ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO
DISCIPLINA: LABORATÓRIO DE PROCESSADORES- PCS3732
1° QUADRIMESTRE/2021

Aula 11

29 de Julho de 2021

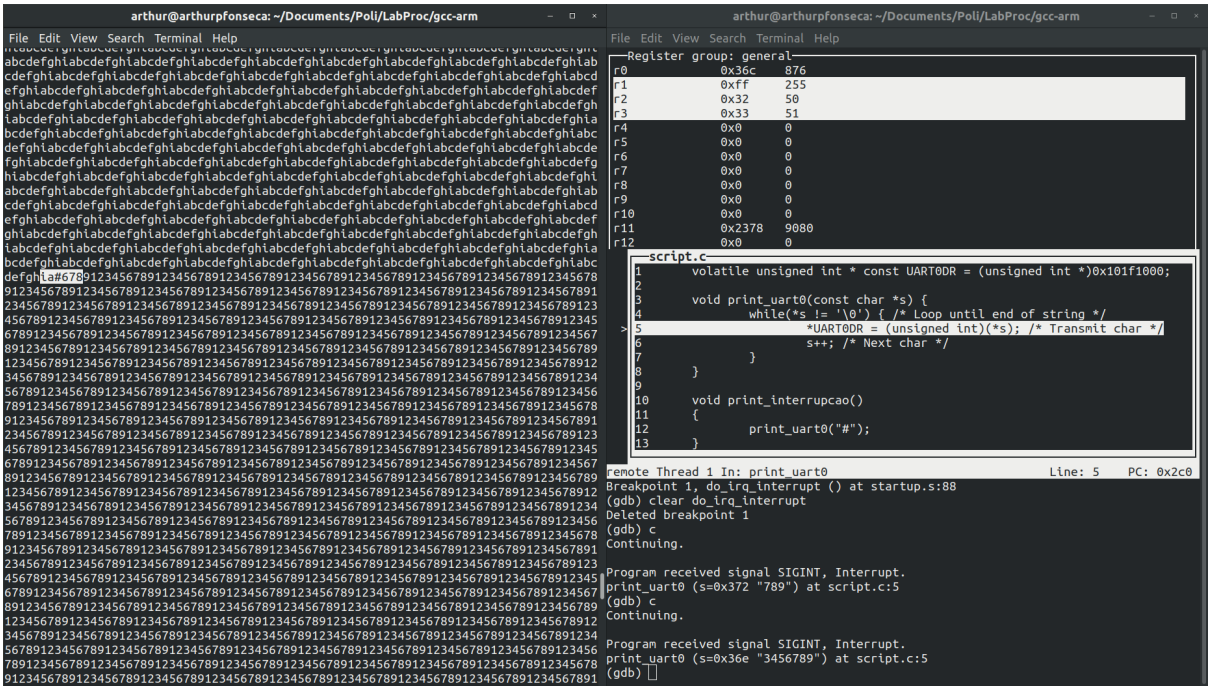
GRUPO 10

Arthur Pires da Fonseca	NUSP: 10773096
Bruno José Móvio	NUSP: 10336852
Iago Soriano Roque Monteiro	NUSP: 8572921

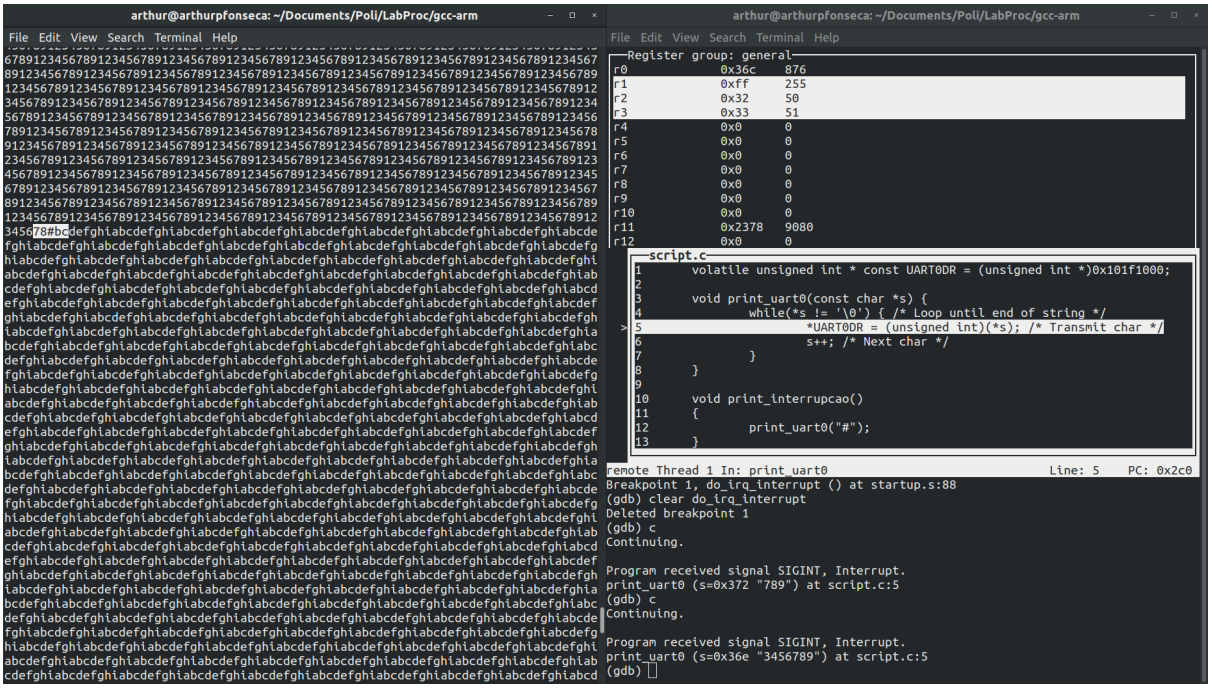
Sumário

Chaveamento de processos

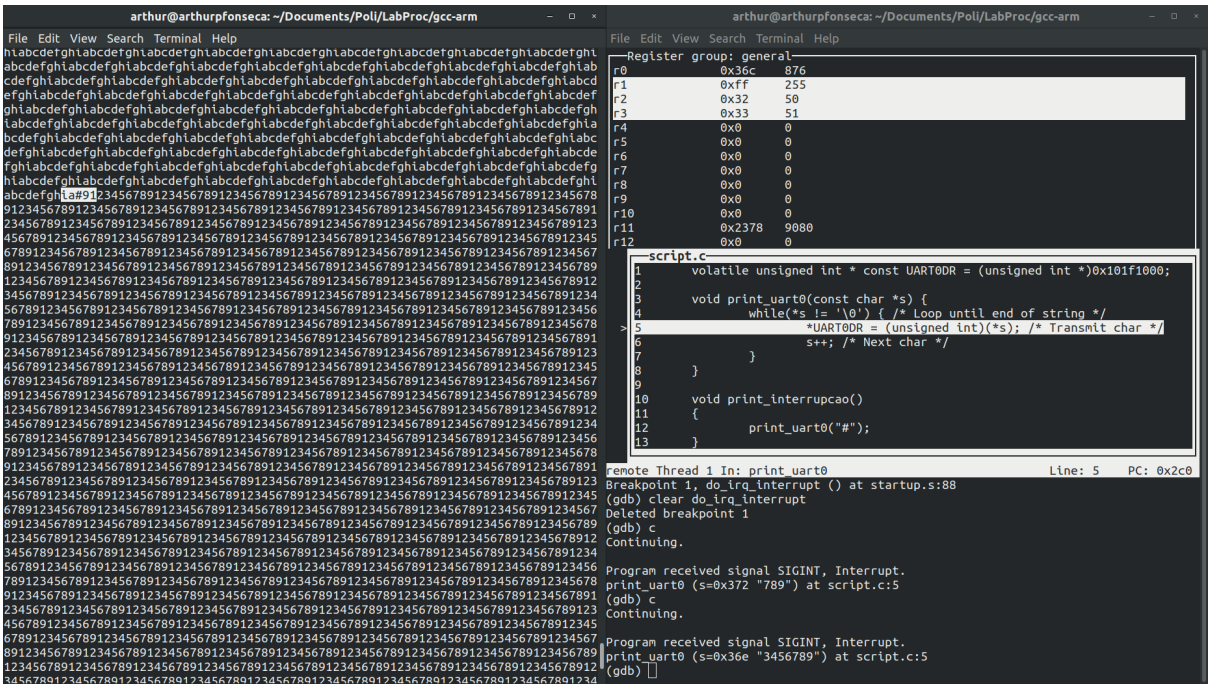
Abaixo temos um print dos vários “123456789” sendo impressos pela tarefa 1, seguidos do “#” da interrupção, e os *“abcdegfhi”* da tarefa B. Notamos que, como era esperado, os processos retomam onde estavam depois de terem sido interrompidos.



Primeiro processo interrompido em “a”, ou seja, logo antes de printar “b”



Novo processo interrompido em “8”, ou seja, logo antes de printar “9”, em seguida retorna para a letra “b”, continuando o processo anterior



Novo processo interrompido em “a”, ou seja, logo antes de printar “b”, em seguida retorna para o número “9”, continuando o processo anterior

Para rodar o código, fazemos

eabi-gcc script.c -o script.o && eabi-as startup.s -o startup.o && eabi-ld -T irqld.ld startup.o script.o -o irq.elf && eabi-bin irq.elf irq.bin && qemu irq.bin

E, em seguida, em outro terminal,

eabi-qemu -se irq.elf

Apêndice

script.c

```
volatile unsigned int * const UART0DR = (unsigned int *)0x101f1000;

void print_uart0(const char *s) {
   while(*s != '\0') { /* Loop until end of string */
                    *UART0DR = (unsigned int)(*s); /* Transmit char */
         s++; /* Next char */
         }
}

void print_interrupcao()
{
   print_uart0("#");
}

void print_loop()
{
   print_uart0(" ");
}

void print_task_a()
{
   print_uart0("123456789");
}

void print_task_b()
{
   print_uart0("abcdefghi");
}
```

startup.s

```
   .global _start
.text

_start:
   b _Reset @posição 0x00 - Reset
   ldr pc, _undefined_instruction @posição 0x04 - Intrução não-definida
   ldr pc, _software_interrupt @posição 0x08 - Interrupção de Software
   ldr pc, _prefetch_abort @posição 0x0C - Prefetch Abort
```

```
    ldr pc, _data_abort @posição 0x10 - Data Abort
    ldr pc, _not_used @posição 0x14 - Não utilizado
    ldr pc, _irq @posição 0x18 - Interrupção (IRQ)
    ldr pc, _fiq @posição 0x1C - Interrupção(FIQ)

_undefined_instruction: .word undefined_instruction
_software_interrupt: .word software_interrupt
_prefetch_abort: .word prefetch_abort
_data_abort: .word data_abort
_not_used: .word not_used

_irq: .word irq
_fiq: .word fiq

INTPND: .word 0x10140000 @Interrupt status register
INTSEL: .word 0x1014000C @interrupt select register( 0 = irq, 1 = fiq)
INTEN: .word 0x10140010 @interrupt enable register
TIMER0L: .word 0x101E2000 @Timer 0 load register
TIMER0V: .word 0x101E2004 @Timer 0 value registers
TIMER0C: .word 0x101E2008 @timer 0 control register
TIMER0X: .word 0x101E200c @timer 0 interrupt clear register

_Reset:
    MRS r0, cpsr                    @ salvando o modo corrente em R0
    MSR cpsr_ctl, #0b11010010     @ alterando para modo interrupt
    LDR sp, =timer_stack_top     @ a pilha de interrupções de tempo é setada
    MSR cpsr, r0                    @ retorna para o modo anterior

    adr r0, linhaB                  @ endereco base do b
    add r0, r0, #52                 @ soma o que ja esta alocado


    ldr r1, =task
    stmfa r0!, {r1}                 @ pc do b

    ldr r1, =stack_top_b
    stmfa r0!, {r1}                 @ sp do b

    ldr r1, =0x0
    stmfa r0!, {r1}                 @ lr do b

    ldr r1, =0xd3
    stmfa r0!, {r1}                 @ cpsr do b

    LDR sp, =stack_top_a

    bl main
```

```
   b .


undefined_instruction:
   b .

software_interrupt:
   b do_software_interrupt @vai para o handler de interrupções de software

prefetch_abort:
        b .

data_abort:
        b .

not_used:
        b .

irq:
   b do_irq_interrupt @vai para o handler de interrupções IRQ

fiq:
   b .


do_software_interrupt: @Rotina de Interrupçãode software
   add r1, r2, r3 @r1 = r2 + r3
   mov pc, r14 @volta p/ o endereço armazenado em r14


do_irq_interrupt:      @ Rotina de interrupções IRQ
   STMFD sp!, {r12}            @ r0 guarda o endereco onde ficarao os valores
dos registradores
   STMFD sp!, {r10, r11}       @ r1 = aux, r2 = aux

   mrs r12, cpsr               @ guarda o cpsr temporariamente
   orr r12, #0xc0              @ setar bits I = 1 e F = 1 (disable interrupt)
   msr cpsr_c, r12             @ cpsr nao deixa ter interrupcoes

   ldr r12, which_proc    @ pega qual o proc atual
   adr r11, which_proc    @ ponteiro para o proc atual

   cmp r12, #0xa       @ ve se e o proc a
   beq chaveia_b       @ chaveia para o b
   cmp r12, #0xb       @ ve se e o proc a
   beq chaveia_a       @ chaveia para o b
deu_pau:
```

```
   b deu_pau          @ isso nao existe

chaveia_b:
   ldr r12, =0xb      @ poe b no lugar
   str r12, [r11]     @ guarda na posicao da label which_proc

   ldmfd sp!, {r10, r11}   @ recupera r1 e r2 (abrir espaco)

   ldr r12, =linhaB   @ endereco de onde pegar os registradores de antes
   stmfd sp!, {r12}   @ poe na pilha o endereco de onde vai pegar os
registradores velhos

   stmfd sp!, {r10, r11}   @ poe r1 e r2 de novo na pilha

   ldr r12, =linhaA   @ endereco pra salvar os registradores atuais

   b chaveando        @ escolheu os enderecos do chaveamento

chaveia_a:
   ldr r12, =0xa      @ poe b no lugar
   str r12, [r11]     @ guarda na posicao da label which_proc

   ldmfd sp!, {r10, r11}   @ recupera r1 e r2 (abrir espaco)

   ldr r12, =linhaA   @ endereco de onde pegar os registradores de antes
   stmfd sp!, {r12}   @ poe na pilha o endereco de onde vai pegar os
registradores velhos

   stmfd sp!, {r10, r11}   @ poe r1 e r2 de novo na pilha

   ldr r12, =linhaB   @ endereco pra salvar os registradores atuais

   b chaveando        @ escolheu os enderecos do chaveamento

chaveando:

   ldmfd sp!, {r10, r11}          @ recupera r1 e r2 originais
   stmfa r12!, {r0-r11}           @ guarda em linhaA ou linhaB todos os
registradores de proposito geral

   ldmfd sp!, {r9}                @ pega o topo da pilha (linhaA ou linhaB)
   ldmfd sp!, {r1}                @ pega o valor de r12, que agora esta no
topo da pilha
   stmfa r12!, {r1}          @ guarda o valor original de r12 em linhaA ou
linhaB

   sub r0, lr, #4                 @ pega o pc certo
```

```
   stmfa r12!, {r0}            @ guarda o pc

   mrs r1, cpsr                    @ salvando o modo corrente em R1
   msr cpsr_ctl, #0b11010011       @ alterando para modo 13 (supervisor) =>
sp atual e o sp certo
   stmfa r12!, {sp}            @ guarda o sp correcto

   mov r2, lr                  @ salva lr
   sub lr, lr, #4                  @ tira 4 para acertar o lr
   stmfa r12!, {lr}            @ guarda o lr
   mov lr, r2                  @ recupera lr

   mrs r2, cpsr                    @ guarda o cpsr temporariamente
   @orr r2, r2, #0xc0              @ setar bits I = 0 e F = 0 (enable interrupt)
   stmfa r12!, {r2}            @ guarda o cpsr em linhaA

   msr cpsr_c, r1
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ volta para o
modo anterior (interrupcao)


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


   BLNE handler_timer              @vai para o rotina de tratamento da
interupção de timer

   add r9, r9, #68                     @ soma o espaco alocado
   ldmfa r9!, {r0}                     @ recupera cpsr atraves de r0
   msr spsr, r0                        @ guarda o proximo cpsr pro ^ ter
efeito
   msr cpsr_c, r0                      @@@@@@@@@@@ volta pra o
modo anterior

   ldmfa r9!, {sp, lr}                 @ recupera sp e lr

   @msr cpsr_c, r2                     @ volta para o modo guardado
anteriormente em r2

   ldmfa r9!, {r0-r12, pc}^        @ pega o valor de todos os outros regs


handler_timer:
```

```
   LDR r0, TIMER0X
   MOV r1, #0x0
   STR r1, [r0] @Escreve no registrador TIMER0X para limpar o pedido de
interrupção

   stmfd sp!, {lr}              @ salva o lr na pilha
   BL print_interrupcao
   ldmfd sp!, {lr}              @ recupera o lr

   mov pc, lr @retorna


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

   LDR r0, TIMER0C
   LDR r1, [r0]
   MOV r1, #0xA0 @enable timer module
   STR r1, [r0]

   LDR r0, TIMER0V
   MOV r1, #0xff @setting timer value
   STR r1,[r0]

   mrs r0, cpsr
   bic r0,r0,#0x80
   msr cpsr_c,r0 @enabling interrupts in the cpsr

   mov pc, lr


which_proc:
   .word 0xa            @ Diz qual processo esta rodando agora

main:
   bl timer_init        @ initialize interrupts and timer 0
task:
   ldr r0, which_proc   @ pega qual o proc atual

   cmp r0, #0xa         @ ve se e o proc a
   bleq print_task_a    @ chama a task a
   beq task             @ continua o loop

   cmp r0, #0xb         @ ve se e o proc b
   bleq print_task_b    @ chama a task b
```

```
   b task

aux:
   .space 4    @ espaco para registrador temporario

linhaA:
   .space 68    @ 68 bytes para o proc a
linhaB:
   .space 68    @ 68 bytes para o proc b
```

irqld.ld

```
ENTRY(_start)
SECTIONS
{
 . = 0x0;
 .text : {
 startup.o (INTERRUPT_VECTOR)
 *(.text)
 }
 .data : { *(.data) }
 .bss : { *(.bss) }
 . = . + 0x1000; /* 4kB of stack memory */
 stack_top_b = .;
 . = . + 0x1000; /* 4kB of stack memory */
 stack_top_a = .;
 . = . + 0x1000; /* 4kB of stack memory */
 timer_stack_top = .;
}
```