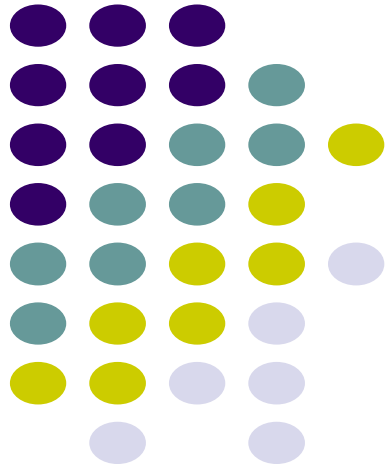
UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO
DEPARTAMENTO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO

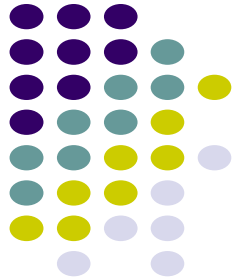
Arquitetura e Organização de Computadores

Aritmética Computacional



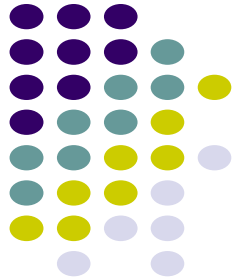
Prof. Sílvio Fernandes

Números com e sem sinal



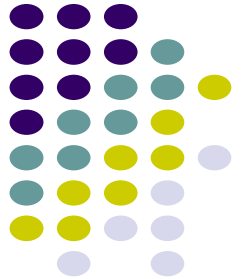
- O HW pode ser projetado para realizar as operações aritméticas nos padrões de bits
- Se o resultado correto de tais operações não puder ser representado por esses bits de HW mais à direita, diz-se que houve **overflow**
  - O Sistema Operacional ou programa determina o que fazer

Números com e sem sinal



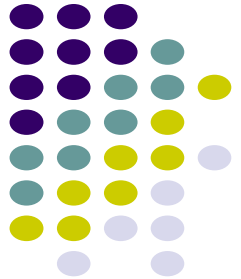
- Como representar números negativos?
  - Representação Sinal-Magnitude
  - Representação em Complemento de Dois

Números com e sem sinal



- Representação Sinal-Magnitude
  - Em uma palavra de n bits
    - O bit mais à esquerda representa o sinal do número inteiro
    - Os n-1 bits mais à direita representam a magnitude do número inteiro
  - Exemplo:
    - +18 = 00010010
    - -18 = 10010010

Números com e sem sinal



- Representação Sinal-Magnitude
  - Há duas representações para o zero
    - +0 = 00000000
    - -0 = 10000000
  - É mais difícil testar se um valor é igual a zero do que no caso em que há apenas uma representação para o zero
  - Por isso, essa representação raramente é usada na implementação da parte inteira de uma ULA (Unidade Lógica e Aritmética)

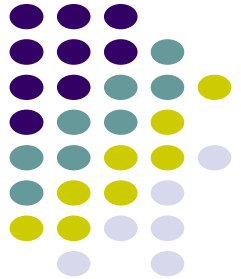
Números com e sem sinal

- Representação em Complemento de Dois

$$A = -2^{n-1} a_{n-1} + \sum_{i=0}^{n-2} 2^i a_i$$

  - Para números inteiros positivos, $a_{n-1} = 0$
  - O número 0 é tratado como um número inteiro positivo
  - **Zeros** iniciais significa **positivo**
  - **Uns** iniciais significa **negativo**

Números com e sem sinal



- Representação em Complemento de Dois
  - Usada para representar números na faixa $-2^n \leftrightarrow 2^n-1$
  - Usada quase universalmente para representar números inteiros dentro do µP

Números com e sem sinal

- Conversão complemento de 2 → decimal

-128	64	32	16	8	4	2	1
1	0	0	0	0	0	1	1

-128 +2 +1 = -125

- Conversão decimal→ complemento de 2

-120 =

-128	64	32	16	8	4	2	1
1	0	0	0	1	0	0	0

-128 +8

Números com e sem sinal

- Exemplos:

+18 = 00010010 (c-2)

Complemento booleano = 11101101

+1

11101110 = -18

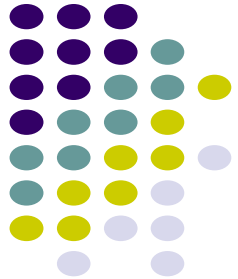
-18 = 11101110 (c-2)

Complemento booleano = 00010001

+1

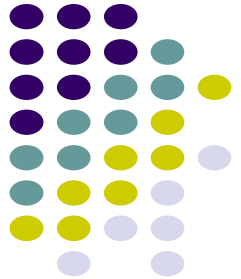
00010010 = +18

Números com e sem sinal



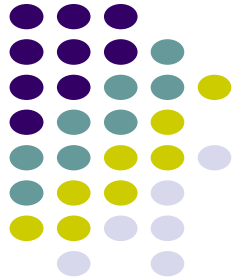
- Representação em Complemento de Dois
  - Operações com números em complemento de dois também pode ocasionar *overflow*
  - Ocorre quando o bit mais à esquerda não é igual ao número infinito de dígitos à esquerda (o bit de sinal está incorreto)
    - 0 à esquerda quando o número é negativo
    - 1 à esquerda quando o número é positivo

Números com e sem sinal



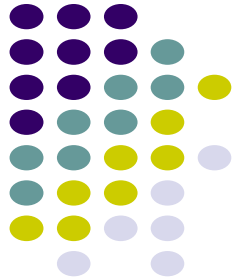
- MIPS oferece dois tipos de load de bytes
  - Load byte (lb)
    - trata o byte como um número **com** sinal, e, portanto, estende o sinal para preencher os 24 bits mais à esquerda do registrador
  - Load byte unsigned (lbu)
    - Trabalho com inteiros **sem** sinal
- Load half (lh) trata a halfword como um número **com** sinal (estende sinal)
- Load half-word unsigned trabalha com inteiros **sem** sinal

Números com e sem sinal



- MIPS oferece duas versões de comparação
  - Set on less than (slt)
  - Set on less than immediate (slti)
    - Ambos trabalham com inteiros **com** sinal
  - Set on less than unsigned (sltu)
  - Set on less than immediate unsigned (sltiu)
    - Trabalham com inteiros **sem** sinal

Números com e sem sinal



- Exemplo:
  - Suponha que os registradores $s0 tenha o número binário

  1111 1111 1111 1111 1111 1111 1111 1111 $_{bin}$

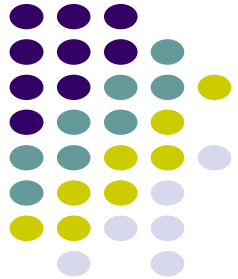
  e que o registrador $s1 tenha o número binário

  0000 0000 0000 0000 0000 0000 0000 0001 $_{bin}$

  Quais são os valores dos registradores $t0 e $t1 após essas duas instruções?

```
slt   $t0, $s0, $s1   # comparação com sinal
sltu  $t1, $s0, $s1   # comparação sem sinal
```

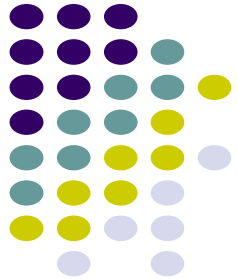
Números com e sem sinal



- Exemplo:
  - O valor do registrador $s0 representa
    - -1 (com sinal)
    - 4.294.967.295 (sem sinal)
  - O valor do registrador $s1 representa o valor 1 nos dois casos

  - $t0 recebe o valor 1, pois $-1_{dec} < 1_{dec}$
  - $t1 recebe o valor 0, pois $4.294.967.295_{dec} > 1_{dec}$

Números com e sem sinal



- Exemplo:
  - Use um atalho para reduzir uma verificação de índice fora dos limites: desvie para *IndiceForaDosLimites* se $a1 >= $t2 ou se $a1 for negativo
    - O código de verificação só utiliza sltu
      - sltu $t0, $a1, $t2 #$t0 = 0 se k >= tamanho ou k <0
      - beq $t0, $zero, IndiceForaDosLimites #se fora dos lim. = erro

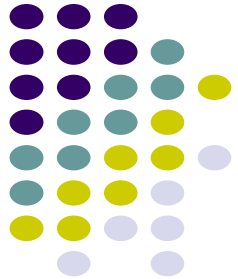
Adição e Subtração

- Quando um overflow ocorre na adição?
  - Quando somamos operandos de sinais diferentes, **não** poderá ocorrer overflow
  - Quando somamos operandos com sinais iguais, **pode** ocorrer overflow
- Como detectar overflow?
  - O overflow ocorre quando se somam dois número positivos, e a soma é negativa, ou vice-versa

Condições de Overflow para adição e subtração

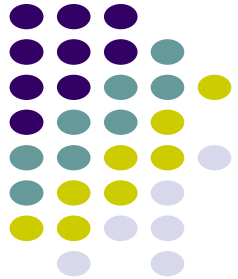
Operation	Operand A	Operand B	Result indicating overflow
$A + B$	≥ 0	≥ 0	< 0
$A + B$	< 0	< 0	≥ 0
$A - B$	≥ 0	< 0	< 0
$A - B$	< 0	≥ 0	≥ 0

Adição e Subtração



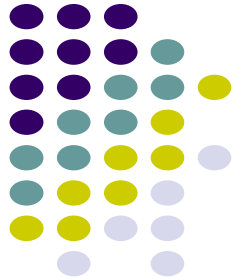
- Os inteiros **sem** sinal normalmente são usados para endereços de memória onde overflows são ignorados
- A solução MIPS é ter dois tipos de instruções aritméticas para reconhecer as duas escolhas
  - Adição (add), adição imediata (addi) e subtração (sub) causam exceções no overflow
  - Adição sem sinal (addu), adição imediata sem sinal (addiu) e subtração sem sinal (subu) **não** causam exceções no overflow

Adição e Subtração



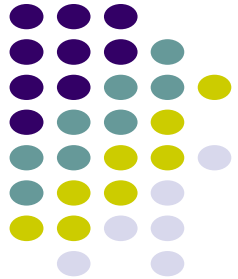
- O MIPS detecta o overflow com uma **exceção**
  - É basicamente uma chamada de procedimento não-planejada
  - O endereço da instrução que gerou o overflow é salvo em um registrador, e o controle é desviado para um endereço predefinido, para invocar a rotina apropriada para essa exceção
  - O MIPS inclui um registrador, *contador de programa de exceção* (EPC) para conter o endereço da instrução que causou a exceção
  - A instrução *move from system control* (mfc0) é usada para copiar o EPC para um registrador de uso geral

Multiplicação



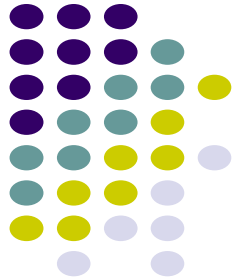
- Observações:
  - número de bits do produto é muito maior que do multiplicando e multiplicador
  - se ignorarmos os bits de sinal, e o tamanho do multiplicando for **n** bits e multiplicador for **m** bits, o produto terá **n+m** bits

Multiplicação



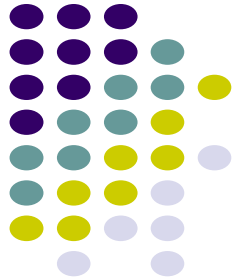
- Multiplicando número negativos
  - Solução 1:
    - Converta para positivo, se for preciso.
    - Multiplique como antes.
    - Se sinais diferentes, negue a resposta.
  - Solução 2:
    - Algoritmo de Booth.

Multiplicação



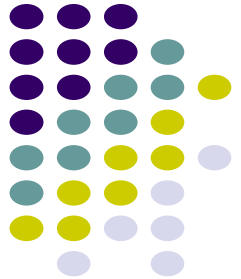
- O MIPS oferece um par separado de registradores de 32 bits, para conter o produto de 64 bits, chamados ***Hi*** e ***Lo***
- Instrução multiply (**mult**): com sinal
- Instrução multiply unsigned (**multu**): sem sinal
- Para apanhar o produto de 32 bits inteiro, o programador usa *move from lo* (**mflo**)
- O montador MIPS gera uma pseudo-instrução para multiplicar, que especifica 3 registradores de uso geral, gerando instruções mflo e mfhi para colocar o produto nos registradores

Divisão



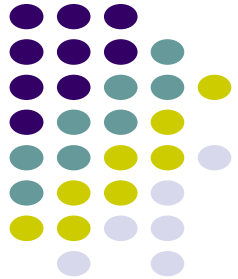
- O MIPS utiliza os registradores Hi e Lo de 32 bits, tanto para multiplicação quanto para divisão
  - **Hi** contém o resto
  - **Lo** contém o quociente
- O MIPS possui 2 instruções
  - Divide (**div**)
  - Divide unsigned (**divu**)
- O montador MIPS permite que as instruções de divisão especifiquem 3 registradores, gerando as instruções mflo ou mfhi para colocar o resultado desejado em um registrador de uso geral

Divisão



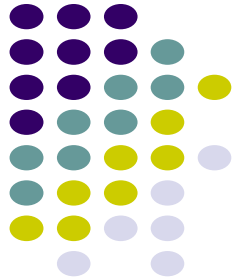
- Instruções de divisão no MIPS ignoram o overflow, de modo que o software precisa determinar se o quociente é muito grande
- O software também precisa verificar se o divisor é igual a zero

Instruções de ponto flutuante no MIPS



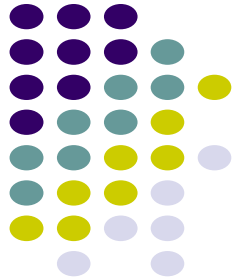
- O MIPS admite precisão simples e dupla do padrão IEEE 754
  - Adição simples em ponto flutuante (add.s) e adição dupla (add.d)
  - Subtração simples em ponto flutuante (sub.s) e subtração dupla (sub.d)
  - Multiplicação simples em ponto flutuante (mul.s) e multiplicação dupla (mul.d)
  - Divisão simples em ponto flutuante (div.s) e divisão dupla (div.d)

Instruções de ponto flutuante no MIPS



- O MIPS admite precisão simples e dupla do padrão IEEE 754
  - Comparação simples em ponto flutuante (c.x.s) e comparação dupla (c.x.d), onde x pode ser igual (eq), diferente (neq), menor que (lt), menor ou igual (le), maior que (gt) ou maior ou igual (ge)
  - Desvio, verdadeiro em ponto flutuante (belt) e desvio falso (bclf)
- Os projetistas do MIPS decidiram acrescentar registradores de ponto flutuante separados ($f0, $f1, $f2, ...)
  - Load e Store para ponto flutuante (lwc1 e swc1)

Instruções de ponto flutuante no MIPS



- Código MIPS para ler 2 números de ponto flutuante, somá-los e armazenar a soma na memória
  - lwc1 $f4, x($sp) #lê no. P.F 32 bits em F4
  - lwc1 $f6, y($sp) #lê no. P.F 32 bits em F6
  - add.s $f2, $f4, $f6 #F2 = F4+F6 precisão simples
  - swc1 $f2, z($sp) #armazena no. P.F. 32 bits de F2

Instru

Categoria	Instrução	Exemplo	Significado	Comentário
Aritmética	Add	add $s1, $s2, $s3	$s1 = $s2+$s3	3 operandos; overflow detectado
	Subtract	sub $s1, $s2, $s3	$s1 = $s2-$s3	3 operandos; overflow detectado
	Add Immediate	dddi $s1, $s2, 100	$s1 = $s2+100	+constante; overflow detectado
	Add unsigned	addu $s1, $s2, $s3	$s1 = $s2 + $s3	3 operandos; overflow não detectado
	Subtract unsigned	subu $s1, $s2, $s3	$s1 = $s2 - $s3	3 operandos; overflow não detectado
	Add immediate unsigned	addiu $s1, $s2, 10	$s1 = $s2 + 10	+contante; overflow não detectado
	move from coprocessor register	mfc0 $s1, $epc	$s1 = $epc	Usado para copiar PC de exceção mais outros registradores especiais
	multiply	mult $s2, $s3	Hi, Lo = $s2*$s3	Produto com sinal, 64 bits, em Hi e Lo
	Multiply unsigned	multu $s2, $s3	Hi, Lo = $s2*$s3	Produto sem sinal, 64 bits, em Hi e Lo
	Divide	div $s2, $s3	Lo = $s2/$s3 Hi = $s2 mod $s3	Lo = quociente, Hi = resto
	Divide unsigned	divu $s2, $s3	Lo = $s2/$s3 Hi = $s2 mod $s3	Quociente e resto sem sinal
	move from Hi	mfhi $s1	$s1 = Hi	Usado para obter cópia de Hi
	move from Lo	mflo $s1	$s1 = Lo	Usado para objer cópia de Lo

Instru

Categoria	Instrução	Exemplo	Significado	Comentário
Lógica	And	and $s1, $s2, $s3	$s1 = $s2 & $s3	3 operandos em reg; AND bit a bit
	Or	or $s1, $s2, $s3	$s1 = $s2 | $s3	3 operandos em reg; OR bit a bit
	Nor	nor $s1, $s2, $s3	$s1 = ~($s2 | $s3)	3 operandos em reg; NOR bit a bit
	And Immediate	andi $s1, $s2, 100	$s1 = $s2 & 100	AND bit a bit entre reg. e constante
	Or Immediate	ori $s1, $s2, 100	$s1 = $s2 | 100	OR bit a bit entre reg. e constante
	Shift left logical	sll $s1, $s2, 10	$s1 = $s2 << 10	Deslocamento à esquerda por const.
	Shift rigth logical	srl $s1, $s2, 10	$s1 = $s2 >> 10	Deslocamento à direita por const.
Transferência de dados	Load word	lw $s1, 100($s2)	$s1 = Mem[$s2+100]	Dados da mem. para o registrador
	Store Word	sw $s1, 100($s2)	Mem[$s2+100] = $s1	Dados do registrador para a mem.
	load half	lh $s1, 100($s2)	$s1 = Mem[$s2+100]	Halfword da memória para registrador
	store half	sh $s1, 100($s2)	Mem[$s2+100] = $s1	Halfword de um reg. para memória
	load byte	lb $s1, 100($s2)	$s1 = Mem[$s2+100]	Byte da memória para registrador
	store byte	sb $s1, 100($s2)	Mem[$s2+100] = $s1	Byte de um registrador para memória
	load upper immed.	lui $s1, 100	$s1 = 100 * 2^{16}	Carrega constante nos 16 bits mais altos

Instruções MIPS reveladas até aqui

Categoria	Instrução	Exemplo	Significado	Comentário
Desvio Condicional	Branch on equal	beq $s1, $s2, L	If($s1 == $s2) go to L	Testa igualdade e desvia
	Branch on not equal	bne $s1, $s2, L	If($s1 != $s2) go to L	Testa desigualdade e desvia
	Set on less than	stl $s1, $s2, $s3	If($s2 < $s3) $s1 = 1; Else $s1 = 0	Compara menor que
	Set less than immediate	stli $s1, $s2, 100	If($s2 < 100) $s1 = 1; Else $s1 = 0	Compara menor que constante
Desvio incondicional	Jump	j L	go to L	Desvia para endereço de destino
	Jump register	jr $ra	go to $ra	Para switch e retorno de procedimento
	Jump and link	jal L	$ra = PC + 4 go to L	Para chamada de procedimento

Instruções de ponto flutuante no MIPS

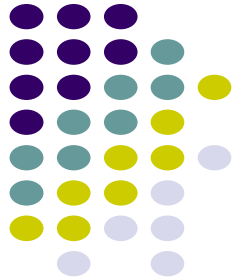
MIPS floating-point operands

Name	Example	Comments
32 floating-point registers	`$f0, $f1, $f2, . . . , $f31`	MIPS floating-point registers are used in pairs for double precision numbers.
2^{30} memory words	Memory[0], Memory[4], . . . , Memory[4294967292]	Accessed only by data transfer instructions. MIPS uses byte addresses, so sequential word addresses differ by 4. Memory holds data structures, such as arrays, and spilled registers, such as those saved on procedure calls.

MIPS floating-point assembly language

Category	Instruction	Example	Meaning	Comments
Arithmetic	FP add single	`add.s $f2,$f4,$f6`	`$f2 = $f4 + $f6`	FP add (single precision)
	FP subtract single	`sub.s $f2,$f4,$f6`	`$f2 = $f4 - $f6`	FP sub (single precision)
	FP multiply single	`mul.s $f2,$f4,$f6`	`$f2 = $f4 x $f6`	FP multiply (single precision)
	FP divide single	`div.s $f2,$f4,$f6`	`$f2 = $f4 / $f6`	FP divide (single precision)
	FP add double	`add.d $f2,$f4,$f6`	`$f2 = $f4 + $f6`	FP add (double precision)
	FP subtract double	`sub.d $f2,$f4,$f6`	`$f2 = $f4 - $f6`	FP sub (double precision)
	FP multiply double	`mul.d $f2,$f4,$f6`	`$f2 = $f4 x $f6`	FP multiply (double precision)
	FP divide double	`div.d $f2,$f4,$f6`	`$f2 = $f4 / $f6`	FP divide (double precision)
Data transfer	load word copr. 1	`lwc1 $f1,100($s2)`	`$f1 = Memory[$s2 + 100]`	32-bit data to FP register
	store word copr. 1	`swc1 $f1,100($s2)`	`Memory[$s2 + 100] = $f1`	32-bit data to memory
Condi-tional branch	branch on FP true	`bc1t 25`	if (cond == 1) go to PC + 4 + 100	PC-relative branch if FP cond.
	branch on FP false	`bc1f 25`	if (cond == 0) go to PC + 4 + 100	PC-relative branch if not cond.
	FP compare single (eq,ne,lt,le,gt,ge)	`c.lt.s $f2,$f4`	if (`$f2 < $f4`) cond = 1; else cond = 0	FP compare less than single precision
	FP compare double (eq,ne,lt,le,gt,ge)	`c.lt.d $f2,$f4`	if (`$f2 < $f4`) cond = 1; else cond = 0	FP compare less than double precision

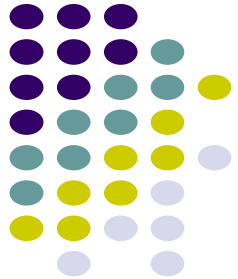
Pseudo-instruções



- Alguns montadores também implementam pseudo-instruções, que são instruções fornecidas por um montador mas não implementadas no hardware

PseudoMIPS	Nome	Formato	Exemplo	Significado
Move	move	R	move $t1, $t0	add $t1, $zero, $t0
Multiply	mult	R	mult $t1, $t0	mult $t1, $t1, $t0
Load immediate	li	I	li $t1, 100	addiu $t1, $0, 100
Branch less than	blt	I	blt $t1, $t0, label	slt $at, $t1, $t0 bne $at, $0, label
Branch less than or equal	ble	I	ble $t1, 34, label	addi $at, $0, 35 slt $at, $t1, $at bne $at, $0, label
Branch greater than	bgt	I	bgt $t1, $t0, label	slt $at, $t0, $t1 bne $at, $0, label
Branch greater than or equal	bge	I	bge $t1, $t0, label	slt $at, $t1, $t0 bne $at, $0, label

Verifique você mesmo



- Algumas linguagens de programação permitem a aritmética de inteiros em complemento de 2 com variáveis declaradas com um byte e meio. Que instruções do MIPS seriam usadas?
  1. Leitura com lbu, lhu; aritmética com add, sub, mult, div; depois armazenamento usando sb, sh.
  2. Leitura com lb, lh; aritmética com add, sub, mult, div; depois armazenamento usando sb, sh.
  3. Leitura com lb, lh; aritmética com add, sub, mult, div,usando and para mascarar o resultado com 8 ou 16 bits após cada operação; depois armazenamento usando sb, sh.

Verifique você mesmo

- Resposta: afirmativa 2

Referências

- PATTERSON, D. A. ; HENNESSY, J.L. Organização e projeto de computadores – a interface hardware software. 3. ed. Editora Campus, 2005.
- STALLINGS, W. Arquitetura e organização de computadores: projeto para o desempenho. 8. ed. Prentice Hall, 2009.