Arquitetura de Computadores I

Instruções executáveis por um processador - a arquitetura MIPS

Representação das instruções

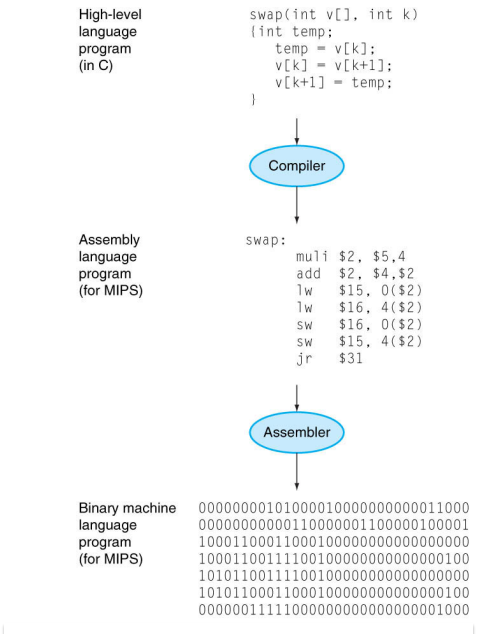
António de Brito Ferrari

ferrari@ua.pt

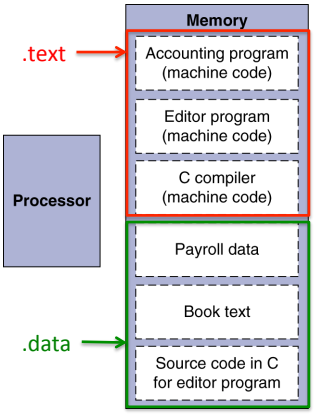
C

Assembly

Linguagem Máquina



The BIG Picture



- Instruções representadas em binário, tal como os dados
- Instruções e dados armazenados na memória
- Programas podem operar sobre programas
  - compilers, linkers, assembler…
- Compatibilidade Binária permite aos programas compilados serem executados em diferentes computadores
  - *Standardized ISAs*

Codificação das instruções

Instruções codificadas em binário: ***código máquina***

Instruções MIPS

Codificadas em 32-bits (***instruction words***)

– comprimento fixo

- Número reduzido de formatos de instrução

Especificam:

Operação a executar – ***opcode*** (código de operação)

Operandos - número dos registos ou endereço de memória

Número dos registos e sua designação em assembly

$t0 – $t7 são os registos r8 – r15

$t8 – $t9 são os registos r24 – r25

$s0 – $s7 são os registos r16 – r23

Formato das instruções aritméticas e lógicas: Tipo R

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

Campos do código de Instrução:

- **op**: operation code (***opcode***)
- **rs**: número do registo do 1º operando
- **rt**: número do registo do 2º operando
- **rd**: número do registo de destino
- **shamt**: shift amount (≠ 0 só nas instruções de shift)
- **funct**: function code (extende o opcode)

Codificação das Instruções tipo-R Exemplos

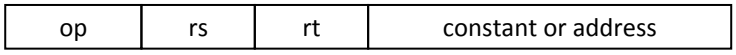
Assembly Code / Field Values / Machine Code

Assembly Code	op	rs	rt	rd	shamt	funct	Machine Code	
add $t0, $s4, $s5	0	20	21	8	0	32	000000 10100 10101 01000 00000 100000	(0x02954020)
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	0 2 9 5 4 0 2 0	

Assembly Code / Field Values / Machine Code

Assembly Code	op	rs	rt	rd	shamt	funct	Machine Code	
add $s0, $s1, $s2	0	17	18	16	0	32	000000 10001 10010 10000 00000 100000	(0x02328020)
sub $t0, $t3, $t5	0	11	13	8	0	34	000000 01011 01101 01000 00000 100010	(0x016D4022)
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits 5 bits 5 bits 5 bits 5 bits 6 bits	

Formatos de instrução: Tipo I



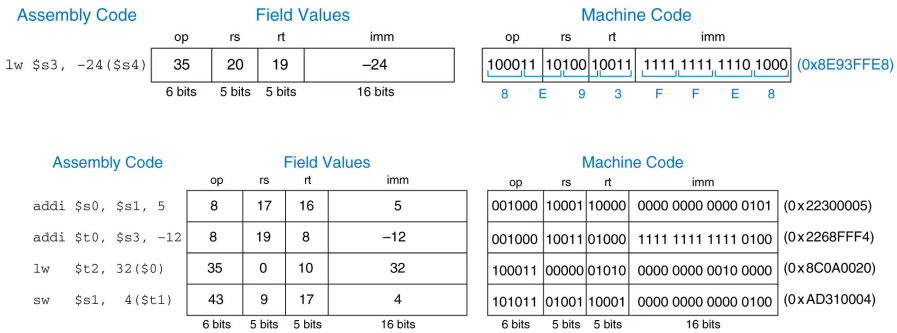
- Instruções ***load*** e ***store*** e operações com imediatos
  - **rs**: *base register* em load/store, operando nas operações com imediato
  - **rt**: *load/store* – registo para/de onde a transferência é feita operações com imediatos – registo destino

  constant: **-2^{15} a $+2^{15} - 1$**

  address: **Endereço de memória = (rs) + offset**

  Design Principle: Good design demands good compromises
  - Diferentes Formatos complicam a descodificação, mas permitem que todas as instruções sejam codificadas em 32-bits
  - Procurar que os formatos sejam tão semelhantes quanto possível

Codificação das instruções Tipo-I



Codificação das instruções: Tipo R e Tipo I

Instruction	Format	op	rs	rt	rd	shamt	funct	address
add	R	0	reg	reg	reg	0	32_{ten}	n.a.
sub (subtract)	R	0	reg	reg	reg	0	34_{ten}	n.a.
add immediate	I	8_{ten}	reg	reg	n.a.	n.a.	n.a.	constant
lw (load word)	I	35_{ten}	reg	reg	n.a.	n.a.	n.a.	address
sw (store word)	I	43_{ten}	reg	reg	n.a.	n.a.	n.a.	address

Tipo R – instruções aritméticas e lógicas
Tipo I – loads, stores e operações com imediatos

Operações Lógicas

- Instruções para manipulação de bits

Operação	C	Java	MIPS
Shift left	<<	<<	`sll`
Shift right	>>	>>>	`srl`
AND bit-a-bit	&&	&&	`and, andi`
OR bit-a-bit	|	|	`or, ori`
NOT bit-a-bit	~	~	`nor`

  - Uteis para extrair e inserir grupos de bits numa *word*

Operações Lógicas

	Source Registers							
$s1	1111	1111	1111	1111	0000	0000	0000	0000
$s2	0100	0110	1010	0001	1111	0000	1011	0111

Assembly Code	Result								
`and $s3, $s1, $s2`	$s3	0100	0110	1010	0001	0000	0000	0000	0000
`or  $s4, $s1, $s2`	$s4	1111	1111	1111	1111	1111	0000	1011	0111
`xor $s5, $s1, $s2`	$s5	1011	1001	0101	1110	1111	0000	1011	0111
`nor $s6, $s1, $s2`	$s6	0000	0000	0000	0000	0000	1111	0100	1000

AND

- Útil para mascarar (selecionar) bits numa *word*
  - Seleciona alguns bits, coloca os outros a 0

    AND(x,1) = x AND(x,0) = 0

`and $t0, $t1, $t2`

$t2	0000 0000 0000 0000 0000 1101 1100 0000
$t1	0000 0000 0000 0000 0011 1100 0000 0000
$t0	0000 0000 0000 0000 0000 1100 0000 0000

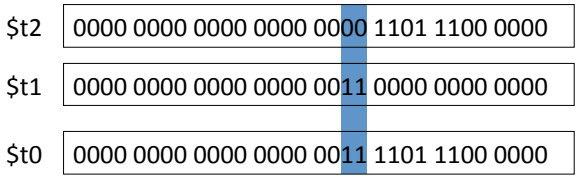
- **`andi $t0, $t1, const`** – *and immediate*

  (**`$t0`**) **`=`** (**`$t1`**) **`AND const`** – constante de 16-bits 0-extended

OR

- Util para incluir bits numa *word*
  - Coloca alguns bits a 1, os restantes não se alteram

```
or $t0, $t1, $t2
```



- **`ori $t0, $t1, const`** – *or immediate*
  (**`$t0`**) **`=`** (**`$t1`**) **`OR const`** – constante de 16-bits 0-extended

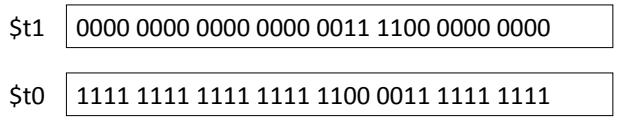
NOT

- Util para inverter bits numa *word*
  - Muda 0 para 1, e 1 para 0
- MIPS tem NOR 3-operand instruction

  a NOR b == NOT (a OR b); NOT (a OR 0) = NOT (a)

```
nor $t0, $t1, $zero
```

Register 0: sempre lido como zero



Instruções de deslocamento (***shift***)

op	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

- ***shamt***: indica quantas posições deslocar; ***rs* = 0**
- Shift left logical
  - Shift left e preenche com zeros bits à direita
  - *sll* *i* bits multiplica por 2^i
- Shift right logical
  - Shift right e preencher com zeros bits à esquerda
  - *srl* *i* bits divide (**operandos *unsigned***) por 2^i
- Shift right arithmetic
  - Shift right e preencher com o bit de sinal os bits à esquerda
  - *sra* *i* bits divide (**operandos *signed***) por 2^i

Assembly Code	Field Values						Machine Code						
	op	rs	rt	rd	shamt	funct	op	rs	rt	rd	shamt	funct	
`sll $t0, $s1, 4`	0	0	17	8	4	0	000000	00000	10001	01000	00100	000000	(0x00114100)
`srl $s2, $s1, 4`	0	0	17	18	4	2	000000	00000	10001	10010	00100	000010	(0x00119102)
`sra $s3, $s1, 4`	0	0	17	19	4	3	000000	00000	10001	10011	00100	000011	(0x00119903)
	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	6 bits	5 bits	5 bits	5 bits	5 bits	6 bits	

Figure 6.16 Shift instruction machine code

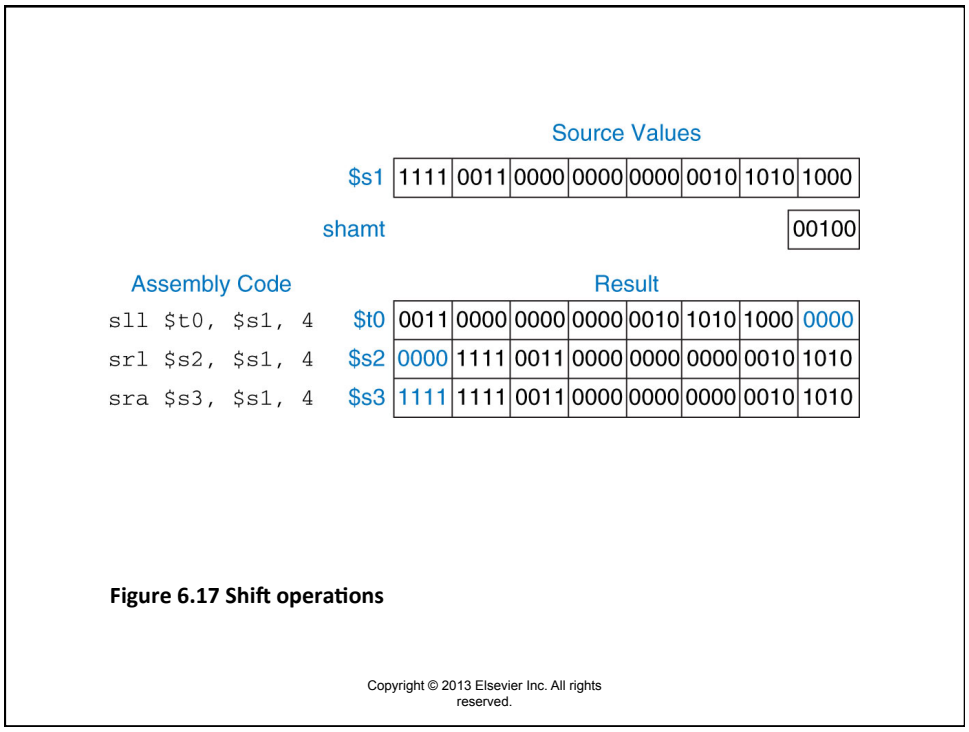


Figure 6.17 Shift operations