

# Atividade de Laboratório 10

## Objetivos

Nesta atividade você começará a desenvolver código a nível de sistema e se familiarizará com o conceito de interrupções.

## Introdução

Na parte 3 do curso de MC404 você desenvolverá habilidades para fazer programas a nível de sistema para a plataforma ARM. Entende-se programas a nível de sistema como códigos que possuem os mesmos privilégios que um sistema operacional normalmente tem sobre a máquina que ele executa. Isso é bastante diferente dos programas desenvolvidos até agora, que executavam em modo de usuário e não tinham acesso direto aos periféricos do computador.

## Atividade

Neste laboratório, você deve escrever um pequeno programa em linguagem de montagem do ARM na modalidade código de sistema. Isso significa que o seu programa não depende mais de um sistema operacional para executar e possui total controle sobre o processador. Além disso, ele tem liberdade para configurar os periféricos de *hardware* e tratar as interrupções geradas por eles.

Seu objetivo é escrever código que se inicie no endereço 0 de memória, que é o primeiro código ARM executado quando o processador é ligado, configurar um dispositivo de *hardware* chamado GPT (*General Purpose Timer*) para gerar interrupções a cada 100 ciclos do relógio (*clock*) de periféricos e tratar tais interrupções. A cada interrupção gerada, você deve incrementar um contador, que deve ser armazenado na primeira posição da seção `.data` do código do sistema, ou seja, no endereço `0x77801800`. Dessa maneira, você criará a base da funcionalidade de um relógio de sistema.

### Programação a nível de sistema

O diagrama abaixo mostra como o espaço de endereçamento físico é mapeado nos periféricos do sistema. Por exemplo, os endereços físicos na faixa `7000 0000 - 8000 0000` são mapeados para a memória principal do sistema; uma memória RAM DDR2 que se situa fora do *chip* do processador.

ARM CORE	AMBA	0000 0000 - 0001 0000	Memória RAM interna de boot
		0FFF C000 - 0FFF FFFF	TZIC ( <i>TrustZone Interrupt Controller</i> )
		53F8 4000 - 53F8 400B	GPIO ( <i>General Purpose Input Output</i> )
		53FA 0000 - 53FA 3FFF	GPT ( <i>General Purpose Timer</i> )
		53FB C000 - 53FB FFFF	UART-1 (Interface serial)
		7000 0000 - 8000 0000	Memória RAM Off-Chip (DDR2)

É importante notar que os primeiros endereços do espaço de endereçamento (endereços `0x00` a `0x20`) são **reservados** para o vetor de interrupções. Quando ocorrer uma interrupção ou exceção no processador, o fluxo de execução será desviado para um desses endereços e o modo de operação do processador será modificado. O modo de execução atual é codificado no registrador CPSR e determina qual o subconjunto de registradores que é acessível pelas instruções. Veja abaixo:

Modo	Endereço de salto	Valor do LR_<modo>	Motivo do salto	Registradores visíveis
Undefined Instruction	04	PC + 4	Uma instrução inválida foi encontrada na posição de memória apontada por PC.	R15 R14_und R13_und R12 R11 R10 ... R1 R0 CPSR SPSR_und
Supervisor (interrupção por <i>software</i> )	08	PC + 4	Uma instrução SVC/SWI foi executada e está solicitando uma função do sistema operacional ( <i>syscall</i> ).	R15 R14_svc R13_svc R12 R11 R10 ... R1 R0 CPSR SPSR_svc

Abort (instrução)	0c	PC + 4	O processador tentou buscar uma instrução da memória, mas o barramento gerou um erro.	R15 R14_abt R13_abt R12 R11 R10 ... R1 R0 CPSR SPSR_abt
Abort (dado)	10	PC + 8	O processador tentou buscar um dado da memória, mas o barramento gerou um erro.	R15 R14_abt R13_abt R12 R11 R10 ... R1 R0 CPSR SPSR_abt
IRQ	18	PC + 8	Ocorreu uma <b>interrupção</b> de <i>hardware</i> .	R15 R14_irq R13_irq R12 R11 R10 ... R1 R0 CPSR SPSR_irq
FIQ	1c	PC + 8	Ocorreu uma <b>interrupção</b> de <i>hardware</i> (modo rápido - <i>Fast Interrupt Request</i> ).	R15 R14_fiq R13_fiq R12_fiq R11_fiq R10_fiq R9_fiq R8_fiq R7 R6 R5 ... R1 R0 CPSR SPSR_fiq

A tabela acima nos diz, por exemplo, que quando ocorrer uma interrupção de *hardware* comum, independentemente de qual instrução esteja sendo executada, e supondo que as interrupções estejam habilitadas, o processador irá saltar para o endereço **0x18** e irá trocar de modo. Lembre-se que o CPSR codifica, além de diversas *flags*, o modo atual do processador, se interrupções IRQ estão habilitadas e se interrupções FIQ estão habilitadas. No momento da troca de modo, o CPSR antigo é salvo no SPSR do novo modo. Também, o valor de PC + 8 é salvo no registrador de retorno R14\_irq (ou LR\_irq), e PC recebe 0x18 para concretizar o salto.

Note ainda que, uma vez no modo de operação IRQ, todas as instruções que acessam registradores, ao acessar os registradores R13 e R14 (SP e LR) irão enxergar uma versão diferente (R13\_irq e R14\_irq), automaticamente. Esse mecanismo é chamado de *banked registers* e é feito para evitar que o código que trata interrupções suje os registradores do código de usuário que estava executando quando a interrupção ocorreu. Entretanto, perceba que, no modo IRQ, os registradores R12 a R0 ainda são os mesmos que o do modo usuário. Portanto, você precisa salvá-los na pilha sempre que for alterar qualquer um desses. É importante notar que o tratador de interrupções não pode modificar a pilha do usuário, dessa forma, cada modo (incluindo o IRQ) deve ter sua própria pilha. Cabe ao programador (desenvolvedor do sistema) inicializar a pilha do modo IRQ. Para isso, programador deve chavear para o modo IRQ e iniciar o registrador SP (R13). O espaço reservado para a pilha deve ser escolhido de forma a não colidir com o espaço de memória utilizado pelo programa do usuário (pilha do usuário, código, dados, etc.).

Ao ligarmos o processador ou ativarmos o sinal de *reset*, o fluxo de execução é desviado para o endereço 0x0 e o modo de execução é modificado para SUPERVISOR (svc). Assim sendo, esta posição de memória deve conter código para tratar a operação de *reset*.

O modo USER é o modo que temos utilizado até então para execução dos programas de usuário. Ele é o único modo que possui restrições quanto às instruções que podem ser executadas - instruções que requerem modo privilegiado não podem ser executadas nesse modo. Por esse motivo, justifica-se a existência de outro modo do ARM não listado acima, o modo SYSTEM: tal modo não contém registradores próprios, ou seja, ele utiliza os mesmos que são visíveis no modo USER. Entretanto, o modo SYSTEM contém privilégios para executar qualquer instrução do ARM.

Quando uma exceção ocorre, as interrupções do tipo IRQ são sempre desabilitadas. É responsabilidade do programador reabilitá-las, caso ele opte por tornar o código supervisor propenso a sofrer interrupções.

## Como escrever seu código

Para escrever seu código, primeiramente coloque uma seção alocável denominada `iv` no endereço 0x0 e nessa seção, um vetor de interrupções. Veja o código abaixo como exemplo:

```
.org 0x0
.section .iv,"a"

_start:

interrupt_vector:

    b RESET_HANDLER
.org 0x18
    b IRQ_HANDLER
```

Note que temos, no endereço 0x0, um salto para `RESET_HANDLER` - esse será o tratador de *reset*. Do mesmo modo, no endereço 0x18 temos um salto para o tratador de interrupções IRQ - é dentro de `IRQ_HANDLER` que você irá incrementar o contador para fazer essa atividade. Note que, em cada endereço, apenas há um salto; não se pode ter mais instruções por endereço, pois, como já foi dito, os endereços de 0x0 a 0x20 são reservados, e a cada um está atribuído um evento.

No início da seção de dados do código de sistema (a seção `.data`), aloque uma variável para armazenar o número de interrupções que aconteceram. Esta variável deve ser iniciada com o valor 0 e incrementada a cada interrupção. A variável deve ser armazenada no primeiro endereço da seção `.data`, ou seja, no endereço `0x77801800`. Deve-se também inicializar o vetor de interrupções no coprocessador 15 do ARM; veja o exemplo de código abaixo, para saber como vai ficar o início de sua seção `.text`:

```
.org 0x100
.text

RESET_HANDLER:

    @ Zera o contador
    ldr r2, =CONTADOR @lembra-se de declarar esse contador em uma secao de dados!
    mov r0, #0
    str r0, [r2]

    @Faz o registrador que aponta para a tabela de interrupções apontar para a tabela interrupt_vector
    ldr r0, =interrupt_vector
    mcr p15, 0, r0, c12, c0, 0

    @ Ajustar a pilha do modo IRQ.
    @ Você deve iniciar a pilha do modo IRQ aqui. Veja abaixo como usar a instrução MSR para chavear de modo.
    @ ...

    @@@...continua tratando o reset
```

No código acima, veja que existe uma seção de dados com o seu contador, sendo ele zerado logo no início da seção de código. Após isso, já existe a rotina de tratamento de *reset*, e nessa rotina as primeiras 2 instruções carregam o vetor de interrupções no coprocessador 15.

Em seguida, ainda no tratador de *reset*, você deve enviar dados para o *hardware* GPT e configurá-lo para a tarefa de contar até 100 (decimal) e, quando chegar a este valor, gerar uma interrupção e voltar a contar do zero. Para tanto, uma referência importante é o *datasheet* do GPT, que pode ser encontrado em: [gpt.pdf](#) ([./IMX53-gpt.pdf](#)). Em especial, preste atenção aos registradores do GPT e de seus respectivos endereços absolutos na memória (veja a tabela na sexta página do *datasheet*). Você pode ler ou escrever em qualquer registrador do GPT através de instruções *load/store* que acessem tais endereços.

#### Para configurar o GPT:

1. Você deve escrever no registrador GPT\_CR (*control register*) o valor 0x00000041 que irá habilitá-lo e configurar o *clock\_src* para periférico. Isto significa que o contador irá contar a cada ciclo do relógio dos periféricos do sistema. Note que o relógio (*clock*) do processador é muito mais alto (~1GHz) do que o relógio dos periféricos (~200MHz).
2. Zere o prescaler (GPT\_PR) e coloque em GPT\_OCR1 o valor que você deseja contar. Quando o comparador do GPT determinar que a contagem se igualou ao conteúdo de GPT\_OCR1, uma interrupção do tipo *Output Compare Channel 1* será gerada.
3. Para demonstrar interesse nesse tipo específico de interrupção do GPT, grave 1 no registrador GPT\_IR. Isto irá habilitar a interrupção *Output Compare Channel 1*, que se inicia desligada.

Após configurar o GPT, você deve configurar o TZIC (*TrustZone Interrupt Controller*) e habilitar as interrupções no ARM. Apenas o GPT e o TZIC controlam diretamente a porta de interrupções do ARM. O GPT conecta-se ao controlador de interrupções e possui um número de interrupção associado. No caso da plataforma iMX, que é a plataforma simulada no simulador ARM da disciplina, **esse número é o 39**. É de responsabilidade do controlador de interrupções determinar se, dentre as várias interrupções que podem estar ocorrendo, essas devem interromper o processador ou não. Para tanto, você deve configurar o TZIC para se importar apenas com a interrupção 39, que é a interrupção

referente ao GPT - o TZIC deve simplesmente passá-la direto para o processador, sempre que ela ocorrer. Para uma análise mais profunda do TZIC, seu *datasheet* está disponível em: [tzic.pdf](#) (./IMX53-tzic.pdf). Abaixo temos um diagrama ilustrando o comportamento do TZIC, dos outros periféricos que solicitam interrupções e do *core* do ARM. A comunicação ocorre da esquerda para a direita. Os periféricos informam o TZIC que querem interromper o processador, e então o TZIC toma a decisão de bloquear ou não o processador.

GPT		ARM
UART-1	TZIC	CORE
...		

Além do TZIC, falta habilitar as interrupções no ARM. Para configurar o processador em um modo que pode ser interrompido, deve-se usar a instrução MSR (este documento ([http://www.altera.com/literature/third-party/archives/ddi0100e\\_arm\\_arm.pdf](http://www.altera.com/literature/third-party/archives/ddi0100e_arm_arm.pdf)) explica em detalhes a instrução). O modelo da instrução que será usada aqui é:

```
msr CPSR_c, #0x13 @ SUPERVISOR mode, IRQ/FIQ enabled
```

A instrução acima irá sobrescrever os *bits* de CONTROLE (c) do registrador CPSR com 00010011 (== 0x13). Tal máscara corresponde a dizer que as interrupções FIQ/IRQ estão HABILITADAS e o modo é SUPERVISOR. Informações detalhadas sobre os *bits* do CPSR podem ser encontradas nesse material ([http://simplemachines.it/doc/arm\\_inst.pdf](http://simplemachines.it/doc/arm_inst.pdf)).

O código-modelo abaixo demonstra como configurar o TZIC para esse laboratório, e também como habilitar as interrupções:

```

SET_TZIC:
    @ Constantes para os enderecos do TZIC
    .set TZIC_BASE,          0x0FFFC000
    .set TZIC_INTCTRL,      0x0
    .set TZIC_INTSEC1,      0x84
    .set TZIC_ENSET1,       0x104
    .set TZIC_PRIOMASK,     0xC
    .set TZIC_PRIORITY9,    0x424

    @ Liga o controlador de interrupcoes
    @ R1 <= TZIC_BASE

    ldr r1, =TZIC_BASE

    @ Configura interrupcao 39 do GPT como nao segura
    mov r0, #(1 << 7)
    str r0, [r1, #TZIC_INTSEC1]

    @ Habilita interrupcao 39 (GPT)
    @ reg1 bit 7 (gpt)

    mov r0, #(1 << 7)
    str r0, [r1, #TZIC_ENSET1]

    @ Configure interrupt39 priority as 1
    @ reg9, byte 3

    ldr r0, [r1, #TZIC_PRIORITY9]
    bic r0, r0, #0xFF000000
    mov r2, #1
    orr r0, r0, r2, lsl #24
    str r0, [r1, #TZIC_PRIORITY9]

    @ Configure PRIOMASK as 0
    eor r0, r0, r0
    str r0, [r1, #TZIC_PRIOMASK]

    @ Habilita o controlador de interrupcoes
    mov r0, #1
    str r0, [r1, #TZIC_INTCTRL]

    @instrucao msr - habilita interrupcoes
    msr CPSR_c, #0x13      @ SUPERVISOR mode, IRQ/FIQ enabled

```

Após habilitar o TZIC, a interrupção pode ser gerada a qualquer momento e o seu código irá saltar para o endereço 0x18 quando isso acontecer. Então você deve, nesse ponto do seu código, entrar em um laço infinito que aguarda a interrupção. Um exemplo de laço infinito é:

```

laco:
    b laco

```

Seu código agora oficialmente não faz mais nada até que uma interrupção aconteça. No entanto, note que uma vez que a interrupção acontece, você deve tratá-la através da rotina que foi cadastrada no endereço 0x18 (no nosso exemplo chamamos de IRQ\_HANDLER). Dentro de IRQ\_HANDLER, a primeira coisa a ser feita é gravar o valor 0x1 no registrador GPT\_SR, do GPT. Isso é necessário pois informa ao GPT que o processador já está ciente de que ocorreu a interrupção, e ele pode limpar a *flag* OF1. Se não for feito isso, o GPT irá continuar sinalizando ao processador que uma interrupção ocorreu.

Ainda em IRQ\_HANDLER, incremente seu contador na memória e retorne da interrupção, lembrando-se de que o valor atual de LR é PC + 8, e deve ser corrigido, subtraindo-se 4 dele antes do retorno. Note que o retorno, nesse caso, não é como o retorno de uma função. A instrução `mov pc, lr` irá retornar para o valor de LR\_irq, mas não irá voltar ao CPSR antigo (modo SUPERVISOR, habilitação de interrupções, etc), que está em SPSR\_irq. Para tanto, use a instrução

```

movs pc, lr

```

que retorna e volta CPSR ao valor correto que estava antes da interrupção.

## Compilando e testando seu programa

Para compilar e rodar seu programa, em geral deve-se seguir os passos do Laboratório 9. Contudo, agora não vamos mais usar o DummyOS - seu programa fará o papel do sistema operacional. Na montagem do cartão SD, observe que usamos um código de usuário denominado "faz\_nada". Esse código de fato só é usado pois o utilitário `mksd.sh` exige que se coloque código de usuário. Para gerar o "faz\_nada", crie um programa que possui o código

```
.text
    and r0,r0,r0
```

e compile/ligue esse executável, colocando a **flag** `-Tdata=0x77803000` no ligador! Abaixo, veja como compilar e testar o programa desse laboratório:

```
# Entre na pasta onde esta o fonte do programa

# Monte o seu ambiente
source /home/specg12-2/mc404/simulador/set_path_player.sh

# Para compilar e ligar o seu programa (note a diferença no ligador)
arm-eabi-as -g raXXXXXX.s -o raXXXXXX.o
arm-eabi-ld raXXXXXX.o -o raXXXXXX -g --section-start=.iv=0x778005e0 -Ttext=0x77800700 -Tdata=0x77801800 -e 0x778005e0

# Monte a imagem do cartão SD: (note que seu programa eh o S0 !!!)
mksd.sh --so raXXXXXX --user faz_nada

# Antes de iniciar o simulador, você precisa abrir uma sessão do Player em outro terminal
source /home/specg12-2/mc404/simulador/set_path_player.sh
player /home/specg12-1/mc404/simulador/simulador_player/worlds_mc404/simple.cfg

# Abra o simulador armsim_player com suporte ao GDB no primeiro terminal.
armsim_player --rom=/home/specg12-1/mc404/simulador/simulador_player/bin/dumboot.bin --sd=disk.img -g

# Abra um terceiro terminal e monte o seu ambiente
source /home/specg12-2/mc404/simulador/set_path_player.sh

# No terceiro terminal, conecte no simulador utilizando o GDB
arm-eabi-gdb raXXXXXX
(gdb) target remote localhost:5000      # conecte no simulador
(gdb) b raXXXXXX.s:<linha>              # substitua <linha> por uma linha dentro de IRQ_HANDLER
(gdb) c                                # execute ate encontrar um breakpoint
(gdb) p *(int)0x77801800                # quando parar no tratador de interrupcoes, imprima o conteudo do contador
(gdb) c
(gdb) ...
```

## Entrega e avaliação

Deve ser submetido apenas um arquivo denominado `raXXXXXX.s` (com XXXXXX sendo seu RA de 6 dígitos) no SuSy. A atividade está em <https://susy.ic.unicamp.br:9999/mc404abef/10ab> (<https://susy.ic.unicamp.br:9999/mc404abef/10ab>) ou <https://susy.ic.unicamp.br:9999/mc404abef/10ef> (<https://susy.ic.unicamp.br:9999/mc404abef/10ef>)

## Sumário dos *links* apresentados nesse laboratório

- Informações sobre a instrução `MSR`: <http://infocenter.arm.com/help/topic/com.arm.doc.ddi0214b/CHDEHACE.html> (<http://infocenter.arm.com/help/topic/com.arm.doc.ddi0214b/CHDEHACE.html>)
- Informações sobre os *bits* de CPSR: [http://simplemachines.it/doc/arm\\_inst.pdf](http://simplemachines.it/doc/arm_inst.pdf) ([http://simplemachines.it/doc/arm\\_inst.pdf](http://simplemachines.it/doc/arm_inst.pdf))
- *Datasheet* do GPT: [gpt.pdf](#) ([./IMX53-gpt.pdf](#))
- *Datasheet* do TZIC: [tzic.pdf](#) ([./IMX53-tzic.pdf](#))
- *Datasheet* do GPIO: [gpio.pdf](#) ([./IMX53-gpio.pdf](#))