Fundamentos de Arquitetura de Computadores - Turma B

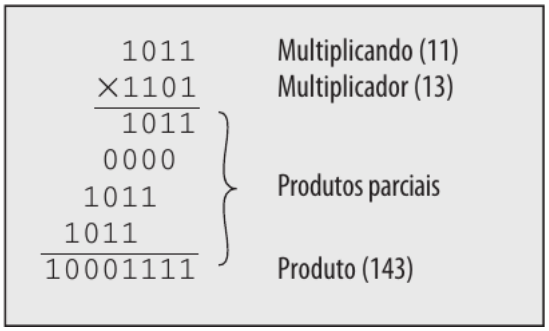
Trabalho 2

Algoritmo para multiplicação de inteiros

21 de outubro de 2019

Efetuar a multiplicação de dois números inteiros não é uma tarefa trivial que pode ser efetuada diretamente no *hardware*. Para fazê-la, é necessário dispor de um algoritmo que use de dispositivos físicos, de forma sistemática, para chegar a um resultado.

A ideia geral de um algoritmo de multiplicação parte da mesma lógica que usamos no papel. Observe a Figura 1.

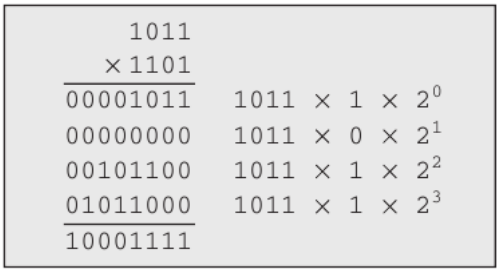


Fonte: [1, Figura 9.7].

Figura 1: Exemplo de multiplicação de dois inteiros.

Chamemos o multiplicando de M e o multiplicador de Q. Consideremos que Q seja tal que $Q = q_{n-1}q_{n-2}\ldots q_2q_1q_0$. O que fazemos no papel é percorrer cada dígito q_i do multiplicador, começando do dígito menos significativo q_0 até o mais significativo q_{n-1}, e calculamos o produto parcial $M \times q_i$. Ao final, todos os produtos parciais, cada um deslocado i dígitos à direita, são somados, o que nos dá o produto final.

Sabemos que deslocar um binário i vezes à direita equivale a multiplicá-lo por 2^i (assim como deslocar um decimal i vezes à direita equivale a multiplicá-lo por 10^i). Ou seja, o que fazemos no papel, ilustrado na Figura 1, equivale ao que está ilustrado na Figura 2.



Fonte: [1, Figura 9.10].

Figura 2: Exemplo de multiplicação de dois inteiros. Note que podemos colocar zeros à direita em cada produto parcial, já que estamos deslocando-os à esquerda.

Logo,

$$M \times Q = M \times (q_0 \times 2^0 + q_1 \times 2^1 + \ldots + q_{n-2}2^{n-2} + q_{n-1} \times 2^{n-1}).$$

O problema é que esse esquema de multiplicação não é compatível com a representação de números negativos em complemento a dois. O que se faz muitas vezes é aplicar o **Algoritmo de Booth**. Esse algoritmo funciona tanto para números negativos quanto positivos, e ainda é mais econômico, em geral, que o primeiro.

Para entender o funcionamento do algoritmo de Booth, consideremos primeiro o caso em que o multiplicador seja positivo. Consideremos, por exemplo, o número 00011110_{bin}, composto por um bloco de 1s cercado de 0s. Temos que

$$\begin{aligned} M \times (00011110) &= M \times (2^4 + 2^3 + 2^2 + 2^1) \\ &= M \times (16 + 8 + 4 + 2) \\ &= M \times 30. \end{aligned}$$

Como vale que

$$2^n + 2^{n-1} + \ldots + 2^{n-k} = 2^{n+1} - 2^{n-k} \tag{1}$$

para $n > 0$ e $1 \leq k \leq n$ (confira você mesmo!), então

$$\begin{aligned} M \times (00011110) &= M \times (2^5 - 2^1) \\ &= M \times (32 - 2) \\ &= M \times 30. \end{aligned}$$

Por isso, o produto pode ser calculado por sucessivas adições e subtrações do multiplicando sempre que houver uma mudança de 0 para 1 (neste caso, subtração) ou de 1 para 0 (neste caso, adição).

Já para o caso negativo, consideremos X um número negativo representado na notação de complemento a dois:

$$\text{Representação de } X = 1x_{n-2}x_{n-3}\ldots x_1x_0.$$

O valor de X é

$$X = -2^{n-1} + (x_{n-2} \times 2^{n-2}) + (x_{n-3} \times 2^{n-3}) + \ldots + (x_1 \times 2^1) + (x_0 \times 2^0).$$

O bit mais significativo de X é 1, já que X é negativo. Vamos supor que o primeiro 0 menos significativo esteja na posição k. Deste modo,

$$\text{Representação de } X = 111\ldots10x_{k-1}x_{k-2}\ldots x_1x_0.$$

Ou seja,

$$X = -2^{n-1} + 2^{n-2} + \ldots + 2^{k+1} + (x_{k-1} \times 2^{k-1}) + \ldots + (x_0 \times 2^0).$$

Usando novamente a relação (1), temos que

$$2^{n-2} + \ldots + 2^{k+1} = 2^{n-1} - 2^{k+1},$$

isto é

$$X = -2^{n-1}+2^{n-2}+\ldots+2^{k+1}+(x_{k-1}\times2^{k-1})+\ldots+(x_0\times2^0) = -2^{k+1}+(x_{k-1}\times2^{k-1})+\ldots+(x_0\times2^0).$$

Logo,

$$X = -2^{k+1} + (x_{k-1} \times 2^{k-1}) + \ldots + (x_0 \times 2^0).$$

Isso indica que, à medida que o algoritmo de Booth passe o zero mais significativo e encontra o próximo 1, ocorre uma transição de 0 para 1 e há uma subtração (-2^{k-1}).

O algoritmo de Booth necessita de

- um registrador de 32 bits para armazenar o `Multiplicador`,
- um registrador de 64 bits para armazenar o `Produto` e
- um bit extra na posição menos significativa do produto para armazenar o último bit descartado.

O algoritmo de Booth é descrito a seguir.

Algoritmo 1: Algoritmo de Booth

Dados: O multiplicando e o multiplicador, números inteiros (negativos em complemento a dois).

Passo 1. Copie o `Multiplicador` para a porção de 32 bits menos significativa e zere a porção de 32 bits mais significativa do registrador `Produto`.

Passo 2. Se `Produto`$[0..-1] = 01$ (`Produto`$[0..-1] = 10$), some (subtraia) o multiplicando à porção de 32 bits mais significativa do registrador `Produto` e armazene o resultado na porção de 32 bits mais significativa do registrador `Produto`.

Passo 3. Faça o deslocamento aritmético do registrador `Produto` 1 bit à direita.

Passo 4. Se não for a 32ª repetição, volte ao Passo 2.

Exemplos

1. Queremos multiplicar $7_{\text{dec}} \times 3_{\text{dec}} = 0111_{\text{bin}} \times 0011_{\text{bin}}$.

It.	Passo	`Produto`
0	Valores Iniciais	0000 0011 **0**
1	2. `Produto`[0..-1] = 10 ⇒ `Produto`[7..4] -= `Multiplicando`	1001 0011 **0**
	3. Deslocamento aritmético à direita do `Produto`	1100 1001 **1**
2	2. `Produto`[0..-1] = 11 ⇒ nada a fazer	1100 1001 **1**
	3. Deslocamento aritmético à direita do `Produto`	1110 0100 **1**
3	2. `Produto`[0..-1] = 01 ⇒ `Produto`[7..4] += `Multiplicando`	0101 0100 **1**
	3. Deslocamento aritmético à direita do `Produto`	0010 1010 **0**
4	2. `Produto`[0..-1] = 00 ⇒ nada a fazer	0010 1010 **0**
	3. Deslocamento aritmético à direita do `Produto`	0001 0101 **0**

Tudo certo, $00010101_{\text{bin}} = 21_{\text{dec}}$.

2. Queremos multiplicar $7_{\text{dec}} \times -3_{\text{dec}} = 0111_{\text{bin}} \times 1101_{\text{bin}}$.

It.	Etapa	Produto
0	Valores Iniciais	0000 1101 **0**
1	2. `Produto`[0..-1] = 10 ⇒ `Produto`[7..4] -= `Multiplicando`	1001 1101 **0**
	3. Deslocamento aritmético à direita do `Produto`	1100 1110 **1**
2	2. `Produto`[0..-1] = 01 ⇒ `Produto`[7..4] += `Multiplicando`	0011 1110 **1**
	3. Deslocamento aritmético à direita do `Produto`	0001 1111 **0**
3	2. `Produto`[0..-1] = 10 ⇒ `Produto`[7..4] -= `Multiplicando`	1010 1111 **0**
	3. Deslocamento aritmético à direita do `Produto`	1101 0111 **1**
4	2. `Produto`[0..-1] = 11 ⇒ nada a fazer	1101 0111 **1**
	3. Deslocamento aritmético à direita do `Produto`	1110 1011 **1**

Tudo certo, $11101011_{\text{bin}} = -21_{\text{dec}}$.

O presente trabalho tem por objetivos

- o exercício de conceitos fundamentais de assembly MIPS e
- o exercício de entendimento e fixação do algoritmo de multiplicação de Booth.

Para tanto, faremos uso do simulador MARS[1].

Este trabalho compõe a média semestral de trabalhos e pode ser feito em **dupla**!

Sua tarefa neste trabalho é **implementar um programa** em assembly MIPS que leia dois inteiros a e b e devolva a multiplicação de a por b. Lembre que uma multiplicação pode potencialmente ocupar *dois registradores* e ser representada em 64 bits. Os **requisitos mínimos** que seu programa **deve** cumprir são:

1. Você deve implementar o Algoritmo 1.
2. Seu programa deve ser implementado em assembly MIPS e funcionar no simulador MARS versão 4.5.
3. Seu programa deve
   (a) Conter um cabeçalho com o nome completo e matrícula dos membros da dupla,
   (b) Conter uma função `multfac` que receba como **argumentos** os dois inteiros e salve o resultado do algoritmo nos registradores `Hi` e `Lo` e
   (c) Conter uma função `main` que leia os dois inteiros, chame a função `multfac` e exiba, ao final, o resultado.

Caso algum dos requisitos mínimos não seja cumprido, serão aplicados descontos proporcionais à nota, *independentemente da corretude do código*. Leitura também é parte deste trabalho!

Depois de ter cumprido todos os requisitos, **teste** seu código. Invente casos de teste, isso também é sua tarefa neste trabalho.

Depois de ter cumprido todos os requisitos e testado seu código, você deve submeter apenas seu código fonte, com a extensão `asm` e nomeado da seguinte forma

[1] Disponível em http://courses.missouristate.edu/KenVollmar/mars/.

nome1_matricula1_nome2_matricula2_**trab02**.asm

no link Entrega do Trabalho 2 na página da nossa disciplina no Moodle até o dia **04 de novembro de 2019** às **23:55**.

Apenas **um membro da dupla** deve submeter o trabalho no Moodle.

Será avaliado no seu trabalho

1. a corretude do seu código em assembly MIPS, ou seja
   - seu código roda sem erros,
   - não contém uso indevido de funções e estruturas e
   - funciona,
2. o capricho no código e na indentação,
3. o cumprimento de todos os requisitos mínimos,
4. o envio correto na plataforma Aprender e
5. a pontualidade na entrega do trabalho.

Farei uma correção manual, então seu código não vai rodar em juiz eletrônico. Se for detectado algum plágio, a nota atribuída será ZERO a **todos** os envolvidos.

Bons estudos!

Prof. John Lenon Gardenghi
john.gardenghi@unb.br
Sala 22-UED

Referências

[1] STALLINGS, W. **Arquitetura e organização de computadores**. 8 ed. Prentice Hall. 2010.