



Planejamento - Experiência 3

Data Processing Operations

Disciplina : PCS3432 - Laboratório de Processadores
Prof: Jorge Kinoshita
Membros:
José Otávio Brochado Colombini - 9795060
Filipe Penna Cerávolo Soares - 10774009

Planejamento	3
O que há de errado nas seguintes instruções:	3
Sem usar a instrução MUL, de as seguintes instruções para multiplicar o registrador R4 por:	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).	5
Escreva uma rotina que desloque um valor de 64-bits (armazenado em 2 registradores r0 e r1) de um bit para a direita	5
Idem 4, para a esquerda.	6
Solução de 3.10.7 (individualmente no seu computador)	7

1. Planejamento

a. O que há de errado nas seguintes instruções:

```
ADD r3,r7, #1023
SUB r11, r12, r3, LSL #32
```

O código não pode ser compilado pois as duas constantes são maiores que o permitido, a constante máxima permitida para o ADD devem estar contida em 8 bits ([0, 255]) e a constante para o LSL deve estar contida em 5 bits ([0, 31]) pela arquitetura ARMv7 utilizada na placa do laboratório.

Segue print da resposta do compilador para o comando:

```
student:~/src$ gcc item-3-1-1.s
item-3-1-1.s: Assembler messages:
item-3-1-1.s:0: Warning: end of file not at end of a line; newline inserted
item-3-1-1.s:17: Error: invalid constant -- `add r3,r7,#1023'
item-3-1-1.s:18: Error: invalid immediate shift -- `sub r11,r12,r3,LSL#32'
```

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

1. 132
2. 255
3. 18
4. 16384

```
main:
    @Multiplicacao sem utilizar o MUL pelos valores indicados nas
    funcoes
    BL fun132
    BL fun255
    BL fun18
```

```

BL    fun16384
SWI   0x123456

fun132:
    LDR r4, =0x1
    MOV r0, r4, LSL #7
    ADD r4, r0, r4, LSL #2
    MOV pc, lr

fun255:
    LDR r4, =0x1
    MOV r0, r4, LSL #8
    SUB r4, r0, r4
    MOV pc, lr

fun18:
    LDR r4, =0x1
    MOV r0, r4, LSL #4
    ADD r4, r0, r4, LSL #1
    MOV pc, lr

fun16384:
    LDR r4, =0x1
    MOV r4, r4, LSL #14
    MOV pc, lr

```

Segue print dos resultados obtidos:

Ao final de fun132:

Register group: general			
r0	0x80	128	
r2	0xffffffff	-1	
r4	0x84	132	

Ao final de fun225:

Register group: general			
r0	0x100	256	
r2	0xffffffff	-1	
r4	0xff	255	

Ao final de fun18:

Register group: general			
r0	0x10	16	
r2	0xffffffff	-1	
r4	0x12	18	

Ao final de fun16384:

Register group: general			
r0	0x10	16	
r2	0xffffffff	-1	
r4	0x4000	16384	
r6	0x0	0	

c. 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).

```
SUBS r0, r2, r4
SUBEQS r1, r3, r5
```

O comando 'SUBS' seta a flag 'Z', caso $r2 = r4$. O comando de baixo, por sua vez, é apenas executado caso a flag 'Z' esteja setado. Ou seja, apenas de $r2 = r4$.

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

```
LDR r0, =0xFFFFFFFF
LDR r1, =0xFFFFFFFF

MOVS r0, r0, LSR #1
MOV r1, r1, RRX
SWI 0x123456
```

Considerando que $r2 = r0 \& r1$ que corresponde ao valor de 64 bits, a utilização do RRX permite com que o dado retirado pelo LSR do registrador r0 seja inserido no registrador r1, pois ele fica salvo na flag Carry no CPSR (MOVS é utilizado para atualizar a flag C).

Segue captura de tela representando o valor obtido com essa rotina.

```
Register group: general
r0 0x7f 127 r1 0x80000000 -2147483
r2 0xffffffff -1 r3 0xa9b8 43448
r4 0x1 1 r5 0x1ffff8 2097144
r6 0x0 0 r7 0x0 0
r8 0x0 0 r9 0x0 0
r10 0x200100 2097408 r11 0x0 0
r12 0x1ffffc 2097100 sp 0x1ffff8 2097144
lr 0x81fc 33276 pc 0x8228 33320
fps 0x0 0 cpsr 0x2000013 53687093
```

```
0x8218 <main> mov r0, #255 ; 0xff
0x821c <main+4> mov r1, #0 ; 0x0
0x8220 <main+8> movs r0, r0, lsr #1
0x8224 <main+12> mov r1, r1, rrx
> 0x8228 <main+16> swi 0x00123456
0x822c <atexit> mov r12, sp
0x8230 <atexit+4> stmbd sp!, {r4, r5, r11, r12, lr, pc}
0x8234 <atexit+8> ldr r5, [pc, #120] ; 0x82bc <$d>
0x8238 <atexit+12> ldr r3, [r5]
0x823c <atexit+16> ldr r1, [r3, #328]
0x8240 <atexit+20> cmp r1, #0 ; 0x0
```

e. Idem 4, para a esquerda.

```
MOVS r1, r1, LSL #1
MOV r0, r0, ROR #31
MOV r0, r0, RRX
MOV r0, r0, ROR #31
SWI 0x123456
```

Novamente, considerando $r2 = r0 \& r1$, utiliza-se o ROR para inverter o registrador e assim inserir o valor do carry, permitido apenas pelo RRX no bit mais significativo. Então para finalizar, o registrador é desinvertido com o ROR novamente.

Segue captura de tela representando o valor obtido com essa rotina.

```
Register group: general
r0 0x1ff 511 r1 0xffffffff -2
r2 0xffffffff -1 r3 0xa9bc 43452
r4 0x1 1 r5 0x1ffff8 2097144
r6 0x0 0 r7 0x0 0
r8 0x0 0 r9 0x0 0
r10 0x200100 2097408 r11 0x0 0
r12 0x1ffff8 2097144 sp 0x1ffff8 2097120
lr 0x81fc 33276 pc 0x8238 33336
fps 0x0 0 cpsr 0xa000013 -1610612
```

```
0x8218 <main> mov r0, #255 ; 0xff
0x821c <main+4> mvn r1, #0 ; 0x0
0x8220 <main+8> movs r1, r1, lsl #1
0x8224 <main+12> mov r0, r0, ror #31
0x8228 <main+16> mov r0, r0, rrx
0x822c <main+20> mov r0, r0, ror #31
0x8230 <atexit> mov r12, sp
```

f. Solução de 3.10.7 (individualmente no seu computador)

```
LDR r0, =32    @ Dividendo
LDR r1, =8     @ Divisor
LDR r2, =0x0   @ Quociente
LDR r3, =0x0   @ Resto

@Elementos auxiliares
LDR r4, =0x80000000 @Identificador de primeiro bit significativo de
r0 e r1
LDR r5, =0x80000000 @Identificador de primeiro bit significativo de
r0 e r1
LDR r6, =0x1F     @Contador de bit mais significativo de r0
LDR r7, =0x1F     @Contador de bit mais significativo de r1

BL shiftdividend
BL opdivision

MOV r3, r0
SWI 0x123456

shiftdividend:
@Primeiro verificamos o bit do dividendo
ANDS    r8, r4, r0
BNE     shiftdivisor
MOV     r4, r4, LSR #1
SUBS    r6, r6, #1
B       shiftdividend
shiftdivisor:
@Depois verificamos o do divisor
ANDS    r8, r5, r1
MOVNE   pc, lr
MOV     r5, r5, LSR #1
SUBS    r7, r7, #1
B       shiftdivisor

opdivision:
SUBS    r8, r6, r7
operation:
MOVPL   r2, r2, LSL #1
MOVMI   pc, lr
CMP     r0, r1, LSL r8
SUBPL   r0, r0, r1, LSL r8
```

```
ADDPL    r2, r2, #1
SUBS     r8, r8, #1
B operation
```

As funções ‘shift dividend’ e ‘shift divisor’ são responsáveis por encontrarem o número de dígitos (em binário) do dividendo e divisor respectivamente.

Com esses valores, ‘opdivision’ executa subtrações com base em shifts para o cálculo do valor em binário para cada uma das casas decimais do quociente.