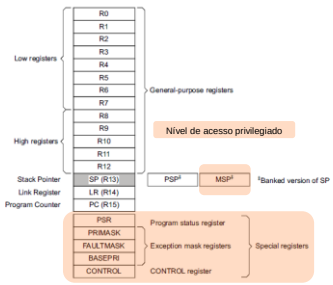
Exceções no ARM Cortex-M4

Prof. Hugo Vieira Neto
2020/1

Objetivo

- Estudar os principais conceitos do modelo de exceções no ARM Cortex-M4:
  - Modo thread e modo handler
  - Stacking, unstacking e lazy stacking
  - Tail-chaining, late arriving e POP preemption

Registradores



MSP = main SP
Kernel do S.O.
Exceções

PSP = process SP
Aplicações
(threads)

CONTROL
Seleciona SP
Seleciona NP

PRIMASK/BASEPRI
Mascaramento
de interrupções

Ponteiro de Pilha

- Pode-se trabalhar apenas com o MSP
- O SP corrente é acessado como R13 ou SP
- O SP sempre está alinhado em 32 bits (i.e. endereços múltiplos de 4)
- Instruções:
  - PUSH (empilha)
  - POP (desempilha)
- Full descending stack

Link Register

- Acessado como R14 ou LR
- Armazena o endereço de retorno de uma subrotina
- Deve ser salvo antes de se chamar outra subrotina

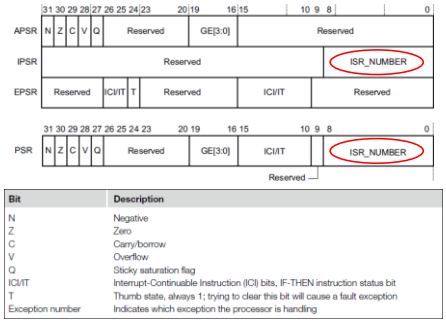
Contador de Programa

- Acessado como R15 ou PC
- O PC contém o endereço da próxima instrução a ser executada (motivo: pipeline)
- O endereço de uma instrução é sempre par (bit0 = 0)
- Bit0 do PC é usado para indicar modo Thumb
  - Em saltos, o bit 0 do PC deve ser sempre 1, caso contrário uma exceção será gerada.

Registradores Especiais

- Program Status Registers (xPSR):
  - Application Program Status Register (APSR) – RW
  - Interrupt Program Status Register (IPSR) – RO
  - Execution Program Status Register (EPSR) – RO
- Acesso por meio de instruções específicas:
  - MRS R0, APSR
  - MSR APSR, R0

Program Status Registers



Bit	Description
N	Negative
Z	Zero
C	Carry/borrow
V	Overflow
Q	Sticky saturation flag
ICI/IT	Interrupt-Continuable Instruction (ICI) bits, IF-THEN instruction status bit
T	Thumb state, always 1; trying to clear this bit will cause a fault exception
Exception number	Indicates which exception the processor is handling

Interrupt Program Status Register

- "ISR_NUMBER" é diferente do número da IRQ!

Bits	Name	Function
[31:6]	-	Reserved
[5:0]	Exception number	This is the number of the current exception: 0 = Thread mode 1 = Reserved 2 = NMI 3 = HardFault 4-10 = Reserved 11 = SVCall 12, 13 = Reserved 14 = PendSV 15 = SysTick 16 = IRQ0 . . .

Somar 16

Registradores Especiais

- Interrupt Mask Registers:
  - PRIMASK: usado para habilitar/desabilitar todas as interrupções e exceções (exceto NMI e HardFault)
  - FAULTMASK: usado para habilitar/desabilitar exceções de falta
  - BASEPRI: define o mascaramento de interrupções a partir de um limiar-base de prioridade
- Instruções MRS, MSR, CPSIE e CPSID (change processor state + interrupt enable/disable)

Mascaramento de Exceções

- BASEPRI = 0, PRIMASK = 0, FAULTMASK = 0
  - Nível de prioridade 256
  - Todas as exceções (prioridade < 256) são atendidas
- BASEPRI = X > 0, PRIMASK = 0, FAULTMASK = 0
  - Nível de prioridade X
  - Somente exceções com prioridade < X são atendidas
- PRIMASK = 1, FAULTMASK = 0
  - Nível de prioridade 0
  - Somente Reset, NMI e Faults (prioridade < 0) são atendidas
- PRIMASK = 1, FAULTMASK = 1
  - Nível de prioridade -1
  - Somente Reset e NMI (prioridade < -1) são atendidas

Registradores Especiais

- CONTROL
  - Bit 1: define o ponteiro de pilha (MSP ou PSP)
  - Bit 0: define o nível de acesso (Privilegiado ou Usuário)
- O bit 0 possui acesso de escrita apenas em nível privilegiado – uma vez em nível usuário, a única forma de retornar ao nível privilegiado é por meio de uma exceção.
- Acesso pelas instruções MRS e MSR
  - Usar instrução ISB (*Instruction Synchronization Barrier*) após a instrução MSR para uso imediato do novo ponteiro de pilha

Modos, Privilégios e Pilhas

- Thread Mode / Handler Mode
  - Modo Thread: execução normal de aplicações
  - Modo Handler: exceções ou interrupções
- Execução privilegiada / não privilegiada
  - Modo Thread: pode ser privilegiada ou não
  - Modo Handler: sempre privilegiada
- Main Stack / Process Stack
  - Ambas as pilhas possuem seu próprio ponteiro
  - Exceções sempre utilizam o MSP em modo handler
  - Aplicações (modo Thread) utilizam o MSP ou o PSP

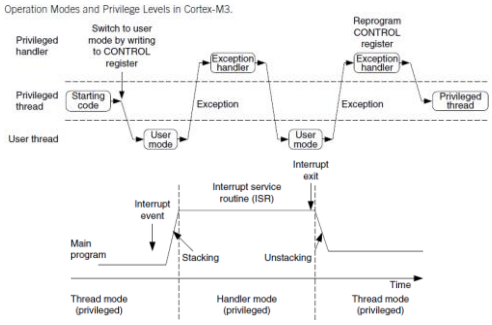
Modos, Privilégios e Pilhas

Modes (Thread out of reset)		Operations (privilege out of reset)	Stacks (Main out of reset)
	Handler - An exception is being processed	Privileged execution Full control	Main Stack Used by OS and Exceptions
	Thread - No exception is being processed - Normal code is executing	Privileged/Unprivileged	Main/Process

	Privileged	User
When running an exception handler	Handler mode (CONTROL[1] 0)	(not allowed)
When not running an exception handler (e.g., main program)	Thread mode (CONTROL[0] 0)	Thread mode (CONTROL[0] 1)

CONTROL[1] can be either 0 or 1

Modos e Privilégios



Privilégios

- No nível privilegiado o código tem acesso a **todos** os recursos.
- No nível usuário o código **não** pode:
  - Executar instruções como CPSIE e CPSID, que permitem alterar FAULTMASK e PRIMASK
  - Ter acesso à maioria dos registradores do System Control Block (SCB)

Exercício 1.1

- A partir da área de trabalho "EK-TM4C1294XL_IAR", selecione o projeto "simple_io_main_sp".
- Inspecione o arquivo "startup_TM4C1294.s":
  - O que a função `Reset_Handler` faz?
- Com o depurador parado no início da função `main`, verifique:
  - Quais registradores do processador possuem bits individualizados? Quais são os significados desses bits?
  - Qual é o estado (modo) do processador? Quais registradores foram usados para obter esta informação?

Exercício 1.2

- Coloque um breakpoint na primeira declaração da função `SysTick_Handler` e execute o programa até parar nele:
  - Qual é o valor do registrador LR?
  - Qual é o estado (modo) do processador?
- Altere o bit NMIPENDSET do registrador ICSR (System Control Block) e volte a executar o programa:
  - O que acontece?
  - Qual é o valor do registrador LR?
  - Qual é o estado (modo) do processador?

Exercício 1.3

- Selecione o projeto “simple_io_process_sp”.
- Inspecione o arquivo “startup_TM4C1294.s”:
  - O que a função `Reset_Handler` faz de adicional?
- Refaça os procedimentos do Exercício 1.2:
  - Qual é o valor do registrador CONTROL?
  - Houve alguma alteração nos valores do registrador LR?

Exceções e Interrupções

- Qualquer solicitação para mudança do fluxo normal de um programa:
  - Exceção: síncrona, p. ex. detecção de erro
  - Interrupção: assíncrona, p. ex. ocorrência de evento em periférico

Visão Geral – Cortex-M

- Nested Vector Interrupt Controller (NVIC)
  - Suporte a múltiplas fontes de interrupção
  - Tratamento eficiente de interrupções aninhadas
  - Arquitetura flexível (altamente configurável)
  - Suporte intrínseco a RTOS

Visão Geral – Cortex-M

- Arquitetura de interrupções com baixa latência
- Algumas instruções com múltiplos ciclos de execução podem ser interrompidas
- Entrada/saída de exceções controlada por hardware

Visão Geral – Cortex-M

- Entrada/saída de interrupções controlada por hardware
  - Salvamento e restauração de contexto realizado automaticamente
  - Tratamento de chegada tardia (late arriving) de interrupções com maior prioridade
  - Tratamento de interrupções pendentes sem restauração / salvamento completo de contexto (tail-chaining)

Estados das Exceções

- Inativo: nem pendente nem ativo
- Pendente: a exceção foi gerada, mas ainda não foi processada
- Ativo: o processamento da exceção foi iniciado, mas ainda não foi completado
  - O processamento de uma exceção pode interromper o processamento de outra – nesse caso as duas exceções estão no estado ativo
- Ativo e Pendente: a exceção está sendo processada e existe uma exceção pendente da mesma fonte

Tipos de Exceção

- Reset – prioridade -3 (modo thread)
- NMI – prioridade -2
- HardFault – prioridade -1
- Faults (MemManage, Bus, Usage)
- SVC – causada pela instrução SVC
- PendSV – serviço pendente (p. ex. chaveamento de contexto depois de interrupção)
- SysTick – causada periodicamente pelo SysTick
- IRQn – requisição de interrupção por periférico

Tipos de Exceção de Falta

- MemManageFault: faltas no acesso a memória detectadas pela MPU – se desabilitada, escala para HardFault
- BusFault: outros tipos de falta no barramento de memória que não as do tipo MemManage
- UsageFault: faltas não relacionadas ao barramento de memória (p. ex. instrução indefinida ou estado inválido)

Propriedades das Exceções

Properties of the different exception types

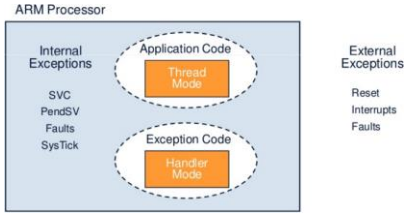
Exception number[1]	IRQ number[1]	Exception type	Priority	Vector address or offset[2]	Activation
1	-	Reset	-3, the highest	0x00000004	Asynchronous
2	-14	NMI	-2	0x00000008	Asynchronous
3	-13	Hard fault	-1	0x0000000C	-
4	-12	Memory management fault	Configurable[3]	0x00000010	Synchronous
5	-11	Bus fault	Configurable[3]	0x00000014	Synchronous when precise, asynchronous when imprecise
6	-10	Usage fault	Configurable[3]	0x00000018	Synchronous
7-10	-	-	-	Reserved	-
11	-5	SVCall	Configurable[3]	0x0000002C	Synchronous
12-13	-	-	-	Reserved	-
14	-2	PendSV	Configurable[3]	0x00000038	Asynchronous
15	-1	SysTick	Configurable[3]	0x0000003C	Asynchronous
16 and above	0 and above	Interrupt (IRQ)	Configurable[4]	0x00000040 and above[5]	Asynchronous

Definições de Prioridades

- Prioridades das exceções do núcleo são definidas no System Control Block (SCB)
  - SHPRn (System Handler Priority Registers)
- Prioridades das interrupções dos periféricos são definidas no Nested Vectored Interrupt Controller (NVIC)
  - IPRn (Interrupt Priority Registers)

Eventos Causadores

- Exceções podem ser causadas por eventos que são internos ou externos ao processador



Estado do Processador

- Modo thread após Reset
- Modo handler após qualquer outro tipo de exceção

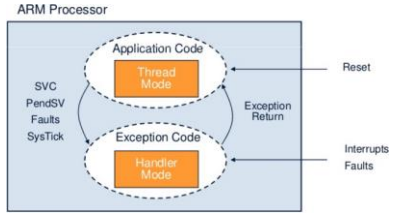


Tabela de Vetores de Exceção

Address		Exception #
0x40 + 4*N	External N	16 + N
...	...	...
0x40	External 0	16
0x3C	SysTick	15
0x38	PendSV	14
0x34	Reserved	13
0x30	Debug Monitor	12
0x2C	SVC	11
0x1C to 0x28	Reserved (x4)	7-10
0x18	Usage Fault	6
0x14	Bus Fault	5
0x10	Mem Manage Fault	4
0x0C	Hard Fault	3
0x08	NMI	2
0x04	Reset	1
0x00	Initial Main SP	N/A

Ver tabela no arquivo startup_TM4C1294.s!

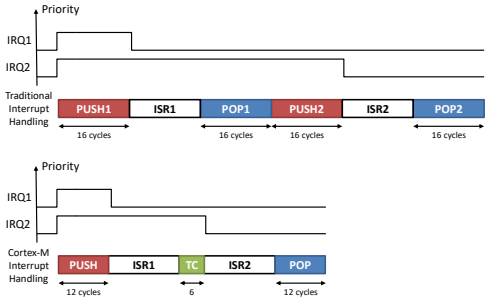
Definições Importantes

- Preempção: uma exceção de maior prioridade pode causar preempção de uma rotina de tratamento (handler) em execução
- Retorno: ocorre ao término da execução da rotina de tratamento (handler), caso
  - Não exista exceção pendente com prioridade suficiente para ser atendida (tail chaining)
  - A rotina que terminou não seja de uma exceção com chegada tardia (late arriving)

Definições Importantes

- Tail-chaining (encadeamento): se ao término da execução de uma handler existir uma exceção pendente apta a ser tratada, então não se executa a sequência de restauração de contexto (unstacking) seguida de novo salvamento de contexto (stacking).

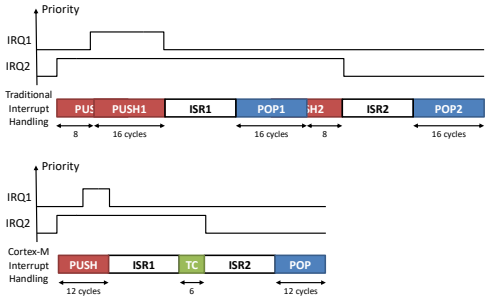
Exemplo de Tail-chaining



Definições Importantes

- Late arriving (chegada tardia): mecanismo que acelera preempção caso uma exceção de maior prioridade ocorra durante o salvamento de contexto (stacking) – nesse caso a exceção de maior prioridade é atendida primeiro e em seguida, por tail-chaining, a exceção de menor prioridade.

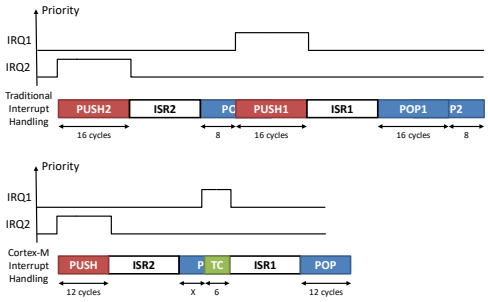
Exemplo de Late Arriving



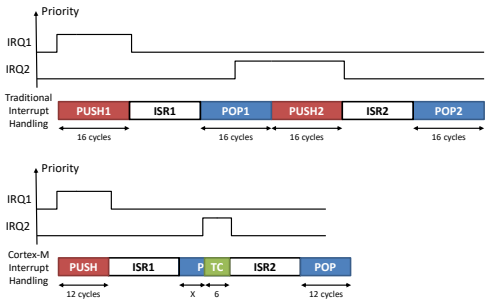
Definições Importantes

- POP preemption (abandono de saída): mecanismo que acelera o atendimento de uma exceção que ocorra durante o restauração de contexto (unstacking) – nesse caso o processo de stacking é abortado e seguido pelo processo de tail-chaining para atendimento da nova exceção.

Exemplo de POP Preemption (1)



Exemplo de POP Preemption (2)



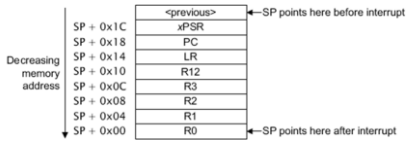
Atendimento de Exceções

- Ocorre quando existe uma exceção no estado pendente com prioridade suficiente* e
  - O processador está em modo thread, ou
  - Essa exceção pendente possui prioridade maior que a que uma exceção que já esteja sendo atendida (preempção em modo handler)

*Prioridade suficiente significa prioridade maior do que o limiar em BASEPRI

Atendimento de Exceções

- Ocorre salvamento automático de contexto (stacking) na entrada, bem como recuperação automática de contexto (unstacking) na saída
- Cortex-M4 (sem FPU):



Atendimento de Exceções

- LR := EXC_RETURN (próximos slides)
- O processador obtém o endereço da rotina de tratamento (handler) da tabela de vetores
- Após o stacking, a execução desvia para a rotina de tratamento e a exceção passa ao estado ativo
- Se ocorrer outra exceção de maior prioridade durante o stacking (condição de late-arriving), então esta será atendida com antecedência

EXC_RETURN – Cortex-M4

EXC_RETURN[31:0]	Description
0xFFFF.FFEE - 0xFFFF.FFF0	Reserved
0xFFFF.FFF1	Return to Handler mode. Exception return uses non-floating-point state from **MSP**. Execution uses **MSP** after return.
0xFFFF.FFF2 - 0xFFFF.FFF8	Reserved
0xFFFF.FFF9	Return to Thread mode. Exception return uses non-floating-point state from **MSP**. Execution uses **MSP** after return.
0xFFFF.FFFA - 0xFFFF.FFFC	Reserved
0xFFFF.FFFD	Return to Thread mode. Exception return uses non-floating-point state from **PSP**. Execution uses **PSP** after return.
0xFFFF.FFFE - 0xFFFF.FFFF	Reserved

EXC_RETURN – Cortex-M4F

EXC_RETURN[31:0]	Description
0xFFFF.FFE0	Reserved
0xFFFF.FFE1	Return to Handler mode. Exception return uses floating-point state from **MSP**. Execution uses **MSP** after return.
0xFFFF.FFE2 - 0xFFFF.FFE8	Reserved
0xFFFF.FFE9	Return to Thread mode. Exception return uses floating-point state from **MSP**. Execution uses **MSP** after return.
0xFFFF.FFEA - 0xFFFF.FFEC	Reserved
0xFFFF.FFED	Return to Thread mode. Exception return uses floating-point state from **PSP**. Execution uses **PSP** after return.
0xFFFF.FFEE - 0xFFFF.FFF0	Reserved

Retorno de Exceções

- Ocorre quando uma instrução escreve no PC um dos valores de EXC_RETURN
- Instruções usadas para retorno de exceção:
  - LDR PC, ...
  - LDM/POP incluindo PC
  - BX LR (preferencial)

Exercício 2.1

- A partir da área de trabalho “EK-TM4C1294XL_IAR”, selecione o projeto “simple_io_main_sp”.
- Coloque um breakpoint na primeira **instrução** da função `SysTick_Handler` (janela Disassembly) e execute o programa até parar nele:
  - Qual é o conteúdo do topo da pilha (primeiras 8 double words)?
  - Compare o conteúdo do topo da pilha (primeiras 8 double words) ao conteúdo dos registradores do núcleo. Qual é a conclusão?

Exercício 2.2

- Refaça o Exercício **1.2** alterando a função main do projeto “simple_io_main_sp” para usar a FPU.
  - Houve alguma alteração nos valores do registrador LR?
  - Qual ponteiro de pilha está sendo usado? Verifique na janela Registers do depurador (banco “CPU Registers”).
- Obs: a FPU será usada caso exista alguma operação com variáveis do tipo float e a macro `__FPU_USED` estiver definida (Options → C/C++ Compiler → Preprocessor).

Exercício 2.3

- Refaça o Exercício **1.3** alterando a função main do projeto “simple_io_process_sp” para usar a FPU.
  - Houve alguma alteração nos valores do registrador LR?
  - Qual ponteiro de pilha está sendo usado? Verifique na janela Registers do depurador (banco “CPU Registers”).
  - Qual é a diferença entre o banco “Current CPU Registers” e o banco “CPU Registers” do depurador?

Atividade Extra-classe

- Leitura:
  - Nota de aplicação: “Cortex-M4(F) Lazy Stacking and Context Switching” (Seções 1 e 2)
  - Livro: “The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors” (Seção 13.3)
  - Website: https://www.keil.com/pack/doc/CMSIS/General/html/index.html#CM_Pack_Content (Introdução)