Universidade de Brasília - UnB Gama

Fundamentos de Arquitetura de Computadores

Relatório - Somativa 2
Multiplicação

Raquel Temóteo Eucaria Pereira da Costa - 202045268

Brasília-DF, 06 de Janeiro de 2023

1 Objetivo

Fazer a descrição do algoritmo de multiplicação realizado e submetido no CD-MOJ. Além de explicar os detalhes da implementação.

2 Problema

O problema a ser resolvido é basicamente criar uma função que multiplique dois inteiros de 4 bytes em Assembly MIPS.

O procedimento deve:

- Ser chamado de `multfac`
- Receber o multiplicando e o multiplicador nos registradores `$a0` e `$a1`;
- Retornar o produto nos registradores `hi` e `lo`;
- Ser capaz de lidar com inteiros com sinal;
- Não ter dados de entrada para serem lidos;
- Não ter dados de saídas para serem impressos.
- Não utilizar instruções `mu*` ou `madd*`

3 Entendendo o problema

3.1 Multiplicação à mão

Revendo os conceitos de multiplicação, fazemos a multiplicação de decimais na mão.

Multiplicando		1000_{dec}
Multiplicador	×	1001_{dec}

		1000
		0000
		0000
		1000

Produto		1001000_{dec}

Figura 1: Multiplicação na mão

O primeiro operando é o multiplicando e o segundo é o multiplicador. O resultado final é chamado de produto. Pegamos os dígitos do multiplicador

um a um, da direita para a esquerda, calculando a multiplicação do multiplicando pelo único dígito do multiplicador e deslocando o produto intermediário um dígito para a esquerda dos produtos intermediários anteriores.

O número de dígitos no produto é muito maior do que o número no multiplicando ou no multiplicador. De fato, se ignorarmos os bits de sinal, o tamanho da multiplicação de um multiplicando de n bits por um multiplicador de m bits é um produto que possui n + m bits de largura. Ou seja, n + m bits são necessários para representar todos os produtos possíveis. A multiplicação precisa lidar com o overflow, pois constantemente desejamos um produto de 32 bits como resultado da multiplicação de dois números de 32 bits. Neste exemplo, restringimos os dígitos decimais a 0 e 1. Com somente duas opções, cada etapa da multiplicação é simples:

- **Igual a 1:** Colocar uma cópia do multiplicando (1 × multiplicando) no lugar apropriado se o dígito do multiplicador for 1 ou
- **Igual a 0:** Colocar 0 (0 × multiplicando) no lugar apropriado se o dígito for 0. Embora o exemplo decimal anterior utilize apenas 0 e 1, a multiplicação de números binários sempre usa 0 e 1 e, por isso, sempre oferece apenas essas duas opções.

3.2 Passos do algoritmo

Para construir um algoritmo representando essa multiplicação, teriamos os seguintes passos:

1. Inicialize $P = 0$ e $contador = 1$;
2. Faça $P = P + Q0 * M$;
3. Faça o deslocamento lógico de um bit à esquerda em M;
4. Faça o deslocamento lógico de um bit à direita em Q;
5. Se $contador = 32$, pare. Se não, $contador = contador + 1$ e volte ao passo 2.

Ao construir o algoritmo estabelecemos 3 registradores, o registrador do Multiplicando(M), a ALU, o registrador do Produto(P) possuem 64 bits de largura, e do Multiplicador(Q) contendo 32 bits. O multiplicando vai sendo deslocado para a esquerda e o multiplicador a direita. O produto começa como 0 e terá um controle que decidirá quando realizar o deslocamento do M e do Q e quando somar/escrever novos valores no P.

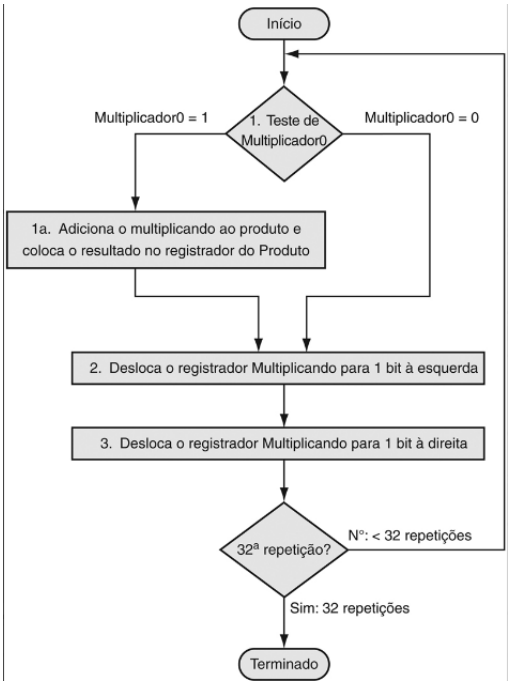

Figura 2: Representação do algotimo

Desse modo, obtemos representação do algoritmo conforme a Figura 2.

As iterações do código aconteceriam com a primeira metade dos bits do multiplicador inutilizados e conforme fosse movido a esquerda eles iriam recebendo os valores multiplicados, resultando no produto desejado. Da seguinte forma:

Iteração	M($\leftarrow$	Q($\rightarrow$)	P
0	0000 1000	100**1**	0000 0000
1	0000 1000	100**1**	0000 1000
2	0001 0000	10**0**	0000 1000
3	0010 1000	1**0**	0000 1000
4	0100 0000	**1**	**0100 1000**

Tabela 1: Iterações de 1000 x 1001

4 Otimização do algoritmo

Para otimizar o algoritmo ao invés de deslocar o multiplicando à esquerda, deslocamos o produto para a direita. E salvamos o multiplicador na porção menos significativa do produto. Assim, os algoritmos que ficavam inutilizados serão substituidos pelo Q e ficarão no final do M, assim a cada deslocamento, será recebido o valor multiplicado, resultando no P.

4.1 Passos

1. $P[63...32] = 0$
2. $P[31...0] = Q$
3. Se $P[0] = 1$, $P[63...32] = P[63...32] + M$
4. Faça um deslocamento de 1 bit à direita em P
5. Se não for a 32^a repetição, volte ao Passo 3.

Já as iterações do código aconteceriam de forma que a primeira metade dos bits do multiplicador seriam inutilizados e conforme fosse movido a direita eles iriam recebendo os valores multiplicados, resultando no produto desejado. Da seguinte forma:

Iteração	Produto	Deslocamento
0	0000 **1001**	x
1	1000 **1001**	0100 010**0**
2	0100 0**100**	0010 001**0**
3	0010 00**10**	0001 000**1**
4	1001 000**1**	**0100 1000**

Tabela 2: Iterações de 1000 x 1001 otimizado

5 Conversão do sinal

Para tratar da questão do sinal dos algarismo, devemos lembrar que o produto só irá ser negativo, se o multiplicando e o multiplicador tiverem sinais diferente (+ - ou - +). Assim, podemos fazer o cálculo com os valores do M e do Q positivos e no final, se os sinais forem diferentes, invertemos o resultado.

6 Instruções hi e lo

É importante entendermos as instruções hi e lo, que são registradores que o MIPS oferece com 32 bits que contém o produto de 64 bits. O hi contém a parte mais significativa e o lo a menos significativa.

Para apanhar o produto de 32 bits inteiro, o programador usa move from lo (mflo). O montador MIPS gera uma pseudoinstrução para multiplicar, que especifica três registradores de uso geral, criando instruções mflo e mfhi que colocam o produto nos registradores.

7 Implementação do código

Com base nas informações acima, o algoritmo de multiplicação foi implementado da seguinte maneira:

- **Passo 1 e 2** - Inicializando o P com a $parteAlta = 0$ e $parteBaixa = Q$.

```
passo1e2:
    move    $t0, $zero # Passo 1 -> P[63...32] = 0
    move    $t1, $a1 # Passo 2 -> P[31...0] = Q
```

Figura 3: Passo 1 e 2 - Produto

- **Passo 3** - Verificando se soma ou não, utilizando uma máscara (andi) que vai pegar o Q e se a parte mais à esquerda tiver 1 ele vai somar o M, se não ele não faz nada.

```
# Passo 3 -> Se P[0]=1, P[63...32] = P[63...32] + M
passo3:
    andi    $t2, $t1, 1     # $t2 = $t1 & 1 -> Verfica se soma ou não
    beq     $t2, $zero,passo4
        add     $t0, $t0, $a0 # ++
```

Figura 4: Passo 3 - Soma ou não

- **Passo 4** - Deslocamento de 1 bit à direita em P

```
passo4:
    andi    $t3, $t0, 1 # máscara para ver se passa 1 ou 0
    srl     $t0, $t0,1
    srl     $t1, $t1,1
    sll     $t3, $t3,31 # deslocamento a esquerda LSB
    add     $t1, $t1,$t3
```

Figura 5: Passo 4 - Deslocamento

- **Passo 5** - Verifica o contador, se não for a 32^a repetição, volte ao Passo 3.

```
#passo 5 -> Verifica contador -> Se não for 32a repeticao, volta passo3
    addi    $t4, $t4, -1
    beq     $t4, $zero, passo6
        j passo3
```

Figura 6: Passo 5 - Loop

- **Passo 6** - Inverte o sinal se os sinais forem diferente. Primeiro foi analisado o sinal do M e do Q no início do código.

```
# verificando o sinal
slt     $t8, $a0, $zero  # se a0 < zero então t8 = 1 senão t8 = 0
slt     $t9, $a1, $zero  # se a1 < zero então t9 = 1 senão t9 = 0

beq     $t8, $zero, comparaSinal
        nor     $a0, $a0, $zero
        addi    $a0, $a0, 1

comparaSinal:
    beq     $t9, $zero, passo1e2
        nor     $a1, $a1, $zero
        addi    $a1, $a1, 1
```

Figura 7: Passo 6.1 - Analisando sinal

```
# Verifica se precisa inverte sinal (+/- ou -/+ ) e inverte
passo6:
    beq $t8,$t9,resultado # (1/0 ou 0/1 - linha 11 e 12)
        nor $t0,$t0,$zero   # inverte parte alta
        # complemento de 2:
        nor $t1,$t1,$zero   # inverte parte baixa
        addi $t1, $t1, 1    # soma 1 parte baixa
```

Figura 8: Passo 6.2 - Tratando sinal

- **Passo 7** - Retornar o `hi` e `lo`

```
# Resultado: Retorno do hi e do lo
passo7:
    mtlo $t1
    mthi $t0

    jr $ra
```

Figura 9: Passo 7 - hi e lo

Referências

[1] David A. Patterson, John LeRoy Hennessy *Organização e Projeto de Computadores*, 5^a Ed.