ESCOLA POLITÉCNICA DA UNIVERSIDADE DE SÃO PAULO Graduação em Engenharia Computação

PCS3732 - Laboratório de Processadores

Professor Jorge Kinoshita



Grupo 10 - Planejamento E11

Arthur Pires da Fonseca NUSP: 10773096

Sumário

3
3
3
5
5
6
6
6
7

Planejamento

Enunciado - o que fazer

Reescreva o código de forma que na interrupção de relógio salvem-se todos os registradores (incluindo LR, SP, PC, CPSR) e sejam recuperados (como se fosse o chaveamento de um processo apenas).

Modificações no código

Foram adicionadas algumas linhas de ARM Assembly no arquivo "startup.s" para adaptar o que foi pedido. O apêndice deste relatório descreve como se pode rodar o programa e inclui outros arquivos auxiliares que fazem parte da compilação.

```
do irq interrupt: @Rotina de interrupções IRQ
       STMFD sp!, {r0, r2} @ r0 guarda o endereco onde ficarao os valores dos
registradores, r2 = aux
       STMFD sp!, {r1}
                                    @ r1 vai guardar o valor de r0
                            @ sobrescreve r1 com r0
       mov r1, r0
       mov r1, r0
ldr r0, =linhaA
                            @ endereco do local onde estarao os valores dos
registradores
      stmfd r0!, {r1}
                            @ quarda em linhaA r0
       Idmfd sp!, {r1}
                            @ recupera r1
       stmfd r0!, {r1-r12}
                            @ guarda em linhaA todos os registradores de proposito
geral
       mov r2, Ir
                                    @ salva lr
       sub Ir, Ir, #4
                                    @ tira 4 para acertar o Ir
       stmfd r0!, {sp, lr, pc}
                                    @ guarda os registradores banked
       mov lr, r2
                                    @ recupera Ir
       mrs r2, cpsr
                                    @ guarda o cpsr temporariamente
                                    @ setar bits I = 1 e F = 1 (unable interrupt)
       orr r2, r2, #0xc0
                                    @ guarda o cpsr em linhaA
       stmfd r0!, {r2}
                                           @ volta para o modo anterior
       msr cpsr c, r2
       LDMFD sp!, {r0, r2}
                                    @ recupera r0 e r2
       STMFD sp!, {r0 - r3, LR} @Empilha os registradores
       LDR r0, INTPND @Carrega o registrador de status de interrupção
       LDR r0, [r0]
       TST r0, #0x0010 @verifica se é uma interupção de timer
       STMFD sp!, {pc}
                                           @ salva pc na pilha do modo de interrupções
       BLNE handler timer @vai para o rotina de tratamento da interupção de timer
       LDMFD sp!, {r0 - r3,lr} @retorna
       sub Ir, Ir, #4
                     @ corrigindo o Ir
       STMFD sp!, {Ir}
```

Arquivo startup.s (parte modificada)

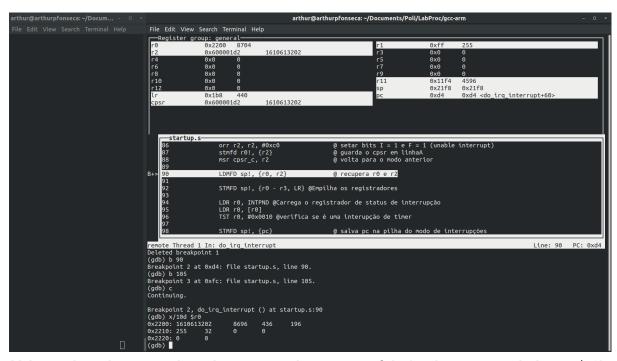
Arquivo irqld.ld

Enunciado - execução

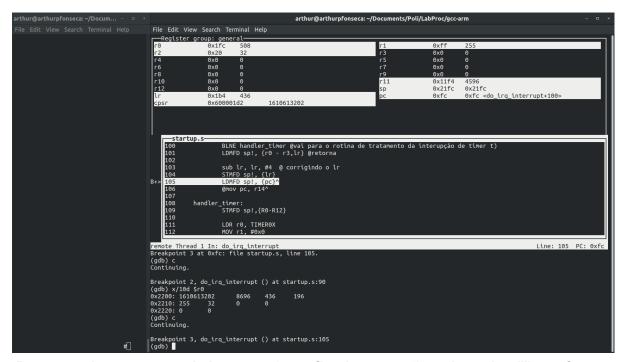
Compile e execute o código em casa. Observe os '#"s e ' 's sendo impressos como acontecia na aula de interrupção.

Execução

Os resultados foram como o esperado e são mostrados a seguir:



Valores de todos os registradores gravados na memória (endereço guardado em \$r0)



Processo de tratamento de interrupção ao fim da execução, printando "#", conforme a especificação

Apêndice

Como rodar o código todo

```
# No primeiro terminal
eabi-gcc script.c -o script.o
eabi-as startup.s -o startup.o
eabi-ld -T irqld.ld irq.o startup.o script.o -o irq.elf
eabi-bin irq.elf irq.bin
qemu irq.bin

# Em outro terminal
eabi-qemu -se irq.elf

# Para parar o qemu
pkill qemu
```

Arquivo script.c

Arquivo completo do 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
       Idr 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
fig: .word fig
INTPND: .word 0x10140000 @Interrupt status register
INTSEL: .word 0x1014000C @interrupt select register(0 = irq, 1 = fiq)
INTEN: .word 0x10140010 @interrupt enable register
TIMEROL: .word 0x101E2000 @Timer 0 load register
TIMEROV: .word 0x101E2004 @Timer 0 value registers
TIMEROC: .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
       LDR sp, =stack_top
       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 irg 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!, {r0, r2} @ r0 guarda o endereco onde ficarao os valores dos
registradores, r2 = aux
       STMFD sp!, {r1}
                                    @ r1 vai guardar o valor de r0
       mov r1, r0
                            @ sobrescreve r1 com r0
       ldr r0, =linhaA
                            @ endereco do local onde estarao os valores dos
registradores
       stmfd r0!, {r1}
                            @ guarda em linhaA r0
       Idmfd sp!, {r1}
                            @ recupera r1
       stmfd r0!, {r1-r12}
                            @ guarda em linhaA todos os registradores de proposito
geral
       mov r2. Ir
                                    @ salva Ir
       sub Ir, Ir, #4
                                    @ tira 4 para acertar o Ir
                                    @ guarda os registradores banked
       stmfd r0!, {sp, lr, pc}
       mov lr, r2
                                    @ recupera Ir
       mrs r2, cpsr
                                    @ guarda o cpsr temporariamente
                                    @ setar bits I = 1 e F = 1 (unable interrupt)
       orr r2, r2, #0xc0
                                    @ guarda o cpsr em linhaA
       stmfd r0!, {r2}
                                           @ volta para o modo anterior
       msr cpsr c, r2
       LDMFD sp!, {r0, r2}
                                    @ recupera r0 e r2
       STMFD sp!, {r0 - r3, LR} @Empilha os registradores
       LDR r0, INTPND @Carrega o registrador de status de interrupção
       LDR r0, [r0]
       TST r0, #0x0010 @verifica se é uma interupção de timer
       STMFD sp!, {pc}
                                           @ salva pc na pilha do modo de interrupções
       BLNE handler timer @vai para o rotina de tratamento da interupção de timer
       LDMFD sp!, {r0 - r3,lr} @retorna
```

```
sub Ir, Ir, #4
                     @ corrigindo o Ir
       STMFD sp!, {Ir}
       LDMFD sp!, {pc}^
       @mov pc, r14^
handler_timer:
       STMFD sp!,{R0-R12}
       LDR r0, TIMER0X
       MOV r1, #0x0
       STR r1, [r0] @Escreve no registrador TIMER0X para limpar o pedido de
interrupção
       BL print_interrupcao
       LDMFD sp!,{R0-R12}
       LDMFD sp!, {pc}
       mov pc, r14 @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, Ir
main:
       bl timer_init @initialize interrupts and timer 0
stop:
       BL print_loop
       b stop
```