ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
Graduação em Engenharia Computação

PCS3732 - Laboratório de Processadores

Professor Jorge Kinoshita

Grupo 10 - Planejamento 2

Arthur Pires da Fonseca NUSP: 10773096

3.1.2 B - Respondam às seguintes perguntas

Sugestão:

- Vejam os códigos de máquinas e observem que certos numeros nao servem para fazer o código de máquina.
- Coloquem no gnuarm e vejam o erro.

```
student:~/src/Lab3$ gcc add_sub.s
add_sub.s: Assembler messages:
add_sub.s:4: Error: invalid constant -- `add R3,R7,#1023'
add_sub.s:5: Error: invalid immediate shift -- `sub R11,R12,R3,LSL#32'
student:~/src/Lab3$ cat add_sub.s
        .text
        .globl  main
main:
        ADD R3, R7, #1023
        SUB R11, R12, R3, LSL #32
```

Erro usando o exemplo da página.

2. Sem usar a instrução MUL, dê as seguintes instruções para multiplicar o registrador R4 por:

a. 132 => 128 + 4 => r0 = r0 * 2^7 + r0 * 2^2

```
@@@@@ mul r0, r0, #132 @@@@@
ldr r0, =0xa
mov r1, r0, lsl #2
add r0, r1, r0, lsl #7
```

b. 255 => 256 - 1 => r0 = r0 * 2^6 - r0

```
@@@@@ mul r0, r0, #255 @@@@@
ldr r0, =0xa
mov r1, r0, lsl #8
sub r0, r1, r0
```

c. 18 => 16 + 2 => r0 = r0 * 2^4 + r0 * 2

```
@@@@@ mul r0, r0, #18 @@@@@
ldr r0, =0xa
mov r1, r0, lsl #4
add r0, r1, r0, lsl #1
```

d. 16384 => 2^{14} => r0 = r0 * 2^{14}

```
@@@@@ mul r0, r0 #16384 @@@@@
ldr r0, =0xa
mov r0, r0, lsl #14
```

3. Escreva uma rotina que compara 2 valores de 64-bits usando somente 2 instruções. (dica: a segunda instrução é condicionalmente executada, baseada no resultado da primeira comparação).

```
@ 0xfedcba98_12345678 == 0x89abcdef_12345678 ??
ldr r0, =0xfedcba98
ldr r1, =0x12345678
ldr r2, =0xfedcba98
ldr r3, =0x12345678
ldr r7, =0x1

subs r5, r3, r1 @ Compara bits menos significativos
subeqs r4, r2, r0 @ Compara bit mais significativos
@subeq r7, r7, r7 @ Se r7 for zero é pq os números são iguais
```

OU

```
cmp r5, r3, r1
cmpeq r4, r2, r0
```

4. Escreva uma rotina que desloque um valor de 64-bits (armazenado em 2 registradores r0 e r1) de um bit para a direita.

```
@ Shift de 1 para a direita
movs r0, r0, lsr #1 @ Shift logico pra direita
movs r1, r1, rrx @ Shift com bit mais significativo puxado do carry
```

5. Idem 4, para a esquerda.

```
@ Shift de 1 para a esquerda
movs r1, r1, lsl #1 @ Shift logico pra esquerda
mov r0, r0, lsl #1  @ Shift logico, mas mantendo o CPSR
addcs r0, r0, #1    @ Adiciona um carry ao mais significativo se teve carry nos menos significativos
```

Shift right e depois left do número 0xa00a_a00a_a00a_a00a

3.1.3 C - prepare a solução de 3.10.7 (individualmente no seu computador)

- **Rascunhe a solução do exercício de divisão 3.10.7; ou seja, como é o algoritmo da divisão. Coloque o algoritmo em código ARM (não é necessário testar). A operação de divisão deve ser feita com shift como faz a profª do primário e não o algoritmo ineficiente e simples que retira um número do outro.**

```
        .text
        .globl  main
main:
        ldr r1, =0x000003FD @ Dividendo (=1021)
        mov r2, #25   @ Divisor (=25)
        mov r3, #0   @ Quociente
        mov r5, #0   @ Resto

        mov r4, #0   @ Tamanho do shift
        mov r0, r2   @ Guarda o valor inicial do divisor
        b align

        @ Divisao
        @ Alinhar bits mais à esquerda

align:
        mov r2, r2, lsl #1
        cmp r2, r1
        ble align    @ r2 (divisor) < r1 (dividendo) => continua

div_loop:
        cmp r1, r2
        bge quociente_1 @ r1 (dividendo) >= r2 (divisor) ?
        b quociente_0
loop_end:
        mov r2, r2, lsr #1 @ Desloca para a direita o divisor
        cmp r1, r0
        blt end @ r1 (dividendo) < r0 (divisor original) => acabou divisão
        b div_loop @ Senao, continua o loop

quociente_1:
        sub r1, r1, r2 @ Tira o divisor do dividendo
        mov r3, r3, lsl #1 @ Desloca o quociente
        add r3, r3, #1 @ Adiciona um 1 ao fim do quociente
        b loop_end

quociente_0:
        mov r3, r3, lsl #1
        b loop_end

end:
        cmp r2, r0
        blt end2          @ Se o divisor voltou ao valor original, nao tem mais o
que fazer
        mov r2, r2, lsr #1 @ Divide divisor por 2
```

```
        mov r3, r3, lsl #1 @ Multiplica o quociente por 2
        b end              @ Continua o loop

end2:
        mov r5, r1 @ Copia o dividendo para o resto
        swi 0x0
```

Veja: http://courses.cs.vt.edu/~cs1104/Division/ShiftSubtract/Shift.Subtract.html e coloque no papel a simulação de 1101 dividido por 10. Existe algum erro nesse algoritmo de divisão? Teste com diversos casos no próprio site antes da aula (porque existe sim um erro).

O erro está em dizer que o quociente já está pronto ao final do loop, o que não é sempre verdade. Em alguns casos, ainda é necessário adicionar alguns zeros ao final do quociente para ajustar o resultado.

Seguindo à risca o algoritmo citado no link, a operação 1000 / 25 dá quociente 5 e resto 0, quando deveria ter dados quociente 40.