetura do Processador MIPS: conjunto de instruções e programação em linguagem simbólica

Prof. José Luís Güntzel

guntzel@ufpel.edu.br

www.ufpel.edu.br/~guntzel/AOC1/AOC1.html

Arquitetura do MIPS

Linguagem Simbólica

32 registradores (de 32 bits) de propósito geral

Os registradores são designados por:

- **$s0, $s1, …, $s7**
  - **registradores que correpondem às variáveis dos programas escritos em linguagem de alto nível (C, por exemplo)**
  - **São mapeados nos registradores reais de número 16 a 23**
- **$t0, $t1…, $t7**
  - **registradores temporários, necessários à tradução dos programas em linguagem de alto nível em instruções do MIPS**
  - **São mapeados nos registradores de número 8 a 16**

Arquitetura do MIPS

Instruções Principais

tipo	linguagem de montagem	descrição (1)	descrição (2)
R	add $s1, $s2, $s3	$s1 = $s2 + $s3	adição
R	sub $s1, $s2, $s3	$s1 = $s2 - $s3	subtração
R	or $s1, $s2, $s3	$s1 = $s2 OR $s3	OU
R	and $s1, $s2, $s3	$s1 = $s2 AND $s3	E
lw	lw $s1, offset($s2)	$s1 = Mem[$s2+offset]	carrega registrador s1
sw	sw $s1, offset($s2)	Mem[$s2+offset] = $s1	armazena registrador s1
beq	beq $s1, $s2, offset	if($s1 == $s2) go to PC+4+offset	salto condicional
jump	j offset	jump to target address	salto incondicional

Arquitetura do MIPS

Linguagem Simbólica

Seja o comando C mostrado abaixo.

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

Um possível resultado da compilação deste comando para o MIPS seria:

```
add   $t0, $s1, $s2     # registrador $t0 contém g + h
add   $t1, $s3, $s4     # registrador $t1 contém i + j
sub   $s0, $t0, $t1     # f recebe $t0 - $t1, resultado final
```

Arquitetura do MIPS

Linguagem Simbólica

Acesso a um operando que está na memória

Suponha que:

- A **seja um array de 100 palavras,**
- **O compilador associa as variáveis** g **e** h **aos registradores** $s1 **e** $s2
- **O endereço inicial do array (endereço-base) está armazenado em** $s3**.**

Traduza o seguinte comando de atribuição, escrito em C para a linguagem de montagem do MIPS.

```
g = h + A[8];
```

Arquitetura do MIPS

Instruções Principais

tipo	linguagem de montagem	descrição (1)	descrição (2)
R	add $s1, $s2, $s3	$s1 = $s2 + $s3	adição
R	sub $s1, $s2, $s3	$s1 = $s2 - $s3	subtração
R	or $s1, $s2, $s3	$s1 = $s2 OR $s3	OU
R	and $s1, $s2, $s3	$s1 = $s2 AND $s3	E
lw	lw $s1, offset($s2)	$s1 = Mem[$s2+offset]	carrega registrador s1
sw	sw $s1, offset($s2)	Mem[$s2+offset] = $s1	armazena registrador s1
beq	beq $s1, $s2, offset	if($s1 == $s2) go to PC+4+offset	salto condicional
jump	j offset	jump to target address	salto incondicional

Arquitetura do MIPS

Linguagem Simbólica

Primeiramente, é necessário transferir A[8] **para um registrador:**

```
lw      $t0, 8 ($s3)        # registrador temporário $t0 recebe A[8]
```

A instrução seguinte pode operar normalmente com o valor trazido da memória, pois ele está armazenado no registrador temporário $t0:

```
add   $s1, $s2, $t0       # g recebe h + A[8]
```

Arquitetura do MIPS

A interface Hardware/Software

Sobre o compilador

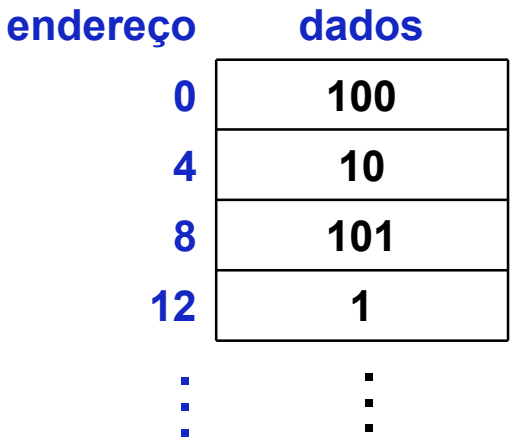
- **O Compilador é responsável por**
  - **associar variáveis a registradores**
  - **alocar em endereços de memória certas estruturas de dados (tais como os arrays)**
  - **otimizar o código gerado**
- **Assim, é fácil para o compilador colocar o endereço inicial nas instruções de transferência de dados**

Arquitetura do MIPS

A interface Hardware/Software

Endereçamento de Memória

- Quase todas as arquiteturas endereçam a memória a bytes
- O endereço de uma palavra deve ser igual ao endereço de um de seus bytes (mas sempre o mesmo)



Arquitetura do MIPS

A interface Hardware/Software

Endereçamento de Memória

- O endereço de duas palavras consecutivas na memória se difere sempre de 4 unidades
- O espaço de endereçamento de memória do MIPS é de 2^{30} palavras (de 32 bits):

endereço	dados
0	100
4	10
8	101
12	1
⋮	⋮
4294967292	77

Arquitetura do MIPS

A interface Hardware/Software

Endereçamento de Memória

- No MIPS as palavras sempre começam em endereços múltiplos de 4 (**restrição de alinhamento**)
- MIPS usa o endereçamento *big endian*

End.	Memória
i	byte3 (mais sig.)
i+1	byte2
i+2	byte1
i+3	byte0 (menos sig.)

- O endereçamento a bytes também afeta a indexação dos arrays: a instrução lw do exemplo anterior precisa ser

```
lw      $t0, 32 ($s3)      # registrador temporário $t0 recebe A[8]
```

Arquitetura do MIPS

Linguagem Simbólica

Uso das Instruções Load e Store

- **Suponha que a variável** h **esteja associada ao registrador** $s2 **e que o endereço do array** A **esteja armazenado em** $s3
- **Qual é o código de montagem do MIPS para o seguinte comando de atribuição, escrito em C?**

A[12] = h + A[8]

```
lw    $t0, 32 ($s3)    # registrador temporário $t0 recebe A[8]
add   $t0, $s2, $t0    # registrador temporário $t0 recebe h + A[8]
sw    $t0, 48 ($s3)    # h + A[8] é armazenado em A[12]
```

Arquitetura do MIPS

Linguagem Simbólica

Usando uma Variável para Indexar Array

Suponha que:

- A **é um array de 100 elementos, cujo endereço-base está armazenado no registrador** $s3
- **O compilador associa as variáveis** g, h **e** i **aos registradores** $s1, $s2 **e** $s4
- **Qual é o código gerado para o MIPS para o seguinte comando de atribuição, escrito em C?**

g = h + A[i]

Arquitetura do MIPS

Linguagem Simbólica

Usando uma Variável para Indexar Array

- **Antes de carregar** A[i] **em um registrador temporário, é preciso conheccer seu endereço**
- **Antes de somar o valor de** i **ao endereço-base do array** A**, é preciso multiplicaro o valor do índice** i **por 4 (endereçamento a byte)**
- **Ao invés de usar a instrução** mul**, usar** add

Arquitetura do MIPS

Linguagem Simbólica

Usando uma Variável para Indexar Array

g = h + A[i]

```
add   $t1, $s4, $s4     # registrador temporário $t1 recebe 2 * i
add   $t1, $t1, $t1     # registrador temporário $t1 recebe 4 * i
add   $t1, $t1, $s3     # registrador temporário $t1 recebe o endereço de A[i]
lw    $t0, 0 ($t1)      # registrador temporário $t0 recebe A[i]
add   $s1, $s2, $t0     # g recebe h + A[i]
```

Arquitetura do MIPS

A interface Hardware/Software

- **Muitos programas usam muito mais variáveis do que o número de registradores que a máquina-alvo possui**
- **O compilador tenta manter nos registradores as variáveis usadas com mais freqüência, deixando as demais na memória (acessáveis via load/store)**
- **O processo de colocar na memória as variáveis menos usadas é chamado de derramamento (*spilling*)**

Arquitetura do MIPS

A interface Hardware/Software

- **Velocidade de acesso a um dado em:**

registrador >> em memória

- **Instruções aritméticas e lógicas no MIPS: fonte e destino em registradores**
- **Instruções de transferência de dados: somente lê um operando ou escreve um operando (não operada sobre dados)**
- **Para obter melhor desempenho, um compilador pecisa usar eficientemente os registradores disponíveis**

Arquitetura do MIPS

Linguagem Simbólica

Instruções de Desvio

- **Usadas para instruções que envolvem tomada de decisão (if, goto etc)**
- **No MIPS:**

```
beq reg1, reg2, L1
```

Branch if equal: desvia para o comando com label L1**, se** reg1 == reg2

```
bne reg1, reg2, L1
```

Branch if not equal: desvia para o comando com label L1**, se** reg1 != reg2

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Comando *if* em uma Instrução de Desvio Condicional

- **Seja código a seguir, escrito em linguagem C:**

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

- **Suponha que as variáveis** f, g, h, i **e** j **sejam alocadas nos registradores** $s0, $s1, $s2, $s3 **e** $s4, **respectivamente**
- **Qual seria o cógdigo gerado pelo compilador do MIPS?**

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Comando *if* em uma Instrução de Desvio Condicional

```
      beq   $s3, $s4, L1     # desvia para L1 se i for igual a j
      add   $s0, $s1, $s2    #  f = g + h (não executa esta instrução se i for igual a j)
L1:   sub   $s0, $s0, $s3    # f = f - i (sempre é executada)
```

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Comando *if-the-else* em Desvios Condicionais

- **Usando as mesmas variáveis do exemplo anterior, obter o código MIPS gerado para o seguinte comando em C:**

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

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Comando *if* em uma Instrução de Desvio Condicional

```
        bne   $s3, $s4, Else   # desvia para Else se i for diferente de j (i ≠ j )
        add   $s0, $s1, $s2    #  f = g + h (salta esta instrução se i ≠ j )
        j     Exit             #  desvia para Exit
Else:   sub   $s0, $s1, $s2    # f = f - h (salta esta instrução se i = j )
 Exit:
```

Arquitetura do MIPS

Linguagem Simbólica

Laços (*Loops*)

- **Os comandos de desvio servem tanto para escolher uma entre duas alternativas (comandos if), quanto para controlar iterações (laços ou *loops*)**

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Laço contendo um Array com Índice Variável

- No laço a seguir, escrito em linguagem C, suponha que
- A seja um array de 100 elementos
- O compilador associa as variáveis f, g, h, i e j aos registradores $s0, $s1, $s2, $s3 e $s4, respectivamente

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

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Laço contendo um Array com Índice Variável

```
Loop:  add   $t1, $s3, $s3      # registrador temporário $t1 recebe 2 * i
       add   $t1, $t1, $t1      # registrador temporário $t1 recebe 4* i
       add   $t1, $t1, $s5      # $t1 recebe o endereço de A[ i ]
       lw    $t0, 0 ($t1)       # registrador temporário $t0 recebe A[ i ]
       add   $s1, $s1, $t0      # g recebe = g + A[ i ]
       add   $t1, $t1, $s5      # i recebe = i + j
       bne   $s3, $s2, Loop     # desvia para Loop se i ≠ j
```

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Laço *While*

- **Em linguagem C, evita-se usar o *go to***
- **Então, um laço normalmente encontrado em C seria**

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

- **Suponha que** i, j, **e** k **correspondam aos registradores** $s3, $s4 **e** $s5, **respectivamente, e que o endereço-base do array** save **esteja em** $s6
- **Qual seria o código MIPS para este laço?**

Arquitetura do MIPS

Linguagem Simbólica

Compilação de um Laço contendo um Array com Índice Variável

```
Loop:  add   $t1, $s3, $s3      # registrador temporário $t1 recebe 2 * i
       add   $t1, $t1, $t1      # registrador temporário $t1 recebe 4* i
       add   $t1, $t1, $s6      # $t1 recebe o endereço de save[ i ]
       lw    $t0, 0 ($t1)       # registrador temporário $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:
```

Arquitetura do MIPS

Linguagem Simbólica

Instrução *set on less than* (slt)

- O testes de igualdade ou desigualdade são os mais populares dentre todos os testes de condição
- Mas às vezes é preciso verificar se o valor de uma variável é menor do que o de outra (exemplo, testar se um índice é menor que zero)
- *Set on less than* (slt): slt reg1, reg2, reg3
- Compara reg2 com reg3,
- Se reg2 < reg3, reg1 □ 1
- Caso contrário, reg1 □ 0

Arquitetura do MIPS

Linguagem Simbólica

Instrução *set on less than* (slt)

- **Os compiladores para o MIPS usam as instruções** slt, bne **e** beq, **juntamente com o valor fixo** $zero **(registrador read-only** $0**) para criar as condições relativas:**
  - **Igual**
  - **Não-igual**
  - **Menor que**
  - **Menor ou igual a**
  - **Maior que**
  - **Maior ou igual a**

Arquitetura do MIPS

Linguagem Simbólica

Instrução *set on less than* (slt)

- **Qual é o código do MIPS para testar se uma variável a (alocada em** $s0**) é menor que a variável b (alocada em** $s1**) e desviar para o label** Less **se a condição for verdadeira?**

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