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

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

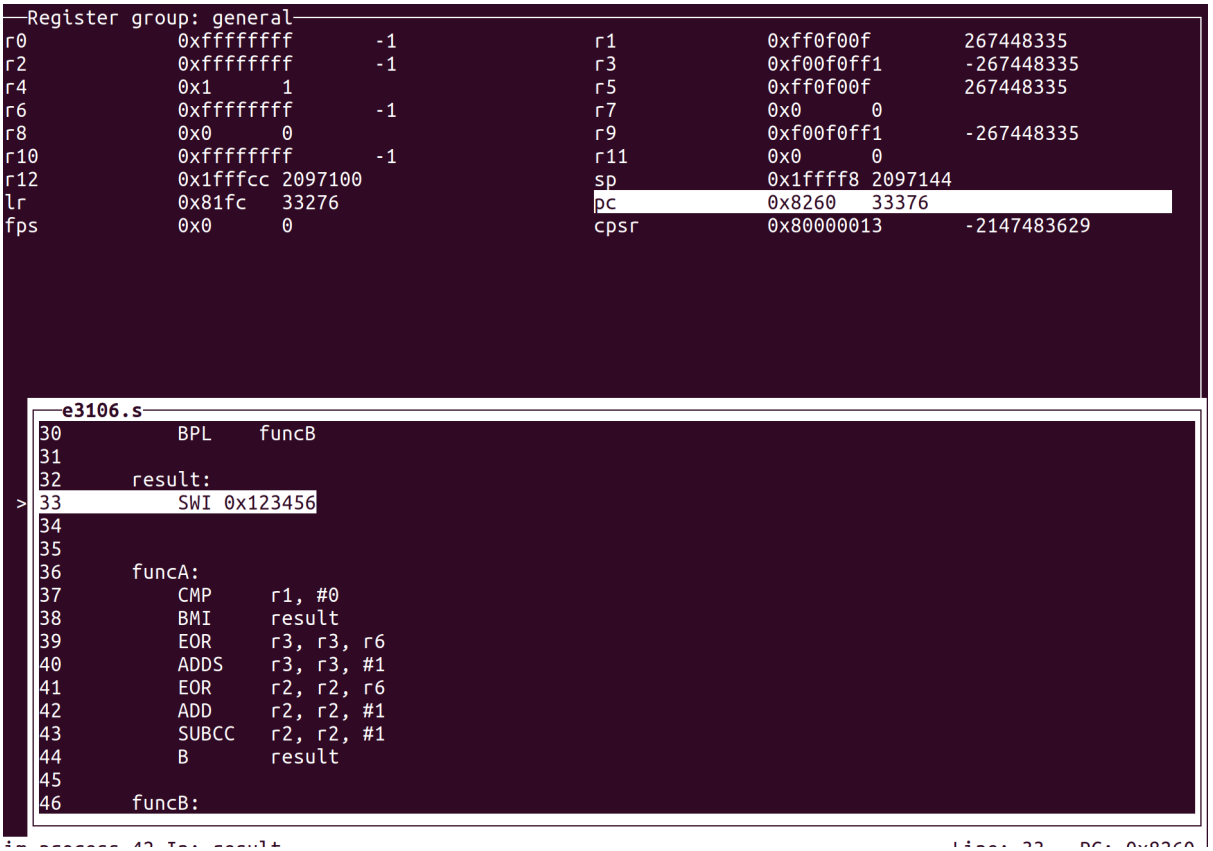
a. 3.10.5 64 bits signed multiplication

```
    .text
    .globl main
main:
    LDR     r0, =0x0FF0F00F
    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:

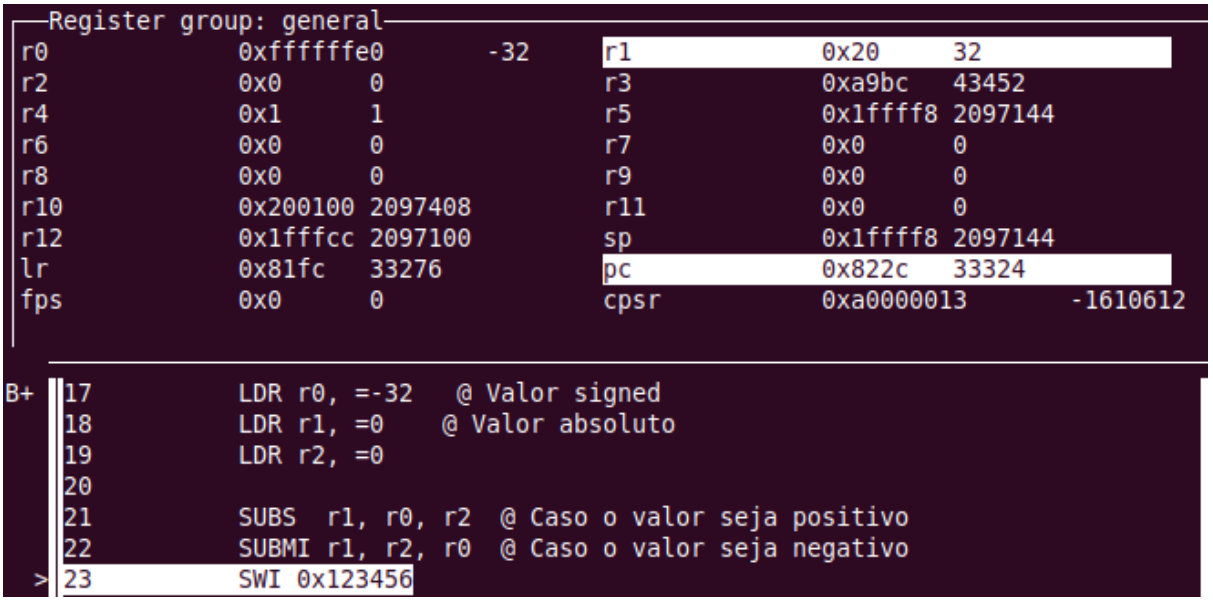


b. 3.10.6 Absolute value

```
    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:



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 signficativo de
r0 e r1
    LDR r5, =0x80000000 @Identificador de primeiro bit signficativo 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
```

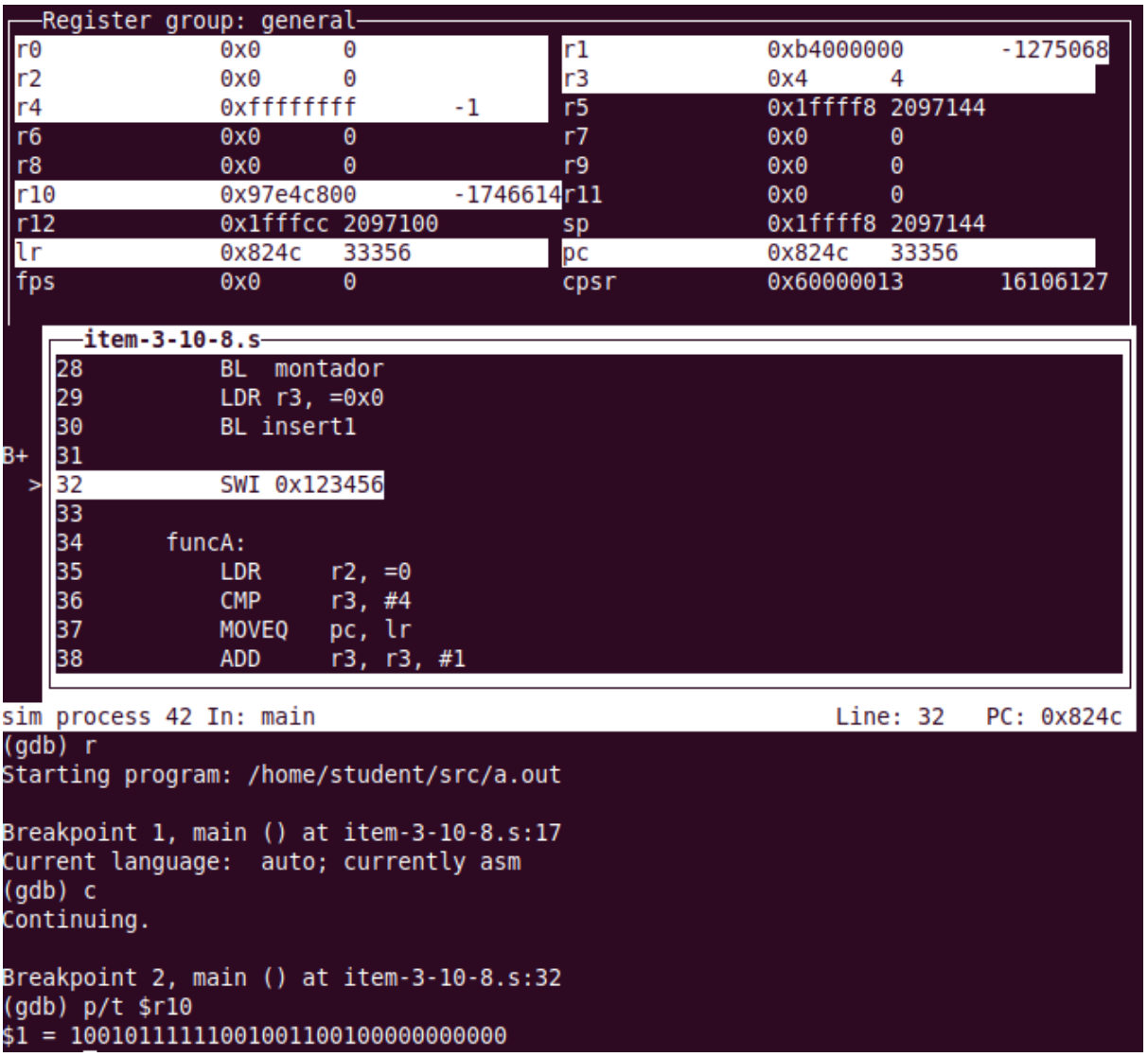
d. 3.10.8 Gray codes

```
    .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 wikipedia[1]



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