

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

Planejamento	3
Enunciado - o que fazer	3
Modificações no código	3
Enunciado - execução	5
Execução	5
Apêndice	6
Como rodar o código todo	6
Arquivo script.c	6
Arquivo completo do startup.s	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 endereço onde ficarão 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 @ endereço do local onde estarão os valores dos
registradores
    stmfd r0!, {r1} @ guarda em linhaA r0

    ldmfd sp!, {r1} @ recupera r1
    stmfd r0!, {r1-r12} @ guarda em linhaA todos os registradores de propósito
geral

    mov r2, lr @ salva lr
    sub lr, lr, #4 @ tira 4 para acertar o lr
    stmfd r0!, {sp, lr, pc} @ guarda os registradores banked
    mov lr, r2 @ recupera lr

    mrs r2, cpsr @ guarda o cpsr temporariamente
    orr r2, r2, #0xc0 @ setar bits I = 1 e F = 1 (unable interrupt)
    stmfd r0!, {r2} @ guarda o cpsr em linhaA
    msr cpsr_c, r2 @ volta para o modo anterior

    LDMFD sp!, {r0, r2} @ recupera r0 e r2

    STMFD sp!, {r0 - r3, LR} @Empilha os registradores

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

    STMFD sp!, {pc} @ salva pc na pilha do modo de interrupções

    BLNE handler_timer @vai para o rotina de tratamento da interrupção de timer
    LDMFD sp!, {r0 - r3, lr} @retorna

    sub lr, lr, #4 @ corrigindo o lr
    STMFD sp!, {lr}
```

```
LDMFD sp!, {pc}^
```

Arquivo startup.s (parte modificada)

```
ENTRY(_start)
SECTIONS
{
    . = 0x0;
    .text : {
        startup.o (INTERRUPT_VECTOR)
        *(.text)
    }
    .data : { *(.data) }
    .bss : { *(.bss) }
    . = . + 0x1000; /* 4kB of stack memory */
    stack_top = .;
    . = . + 0x1000; /* 4kB of stack memory */
    timer_stack_top = .;
    . = . + 0x44; /* memoria para chaveamento de processos */
    linhaA = .;
}
```

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:

```
arthur@arthurpfonseca: ~/Documents/Pol/LabProc/gcc-arm
File Edit View Search Terminal Help

Register group: general
r0 0x2200 8704
r2 0x600001d2 1610613202
r4 0x0 0
r6 0x0 0
r8 0x0 0
r10 0x0 0
r12 0x0 0
lr 0x1b8 440
cpsr 0x600001d2 1610613202
r1 0xff 255
r3 0x0 0
r5 0x0 0
r7 0x0 0
r9 0x0 0
r11 0x11f4 4596
sp 0x21f8 0x21f8
pc 0xd4 0xd4 <do_irq_interrupt+60>

startup.s
86 orr r2, r2, #0xc0 @ setar bits I = 1 e F = 1 (unable interrupt)
87 stmf r0!, {r2} @ guarda o cpsr em linhaA
88 msr cpsr_c, r2 @ volta para o modo anterior
89
B-> 90 LDMFD sp!, {r0, r2} @ recupera r0 e r2
91
92 STMFD sp!, {r0 - r3, LR} @Emplha os registradores
93
94 LDR r0, INTEND @Carrega o registrador de status de interrupção
95 LDR r0, {r0}
96 TST r0, #0x0010 @verifica se é uma interrupção de timer
97
98 STMFD sp!, {pc} @ salva pc na pilha do modo de interrupções

remote Thread 1 In: do_irq_interrupt
Deleted breakpoint 1
(gdb) b 90
Breakpoint 2 at 0xd4: file startup.s, line 90.
(gdb) b 105
Breakpoint 3 at 0xfc: file startup.s, line 105.
(gdb) c
Continuing.

Breakpoint 2, do_irq_interrupt () at startup.s:90
(gdb) x/10d $r0
0x2200: 1610613202 8696 436 196
0x2210: 255 32 0 0
0x2220: 0 0 0 0
(gdb)
```

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

```
arthur@arthurpfonseca: ~/Documents/Pol/LabProc/gcc-arm
File Edit View Search Terminal Help

Register group: general
r0 0x1fc 508
r2 0x20 32
r4 0x0 0
r6 0x0 0
r8 0x0 0
r10 0x0 0
r12 0x0 0
lr 0x1b4 436
cpsr 0x600001d2 1610613202
r1 0xff 255
r3 0x0 0
r5 0x0 0
r7 0x0 0
r9 0x0 0
r11 0x11f4 4596
sp 0x21fc 0x21fc
pc 0xfc 0xfc <do_irq_interrupt+100>

startup.s
100 BLNE handler_timer @val para o rotina de tratamento da interrupção de timer t)
101 LDMFD sp!, {r0 - r3,lr} @retorna
102
103 sub lr, lr, #4 @ corrigindo o lr
104 STMFD sp!, {lr}
B-> 105 LDMFD sp!, {pc}^
106 @mov pc, r14^
107
108 handler_timer:
109 STMFD sp!, {R0-R12}
110
111 LDR r0, TIMER0X
112 MOV r1, #0x0

remote Thread 1 In: do_irq_interrupt
Breakpoint 3 at 0xfc: file startup.s, line 105.
(gdb) c
Continuing.

Breakpoint 2, do_irq_interrupt () at startup.s:90
(gdb) x/10d $r0
0x2200: 1610613202 8696 436 196
0x2210: 255 32 0 0
0x2220: 0 0 0 0
(gdb) c
Continuing.

Breakpoint 3, do_irq_interrupt () at startup.s:105
(gdb)
```

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

```
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(" ");
}
```

Arquivo completo do startup.s

```
.global _start
.text

_start:
    b _Reset @posição 0x00 - Reset
    ldr pc, _undefined_instruction @posição 0x04 - Instrução 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

    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_irq_interrupt @vai para o handler de interrupções IRQ

fiq:
    b .

do_software_interrupt: @Rotina de Interrupção de 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 endereço onde ficarão 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 @ endereço do local onde estarão os valores dos
registradores
    stmfd r0!, {r1} @ guarda em linhaA r0

    ldmfd sp!, {r1} @ recupera r1
    stmfd r0!, {r1-r12} @ guarda em linhaA todos os registradores de propósito
geral

    mov r2, lr @ salva lr
    sub lr, lr, #4 @ tira 4 para acertar o lr
    stmfd r0!, {sp, lr, pc} @ guarda os registradores banked
    mov lr, r2 @ recupera lr

    mrs r2, cpsr @ guarda o cpsr temporariamente
    orr r2, r2, #0xc0 @ setar bits I = 1 e F = 1 (unable interrupt)
    stmfd r0!, {r2} @ guarda o cpsr em linhaA
    msr cpsr_c, r2 @ volta para o modo anterior

    LDMFD sp!, {r0, r2} @ recupera r0 e r2

    STMFD sp!, {r0 - r3, LR} @Empilha os registradores

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

    STMFD sp!, {pc} @ salva pc na pilha do modo de interrupções

    BLNE handler_timer @vai para o rotina de tratamento da interrupção de timer
    LDMFD sp!, {r0 - r3, lr} @retorna

```



```
sub lr, lr, #4    @ corrigindo o lr
STMFD sp!, {lr}
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, lr
```

main:

```
bl timer_init @initialize interrupts and timer 0
```

stop:

```
BL print_loop
```

```
b stop
```