Anexo B

Multiplicações não inteiras

Frequentemente é necessário multiplicar um numero inteiro (**`ni`**) por uma constante não inteira (**`k`**).

Se for usado um processador de virgula flutuante, o caso fica resolvido automaticamente.

Se for usado um processador de virgula fixa como é o caso do μC SAB80C167, a solução mais imediata é a utilização de números em virgula flutuante através da sua codificação nas palavras nativas do μC e realizar as várias operações aritméticas com o recurso a rotinas em software.

No entanto o uso de números em virgula flutuante numa máquina de virgula fixa, leva a um peso computacional elevado. Uma simples soma entre dois números requer um conjunto razoável de instruções máquina.

Outra hipótese é separar as partes inteira (**`ki`**) e fraccionária (**`kf`**) de **`K`** e realizar o calculo em três partes:

- Multiplicar a parte inteira: **`r1=ni*ki`**;
- Multiplicar a parte fraccionária através da instrução de divisão: **`r2=ni/(1/ki)`**;
- Somar os resultados: **`r = ni*k = r1+r2`**.

Como é óbvio, isto só é possível numa máquina que possua uma instrução de divisão, como é o caso do μC SAB80C167.

No entanto a instrução de divisão demora muito mais tempo que uma instrução de multiplicação.

Para tornar o processo mais rápido, a multiplicação da parte fraccionária pode ser realizada segundo o procedimento deduzido a seguir.

A instrução *MUL* do C167 executa a multiplicação de 2 números de 16bits e coloca o resultado em dois registos de 16bits: MDH e MDL. MDH representa os 16bits mais significativos da multiplicação e MDL os menos significativos.

Multiplicar um numero binário por 2^n é equivalente juntar **n** zeros à direita do numero.

Dividir um numero binário por 2^n é equivalente a eliminar os seus **n** bits menos significativos (naturalmente o resultado é truncado à parte inteira). A eliminação é feita através de uma rotação de **n** casas binárias.

Isto é o equivalente a multiplicar ou dividir um numero decimal por 10^n. Na verdade, multiplicar um numero representado numa base **b** por um numero b^n é o mesmo que juntar **n** zeros à sua direita.

Se depois de executar a instrução de multiplicação, tomarmos apenas o registo MDH como o resultado, estamos a rodar o resultado **16** posições, que é equivalente a dividi-lo por 2^{16}.

O código descrito a seguir exemplifica isto:

```
função multiplica_divide216:
MOV R1, var1     ;
MOV R2, var2     ;
MUL R1, R2       ;
MOV var3, MDH    ;
```

O código realiza a seguinte atribuição (var1, var2, e var3 são posições de memória (variáveis)):

$$\mathbf{var3 <- var1 * var2 / 2^{16}}$$

Desta forma para realizar a multiplicação de um inteiro **ni**, pela constante fraccionária **kf**, basta utilizar o código acima descrito onde:

$$\mathbf{R1 = ni,}$$

$$\mathbf{R2 = kf * 2^{16}}$$

Assim, o resultado obtido (**var3**) é o valor pretendido:

$$\mathbf{var3 = (ni * kf * 2^{16}) / 2^{16} = ni * kf}$$

Obviamente, o resultado é truncado à parte inteira.

Como o valor $\mathbf{R2 = kf * 2^{16}}$ tem de ser arredondado para um numero inteiro, e o resultado final é truncado à parte inteira, existe um erro.

Na verdade o valor atribuído a **var3** vale:

$$\mathbf{var3 = parte\ inteira\ (\ ni * arredondamento(kf*2^{16}) / 2^{16}\ )}$$

pelo que o erro é dado por:

$$\mathbf{erro = ni*kf - parte\ inteira\ (\ ni * arredondamento(kf*2^{16}) / 2^{16}\ )}$$

Como exemplo suponha-se que é necessário multiplicar a variável *luar* (16 bits) por 7.453 e colocar o resultado na variável *praia* (16 bits).

O código descrito a seguir realiza esta função.

```
; praia = luar * 7.453
; multiplica a parte inteira
  MOV R1, luar   ;
  MOV R2, 7      ;
  MUL R1, R2     ;
  MOV R3, MDL    ;  R3 <- luar * 7
; multiplica a parte fraccionária
  MOV R2, #29688  ;  29688 = round(0.453*2^16)
  MUL R1, R2     ;
  MOV R4, MDH    ;  R4 <- int(luar * round(0.453*2^16))
; soma os resultados
  ADD R3, R4
  MOV praia, R3
```

O erro cometido é dado por:

erro = luar*7.453 - (luar*7 + int(luar*29688/2^{16}))
erro ≅ luar*7.453 - (luar*7 + luar*29688/2^{16})
= luar (7.453-7+29688/2^{16}) = luar (7.453-7.4530029)

que é obviamente desprezável.

Seguindo o mesmo raciocínio, é possível evitar a utilização da instrução de divisão e mesmo o uso de variáveis auxiliares de 32 bits noutras situações.

Suponha-se que se quer obter o seguinte valor:

$$n = \frac{a * b}{50000}$$

onde a, b, e n são variáveis de 16bits.

A solução mais obvia é começar por multiplicar a*b, obtendo um valor de 32bits, seguindo-se a divisão (dos 32bits) por 50000. Finalmente guardam-se os 16bits menos significativos.

No entanto o valor n pode ser dado por:

$$n = \frac{a * b}{50000} \cong \frac{\left[\left(\frac{a * b}{2^{16}}\right) * 42951\right]}{2^{16}} * 2$$

cujo código é o seguinte:

```
; n = a* b /50000
  MOV R1, a   ;
  MOV R2, b      ;
  MUL R1, R2     ;
  MOV R1, MDH    ;  R1 <- a*b/2^16
  MOV R2, #42951 ;
  MUL R1, R2     ;
  MOV R4, MDH    ;  R4 <- ((a*b/2^16)*42951)/2^16
  ROL R4, 1      ;  R4 <- R4 /2 (roda R4 à esquerda)
  MOV n, R3
```

Repare-se que para multiplicar o registo R4 por 2 utilizou-se a instrução

```
ROL Rx,n
```

que roda o registo Rx n posições à esquerda, introduzindo zeros à sua direita. Esta instrução é executada num único ciclo máquina.

O erro cometido è dado por:

$$erro = \mathrm{a} * \mathrm{b}\left[\frac{1}{50000} - \frac{\left[\left(\frac{1}{2^{16}}\right) * 42951\right]}{2^{16}} * 2\right] = a * b * \left(\frac{1}{50000} - \frac{1}{49998.45517}\right) = a * b * 6.179E - 10$$