



Relatório Experiência 5

Conditional Executiong Loops

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
Exercício 5.5.1	3
Exercício 5.5.2	5
Exercício 5.5.3	6
Exercício 5.5.4	8
Exercício 5.5.5	9

1.Introdução

Estudo do capítulo 4 do apostila

2.Objetivo

Estudar a manipulação de dados alocados na memórias (loads e stores)

3.Planejamento

Disponível no documento de Planejamento da Experiência.

4.Relatório

a. Exercício 5.5.1

No exercício 5.5.1 tivemos um desafio semelhante a um exercício do planejamento da experiência anterior com um complicador adicional. Além de executar o laço “for”, deveríamos fazer a alocação dos vetores ‘a’ e ‘b’ em endereços de memória pré definidos (0x4000 para ‘a’ e 0x5000 para ‘b’).

```
.text
.globl main
main:
    MOV    r0, #0x4000
    MOV    r1, #0x5000

    ADR    r2, A @ estara decrescente
    ADR    r3, B @ estara crescente
```

@ Devemos carregar os vetores nos espaços de memória pre definidos

```
LDR r4, [r2]
LDR r5, [r2,#4]
LDR r6, [r3]
LDR r7, [r3,#4]
```

```
STR r4, [r0]
STR r5, [r0,#4]
STR r6, [r1]
STR r7, [r1,#4]
```

```
LDR r10, =0x0
BL loop
SWI 0x123456
```

loop:

```
CMP r10, #8
MOVPL pc, lr
```

```
RSB r8, r10, #7
LDRB r9, [r1, r8]
STRB r9, [r0, r10]
```

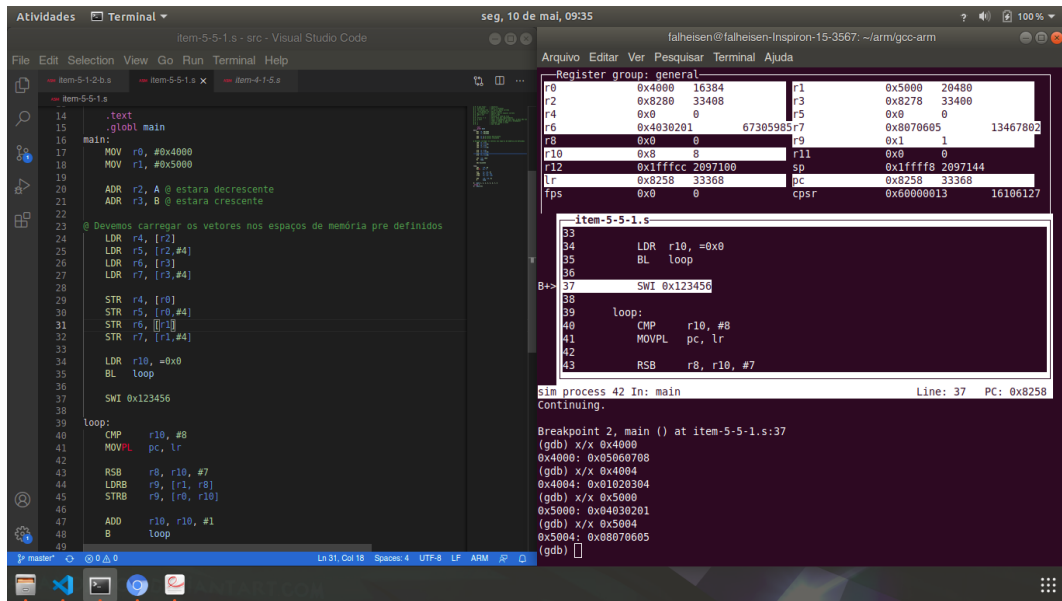
```
ADD r10, r10, #1
B loop
```

B: .byte 1, 2, 3, 4, 5, 6, 7, 8

.align 3

A: .skip 8*1

Como a imagem abaixo indica, ao final do código obtemos na posição 0x4000 os dados do vetor b de forma invertida. Os dados do vetor b, por sua vez, se encontram em 0x5000.



Resultado da rotina do código 5.5.1

b. Exercício 5.5.2

Para esse exercício, deveríamos realizar um código que executa a conta fatorial. Abaixo o código sintetizado.

```

.text
.globl main
main:
    MOV r6, #0xA
    MOV r4, r6

loop:
    SUBS    r4, r4, #1
    MULNE   r7, r4, r6
    MOVNE   r6, r7
    BNE     loop

    SWI 0x123456

```

```
Register group: general
r0      0x1      1          r1      0x1ffff8 2097144
r2      0xffffffff -1       r3      0xa9c0  43456
r4      0x0      0          r5      0x1ffff8 2097144
r6      0x375f00 3628800    r7      0x375f00 3628800
r8      0x0      0          r9      0x0      0
r10     0x200100 2097408    r11     0x0      0
r12     0x1fffc 2097100     sp      0x1ffff8 2097144
lr      0x81fc  33276       pc      0x8230  33328
fps      0x0      0        cpsr     0x60000013 1610612755

B+ 7      MOV r6, #0xA
8      MOV r4, r6
9      loop:
10     SUBS r4, r4, #1
11     MULNE r7, r4, r6
12     MOVNE r6, r7
13     BNE loop
14
B+> 15     SWI 0x123456
16
17
18
19
20
21
22
23

sim process 42 In: loop Line: 15 PC: 0x8230
(gdb) s
loop () at E52.s:10
(gdb) v 15
Undefined command: "v". Try "help".
(gdb) b 15
Breakpoint 2 at 0x8230: file E52.s, line 15.
(gdb) c
Continuing.

Breakpoint 2, loop () at E52.s:15
(gdb) █
```

Como podemos ver o resultado em r6 é 3628800 = 10!

c. Exercício 5.5.3

Para esse exercício, deveríamos encontrar o maior valor em um array de inteiros unsigned de 32 bits. O código que executa essa rotina está abaixo.

```
.text
.globl main
main:
    ADR    r0, serie
    ADR    r1, comps
    ADR    r2, max
```

```

LDR    r3, [r0], #4
LDR    r5, [r1]
SUBS   r5, r5, #1
BLPL   loop

STR    r3, [r2]
SWI    0x123456

loop:
LDR    r4, [r0], #4
CMP    r4, r3
MOVHI  r3, r4
SUBS   r5, r5, #1
MOVEQ  pc, lr
B      loop

serie: .word  58, 9, 1024, -1, 4589, -59, 3628800
comps: .word  7

max:   .word  0

```

Abaixo, uma captura de tela com um vetor de valores, sendo que o maior valor em complemento de 2 unsigned corresponde a -1 signed.

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

Register window:

Register	Value	Comment
r0	0x8270	33392
r2	0x8274	33396
r4	0x375f00	3628800
r5	0x0	0
r6	0x0	0
r7	0x0	0
r8	0x0	0
r9	0x0	0
r10	0x200100	2097408
r11	0x0	0
r12	0x1fffcc	2097100
sp	0x1ffff8	2097144
lr	0x8234	33332
pc	0x8238	33336
fps	0x0	0
cpsr	0x60000013	16106127

Command history:

```

(gdb) b 27
Breakpoint 2 at 0x8238: file item-5-5-3.s, line 27.
(gdb) r
Starting program: /home/student/src/a.out

Breakpoint 1, main () at item-5-5-3.s:17
Current language: auto; currently asm
(gdb) c
Continuing.

Breakpoint 2, main () at item-5-5-3.s:27
(gdb)

```

Resultado da rotina do código 5.5.3

d. Exercício 5.5.4

Sob orientação dos professores refizemos o exercício 5.5.4 para satisfazer as necessidades de forma correta, corrigindo os problemas apontados no planejamento.

```
.text
.globl main
main:
    LDR    r1, =0x16C764CB @X de entrada
    LDR    r2, =0x0        @Z de saída
    LDR    r8, =0b1011     @Sequencia a ser reconhecida
    LDR    r9, =4          @Tamanho da sequencia

    RSB    r5, r9, #32
    MOV    r11, r8, LSL r5

    LDR    r3, =0x80000000 @Ponta de escrita
    LDR    r4, =0x80000000 @Ponta de leitura
    MOV    r10, r9         @Contador de estados

    BL     Estat1

    SWI    0x123456

Estat1:
    AND    r0, r1, r4
    AND    r7, r11, r4
    CMP    r7, r0

    BEQ    Estat2

    MOV    r10, r9
    MOV    r11, r8, LSL r5
    B      Err

Estat2:
    MOV    r11, r11, LSL #1
    SUBS    r10, r10, #1
    BEQ    Estat3
    B      Eatt

Estat3:
    MOV    r10, r9
    MOV    r11, r8, LSL r5
```



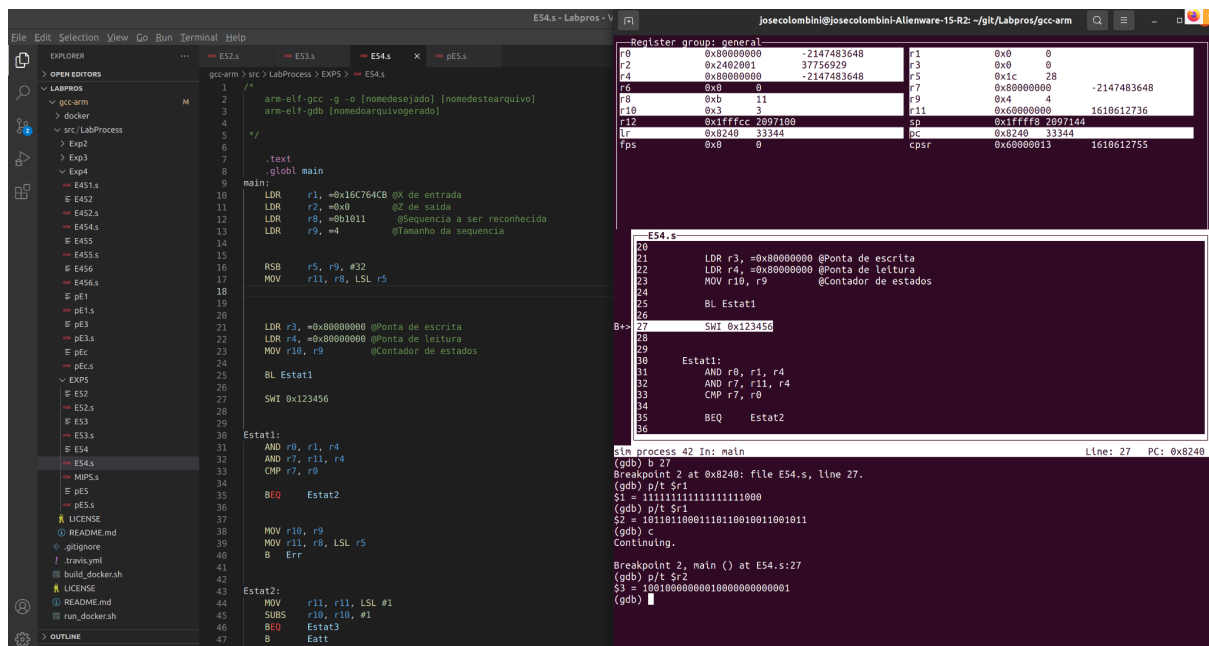
```
ADD r2, r2, r3
B Estat1
```

Eatt:

```
MOVS r3, r3, LSR #1
MOV r1, r1, LSL #1
MOVEQ pc, lr
B Estat1
```

Err:

```
AND r0, r1, r4
AND r7, r11, r4
CMP r7, r0
BEQ Estat2
B Eatt
```



Utilizamos as mesmas entradas do planejamento e os resultados foram iguais, provando a funcionalidade do novo método.

e. Exercício 5.5.5

Neste exercício, deveríamos retornar no registrador r1 um identificador da paridade de 1's do valor armazenado em r0. Em caso de um número par de 1's, deveríamos retornar 0 em r1, já no caso em que existem um número ímpar de 1's, r1 receberia 1.

Abaixo o código sintetizado para produzir essa função:

```
.text
.globl main
main:
    LDR    r0, =0x68123912    @ valor inicial
    LDR    r1, =0x0           @ valor a ser retornado
    LDR    r2, =0x0
    LDR    r3, =0x0           @ contador de bits 1 na palavra
    LDR    r4, =32

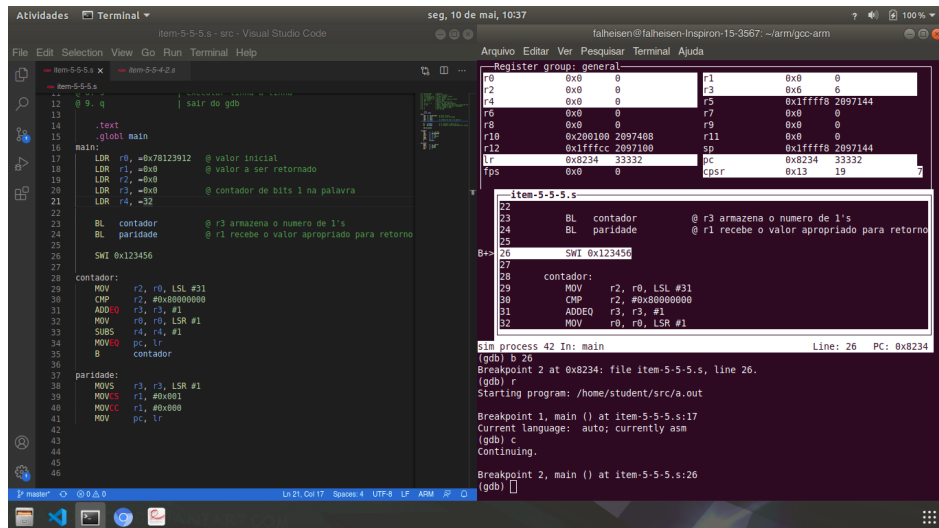
    BL     contador           @ r3 armazena o numero de 1's
    BL     paridade           @ r1 recebe o valor apropriado para retorno

    SWI    0x123456

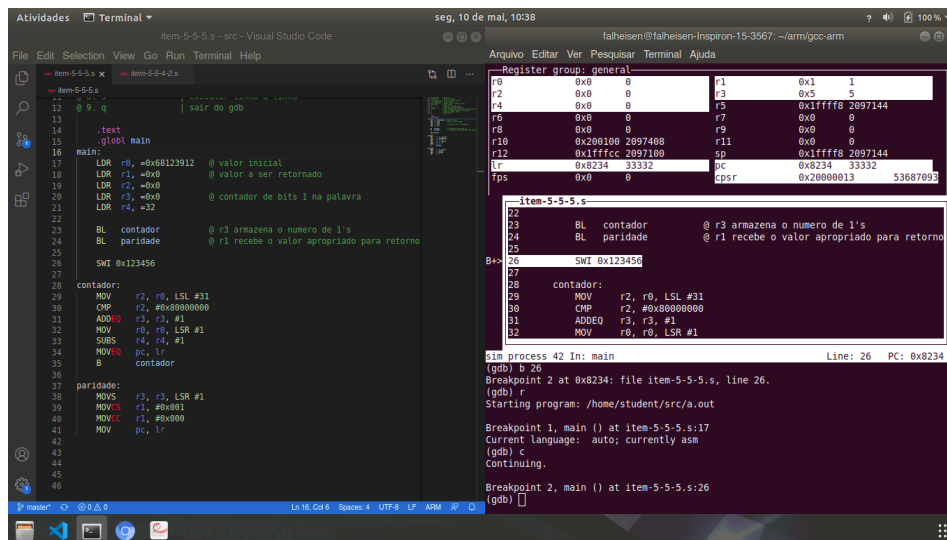
contador:
    MOV     r2, r0, LSL #31
    CMP     r2, #0x80000000
    ADDEQ   r3, r3, #1
    MOV     r0, r0, LSR #1
    SUBS    r4, r4, #1
    MOVEQ   pc, lr
    B       contador

paridade:
    MOVS    r3, r3, LSR #1
    MOVCS   r1, #0x001
    MOVCC   r1, #0x000
    MOV     pc, lr
```

A sub rotina do contador armazena no registrador r3 o número de bits 1 na palavra armazenada em r0. Por sua vez, a sub-rotina paridade avalia se o número é par ou ímpar e a partir desse valor atribui o valor à r1.



Exemplo utilizando uma palavra com número par de bits 1



Exemplo utilizando uma palavra com número par de bits 0