



Relatório - Experiência 3

Data Processing Operations

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

Introdução	3
Objetivo	3
Planejamento	3
Relatório	3
3.10.5 64 bits signed multiplication	3
3.10.6 Absolute value	5
3.10.7 Division	6
3.10.8 Gray codes	7

1.Introdução

Neste experimento estudou-se as operações e manipulações de dados não vistas na experiência 2: desvio condicional, multiplicação com sinal, código de gray, divisão e valores absolutos .

2.Objetivo

Finalizar os estudos do capítulo 3 do livro.

3.Planejamento

Disponível no documento de Planejamento da Experiência.

4.Relatório

a. 3.10.5 64 bits signed multiplication

```
.text
.globl main
main:
    LDR    r0, =0xFF0F00F
    LDR    r1, =0x00000040
    LDR    r2, =0x0
    LDR    r3, =0x0
    LDR    r6, =0xFFFFFFFF

    @SMULL    r9, r10, r0, r1

    CMP    r0, #0
```

```

EORMI    r4, r0, r6
ADDMI    r4, r4, #1
MOVPL    r4, r0
CMP      r1, #0
EORMI    r5, r1, r6
ADDMI    r5, r5, #1
MOVPL    r5, r1

UMULL    r3, r2, r5, r4

CMP      r0, #0
BMI      funcA
BPL      funcB

```

```

result:
    SWI 0x123456

```

```

funcA:
    CMP    r1, #0
    BMI    result
    EOR     r3, r3, r6
    ADDS    r3, r3, #1
    EOR     r2, r2, r6
    ADD     r2, r2, #1
    SUBCC   r2, r2, #1
    B      result

```

```

funcB:
    CMP    r1, #0
    BPL    result
    EOR     r3, r3, r6
    ADDS    r3, r3, #1
    EOR     r2, r2, r6
    ADD     r2, r2, #1
    SUBCC   r2, r2, #1
    B      result

```

Neste algoritmo realizamos o complemento de 2 em caso dos operadores serem negativos, por fim realizamos a multiplicação sem sinal e verificamos qual seria o sinal do resultado, por fim realizando um complemento de 2 caso necessário nele (atento à necessidade de carry para o registrador de bits mais significativos).

Na imagem a seguir temos salvo em {r10, r9} a multiplicação utilizando SMULL como controle e em {r2, r3} a multiplicação pelo algoritmo, para comparação:

```

Register group: general
r0      0xffffffff -1
r2      0xffffffff -1
r4      0x1        1
r6      0xffffffff -1
r8      0x0        0
r10     0xffffffff -1
r12     0x1fffcc 2097100
lr      0x81fc 33276
fps     0x0        0
r1      0xff0f00f 267448335
r3      0xf00f0ff1 -267448335
r5      0xff0f00f 267448335
r7      0x0        0
r9      0xf00f0ff1 -267448335
r11     0x0        0
sp      0x1ffff8 2097144
pc      0x8260 33376
cpsr    0x80000013 -2147483629

e3106.s
30      BPL      funcB
31
32      result:
> 33      SWI 0x123456
34
35
36      funcA:
37          CMP    r1, #0
38          BMI    result
39          EOR    r3, r3, r6
40          ADDS   r3, r3, #1
41          EOR    r2, r2, r6
42          ADD    r2, r2, #1
43          SUBCC  r2, r2, #1
44          B      result
45
46      funcB:

```

b. 3.10.6 Absolute value

```

LDR r0, =-32    @ Valor signed
LDR r1, =0      @ Valor absoluto
LDR r2, =0

SUBS r1, r0, r2 @ Caso o valor seja positivo
SUBMI r1, r2, r0 @ Caso o valor seja negativo
SWI 0x123456

```

O comando SUBS atualiza a flag de operação negativa, caso r0 seja um número negativo e nesse caso o valor de r1 é sobrescrito com o oposto de r0. Na situação em que r0 é um número positivo, a flag MI não é ativada e r1 recebe o valor de r0. Em qualquer um dos casos, r1 recebe o valor absoluto de r0.

Segue print do valor dos registradores para r0 = -32:

```

Register group: general
r0      0xfffffffffe0      -32
r2      0x0      0
r4      0x1      1
r6      0x0      0
r8      0x0      0
r10     0x200100 2097408
r12     0x1fffcc 2097100
lr      0x81fc 33276
fps     0x0      0
r1      0x20      32
r3      0xa9bc 43452
r5      0x1ffff8 2097144
r7      0x0      0
r9      0x0      0
r11     0x0      0
sp      0x1ffff8 2097144
pc      0x822c 33324
cpsr    0xa0000013 -1610612

B+ 17      LDR r0, =-32 @ Valor signed
    18      LDR r1, =0 @ Valor absoluto
    19      LDR r2, =0
    20
    21      SUBS r1, r0, r2 @ Caso o valor seja positivo
    22      SUBMI r1, r2, r0 @ Caso o valor seja negativo
> 23      SWI 0x123456

```

c. 3.10.7 Division

Utilizamos o código desenvolvido na preparação da experiência. Dessa forma, obtivemos o seguinte código para o algoritmo:

```

.text
.globl main
main:
    LDR r0, =1234567 @ Dividendo
    LDR r1, =1234 @ 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

```

d. 3.10.8 Gray codes

```

.text
.globl main
main:
    LDR r0, =0xB4 @2bitGrayCode
    MOV r1, r0
    LDR r3, =0x0
    LDR r8, =0
    LDR r4, =0xFFFFFFFF
    LDR r10, =0x0
    LDR r7, =0

    BL funcA
    ldr r2, =0
    ldr r3, =0
    BL montador
    LDR r3, =0x0
    BL insert1

    SWI 0x123456

```

```
funcA:
    LDR    r2, =0
    CMP    r3, #4
    MOVEQ  pc, lr
    ADD    r3, r3, #1
funcB:
    MOVS   r1, r1, ROR #1
    MOV    r10, r10, RRX
    CMP    r2, #1
    MOVEQ  r10, r10, LSR #1
    BEQ    funcA
    ADD    r2, r2, #1
    B      funcB
```

```
montador:
    CMP    r3, #4
    MOVEQ  pc, lr
    MOVS   r0, r0, LSR #1
    MOV    r1, r1, RRX
    CMP    r2, #1
    ldrEQ  r2, =0
    BEQ    recompoe
    ADD    r2, r2, #1
    B      montador
```

```
recompoe:
    MOVS   r1, r1, LSL #1
    MOV    r8, r8, RRX
    CMP    r2, #1
    ldrEQ  r2, =0
    ADDEQ  r3, r3, #1
    BEQ    montador
    ADD    r2, r2, #1
    B      recompoe
```

```
insert1:
    LDR    r2, =0
    CMP    r3, #4
    MOVEQ  pc, lr
    ADD    r3, r3, #1
```

```
loop1:
    MOVS   r8, r8, LSL #1
    MOV    r10, r10, RRX
    MOVS   r4, r4, ROR #1
    CMP    r2, #1
    MOVEQ  r10, r10, RRX
    BEQ    insert1
    ADD    r2, r2, #1
```



```
B      loop1
```

Com o código acima, ao final da rotina de execução, se obtém o código de gray de 3 dígitos a partir de código de gray de 2 dígitos conforme algoritmo presente na wikipedia¹

```
Register group: general
r0      0x0      0
r1      0xb4000000 -1275068
r2      0x0      0
r3      0x4      4
r4      0xffffffff -1
r5      0x1ffff8 2097144
r6      0x0      0
r7      0x0      0
r8      0x0      0
r9      0x0      0
r10     0x97e4c800 -1746614
r11     0x0      0
r12     0x1fffcc 2097100
sp      0x1ffff8 2097144
lr      0x824c   33356
pc      0x824c   33356
fps     0x0      0
cpsr    0x60000013 16106127

item-3-10-8.s
28      BL montador
29      LDR r3, =0x0
30      BL insert1
31
32      SWI 0x123456
33
34      funcA:
35      LDR    r2, =0
36      CMP    r3, #4
37      MOVEQ   pc, lr
38      ADD    r3, r3, #1

sim process 42 In: main                               Line: 32   PC: 0x824c
(gdb) r
Starting program: /home/student/src/a.out

Breakpoint 1, main () at item-3-10-8.s:17
Current language: auto; currently asm
(gdb) c
Continuing.

Breakpoint 2, main () at item-3-10-8.s:32
(gdb) p/t $r10
$1 = 10010111111001001100100000000000
```

¹ https://en.wikipedia.org/wiki/Gray_code#Constructing_an_n-bit_Gray_code