

Trabalho 2 – Programação em *assembly* ARMv7

Este trabalho tem como objetivo desenvolver aptidões de programação e análise de código em *assembly*. Para tal, iremos recorrer à ferramenta VisUAL (<https://salmanarif.bitbucket.io/visual/index.html>), disponível para Windows, Linux e MAC OS.

Neste trabalho, tirando partido daquilo que aprenderam nas aulas sobre operações em vírgula flutuante, deve ser implementado um conjunto de sub-rotinas que irão permitir realizar operações aritméticas sobre números no formato IEEE-754.

As secções seguintes descrevem as tarefas do trabalho, devendo ser realizadas pela ordem apresentada. Assumir que todas os operandos (com exceção de 0) estão normalizados.

1 Análise e simulação de um programa em *assembly*

Considere um programa que multiplica por 2 todos os elementos de uma sequência de números inteiros armazenada em memória. O código *assembly* seguinte é uma possível solução, na qual se inclui uma sequência com 7 números inteiros para se proceder à respectiva simulação com a ferramenta VisUAL.

```
1 seqA DCD 3, -5, 21, 2, -10, 50, 9 ; sequência
2 tseqA DCD 7 ; tamanho da sequência
3
4     ldr R0, =tseqA ; carrega endereço do número de elementos
5     ldr R0, [R0] ; coloca o número de elementos em R0
6     ldr R1, =seqA ; carrega endereço do primeiro elemento
7 ciclo cmp R0, #0 ; verifica se chegou ao final
8     beq fim ; termina
9     ldr R2, [R1] ; carrega elemento atual
10    lsl R2, R2, #1 ; multiplica por 2
11    str R2, [R1] ; armazena em memória
12    add R1, R1, #4 ; atualiza endereço para ler próximo elemento
13    sub R0, R0, #1 ; atualiza o número de elementos a ler
14    b ciclo
15 fim END
```

1. Após iniciar o VisUAL escreva o código apresentado anteriormente e grave-o num ficheiro (opção **Save** no menu superior).
2. Antes de iniciar a simulação, no menu superior selecione a opção **Tools** → **View memory contents**.
3. Para simular a execução existem duas opções:

- **Execute**, executa o programa completo;
- **Step Forwards**, executa uma instrução, de cada vez que clica nesta opção.
- **Step Backwards**, reverte a execução da última instrução.

Os modos passo-a-passo são mais úteis quando se averigua o funcionamento de um fragmento de código. Em alternativa, no modo **Execute** podem definir-se pontos de paragem (*breakpoints*) em instruções “estratégicas”, onde se pretenda observar o estado de registos e/ou memória, facilitando a depuração de um programa.

4. Responda às seguintes questões:

- (a) Indique o endereço de memória onde está guardado o tamanho da sequência **seqA** (**tseqA**).

R: 0x11C

- (b) Qual é a posição de memória do 4º elemento da sequência **seqA**?

R: 0x10C

- (c) Quantos bytes ocupa a sequência **seqA** em memória?

R: Ocupa 28 byte.

- (d) Quantas vezes é executada a instrução **lsl R2, R2, #1**? (Dica: Use *breakpoints*.)

R: 7 vezes.

- (e) Indique qual ou quais as linhas de código que alteram o valor do registo **R1**. (Dica: Consulte o histórico dos valores do registo.)

R: Linhas 6 e 12.

- (f) Indique quais são os valores de **seqA** após a execução do programa.

R: 6, -10, 42, 4, -20, 100, 18

- (g) Repita a alínea anterior após alterar o valor de **tseqA** para 3.

R: 6, -10, 42

- (h) Se substituir a instrução **lsl R2, R2, #1** por **lsl R2, R2, #3**, que funcionalidade adquire o programa?

R: Multiplica todos os números da sequência por 8.

2 Extração das componentes (sinal, expoente e mantissa) de um número no formato IEEE 754

Um número representado no formato IEEE 754 (precisão simples) é composto por:



Na figura, **S** é o bit de sinal, **E** é o expoente codificado e **M** a parte fracionária da mantissa.

Nesta primeira fase do trabalho serão desenvolvidas três sub-rotinas cada uma com a função de extrair uma das componentes de um número no formato **IEEE-754**. As rotinas devem ter as seguintes características:

- A sub-rotina **sinal** deverá receber como argumento um número no formato **IEEE-754** e retornar 1 se o número for negativo ou 0 (zero) caso contrário. Por exemplo, para o valor de entrada `0xc20a0000` o valor de retorno será 1.
- A sub-rotina **expoentereal** deverá receber como argumento um número no formato **IEEE-754** e retornar o respetivo valor real do expoente (deve ter em atenção que o expoente está codificado em excesso 127). Por exemplo, para o valor de entrada `0xc20a0000` o valor de retorno será 5.
- A sub-rotina **mantissa** deverá receber como argumento um número no formato **IEEE-754** e retornar a respetiva mantissa (incluindo a parte inteira implícita). Por exemplo, para o valor de entrada `0xc20a0000` o valor de retorno será `1000101000000000000000002`.

Para facilitar a implementação das sub-rotinas anteriores deve recorrer à utilização de máscaras. Para isso, pode inicializar a memória de dados com os seguintes valores:

```
Aux DCD 0x7f800000, 0x007fffff, 0x7fffffff
```

3 Realização de adições e subtrações entre dois números no formato **IEEE-754**

Tirando partido das sub-rotinas desenvolvidas anteriormente, implementar a sub-rotina **soma** que recebe como argumentos 2 números (**NumA** e **NumB**) e retorna o valor da operação **NumA+NumB**.

Para implementar esta rotina pode seguir os seguintes passos:

1. Verificar se um dos argumentos é zero. Caso seja, o resultado será o outro argumento.
2. Calcular diferença de expoentes.
3. Alinhar mantissa do número de menor expoente.
4. Determinar que operação realizar com as mantissas (adição ou subtração) e realizar essa operação.
5. Verificar se é necessário alinhar a mantissa do resultado da operação anterior e corrigir expoente.
6. Montar o resultado no formato **IEEE-754** precisão simples.

Deve testar a sua sub-rotina com (pelo menos) os seguintes valores:

NumA	NumB	NumA+NumB
0x40900000	0xc0700000	0x40300000
0x42040000	0xc0380000	0x41F10000
0xc20a0000	0xc1200000	0xC2320000
0xc0380000	0x42040000	0x41F10000
0x420c0000	0x42040000	0x42880000
0x00000000	0xc0380000	0xC0380000
0x83800000	0x00000000	0x83800000
0x40400000	0xc0380000	0x3F900000

Tirando partido da sub-rotina `soma`, implementar a sub-rotina `subtracao`, que recebe como argumentos 2 números (`NumA` e `NumB`) e retorna o valor da operação `NumA-NumB`.

4 Realização de multiplicações entre dois números no formato IEEE-754

Tirando novamente partido das sub-rotinas desenvolvidas na secção 2, implementar a sub-rotina `multiplica` que recebe como argumentos 2 números (`NumA` e `NumB`) e retorna o valor da operação `NumA×NumB`.

Para implementar esta rotina deverá seguir os seguintes passos:

1. Verificar se um dos argumentos é zero. Caso seja o resultado será zero.
2. Calcular o sinal final.
3. Calcular o expoente final.
4. Multiplicar mantissas.
5. Normalizar o resultado (se necessário).
6. Montar o número final

Como não dispomos de instruções para multiplicar 2 números (que sejam suportadas pela ferramenta `VisUAL`), uma solução possível seria recorrer a um ciclo que iria realizar somas sucessivas. No entanto essa solução não é viável para valores grandes, provocando em alguns casos o encravamento do emulador `VisUAL`. De forma a evitar esses casos deverá criar as seguintes sub-rotinas auxiliares para serem utilizadas no passo 4:

- `calcshiftsbits`: Esta rotina recebe como argumentos 2 mantissas (`MNumA` e `MNumB`) e retorna o número máximo de deslocamentos que se pode aplicar a ambos os números sem perda de informação. Por exemplo, se `MNumA=1000101000000000000000002` e `MNumB=100010110000000000000000002` o resultado será 16. Isto permite que posteriormente a multiplicação possa ser realizada com valores mais pequenos, neste caso `100010102 × 100010112`.

- **peasantmul**: Esta rotina recebe como argumentos duas mantissas (MNumA e MNumB) (já no seu tamanho reduzido) e retorna o resultado da sua multiplicação recorrendo ao algoritmo do *camponês* (https://en.wikipedia.org/wiki/Multiplication_algorithm#Peasant_or_binary_multiplication).

Este algoritmo consiste na divisão sucessiva de MNumA por 2 (**shift right**) e na multiplicação sucessiva do MNumB por 2 (**shift left**). Em cada iteração acumula-se o valor de MNumB se MNumA for ímpar. O algoritmo termina quando o valor de MNumA chegar a 1. Por exemplo, se MNumA=1011₂ e MNumB=11₂ os passos de execução são:

MNumA	MNumB	Acumulador
1011	11	11
101	110	1001
10	1100	1001
1	11000	100001

Testar a sub-rotina com (pelo menos) os seguintes valores:

NumA	NumB	NumA × NumB
0x40900000	0xc0700000	0xC1870000
0x42040000	0xc0380000	0xC2BDC000
0xc20a0000	0xc1200000	0x43AC8000
0xc0380000	0x42040000	0xC2BDC000
0x420c0000	0x42040000	0x44906000
0x00000000	0xc0380000	0x00000000
0x83800000	0x00000000	0x00000000
0x40400000	0xc0380000	0xC10A0000

Fim