[AULA 03] Conjunto de instruções 2

Prof. João F. Mari
joaof.mari@ufv.br

Roteiro

- Introdução
- Operações no hardware do computador
- Operandos do hardware do computador
- Representando instruções no computador
- Operações lógicas
- Instruções para tomada de decisões
- Suporte para procedimentos no hardware do computador

REPRESENTANDO INSTRUÇÕES NO COMPUTADOR

Representando instruções no computador

- Quando programa-se em *assembly* utiliza-se mnemônicos (como `lw`, `sw`, `add` e `sub`);
  - Porém, instruções são representadas e executadas através de um conjunto de bits.
- Como os registradores são parte de quase todas as instruções,
  - É preciso haver uma convenção para mapear os nomes dos registradores em números binários:
    - `$s0` a `$s7` são mapeados nos registradores de 16 a 23;
    - `$t0` a `$t7` são mapeados nos registradores de 8 a 15;

Representando instruções no computador

- Traduzindo uma instrução *assembly* MIPS para uma instrução de máquina:
  - `add $t0, $s1, $s2`

31 – 26	*25 – 21*	*20 – 16*	*15 – 11*	*10 – 6*	*5 – 0*
opcode	**rs**	**rt**	**rd**	**shamt**	**funct**
0	17	18	8	0	32

- Cada segmentos de uma instrução é chamado **campo**:
  - O **primeiro** e o **último** campos (**opcode** e **funct**) combinados:
    - Dizem ao computador MIPS que essa instrução realiza soma;
  - O **segundo** campo (**rs**): indica o número do registrador que é o primeiro operando de origem da operação de soma (17 = `$s1`);
  - O **terceiro** campo (**rt**): indica o outro operando de origem (18 = `$s2`);
  - O **quarto** campo (**rd**): contém o número do registrador que receberá o resultado (8 = `$t0`);
  - O **quinto** campo (**shamt**): não é empregado nessa instrução.

Representando instruções no computador

- **Problemas de endereçamento:**
  - Quando uma instrução precisa de **campos maiores** do que aqueles mostrados.
    - **[EX]** A instrução lw precisa especificar dois registradores e uma constante; se o endereço tivesse apenas 5 bits do formato anterior, a constante estaria limitada a 32 (2^5).
  - Existe um **conflito** entre o desejo de manter todas as instruções com o mesmo tamanho e o desejo de ter uma instrução único.
  - O **compromisso** escolhido pelos projetistas do MIPS é manter todas as instruções com o mesmo tamanho:
    - Exigindo diferentes formatos para os campos de diferentes tipos de instruções
  - O formato anterior é chamado de tipo R (de registrador) ou **formato R**.
  - Um segundo tipo de formato de instrução é chamado de **formato I**:
    - Utilizando pelas instruções imediatas e de transferência de dados.

- **Princípio de Projeto 4**
  - Um bom projeto exige bons compromissos

Representando instruções no computador

- O formato **I** (imediato):
  - O endereço possui 16 bits:
    - Uma instrução `lw` pode carregar qualquer palavra (*word*) dentro de uma região de +/- 2^{15} do endereço do registrador base (8.192 palavras ou 32.768 bytes)
  - De modo semelhante, a soma imediata (`addi`) é limitada a constantes que não sejam maiores do que 2^{15} (em magnitude).
  - `lw $t0, 32($s3)`
    - Aqui, 19 (para `$s3`) é colocado no campo **rs**, 8 (para `$t0`) é colocado no campo **rt** e 32 é colocado no campo de endereço:
  - O formato mudou: o campo **rt** especifica o registrador de destino, que recebe o resultado do `lw`.

opcode	**rs**	**rt**	**Endereço (constante)**
31 – 26	25 – 21	20 – 16	15 – 0
6 bits	*5 bits*	*5 bits*	*16 bits*

[EX] Assembly do MIPS para linguagem de máquina

- Se `$t1` armazena a base do *array* `A` e `$s2` corresponde a `h`:
  - `A[300] = h + A[300];`

- É compilado para:
  - `lw $t0, 1200($t1) # reg. $t0 recebe A[300]`
  - `add $t0, $s2, $t0 # reg. $t0 recebe h+A[300]`
  - `sw $t0, 1200($t1) # armazena h+A[300] na mem.`

	opcode	rs	rt	rd/ endereço	shamt/ endereço	funct/ endereço
`lw $t0, 1200($t1)`	35	9	8	1200		
`add $t0, $s2, $t0`	0	18	8	8	0	32
`sw $t0, 1200($t1)`	43	9	8	1200		

[EX] Assembly do MIPS para linguagem de máquina

	opcode	rs	rt	rd/ endereço	shamt/ endereço	funct/ endereço
`lw $t0, 1200($t1)`	35	9	8	1200		
`add $t0, $s2, $t0`	0	18	8	8	0	32
`sw $t0, 1200($t1)`	43	9	8	1200		

- **Observações:**
  - A instrução `lw` é representada por 35 no **opcode**.
  - O registrador base 9 (`$t1`) é especificado no segundo campo (**rs**).
  - O registrador de destino 8 (`$t0`) é especificado no terceiro campo (**rt**).
  - O *offset* para selecionar `A[300]` (1200 = 300 × 4) aparece no campo final.
  - A instrução `add` é especificada com **opcode** 0 e **funct** 32.
  - A instrução `sw` é identificada com 43 no **opcode**.
- IMPORTANTE: Os valores estão representados em **decimal**, mas na verdade são representados em **binário**.

OPERAÇÕES LÓGICAS

Operações lógicas

- O projeto dos primeiros computadores se concentrava em palavras completas:
  - Entretanto, é útil atuar sobre campos de bits de uma palavra.
  - Operações lógicas são uteis para **empacotar** e **desempacotar** grupos de bits em palavras.

Operações lógicas	**Operadores C**	**Operadores Java**	**Instruções MIPS**
Shift à esquerda	<<	<<	`sll`
Shift à direita	>>	>>	`srl`
AND bit a bit	&	&	`and, andi`
OR bit a bit	|	|	`or, ori`
NOT bit a bit	~	~	`nor`

Operações lógicas

- Operações de deslocamento (*shifts*)
  - Movem todos os bits de uma palavra para esquerda ou para direita;
    - Preenche com zero os bits que ficaram vazios.

- *Shift left logical* (deslocamento lógico à esquerda):
  - `sll $t2, $s0, 4`
  - `# reg $t2 = reg $s0 << 4 bits`

  - O campo **shamt** (*shift amount*) da instrução MIPS:
    - Usado nas instruções de deslocamento.
  - `ssl` é codificada com zeros nos campos **op** e **funct;**
    - **rd** contém `$t2`, **rt** contém `$s0` e **shamt** contém 4.
  - **rs** não é utilizado.

31 – 26	*25 – 21*	*20 – 16*	*15 – 11*	*10 – 6*	*5 – 0*
opcode	**rs**	**rt**	**rd**	**shamt**	**funct**
0	0	16	10	4	0

Operações lógicas

- Operações de deslocamento:
  - O deslocamento lógico à esquerda de *i* bits **equivale** a multiplicar por 2^i.
- Operações lógicas:
  - A operação AND é útil para isolar bits de uma palavra (operações com máscara) :
    - `and $t0, $t1, $t2 # $t0 = $t1 & $t2`
  - Para colocar um bit em 1 em um grupo de bits, pode-se utilizar a operação OR:
    - `or $t0, $t1, $t2 # $t0 = $t1 | $t2`

Operações lógicas

- Os projetistas do MIPS decidiram incluir a instrução NOR no lugar de NOT:
  - Se um operando for zero, a instrução é equivalente a NOT:
    - A NOR 0 = NOT (A OR 0) = NOT (A)
    - `nor $t0, $t1, $t3  # $t0 = ~($t1 | $t3)`
    - `nor $t0, $t1, $zero  # $t0 = ~$t1`

- O MIPS oferece instruções lógicas para trabalhar com constantes (modo de endereçamento imediato):
  - AND imediato (`andi`);
  - OR imediato (`ori`).

INSTRUÇÕES PARA TOMADA DE DECISÃO

Instruções para tomadas de decisão

- O que diferencia o computador de uma calculadora simples é a capacidade de tomar decisões:
  - Em linguagens de alto nível: `if`, `else`, `goto`, ...

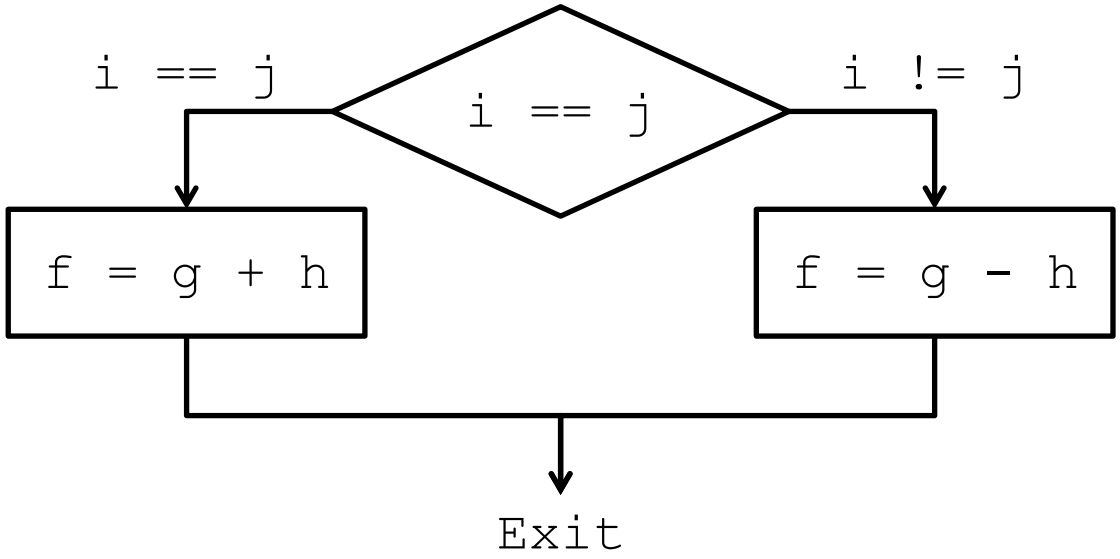
- Instruções de desvio condicional no MIPS
  - ***Branch if equal*** (desvie se igual)
    - `beq registrador1, registrador2, L1`
  - Essa instrução significa ir até a instrução rotulada por `L1` se o valor no `registrador1` for igual ao valor no `registrador2;`
  - O rótulo, `L1`, é alterado pelo compilador por endereços de memória.

Instruções para tomadas de decisão

```
        bne $s3, $s4,Else
        add $s0, $s1, $s2
        j Exit
Else:   sub $s0, $s1, $s2
Exit:   # o código continua...
```

- A instrução `j` implementa o desvio incondicional (*jump*).

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



Instruções para tomadas de decisão

- Interface hardware/software
  - Compiladores criam estruturas mais próximas a linguagem humana:
    - `while`, `do until`, etc.
  - Compilando um *loop* `while` em C:

```
1. while (save[i] == k)
2.     i += 1;
```

  - Suponha que `i` e `k` correspondam aos registradores `$s3` e `$s5` e a base do vetor `save` esteja em `$s6`.
    - Qual o código em MIPS que corresponde a esse segmento C?

Instruções para tomadas de decisão

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

- `i` e `k` correspondem aos registradores `$s3` e `$s5`;
- A base do vetor `save` esteja em `$s6`.

```
Loop: sll  $t1, $s3, 2     # $t1 = 4 * i
      add  $t1, $t1, $s6   # $t1 = endereço de
                           # save[i]
      lw   $t0, 0($t1)     # $t0 = save[i]
      bne  $t0, $s5, Exit  # Vá para Exit se
                           # save[i] <> k
      addi $s3, $s3, 1     # i += 1
      j Loop
Exit: # o código continua...
```

Instruções para tomadas de decisão

- Bloco básico:
  - Uma sequencia de instruções sem desvios (exceto, possivelmente no final) e sem destinos de desvio ou rótulos de desvio (exceto, possivelmente, no início)
  - *Uma das primeiras fases da compilação é desmembrar o programa em blocos básicos*.

- Testes de igualdade em assembly MIPS
  - Comparações são realizadas de forma que a instrução compara dois registradores e atribui 1 a um terceiro registrador se o primeiro for menor que o segundo; caso contrário, é atribuído 0.

  - *Set on less than* (atribuir se menor que):
    - `slt $t0, $s3, $s4`
      - É atribuído 1 ao registrador `$t0` se o valor no registrador `$s3` for menor do que o valor no registrador `$s4`.

Instruções para tomadas de decisão

- Testes de igualdade em assembly MIPS:
  - Operadores constantes são populares nas comparações;
  - Como o registrador `$zero` sempre tem 0, pode-se comparar com zero; para comparar com outros valores, existe uma versão com endereçamento imediato da instrução `slt`:
    - `slti $t0, $s2, 10  # $t0 = 1 se $s2 < 10`

- Interface hardware/software
  - Os compiladores MIPS utilizam as instruções `slt`, `slti`, `beq`, `bne` e o valor fixo 0 (`$zero`) para criar todas as condições relativas:
    - `==; !=; <; >; <=; >=.`
  - Dessa forma, as construções lógicas de linguagens de alto nível, como C e Java, são mapeadas em instruções *assembly* de desvio condicional.

Instruções para tomadas de decisão

C	Assembly #1

`if (a<b) {`
`  < instruções T >`
`} else {`
`  < instruções F >`
`}`
`// a em $s0`
`// b em $s1`
	`  slt $t0, $s0, $s1 # $t0=1 se $s0<$s1 (a<b)`
	`  beq $t0, $zero, ELSE  # Desvia se $t0=0`
	`  < bloco de instruções T >`
	`  j END  # Evita o bloco F.`
	`ELSE:`
	`  < bloco de instruções F >`
	`END:`

`if (a>b) {`
`  < instruções T >`
`} else {`
`  < instruções F >`
`}`
`// T - Verdadeiro`
`// F - Falso`
	`  slt $t0, $s1, $s0 # $t0=1 se $s1<$s0 ((b<a) == (a>b))`
	`  beq $t0, $zero, ELSE  # Desvia se $t0==0`
	`  < bloco de instruções T >`
	`  j END  # Evita o bloco F.`
	`ELSE:`
	`  < bloco de instruções F >`
	`END:`

`if (a>=b) {`
`  < instruções T >`
`} else {`
`  < instruções F >`
`}`
	`  slt $t0, $s0, $s1 # $t0=1 se $s0<$s1 ((a>=b) == !(a<b))`
	`  **bne** $t0, $zero, ELSE  # Desvia se $t0!=0 ($t0==1)`
	`  < bloco de instruções T >`
	`  j END  # Evita o bloco F.`
	`ELSE:`
	`  < bloco de instruções F >`
	`END:`

`if (a<=b) {`
`  < instruções T >`
`} else {`
`  < instruções F >`
`}`
	`  slt $t0, $s1, $s0 # $t0=1 se $s1<$s0 ((a<=b) == !(a>b))`
	`  **bne** $t0, $zero, ELSE # Desvia se $t0!=0 ($t0==1)`
	`  < bloco de instruções T >`
	`  j END  # Evita o bloco F.`
	`ELSE:`
	`  < bloco de instruções F >`
	`END:`

Assembly do MIPS

Categoria	Instrução	Exemplo	Significado
Aritmética	Add	`add $s1, $s2, $s3`	$s1 = $s2 + $s3
	Subtract	`sub $s1, $s2, $s3`	$s1 = $s2 - $s3
Transf. de Dados	Load word	`lw $s1, 100($s2)`	$s1 = Memória[$s2 + 100]
	Store word	`sw $s1, 100($s2)`	Memória[$s2 + 100] = $s1
Lógica	And	`and $s1, $s2, $s3`	$s1 = $s2 & $s3
	Or	`or $s1, $s2, $s3`	$s1 = $s2 | $s3
	Nor	`nor $s1, $s2, $s3`	$s1 = ~($s2 | $s3)
	And immediate	`andi $s1, $s2, 100`	$s1 = $s2 & 100
	Or immediate	`ori $s1, $s2, 100`	$s1 = $s2 | 100
	Shift left logical	`sll $s1, $s2, 10`	$s1 = $s2 << 10
	Shift righ logical	`srl $s1, $s2, 10`	$s1 = $s2 >> 10
Desvio condicional	Branch on equal	`beq $s1, $s2, L`	If ($s1 == $s2) go to L
	Branch on not equal	`bnq $s1, $s2, L`	If ($s1 != $s2) go to L
	Set on less than	`slt $s1, $s2, $s3`	If ($s2 < $s3) $s1 = 1 else $s1 = 0
	Set on less than immediate	`slti $s1, $s2, 100`	If ($s2 < 100) $s1 = 1 else $s1 = 0
Desvio incondicional	Jump	`j L`	Go to L

BIBLIOGRAFIA

- PATTERSON, D.A; HENNESSY, J.L. **Organização e Projeto de Computadores: A Interface Hardware/Software.** 3a. Ed. Elsevier, 2005.
  - Capítulo 2.

- Notas de aula do prof. Luciano J. Senger:
  - http://www.ljsenger.net/classroom.html

[FIM]

- FIM:
  - **[AULA 03]** Conjunto de instruções 2
- Próxima aula:
  - **[AULA 04]** Conjunto de instruções 3