PCS3432 - Laboratório de Processadores

Planejamento - E3

Bruno Mariz - 11261826

3.1.2 B - Respondam as seguintes perguntas

1. O que há de errado nas seguintes instruções:

a. ADD r3,r7, #1023

A instrução acima está incorreta pois o último operando de instruções de processamento de dados pode ser:

- um inteiro de 8 bits rotacionado para a direita por um número par de posições, que não é o caso da instrução acima, cujo inteiro precisa de 10 bits para ser representado, e não está rotacionado
- um registrador opcionalmente rotacionado ou com shift por uma constante de 5 bits ou outro registrador, o que também não é o caso

b. SUB r11, r12, r3, LSL #32

A instrução acima está incorreta pois o último operando de instruções de processamento de dados pode ser:

- um inteiro de 8 bits rotacionado para a direita por um número par de posições, que não é o caso da instrução acima.
- um registrador opcionalmente rotacionado ou com shift por uma constante de 5 bits ou outro registrador, o que não ocorre pois a constante do shift possui 6 bits, o que resultaria apenas em zero após o deslocamento do LSL.

2. Sem usar a instrucao MUL, de identifique as instruções para multiplicar o registrador R4 por:

a. 132

```
MOV     r4, #0x4
MOV     r3, r4, LSL #7
ADD     r6, r3, r4, LSL #2
```

Ao fim do programa, o registrador r6 terá o valor de

r4 * 2^7 + r4 * 2^2 =

r4 * 128 + r4 * 4 =

132 * r4,

nesse caso, 528.

b. 255

```
RSB     r6, r4, r4, LSL #8
```

Essa instrução irá subtrair o primeiro operando do segundo após o deslocamento, resultando em r4 * 2^8 - r4, ou r4 * 256 - r4, ou r4 * 255.

c. 18

```
RSB     r3, r4, r4, LSL #2  @ r3 = r4 * 3
ADD     r5, r3, r3, LSL #2  @ r5 = r3 * 5  = r4 * 15
ADD     r4, r5, r3          @ r4 = r5 + r3 = r4 * 15 + r4 * 3 = r4 * 18
```

ou

```
RSB     r3, r4, r4, LSL #2  @ r3 = r4 * 3
RSB     r3, r3, r3, LSL #2  @ r3 = r3 * 3 = r4 * 9
ADD     r6, r3, r3          @ r6 = r3 + r3 = r4 * 18
```

d. 16384

```
MOV     r6, r4, LSL #14
```

Após a instrução, r6 = r4 _ 2^14 = r4 _ 16384.

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).

No cap. 5 da apostila existe uma tabela com todas as condições possíveis. Exemplo de instrução condicional: ADDEQ R0,R1,R2 = execute a soma se a flag Z no CPSR == 1. Alguns alunos reclamaram ser necessário consultar o cap. 5, mas não se consegue fazer o cap. 3 sem uma pequena introdução as instruções condicionais.

Considerando o primeiro valor de 64 bits estando nos tegistradores r1 e r2, e o segundo valor nos registradores r3 e r4, pode-se compará-los utilizando as seguintes instruções:

```

```

```
@ EQ
CMP     r1, r3
CMPEQ   r2, r4

@ GT
CMP     r1, r3
CMPGT   r2, r4

@ LT
CMP     r1, r3
CMPLT   r2, r4
```

Dessa forma, se compara primeiro os 32 bits mais significativos de cada valor, e caso as flags correspondentes a cada comparação estejam ativas, se compara os 32 bits menos significativos.

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 o Professor do Ensino Fundamental e não o algoritmo ineficiente e simples que subtrai um número do outro.

dividendo: r1

divisor: r2

quociente: r3

resto: r5

Algoritmo:

1. alinhar bits mais significativos do divisor (r2) aos bits mais significativos do dividendo (r1)
2. subtrair divisor (r1) dos bits correspondentes do dividendo a. se a subtração resultou em um número negativo, adicionar 0 no bit menos significativo do quociente (r3) b. caso contrário, adicionar 1 ao bit menos significativo do quociente e guardar resultado da subtração no registrador de resto (r5)
3. adicionar próximo bit do dividendo ao bit menos significativo do resultado da subtração (registrador de resto r5)
4. se o resto r5 for maior que o divisor, voltar para passo 2, caso contrario encerrar

```
@ r7: tmp
@ r1: dividendo
@ r2: divisor
@ r3: quociente
@ r5: resto

alinhar_bits:
    @ r5 contendo msb alinhados com r2
```

```
loop:
    SUBS r7, r5, r2
    BLT resto_lt_divisor

resto_ge_divisor:
    @ adiciona 1 ao quociente
    LSL r3, #1
    ADD r3, #1
    @ salva resultado da subtracao no registrador do resto
    MOV r5, r7
    B adicionar_proximo_bit

resto_lt_divisor:
    LSL r3, #1 @ adiciona 0 ao quociente
    B adicionar_proximo_bit

adicionar_proximo_bit:
    LSL r5, #1 @ "abre espaco" para novo bit
    ADD r5, <proximo_bit_do_dividendo>

condicao_do_loop:
    CMP r5, r2
    BGE loop

fim:
    B pc
```