Universidade Federal do Rio Grande do Sul
Instituto de Informática

Organização de Computadores

Aula 3

Arquitetura do processador MIPS

Arquitetura do processador MIPS

1. **Introdução e História do MIPS**
3. **Registradores**
5. **Tipos de dados**
7. **Modos de endereçamento**
9. **Formatos das instruções**
11. **Tipos de instruções**

1. História do MIPS

- **Em 1981 John L. Hennessy com um grupo da Stanford University iniciou o projeto do primeiro processador MIPS. O conceito básico era incrementar o desempenho usando pipelines de instruções profundos.**
- **Um dos principais aspectos do projeto do MIPS era que todas instruções tomassem somente um ciclo para sua execução.**
- **Este projeto levou a eliminação de um conjunto de instruções, tais como multiplicação e divisão que tomavam múltiplos ciclos para sua execução.**
- **A eliminação destas instruções levou a diversos observadores argumentarem que trocar instruções complexas por diversas instruções simples não trariam ganho de velocidade. Esta análise não levou em conta que o ganho maior do projeto vinha do uso dos pipelines e não das instruções.**

História do MIPS

- **Em 1984 Hennessy convenceu-se do futuro comercial do projeto e abandonou a Universidade de Stanford para criar a MIPS Computer Systems. Em 1985 surge o R2000, depois em 1988 é lançado o R3000. Estas CPUs de 32-bit formaram a base da Companhia, e foram empregadas intensamente nas wokstations da SGI.**

- **In 1991 MIPS lançou o primeiro microprocessador de 64-bit, o R4000. No entanto a MIPS teve dificuldades em abrir seu espaço comercial. Por outro lado o projeto era tão importante para a SGI, que esta comprou a companhia em 1992. Tornou-se uma subsidiária da SGI e conhecida por MIPS Technologies.**

- **No final dos anos 1990, MIPS iniciou o licenciamento de seus projetos para terceiros. Provando o sucesso da simplicidade do core, permitindo seu uso em diversas aplicações que preferiram usar menos instruções CISC.**

- **Outro aspecto é o preço da CPU que normalmente é relacionado com o número de portas e número de pinos externos**

- **Posteriormente, Sun Microsystems licenciou seu core SPARC. Pelos anos 1990s MIPS obteve sucesso na área de processadores embarcados e em 1997 a CPU MIPS com 48-milhões, tornou-se a primeira CPU RISC da família Motorola 68000.**

Família MIPS

MIPS microprocessor specifications

Model	Frequency [MHz]	Year	Process [µm]	Transistors [millions]	Die size [mm²]	IO Pins	Power [W]	Voltage	Dcache [k]	Icache [k]	Scache [k]
R2000	8-16.7	1985	2.0	0.11	--	--	--	--	32	64	none
R3000	20-40	1988	1.2	0.11	66.12	145	4	--	64	64	none
R4000	100	1991	0.8	1.35	213	179	15	5	8	8	1024
R4400	100-250	1992	0.6	2.3	186	179	15	5	16	16	1024
R4600	100-133	1994	0.64	2.2	77	179	4.6	5	16	16	512
R5000	150-200	1996	0.35	3.7	84	223	10	3.3	32	32	1024
R8000	75-90	1994	0.5	2.6	299	591	30	3.3	16	16	1024
R10000	150-250	1995	0.35	6.8	299	599	30	3.3	32	32	512
R12000	270-400	1998	0.18–0.25	6.9	204	600	20	4	32	32	1024
R14000	500-600	2001	0.13	7.2	204	527	17	--	32	32	2048
R16000	700-800	2002	0.11	--	--	--	20	--	64	64	4096

Introdução: Microarquitetura de processador

- **É composta de um caminho de dados e uma unidade de controle.**
  - **O caminho de dados é composto de vários componentes construídos a partir dos componentes básicos da lógica digital (portas, circuitos lógicos etc.)**
  - **A unidade de controle indica o caminho pelo qual os dados de uma instrução devem trafegar e como estes dados devem ser processados no caminho de dados.**

- **Os principais componentes de uma microarquitetura são:**
  - **o contador de programa ou *program counter* (*PC*),**
  - **memória de instruções e dados (memória cache),**
  - **banco de registradores,**
  - **ULA (Unidade Lógica Aritmética) e a**
  - **unidade de controle.**

- **A microarquitetura é responsável pela busca, decodificação e execução de instruções. As instruções são os comandos enviados para o processador. Neste curso analisaremos a microarquitetura do processador MIPS.**

Introdução: Microarquitetura de Processador

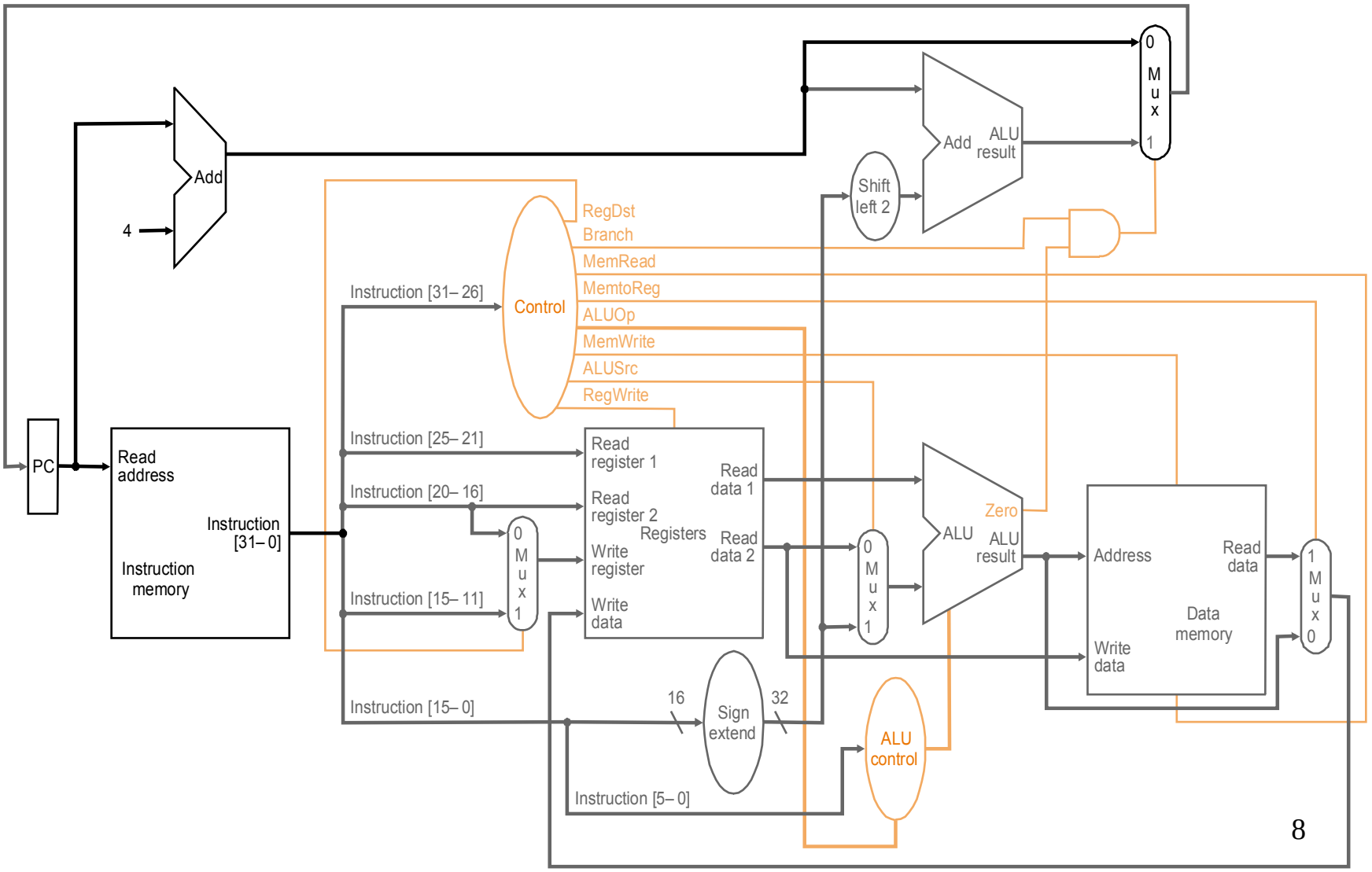
- **PC (program counter):** armazena a posição de memória, onde se encontra a próxima instrução a ser executada.

- **Memória de Dados e de Instruções:** é a memória cache, na qual se armazenam os dados e instruções mais utilizados.

- **Banco de Registradores:** conjunto de registradores que armazenam os dados que estão sendo processados no momento. O MIPS possui 32 registradores de 32 bits, portanto ele trabalha com dados e 4 bytes ou uma palavra.

- **ULA (Unidade Lógica e Aritmética):** dadas algumas entradas de dados, ela realiza operações lógicas (e, ou etc.) e aritméticas (soma, subtração etc.) sobre estes dados.

Introdução: Microarquitetura do MIPS



2. Registradores

- **32 registradores de propósitos gerais de 32 bits**
  - **$0, $1, …, $31**
  - **operações inteiras**
  - **Endereçamento**
- **$0 tem sempre valor 0**
- **$31 guarda endereço de retorno de sub-rotina**
- **32 registradores de ponto flutuante de 32 bits (precisão simples) $f0, $f1, …, $f31**
  - **podem ser usados em pares para precisão dupla**
- **Registradores Hi e Lo para uso em multiplicação e divisão**

Registradores do MIPS

Name	Register number	Usage
$zero	0	the constant value 0
$v0-$v1	2-3	values for results and expression evaluation
$a0-$a3	4-7	arguments
$t0-$t7	8-15	temporaries
$s0-$s7	16-23	saved
$t8-$t9	24-25	more temporaries
$gp	28	global pointer
$sp	29	stack pointer
$fp	30	frame pointer
$ra	31	return address

3. Tipos de dados

- **Dados inteiros disponíveis em instruções load e store**
  - **bytes**
  - **meias-palavras de 16 bits**
  - **palavras de 32 bits**

- **Dados inteiros disponíveis em instruções aritméticas e lógicas**
  - **meias-palavras de 16 bits (estendidos para 32 bits)**
  - **palavras de 32 bits**

- **Dados em ponto flutuante**
  - **precisão simples em 32 bits (expoente: 8 bits, mantissa e sinal: 24 bits)**
  - **precisão dupla em 64 bits (expoente: 11 bits, mantissa e sinal: 53 bits)**

4. Modos de endereçamento

- **Acessos à memória devem ser alinhados**
  - **dados de 32 bits devem estar em endereços múltiplos de 4**
  - **dados de 16 bits devem estar em endereços múltiplos de 2**

- **Modo registrador**
  - **para instruções aritméticas e lógicas: dado está em registrador**
  - **para instruções de desvio incondicional: endereço está em registrador**

- **Modo base e deslocamento**
  - **para instruções load e store**
  - **base é registrador inteiro de 32 bits**
  - **deslocamento de 16 bits contido na própria instrução**

- **Modo relativo ao PC**
  - **para instruções de branch condicional**
  - **endereço é a soma do PC com deslocamento contido na instrução**
  - **deslocamento é dado em palavras e precisa ser multiplicado por 4**

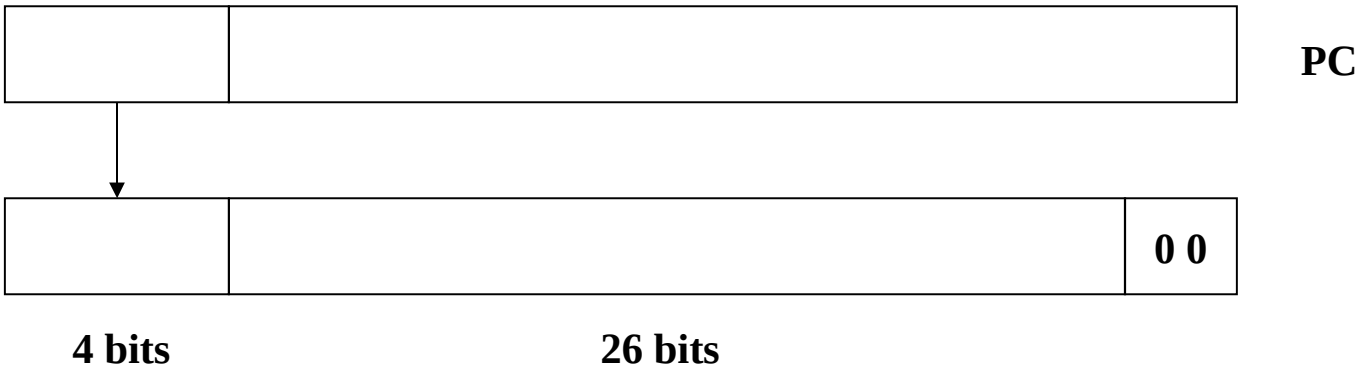
Modos de endereçamento

- **Modo imediato**
  - **para instruções aritméticas e lógicas**
  - **dado imediato de 16 bits contido na própria instrução**
  - **dado é estendido para 32 bits**
    - **extensão com sinal nas instruções aritméticas**
    - **extensão sem sinal nas instruções lógicas**

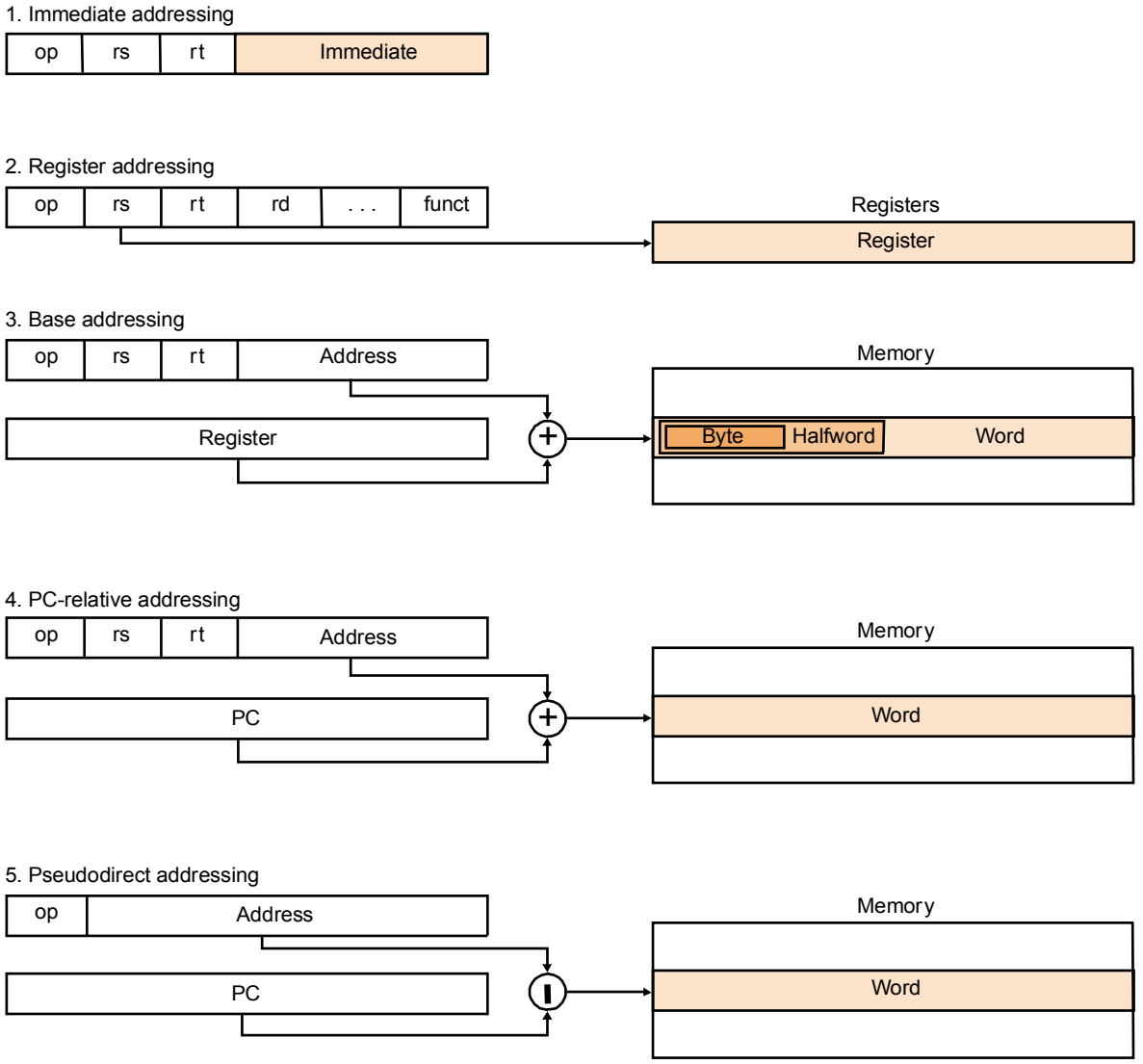
- **Para que se possa especificar constantes de 32 bits**
  - **instrução lui (load upper immediate)**
  - **carrega 16 bits imediatos na parte superior do registrador**
  - **parte inferior do registrador é zerada**
  - **instrução seguinte precisa fazer soma imediata do registrador com 16 bits da parte inferior**

Modos de endereçamento

- **Modo absoluto**
  - **para instruções de desvio incondicional**
  - **instrução tem campo com endereço de palavra com 26 bits**
  - **endereço de byte obtido com dois bits menos significativos iguais a 0**
  - **4 bits mais significativos obtidos do PC**
  - **só permite desvios dentro de uma área de 256 Mbytes**



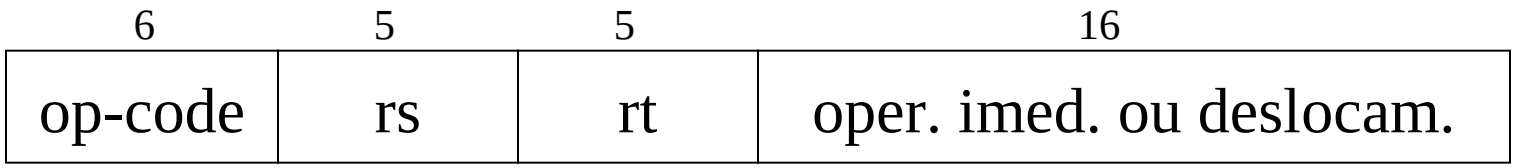
Modos de endereçamento do MIPS



5. Formatos das instruções

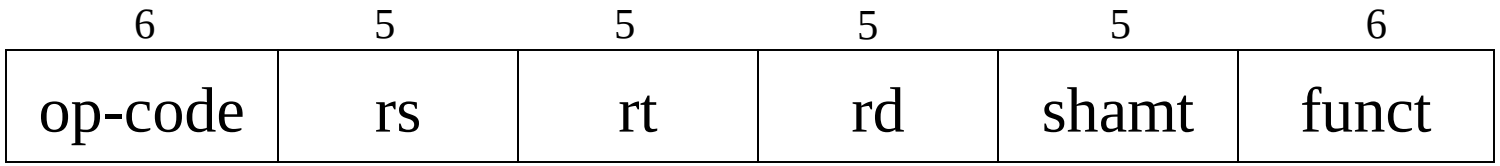
- **Todas as instruções têm 32 bits**
  - **todas têm op-code de 6 bits**
  - **modo de endereçamento é codificado juntamente com o op-code**

- **Instruções de tipo I**
  - **loads, stores**
  - **operações aritméticas e lógicas com operando imediato**
  - **desvios condicionais (“branches”)**
  - **desvios incondicionais para endereço em registrador**

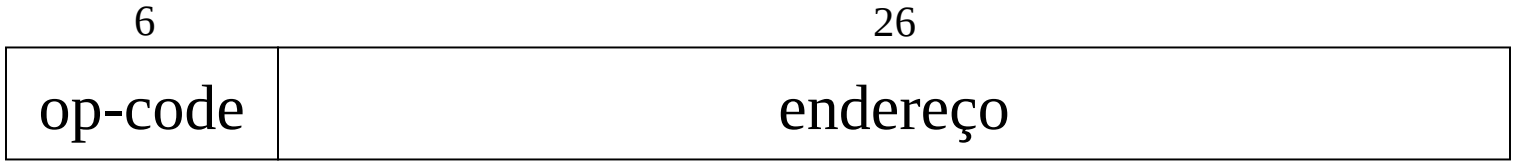


Formatos das instruções

- **Instruções de tipo R**
  - **instruções aritméticas e lógicas**
  - **instruções de movimentação entre registradores**
  - **"shamt" (shift amount) é usado em instruções de deslocamento**
  - **"funct" é a operação a ser feita pela ALU**



- **Instruções de tipo J**
  - **desvios com endereçamento absoluto**
  - **chamada de sub-rotina**

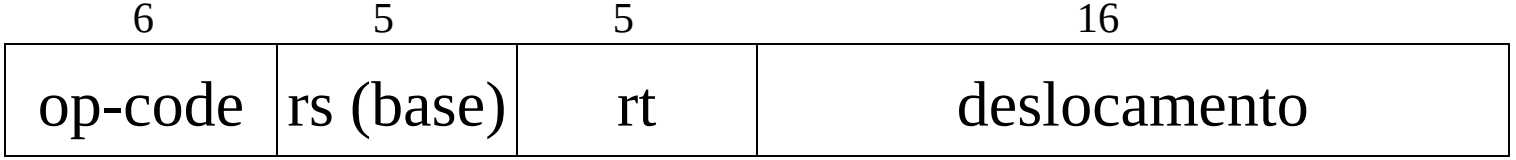


Instruções de máquina do MIPS

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

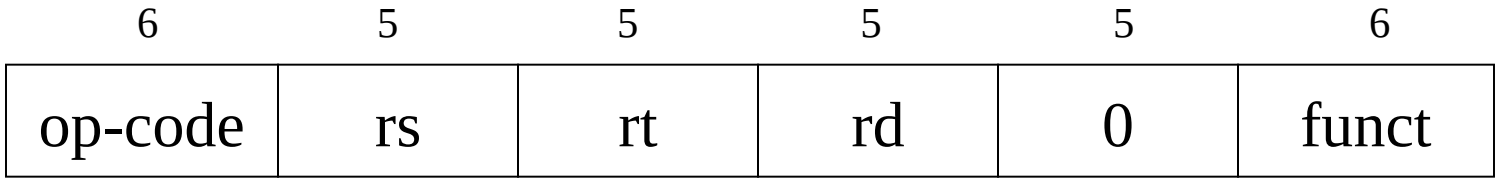
6. Tipos de instruções

- **Instruções load / store**
  - **são sempre tipo I**
  - **qualquer registrador de propósitos gerais pode ser carregado ou armazenado da / na memória**
  - **pode-se carregar ou armazenar bytes, meias palavras, palavras**
  - **endereçamento sempre por base e deslocamento**
- **lb, lh, lw – load byte, halfword, word**
  - **sinal é estendido em lb e lh**
- **lbu, lhu – load byte, halfword sem extensão de sinal**
- **sb, sh, sw – store byte, halfword, word**

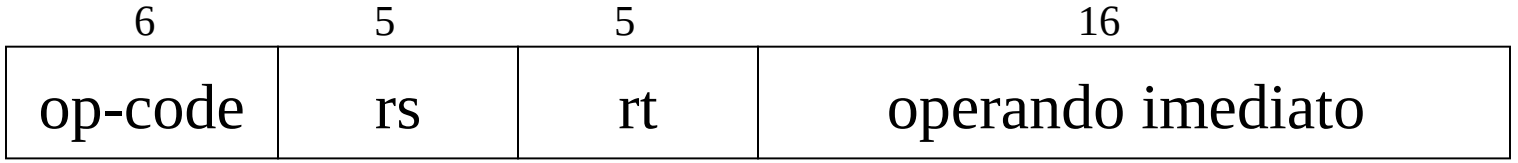


Instruções aritméticas e lógicas

- **Operação entre 2 registradores, resultado num terceiro registrador**
  - **tipo R**
- **add, sub, and, or, nor, xor**
- **Comparação slt – compara dois registradores e coloca valor 1 ou 0 em registrador destino**



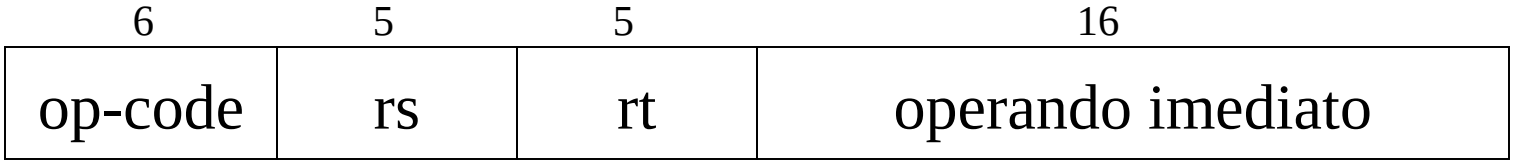
- **Existem versões com operando imediato, de tipo I**
  - **addi, andi, ori, xori, slti**



Instruções aritméticas e lógicas

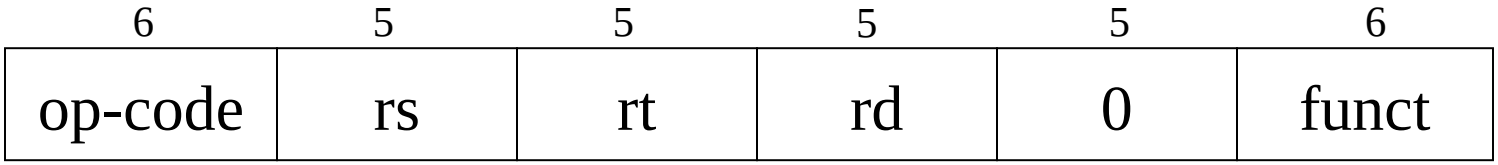
- **$0 usado para sintetizar operações populares**
- **Carga de constante = soma imediata onde $0 é um dos operandos**

  addi $S5, $0, 10



- **Mover de registrador para registrador = soma com $0**

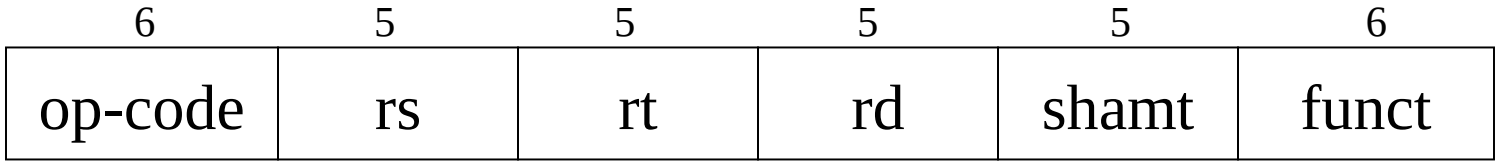
  add $S6, $S2, $0



Instruções aritméticas e lógicas

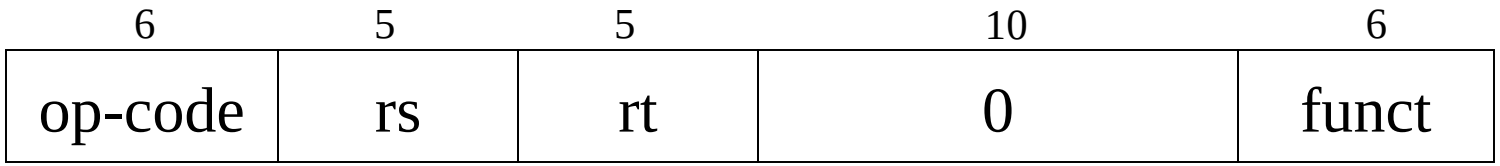
- **Instruções de deslocamento variável**
  - **tipo R**
  - **sllv, srlv – shift lógico (entra 0 na extremidade)**
  - **srav – shift aritmético (duplica sinal)**
  - **desloca registrador rt pela distância especificada no registrador rs e coloca resultado no registrador rd**

- **Instruções de deslocamento constante**
  - **tipo R**
  - **sll, sra, srl**
  - **desloca registrador rt pela distância especificada no campo “shamt” e coloca resultado no registrador rd**



Instruções aritméticas e lógicas

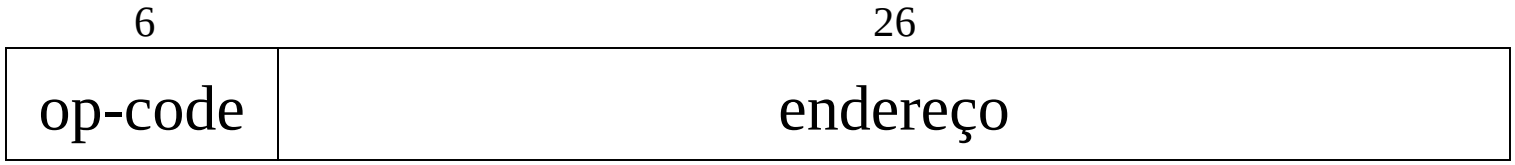
- **Instrução de multiplicação: mul**
  - **multiplica registradores rs e rt**
  - **resultado colocado em hi (32 msb) e lo (32 lsb)**
- **Instrução de divisão: div**
  - **divide registrador rs pelo registrador rt**
  - **quociente colocado em lo**
  - **resto colocado em hi**



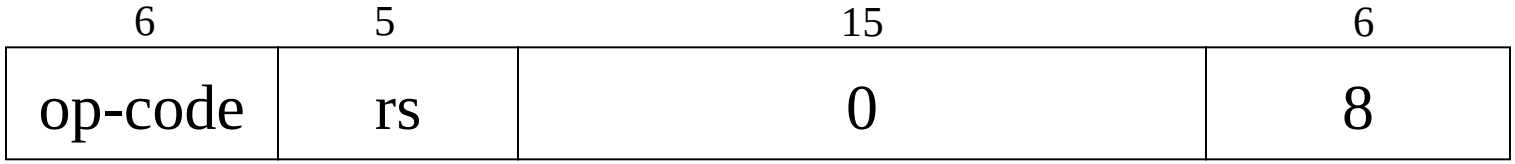
- **Instruções de movimentação permitem transferir dados entre hi e lo e os demais registradores**
  - **mfhi Rd, mflo Rd**
  - **mthi Rs, mtlo Rs**

Instruções de desvio incondicional

- **Instrução j**
  - **tipo J**
  - **endereço destino = concatenação dos 4 msb do PC com endereço imediato de 26 bits.**

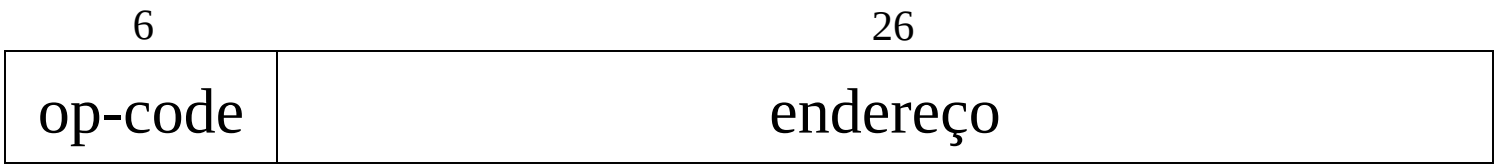


- **Instrução jr**
  - **tipo I: endereço destino contido em registrador**
  - **também serve para retornar de sub-rotina**

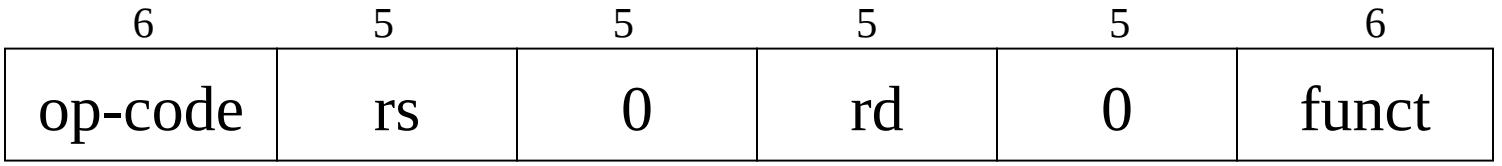


Instruções de desvio incondicional

- **Instrução jal (jump and link)**
  - **tipo J**
  - **desvio para sub-rotina, endereço especificado na instrução**
  - **endereço de retorno salvo em $31**



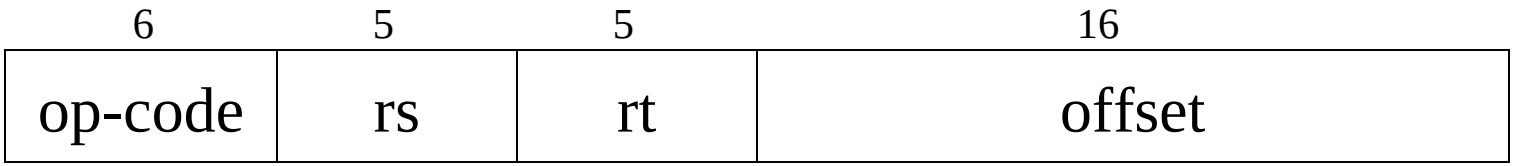
- **Instrução jalr (jump and link register)**
  - **tipo R**
  - **desvio para sub-rotina, endereço especificado em rs**
  - **endereço de retorno salvo em rd**



Instruções de desvio condicional

- **São sempre tipo I**
- **Endereço destino = soma do PC com offset imediato de 16 bits**
- **Instruções que testam um único registrador**
  - **bgez, bgtz, blez, bltz – desvia se registrador é ≥, >, ≤, < zero**
  - **bgezal, bltzal – como bgez e bltz, mas salva endereço de retorno em $31**

- **Instruções que comparam dois registradores**
  - **beq, bne – desvia se registradores são iguais (ou diferentes)**



Overflow em instruções aritméticas e lógicas

- **Instruções “unsigned” não geram overflow**
  - **addu, addiu, divu, multu, subu, sltu, sltiu**

- **Instruções “não-unsigned” geram overflow**
  - **endereço da instrução que causou overflow é colocado no registrador EPC = registrador 0 de um co-processador**
  - **instrução “mfc0” copia valor do EPC para outro registrador qualquer**
  - **instrução “jr” (jump para endereço especificado por registrador) é usada para desviar para rotina de atendimento**

Instruções de ponto flutuante

- **Mover operandos de precisão simples ou dupla entre registradores**
  - **mov.d, mov.s**
- **Soma, subtração, negação, multiplicação, divisão em precisão simples e dupla**
  - **add.s, sub.s, neg.s, mul.s, div.s**
  - **add.d, sub.d, neg.d, mul.d, div.d**
- **Comparações em precisão simples e dupla**
  - **c.eq.s, c.le.s, c.lt.s,**
  - **c.eq.d, c.le.d, c.lt.d,**
- **Conversão para ponto flutuante de precisão simples (ou dupla)**
  - **cvt.s.d, cvt.s.w – converter FP double (ou inteiro) para FP single**
  - **cvt.d.s, cvt.d.w – converter FP single (ou inteiro) para FP double**
- **Conversão de ponto flutuante para inteiro**
  - **cvt.w.s, cvt.w.d – converter FP double (ou single) para inteiro**

Instruções do MIPS

MIPS assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	add	`add $s1, $s2, $s3`	`$s1 = $s2 + $s3`	Three operands; data in registers
	subtract	`sub $s1, $s2, $s3`	`$s1 = $s2 - $s3`	Three operands; data in registers
	add immediate	`addi $s1, $s2, 100`	`$s1 = $s2 + 100`	Used to add constants
Data transfer	load word	`lw $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100]	Word from memory to register
	store word	`sw $s1, 100($s2)`	Memory[`$s2` + 100] = `$s1`	Word from register to memory
	load byte	`lb $s1, 100($s2)`	`$s1` = Memory[`$s2` + 100]	Byte from memory to register
	store byte	`sb $s1, 100($s2)`	Memory[`$s2` + 100] = `$s1`	Byte from register to memory
	load upper immediate	`lui $s1, 100`	$s1 = 100 * 2^{16}	Loads constant in upper 16 bits
Conditional branch	branch on equal	`beq $s1, $s2, 25`	if (`$s1 == $s2`) go to PC + 4 + 100	Equal test; PC-relative branch
	branch on not equal	`bne $s1, $s2, 25`	if (`$s1 != $s2`) go to PC + 4 + 100	Not equal test; PC-relative
	set on less than	`slt $s1, $s2, $s3`	if (`$s2 < $s3`) `$s1` = 1; else `$s1` = 0	Compare less than; for beq, bne
	set less than immediate	`slti $s1, $s2, 100`	if (`$s2 < 100`) `$s1` = 1; else `$s1` = 0	Compare less than constant
Uncondi-tional jump	jump	`j 2500`	go to 10000	Jump to target address
	jump register	`jr $ra`	go to `$ra`	For switch, procedure return
	jump and link	`jal 2500`	`$ra` = PC + 4; go to 10000	For procedure call

FIM