

### 3. Aritmética de Números Inteiros no MIPS<sup>1</sup>

#### Objetivos

Após completar as atividades das etapas do laboratório 3, você deverá entender como funcionam as operações aritméticas com números inteiros no MIPS.

#### Aritmética Inteira

Adição e subtração são realizadas em números de 32 bits armazenados em registradores de propósito geral (cada um com 32 bits, numerados de \$0 a \$31). O resultado da execução da instrução é sempre um número de 32 bits. Dois tipos de instruções estão incluídas no conjunto de instruções MIPS para fazer a adição e subtração:

- Instruções para aritmética “com sinal”: os números de 32 bits são representados sob a forma de valores em complemento de 2. Neste caso, a execução da instrução de adição ou subtração **pode gerar um sinal (de interrupção) de overflow**, isto é, o resultado não pode ser representado em 32 bits.
- Instruções para aritmética “sem sinal”: os números de 32 bits são representados sob a forma de valores padrão em binário. Neste caso, a execução deste tipo de instrução **nunca gerará um sinal (interrupção) de overflow** mesmo que o resultado da operação “supostamente gerasse” um sinal (de interrupção) de overflow.

A multiplicação pode gerar resultados maiores que a faixa de valores representáveis em 32 bits  $[-2^{31}$  a  $2^{31}-1]$ . A divisão inteira retorna dois valores de 32 bits, resto e quociente, que exigem 32 bits, cada, para sua representação. A arquitetura MIPS fornece 2 registradores especiais de 32 bits que são destino para as operações de multiplicação e divisão. Estes registradores extras, chamados **hi** e **lo**, são destinos implícitos nestas instruções **mult** e **div** do conjunto de instruções do MIPS. O registrador **hi** armazena os bits de mais alta ordem do resultado da multiplicação (bits 63 a 32) e, no caso da divisão, o resto da divisão de dois inteiros. O registrador **lo** armazena os bits de mais baixa ordem (bits 31 a 0) do resultado da multiplicação e, no caso da divisão, o quociente da divisão de dois inteiros. Duas instruções especiais são oferecidas pelo conjunto de instruções MIPS para movimentar os dados dos registradores **hi** e **lo** para registradores de propósito geral.

Instrução MIPS	Semântica
mfhi Rdest	hi → Rdest
mflo Rdest	lo → Rdest
mthi Rsrc	hi ← Rsrc
mtlo Rsrc	lo ← Rsrc

Dois tipos de instruções estão incluídas no conjunto de instruções MIPS para fazer a multiplicação e divisão:

- Instruções para aritmética “com sinal” (*signed numbers*): os números de 32 bits são representados sob a forma de valores em complemento de 2. A execução da instrução de multiplicação **nunca gera overflow**, mas a divisão **pode gerar overflow**. Note que a CPU não gera um sinal de overflow nesse último caso. O overflow deve ser resolvido por software (lembre-se de que o programador deve evitar, por exemplo, a divisão por zero).

<sup>1</sup> Fontes: Capítulo 3 do Livro Texto e apostilas do curso do Prof. Virgil Bistriceanu, 1996 [www.cs.iit.edu/~virgil/cs470/Labs/].

- **Instruções para aritmética “sem sinal”** (*unsigned numbers*): os números de 32 bits são representados sob a forma de valores padrão em **binário**. O bit 31 é o bit mais significativo do número armazenado. Neste caso, a execução deste tipo de instrução **nunca gerará um sinal (interrupção) de overflow** mesmo que o resultado da operação “supostamente gerasse” um erro de *overflow*.

## Atividade 1

O foco aqui está nas questões relacionadas à diferença entre as operações com sinal e sem sinal, além dos erros de *overflow*. Um *overflow* ocorrerá quando o resultado de uma operação aritmética não puder ser representado na quantidade de bits disponíveis para armazená-lo. No caso MIPS, o resultado com sinal deve poder ser representado em 32 bits.

## Adição e subtração

Um tipo inteiro “sem sinal” identifica um dado que sempre terá um valor inteiro e positivo. Com  $n$  bits para representação, o menor número sem sinal representável com este tipo é 0 e o maior,  $2^n - 1$ . Em programas C/C++, as variáveis inteiras que nunca receberão valores negativos poderiam ser declaradas como `unsigned int` (embora você possa ter visto muitos programas onde variáveis armazenam tipicamente valores positivos terem sido declaradas como `int`, i.e. inteiros com sinal...). Inteiros “com sinal” podem ser negativos ou positivos. Com  $n$  bits para representação e com os negativos representados em complemento de 2, o menor número sem sinal representável é  $-2^{n-1}$  e o maior,  $2^{n-1} - 1$ . No nível de programação C/C++ este tipo de diferença é normalmente ignorado, em especial porque os programadores não precisam utilizar valores próximos do início e do fim do intervalo de representação (por exemplo, contadores de interação normalmente armazenam valores muito menores do que  $2^{n-1} - 1$ ). Entretanto, é simples verificar que declarar uma variável “sempre positiva” como `int` (quando ela de fato deveria ter sido representada como `unsigned int`) reduz a faixa de valores representáveis à metade da escala. Por exemplo, considere o código C a seguir:

```
// Assuma inteiros em 32 bits
{
    unsigned int ex1; // ex1 entre [0..4294967295]
    int ex2; // ex1 entre [-2147483648..2147483647]
}
```

Se a variável `ex2` for sempre positiva, então a faixa de valores possíveis para esta variável ficará entre 0 e 2147483647, ou seja, a metade dos valores possíveis que um inteiro sem sinal (`unsigned int`) teria.

## Questão 1:

Um número negativo é representado como um padrão de bits no qual o bit mais significativo é 1 (considerando a representação em complemento de 2). O mesmo padrão de bits estaria associado à representação de um valor inteiro elevado se este padrão fosse tratado como a representação de um valor “sem sinal”. Preencha a tabela a seguir com os valores inteiros representados pelos padrões de 16 bits a seguir.

Padrão de bits	Inteiro com sinal	Inteiro sem sinal
0x0000		
0x7fff		
0x8000		
0xffff		

Note que a extensão da representação de 16 para 32 bits pode ser feita de maneira simples, copiando o bit mais significativo da representação com menos bits para todos os bits de mais alta ordem incluídos na representação estendida. Reescreva a tabela anterior, considerando que a representação dos padrões de bits anteriores estariam agora representadas em 32 bits, conforme o exemplo da primeira linha da tabela.

Padrão de bits	Inteiro com sinal	Inteiro sem sinal
0x00000000	0	0

No nível da linguagem de montagem (*assembly* MIPS) a diferença entre um número com sinal e sem sinal é mais sutil. Muitas instruções aritméticas com números com sinal podem gerar *overflow* e este *overflow* será sinalizado ao “controle da máquina” (o sistema operacional). As instruções que fazem aritmética com inteiros “sem sinal” por outro lado, ignorarão um *overflow* mesmo se a operação gerar um resultado não representável na quantidade de bits disponível para armazená-lo.

## Questão 2:

Em que caso uma operação poderia gerar overflow? Preencha a tabela a seguir com “S” ou “N” para indicar a ocorrência ou não de overflow nas operações de adição e subtração. Mostre um exemplo de operação com os operandos Oper<sub>1</sub> e Oper<sub>2</sub> para cada situação.

Oper <sub>1</sub>	Oper <sub>2</sub>	Oper <sub>1</sub> + Oper <sub>2</sub>	Oper <sub>1</sub> – Oper <sub>2</sub>	Exemplo
valor > 0	valor > 0			
valor > 0	valor < 0			
valor < 0	valor > 0			
valor < 0	valor < 0			
0	valor > 0			
0	valor < 0			
valor > 0	0			
valor < 0	0			

## Atividade 2

Escreva um programa em C, chamado *lab3.1.c*, no qual você:

- Declara 2 variáveis inteiras, *a* e *b*. O valor inicial de *a* e *b* é o maior valor possível para um inteiro (INT\_MAX quando você usa um *include limits.h* em seu programa C);
- Soma *a* e *b* e imprime o resultado da soma;
- Subtrai 0 de *b* e imprime o resultado da subtração.

Compile e execute o programa *lab3.1.c* e mostre a saída do programa.

## Questão 3:

Você verificou a ocorrência de overflow na execução do programa *lab3.1.c*? Explique o que aconteceu e qual o resultado a execução do programa.

## Atividade 3

Escreva um programa usando MIPS e o simulador MARS, denominado *lab3.2.asm*, que:

- Declare 2 variáveis inteiras, *a* e *b*. Estas variáveis serão usadas para armazenar 2 inteiros digitados pelo usuário, os quais serão armazenados em *\$t0* e *\$t1*;
- Some *\$t0* e *\$t1* usando uma instrução nativa para inteiros com sinal e insira o resultado em *\$t2*;
- Imprima o resultado da soma.

Rode o programa e preencha a tabela a seguir com seu “plano de testes”. Escolha valores que gerem overflow. Na coluna “Overflow” use “S/N” para indicar que houve (não houve) overflows. Na coluna “Comentários” descreva o motivo da ocorrência do overflow.

**OBS:** Escolha os números cuidadosamente para seus testes serem efetivos. A chamada de sistema que lê um inteiro do usuário irá truncar um valor muito grande que ultrapassaria o limite do registrador (32 bits). Tente colocar um número inteiro muito grande (ex 8.589.934.593) e observe o resultado da execução.

Oper_1	Oper_2	Resultado Impresso	Overflow	Justificativa
+2000000000	+2000000000		Y	
+	–			
–	+			
–	–			

## Atividade 4

Escreva um programa usando MIPS e o simulador MARS, denominado *lab3.3.asm* (muito similar ao anterior), que:

- Declare 2 variáveis inteiras, *a* e *b*. Estas variáveis serão usadas para armazenar 2 inteiros digitados pelo usuário, os quais serão armazenados em **\$t0** e **\$t1**
- Subtraia **\$t0** e **\$t1** usando uma instrução nativa para inteiros com sinal e insira o resultado em **\$t3**.
- Imprima o resultado da subtração.

Rode o programa e preencha a tabela a seguir com seu “plano de testes”. Sempre que possível, escolha valores que gerem overflow. Na coluna “Overflow” use “S/N” para indicar que houve (não houve) overflows. Na coluna “Comentários” descreva o motivo da ocorrência do overflow.

Oper_1	Oper_2	Resultado Impresso	Overflow	Comentários (justificativa)
+2000000000	+2000000000		Y	
+	–			
–	+			
–	–			

## Atividade 5

Agora, vamos testar o uso de instruções que desconsideram o overflow. O simulador MARS sempre imprime o conteúdo dos registradores de propósito geral como sendo um valor inteiro com sinal. O fato de que um valor válido “muito grande” sendo representado nos 32 bits e sendo impresso como um valor negativo poderia confundir um programador menos avisado. Os valores tratados como “*unsigned numbers*” podem usar o bit mais significativo como sendo um valor e não a representação do sinal. Entretanto, eles serão interpretados como negativos pelo serviço de impressão do sistema operacional.

Escreva agora um programa usando MIPS e o simulador MARS, denominado *lab3.4.asm* (também similar aos anteriores) para:

- Declarar 2 variáveis inteiras, *a* e *b*. Estas variáveis serão usadas para armazenar 2 inteiros digitados pelo usuário, os quais serão armazenados em **\$t0** e **\$t1**
- Somar **\$t0** e **\$t1** usando uma instrução nativa para inteiros sem sinal (*unsigned*) e armazenar o resultado em **\$t2**.
- Imprima o resultado da adição.
- Subtrair **\$t0** e **\$t1** usando uma instrução nativa para inteiros sem sinal (*unsigned*) e armazenar o resultado em **\$t3**.
- Imprima o resultado da subtração.

Rode o programa e preencha a tabela a seguir com seu “plano de testes”. Sempre que possível, escolha valores que gerem overflow. Na coluna “Overflow” use “S/N” para indicar que houve (não houve) overflows. Na coluna “Comentários” descreva o motivo da ocorrência do overflow.

Oper <sub>1</sub>	Oper <sub>2</sub>	Resultado Impresso	Overflow	Comentários (Justificativas)
Adição				
+	+			
+	-			
-	+			
-	-			
Subtração				
+	+			
+	-			
-	+			
-	-			

## Atividade 6

A multiplicação de inteiros pode ser feita usando números com e sem sinal. A arquitetura especifica 2 registradores especiais, **hi** and **lo** (32 bit each), os quais são os destinos dos resultados de divisão e multiplicação de inteiros. Como visto, a multiplicação inteira pode gerar resultados maiores que a faixa de valores representáveis em 32 bits  $[-2^{31}$  a  $2^{31}-1]$ . A divisão inteira retorna dois valores de 32 bits, resto e quociente, que exigem 32 bits, cada, para sua representação.

Multiplicar 2 valores inteiros “*unsigned*” de  $n$  bits pode resultar em valores que requerem  $2n$  bits para serem representados. Dividir 2 valores inteiros “*unsigned*” de  $n$  bits resulta em 2 valores de  $n$  bits, um para o quociente e outro para o resto da divisão. Para multiplicação com valores inteiros “*signed*”, os valores negativos estarão representados em complemento de 2. O resultado, se for negativo, também estará representado em complemento de 2. Assim, o resultado de uma multiplicação “com sinal” requer  $2 \cdot (n - 1)$  bits para a representação da magnitude do valor e mais 1 bit de sinal.

A multiplicação inteira com ou sem sinal nunca requererá mais que 64 bits para o resultado e, portanto, nunca ocorrerá um overflow quando instruções nativas de multiplicação inteira forem utilizadas. As instruções nativas para a multiplicação são as seguintes:

Instruction	Comment
<code>mult Rsrc1, Rsrc2</code>	Multiply the signed numbers in <code>Rsrc1</code> and <code>Rsrc2</code> . The higher 32 bits of the result go in register <b>hi</b> . The lower 32 bits of the result go in register <b>lo</b>
<code>multu Rsrc1, Rsrc2</code>	Multiply the unsigned numbers in <code>Rsrc1</code> and <code>Rsrc2</code> . The higher 32 bits of the result go in register <b>hi</b> . The lower 32 bits of the result go in register <b>lo</b>

Por outro lado, a máquina virtual do simulador oferece várias instruções de multiplicação que produzem um resultado em 32 bits, com o registrador destino especificado como sendo um registrador de uso geral (e não os registradores especiais **hi** e **lo**).

## Pseudoinstruções Nativas para Multiplicação Inteira em MIPS

Instruction	Comment
<code>mul Rdest, Rsrc1, Rsrc2</code>	Multiply the signed numbers in <code>Rsrc1</code> and <code>Rsrc2</code> . The lower 32 bits of the result go in register <code>Rdest</code> .
<code>mulo Rdest, Rsrc1, Rsrc2</code>	Multiply the signed numbers in <code>Rsrc1</code> and <code>Rsrc2</code> . The lower 32 bits of the result go in register <code>Rdest</code> . Signal overflow
<code>mulou Rdest, Rsrc1, Rsrc2</code>	Multiply the unsigned numbers in <code>Rsrc1</code> and <code>Rsrc2</code> . The lower 32 bits of the result go in register <code>Rdest</code> . Signal overflow

O resultado de uma divisão inteira é composto por um quociente (armazenado no registrador **lo**) e um resto (armazenado no registrador **hi**), sendo ambos números inteiros de 32 bits. Existe uma complicação extra, relacionada com a divisão inteira: o sinal do resto. Existem duas abordagens para tratar o problema: (1) Seguir o teorema de divisão da matemática ou (2) usar uma solução baseada na visão da ciência da computação.

Dividir um inteiro *DD* (dividendo) por um **valor positivo** *DR* (divisor), as seguintes condições devem ser sempre verdadeiras (teorema da divisão):

$$\begin{aligned} DD &= DR \cdot Q + RM & (1) \\ 0 \leq RM < DR & & (2) \end{aligned}$$

Observe que a divisão gera um *Q* (quociente) e um *RM* (resto – *remainder*) que são inteiros únicos e que o **resto é sempre positivo**. Assim, se o dividendo e o divisor tiverem o mesmo sinal, então o quociente é positivo; do contrário, será negativo. **Um resto diferente de zero sempre terá o mesmo sinal do dividendo.**

## Questão 4:

Quais são os resultados das seguintes divisões inteiras?

Dividendo	Divisor	Quociente	Resto
22	7		
-22	7		
22	-7		
-22	-7		

No MIPS, se um dos operandos numa divisão inteira for negativo, então o sinal do resto é “não especificado”. Para se obter o resultado correto da divisão, alguns passos extras devem ser executados:

- Converter ambos os operandos em valores positivos;
- Realizar a divisão;
- Ajustar o resultado para a representação correta, tendo como base os sinais iniciais do dividendo e do divisor.

Uma operação de **divisão com sinal pode gerar overflow**, uma vez que a representação em complemento de 2 de inteiros é assimétrica (a representação do 0 fica do lado das representações positivas e, com isso existe um número negativo a mais do que os valores positivos). Note entretanto, que um *overflow* numa divisão inteira **não gera um sinal** de overflow como no caso da adição/subtração. Ou seja, fica a cargo do programador o tratamento deste erro por meio de um código próprio para tratar o *overflow*. Dividir um valor inteiro por zero é uma operação ilegal! Entretanto, mesmo que o divisor seja zero, a operação de divisão será feita sem reportar um erro de execução. Novamente, cabe ao compilador ou ao programador a tarefa de gerar um código que detecta e trata este tipo de operação ilegal antes que ela aconteça.

## Instruções Nativas para Divisão Inteira em MIPS

Instruction	Comment
<code>div Rsrc1, Rsrc2</code>	Divide the signed integer in <code>Rsrc1</code> by the signed integer in <code>Rsrc2</code> . The quotient (32 bits) goes in register <code>lo</code> . The remainder goes in register <code>hi</code> . Can overflow.
<code>divu Rsrc1, Rsrc2</code>	Divide the unsigned integer in <code>Rsrc1</code> by the unsigned integer in <code>Rsrc2</code> . The quotient (32 bits) goes in register <code>lo</code> . The remainder goes in register <code>hi</code> . Does not overflow.

## Pseudoinstruções para Divisão Inteira em MIPS

Instruction	Comment
<code>div Rdest, Rsrc1, Rsrc2</code>	Divide the signed integer in <code>Rsrc1</code> by the signed integer in <code>Rsrc2</code> . Store the quotient (32 bits) in register <code>Rdest</code> . Can overflow.
<code>divu Rdest, Rsrc1, Rsrc2</code>	Divide the unsigned integer in <code>Rsrc1</code> by the unsigned integer in <code>Rsrc2</code> . Store the quotient in <code>Rdest</code> . Does not overflow.
<code>rem Rdest, Rsrc1, Rsrc2</code>	Divide the signed integer in <code>Rsrc1</code> by the signed integer in <code>Rsrc2</code> . Store the remainder <sup>a</sup> (32 bits) in register <code>Rdest</code> . Can overflow.
<code>remu Rdest, Rsrc1, Rsrc2</code>	Divide the signed integer in <code>Rsrc1</code> by the signed integer in <code>Rsrc2</code> . Store the remainder <sup>a</sup> (32 bits) in register <code>Rdest</code> . Does not overflow.

Se um dos operandos for negativo, o resto terá sinal indefinido.

Assim como na multiplicação, a máquina virtual do MARS oferece instruções de divisão que podem ser armazenadas em registradores de propósito geral (e não os registradores especiais `hi` e `lo`).

## Atividade 7

Escreva um programa usando linguagem C, chamado *lab3.3.c*, no qual você:

- Declara 2 variáveis inteiras, `a` e `b`. Os valores iniciais de `a` e `b` são os maiores valores possível para um inteiro (`INT_MAX` quando você usa um `include limits.h` em seu programa C);
- Multiplica `a` e `b`;
- Imprime o resultado da multiplicação.

## Questão 5:

O programa gera um *overflow*? Explique.

## Questão 6:

O resultado impresso está correto? Explique.

## Atividade 8

Escreva um programa MIPS, usando o simulador MARS, chamado *lab3.4.asm*, no qual você:

- Declara 2 variáveis inteiras, *a* e *b*. Estas variáveis serão usadas para armazenar 2 inteiros digitados pelo usuário, os quais serão armazenados em **\$t0** e **\$t1**;
- Multiplica **\$t0** e **\$t1** usando uma instrução nativa para **inteiros com sinal**;
- Imprime o resultado da multiplicação.

Execute o programa e preencha a tabela a seguir com seu plano de testes. Use as ferramentas do simulador MARS para verificar o conteúdo dos registradores **hi** e **lo**. Inclua o maior e menor inteiro representável em 32 bits para os valores *a* e *b* nas duas últimas linhas da tabela.

**Conteúdos dos registradores para os testes do programa *lab3.4.asm* (multiplicação com sinal)**

a	b	Registrador <b>hi</b> (hexadecimal)	Registrador <b>lo</b> (hexadecimal)
2	1		
2	-1		
$2^{18}$	$2^{14}$		
$2^{18}$	$-2^{14}$		

Repita a execução do programa *lab3.4.asm* usando os mesmos valores anteriores e preencha a tabela a seguir com os resultados esperados e obtidos nesta execução.

**Saída para os testes do programa *lab3.4.asm* (multiplicação com sinal)**

a	b	Resultado Esperado	Resultado Obtido
2	1		
2	-1		
$2^{18}$	$2^{14}$		
$2^{18}$	$-2^{14}$		

## Questão 7:

Alguns dos resultados impressos não estão corretos. Por quê?

## Atividade 9

Escreva um programa, chamado *lab3.5.asm*, no qual você:

- Declara 2 variáveis inteiras, *a* e *b*. Estas variáveis serão usadas para armazenar 2 inteiros digitados pelo usuário, os quais serão armazenados em **\$t0** e **\$t1**
- Multiplica **\$t0** e **\$t1** usando uma instrução nativa para **inteiros sem sinal**.
- Imprime o resultado da multiplicação.

De forma similar ao que foi feito anteriormente, execute o programa e preencha a tabela a seguir com seu plano de testes. Inclua o maior e menor inteiro representável em 32 bits para os valores *a* e *b* nas duas últimas linhas da tabela.

**Conteúdos dos registradores para os testes do programa *lab3.4.asm* (multiplicação sem sinal)**

a	b	Registrador <b>hi</b> (hexadecimal)	Registrador <b>lo</b> (hexadecimal)
2	1		
2	-1		
$2^{18}$	$2^{14}$		
$2^{18}$	$-2^{14}$		



--	--	--	--

Repita a execução do programa *lab3.5.asm* usando os mesmos valores anteriores e preencha a tabela a seguir com os resultados esperados e obtidos nesta execução.

## Atividade 10

Escreva um programa usando MIPS e o simulador MARS, chamado *lab3.6.asm*, no qual você:

- Declara 2 variáveis inteiras, *a* e *b*. Estas variáveis serão usadas para armazenar 2 inteiros digitados pelo usuário, os quais serão armazenados em **\$t0** e **\$t1**;
- Divide **\$t0** e **\$t1** usando uma instrução nativa para **inteiros com sinal**;
- Imprime o quociente e o resto da divisão.

Execute o programa e preencha a tabela a seguir com seu plano de testes. Use as ferramentas do simulador MARS para verificar o conteúdo dos registradores **hi** e **lo**. Use números “pequenos” que gerem resto diferente de zero. Para as últimas duas linhas, insira valores em **\$t0** e **\$t1** que gerem *overflow* na execução da divisão. Note que divisão por 0 (que é uma operação ilegal) e *overflow* são coisas diferentes em termos de execução. Na coluna “Erro” indique quando um erro de execução for observado. Destaque as colunas com valores errados.

**Plano de testes do programa *lab3.6.asm* (divisão com sinal)**

Oper <sub>1</sub>	Oper <sub>2</sub>	Quociente impresso	Resto impresso	Erro
> 0	> 0			
> 0	< 0			
< 0	> 0			
< 0	< 0			
-1	2			
	= 0			

## Atividade 11

Escreva um programa usando MIPS e o simulador MARS, chamado *lab3.7.asm*, no qual você:

- Declara 2 variáveis inteiras, *a* e *b*. Estas variáveis serão usadas para armazenar 2 inteiros digitados pelo usuário, os quais serão armazenados em **\$t0** e **\$t1**;
- Divide **\$t0** e **\$t1** usando uma instrução nativa para **inteiros sem sinal**;
- Imprime o quociente e o resto da divisão.

De maneira similar ao feito para a divisão de inteiros com sinal, execute o programa e preencha a tabela a seguir com seu plano de testes, usando os mesmos valores anteriores de entrada dos operandos. Destaque (em negrito) as colunas com valores errados.

**Plano de testes do programa *lab3.6.asm* (divisão com sinal)**

Oper <sub>1</sub>	Oper <sub>2</sub>	Quociente impresso	Resto impresso	Erro
> 0	> 0			
> 0	< 0			
< 0	> 0			
< 0	< 0			
-1	2			
	= 0			