Esse é o sexto artigo da série escrita pelo engenheiro Ismael Lopes da Silva, exclusivamente para o site "www.embarcados.com.br". Nessa série focarei no Microcontrolador da STMicroelectronics, o MCU STM32F103C8T6, que é um ARM Cortex-M3. Os pré-requisitos para uma boa compreensão dos artigos é ter o domínio da Linguagem C Embedded e conceitos de eletrônica.

Níveis de Acesso ao Processador ARM Cortex-M3

O Processador ARM Cortex-M3 opera em dois níveis diferentes de acesso, que são:

- ✔ Nível de Acesso Privilegiado (PAL);
- ✔ Nível de Acesso Sem-Privilégios (NPAL).

Se seu código estiver rodando com PAL, então, seu código tem total acesso aos recursos do processador e aos Registradores restritos. Se seu código estiver rodando com NPAL, seu código não terá acesso somente aos Registradores restritos do processador.

O processador em modo de operação Thread pode rodar nos dois níveis de acesso, PAL e NPAL. O modo de operação Handler sempre roda somente em PAL. Quando se inicia uma aplicação o processador por padrão roda no modo de operação Thread e com nível de acesso PAL. Conseguimos modificar o nível de acesso através do Registrador CONTROL.

Em modo de operação Thread é possível mudar o nível de acesso. Fazemos essa mudança através do código. Quando isso ocorrer, ainda em modo Thread, você não consegue voltar o nível de acesso para PAL, porque há restrições de acesso que não permitirá acessar os Registradores restritos.

Para conseguir voltar o nível de acesso para PAL, você deve levar o modo de operação do processador para Handler, então, como nesse modo o nível de acesso sempre será PAL, você consegue através do código modificar Registradores restritos, e quando voltar para o modo de operação Thread, o nível de acesso PAL é recuperado.

Os níveis de privilégios para execução de software, são:

Sem privilégios (NPAL):

- ✔ Possui acesso limitado às instruções MSR e MRS e não pode usar a instrução CPS;
- ✔ Não pode acessar o Temporizador do Sistema (SysTick), NVIC ou Bloco de Controle do Sistema (System Control Block);
- ✔ Pode ter acesso restrito à memória ou periféricos.

Privilegiado (PAL):

- ✔ O software pode usar todas as instruções e ter acesso a todos os recursos.

Entendendo o Registrador CONTROL do Core (núcleo) dos Registradores

No documento ARM Generic User Guide Cortex-M3 (DUI0552A), disponibilizado no repositório do Github, na página 16, podemos ver todos os Registradores do processador ARM Cortex-M3. Todos os Registradores são de 32 bits.

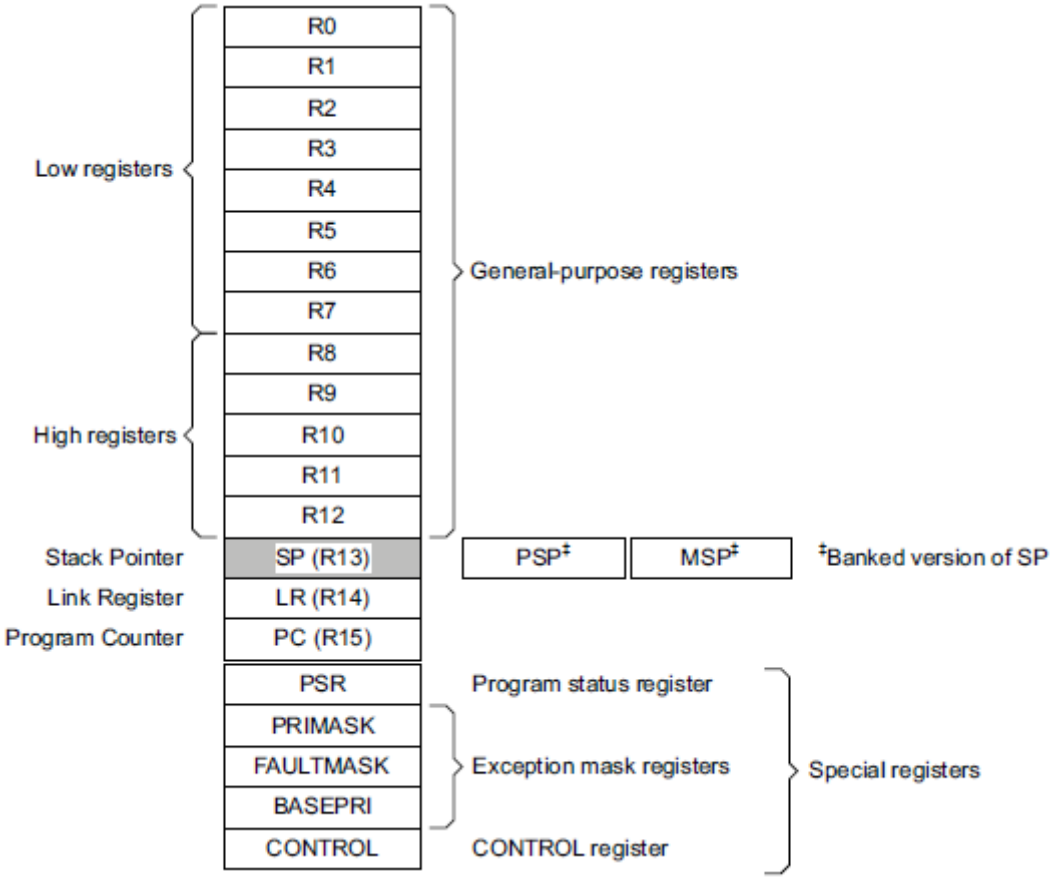

Figura 1 – Registradores do processador ARM Cortex-M3

Nesse artigo o foco é o Registrador CONTROL, que como podemos ver na figura 1, faz parte do grupo de Registradores especiais, portanto, dependendo do nível de acesso serão acessíveis ou não. Na figura 2, ilustra os bits desse Registrador e o bit nPRIV [0]. Na página 22 do documento ARM Generic User Guide Cortex-M3 (DUI0552A) você tem os detalhes.

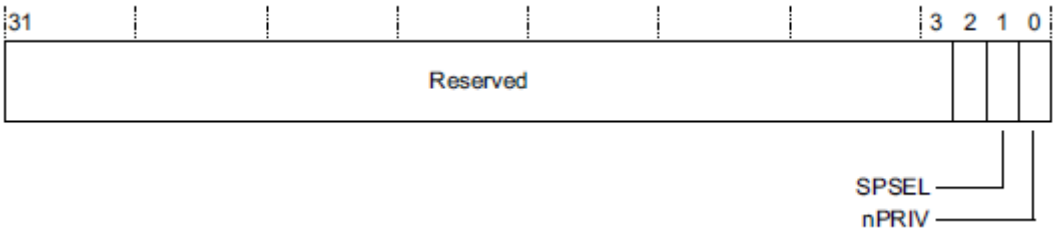

Figura 2 – Registrador CONTROL

[0] nPRIV Define o nível de acesso no modo Thread:

- ✔ 0 = Privileged (PAL);
- ✔ 1 = Unprivileged (NPAL).

Nível de Acesso do Processador ARM Cortex-M3 na Prática

Segue o arquivo main.c, que é uma aplicação para verificarmos o comportamento do processador quando estamos em modo Thread e sem privilégios (NPAL), e tentarmos escrever em um Registrador especial. Os comentários auxiliam no entendimento do código, mas, o foco não é explicar código em linguagem C, mas, entender nível de acesso ao processador.

```
/******************************************************************************
  * @file    main.c
  * @author  Auto-generated by STM32CubeIDE
  * @version V1.0
  * @brief   Default main function.
  ******************************************************************************/

#if !defined(__SOFT_FP__) && defined(__ARM_FP)
  #warning "FPU is not initialized, but the project is compiling for an FPU. Please initialize the FPU before use."
#endif

#include<stdio.h>
#include<stdint.h>

// Função é executada em modo Thread
void gera_interrupcao()
{
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;
        // Habilita a interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);
        // Gera uma interrupção de software para IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}

// Função é escrita em assembly para acessar o Registrador especial CONTROL
// O acesso é indireto, via Registradores de Proposito Geral
void muda_nivel_acesso(void)
{
        // Ler o nivel de acesso
        // Lê o conteúdo de CONTROL para R0
        __asm volatile ("MRS R0,CONTROL");
        // Modificar o nivel de acesso
        // Com o conteúdo de R0, faz uma OR com o dado imediato #1, para setar o bit mPRIV
        __asm volatile ("ORR R0,R0,#0x01");
        // Escrever o nivel de acesso
        // Escreve o conteúdo do R0 para o CONTROL, seta o bit mPRIV = 1 = NPAL
        __asm volatile ("MSR CONTROL,R0");
}

// Função inicialmente é executada em modo Thread e nivel de acesso = PAL = mPRIV = 0
int main(void)
```

```
{
	printf("Em modo Thread: antes da interrupcao\n");
	muda_nivel_acesso();
	gera_interrupcao();
	printf("Em modo Thread: depois da interrupcao\n");
	for(;;);
}

// Função é executada em modo Handler
void RTC_IRQHandler(void)
{
	printf("Em modo Handler: Tratando a Rotina do Servico de Interrpcao\n");
}

// Função é executada em modo Handler
// Uma falha é gerada ao tentar escrever em um Registrador com nivel de acesso NPAL
void HardFault_Handler(void)
{
	printf("Falha detectada\n");
	// Ler o nivel de acesso
	// Lê o conteúdo de CONTROL para R0
	__asm volatile ("MRS R0,CONTROL");
	// Modificar o nivel de acesso
	// Com o conteúdo de R0, faz uma AND com o dado imediato #0, para resetar o bit mPRIV
	__asm volatile ("AND R0,R0,#0x00");
	// Escrever o nivel de acesso
	// Escreve o conteúdo do R0 para o CONTROL, reseta o bit mPRIV = 0 = PAL
	__asm volatile ("MSR CONTROL,R0");
}
```

Nó código pode ser visto que para mudar nível de acesso usamos uma função com instruções em linguagem Assembly, porque para ler ou escrever no Registrador especial CONTROL, isso deve ser realizado indiretamente através dos Registradores de Propósito Geral.

Usando o STM32CubeIDE, no mesmo workspace que criamos no artigo 3, então, vamos copiar o projeto “02ModoProcessador” como “03NivelAcesso”. Na janela “Projetc Explorer”, clique com o botão direito do mouse sobre o projeto “02ModoProcessador” e selecione “Copy”. Novamente na janela “Projetc Explorer”, clique com o botão direito do mouse sobre o projeto “02ModoProcessador” e selecione “Paste”. Uma janela para renomear a aplicação será mostrada, portanto, entre como o nome “03NivelAcesso”. Depois clique no botão [Copy].

Fizemos uma cópia porque tudo que preparamos é mantido, então, vamos apenas editar o arquivo main.c, conforme disponibilizado nesse artigo. Apenas temos que refazer a configuração do depurador, para habilitar o SWV (Serial Wire Viewer), já detalhado em outro artigo.

Como nessa aplicação continuamos usando a função “printf”, então, o processo de depuração precisa captar as mensagens da aplicação, então, temos que manter o uso da janela “SWV ITM Data Console”. Para mais detalhes vejam os arquivos anteriores.

Como temos uma nova aplicação “03NivelAcesso”, novo arquivo main.c, então, vamos dar um “Clean Project”, “Build Project” e ative a perspectiva de depuração. Quando iniciar a depuração

(debugging), habilite janela “SWV ITM Data Console”, ative a porta 0, para capturar as mensagens enviadas pelas funções “printf”.

Monitorar o Nível de Acesso do Processador na Prática

Com a aplicação está sendo depurada (debugging), na STM32CubeIDE, insira um breakpoint na linha do programa, conforme mostrado a seguir:

```
void HardFault_Handler(void)
        {
●               printf("Falha detectada\n");
                // Ler o nivel de acesso
                // Lê o conteúdo de CONTROL para R0
                __asm volatile ("MRS R0,CONTROL");
                // Modificar o nivel de acesso
                // Com o conteúdo de R0, faz uma AND com o dado imediato #0, para resetar o bit mPRIV
                __asm volatile ("AND R0,R0,#0x00");
                // Escrever o nivel de acesso
                // Escreve o conteúdo do R0 para o CONTROL, reseta o bit mPRIV = 0 = PAL
                __asm volatile ("MSR CONTROL,R0");

        }
```

Mas, como estamos iniciando a depuração, então, a linha atual de depuração é a seguinte:

```
int main(void)
{
        printf("Em modo Thread: antes da interrupcao\n");
        muda_nivel_acesso();
        gera_interrupcao();
        printf("Em modo Thread: depois da interrupcao\n");
        for(;;);
}
```

Temos que monitorar o Registradores CONTROL, portanto, temos que visualizar a janela chamada “Registers”. No menu “Window”, parte superior da tela, selecione “Show View” e “Registers”. A janela “Registers” será mostrada, conforme já detalhado em outro artigo. Selecione o Registrador CONTROL, que é o Registrador onde monitoraremos o bit [0] = mPRIV.

Voltando a depuração, estamos parado na primeira linha da função main.c. Na janela “Registers”, quando selecionamos o Registrador CONTROL seu conteúdo é mostrado logo abaixo, conforme a seguir:

Name: CONTROL
        Hex:0x0
        Decimal:0
        Octal:0
        Binary:0 (nível de acesso PAL, mPRIV = 0)
        Default:0

Os valores são mostrados em vários formatos numéricos, portanto, vamos apenas analisar o formato binário, que nos mostrará em detalhes o conteúdo do bit [0]. Inicialmente o processador em nível de acesso PAL, o que é confirmado porque CONTROL [0] = 0.

1) Agora vamos pressionar a tecla [F5], que é um "step into", onde caminharemos linha por linha no processo de depuração. Nesse momento focaremos apenas no CONTROL [0], para acompanhar o nível de acesso do processador em cada parte da aplicação.

```
int main(void)
{
        printf("Em modo Thread: antes da interrupcao\n");
        muda_nivel_acesso();
        gera_interrupcao();
        printf("Em modo Thread: depois da interrupcao\n");
        for(;;);
}
```

Name: CONTROL
Binary:0 (nível de acesso PAL, mPRIV = 0)

2) Vamos dar mais um passo, pressionando a tecla [F5].

```
void muda_nivel_acesso(void)
{
        // Ler o nivel de acesso
        // Lê o conteúdo de CONTROL para R0
        __asm volatile ("MRS R0,CONTROL");
        // Modificar o nivel de acesso
        // Com o conteúdo de R0, faz uma OR com o dado imediato #1, para setar o bit mPRIV
        __asm volatile ("ORR R0,R0,#0x01");
        // Escrever o nivel de acesso
        // Escreve o conteúdo do R0 para o CONTROL, seta o bit mPRIV = 1 = NPAL
        __asm volatile ("MSR CONTROL,R0");
}
```

Name: CONTROL
Binary:0 (nível de acesso PAL, mPRIV = 0)

3) Vamos dar mais um passo, pressionando a tecla [F5].

```
void muda_nivel_acesso(void)
{
        // Ler o nivel de acesso
        // Lê o conteúdo de CONTROL para R0
        __asm volatile ("MRS R0,CONTROL");
        // Modificar o nivel de acesso
        // Com o conteúdo de R0, faz uma OR com o dado imediato #1, para setar o bit mPRIV
        __asm volatile ("ORR R0,R0,#0x01");
        // Escrever o nivel de acesso
```

```
        // Escreve o conteúdo do R0 para o CONTROL, seta o bit mPRIV = 1 = NPAL
        __asm volatile ("MSR CONTROL,R0");
}
```

Name: CONTROL
Binary:0 (nível de acesso PAL, mPRIV = 0)

4) Vamos dar mais um passo, pressionando a tecla [F5].

```
void muda_nivel_acesso(void)
{
        // Ler o nivel de acesso
        // Lê o conteúdo de CONTROL para R0
        __asm volatile ("MRS R0,CONTROL");
        // Modificar o nivel de acesso
        // Com o conteúdo de R0, faz uma OR com o dado imediato #1, para setar o bit mPRIV
        __asm volatile ("ORR R0,R0,#0x01");
        // Escrever o nivel de acesso
        // Escreve o conteúdo do R0 para o CONTROL, seta o bit mPRIV = 1 = NPAL
        __asm volatile ("MSR CONTROL,R0");
}
```

Name: CONTROL
Binary:0 (nível de acesso PAL, mPRIV = 0)

5) Vamos dar mais um passo, pressionando a tecla [F5].

```
void muda_nivel_acesso(void)
{
        // Ler o nivel de acesso
        // Lê o conteúdo de CONTROL para R0
        __asm volatile ("MRS R0,CONTROL");
        // Modificar o nivel de acesso
        // Com o conteúdo de R0, faz uma OR com o dado imediato #1, para setar o bit mPRIV
        __asm volatile ("ORR R0,R0,#0x01");
        // Escrever o nivel de acesso
        // Escreve o conteúdo do R0 para o CONTROL, seta o bit mPRIV = 1 = NPAL
        __asm volatile ("MSR CONTROL,R0");
}
```

Name: CONTROL
Binary:1 (nível de acesso NPAL, mPRIV = 1)

ATENÇÃO! O nível de acesso mudou para NPAL, porém, nenhuma falha subiu porque ainda não infringimos nenhum acesso restrito.

6) Vamos dar mais um passo, pressionando a tecla [F5].

```
int main(void)
```

```
{
        printf("Em modo Thread: antes da interrupcao\n");
        muda_nivel_acesso();
        gera_interrupcao();
        printf("Em modo Thread: depois da interrupcao\n");
        for(;;);
}
```

Name: CONTROL
Binary:1 (nível de acesso NPAL, mPRIV = 1)

ATENÇÃO! O nível de acesso em NPAL, porém, nenhuma falha subiu porque ainda não infringimos nenhum acesso restrito.

7) Vamos dar mais um passo, pressionando a tecla [F5].

```
void gera_interrupcao()
{
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;
        // Habilita a interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);
        // Gera uma interrupção de software para IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}
```

Name: CONTROL
Binary:1 (nível de acesso NPAL, mPRIV = 1)

ATENÇÃO! O nível de acesso em NPAL, porém, nenhuma falha subiu porque ainda não infringimos nenhum acesso restrito.

8) Vamos dar mais um passo, pressionando a tecla [F5].

```
void gera_interrupcao()
{
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;
        // Habilita a interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);
        // Gera uma interrupção de software para IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}
```

Name: CONTROL
Binary:1 (nível de acesso NPAL, mPRIV = 1)

ATENÇÃO! O nível de acesso em NPAL, porém, nenhuma falha subiu porque ainda não infringimos nenhum acesso restrito.

9) Vamos dar mais um passo, pressionando a tecla [F5].

```
void gera_interrupcao()
{
        uint32_t *pNVIC_ISER0 = (uint32_t*)0xE000E100;
        uint32_t *pNVIC_STIR  = (uint32_t*)0xE000EF00;
        // Habilita a interrupção IRQ3
        *pNVIC_ISER0 |= (1 << 3);
        // Gera uma interrupção de software para IRQ3
        *pNVIC_STIR = (3 & 0x1FF);
}
```

Name: CONTROL
Binary:1 (nível de acesso NPAL, mPRIV = 1)

ATENÇÃO! O nível de acesso em NPAL, porém, nenhuma falha subiu porque ainda não infringimos nenhum acesso restrito.

10) Vamos dar mais um passo, pressionando a tecla [F5]. Atenção nesse passo!

```
void HardFault_Handler(void)
{
        printf("Falha detectada\n");
        // Ler o nivel de acesso
        // Lê o conteúdo de CONTROL para R0
        __asm volatile ("MRS R0,CONTROL");
        // Modificar o nivel de acesso
        // Com o conteúdo de R0, faz uma AND com o dado imediato #0, para resetar o bit mPRIV
        __asm volatile ("AND R0,R0,#0x00");
        // Escrever o nivel de acesso
        // Escreve o conteúdo do R0 para o CONTROL, reseta o bit mPRIV = 0 = PAL
        __asm volatile ("MSR CONTROL,R0");
}
```

Name: CONTROL
Binary:1 (nível de acesso NPAL, mPRIV = 1)

ATENÇÃO! O nível de acesso em NPAL, e subiu uma falha porque infringimos uma regra de acesso restrito, ao tentarmos escrever no Registrador NVIC_ISER0.

11) Podemos interromper a depuração, porque já caminhamos pelos trechos de código que nos interessava. Vimos o comportamento do nível de acesso conforme os conceitos passados nesse artigo.

Para finalizar colocaremos novamente em processo de depuração, e visualizar a saída da janela "SWV ITM Data Console", e apenas ver as mensagens capturadas. Destaque a janela "SWV ITM Data Console", "start trace" e pressione a tecla [F8], para resumir a depuração. A depuração rodará até o breakpoint, então, pressione novamente a tecla [F8] para concluir a depuração. Para finalizar interrompa o processo de depuração.

Figura 3 - Janela "SWV ITM Data Console" e as mensagens capturadas

As mensagens capturadas são mostradas a seguir, e podemos confirmar tudo que vimos em detalhes no passo a passo.

- ✔ Em modo Thread: antes da interrupcao
- ✔ Falha detectada
- ✔ Em modo Handler: Tratando a Rotina do Servico de Interrpcao
- ✔ Em modo Thread: depois da interrupcao