

## PCS3432 - Laboratório de Processadores



### Relatório 10

Nome	NUSP
Victor Hugo Chimenez Queiroz	11288405

Prof: Wilian França Costa/Marco Túlio Carvalho de Andrade

Para mudar o valor do timer, por exemplo para metade do tempo, basta executar um shift right na seguinte instrução:

```
timer_init:
    @ Enable timer0 interrupt
    LDR r0, INTEN
    LDR r1,=0x10 @bit 4 for timer 0 interrupt enable
    STR r1,[r0]

    @ Set timer value
    LDR r0, TIMER0L
    LDR r1, =0xff @setting timer value
    STR r1,[r0]

    @ Enable timer0 counting
    LDR r0, TIMER0C
    MOV r1, #0xE0 @enable timer module
    STR r1, [[r0]]

    @ Enable processor interrupt in CPSR
    mrs r0, cpsr
    bic r0,r0,#0x80
    msr cpsr_c,r0 @enabling interrupts in the cpsr

    mov pc, lr
```

Na subrotina timer\_init precisaríamos adicionar algo do tipo:

**MOV r1, r1, LSL #1**

Na seção Set timer value antes do LDR de r1.

Dessa maneira dividiríamos o timer pela metade.

Para adicionar uma terceira tarefa, precisamos mudar a rotina timer\_irq para que uma nova comparação seja adicionada:

```
timer_irq:
    STMFD sp!, {r10-r11, lr}
    BL handler_timer @vai para o rotina de tratamento da interupção de timer

    ADR r11, nproc @ r11 recebe o endereço de nproc
    LDR r10, [r11] @ r10 recebe o valor de nproc
    CMP r10, #0

    @ Guarda o r0 na primeira posição
    @ Usa r0 como base para indexar e armazena registradores
    MOVEQ r10, #1
    MOVNE r10, #0

    STR r10, [r11] @ Chaveamento entre os processos
```

Da maneira que está, o código compara se nproc é 0. Se for ele vai pra um, se não ele vai pra outro. Precisaremos adicionar um terceiro step, dessa maneira:

```
timer_irq:
    STMFD sp!, {r10-r11, lr}
    BL handler_timer @vai para o rotina de tratamento da interupção de timer

    ADR r11, nproc @ r11 recebe o endereço de nproc
    LDR r10, [r11] @ r10 recebe o valor de nproc

    @ Guarda o r0 na primeira posição
    @ Usa r0 como base para indexar e armazena registradores
    @ Se r0 for 0 precisa ir pra 1 (taskB)
    CMP r10, #0
    MOVEQ r10, #1
    @ Se r0 for 1 (taskB) precisa ir pra 2 (taskC)
    CMP r10, #1
    MOVEQ r10, #2
    @ Se for 2 (taskC) então deve voltar pra 0 (taskA)
    CMP r10, #2
    MOVEQ r10, #0

    STR r10, [r11] @ Chaveamento entre os processos

    LDMFD sp!, {r10-r11, lr}
    MOV pc, lr
```

Mas agora precisamos alterar também a persistência e restauração dos registradores em memória entre as tarefas.

Adicionando “mais IFs” na lógica, fica desta maneira na subrotina do\_irq\_interrupt para persistência dos registradores:

```

do_irq_interrupt: @ Rotina de interrupções IRQ
    @ Corrige lr
    SUB lr, lr, #4

    STMFD sp!, {r10-r11}

    ADR r11, nproc @ r11 = endereço de nproc
    LDR r10, [r11] @ r10 = valor de nproc

    @ Guarda o r0 na primeira posição
    @ Usa r0 como base para indexar e armazena registradores

    @ Se 0 então taskA
    CMP r10, #0
    STREQ r0, linhaA
    ADREQ r0, linhaA

    @ Se 1 então taskB
    CMP r10, #1
    STREQ r0, linhaB
    ADREQ r0, linhaB

    @ Se 2 então taskC
    CMP r10, #2
    STREQ r0, linhaC
    ADREQ r0, linhaC

    LDMFD sp!, {r10-r11}

    ADD r0, r0, #4
    STMIA r0!, {r1-r12, lr}

```

Agora para restauração deles:

```

@@ Interrupção
LDR r0, INTPND @Carrega o registrador de status de interrupção
LDR r0, [r0]
TST r0, #0x0010 @verifica se é uma interrupção de timer
BLNE timer_irq

@ Preparando o processo para a proxima task
ADR r11, nproc @ r11 = endereco de nproc
LDR r10, [r11] @ r10 = valor de nproc

@ Restaura todos os registradores para a proxima task
@ Se igual a 0 então taskA
CMP r10, #0
ADREQ r0, linhaA
@ Se igual a 1 então taskB
CMP r10, #1
ADREQ r0, linhaB
@ Se igual a 2 então taskC
CMP r10, #2
ADREQ r0, linhaC

ADD r0, r0, #68

MRS r1, cpsr
MSR cpsr_ctl, #0b11010011 @ Modo supervisor
LDMDB r0!, {sp, lr} @ Restaura sp e lr
vesp_saida:
MSR cpsr, r1 @ Sai modo supervisor

LDMDB r0!, {r1}
MSR spsr, r1 @ Restaura spsr

```

Por fim precisamos inicializar a pilha para C:

```

3 Reset:
4     LDR sp, =supervisor_stack_top
5
6     @ Inicializacao das stacks
7     MRS r0, cpsr @ salvando o modo corrente em R0
8     MSR cpsr_ctl, #0b11010010 @ alterando o modo para irq - o SP eh automaticamente chaveado ao chavear o modo
9     LDR sp, =irq_stack_top @ a pilha de irq eh setada
10    MSR cpsr, r0 @ volta para o modo anterior
11
12    bl timer_init @ initialize interrupts and timer 0
13
14    MRS r0, cpsr @ valor de cpsr para taskB
15    LDR r1, _taskB @ valor de pc para taskB
16    SUB r2, sp, #0x500 @ valor de sp para taskB
17
18    ADR r3, linhaB
19    STR r1, [r3, #52] @ guarda pc
20    STR r0, [r3, #56] @ guarda cpsr
21    STR r2, [r3, #60] @ guarda sp
22
23
24    MRS r0, cpsr @ valor de cpsr para taskC
25    LDR r1, _taskC @ valor de pc para taskC
26    SUB r2, sp, #0x800 @ valor de sp para taskC
27
28    ADR r3, linhaC
29    STR r1, [r3, #52] @ guarda pc
30    STR r0, [r3, #56] @ guarda cpsr
31    STR r2, [r3, #60] @ guarda sp
32
33    b taskA

```

Podemos ver a alteração entre os processos no print abaixo:

```
victor ~/assembly/p2 main 10:59 ./roda1_qemu.sh
pulseaudio: set_sink_input_volume() failed
pulseaudio: Reason: Invalid argument
pulseaudio: set_sink_input_mute() failed
pulseaudio: Reason: Invalid argument

11288405 11288405 11288405 11288405 11288405 11288405

Laboratorio de Processadores
Victor
11288405
Laboratorio de Processadores
Victor
11288405
```

de maneira continuada:

[illegible]

Agora tentarei aumentar o tempo para ficar mais visível:

