Arquitectura de Sistemas e Computadores I

Semana 2

Miguel Barão
mjsb@di.uevora.pt

u évora
ESCOLA DE CIÊNCIAS E TECNOLOGIA
DEPARTAMENTO DE INFORMÁTICA

Resumo

- Diferença entre arquitectura e implementação.
- Processador: registos, program counter. Exemplo de execução de um conjunto de instruções ADD para somar e multiplicar números inteiros.

Definição (Arquitectura)

É a descrição funcional de um computador.
Descreve a ISA (*Instruction Set Architecture*), que é a imagem do hardware quando se programa em código máquina. Consiste na especificação dos registos disponíveis para serem usados num programa, instruções suportadas e respectiva função, etc. Não especifica coisas como o tamanho da memória cache.

Definição (Arquitectura)

É a descrição funcional de um computador.
Descreve a ISA (*Instruction Set Architecture*), que é a imagem do hardware quando se programa em código máquina. Consiste na especificação dos registos disponíveis para serem usados num programa, instruções suportadas e respectiva função, etc. Não especifica coisas como o tamanho da memória cache.

Definição (Implementação da arquitectura)

Diz respeito à implementação em hardware das especificações da arquitectura. Podem existir várias implementações diferentes da mesma arquitectura (vários fabricantes produzem CPUs diferentes mas compatíveis entre si)

Definição (Arquitectura)

É a descrição funcional de um computador.
Descreve a ISA (*Instruction Set Architecture*), que é a imagem do hardware quando se programa em código máquina. Consiste na especificação dos registos disponíveis para serem usados num programa, instruções suportadas e respectiva função, etc. Não especifica coisas como o tamanho da memória cache.

Definição (Implementação da arquitectura)

Diz respeito à implementação em hardware das especificações da arquitectura. Podem existir várias implementações diferentes da mesma arquitectura (vários fabricantes produzem CPUs diferentes mas compatíveis entre si)

Nesta disciplina vamos focar-nos apenas na arquitectura MIPS32.

Processador: registos

Um processador é tipicamente composto por:

Registos genéricos. Existem em número limitado. Permitem armazenar números temporariamente. (MIPS tem 32 registos de 32 bits.)

Processador: registos

Um processador é tipicamente composto por:

Registos genéricos. Existem em número limitado. Permitem armazenar números temporariamente. (MIPS tem 32 registos de 32 bits.)

Registos especializados São registos cuja função é armazenar números com significados específicos (*e.g.*, *"program counter"*, códigos de erros quando são executadas instruções inválidas, etc)

Processador: registos

Um processador é tipicamente composto por:

Registos genéricos. Existem em número limitado. Permitem armazenar números temporariamente. (MIPS tem 32 registos de 32 bits.)

Registos especializados São registos cuja função é armazenar números com significados específicos (*e.g.*, *"program counter"*, códigos de erros quando são executadas instruções inválidas, etc)

Os registos podem ser acessíveis apenas para leitura, apenas para escrita ou para leitura e escrita.

Processador: registos MIPS

0	zero	16	s0
1	at	17	s1
2	v0	18	s2
3	v1	19	s3
4	a0	20	s4
5	a1	21	s5
6	a2	22	s6
7	a3	23	s7
8	t0	24	t8
9	t1	25	t9
10	t2	26	k0
11	t3	27	k1
12	t4	28	gp
13	t5	29	sp
14	t6	30	fp
15	t7	31	ra

pc

hi

lo

MIPS32: 32 registos de 32 bits

Exemplo: alguns dos registos MIPS

$zero vale $0x00000000$. Apenas permite leitura. A escrita neste registo não tem efeito.

Exemplo: alguns dos registos MIPS

$zero vale $0x00000000$. Apenas permite leitura. A escrita neste registo não tem efeito.

$a0 – $a3 são usados para passar argumentos para funções. (iremos estudar mais adiante)

Exemplo: alguns dos registos MIPS

$zero vale $0x00000000$. Apenas permite leitura. A escrita neste registo não tem efeito.

$a0 – $a3 são usados para passar argumentos para funções. (iremos estudar mais adiante)

$v0 – $v1 são usados para devolver o resultado de uma função. (iremos estudar mais adiante)

Exemplo: alguns dos registos MIPS

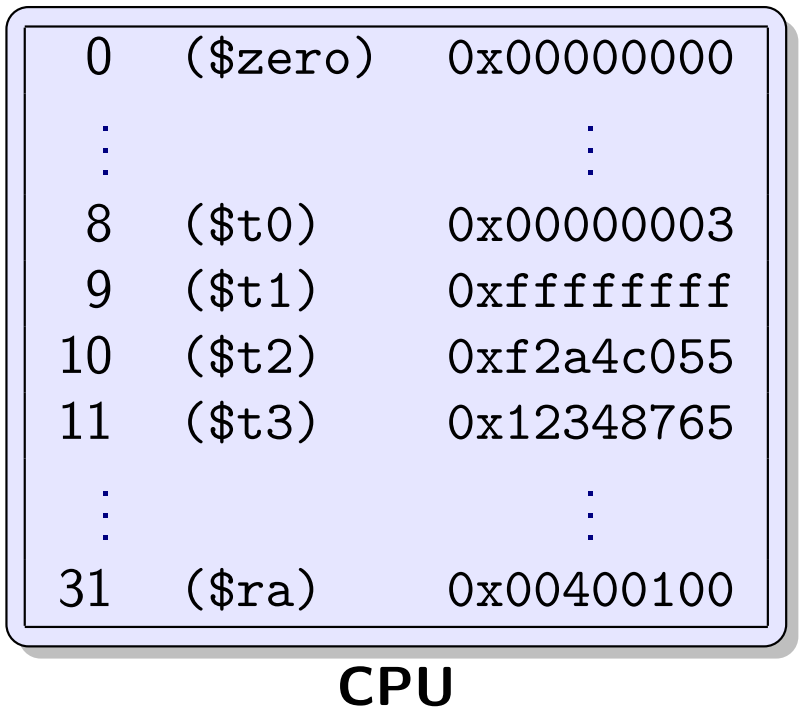
$zero vale $0x00000000$. Apenas permite leitura. A escrita neste registo não tem efeito.

$a0 – $a3 são usados para passar argumentos para funções. (iremos estudar mais adiante)

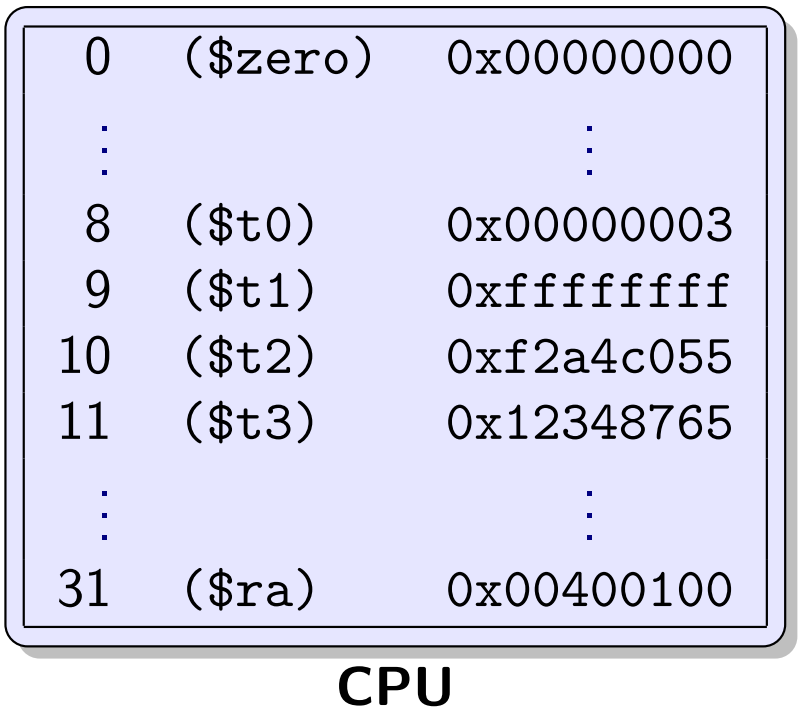
$v0 – $v1 são usados para devolver o resultado de uma função. (iremos estudar mais adiante)

$t0 – $t9 são registos temporários para serem usados nas aplicações.

Exemplo de uma soma entre registos



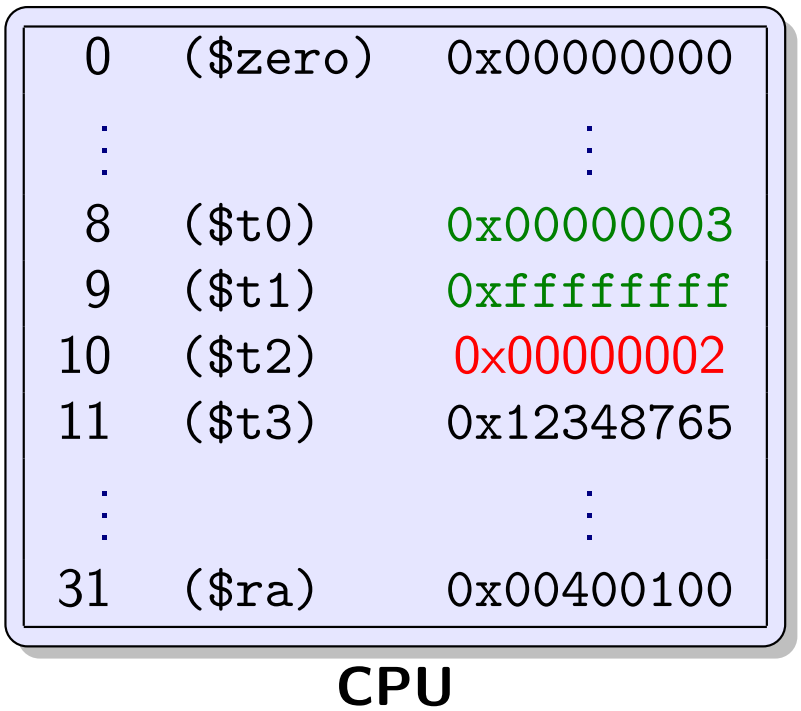
Exemplo de uma soma entre registos



Pseudocódigo:

Somar `$t0 + $t1` e colocar o resultado em `$t2`.

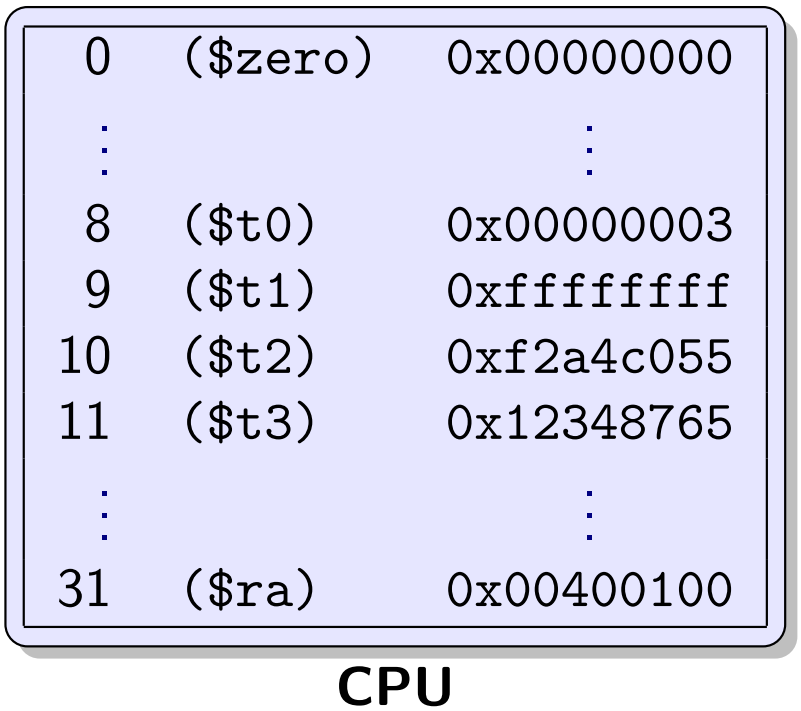
Exemplo de uma soma entre registos



Pseudocódigo:

Somar `$t0 + $t1` e colocar o resultado em `$t2`.

Exemplo de uma soma entre registos



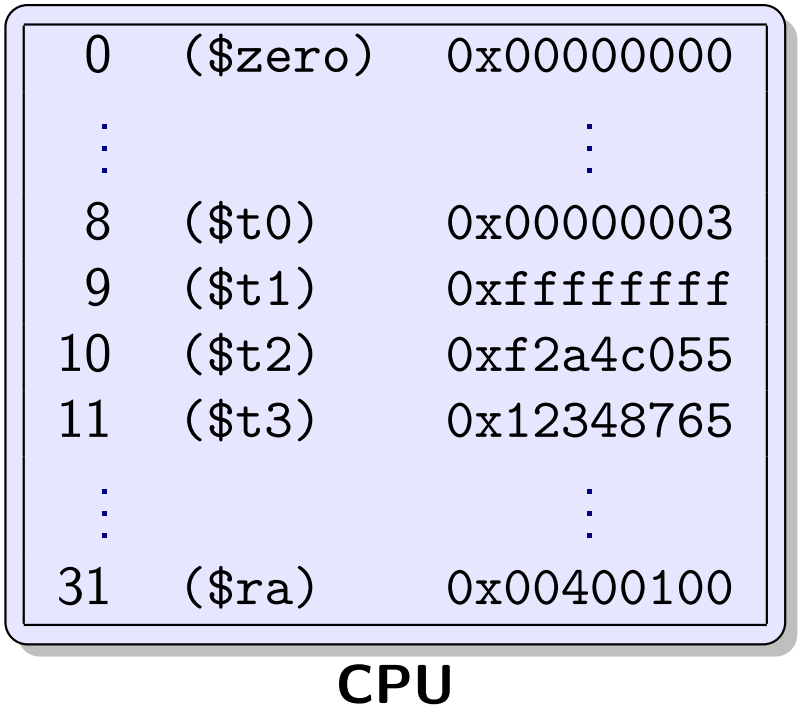
Pseudocódigo:

Somar `$t0 + $t1` e colocar o resultado em `$t2`.

Código máquina:

`0x01095020`

Exemplo de uma soma entre registos



Pseudocódigo:

Somar `$t0 + $t1` e colocar o resultado em `$t2`.

Código máquina:

`0x01095020`

Assembly:

```
add $t2, $t0, $t1
```

Execução da instrução de soma

instrução em memória (little endian), fetch, execução.

Ver no quadro de giz $\longrightarrow$

Um programa um pouco maior

Pretende-se multiplicar $t1 = 3 \times t0$.

Um programa um pouco maior

Pretende-se multiplicar $t1 = 3 \times t0$.

Solução usando somas (pseudocódigo):

- $t1 \leftarrow t0 + t0$
- $t1 \leftarrow t1 + t0$

Um programa um pouco maior

Pretende-se multiplicar $t1 = 3 \times t0$.

Solução usando somas (pseudocódigo):

- $t1 \leftarrow t0 + t0$
- $t1 \leftarrow t1 + t0$

Solução usando somas:

Um programa um pouco maior

Pretende-se multiplicar $t1 = 3 \times t0$.

Solução usando somas (pseudocódigo):

- $t1 \leftarrow t0 + t0$
- $t1 \leftarrow t1 + t0$

Solução usando somas:

Assembly

```
add $t1, $t0, $t0
add $t1, $t1, $t0
```

Um programa ainda maior

Pretende-se multiplicar $t1 = 5 \times t0$.

Um programa ainda maior

Pretende-se multiplicar $t1 = 5 \times t0$.

Solução usando somas e shifts (pseudocódigo):

- $t1 \leftarrow$ shift left t0 by 2 bits
- $t1 \leftarrow t1 + t0$

Um programa ainda maior

Pretende-se multiplicar $t1 = 5 \times t0$.

Solução usando somas e shifts (pseudocódigo):

- $t1 \leftarrow$ shift left t0 by 2 bits
- $t1 \leftarrow t1 + t0$

Assembly

```
sll $t1, $t0, 2
add $t1, $t1, $t0
```