Fundamentos de Arquitetura de Computadores - 2022/2
Avaliação Somativa 2

Edilberto Almeida Cantuaria
22/2014984

06/02/2023

Relatório do código

Trata-se de um código em Assembly Mips que realiza a multiplicação de 2 inteiros de 32 bits que são armazenados nos registradores $a0 e $a1. O código começa verificando se os registradores $a0 e $a1 é menor que zero (linha 3 e 4) utilizando a instrução "slti", conforme pode ser observado na figura 1. Caso $a0 seja negativo, a rotina "azeroNegativo"(linha 11) é executada e nega o valor do registrador $a0 ("convertido"em um número positivo) utilizando a instrução "nor"e incrementado em 1 (linhas 12 e 13). O mesmo procedimento ocorre para o registrador $a1, executando a rotina "aumNegativo"(linha 19 a 21). O procedimento de negar um valor utilizando o comando "nor"seguido do incremento de 1 denomina-se negação em complemento de dois.

```
1   multfac:
2
3   slti $t8,$a0, 0 #se a0<0 (ou seja, a0 e negativo), t8 = 1, senao t8=0 (ou seja, a0 e positivo)
4   slti $t9,$a1, 0 #se a0<0 (ou seja, a0 e negativo), t9 = 1, senao t9=0 (ou seja, a0 e positivo)
5
6   li $t2, 1 #auxiliar
7
8   beq $t8, $t2, azeroNegativo #se t8==t2 (ou seja, a0 negativo)
9   j verificar_aumNegativo   #senao, pula para verificar_aumNegativo (ou seja, a0 e positivo)
10
11     azeroNegativo: #invertendo o sinal do a0
12     nor $a0, $a0, $zero
13     addi $a0, $a0, 1
14
15  verificar_aumNegativo:
16  beq $t9, $t2, aumNegativo #se t9==t2 (ou seja, a1 negativo)
17  j multiplicacao #senao, comeca a multiplicacao
18
19     aumNegativo:
20     nor $a1, $a1, $zero
21     addi $a1, $a1, 1
22
```

Figura 1:

Conforme mostra a figura 2, nas linhas seguintes é realizado a multiplicação. Na linha 25 o registrador $t0 é inicializado com valor zero, o que é importante para armazenar o resultado da multiplicação, principalmente a parte alta que corresponde a 32 bits. O registrador $t1 é inicializado com valor do registrador $a1, que é um dos operandos da multiplicação (linha 26). Já na linha 29 é feita uma verificação do último bit do operando

$t1 usando a instrução "andi". A máscara usada é 1, o permitindo verificar se o bit é 0 ou 1. Se for 0, o controle é transferido para a etapa de deslocamento de bit (linha 33), caso contrário, o registrador $t0 é incrementado com o valor de $a0.

A partir da linha 33 a linha 39 ocorre a etapa de deslocamento de bit, onde o registrador $t0 é deslocado 1 bit para a direita usando o comando "srl". O mesmo acontece com o registrador $t1. O bit "perdido"é adicionado ao registrador $t1 para preservar o resultado da multiplicação. E as linhas 42 a 45 são as responsáveis por verificar o número de iterações, utilizando a instrução "slti": se o contador for menor que 32, o controle é transferido de volta para a etapa de conferência do último bit (linha 28). Caso contrário, a multiplicação é encerrada.

Vale a pena dar uma atenção especial para a label "deslocaBit". A técnica de deslocamento de bits é fundamental para realizar multiplicação em processadores, principalmente em baixo nível, como o MIPS, onde as operações matemáticas como adição e multiplicação são mais caras em tempo e recursos. A técnica de multiplicação bit-a-bit (Algoritmo de Booth) usa deslocamento de bits para realizar a multiplicação de forma mais eficiente, verificando cada bit do número a ser multiplicado, e caso o bit seja 1, adicionar o outro número ao resultado parcial. Depois de cada verificação de bit, ambos os números são deslocados para a direita para que o próximo bit possa ser verificado.

```
23  #Inciando multiplicacao:
24  multiplicacao:
25  move $t0, $zero  #inicializando P - parte alta
26  move $t1 , $a1    #t1 = a1 - parte baixa
27
28  conferirUltimoBit:
29  andi $t3, $t1, 1 #mascara para conferir se o bit eh o mesmo. 0 se falso e 1 se verdadeiro
30  beq $t3, $zero, deslocaBit #se t3==0, pula para o proximo passo que eh o deslocamento de bit
31  addu $t0, $t0, $a0 #t0 = t0 + a0
32
33  deslocaBit:
34  andi $t6, $t0, 1 # verifica o valor do último bit do numero armazenado em $t0. O resultado é armazenado em $t6.
35  srl $t0, $t0, 1    # realiza o deslocamento de bits em $t0, movendo todos os bits 1 posicao para a direita e descartando o último bit.
36  srl $t1, $t1, 1    # semelhante a linha 35
37
38  sll $t6, $t6, 31 # realiza um deslocamento de bits do resultado armazenado em $t6, movendo-o 31 posicoes para a esquerda
39  add $t1, $t1, $t6 # adiciona o resultado do deslocamento de bits em $t6 ao numero armazenado em $t1.
40
41  #verificandoNumIteracoes
42  slti $t5, $t2, 32
43  beq $t5, $zero, fimCodigo
44  addi $t2, $t2, 1
45  j conferirUltimoBit
46
```

Figura 2:

O trecho "fimCodigo"encaminha o código para finalizar seu procedimento. a Instrução declarada na linha 48 verifica se os valores dos registradores $t8 e $t9 (declarados nas linhas 3 e 4 conforme pode ser visto na figura 1) são iguais. Se os valores são iguais, signifca que os sinais do registradores $a0 e $a1 são os mesmos, portanto o produto será positivo. Porém, caso contrário, o produto será negativo e há uma necessidade de "desfazer"a operação executada nas linhas 8 a 21 (conforme pode ser visto na figura 1) seguido da adição de 1 ao valor armazenado no registrador $t1 com a instrução da linha 51. Em seguida as instruções "mtlo $t1"e "mthi $t0"são executadas para mover os valores armazenados em $t0 e $t1 para os registradores mthi e mtlo, respectivamente. Por fim, a instrução "jr $ra"é executada para retornar ao endereço de chamada da subrotina, voltando para a main. A parte final do código pode ser visto na figura 3.

```
47  fimCodigo:
48  beq $t8, $t9, exit #se $t8==$t9, vai para o final
49  nor $t0, $t0, $zero
50  nor $t1, $t1, $zero
51  addi $t1, $t1, 1
52
53
54  exit:
55  mtlo $t1
56  mthi $t0
57
58  jr $ra
59
```

Figura 3:

A figura , retirada do livro Computer Organization and Design: The Hardware/Software Interface"de David A. Patterson e John L. Hennessy, 5ª edição, ilustra um fluxograma do código.

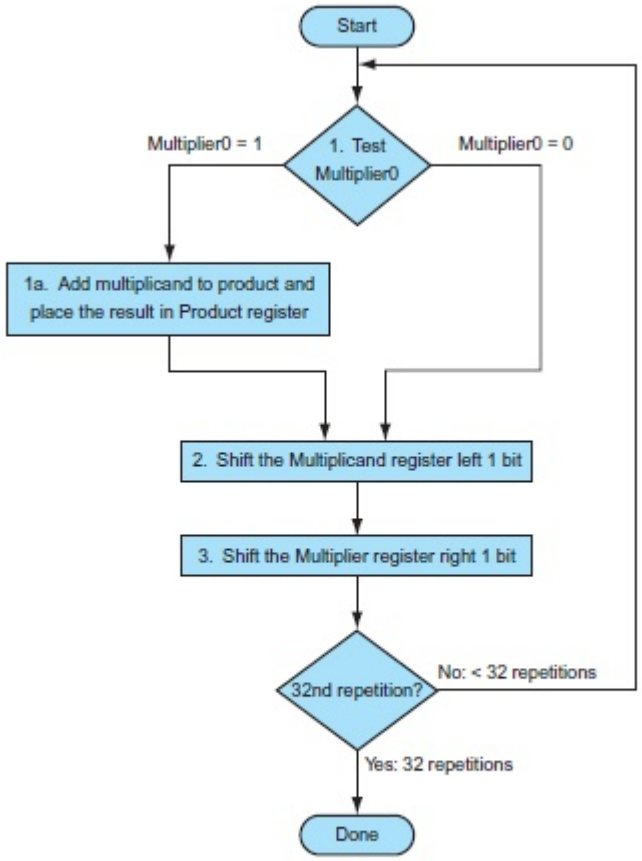

FIGURE 3.4 The first multiplication algorithm, using the hardware shown in Figure 3.3. If the least significant bit of the multiplier is 1, add the multiplicand to the product. If not, go to the next step. Shift the multiplicand left and the multiplier right in the next two steps. These three steps are repeated 32 times.

Figura 4: