

ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO



Experiência 3

Relatório

Turma: 02

11/04/2023

Nome	Número USP
Natanael Magalhães Cardoso	8914122
Tomas Gorescu Caldeira	9300881

Exercício 3.10.5

Enunciado: Assume that a signed long multiplication instruction is not available. Write a program that performs long multiplications, producing 64 bits of result. Use only the UMULL instruction and logical operations such as MVN to invert, XOR, and ORR. Run the program using the two operands -2 and -4 to verify.

O programa abaixo faz a multiplicação com sinal de duas palavras de 64-bits e armazena o resultado nos registradores R7 (mais significativo) e R6 (menos significativo)

A ideia é fazer a divisão longa sem sinal, verificar e corrigir o sinal posteriormente.

```
@ Descrição do algoritmo
@ -----
@ Implementação da multiplicação longa com signal
@ a partir da instrução UMULL
@
@
@ Descrição dos Registradores
@ -----
@ R0 - Primeiro Operando
@ R1 - Segundo Operando
@ R7 - Resultado (32-bits mais significativos)
@ R6 - Resultado (32-bits menos significativos)
@ R2, R3, R4 - Registradores de Trabalho
@
@
@ Instruções de uso
@ -----
@ arm build ex_3_10_5.s (montagem)
@ arm debug (depuração)

.text
.globl main

main:
    LDR r0, = -4        @ operando 1
    LDR r1, = -2        @ operando 2
```

preparingRegisters:

```
MOV r4, #0           @ r4 recebe 0
MOVS r2, r0           @ r2 recebe operando 1 com atualização de
flags
RSBMI r2, r0, #0      @ r2 = modulo(operando 1) - (r2 = 0 -r2 se r2
é negativo)
MOVMI r4, #1          @ se r2 é negativo marca r4 como 1
MOVS r3, r1           @ r3 recebe operando 2 com atualização de
flags
EORMI r4, r4, #1      @ se operando 2 também é negativo, r4 fica 0
(1 XOR 1 ou 0 XOR 0). Se o sinal é diferente r4 = 1
RSBMI r3, r1, #0      @ r3 = modulo(operando 2) -> (r3 = 0 -r3 se r3
é negativo)
```

multiplication:

```
UMULL r6, r7, r2, r3 @ multiplicação de r2 por r3
CMP r4, #0           @ atualiza o cpsr da flag que setamos em r4
BNE invert           @ se r4 != 0
B fim                @ sai do loop
```

invert:

```
MVN r7, r7           @ troca o sinal do mais significativo
MVN r6, r6           @ troca o sinal do menos significativo
ADDS r6, r6, #1       @ adiciona 1 pela correção da inversão na
troca do sinal
ADC r7, r7, #0        @ adiciona o carry ao r7
```

fim:

```
SWI 0x1234           @ fim do programa
```

Abaixo a imagem da depuração do programa implementado

```
Activities Terminal abr 11 09:33 tomas@tomas-Latitude-3420: ~/LabDig/E3

Register group: general
r0 0xfffffffffc -4 r1 0xfffffffffe -2 r2 0x4 4
r3 0x2 2 r4 0x0 0 r5 0x1 1
r6 0x8 8 r7 0x0 0 r8 0x103ec 66540
r9 0x3fffed78 1073737080 r10 0x0 0 r11 0x2ff14 196372
r12 0x3ffa0000 1073348608 sp 0x407ffee0 0x407ffee0 lr 0x3fe2e3f8 1071834104
pc 0x10430 0x10430 <fin> cpsr 0x60000010 1610612752 fpscr 0x0 0
fpsid 0x410430f0 1090793712 fpexc 0x40000000 1073741824 AFSR0_EL1 0x0 0
AFSR1_EL1 0x0 0 DBGDIOR 0x3515f021 890630177 DBGDSAR 0x0 0
DBGBVR 0x0 0 DBGBCR 0x0 0 DBGWVR 0x0 0
DBGWCR 0x0 0 PAR 0x0 0 DBGBVR 0x0 0

23 MVN r6, r6 @ troca o sinal do menos significativo
24 ADDS r6, r6, #1 @ adiciona 1 pela correção da inversão na troca do sinal
25 ADC r7, r7, #0 @ adiciona o carry ao r7
26 fin:
B+> 27 SWI 0x1234 @ resultado em r7 - r6
28
29
30
31
32
33

remote Thread 1.16597 In: fin L27 PC: 0x10430
(gdb) b fin
Breakpoint 1 at 0x10430: file item-3.10.5.s, line 27.
(gdb) c
Continuing.

Breakpoint 1, fin () at item-3.10.5.s:27
(gdb) 
```

Exercício 3.10.6

Enunciado: Write ARM assembly to perform the function of absolute value. Register r0 contains the initial value, and r1 contains the absolute value. Try to use only two instructions, not counting the SWI to terminate the program.

Este programa compara o valor armazenado no registrador R0 e efetua a operação $R0 = 0 - R0$ apenas se o valor for menor que zero.

```
@ Descrição do algoritmo
@ -----
@ Algoritmo de obtenção do valor absoluto de um
@ número armazenado no registrador R0
@
@
@ Descrição dos Registradores
@ -----
@ R0 - Número a ser calculado o valor absoluto
@
@
@ Instruções de uso
@ -----
@ arm build ex_3_10_6.s (montagem)
@ arm debug (depuração)

.text
.global main

main:
    LDR R0, =-360    @ Carrega operando

    CMP R0, #0       @ Compara R0 com 0
    RSBLT R0, R0, #0  @ Efetua R0 = 0 - R0 condicionalmente,
                     @ se R0 < 0

    LDR R7, =0x1      @ exit(0)
    SWI 0x0           @ Termina o programa
```

As Figuras abaixo mostram o valor do registrador R0 antes e depois da operação valor absoluto

```
natan@asus: ~/repos/poli/lab-processadores/exp03

Register group: general
r0      0xfffffe98      -360      r1      0x407ffe64      1082130020
r2      0x407ffe6c      1082130028      r3      0x103ec        66540
r4      0x3ff90000      1073283072      r5      0x1           1
r6      0x2ff14         196372          r7      0x3ffff048     1073737800
r8      0x103ec         66540           r9      0x3fffed78     1073737080
r10     0x0             0               r11     0x2ff14         196372
r12     0x3ff90000      1073283072      sp      0x407ffcfc0     0x407ffcfc0
lr      0x3fe1e3f8      1071768568      pc      0x103f0        0x103f0 <main+4>
cpsr    0x60000010      1610612752      fpscr   0x0           0

ex_3_10_6.s
B+ 21      LDR R0, --360      @ Carrega operando
    22
    > 23      CMP R0, #0      @ Compara R0 com 0
    24      RSBLT R0, R0, #0  @ Efetua R0 = 0 - R0 condicionalmente,
    25      @ se R0 < 0
    26
    27      LDR R7, =0x1      @ exit(0)
    28      SWI 0x0           @ Termina o programa
    29

remote Thread 1.19693 In: main                                L23  PC: 0x103f0
(gdb) b main
Ponto de parada 1 at 0x103ec: file ex_3_10_6.s, line 21.
(gdb) c
Continuing.

Breakpoint 1, main () at ex_3_10_6.s:21
(gdb) s
(gdb) □
```

```
natan@asus: ~/repos/poli/lab-processadores/exp03

Register group: general
r0      0x168           360      r1      0x407ffe64      1082130020
r2      0x407ffe6c      1082130028      r3      0x103ec        66540
r4      0x3ff90000      1073283072      r5      0x1           1
r6      0x2ff14         196372          r7      0x3ffff048     1073737800
r8      0x103ec         66540           r9      0x3fffed78     1073737080
r10     0x0             0               r11     0x2ff14         196372
r12     0x3ff90000      1073283072      sp      0x407ffcfc0     0x407ffcfc0
lr      0x3fe1e3f8      1071768568      pc      0x103f8        0x103f8 <main+12>
cpsr    0xa0000010      -1610612720     fpscr   0x0           0

ex_3_10_6.s
B+ 21      LDR R0, --360      @ Carrega operando
    22
    23      CMP R0, #0      @ Compara R0 com 0
    24      RSBLT R0, R0, #0  @ Efetua R0 = 0 - R0 condicionalmente,
    25      @ se R0 < 0
    26
    > 27      LDR R7, =0x1      @ exit(0)
    28      SWI 0x0           @ Termina o programa
    29

remote Thread 1.19693 In: main                                L27  PC: 0x103f8
(gdb) b main
Ponto de parada 1 at 0x103ec: file ex_3_10_6.s, line 21.
(gdb) c
Continuing.

Breakpoint 1, main () at ex_3_10_6.s:21
(gdb) s
(gdb) s
(gdb) s
(gdb) □
```

Exercício 3.10.7

Enunciado: Write ARM assembly to perform the function of division. Registers r1 and r2 contain the dividend and divisor, r3 contains the quotient, and r5 contains the remainder. For this operation, you can either use a single shift-subtract algorithm or another more complicated one.

O programa abaixo faz a divisão de um número armazenado no registrador R1 por outro armazenado no registrador R2 e coloca o quociente no registrador R3 e o resto no registrador R5.

O programa funciona descolando o divisor para esquerda até que seu valor seja o máximo possível menor que o dividendo. Então, começa o loop de multiplicação com tantas iterações quantas ocorreram na etapa de alinhamento.

A cada etapa, verifica-se se o dividendo é maior que $2^n \cdot \text{divisor}$, sendo n o número de shifts a esquerda. Se o dividendo for maior, subtrai-se $2^n \cdot \text{divisor}$ do dividendo, adiciona-se 2^n ao quociente e volta-se a decrementar n, até que n seja 0.

O que resta das consecutivas subtrações no dividendo é o resto, é o quociente é o resultado das somas ao longo da iteração.

```
@ Descrição do algoritmo
@ -----
@ Algoritmo de divisão inteira usando subtrações
@ sucessivas
@
@
@ Descrição dos Registradores
@ -----
@ R1 - Dividendo (Numerador)
@ R2 - Divisor (Denominador)
@ R3 - Quociente
@ R5 - Resto
@
@
@ Instruções de uso
@ -----
@ arm build ex_3_10_7.s (montagem)
@ arm debug (depuração)
```

.text

```

.globl main
main:
    LDR r1, =8914122    @ inicializa dividendo
    LDR r2, =1000        @ inicializa divisor
    MOV r3, #0           @ inicializa quociente
    MOV r5, #0           @ inicializa resto
    MOV r4, r2           @ copia de divisor
    MOV r9, r1           @ copia do dividendo
    MOV r7, #0           @ inicializa contador
    MOV r6, #1           @ constante = 1
alinhamento:
    MOV r4, r4, LSL #1   @ divisor shiftado para esquerda
    ADD r7, r7, #1       @ incrementa contador de shifts
    CMP r4, r9           @ verifica se o divisor é maior que o
dividendo
    BLT alinhamento     @ volta para mais um shift left
    MOV r4, r4, LSR #1   @ garantia que ao final do loop da pra
dividir um pelo outro
    SUB r7, r7, #1       @ decrementa o contador após o shift right
loop:
    CMP r9, r4           @ compara divisor e dividendo
    SUBGE r9, r9, r4     @ subtrai divisor de dividendo
    ADDGE r3, r3, r6, LSL r7 @ quociente recebe resto 1 shiftado do
numero de alinhamentos
    SUB r7, r7, #1       @ subtrai 1 do contador
    MOV r4, r4, LSR #1   @ divisor shiftado para esquerda n
vezes
    CMP r7, #0           @ compara contador com 0
    BPL loop            @ continua se positivo ou 0
    MOV r5, r9           @ r5 recebe o resto
fim:
    SWI 0x0             @ fim do programa

```

As Figuras abaixo mostram a depuração do programa para as três operações indicadas.


```

natan@asus: ~/repos/poli/lab-processadores/exp03

Register group: general
r0      0x1      1      r1      0x12d687    1234567
r2      0x4d2    1234    r3      0x3e8     1000
r4      0x269    617     r5      0x237     567
r6      0x1      1      r7      0xffffffff -1
r8      0x103ec  66540   r9      0x237     567
r10     0x0      0      r11     0x2ff14   196372
r12     0x3ff90000 1073283072 sp      0x407ffc0    0x407ffc0
lr      0x3fe1e3f8 1071768568 pc      0x10444   0x10444 <fim>
cpsr    0xa000010 -1610612720 fpscr   0x0      0

ex_3_10_7.s
44      CMP r7, #0      @ compara contador com 0
45      BPL loop        @ continua se positivo ou 0
46      MOV r5, r9       @ r5 recebe o resto
47      fim:
B+> 48      SWI 0x0      @ fim do programa
49
50
51
52
53

remote Thread 1.23727 In: fim                                L48    PC: 0x10444
(gdb) c
Continuing.

Breakpoint 1, main () at ex_3_10_7.s:23
(gdb) b fim
Ponto de parada 2 at 0x10444: file ex_3_10_7.s, line 48.
(gdb) c
Continuing.

Breakpoint 2, fim () at ex_3_10_7.s:48
(gdb) 

```

$$1234567/1000 = (1234 * 1000) + 567$$

```

natan@asus: ~/repos/poli/lab-processadores/exp03

Register group: general
r0      0x1      1      r1      0x75bcd15   123456789
r2      0x4d2    1234    r3      0x186ce    100046
r4      0x269    617     r5      0x19       25
r6      0x1      1      r7      0xffffffff -1
r8      0x103ec  66540   r9      0x19       25
r10     0x0      0      r11     0x2ff14   196372
r12     0x3ff90000 1073283072 sp      0x407ffc0    0x407ffc0
lr      0x3fe1e3f8 1071768568 pc      0x10444   0x10444 <fim>
cpsr    0xa000010 -1610612720 fpscr   0x0      0

ex_3_10_7.s
44      CMP r7, #0      @ compara contador com 0
45      BPL loop        @ continua se positivo ou 0
46      MOV r5, r9       @ r5 recebe o resto
47      fim:
B+> 48      SWI 0x0      @ fim do programa
49
50
51
52
53

remote Thread 1.23952 In: fim                                L48    PC: 0x10444
(gdb) c
Continuing.

Breakpoint 1, main () at ex_3_10_7.s:23
(gdb) b fim
Ponto de parada 2 at 0x10444: file ex_3_10_7.s, line 48.
(gdb) c
Continuing.

Breakpoint 2, fim () at ex_3_10_7.s:48
(gdb) 

```

$$123456789 = (100046 * 1234) + 25$$

```
natan@asus: ~/repos/poli/lab-processadores/exp03

Register group: general
r0      0x1      1      r1      0x64      100
r2      0xa     10      r3      0xa     10
r4      0x5     5       r5      0x0     0
r6      0x1     1       r7      0xffffffff -1
r8      0x103ec 66540   r9      0x0     0
r10     0x0     0       r11     0x2ff14 196372
r12     0x3ff90000 1073283072 sp      0x407ffc0 0x407ffc0
lr      0x3fe1e3f8 1071768568 pc      0x10444 0x10444 <fim>
cpsr    0xa000010 -1610612720 fpscr   0x0     0

ex_3_10_7.s
44      CMP r7, #0      @ compara contador com 0
45      BPL loop      @ continua se positivo ou 0
46      MOV r5, r9      @ r5 recebe o resto
47      fim:
B+> 48      SWI 0x0      @ fim do programa
49
50
51
52
53

remote Thread 1.24073 In: fim L48 PC: 0x10444
(gdb) c
Continuing.

Breakpoint 1, main () at ex_3_10_7.s:23
(gdb) b fim
Ponto de parada 2 at 0x10444: file ex_3_10_7.s, line 48.
(gdb) c
Continuing.

Breakpoint 2, fim () at ex_3_10_7.s:48
(gdb) 
```

$$100 = 10 * 10 + 0$$

Exercício 3.10.8

Enunciado: A Gray code is an ordering of 2^n binary numbers such that only one bit changes from one entry to the next. One example of a 2-bit Gray code is b10 11 01 00. The spaces in this example are for readability. Write ARM assembly to turn a 2-bit Gray code held in r1 into a 3-bit Gray code in r2.

Para a solução deste exercício, usamos o algoritmo demonstrado em aula, que difere um pouco do utilizado no livro texto. Para um código de gray de n bits, podemos obter o código de gray de $n+1$ fazendo:

- Duplicamos a sequência, invertendo a ordem
- Na primeira parte, adicionamos um 0 como pré-fixo
- Na segunda parte, adicionamos um 1 como pré fixo

A sequência dos passos para gray de 2 para 3 bits, ficaria:

- Partimos da sequência 00 01 11 10
- Duplicamos a sequência: 00 01 11 10 || 10 11 01 00
- Adicionamos os 0's na primeira sequência: 000 001 011 010 || 10 11 01 00
- Adicionamos os 1's na segunda sequência: 000 001 011 010 || 110 111 101 100

No código abaixo seguimos o algoritmo descrito. Nós usamos os Shift right para isolar cada um dos segmentos da sequência de 2 bits. Também é importante ressaltar que para poder isolar os fragmentos que não são “extremos”, precisamos ter um cópia de r1 da qual vamos subtraindo os pares pelos quais já passamos (r3).

```
@ Descrição do algoritmo
@ -----
@ Cálculo do Código de Gray de 3 bits a partir do
@ Código de Gray de 2 bits
@
@
@ Instruções de uso
@ -----
@ arm build ex_3_10_8.s (montagem)
@ arm debug (depuração)

.text
.globl main

main:
    LDR r1, =0b00011110 @ r1 recebe código a ser transformado
    MOV r0, #8           @ r0 recebe o tamanho da palavra de r1
```

```

MOV r3, r1          @ r3 como uma cópia de r1
MOV r2, #0          @ resultado
MOV r6, #1          @ constante = 1

loop:
    SUB r0, r0, #2    @ subtrai a ordem do código de gray do
tamanho da palavra
    MOV r4, r3, LSR r0 @ r4 recebe o par da vez
    ADD r2, r2, r4     @ incrementa r2 com o par da vez
    SUB r3, r3, r4, LSL r0 @ remove-se de r3 o bit adicionado a r2
    MOV r2, r2, LSL #3 @ add 0 à direita em r2 e abre 2 bits de
espaço p/ soma
    CMP r0, #0        @ Para quando r0 = 0
    BNE loop          @ continua loop se r0 != 0
    MOV r2, r2, LSR #3 @ Remove os bits adicionados a mais

lista_inversa:
    LDR r5, =0b10110100 @ r1 recebe código inverso de r1
transformado
    MOV r0, #8          @ r0 recebe o tamanho da palavra de r1
    MOV r3, r5          @ move r5 para registrador de trabalho

loop_inverso:
    ADD r2, r6, r2, LSL #1 @ adiciona um 1 a direita em r2
    MOV r2, r2, LSL #2    @ adiciona 2 espaços para soma em r2
    SUB r0, r0, #2        @ subtrai a ordem do código de gray do
tamanho da palavra
    MOV r4, r3, LSR r0    @ r4 recebe o par da vez
    ADD r2, r2, r4        @ incrementa r2 com o par da vez
    SUB r3, r3, r4, LSL r0 @ remove-se de r3 o bit adicionado a r2
    CMP r0, #0            @ Para quando r0 = 0
    BNE loop_inverso     @ continua loop se r0 != 0

fim:
    MOV r7, #0x1         @ exit(0)

```

```
SWI 0x0 @ system call
```

A Figura abaixo mostra a depuração do algoritmo implementado

The screenshot shows a GDB terminal window with the following content:

```
Register group: general
r0      0x0      0      r1      0x1e     30
r2      0x5adec  372204  r3      0x0     0
r4      0x0      0      r5      0xb4     180
r6      0x1      1      r7      0x3ffff048 1073737800
r8      0x103ec  66540  r9      0x3fffd78  1073737080
r10     0x0      0      r11     0x2ff14   196372
r12     0x3ffa0000 1073348608  sp      0x407ffee0  0x407ffee0
lr      0x3fe2e3f8 1071834104  pc      0x1044c   0x1044c <fin>
cpsr    0x60000010 1610612752  fpscr   0x0     0
fpsidr  0x410430f0 1090793712  fpexc   0x40000000 1073741824

item-3.10.8.s
26      MOV     r2, r2, LSL #2      @ adiciona 2 espaços para soma em r2
27      SUB     r0, r0, #2         @ subtrai a ordem do código de gray do tamanho da palavra
28      MOV     r4, r3, LSR r0     @ r4 recebe o par da vez
29      ADD     r2, r2, r4         @ incrementa r2 com o par da vez
30      SUB     r3, r3, r4, LSL r0 @ remove se de r3 o bar adicionado a r2
31      CMP     r0, #0            @ Para quando r0 = 0
32      BNE     loop_inverso
33      fin:
34      MOV     r0, #0x0
35      MOV     r7, #0x1          @ exit(0)
36      SWI     0x0              @ system call

remote Thread 1.16129 In: fin L34 PC: 0x1044c
Breakpoint 1 at 0x103ec: file item-3.10.8.s, line 6.
(gdb) c
Continuing.

Breakpoint 1, main () at item-3.10.8.s:6
(gdb) s
loop () at item-3.10.8.s:12
lista_inversa () at item-3.10.8.s:21
loop_inverso () at item-3.10.8.s:25
fin () at item-3.10.8.s:34
(gdb) p/t $r2
$1 = 1011010110111101100
(gdb)
```

Como o comando p/t não mostra os bits da direita, temos que saber que a sequência é “liderada” por 0’s, formando:

```
000 001 011 010 110 111 101 100
```

que é o código de Gray de 3 bits que esperávamos formar.