Capítulo 3

Processadores ARM

3.1. Introdução aos processadores ARM

Atualmente, o termo ARM se refere a várias famílias de arquiteturas *RISC* (*Reduced Instruction Set Computer*) e a várias famílias de núcleos completos de CPUs produzidas por muitos fabricantes de processadores e microcontroladores.

Originalmente Acorn RISC Machine, e depois Advanced RISC Machine, a arquitetura ARM é desenvolvida pela empresa britânica *ARM Holdings,* vendida em 2020 para a NVidia. A ARM não fabrica os seus próprios chips, apenas desenvolve o projeto e licencia o uso de suas CPUs para que outros fabricantes (como *Cirrus Logic*, *ST Microelectronics*, *NXP*, *Texas Instruments*, *Atmel*, *Samsung*, *Sharp*, *Qualcomm*, *Nvidia*, etc) possam implementar em seus próprios produtos. Depois que uma empresa licencia o design do processador, a ARM fornece o código-fonte do projeto em uma linguagem chamada *System Verilog-HDVL* (*Hardware Description and Verification Language*). Os engenheiros de design dessas empresas adicionam seus próprios blocos funcionais, como periféricos e memórias, e usam várias ferramentas de software para converter todo o projeto em um layout de um chip, desenvolvendo melhores periféricos, diminuindo o consumo de energia das memórias e adicionando seus próprios segredos industriais para tentar tornar seus produtos melhores do que outros no mercado. Além disso, eles também precisam desenvolver exemplos de software e materiais de suporte para facilitar o uso dos chips pelos projetistas de produtos embarcados. A empresa *ARM Holdings* também fornece outros produtos de Propriedade Intelectual (*IP – Intellectual Property*) que também podem ser usados por essas empresas em seus produtos, como bibliotecas para o projeto de portas lógicas, memórias, barramentos de comunicações e componentes de depuração (*debug*). A empresa ARM baseia seu sucesso em um design original simples e poderoso, que até hoje continua a melhorar através de constante inovação tecnológica. Os principais conceitos por trás da arquitetura ARM são a simplicidade da filosofia *RISC*, baixo custo e baixo consumo.

Os processadores ARM são encontrados em vários segmentos de mercado, incluindo redes, automotivo, dispositivos móveis e de consumo, armazenamento em massa e geração de imagens, entre outros. Dentro de cada segmento, os processadores ARM podem ser encontrados em várias aplicações. Por exemplo, o processador ARM é encontrado em aplicações de rede, como gateways domésticos, modems DSL para comunicação em internet de alta velocidade e comunicação sem fio. O segmento de dispositivos móveis é a maior área de aplicação para processadores ARM por causa dos *smartphones*. Os processadores ARM também são encontrados em dispositivos de armazenamento em massa, como discos rígidos e produtos de plotagem, como impressoras no geral.

ARM é a arquitetura mais amplamente usada em termos de quantidade de dispositivos empregados. Os produtos baseados em ARM tornaram-se extremamente populares. Os processadores baseados em ARM equipam a maior parte dos dispositivos móveis do mundo. Muitas CPUs populares, de vários núcleos e com arquitetura de 64 bits, usadas em dispositivos que se tornaram ícones na indústria eletrônica (por exemplo, o *iPhone* da *Apple*), são baseadas em uma arquitetura ARM (ARMv8-A). No ano de 2020, a *Apple* anunciou o lançamento do primeiro *Macbook* cujo processador é ARM. As placas de desenvolvimento *Raspberry Pi* também usam processadores com núcleos ARM.

Uma arquitetura ARM é um conjunto de especificações relacionadas ao conjunto de instruções, modelo de execução, organização e layout da memória, ciclos de execução de instruções, entre outros detalhes que descrevem com precisão uma máquina que implementará a referida arquitetura. Microprocessadores com uma arquitetura *RISC,* em geral, necessitam de menos transistores do que microprocessadores *CISC*, como os da arquitetura x86, comumente encontrada em computadores pessoais do tipo desktop ou notebooks. Essa característica permite um menor consumo de energia, menor custo e menor dissipação de calor, o que faz dessa arquitetura algo desejado por fabricantes de dispositivos de tamanho reduzido, portáteis e alimentados por bateria, como *smartphones*, *tablets*, *laptops* entre outros sistemas embarcados.

Ao lidar com processadores baseados na arquitetura ARM, muita confusão pode surgir devido ao fato de haver muitas revisões diferentes dessa arquitetura (ARMv6, ARMv6-M, ARMv7-M, ARMv7-A e assim por diante). Por exemplo, um processador baseado no núcleo Cortex-M4 foi projetado na arquitetura ARMv7-M.

A *ARM Holdings* publica periodicamente atualizações para suas arquiteturas, todas elas com espaço de endereçamento e aritmética de 32/64 bits. Instruções para chips da ARM possuem tamanho fixo de 32 bits e nas versões mais recentes também suportam instruções de 16 bits para melhorar a densidade de código. Desde 2011, a arquitetura ARM também suporta endereçamento e aritmética de 64 bits, com instruções de tamanho fixo de 32 e 16 bits.

Por ser um padrão generalizado, existem muitos compiladores e ferramentas, além de sistemas operacionais (o Linux é o sistema operacional mais usado nos processadores ARM Cortex-A) que suportam essas arquiteturas, oferecendo aos desenvolvedores muitas opções para construir suas aplicações.

Um núcleo ARM usa uma arquitetura *RISC*, que traz uma filosofia de design que visa fornecer instruções simples, porém poderosas, que são executadas em um único ciclo de clock e em alta velocidade. A filosofia *RISC* concentra-se em reduzir a complexidade das instruções executadas pelo hardware, porque é mais fácil fornecer maior flexibilidade e inteligência no software, em vez do hardware. Como consequência, um design *RISC* exige mais do compilador.

A filosofia *RISC* na arquitetura ARM é implementada com quatro regras principais de design:

1. Instruções - Os processadores *RISC* têm um número reduzido de tipos de instrução. Esses tipos fornecem operações simples que podem ser executadas em um único ciclo de clock. O compilador ou programador sintetiza operações complexas (por exemplo, o cálculo do determinante de uma matriz) combinando várias instruções simples. Cada instrução tem um comprimento fixo para permitir que o *pipeline* busque instruções futuras antes de executar a instrução atual.

2. Pipelines - O processamento das instruções é dividido em unidades menores que podem ser executadas em paralelo. Idealmente, o *pipeline* avança uma etapa em cada ciclo para obter o máximo rendimento na execução das instruções.

3. Registradores - as máquinas *RISC* têm um grande conjunto de registradores de uso geral. Qualquer registrador pode conter dados ou um endereço de memória. Os registradores agem como o armazenamento rápido de memória local para todas as operações de processamento de dados.

4. Arquitetura Load/Store - O processador opera com dados mantidos em registradores. Instruções separadas de carregamento (*Load*) e armazenamento (*Store*) transferem dados entre o banco de registradores e a memória externa. Os acessos à memória são lentos, portanto, separar os acessos à memória do processamento de dados oferece uma vantagem, pois pode-se usar os itens de dados mantidos no banco de registradores várias vezes, sem precisar de vários acessos à memória, aumentando a velocidade na execução do código.

Essas regras de design permitem que um processador *RISC* seja mais simples e, portanto, o núcleo pode operar com frequências de clock mais altas. Por outro lado, os processadores *CISC* tradicionais são mais complexos e operam em frequências de clock mais baixas. Ao longo de duas décadas, no entanto, a distinção entre *RISC* e *CISC* diminuiu à medida que os processadores *CISC* implementaram mais conceitos *RISC*.

Há vários recursos físicos que impulsionaram o design dos processadores ARM. Primeiro, os sistemas embarcados portáteis são normalmente alimentados por baterias. O processador ARM foi projetado especificamente para ser pequeno, para reduzir o consumo de energia e estender a operação da bateria, o que é essencial para aplicações como *smartphones*, por exemplo.

A alta densidade de código é outro requisito importante, uma vez que os sistemas embarcados têm memória limitada devido às restrições de custo e/ou tamanho físico. A alta densidade de código é útil para aplicações com memória interna limitada, como *smartphones* e dispositivos de armazenamento em massa. Além disso, os sistemas embarcados são sensíveis ao preço e usam dispositivos de memória lentos e de baixo custo. Para aplicações de alto volume, como câmeras digitais, cada centavo deve ser contabilizado no design. A capacidade de usar dispositivos de memória de baixo custo produz economias substanciais.

Outro requisito importante é reduzir a área do chip ocupada pelo processador. Para uma solução de chip único, quanto menor a área usada pelo processador, maior o espaço disponível para periféricos especializados. Isso, por sua vez, reduz o custo do projeto e da fabricação, já que são necessários menos chips discretos para compor o produto final.

A ARM incorporou a tecnologia de depuração no próprio hardware do processador para que os engenheiros de software possam ver o que está acontecendo enquanto o processador está executando o código. Com maior visibilidade, os engenheiros de software podem resolver os problemas mais rapidamente, o que afeta diretamente o tempo de lançamento de produtos no mercado e reduz os custos gerais de desenvolvimento.

O núcleo ARM não é uma arquitetura *RISC* pura devido às restrições de sua principal aplicação: os sistemas embarcados. Em certo sentido, a força do núcleo ARM é que ele não leva o conceito *RISC* longe demais. Nos sistemas embarcados de hoje, a chave não é a velocidade do processador em si, mas o desempenho efetivo total do sistema e o consumo de energia.

O conjunto de instruções ARM difere da definição pura de uma máquina *RISC* de várias maneiras que o tornam adequado para sistemas embarcados e tornam o processador ARM um dos processadores com núcleo de 32/64 bits mais usados no mundo:

■ Execução de ciclo variável para determinadas instruções - nem toda instrução ARM é executada em um único ciclo de clock. Por exemplo, instruções múltiplas de *Load-Store* variam no número de ciclos de execução, dependendo da quantidade de registradores sendo transferidos. A transferência pode ocorrer em endereços sequenciais de memória, o que aumenta o desempenho, pois os acessos sequenciais à memória geralmente são mais rápidos que os acessos aleatórios. A densidade do código também é aprimorada, pois várias transferências de registradores são operações comuns no início e no final das funções.

■ Deslocador de bits em linha (*Barrel Shifter*), levando a instruções mais complexas – O deslocador de bits em linha é um importante componente de hardware que processa previamente um dos registradores de entrada antes de ser utilizado por uma instrução. Isso expande a capacidade de muitas instruções para melhorar o desempenho do núcleo e a densidade do código.

■ Conjunto de instruções *Thumb* - A ARM aprimorou o núcleo do processador adicionando um segundo conjunto de instruções de 16 bits chamado *Thumb*, que permite que o núcleo ARM execute instruções de 16 ou 32 bits de comprimento. As instruções de 16 bits melhoram a densidade do código em cerca de 30% em relação às instruções de comprimento fixo de 32 bits.

■ Execução condicional - uma instrução é executada apenas quando uma condição específica é atendida. Esse recurso melhora o desempenho e a densidade do código, reduzindo as instruções de desvio.

■ Instruções aprimoradas - As instruções de processamento digital de sinais (*DSP – Digital Signal Processing*) foram adicionadas ao conjunto de instruções ARM padrão para suportar operações de saturação e multiplicações rápidas de 16 × 16 bits. Essas instruções permitem que um processador ARM de desempenho mais rápido, em alguns casos, substitua as combinações tradicionais de um processador mais um núcleo *DSP*.

Podemos ver um núcleo ARM como unidades funcionais conectadas por barramentos de dados como mostrado na Figura 1, onde as setas representam o fluxo de dados, as linhas representam os barramentos e as caixas representam uma unidade operacional ou uma área de armazenamento. A figura mostra não apenas o fluxo de dados, mas também os componentes que compõem um núcleo ARM.

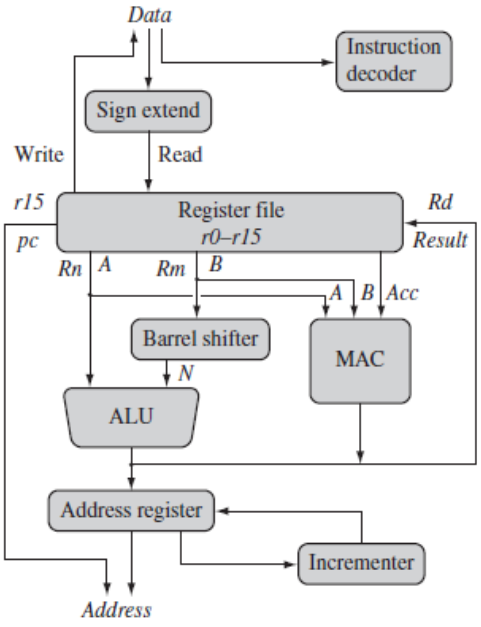

Figura 1 – Modelo de fluxo de dados do núcleo ARM.

Os dados entram no núcleo do processador através do barramento de dados (*Data*). O conteúdo do barramento de dados pode ser uma instrução ou efetivamente um item de dado. A Figura 1, por simplicidade, mostra uma implementação de von Neumann do núcleo ARM: Itens de dados e instruções compartilham o mesmo barramento. Por outro lado, as implementações de Harvard do núcleo ARM usam dois barramentos diferentes. O decodificador de instruções (*Instruction decoder*) traduz instruções antes de serem executadas. Cada instrução executada pertence a um conjunto de instruções específico.

O processador ARM, como todos os processadores *RISC*, usa uma arquitetura *Load-Store*. Isso significa que ele tem dois tipos de instruções para transferir dados para dentro e para fora do processador: Instruções de carregamento (*Load*) copiam dados da memória para registradores no núcleo e, inversamente, as instruções de armazenamento (*Store*) copiam dados dos registradores para a memória. Não há instruções de processamento de dados que manipulem diretamente os dados na memória. Assim, o processamento de dados é realizado apenas em dados armazenados nos registradores.

Os itens de dados são colocados no banco de registradores (*Register file*) composto por 16 registradores de 32 bits nomeados de *r0* a *r15*. Como o núcleo ARM é um processador de 32 bits, a maioria das instruções trata os registradores como se estivessem armazenando valores de 32 bits, com ou sem sinal. O hardware de extensão de sinal (*Sign extend*) converte números com sinal de 8 e 16 bits em números com sinal de 32 bits à medida que são lidos da memória e colocados em um registrador.

As instruções ARM, normalmente, usam dados de dois registradores de origem, *Rn* e *Rm*, e um único registrador de resultado ou destino, *Rd*. Os operandos de origem são lidos do banco de registradores usando os barramentos internos *A* e *B*, respectivamente. A unidade lógica e aritmética (*ALU -Arithmetic Logic Unit*) ou a unidade de multiplicação e acumulação (*MAC – Multiply and Accumulate*) acessa os valores dos registradores *Rn*, *Rm* e acumulador pelos barramentos *A*, *B* e *Acc* e calcula o resultado da operação. As instruções de processamento de dados gravam o resultado em *Rd,* diretamente no banco de registradores. As instruções de carregamento e armazenamento usam a ULA para gerar um endereço a ser mantido no registrador de endereços (*Address register*) e o envia ao barramento de endereços (*Address*).

Uma característica importante do núcleo ARM é que o registrador *Rm*, alternativamente, pode ser pré-processado no deslocador de bits em linha (*Barrel shifter*) antes de entrar na ULA pela entrada *N*. Juntos, o deslocador e a ULA podem calcular uma ampla variedade de expressões e endereços.

Depois de passar pelas unidades funcionais, o resultado é gravado de volta no banco de registradores, em *Rd*, usando o barramento de resultados (*Result*). Para instruções de carregamento e armazenamento, o incrementador (*Incrementer*) atualiza o registrador de endereços antes que o núcleo leia ou grave o próximo valor do registrador a partir do ou para o próximo local de memória sequencial. O processador continua executando as instruções até que uma exceção ou interrupção altere o fluxo normal de execução do programa.

3.2. Família ARM Cortex

Em 2004, a ARM introduziu no mercado uma nova família de processadores, conhecida como ARM Cortex. ARM Cortex é um amplo conjunto de arquiteturas e núcleos de 32/64 bits, bastante popular no mundo dos sistemas embarcados. Os processadores ARM Cortex são divididos em três subfamílias principais, chamadas de perfis:

• **Cortex-A**, que significa *Application*, é uma série de processadores que fornecem uma variedade de soluções para dispositivos que realizam tarefas computacionais complexas, como hospedar um sistema operacional (SO) (o Linux e seu derivado Android são os mais comuns) e com suporte a vários aplicativos de software. Os núcleos Cortex-A equipam os processadores encontrados na maioria dos dispositivos móveis, como *smartphones* e *tablets*. Nesse segmento de mercado, podemos encontrar vários fabricantes de chips, desde os que vendem peças de catálogo até os que produzem processadores para outros fabricantes de produtos embarcados. Entre os núcleos mais comuns desse segmento, podemos encontrar os processadores Cortex-A7 e Cortex-A9 de 32 bits, bem como os núcleos de alto desempenho Cortex-A53 e Cortex-A57 de 64 bits. Exemplos de produtos que usam esse perfil incluem *smartphones*, *tablets*, aparelhos de TV, receptores de TV por satélite e servidores.

• **Cortex-M**, que significa *eMbedded*, é uma gama de processadores escaláveis, compatíveis entre si, eficientes em termos de energia e fáceis de usar, projetados para o mercado de sistemas embarcados. A família Cortex-M é otimizada para utilização como CPUs em microcontroladores sensíveis a custos e energia, adequados para aplicações como internet das coisas, conectividade, controle de motores, medição inteligente, dispositivos de interface humana, sistemas de controle industrial e automotivo, eletrodomésticos, produtos domésticos, produtos médicos e instrumentação no geral. Nesse segmento de mercado, podemos encontrar muitos fabricantes de chips que produzem processadores Cortex-M. A *ST Microelectronics* é um dos fabricantes que possui um portfólio completo de produtos desse perfil.

• **Cortex-R**, que significa *Real-Time*, é uma série de processadores que oferecem soluções de computação de alto desempenho para sistemas embarcados em que confiabilidade, alta disponibilidade, tolerância a falhas, manutenibilidade e resposta determinística em tempo real são essenciais. Os processadores da série Cortex-R oferecem processamento rápido, determinístico e alto desempenho, atendendo a restrições desafiadoras em tempo real. Eles combinam esses recursos em um pacote otimizado de desempenho, energia e área, tornando-os a escolha certa em sistemas confiáveis que exigem tolerância a falhas. Exemplos de aplicação são controladores de disco rígido, controladores de banda base para comunicações móveis e sistemas automotivos, nos quais são essenciais alta capacidade de processamento e alta confiabilidade, e para os quais é importante baixa latência e determinismo.

3.3. Série ARM Cortex-M

A série Cortex-M é uma família de núcleos de processamento *RISC* de 32 bits projetados especificamente para atingir o mercado de *MCU* (microcontroladores), que já é bastante movimentado. A série Cortex-M é construída sobre a arquitetura ARMv7-M (usada para Cortex-M3, Cortex-M4 e Cortex-M7), os menores Cortex-M0 e Cortex-M0+ são construídos sobre a arquitetura ARMv6-M, e o Cortex-M1 voltado para aplicações com FPGA.

O primeiro processador Cortex-M foi lançado em 2005 (Cortex-M3) e rapidamente ganhou popularidade quando alguns dos principais fornecedores de *MCUs* adotaram o núcleo e começaram a produzir seus dispositivos. O Cortex-M4, por sua vez, foi lançado em 2010. É seguro dizer que o Cortex-M se tornou para o mundo de 32 bits o que o processador 8051 é para o de 8 bits - um padrão da indústria adotado por muitos fornecedores, cada um dos quais mergulha o núcleo em seus próprios produtos para fornecer diferenciação no mercado.

A série Cortex-M pode ser implementada como um núcleo flexível em um FPGA (como o Cortex-M1), mas é muito mais comum encontrá-los implementados em *MCU* com memórias, circuitos de clock e periféricos integrados. Alguns são otimizados para eficiência energética, outros para alto desempenho e outros são adaptados para um segmento de mercado específico.

O Cortex-M3 e o Cortex-M4 são núcleos muito semelhantes. Cada um oferece execução de instruções com um *pipeline* de 3 estágios (no Cortex-M7 são 6 estágios), vários barramentos de 32 bits, velocidades de clock de até 216 MHz e opções de depuração muito eficientes. A diferença significativa é a capacidade do núcleo Cortex-M4 para *DSP*. O Cortex-M3 e o Cortex-M4 compartilham a mesma arquitetura e conjunto de instruções (*Thumb-2*). No entanto, o Cortex-M4 adiciona uma série de instruções de saturação e *SIMD* (*Single Instruction Multiple Data*), especificamente otimizadas para lidar com algoritmos *DSP*. Por exemplo, considere o caso de uma *FFT* (*Fast Fourier Transform*) de 512 pontos em execução a cada 0,5 segundo em *MCUs* Cortex-M3 e Cortex-M4 disponíveis no mercado. Para comparação, o Cortex-M3 consumiria cerca de três vezes a energia que um Cortex-M4 precisaria para o mesmo trabalho. Também há a opção de obter uma unidade de ponto flutuante (*FPU – Floating Point Unit*) de precisão simples em um Cortex-M4. Se a aplicação exigir matemática de ponto flutuante, os cálculos ocorrerão consideravelmente mais rápidos em um Cortex-M4 do que em um Cortex-M3. Dito isto, para uma aplicação que não está usando os recursos *DSP* ou *FPU* do Cortex-M4, verifica-se o mesmo nível de desempenho e consumo de energia em um Cortex-M3. O Cortex-M7 oferece FPU com precisão dupla e mais instruções *DSP*.

Para aplicações particularmente sensíveis ao custo ou que estão migrando de 8 para 32 bits, o menor membro da série Cortex-M pode ser a melhor opção. O desempenho do Cortex-M0 fica um pouco abaixo do desempenho do Cortex-M3 e Cortex-M4, mas ainda é compatível com seus irmãos maiores. O Cortex-M0+ usa um subconjunto do conjunto de instruções *Thumb-2* e essas instruções usam operandos predominantemente de 16 bits (embora todas as operações de dados sejam de 32 bits), que se prestam muito bem ao *pipeline* de dois estágios que o Cortex-M0+ oferece. Isso gera uma economia geral de energia para o sistema. O Cortex-M0+ também possui um barramento dedicado para GPIO de ciclo único, o que significa que é possível implementar determinadas interfaces com GPIO baseado em bits, como faria em um *MCU* de 8 bits, mas com o desempenho de um núcleo de 32 bits para processar os dados.

O nível de informações de depuração que se pode extrair de um processador Cortex-M é significativamente maior do que o obtido em um *MCU* de 8 bits, o que significa que os erros difíceis de resolver são bem mais fáceis de corrigir.

Os processadores ARM Cortex-M têm muitas vantagens técnicas e não técnicas quando comparados com outras arquiteturas. Comparado a outros designs de processadores de 32 bits, os processadores Cortex-M são relativamente pequenos. Os processadores Cortex-M também são otimizados para baixo consumo de energia. Atualmente, muitos microcontroladores Cortex-M têm consumo de energia inferior a 200 μA/MHz, com alguns deles abaixo de 100 μA/MHz. Além disso, os processadores Cortex-M também incluem suporte para recursos do modo de suspensão de operação e podem ser usados com várias tecnologias avançadas de design de ultrabaixa potência. Tudo isso permite que os processadores Cortex-M sejam usados em vários produtos de potência ultrabaixa. Como alternativa, é possível executar programas com uma velocidade de clock muito mais lenta para reduzir ainda mais o consumo de energia.

O set de instruções dos processadores Cortex-M, chamado de *Thumb*, fornece excelente densidade de código. Isso significa que, para realizar as mesmas tarefas, é necessário um tamanho de programa menor. Como resultado, o custo e o consumo de energia podem ser reduzidos usando um microcontrolador com tamanho menor de memória e os fabricantes podem produzir chips com encapsulamentos menores.

Os processadores Cortex-M3 e Cortex-M4 têm um controlador de interrupções configurável que pode suportar até 240 interrupções e vários níveis de prioridade (de 8 a 256 níveis). O aninhamento de interrupções é tratado automaticamente pelo hardware e a latência de interrupção pode chegar a apenas 12 ciclos de clock. A capacidade de processamento de interrupção também torna os processadores Cortex-M adequados para muitas aplicações de controle em tempo real.

3.4. ARM Cortex-M3/M4

O processador Cortex-M3 foi o primeiro da geração de processadores Cortex-M, lançado pela ARM em 2005. O processador Cortex-M4 foi lançado em 2010. Os processadores Cortex-M3/M4 usam uma arquitetura de 32 bits. Os registradores internos do banco de registradores, o *datapath* e as interfaces de barramento têm 32 bits de largura. A arquitetura do conjunto de instruções nos processadores Cortex-M é chamada de *Thumb ISA* e é baseada na tecnologia *Thumb-2*, que suporta tanto instruções de 16 bits como de 32 bits de largura.

Os núcleos Cortex-M3 e Cortex-M4 têm:

• *Pipeline* de três estágios;
• Arquitetura Harvard com espaço de memória unificado: instruções e dados usam o mesmo espaço de endereço;
• Endereçamento de 32 bits, suportando 4 GB de espaço de memória;
• Interfaces de barramento no chip baseadas na tecnologia ARM AMBA (*Advanced Microcontroller Bus Architecture*), que permite operações de barramentos em paralelo e alta velocidade para maior produtividade;
• Um controlador de interrupções chamado *NVIC* (*Nested Vectored Interrupt Controller*), dando suporte a até 240 solicitações de interrupção e de 8 a 256 níveis de prioridade (dependendo da implementação real do dispositivo);
• Suporte a vários recursos para implementação de sistema operacional, como cronômetro de sistema, ponteiro de pilha dupla, unidade de gerenciamento de memória, etc;
• Suporte ao modo de suspensão e vários recursos de baixo consumo de energia;
• Suporte para uma unidade de proteção de memória (*MPU – Memory Protection Unit*) opcional para fornecer recursos de proteção de memória, como memória programável ou controle de permissão de acesso;
• Suporte para acesso a bits de dados em duas regiões específicas da memória usando o recurso chamado *Bit Band;*
• A opção de ser usado em modelos de processador único ou multiprocessador.

A *ISA* usada nos processadores Cortex-M3/M4 fornece uma ampla variedade de instruções:

• Processamento geral de dados, incluindo instruções de divisão por hardware;
• Instruções de acesso à memória que suportam dados de 8, 16, 32 e 64 bits, bem como instruções para transferir vários dados de 32 bits;
• Instruções para processamento de campos de bits;
• Instruções de multiplicação e acumulação (MAC) e saturação;
• Instruções para desvios, desvios condicionais e chamadas de função;
• Instruções para controle do sistema, suporte a SO, etc.
• Instruções SIMD (*Single Instruction Multiple Data*);
• MAC rápido e instruções de multiplicação;
• Instruções opcionais de ponto flutuante;
• Instruções para *DSP* (no Cortex-M4);

Em geral, os processadores ARM Cortex-M3/M4 são considerados processadores *RISC*. Alguns podem argumentar que certas características dos processadores Cortex-M3 e Cortex-M4, como o rico conjunto de instruções e tamanhos variados de instruções, estão mais próximos dos processadores *CISC*. Mas conforme as tecnologias avançam, o conjunto de instruções dos processadores *RISC* também estão ficando mais complexos, tanto que a fronteira tradicional entre a definição de processador *RISC* e *CISC* está a cada dia sendo menos aplicada.

Existem muitas semelhanças entre os processadores Cortex-M3 e Cortex-M4. A maioria das instruções está disponível nos dois processadores e ambos têm o mesmo modelo de programação para o controlador *NVIC*, *MPU*, etc. No entanto, existem algumas diferenças em seus designs internos, que permitem que o processador Cortex-M4 ofereça maior desempenho em aplicativos *DSP*, bem como no apoio a operações de ponto flutuante.

3.5. Arquitetura dos processadores Cortex-M3/M4

Os processadores Cortex-M3 e Cortex-M4 são baseados na arquitetura ARMv7-M. A arquitetura original do ARMv7-M foi definida quando o processador Cortex-M3 foi desenvolvido e, quando o Cortex-M4 foi lançado, a arquitetura foi estendida para incluir instruções e recursos adicionais. A arquitetura estendida às vezes é chamada de arquitetura ARMv7E-M. Os recursos do ARMv7-M e ARMv7E-M estão documentados no mesmo documento de especificação da arquitetura: o Manual de Referência da Arquitetura do ARMv7-M.

O Manual de Referência da Arquitetura ARMv7-M é um documento massivo com cerca de 1000 páginas. Ele fornece requisitos arquitetônicos muito detalhados do comportamento do processador, do conjunto de instruções, sistema de suporte à memória e à depuração. Embora seja útil para especialistas, como projetistas de processadores, projetistas de compiladores C e ferramentas de desenvolvimento, este documento não é fácil de ler, especialmente para leitores iniciantes na arquitetura ARM.

Para usar um processador Cortex-M em aplicações típicas com microcontroladores, não é necessário ter conhecimento detalhado dessa arquitetura. Você só precisa ter um entendimento básico do modelo de programação, como as exceções (e interrupções) são tratadas, o mapa de memória, como usar os periféricos e como usar os arquivos da biblioteca de drivers de software dos fornecedores de microcontroladores.

Estados e modos de operação

Os processadores Cortex-M3 e Cortex-M4 possuem dois estados de operação, *Thumb* e *Debug*. No estado *Thumb,* possui dois modos: *Thread* e *Handler*. Além disso, o modo *Thread* pode ter níveis de acesso privilegiados e não privilegiados. Eles são mostrados na Figura 2. O nível de acesso privilegiado pode acessar todos os recursos do processador, enquanto o nível de acesso sem privilégios significa que algumas regiões de memória estão inacessíveis e algumas operações não podem ser usadas. Em alguns documentos, o nível de acesso não privilegiado também pode ser chamado de "nível de usuário".

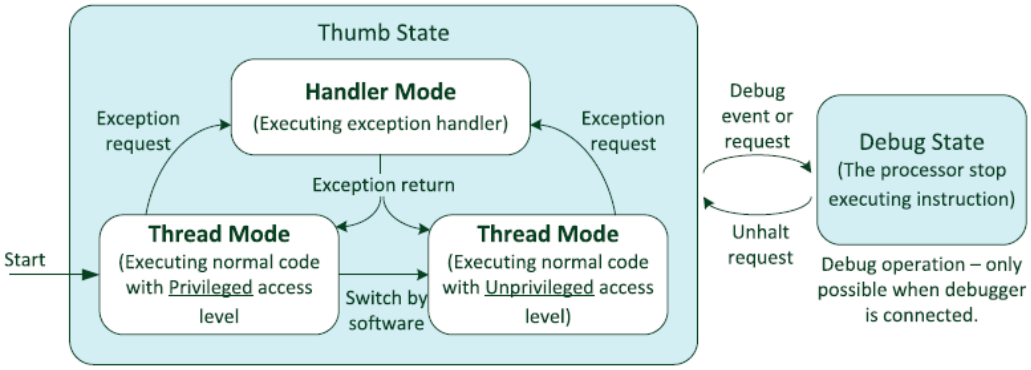

Figura 2 – Estados e modos de operação dos processadores ARM Cortex-M3/M4.

Se o processador estiver executando o código do programa (instruções *Thumb*), ele estará no estado *Thumb*. Quando o processador é parado (por exemplo, pelo *debugger* ou após atingir um *breakpoint* de software), ele entra no estado de depuração (*Debug State*) e para de executar as instruções.

Ao executar uma rotina de atendimento à interrupção, dizemos que o processador está no modo *Handler*. Nesse modo, o processador sempre tem nível de acesso privilegiado. Ao executar o código normal da aplicação, dizemos que o processador está no modo *Thread*. Nesse modo, o processador pode estar no nível de acesso privilegiado ou no nível de acesso não privilegiado. Isso é controlado por um registrador especial chamado "CONTROL".

O software pode alternar o processador do modo *Thread* privilegiado para o modo não privilegiado. No entanto, ele não pode voltar de não privilegiado para privilegiado. Se isso for necessário, o processador precisará usar o mecanismo de exceção para permitir a comutação dos modos.

A separação dos níveis de acesso privilegiado e não privilegiado permite que os projetistas desenvolvam sistemas embarcados robustos, fornecendo um mecanismo para proteger o acesso à memória em regiões críticas e implementando um modelo básico de segurança. Por exemplo, um sistema pode conter um *kernel* de um sistema operacional (SO) que é executado no nível de acesso privilegiado e tarefas de aplicativos que são executadas no nível de acesso não privilegiado. Dessa forma, podemos configurar permissões de acesso à memória usando a unidade de proteção de memória (*MPU - Memory Protection Unit*) para impedir que uma tarefa de aplicativo corrompa a memória e os periféricos usados pelo *kernel* do SO e outras tarefas. Se uma tarefa do aplicativo falhar, as tarefas restantes do aplicativo e o *kernel* do SO ainda poderão continuar em execução.

Por padrão, os processadores Cortex-M iniciam no estado *Thumb* no modo *Thread* privilegiado. Em muitas aplicações simples, não há necessidade de usar o modo *Thread* sem privilégios.

O estado *Debug* é usado apenas para operações de depuração. Esse estado é alcançado por uma solicitação de parada do depurador ou por eventos de depuração gerados a partir de componentes de depuração no processador. Esse estado permite que o depurador acesse ou altere os valores dos registradores do processador.

Banco de registradores

Os processadores Cortex-M3 e Cortex-M4 possuem vários registradores para o controle e execução do processamento dos dados. A maioria desses registradores está agrupada em uma unidade chamada banco de registradores. Cada instrução de processamento de dados especifica a operação necessária, o(s) registrador(s) de origem e o(s) registrador(s) de destino, se aplicável. A arquitetura ARM é do tipo *Load-Store*, isto é, se os dados na memória devem ser processados, eles precisam ser carregados da memória para os registradores do banco, processados dentro do processador e, em seguida, gravados na memória, se necessário. Por ter um número suficiente de registradores no banco, esse arranjo é fácil de usar e permite que um código de programa eficiente seja gerado usando compiladores C. Por exemplo, várias variáveis de dados podem ser armazenadas no banco de registradores por um curto período de tempo, enquanto outro processamento de dados ocorre sem a necessidade de ser atualizado para a memória do sistema e lido novamente toda vez que são usados.

O banco de registradores nos processadores Cortex-M3 e Cortex-M4 possui 16 registradores de 32 bits. Treze deles são registradores de uso geral e os outros três têm usos especiais, como pode ser visto na Figura 3.

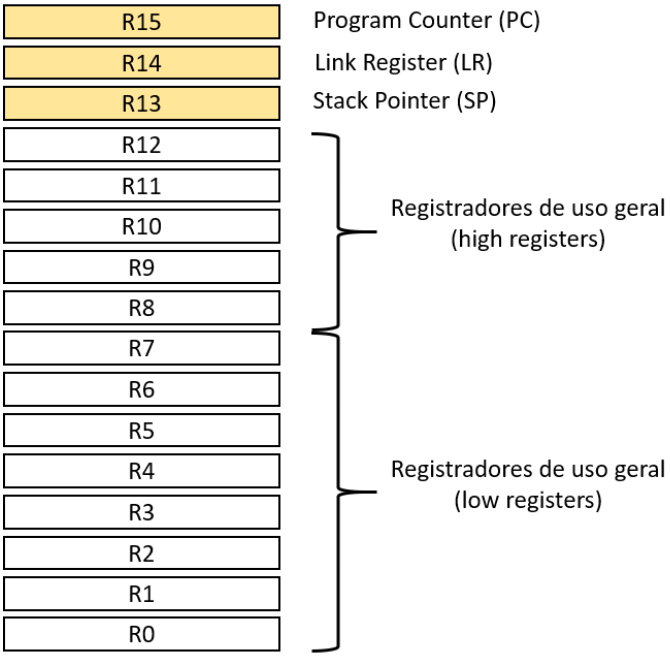

Figura 3 – Banco de registradores do ARM Cortex-M3/M4.

Os registradores nomeados de R0 a R12 são de uso geral. Os oito primeiros (R0 a R7) também são chamados de registradores baixos (*low registers*). Devido ao espaço disponível limitado no conjunto de instruções, muitas instruções de 16 bits de comprimento podem acessar apenas os registradores baixos. Os registradores altos (R8 a R12) podem ser usados com instruções de 32 bits e alguns com instruções de 16 bits. Os valores iniciais de R0 a R12 são indefinidos.

R13 é o ponteiro da pilha (*Stack Pointer - SP*). É usado para acessar a pilha da memória através de operações PUSH e POP. Fisicamente, existem dois diferentes ponteiros de pilha: O ponteiro de pilha principal (*MSP – Master Stack Pointer*) é o ponteiro de pilha padrão. É selecionado após um reset ou quando o processador está no modo *Handler*; O outro ponteiro de pilha é chamado de ponteiro de pilha de processo (*PSP – Process Stack Pointer*). O *PSP* pode ser usado apenas no modo *Thread*.

A seleção do ponteiro da pilha é determinada pelo registrador especial CONTROL. Em programas simples, apenas um desses ponteiros de pilha estará visível. O *MSP* e o *PSP* são de 32 bits, mas os dois bits mais baixos são sempre zero e as gravações nesses dois bits são ignoradas. Nos processadores ARM Cortex-M, as operações PUSH e POP são sempre de 32 bits e os endereços das transferências nas operações de pilha devem estar alinhados aos limites de palavras de 32 bits.

Na maioria dos casos, não é necessário usar o *PSP* se a aplicação não exigir um sistema operacional. O *PSP* é normalmente usado quando um SO está envolvido, onde a pilha para o *kernel* do sistema operacional e as tarefas de aplicativos são separadas. O valor inicial do *PSP* é indefinido e o valor inicial do *MSP* é obtido da primeira palavra da memória durante a sequência de inicialização do processador, como será discutido mais adiante.

R14 também é chamado de *Link Register* (LR). É usado para armazenar o endereço de retorno ao chamar uma função ou sub-rotina. No final da função ou sub-rotina, o controle do programa pode retornar ao programa de chamada carregando o valor de LR no Contador de Programa (PC). Quando uma chamada de função ou sub-rotina é feita, o valor de LR é atualizado automaticamente. Se uma função precisar chamar outra função ou sub-rotina, será necessário salvar o valor de LR na pilha primeiro. Caso contrário, o valor atual em LR será perdido quando a segunda chamada de função for feita. Durante o tratamento de exceções, o LR também é atualizado automaticamente para um valor especial chamado de EXC_RETURN (retorno de exceção), que é usado para acionar o retorno de exceção no final da rotina de atendimento da exceção.

R15 é o contador de programa (PC). Pode ser lido e gravado: uma leitura retorna o endereço da próxima instrução. Gravar no PC (por exemplo, usando instruções de transferência/processamento de dados) causa uma operação de desvio. Na maioria dos casos, desvios e chamadas de função são tratadas pelas instruções dedicadas a essas operações. Não é comum usar instruções de processamento de dados para atualizar o PC. No entanto, o valor do PC é útil para acessar dados literais armazenados na memória de programa. Assim, você pode encontrar frequentemente operações de leitura de memória com o PC como registrador de endereço base adicionado a offsets gerados por valores imediatos nas instruções.

Registrador de status do processador CPSR

Além dos registradores do banco, há vários registradores especiais. Esses registradores contêm o status do processador e definem os estados da operação e o mascaramento (habilitação/inibição) de interrupções/exceções.

No desenvolvimento de aplicativos simples com linguagens de programação de alto nível, como C, não existem muitos cenários que exijam acesso a esses registradores. No entanto, eles são necessários para o desenvolvimento de um sistema operacional ou quando são necessários recursos avançados de interrupção. Registradores especiais não são mapeados na memória e podem ser acessados apenas usando instruções especiais.

O núcleo ARM usa o *CPSR* (*Current Program Status Register*) para monitorar e controlar operações internas. O *CPSR* é um registrador dedicado de 32 bits. A Figura 4 mostra o layout desse registrador.

31	*30*	*29*	*28*	*27*	*26:25*	*24*	*23:20*	*19:16*	*15:10*	*9*	*8*	*7*	*6*	*5*	*4:0*
N	Z	C	V	Q	ICI/IT	T		GE*	ICI/IT		Exception Number				

Figura 4 – Layout do registrador CPSR.

O *CPSR* possui bits sinalizadores (*flags*), bits de status, bits de extensão e bits de controle. Os bits N, Z, C, V, Q são bits de status. O bit Q sinaliza uma operação de saturação. O bit T sinaliza o estado *Thumb* e é sempre lido como 1. GE é um campo de bits sinalizadores que são atualizados por instruções que operam sobre múltiplos dados, no qual cada bit representa uma comparação do tipo "maior que ou igual" (*Greater than or Equal*) para cada byte dos operandos. O campo ICI/IT habilita globalmente as interrupções e o campo *Exception Number* indica o número da interrupção que está sendo atendida, como será visto no tópico sobre exceções.

Mapa de memória

O barramento de endereços dos processadores ARM Cortex-M3/M4 é de 32 bits, o que suporta um espaço de endereçamento de memória de 2^{32} = 4 GB. Os endereços vão de 0x00000000 a até 0xFFFFFFFF.

Um importante aspecto dos processadores Cortex-M é que a memória de programa, a memória de dados, os periféricos externos e internos ocupam o mesmo espaço de endereçamento de memória, embora esses dispositivos sejam fisicamente distintos e os barramentos de endereçamento sejam diferentes.

Os tipos básicos de dados armazenados têm comprimento de 8 bits (bytes), 16 bits (*halfwords*) e 32 bits (*words*). Como o menor tipo de dado é o byte, dizemos que a memória é byte endereçável. Isso quer dizer que cada endereço de memória corresponde ao endereço de um byte. Entretanto, como o barramento de dados também é de 32 bits, a cada endereçamento na memória temos acesso simultâneo a 4 bytes consecutivos. Por exemplo, ao acessar o endereço 0x00001B00, também acessamos diretamente o conteúdo dos endereços 0x00001B01, 0x00001B02 e 0x00001B03. O endereçamento é sempre realizado com alinhamento de 32 bits.

O espaço de endereçamento de memória possui várias regiões com características e finalidades distintas. A Figura 5 mostra essas diferentes regiões, onde cada uma ocupa 512MB do espaço de endereçamento. Na Figura 5, também é mostrado, do lado direito, os endereços iniciais e finais das regiões que apresentam características semelhantes.

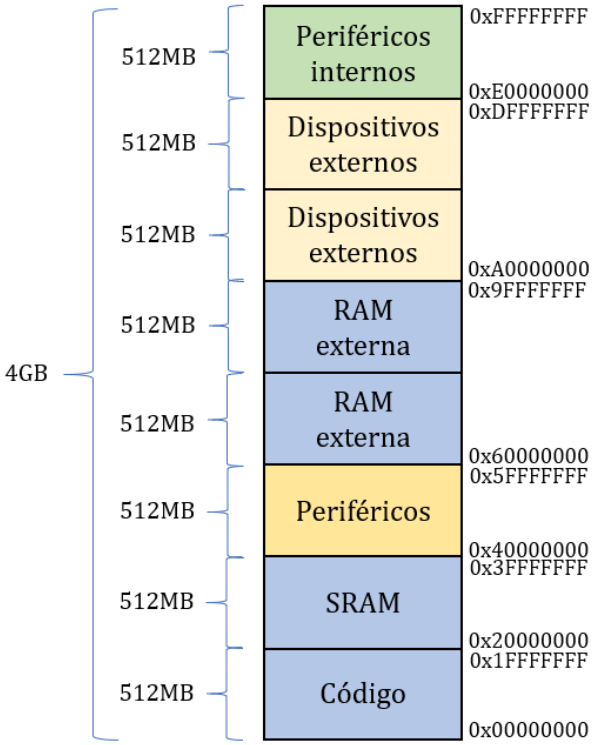

Figura 5 – Mapa de memória dos processadores ARM Cortex-M3/M4.

Os primeiros 512MB, iniciando do endereço 0x00000000 ao endereço 0x1FFFFFFF é o espaço de endereçamento reservado para o armazenamento do código do programa. As duas primeiras palavras de memória armazenam, respectivamente, o endereço inicial da pilha, e o vetor de reset (endereço inicial da função de atendimento ao reset). As palavras seguintes armazenam a tabela de vetores de interrupção. Esse espaço também é utilizado para o armazenamento de constantes definidas e utilizadas nos programas.

A região seguinte, é reservada à alocação da memória RAM estática, utilizada primariamente para o armazenamento de dados, embora possa também conter código executável. Nesse espaço de memória é também alocada a área da pilha.

Iniciando no endereço 0x40000000, temos a região destinada aos periféricos internos, que é seguida por duas regiões que formam um espaço de 1GB reservado à memória RAM externa. Muitos processadores não suportam acesso à memória externa porque o barramento de acesso exige uma grande quantidade de pinos disponíveis no chip, o que requer que o encapsulamento seja muito grande, caro, além de exigir uma quantidade maior de energia para a sua operação.

O espaço de 1GB seguinte é destinado ao endereçamento de dispositivos (periféricos) externos e a última região, iniciando no endereço 0xE0000000 é destinada aos periféricos que controlam as operações internas do próprio processador, como o controlador de interrupções (*NVIC*), o timer do sistema (*System Tick Timer SysTick*), registradores de controle e de suporte à depuração, além de um espaço para que os fabricantes de microcontroladores insiram seus periféricos de controle específicos pra cada produto.

Na Figura 5, é mostrado que os primeiros 1,5GB são reservados à memória interna do chip, isto é, à memória de programa (normalmente uma memória *flash* nos microcontroladores), à memória de dados (uma RAM estática) e aos periféricos (como portas de I/O, conversores A/D e D/A, contadores, portas de comunicação serial, etc.). Cada um desses segmentos de memória ocupa 0,5GB e podem ser usados barramentos fisicamente separados para cada espaço de endereçamento.

Na prática, muitos microcontroladores usam apenas uma porção muito pequena de cada região. Algumas regiões podem, inclusive, nem serem utilizadas porque os microcontroladores de fabricantes diferentes têm tamanhos de memória de programa e RAM diferentes, quantidade e localização de periféricos que variam de um fabricante para outro. Essas informações são geralmente descritas nos manuais do usuário ou nas fichas técnicas dos fornecedores de microcontroladores.

Exceções e interrupções

Exceções são eventos que causam alterações no fluxo do programa. Quando isso acontece, o processador suspende a tarefa em execução atual e executa um trecho de código chamado de manipulador de exceções (como as rotinas de atendimento à interrupção). Após a execução da rotina de tratamento de exceções, o processador retoma a execução normal do programa.

Na arquitetura ARM, as interrupções são um tipo de exceção. As interrupções geralmente são geradas a partir de entradas periféricas ou externas e, em alguns casos, podem ser acionadas por software. Os manipuladores de exceção para interrupções também são chamados de *ISR* (*Interrupt Service Routines*).

Nos processadores Cortex -M, existem várias fontes de exceção que são processadas pelo controlador de interrupções vetoradas e aninhadas (*NVIC - Nested Vectored Interrupt Controller)*, conforme mostrado na Figura 6.

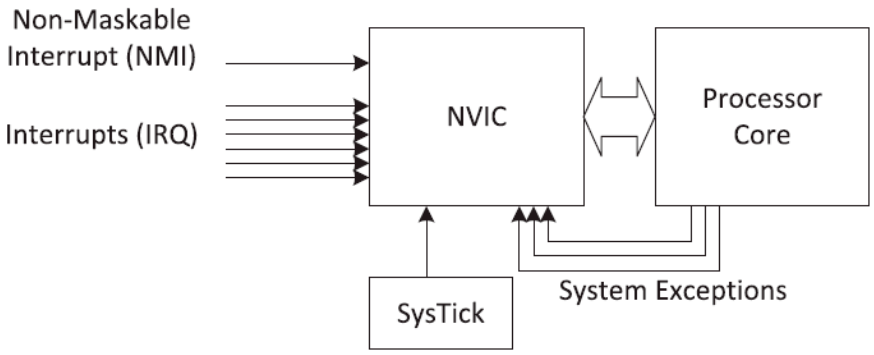

Figura 6 – Várias fontes de interrupção controladas pelo NVIC.

O *NVIC* pode lidar com várias solicitações de interrupção (*IRQs*) e uma interrupção não mascarável (*Non-Maskable Interrupt - NMI*). Normalmente, as *IRQs* são geradas por periféricos no chip ou a partir de entradas de interrupção externas através de portas de I/O. No processador, há também um timer chamado *SysTick*, que pode gerar uma solicitação de interrupção periódica, que pode ser usada pelos SOs para cronometragem ou para controle simples de tempo em aplicações que não exigem um SO. O próprio processador também é uma fonte de eventos de exceção (*System Exceptions*). Estes podem ser eventos de falha que indicam condições de erro do sistema ou exceções geradas pelo software para suportar operações em SOs.

Cada fonte de exceção possui um número identificador da exceção. Os números de exceção de 1 a 15 são classificados como exceções do sistema e as exceções de 16 acima enumeram as interrupções. O número da interrupção é atualizado automaticamente no campo *Exception Number* do registrador *CPSR* quando a rotina específica de atendimento à interrupção é executada.

O design do *NVIC* nos processadores Cortex-M3 e Cortex-M4 pode suportar até 240 entradas de interrupção. No entanto, na prática, o número de entradas de interrupção implementadas em um chip de um fabricante específico é muito menor, geralmente na faixa de 16 a 100. Dessa forma, o tamanho do design do chip pode ser reduzido, o que também reduz o consumo de energia.

Os vetores de exceção estão localizados em uma tabela de vetores armazenada no início da primeira região do mapa de memória e o processador lê esta tabela para determinar o endereço inicial de um manipulador de exceção específico durante a sequência de entrada na exceção.

A latência de interrupção do Cortex-M3 e Cortex-M4 é muito baixa, apenas 12 ciclos de clock, o que faz com que o atendimento à interrupção seja muito rápido.

Reset é um tipo especial de exceção. Quando o processador sai de um reset, ele executa o manipulador de *Reset* no modo *Thread* (em vez do modo *Handler*, como em outras exceções).

O *NVIC* é programável e seus registradores de controle estão localizados numa área de endereçamento de memória chamada *System Control Space* (SCS), localizado na última região do mapa de memória. O *NVIC* lida com as exceções e configurações de interrupção, priorização e mascaramento de interrupção, possuindo os seguintes recursos:

- Gerenciamento flexível de exceções e interrupções;
- Suporte de exceção/interrupção aninhada;
- Entrada de exceção/interrupção vetorial;
- Máscara de interrupção;

Cada interrupção (exceto a NMI) pode ser ativada ou desativada independentemente e pode ter seu status definido ou limpo pelo software. O *NVIC* pode lidar com vários tipos de fontes de interrupção:

• Solicitação de interrupção por pulso: é a solicitação de interrupção que dura pelo menos um ciclo de clock. Quando o *NVIC* recebe um pulso na entrada de interrupção, o status pendente é definido e mantido até que a interrupção seja atendida.
• Solicitação de interrupção acionada por nível: a fonte de interrupção mantém a solicitação alta até que a interrupção seja atendida. A entrada do *NVIC* é ativa em nível alto. No entanto, a entrada de interrupção externa real no microcontrolador pode ser projetada de maneira diferente e é convertida em um nível alto de sinal ativo pela lógica no chip.

Cada exceção tem um nível de prioridade. Algumas exceções, como interrupções, têm níveis de prioridade programáveis e outras (*NMI,* por exemplo) têm um nível de prioridade fixo.

Quando ocorre uma exceção durante o atendimento de uma exceção anterior, o *NVIC* comparará o nível de prioridade dessa exceção com o nível atual. Se a nova exceção tiver uma prioridade mais alta, a tarefa em execução atual será suspensa. Alguns dos registradores serão armazenados na pilha de memória e o processador começará a executar o manipulador de exceções da nova exceção. Esse processo é chamado de "preempção". Quando o manipulador de exceções de mais alta prioridade é concluído, ele é finalizado com uma operação de retorno de exceção e o processador restaura automaticamente os registradores da pilha e retoma a tarefa que estava sendo executada anteriormente. Esse mecanismo permite aninhar serviços de exceção sem sobrecarga de software.

Quando ocorre uma exceção, o processador precisará localizar o endereço inicial do manipulador de exceções correspondente. Os processadores Cortex-M localizam automaticamente o endereço inicial do manipulador de exceções na tabela de vetores na memória. Como resultado, os atrasos desde a requisição da exceção até a execução dos manipuladores de exceção são reduzidos.

A tabela de vetores é uma matriz dentro da memória *flash* dos microcontroladores, onde cada elemento representa o endereço inicial de um manipulador de exceção.

Reset e sequência de reinicialização

Nos microcontroladores Cortex -M3/M4, pode haver três tipos de reset:

• *Power on reset* – reseta todos os subsistemas no microcontrolador. Isso inclui o processador e seus componentes de suporte à depuração e periféricos. É o reset que ocorre ao energizar/desenergizar o processador.
• *System reset* - reseta apenas o processador e os periféricos, mas não os componentes de suporte à depuração do processador;
• *Processor reset* - reseta apenas o processador.

Durante as operações de depuração do sistema ou reset do processador ou reset do sistema, os componentes de depuração nos processadores Cortex-M3 ou Cortex-M4 não são resetados para que a conexão entre o *host* de depuração (por exemplo, software de depuração em execução no computador) e o microcontrolador possa ser mantida. O *host* de depuração pode gerar um reset do sistema (*System reset*) ou do processador (*Processor reset*).

A duração do *Power on reset* e *System reset* depende do design do microcontrolador. Em alguns casos, o reset dura alguns milissegundos, pois o controlador de reset precisa aguardar que uma fonte de clock, como um oscilador externo baseado em cristal, se estabilize.

Após o reset e antes do processador iniciar a execução do programa, os processadores Cortex-M leem automaticamente as duas primeiras palavras da memória de programa. O início do espaço de memória contém a tabela de vetores, e as duas primeiras palavras na tabela de vetores são o valor inicial do *Main Stack Pointer* (*MSP*) e o vetor de reset, que é o endereço inicial do manipulador de reset. Depois que essas duas palavras são lidas, o processador configura o *MSP* e o Contador de Programas (*PC*) com esses valores, respectivamente.

Observe que, na maioria dos ambientes de desenvolvimento C, o código de inicialização C também atualiza o valor do *MSP* antes de entrar no programa principal *main()*. Essa inicialização da pilha em duas etapas permite que um microcontrolador com memória externa use essa memória para a pilha.

Dentro do manipulador de reset há uma chamada para a função *main()*, de onde o processador passa a finalmente executar o programa do usuário. ■