Arquitetura de Computadores I

Cap. 02 – Instruções Básicas: A Linguagem da Máquina

(Aula 02)

Plano de aula

- Representação de Instruções
- Conceito de programa armazenado
- Instruções de Desvio

Representação de Instruções

- Os números podem ser representados em qualquer base
  - Ex.: $123_{10} = 1111011_2$

- Os números são mantidos no hardware do computador como um conjunto de sinais eletrônicos
  - Os número são considerados na base 2. Isto é, *números binários*
  - Podem assumir somente 2 valores: Alto e Baixo
  - 1 dígito binário é conhecido como bit e é o átomo da computação.
  - 1 bit assume então apenas 1 de 2 valores: Alto, baixo; ligado, desligado ou 0 e 1

Representação de Instruções

- As instruções também são mantidas no computador como uma série de sinais eletrônicos altos e baixos
  - Podem portanto, ser representadas como números
  - Na verdade, cada parte de uma instrução é representada como um número separado
    - A colocação de tais números, um ao lado do outro forma a instrução

Representação de Instruções

- Há uma convenção para mapear os registradores em números
  - `$S0` a `$S7` são mapeados nos registradores de 16 a 23
  - `$t0` a `$t7` são mapeados nos registradores de 8 a 15.
  - Ex.: `$s0` significa o registrador 16, `$s1` o registrador 17 e assim por diante

Representação de Instruções

- Exemplo:
- Linguagem de montagem
  - `add, $t0, $s1, $s2`
- Linguagem de máquina em decimal

0	17	18	8	0	32

- Linguagem de máquina em binário

000000	10001	10010	01000	00000	100000
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

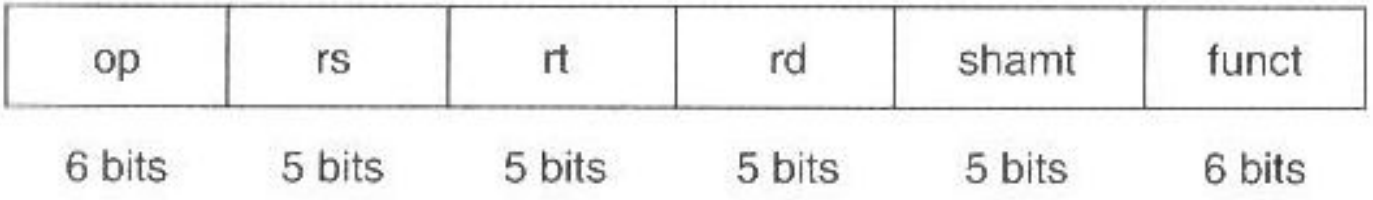
  - **Primeiro** e **Sexto** campos são combinados para informar ao processador que a instrução deseja fazer uma soma
  - **Segundo** campo é o primeiro registrador fonte
  - **Terceiro** campo é o segundo registrador fonte
  - **Quarto** campo é o registrador destino
  - **Quinto** campo não é usado nesse tipo de instrução

Atividade

- Converta as seguintes instruções para a linguagem de máquina do MIPS
  - `add $s3, $s1, $s2`
  - `add $s2, $s2, $s2`
  - `add $t0, $t1, $s5`

Representação de Instruções

- A instruções do MPIS possuem exatamente 32 bits, o mesmo tamanho de uma palavra de memória e de um dado.
- Os campos das instruções de máquina do MIPS recebem nomes
- Abaixo segue o que chamamos de *formato de instrução*



- **op:** a operação básica a ser realizada pela instrução (código de operação)
- **rs:** o registrador contendo o primeiro operando-fonte
- **rt:** o registrador contendo o segundo operando-fonte
- **rd:** o registrador que guarda o resultado da operação, (registrador destino)
- **shamt:** deslocamento (será explicado futuramente)
- **funct:** Função: Uma variação específica da operação apontada no campo op

Representação de Instruções

- A instrução `lw` precisa especificar 2 registradores e uma constante.

- O que aconteceria se o campo de constante usasse apenas os 5 bits reservados aos campos de registrador?
  - A constante estaria limitada a 2^5 - 1= 31.

Portanto temos um conflito de desejos!!!

Manter apenas 1 formato de instrução mas com instruções de tamanhos variados

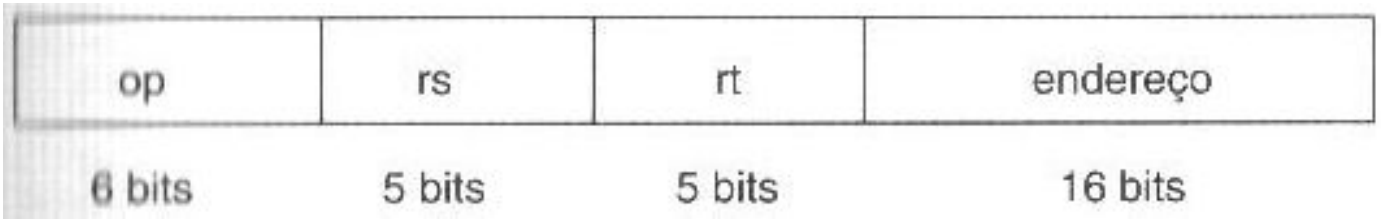
ou

Possuir vários formatos e manter o tamanho fixo em 1 palavra

Representação de Instruções

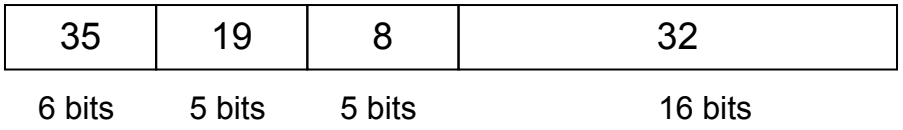
Princípios de projeto 3: Um bom projeto demanda compromisso

- O compromisso escolhido pelos projetistas do MIPS foi manter todas instruções do mesmo tamanho (1 palavra)
  - Surgindo formatos diferentes para tipos diferentes de instruções.
  - Formato **tipo R**: O formato apresentado anteriormente)
  - Formato **tipo I**: Usado na representação de instruções de transferência de dados



Representação de Instruções

- Ex. lw $t0, 32($s3)



  - **op:** Código de operação
  - **rs:** registrador base
  - **rt:** registrador destino
  - **endereço:** deslocamento

- Observações.
  - Os dois formatos (R e I) possuem os mesmo 3 campos iniciais.
  - O tamanho do último campo do formato I coincide com a soma dos 3 campos finais do formato R.
  - O processador utiliza o valor presente no campo `op` para diferenciar os 2 formatos.

Representação de Instruções

Instrução	Formato	op	rs	rt	rd	shamt	funct	endereço
add	R	0	reg	reg	reg	0	32	n.-a.
sub (subtract)	R	0	reg	reg	reg	0	34	n.-a.
lw (load word)	I	35	reg	reg	n.-a.	n.-a.	n.-a.	endereço
sw (store word)	I	43	reg	reg	n.-a.	n.-a.	n.-a.	endereço

Figura 3.5 Codificação das instruções do MIPS. Na tabela acima, "reg" significa o número de um registrador entre 0 e 31, "endereço" um valor de 16 bits, e, "n.-a." (não-aplicável) significa que o campo em questão não existe nesse formato. Note que as instruções add e sub têm o mesmo valor no campo *op*; o hardware usa o campo funct para decidir qual a operação a ser executada: soma (32) ou subtração (34).

Representação de Instruções

- Exemplo
  - Suponha que $t1 tem o valor-base do array A, e que $s2 corresponde a h, o comando de atribuição escrito em C mostrado a seguir

```
A[300] = h + A[300];
```

  - É compilado em

```
lw $t0, 1200($t1) #Reg. temp. $t0 recebe A[300]
add $t0, $s2, $t0 #Reg. temp. $t0 recebe h + A[300]
sw $t0, 1200($t1) #h + A[300] é armazenado de volta na mem. em A[300]
```

Representação de Instruções

```
lw $t0, 1200($t1) #Reg. temp. $t0 recebe A[300]
add $t0, $s2, $t0 #Reg. temp. $t0 recebe h + A[300]
sw $t0, 1200($t1) #h + A[300] é armazenado de volta na mem. em A[300]
```

- Qual a representação do código acima em linguagem de máquina?

op	rs	rt	rd	endereço/ shamt	funct
35	9	8	1200		
0	18	8	8	0	32
43	9	8	1200		

10°011	01001	01000	0000 0100 1011 0000		
000000	10010	01000	01000	00000	100000
10¹011	01001	01000	0000 0100 1011 0000		

Representação de Instruções

Nome	Exemplo	Comentários
32 registradores	`$s0, $s1, ..., $s7` `$t0, $t1, ..., $t7`	Posições de acesso rápido para armazenamento de dados. No MIPS, os dados devem estar em registradores para que as operações aritméticas possam ser realizadas. **Os registradores** `$s0-$s7` **são mapeados nos registradores reais de número 16-23 e os registradores** `$t0-$t7` **nos de número 8-15.**
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

Linguagem de máquina do MIPS

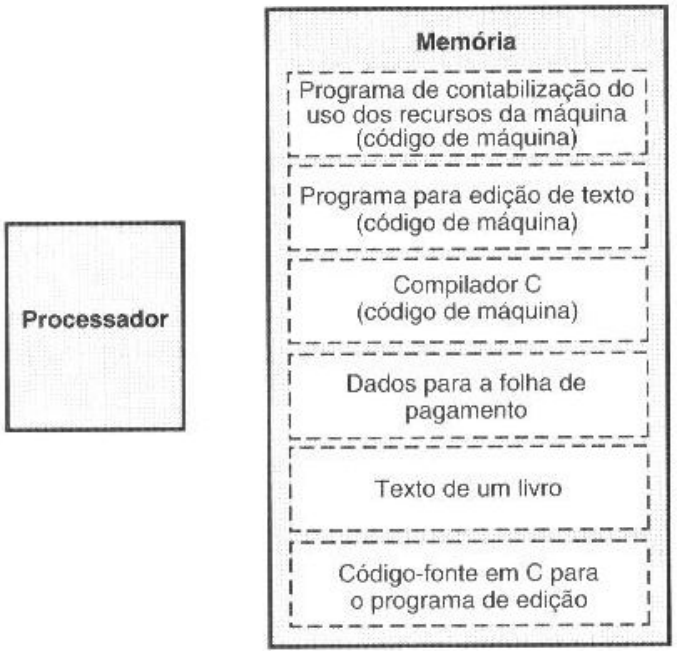
Nome	Formato	Exemplo						Comentários
`add`	R	0	18	19	17	0	32	`add $s1, $s2, $s3`
`sub`	R	0	18	19	17	0	34	`sub $s1, $s2, $s3`
`lw`	I	35	18	17	100			`lw $s1, 100($s2)`
`sw`	I	43	18	17	100			`sw $s1, 100($s2)`
Tamanho do campo		6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	Todas as instruções do MIPS têm 32 bits
Formato R	R	op	rs	rt	rd	shamt	funct	Formato das instruções aritméticas
Formato I	I	op	rs	rt	endereço			Formato das instruções de transferência de dados

Atividade

- Traduza o seguinte programa em linguagem de montagem, para a linguagem de máquina do MIPS. Utilize a notação decimal e depois a notação binária

```
lw $t0, 0($s0)    #Reg. temp. $t0 recebe um dado de memória
lw $t1, 4($s0)    #Reg. temp. $t1 recebe o dado seguinte de mem.
sub $t2, $t0, $t1     #Reg. Temp. $t2 recebe $t0 + $t1
sw $t2, 8($s0)    #Memória recebe $t0+$t1
```

Conceito de programa armazenado



- A construção dos computadores de hoje obedece a dois princípios
  - As instruções são representadas como se fossem números
  - Os programas devem ser armazenados na memória antes de serem executados

Instruções de Desvio

A diferença entre um computador e uma simples calculadora eletrônica é a capacidade de escolher entre diversos caminhos de execução.

- Em outras palavras, com base em dados de entrada e em valores criados durante a execução do programa, diferentes instruções poderão vir a ser executadas.

- Instruções de *desvio condicional*

  – Instrução de desvio **`beq`** `(branch if equal)`

```
beq registrador1, registrador2, L1

#Se o valor contido no registrador1 é igual ao valor
#contido no registrador 2, então "pule" a execução para
#a instrução "marcada" como L1
```

Instruções de Desvio

– Instrução de desvio **bne** (branch if not equal)

```
bne registrador1, registrador2, L1

#Se o valor contido no registrador 1 é diferente do
#valor contido no registrador 2 “pule” para a instrução
#“marcada” como L1
```

Instruções de Desvio

- Exemplo:

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

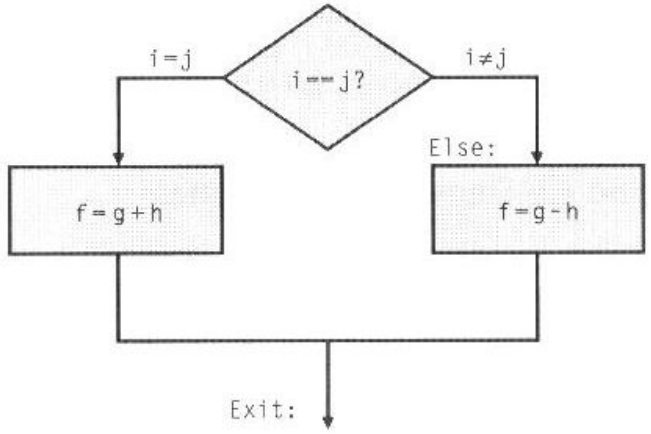
Supondo que as cinco variáveis de f até j correspondem aos registradores de $s0 a $s4, qual o código MIPS gerado pelo compilador?

```
    beq $s3, $s4, L1
    add $s0, $s1, $s2
L1: sub $s0, $s0, $s3
```

Instruções de Desvio

- Compilação do comando if-then-else em desvios condicionais
- Exemplo: (Considere as mesmas variáveis e regs. Do exemplo anterior)

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




```
      bne $s3, $s4, Else    #desvia para Else se i != j
      add $s0, $s1, $s2     #f = g + h (salta essa instrução se i !=
   j)
      j Exit                #desvia incondicionalmente para Exit
Else: sub $s0, $s1, $s2     #f = g - h (salta essa instrução se i ==
   j)
Exit:
```

Atividade

- Supondo que as quatro variáveis de **a** até **d** correspondem aos registradores de **$s0** a **$s3**, qual o código MIPS gerado pelo compilador para o trecho em linguagem C abaixo?

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

Reflexão

"Grandes realizações são possíveis quando se dá importância aos pequenos começos."

Lao-Tsé

Filósofo Chinês

Referências

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