1º Semestre de 2007/2008

Bernardo Cunha, José Luís Azevedo, Arnaldo Oliveira

Aula 4

Aspectos chave da arquitectura do MIPS

Instruções e classes de instruções

Princípios básicos de projecto de uma Arquitectura

Instruções aritméticas

Representação de instruções no MIPS:

- Instruções do tipo R

Introdução: A máquina e a sua linguagem

Princípios básicos dos computadores actuais:

- As instruções são representadas da mesma forma que os números
- Os programas podem ser armazenados em memória, para serem lidos e escritos, tal como os números

Estes princípios formam os fundamentos do conceito da arquitectura ***stored-program***

- O conceito de *stored-program* pressupõe que num mesmo espaço de armazenamento co-residam informações de natureza tão variada como sejam o código fonte de um programa em C, um editor de texto, um compilador, e o próprio programa resultante da compilação

ISA: Instruções e classes de instruções

"It is easy to see by formal-logical methods that there exist certain instruction sets that are in abstract adequate to control and cause the execution of any sequence of operations... The really decisive considerations from the present point of view, in selecting an instruction set, are more of a practical nature: simplicity of the equipment demanded by the instruction set, and the clarity of its application to the actually important problems together with the speed of its handling of those problems"

Burks, Goldstine and von Neumann, 1947

Instruções e classes de instruções

Questão:

- Será, portanto, possível considerar a existência de um grupo limitado de classes de instruções que possam ser consideradas comuns à generalidade das arquitecturas?

Exemplo: *There must certainly be instructions for performing the fundamental arithmetic operations*

Burks, Goldstine and von Neumann, 1947

Instruções e classes de instruções

Questão:

- Quais serão então essas classes?

Resposta:

- Instruções de processamento (aritméticas e lógicas)
- Instruções de transferência de informação
- Instruções de controlo de fluxo de execução

- No projecto de um processador a definição do set de instruções exige um compromisso entre:
  - as facilidades oferecidas aos programadores (por ex. instruções de manipulação de *strings*)
  - a complexidade do hardware envolvido na sua implementação
- Quatro princípios básicos estão subjacentes a um bom *design* ao nível hardware:

1. **A simplicidade favorece a regularidade**
2. **Quanto mais pequeno mais rápido**
3. **O que é mais comum deve ser mais rápido**
4. **Um bom *design* implica compromissos adequados**

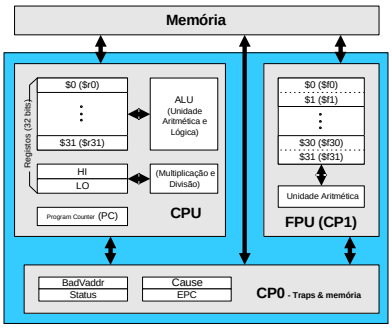
1. A simplicidade favorece a regularidade
- Ex1: todas as instruções do set de instruções são codificadas com o mesmo número de bits
- Ex2: instruções aritméticas operam sempre sobre registos internos e depositam o resultado também num registo interno

2. Quanto mais pequeno mais rápido

3. O que é mais comum deve ser mais rápido
- Ex: quando o operando é uma constante ela deve fazer parte da instrução (é vulgar que mais de 50% das instruções que envolvem a ALU num programa sejam constantes)

4. Um bom *design* implica compromissos adequados
- Ex: o compromisso que resulta entre a possibilidade de se poder codificar constantes de maior dimensão nas instruções e a manutenção da dimensão fixa nas instruções

Aspectos chave da arquitectura MIPS:



- 32 Registos de uso geral de 32 bits
- ISA baseado em instruções de dimensão fixa (32 bits)
- Barramento de endereço externo de 32 bits (2^{32} endereços possíveis)
- Organização de memória em bytes (memória *byte addressable*)
- Barramento de dados externo de 32 bits (Word)
- *Register-register operation (load-store machine)*

1. Instruções para operar aritmeticamente

Operações básicas: **soma e subtracção**:

Formato tipo de uma instrução *Assembly* do Mips:

```
add   a, b, c     # Soma b com c e armazena o resultado em a
```

Exemplo: Uma adição do tipo z = a + b + c + d

tem de ser decomposta em

```
add   z, a, b     # Soma a com b e armazena o resultado em z
add   z, z, c     # Soma z com c e armazena o resultado em z
add   z, z, d     # Soma z com d e armazena o resultado em z
```

Operações aritméticas

Subtracção:

Formato tipo de uma instrução *Assembly* do Mips:

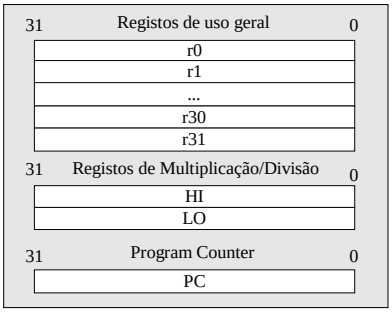
sub a, b, c # Subtrai **c** a **b** e armazena o resultado em **a**

Exemplo: A operação z = (a + b) – (c + d)

tem de ser decomposta em

```
add   t1, a, b    # Soma a com b e armazena o resultado em t1
add   t2, c, d    # Soma c com d e armazena o resultado em t2
sub   z, t1, t2   # Subtrai t2 a t1 e armazena o resultado em z
```


Os registos do MIPS:



Porquê 32?

Exemplo anterior revisitado:

```
add   $8, $17, $18    # Soma $17 com $18 e armazena o resultado em $8
add   $9, $19, $20    # Soma $19 com $20 e armazena o resultado em $9
sub   $16, $8, $9     # Subtrai $9 a $8 e armazena o resultado em $16
```

O equivalente em C

```
// a é $17, b é $18 c é $19, d é $20 e z é $16
// $8 e $9 representam variáveis temporárias não explicitadas em C

        int  a, b, c , d , z;
        z = (a + b) – (c + d);
```

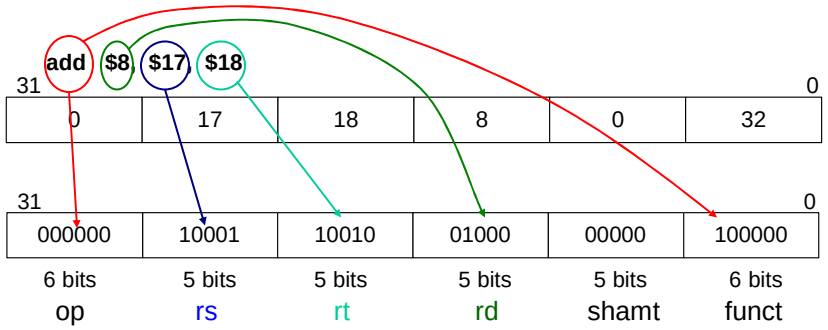

Representação de instruções no MIPS

Instruções do Tipo R

Campos da instrução:

- **op**: *opcode* (0 nas instruções tipo R)
- **rs**: Nº do registo que contém o 1º operando fonte
- **rt**: Nº do registo que contém o 2º operando fonte
- **rd**: Nº do registo onde o resultado vai ser armazenado
- **shamt**: *shift amount* (útil em instruções de deslocamento)
- **funct**: código da operação a realizar

Representação de instruções no MIPS



add rd, rs, rt

Instruções lógicas:

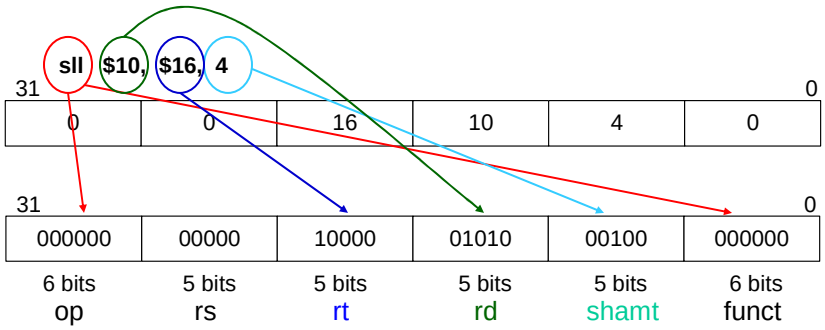
- and Rdst, Rsrc1, Rsrc2 # Rdst = Rsrc1 **&** Rsrc2
- or Rdst, Rsrc1, Rsrc2 # Rdst = Rsrc1 **|** Rsrc2
- nor Rdst, Rsrc1, Rsrc2 # Rdst = **~**(Rsrc1 **|** Rsrc2)
- xor Rdst, Rsrc1, Rsrc2 # Rdst = (Rsrc1 **^** Rsrc2)

- sll Rdst, Rsrc, k # Rdst = Rsrc << k
- srl Rdst, Rsrc, k # Rdst = Rsrc >> k

Operadores lógicos *bitwise* em C:

- **&** - AND
- **|** - OR
- **^** - XOR
- **~** - NOT

Representação de instruções no MIPS



sll rd, rt, shamt

2. Instruções de transferência de informação

Transferência entre registos internos

- Transferência entre registos internos: Rdst = Rsrc
- Registo $0 do MIPS tem sempre o valor 0x00000000 (apenas pode ser lido)
- Utilizando o registo $0 e a instrução lógica OR é possível realizar a operação de transferência:
  - **or Rdst, $0, Rsrc** # Rdst = 0 + Rsrc = Rsrc
- Instrução da **máquina virtual** que pode ser utilizada com vantagem do ponto de vista da legibilidade:
  - **move Rdst, Rsrc** # Rdst = Rsrc