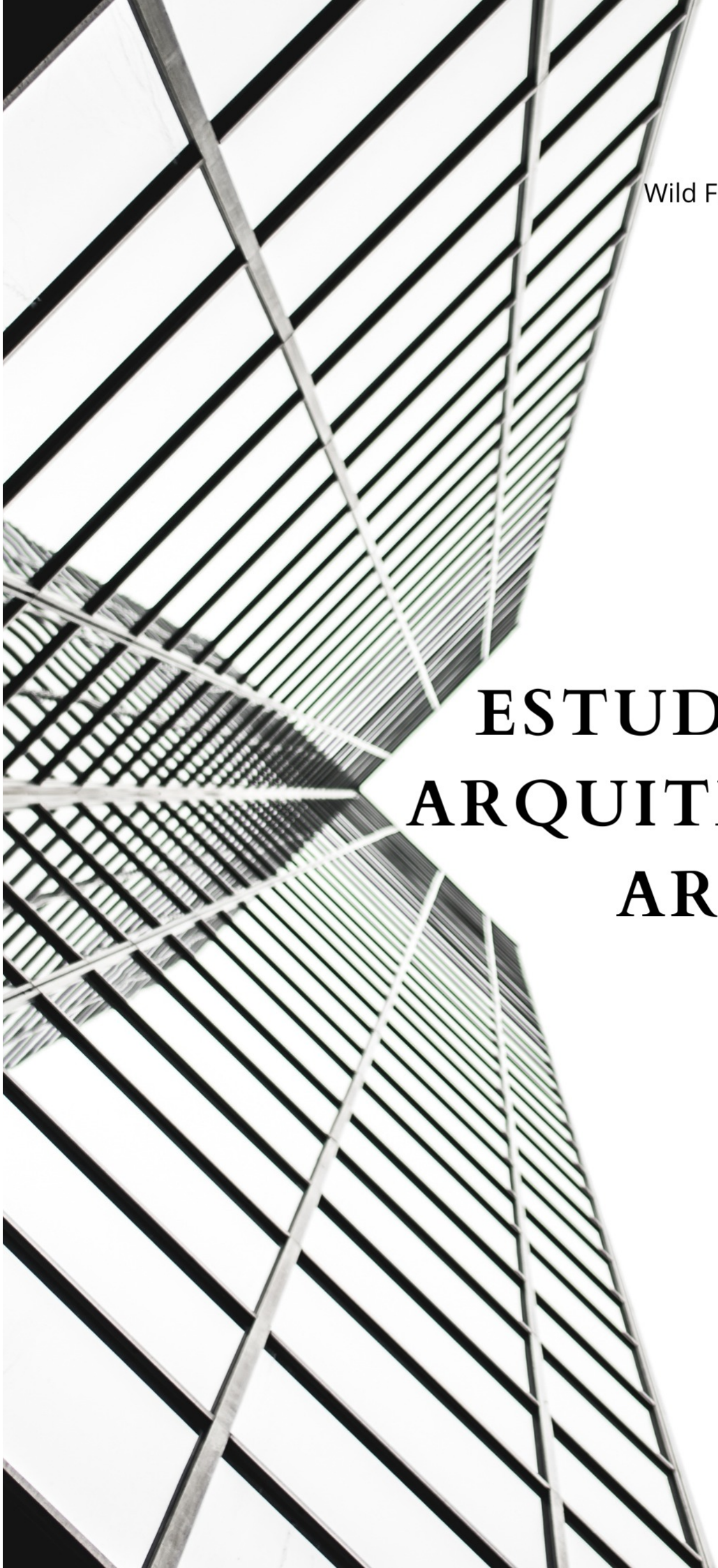

Wild Freitas da Silva Santos

ESTUDO DA ARQUITETURA ARM

EDITORA CONHECIMENTO LIVRE

WILD FREITAS DA SILVA SANTOS

ESTUDO DA ARQUITETURA ARM

1ª ed.

Piracanjuba-GO
Editora Conhecimento Livre
Piracanjuba-GO

1ª ed.

Dados Internacionais de Catalogação na Publicação (CIP)

SANTOS, WILD FREITAS DA SILVA
S237E ESTUDO DA ARQUITETURA ARM

/ WILD FREITAS DA SILVA SANTOS. – Piracanjuba-GO

Editora Conhecimento Livre, 2023

84 f.: il
DOI: 10.37423/2023.edcl669
ISBN: 978-65-5367-252-9
Modo de acesso: World Wide Web
Incluir Bibliografia

1. arquitetura-arm 2. arm7 3. família-cortex I. SANTOS, WILD FREITAS DA SILVA II. Título

CDU: 620

https://doi.org/10.37423/2023.edcl669

EDITORA CONHECIMENTO LIVRE

Corpo Editorial

Editora Conhecimento Livre
Piracanjuba-GO
2022

10.37423/2023.edcl669

Resumo: Este trabalho visa estudar a arquitetura de processadores ARM. Será estudada a base da arquitetura, a estrutura básica do núcleo do processador, seu *pipeline*, conjunto de instruções, e sistemas de memória. O foco do trabalho será descrever a base da arquitetura, diferenciando os dois processadores ARM mais utilizados, o ARM7 e o ARM9. No final serão demonstradas as tendências futuras da arquitetura ARM, descrevendo a mais recente família de processadores ARM, a família Cortex, e os seus novos recursos, além de demonstrar a evolução dos processadores ARM através de uma comparação entre os processadores ARM7 e Cortex-M3, e entre o ARM9 e o Cortex-R4.

Palavras-chave: Arquitetura ARM; ARM7; ARM9; família Cortex; Cortex-M3; Cortex-R4.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE ABREVIATURAS E SIGLAS

AMBA	Advanced Microcontroller Bus Architecture
AMP	Asymmetric Multi-Processing
APSR	Application Program Status Register
ARM	Advanced RISC Machine
ASID	Application Space Identifier
CISC	Complex Instruction Set Computer
CP15	Co-processador 15
CPSR	Current Program Status Register
CPU	Central Processing Unit
ECC	Error Correction Codes
EPSR	Execution Program Status Register
DFAR	Data Fault Address Register
DFSR	Data Fault Status Register
FIQ	Fast Interrupt Request
GPS	Global Positioning System
I/O	Input/output
ISA	Instruction Set Architecture
IPSR	Interrupt Program Status Register
IRQ	Interrupt Request
IT	If-Then
LIFO	Last In First Out
LR	Link Register
MIPS	Milhões de Instruções por Segundo
MMU	Memory Management Unit
MPU	Memory Protection Unit
NSTID	Non-Secure Table Identifier
NVIC	Nested Vectored Interrupt Controller
PA	Physical Address
PC	Program Counter

PDS Processamento Digital de Sinais

PFU Prefetch Unit

PIN Personal Identification Number

PMSA Protected Memory System Architecture

RAM Random Access Memory

RISC Reduced Instruction Set Computer

RTI Rotina de Tratamento de Interrupção

SIMD Single Instruction, Multiple Data

SMP Symmetric Multi-Processing

SP Stack Pointer

SPSR Saved Program Status Register

SWI Software Interruption

SysTick System Tick

TCM Tightly Coupled Memory

Thumb-2EE Thumb-2 Execution Environment

TLB Translation Lookaside Buffers

ULA Unidade Lógica e Aritmética

VA Virtual Address

VMSA Virtual Memory System Architecture

WFE Wait For Event

WFI Wait For Interrupt

Sumário

1 INTRODUÇÃO

Os microcontroladores de 32 bits ARM (inicialmente Acorn RISC Machine, atualmente Advanced RISC Machine) são atualmente um dos principais chips utilizados em aplicações de sistemas embarcados, representando cerca de 60% dos microcontroladores utilizados na China para essa finalidade, segundo a Information Quartely (2005), e dentre eles, os microcontroladores ARM7 e ARM9 são os mais utilizados atualmente.

Ambos, como qualquer processador ARM, seguem a filosofia RISC (Reduced Instruction Set Computer) de microprocessadores, ou seja, ele implementa em hardware um conjunto reduzido de instruções que possuem o tempo de execução de 1 ciclo do clock do processador, impossibilitando a quebra de pipeline e não utilizando micro-códigos para pré- configurações de execução, o que reduziria seu desempenho.

Porém, para seguir os principais conceitos por trás da arquitetura ARM, que são a simplicidade, o baixo custo, pequeno consumo energético e modularidade, modificaram-se assim os conceitos de projeto RISC e acrescentaram novas características a fim de atender os mais diversos perfis de aplicação.

Uma dessas características importantes para a utilização do microcontrolador ARM em sistemas embarcados é a questão do consumo de energia. Além de ter sido totalmente projetado com a finalidade de usar a menor quantidade possível de energia, ele também foi projetado de tal maneira que o programador tem acesso ao sistema regulador de tensão, ao subsistema de clock, entre outros recursos que possibilitam reduzir ainda mais o consumo de energia.

Outro recurso interessante no microcontrolador ARM são as instruções Thumb. Normalmente, as instruções do ARM possuem o tamanho de 32 bits, porém quando se utiliza uma instrução no modo Thumb, este possui o tamanho de 16 bits, metade do tamanho normal. Tal redução tem como finalidade a redução do uso da memória, trazendo uma maior densidade de código, e dependendo do caso, numa maior velocidade de processamento. Porém, dependendo da instrução e da situação, tal modo pode ter o efeito contrário, tornando o programa mais lento. Isto ocorre por que as instruções Thumb são menos flexíveis e não permitem a realização de diversas operações simultaneamente como no modo ARM.

Assim, a presente monografia pretende descrever a base da arquitetura ARM e seus recursos mais importantes, destacando as diferenças entre os dois principais processadores, o ARM7 e o ARM9, como em relação ao sistema de memória, não apenas a modificação da memória cache, mas também com relação ao acesso e gerenciamento de memória, com o acréscimo da MMU (Memory Management Unit) e da MPU (Memory Protection Unit), além de mostrar as tendências futuras da arquitetura ARM.

No primeiro capítulo será descrito a filosofia de projeto dos microcontroladores ARM, as diretrizes principais, descrevendo as características adicionadas ao projeto tradicional RISC, que ajudaram a alcançar o público alvo da arquitetura, os sistemas embarcados.

No segundo capítulo será o início da descrição da arquitetura, diferenciando os processadores, e introduzindo os conceitos implementados pela ARM, como o conjunto de instruções Thumb.

O terceiro capítulo continuará a descrição da arquitetura, dando um foque nos sistemas de memória, como a memória cache, a VMSA (Virtual Memory System Architecture) com a MMU, e a PMSA (Protected Memory System Architecture) com a MPU, e assim, continuando a diferenciação entre os dois processadores.

Por fim, o último capítulo tratará das tendências futuras da arquitetura ARM, descrevendo a mais nova família de processadores, a família Cortex, e seus novos recursos, como o Thumb-2, o NEON e o TrustZone, e em seguida comparando os novos processadores com os seus antecessores correspondentes.

2 FILOSOFIA DE PROJETO DOS PROCESSADORES ARM

Segundo Pereira (2007), a arquitetura ARM de microcontroladores possui quatro princípios básicos que devem ser seguidos em todo seu projeto, que são a simplicidade, o baixo custo, o pequeno consumo energético e a modularidade.

Logo, seguindo esses princípios, os núcleos dos microcontroladores ARM utilizam a filosofia de arquitetura RISC onde tal filosofia de projeto concentra-se, segundo Hamacher, Vranesic e Zaky (1996), na redução da complexidade das instruções executadas pelo hardware, resultando numa menor complexidade deste e provendo uma maior flexibilidade no software.

A filosofia RISC implementa basicamente quatro regras de projeto, que são:

1. instruções – Os processadores RISC possuem um menor número de instruções implementadas em hardware. Esta característica provê uma maior simplicidade das operações a serem executadas pelo processador, facilitando sua implementação para serem executadas em um ciclo do clock. Com isso, é de responsabilidade do compilador traduzir instruções mais complexas em conjunto de instruções simples que possuem tamanho fixo;
2. pipelines – O processamento de instruções pode ser quebrado em pequenas unidades que podem ser executadas em paralelo pelo pipeline. Idealmente, o avanço do pipeline é realizado a um passo por ciclo do clock. As instruções podem ser decodificadas em um único estágio do pipeline, não havendo necessidade da instrução ser executadas através de microprogramas chamados micro-códigos, como ocorrem nos processadores CISC (Complex Instruction Set Computer);
3. registradores – Os processadores RISC possuem um grande banco de registradores de propósito geral, onde se pode conter um dado ou um endereço em cada um deles. Tais registradores atuam como uma memória de rápido acesso para todos os dados que estão sendo processados;
4. arquitetura load-store - Os processadores RISC operam com os dados armazenados nos registradores. Separam as instruções load e store, transferindo os dados entre o banco de dados e a memória externa. O acesso a memória é dispendiosa, logo a separação do acesso a memória do processamento de dados é um grande avanço porque possibilita que diferentes dados possam ser salvos no banco de registradores sem a necessidade de acessos múltiplos a memória.

Estas regras descritas permitem que o processador RISC possa ser simples, e que o seu núcleo opere em altas freqüências do clock, sendo assim, indo de acordo com o princípio da simplicidade dos processadores ARM.

Há ainda uma série de recursos físicos implementados na filosofia de projeto dos processadores ARM, para seguir com o princípio do baixo consumo de energia e a fim de atender os mais diversos perfis de aplicação.

Estes recursos vão desde a criação do projeto do processador com o intuito de reduzir ao máximo o seu consumo de energia, até o gerenciamento de energia e otimização da aplicação, como, por exemplo, a otimização do tamanho e forma de uso das caches, possíveis nos ARM9 e posteriores, já

que este é responsável por cerca de 30% do consumo de energia do chip, segundo Verma e Marwedel (2007).

Outro aspecto é o reduzido tamanho do chip ARM, que vem a atender dois pontos importantes para sistemas embarcados. O primeiro ponto é em relação a área disponível para o processador embarcado, pois para uma solução de um único chip, quanto menor a área utilizada para o processador, maior a área disponível para os periféricos especializados.

O segundo ponto, segundo Verma e Marwedel (2007), é o tamanho da área de silício, esta diretamente relacionada com o consumo de energia. Logo quanto menor for o chip, possivelmente menor será o consumo deste.

Todavia, outra pretensão básica da arquitetura ARM é atender os mais diversos perfis de aplicação, sendo assim, existe uma série de recursos destinados a otimização máxima da aplicação, como por exemplo, a alta densidade de código.

A densidade de código é um recurso importante para sistemas embarcados que possuem memória limitada pelo custo ou por restrições físicas. Este recurso é implementado nos processadores ARM graças às instruções Thumb, instruções com apenas 16 bits de tamanho.

Sendo assim, o núcleo ARM não é puramente de arquitetura RISC devido a restrições da sua aplicação primária a qual foi projetado, os sistemas embarcados. Segundo Pereira (2007), a força dos núcleos ARM é justamente devida que este não implementou totalmente os conceitos RISC.

Outros aspectos que diferenciam as instruções ARM das provindas da definição pura RISC, de acordo com Sloss, Symes e Wright (2004), são, por exemplo:

- variação do tempo de execução por determinadas instruções – Não é toda instrução ARM que é executada em um único ciclo. Por exemplo, as instruções múltiplas de load/store variam em número de ciclos para executá-las dependendo do número de registradores a serem transferidos. A transferência pode ocorrer com acesso seqüencial de memória, que aumenta o desempenho, desde que o acesso a memória seqüencial seja mais rápida do que o acesso randômico.
- instruções Thumb de 16 bits – o processador ARM possui um grupo de instruções de

- 16 bits chamado Thumb que permite ao núcleo do processador executar tanto instruções de 16 bits, quanto de 32 bits. As instruções de 16 bits melhoram a densidade de código em cerca de 30% sobre as instruções de 32 bits de tamanho;
- execução condicional – Há instruções que só são executadas quando condições especificas são satisfeitas. Esta característica melhora o desempenho e a densidade do código pela redução das instruções;
- instruções melhoradas – Um melhor grupo de instruções de processamento digital de sinais (PDS) foi adicionado às instruções básicas ARM, suportando operações de multiplicação de 16×16-bits e saturação. Estas instruções permitem o melhor desempenho do processador ARM em alguns casos onde se trocam as tradicionais combinações de instruções do processador pelas de PDS.

Assim, todas essas características adicionais têm feito o processador ARM ser o mais comumente núcleo de processador embarcado de 32 bits a ser usado, sendo o ARM7 o microcontrolador de 32 bits mais utilizado no mundo, segundo ARM (2010), já que, muitas vezes em sistemas embarcados atuais, a velocidade do processador não é o ponto chave, mas sim o efeito total do desempenho do sistema e do consumo de energia deste.

3 A ARQUITETURA ARM

3.1 ESTRUTURAÇÃO INTERNA

Afim de sempre melhorar o desempenho de seus processadores, a ARM modificou a estrutura interna básica de seus processadores ao longo do tempo, onde uma grande modificação ocorreu dos processadores ARM7 para o ARM9.

Até então, os processadores ARM7 e seus antecessores utilizavam a arquitetura do tipo Von Neumann, onde existe apenas uma cache única, utilizada tanto para dados, quanto para instruções, segundo Hamacher, Vranesic e Zaky (1996), como pode ser visualizado na Figura 1.

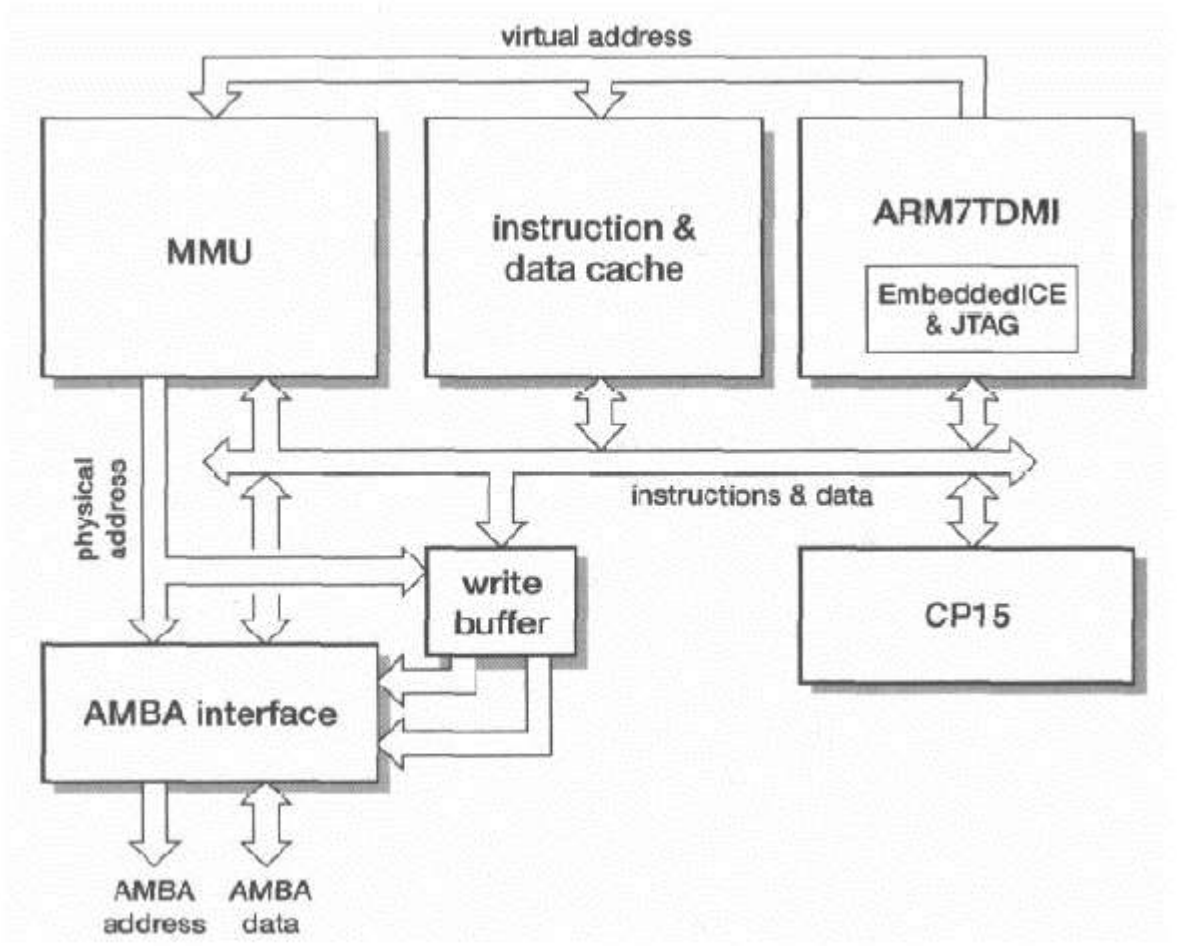

Figura 1 – Diagrama de blocos do núcleo do ARM710T. Fonte: Furber, 2000.

No diagrama de bloco da Figura 1, pode ser visualizado o núcleo do processador, que no caso é um ARM7TDMI, a estruturação do tipo Von Neumann, como foi descrito acima, com o seu único barramento para dados e instruções e sua *cache* única, o CP15 (Co- processador 15), e a interface do barramento AMBA (*Advanced Microcontroller Bus Architecture*).

Além disso, o ARM710T foi um dos primeiros processadores ARM a incorporar a MMU na sua estrutura interna básica, trazendo uma série de novas aplicações possíveis e de recursos, e tal acréscimo sendo solidificado com a família ARM9, como pode ser visualizado na Figura 2.

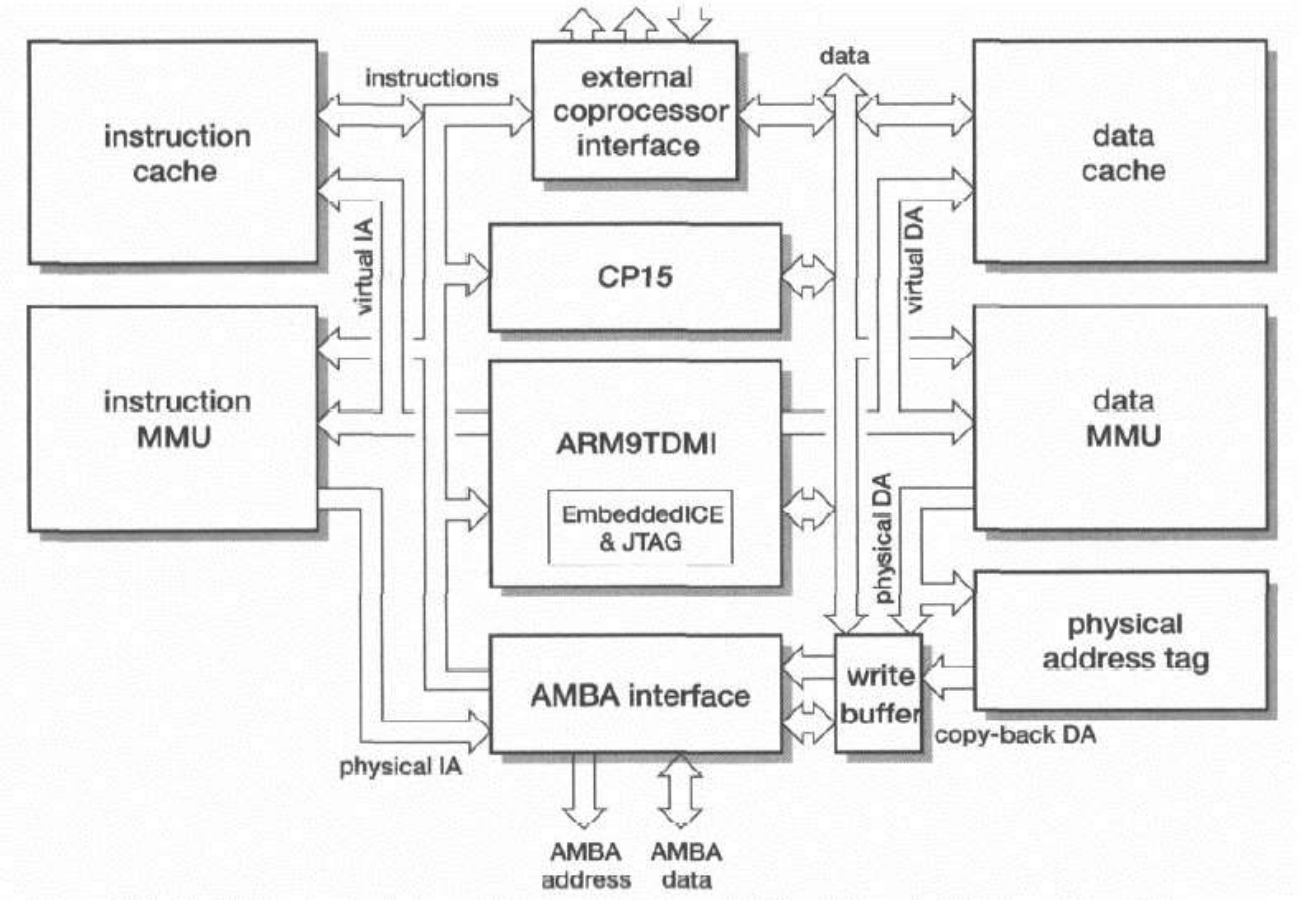

Figura 2 – Diagrama de blocos do núcleo do ARM920T. Fonte: Furber, 2000.

Já no diagrama de blocos do ARM920T, pode-se visualizar a diferença entre a arquitetura Von Neumann e a arquitetura Harvard, que então esta sendo utilizada, com duas caches separadas, cache de dados e a cache de instruções, e dois barramentos exclusivos para cada. Além disso, também a uma separação da MMU, em uma para dados e outra parte para instruções e um barramento próprio e único para a MMU acessar cada cache, a virtual IA e a virtual DA.

Há também na família ARM9 a adição da interface de co-processadores externos, já que a partir dessa família começa a existir a adição opcional de co-processadores, como por exemplo, o Jazelle, para processamento de instruções Java, e o MOVE, para processamento de instruções relativas ao MPEG4, sendo esta interface a responsável pela comunicação deste como o processador principal, e pelo acesso do programador a instruções destes.

3.2 TIPO DE DADOS

De acordo com ARM (2000a), os processadores ARM podem suportar os seguintes tipos de dados:

- bytes – 8 bits;
- halfword – 16 bits (Halfwords devem ser alinhados para dois bytes de limite);
- word - 32 bits (Words devem ser alinhados para quatro bytes de limite).
- Existem algumas restrições importantes sobre a manipulação dos diferentes tipos de dados, algumas delas são:
- quando qualquer um desses tipos é descrito como unsigned, o valor do dado de N-bits representa um numero inteiro não-negativo entre 0 a 2(N+1), usando o formato binário normal;
- quando qualquer um desses tipos é descritos como signed, o valor do dado de N-bits representa um inteiro de - 2(N+1) a 2(N+1) - 1, usando o formato de complemento de dois;
- todas as operações de manipulação de dados, como por exemplo, o ADD, são projetado para dados do tipo Word;
- instruções load/store podem manipular os três tipos de dados na memória;
- instruções do tipo ARM são exatamente uma Word, enquanto instruções do tipo
- Thumb são exatamente uma halfword.

3.1 PIPELINE

Hamacher, Vranesic e Zaky (1996) descrevem o *pipeline* como o mecanismo que o processador realiza em quebrar uma instrução em etapas a serem executadas separadamente, possibilitando a execução de instruções em paralelo. No caso do ARM7, este possui três etapas de *pipeline*: *fetch*, *decode* e *execute*.

“Na etapa *fetch* é realizada a busca da instrução na memória, a etapa *decode* que decodifica a instrução a ser executada, e a etapa *execute* é a etapa de execução da instrução e escreve o resultado de volta ao registrador”. (SLOSS et al., 2004, p. 30) A Figura 3 ilustra o *pipeline* de três estágios utilizado pela família ARM7.

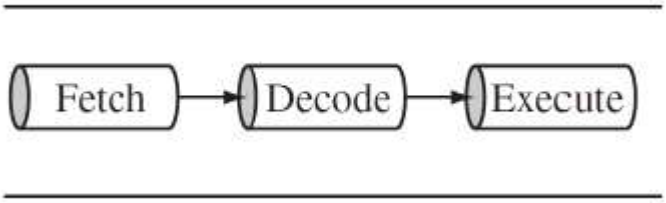

Figura 3 – Pipeline de três estágios do ARM7. Fonte: Sloss et al., 2004.

Porém, “aumentando-se o tamanho do *pipeline*, aumentando o número de seus estágios, o trabalho realizado a cada etapa é reduzido, permitindo que o processador execute em altas freqüências, aumentando o desempenho deste.” (SLOSS et al., 2004, p. 30)

Logo, com o intuito de melhorar o desempenho dos processadores, atendendo a necessidade de mercado, cada família ARM possui a estruturação do *pipeline* diferente.

Por exemplo, o ARM9, possui o *pipeline* de cinco estágios, como mostrado na Figura

4. Adicionando-se os estágios *memory* e *write* após o *execute*, “[...] foi possível ao ARM9 alcançar o processamento médio de 1.1 Dhrystone MIPS (Milhões de Instruções Por Segundo) por MHz, o que significa um crescimento em torno de 13% comparado com o ARM7.” (SLOSS et al., 2004, p. 31)

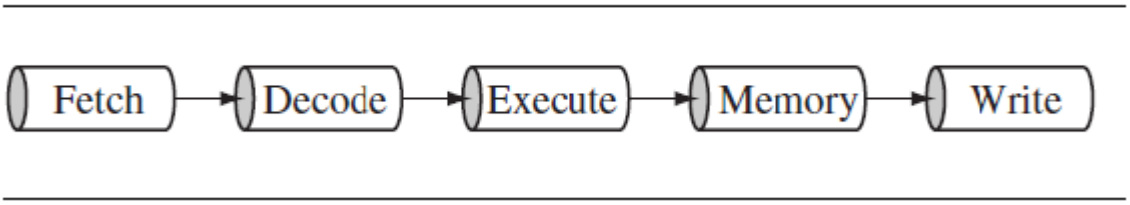

Figura 4 – Diagrama do pipeline de cinco estágios do ARM9. Fonte: Sloss et al., 2004.

Vale ressaltar que, Sloss, Symes e Wright (2004) afirmam que apesar do ARM9 possuir *pipeline* diferente, ele continua possuindo a mesma característica de execução de *pipeline* do ARM7, ou seja, um código escrito para o ARM7 poderá ser executado no ARM9. Para demonstrar essa compatibilidade, basta verificar a organização dos dois tipos de *pipeline* na Figura 5:

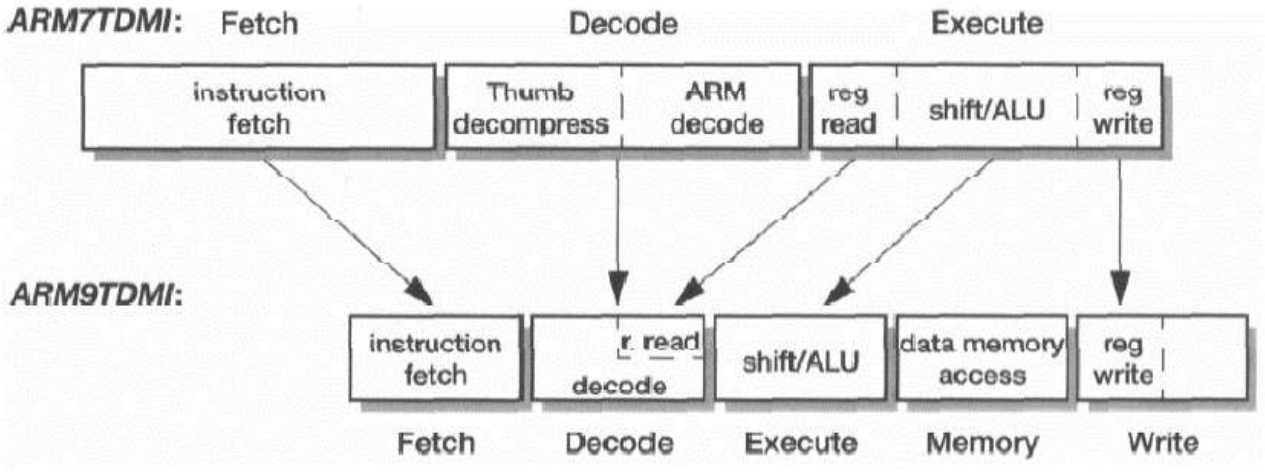

Figura 5 – Comparação entre o *pipeline* do ARM7 com o do ARM9. Fonte: Furber, 2000.

Como pode ser visualizado na Figura 5, o estágio fetch continua o mesmo, o decode possui o estágio de leitura dos registradores, que originalmente era do execute, acrescido a ele, o execute unicamente com a parte de shift e da ULA (Unidade Lógica e Aritmética), enquanto que a escrita nos registradores é feito pelo novo estágio write, demonstrando-se assim a total compatibilidade de código entre ambos processadores.

3.4 ESTADOS DA CPU

São disponíveis na arquitetura básica ARM dois tipos de estados da CPU (Central Processing Unit): o estado ARM e o estado Thumb. O estado ARM da CPU é composto por um conjunto de 34 instruções de 32 bits, enquanto o estado Thumb é composto por um conjunto de 30 instruções de 16 bits.

“Tecnicamente, as instruções do estado Thumb são um subconjunto das instruções ARM comprimidas de 32 para 16 bits, que são descomprimidas em tempo real (sem perda de

desempenho) e convertidas nas instruções ARM equivalentes, antes de serem executadas.” (PEREIRA, 2007, p. 69)

Como resultado da largura reduzida das instruções, de acordo com Pereira (2007), elas possuem determinadas restrições como:

- acesso direto a apenas 8 registradores da CPU;
- inexistência de instruções condicionais, exceto as de desvio condicionais;
- inexistência de instruções para os co-processadores;

- impossibilidade de alterar o modo de processamento;
- impossibilidade de efetuar deslocamento de um dado em conjunto com uma operação da ULA;
- impossibilidade de efetuar certas operações como multiplicação e acúmulo.

Apesar da série de restrições impostas pelo modo Thumb, esse provê um aumento significativo da densidade de código em relação às instruções ARM. “Uma seqüencia de código Thumb tende a ocupar cerca de 30% menos memória que a mesma seqüência codificada com instruções ARM”. (PEREIRA, 2007, p. 69)

Entretanto, “[...] o código utilizando apenas instruções ARM tende a ser 40% mais rápido, já que são necessárias mais instruções Thumb para executar o mesmo trabalho”. (PEREIRA, 2007, p. 69)

Além disso, o desempenho dos modos é diretamente dependente do tipo de memória que esta se trabalhando, pois, segundo Furber (2000), com um sistema de memória de 32 bits, o código em ARM é cerca de 40% mais rápido do que em Thumb, porém este é mais rápido em cerca de 50% do que o código em ARM se for utilizado um sistema de memória de 16 bits.

Como resultado dessas diferenças entre os estados da CPU, “[...] de maneira geral, utiliza-se códigos ARM nas rotinas em que a temporização é crítica, ou seja, onde a velocidade é fundamental. No restante do programa, utiliza-se normalmente código Thumb por uma simples questão de economia de memória”. (PEREIRA, 2007, p. 70)

Para uma melhor visualização da diferença de códigos e da densidade de códigos resultantes do modo Thumb, segue-se os dois programas abaixo retirados de Furber (2000), o primeiro em modo ARM, e o outro sendo sua tradução para o modo Thumb.

```
            Area        HelloW, CODE, READONLY
SWI_WriteC              EQU         &0          ; output character in r0
SWI_Exit                EQU         &11         ; finish program
            ENTRY                               ; code entry point
START ADR       r1, TEXT                        ; r1 -> "Hello World"
LOOP        LDRB        r0, [r1], #1            ; get the next byte
            CMP         r0, #0                  ; check for text end
            SWINE SWI_WriteC                    ; if not end print ..
            BNE         LOOP                    ; .. and loop back
            SWI         SWI_Exit                ; end of execution
TEXT        =           "Hello World", &0a, & 0d, 0 ;
END                                             ;end of program source
```

Código em modo ARM

```
            Area        HelloW_Thumb, CODE, READONLY
SWI_WriteC              EQU         &0          ; output character in r0
SWI_Exit                EQU         &11         ; finish program
            ENTRY                               ; code entry point
            CODES2                              ; enter in ARM state
            ADR         r0, START +1            ; get Thumb entry adress
            BX          r0                      ; enter Thumb area
            CODE16                              ; Thumb code follows..
START ADR       r1, TEXT                        ; r1 -> "Hello World"
LOOP        LDRB        r0, [r1]                ; get the next byte
            ADD         r1, r1, #1              ; increment pointer   **T
            CMP         r0, #0                  ; check for text end
            BEQ         DONE                    ; finished?   **T
            SWI         SWI_WriteC              ; if not end print...
            B           LOOP                    ;.. and loop back
DONE        SWI         SWI_Exit                ; end of execution
            ALIGN                               ; to ensure ADR works
TEXT        DATA                                ;
                "Hello World", &0a, & 0d, 00 ;
END                                             ;
```

Código em modo *Thumb*

Comparando os dois códigos acima, pode-se observar que o código em modo Thumb necessita de duas instruções para compensar as características que faltam as suas instruções, sendo sinalizadas com o "**T" aos comentários ao lado.

Em relação ao tamanho do código, o código em ARM do programa acima possui seis instruções mais 14 bytes de dados, ou seja, um total de 38 bytes de tamanho. Já no código em Thumb, este possui oito instruções mais 14 bytes de dados (ignorando o preâmbulo requerido para a troca no processador para a execução das instruções em Thumb), totalizando 30 bytes de tamanho, ou cerca de 80% do código em ARM.

Além disso, segundo Furber (2000), este exemplo serve para ilustrar outros pontos importantes para se ter em mente quando esta se escrevendo um código em Thumb, como:

- o compilador necessita saber quando será produzido código em ARM e quando será produzido código em Thumb. Neste exemplo, a diretivas "CODES 2" e "CODE16" fornecem essa informação. Já em linguagem C, utilizando, por exemplo, o compilador IAR, é possível utilizar os identificadores _arm e _thumb para forçar o compilador a gerar código em ARM ou em Thumb, respectivamente, para cada função especificada. Neste caso, o compilador se encarregará de usar a instrução BX (desvio com mudança de estado), para realizar a chamada com alteração do estado da CPU.
- no código Thumb, o "ADR" pode somente gerar um endereço de word. Como as instruções Thumb são halfwords, não há garantia que o local de memória seguinte será alinhado como word, já que possuirá um número arbitrário de instruções Thumb. Entretanto, o programa exemplo possui explicitamente o "ALIGN" antes da string de texto.

Uma observação importante em relação à mudança de estado é que após uma exceção (um evento que implique na mudança de fluxo do programa) ou um reset (que também é um tipo de exceção), a CPU automaticamente é forçada para o estado ARM e o pipeline é esvaziado, garantindo assim que este seja preenchido apenas com instruções ARM. Retornando da exceção, o estado anterior é restaurado.

3.5 REGISTRADORES

A arquitetura ARM possui o total de 37 registradores de 32 bits, sendo que, segundo Dandamudi (2005), os registradores são divididos em dois grupos:

- registradores de propósito geral: Existem 31 registradores de propósito geral. Destes, o usuário pode acessar apenas 16 registradores (R0 – R14 e o Program Counter);
- Program Status Register: O ARM possui 6 registradores de status onde armazena o status do programa. O Current Program Status Register (CPSR) é disponível em todos os modos do processador. A visibilidade dos outros registradores depende do modo de operação.

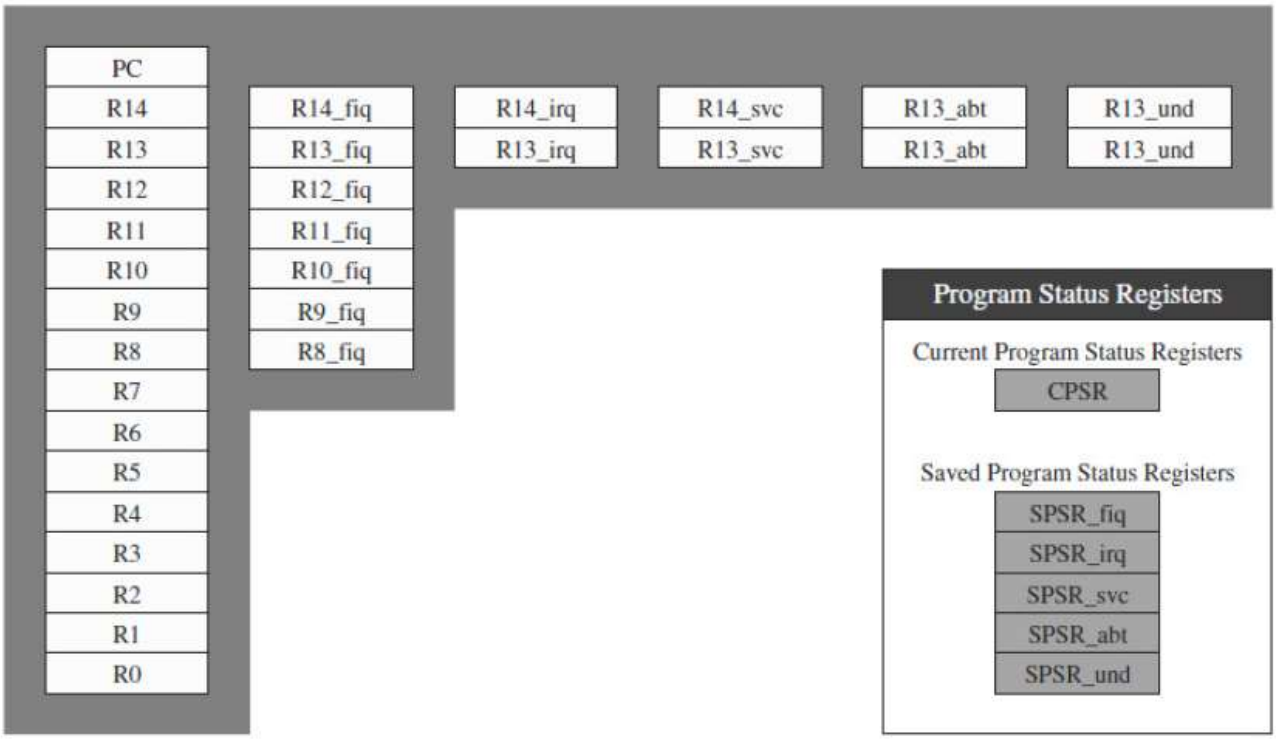

Figura 6 – Os registradores da CPU da arquitetura ARM. Fonte: Dandamudi, 2005.

3.5.1 REGISTRADORES DE PROPÓSITO GERAL

Os registradores de propósito geral podem ser divididos em três categorias: os registradores baixos R0-R7, os registradores altos R8-R14 e o PC (Program Counter).

Segundo ARM (2000a), os registradores R0 a R7, chamados de registradores baixos, são os unbanked registers, ou seja, fisicamente eles são os mesmo 32 bits para a mesma referência em todos os modos do processador. Eles são completamente de propósito geral, com nenhum tipo de restrição, podendo ser utilizados em qualquer instrução onde é especificada com um registrador de propósito geral.

Em relação aos registradores R8 a R14, esses são chamados por ARM (2000a) de registradores altos ou de banked registers, ou seja, os registradores físicos referentes a estes dependem de que modo de processamento se encontra o processador.

Já o registrador R15 é reservado como PC, ou seja, este registrador "aponta para o endereço de memória em que a próxima instrução será buscada." (PEREIRA, 2007, p. 71), lembrando que a cada instrução ARM executada, o PC é incrementado em 4 bytes, já que cada instrução possui 32 bits.

Ele é considerado um registrador de propósito geral porque este "[...] pode assumir o lugar de um dos outros registradores de propósito geral. Entretanto, este possui uma série de restrições e instruções específicas para a sua manipulação." (ARM, 2000a, p. A2-7).

3.5.2 REGISTRADORES DO ESTADO THUMB

"No estado Thumb, apenas os oito registradores (R0 a R7) são diretamente acessíveis à maioria das instruções (também denominados registradores baixos)" (PEREIRA, 2007, p. 73), porém algumas instruções têm o direito de acesso aos registradores altos (R8 a R15).

Neste estado, "[...] o registrador R13 é normalmente utilizado como apontador de pilha (SP)" (ARM, 2000a, p. A2-6), sendo este implementado com estrutura LIFO (Last In First Out), ou seja, último a entrar, primeiro a sair.

O registrador R14, LR (Link Register), "[...] não pode ser acessado pela maioria das instruções, mas é utilizado implicitamente nas instruções de chamada e retorno de sub-rotina." (PEREIRA, 2007, p. 73)

O registrador R15, PC, igualmente, não pode ser acessado pela maioria das instruções. Diferentemente do PC no modo ARM, no modo Thumb o PC incrementa 2 bytes a cada instrução executada, devido ao tamanho das instruções nesse modo possuírem apenas 16 bits.

Quadro 1 – Registradores dos diferentes estados. Fonte: Pereira, 2007.

Modo ARM	Modo *Thumb*	Função
Registrador	Registrador	
R0	R0	Uso Geral
R1	R1	
R2	R2	
R3	R3	
R4	R4	
R5	R5	
R6	R6	
R7	R7	
R8		
R9		
R10		
R11		
R12		
R13	SP	Apontador da Pilha (SP)
R14	LR	Endereço de Retorno (LR)
R15	PC	Contador de Programa (PC)
CSPR	CSPR	Estado da CPU

3.5.3 CURRENT PROGRAM STATUS REGISTER (CPSR)

"O registrador CPSR é um registrador de 32 bits que é utilizado para monitorar e controlar operações internas do processador, [...] e é dividido em quatro campos de 8 bits cada: flags, status, extensão e controle." (SLOSS et al., 2004, p. 22)

O campo de flags são os 8 bits mais significativos, sendo que os quatro bits mais significativos são os bits referentes ULA: bit N (indicador de negativo), bit Z (Indicador de zero), bit C (indicador de transporte/empréstimo) e bit V (indicador de overflow).

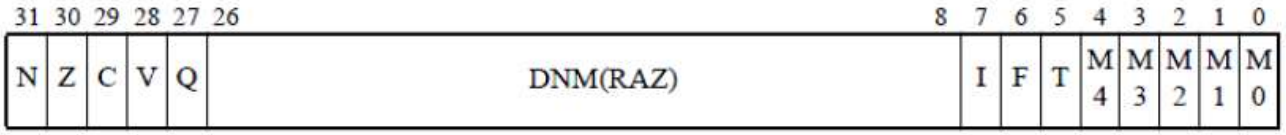

Figura 7 – Current Program Status Register. Fonte: ARM, 2000a.

• N: o flag N é setado sempre que o resultado de uma operação é negativo, ou seja, para valores binários em que o bit mais significativo do resultado é igual a “1”.

0n– resultado positivo ou zero.

1 – resultado negativo.

• Z: o flag Z é setado sempre que o resultado de uma operação é igual a zero (todos os

bits iguais a “0”).

0 – resultado diferente de zero.

1 – resultado igual a zero.

• C: após uma operação de adição ou comparação (CMN):

0 – não houve transbordo (transporte) no resultado da operação.

1 – houve transbordo (transporte) no resultado da operação.

Após uma operação de subtração ou comparação (CMP):

0 – houve empréstimo para a realização da subtração (resultado negativo).

1 – não houve empréstimo para a realização da subtração.

• V: esse flag sinaliza um estouro de representação quando utilizado em operações aritméticas (adição, subtração e comparação) que envolvem operandos sinalizados. O flag V é setado sempre que uma das operações descritas resultarem em um número sinalizado maior do que 2.147.483.647 ou menor que -2.147.483.648.

O bit seguinte, o bit Q, é utilizado em instruções dedicadas para PDS, e é chamado de

bit de saturação. Este flag é setado nas seguintes situações:

- saturação do resultado de uma adição utilizando as instruções QADD ou QDADD;
- saturação do resultado de uma subtração utilizando as instruções QSUB ou QDSUB;
- overflow sinalizado durante a instrução SMLA ou SMLAW.

O flag Q é setado pelo resultado da instrução, não sendo modificado pelas instruções posteriores, e deverá ser zerado pelo código da aplicação.

O campo de controle são os 8 bits menos significativos, sendo estes manipulados pela código da aplicação. São eles:

- I: bit de controle das interrupções IRQ (Interrupt Request). Quando o bit I está setado (nível "1"), as interrupções IRQ estão desabilitadas.
- F: bit de controle das interrupções FIQ (Fast Interrupt Request). Quando o bit F está setado (nível "1"), as interrupções FIQ estão desabilitadas.
- T: o bit T é o bit Thumb, ou bit de controle de estado da CPU, Esse bit controla o estado da CPU da seguinte maneira:

0 – Estado ARM 1 – Estado Thumb

Apesar da possibilidade de manipulação deste bit diretamente pelo programa em execução (nos modos privilegiados), o programador nunca devera fazê-lo. Para qualquer mudança do estado da CPU, deverá ser utilizada a instrução assembly BX.

Os últimos cinco bits menos significativos são os bits de modo de operação. O ARM possui 7 diferentes modos de operação. Pode-se setar os diferentes modos de operação da seguinte maneira:

M4	M3	M2	M1	M0	Modo
1	0	0	0	0	*User*
1	0	0	0	1	*FIQ*
1	0	0	1	0	*IRQ*
1	0	0	1	1	*Supervisor*
1	0	1	1	1	*Abort*
1	1	0	1	1	*Undefined*
1	1	1	1	1	*System*

Quadro 2 – *Bits* de seleção de modo de operação. Fonte: Pereira, 2007.

Os 16 bits restantes são os campos de extensão e status que são bits reservadores para uso futuro. Alguns processadores ARM possuem alguns bits extras alocados. Por exemplo, o bit J encontrado no campo de flags, disponível para os co-processador com Jazelle, que executa instruções de 8 bits da maquina virtual Java.

3.6 MODOS DE PROCESSAMENTO

Na arquitetura ARM implementada nos processadores ARM7 e ARM9, existe 7 tipos diferentes de modo de processamento, sendo que estes podem ser divididos em dois grupos: modo não-privilegiado (User) e modos privilegiados (System, Supervisor, Abort, Undefined, Interrupt e Fast Interrupt).

A diferença básica entre os dois modos é que no modo privilegiado é permitido ao programa alterar algumas características de operação da CPU, como a habilitação/desabilitação das interrupções e alteração do modo de processamento, que são especialmente utilizados pelo núcleo de sistemas operacionais, enquanto o modo não- privilegiado não é permitido alterá-los.

3.6.1 MODO USER (USR)

Pereira (2007) descreve o modo User como o modo não-privilegiado e utilizado para operações cotidianas do programa. O programa em execução não pode alterar o modo de operação nem habilitar/desabilitar interrupções.

Com isso, as únicas maneiras de se sair desse modo de operação são “[...] o reset ou a execução de uma interrupção em software (SWI). Em ambos os casos, o novo modo de operação passa a ser o supervisor.” (PEREIRA, 2007, p. 76).

3.6.2 MODO SYSTEM (SYS)

O modo privilegiado System é normalmente empregado para a execução de sistemas operacionais, já que este modo “permite acesso ao mesmo conjunto de registradores do modo usuário, mas não possui as limitações dele”. (PEREIRA, 2007, p. 76) Lembrando que como este modo é privilegiado, ele pode selecionar qualquer outro modo de processamento a partir dele.

3.6.3 MODO SUPERVISOR (SVC)

“O modo privilegiado Supervisor é automaticamente selecionado após um reset ou uma interrupção por software (SWI)” (PEREIRA, 2007, p. 76), onde neste caso, o endereço de retorno (o endereço da

instrução seguinte a SWI) é salvo no registrador LR (R14) e o conteúdo do CPSR é salvo no registrador SPSR_svc (Saved Program Status Register).

Seqüencialmente, a CPU setar os bits corretos de M4 a M0 do registrador CPSR, no caso seta 10011, seleciona o estado ARM, ou seja, bit T do CPSR torna-se 0, e desabilita as interrupções IRQ, bit I=1 do CPSR.

“Após isso, o PC (R15) é carregado com o valor 0x00000008, o que faz com que o fluxo do programa seja desviado para esse endereço. O pipeline é esvaziado e ele passa a ser preenchido pelas com as instruções do novo endereço e seguintes”. (PEREIRA, 2007, p. 77)

Para poder sair deste modo de processamento, deve-se executar a instrução MOVS PC, R14, ou seja, copiar o conteúdo do registrador R14_svc para o PC (R15) e o do SPSR_svc para o CPSR. “Após a execução dessa instrução, o conteúdo do pipeline é novamente esvaziado e carregado com as instruções seguintes à SWI”. (PEREIRA, 2007, p. 77)

Vale ressaltar que neste modo de operação existem os registradores R13_svc e R14_svc, que são fisicamente diferentes dos registradores R13 e R14 do modo User, igualmente o registrador SPSR_svc é fisicamente diferente dos seus demais semelhantes.

3.6.4 MODO ABORT (ABT)

O modo Abort é outro tipo de “modo privilegiado utilizado para o processamento dos chamados aborts, que consistem basicamente na tentativa de acesso a posições de memória não implementadas ou permitidas”. (PEREIRA, 2007, p. 77)

O modo Abort possui duas modalidades, uma “[...] por busca de instrução em endereço não permitido (prefetch abort) e outra por acesso de dado em endereço não permitido (data abort)”. (PEREIRA, 2007, p. 77)

3.6.4.1 PREFETCH ABORT

O prefetch abort “[...] somente ocorre ao tentar executar uma instrução buscada em uma posição de memória inválida ou não permitida” (PEREIRA, 2007, p. 77), assim então a instrução é abortada. Observe que o nome prefetch é oriundo do fato de que a “’[...] instrução que ocasiona a exceção já foi previamente carregada da memória (prefetch) e encontra-se no pipeline, aguardando o instante da sua execução”. (PEREIRA, 2007, p. 77) Exatamente neste momento, quando a instrução alcançar a etapa de execução do pipeline, é que ocorre a exceção.

Ocorrendo a exceção, o endereço da instrução seguinte a que originou a exceção é salvo no registrador LR e o conteúdo do CPSR é guardado em SPSR_abt.

Os bits de modo de processamento do CPSR são setados para o modo abort (Quadro 2), o estado ARM é selecionado (bit T do CPSR igual a 0), e as instruções IRQ desabilitadas (bit I do CPSR setado).

O registrador PC é carregado com o vetor de prefetch abort (0x0000000C), desviando o programa para este endereço, e esvaziando o pipeline.

A saída deste modo pode ser efetuada utilizando a instrução SUBS PC, R14,#4, "[...] que irá subtrair 4 do R14_abt e armazenar o resultado no PC, fazendo com que o programa retorne para a mesma instrução que causou a exceção de abort (supondo que a rotina de tratamento da exceção tenha solucionado o problema de acesso a memória)."(PEREIRA, 2007, p. 77)

Após a execução dessa instrução, o conteúdo do CPSR também é restaurado pelo valor salvo em SPSR_abt, fazendo com que a CPU retorne ao modo e estado anterior a exceção.

3.6.4.2 DATA ABORT

O data abort possui comportamento similar ao anterior, com a diferença de que este é "[...] provocado pela tentativa de escrever ou ler um dado de uma posição de memória não permitida ou não implementada."(PEREIRA, 2007, p. 77) E novamente a detecção da exceção ocorre durante a execução da instrução.

Depois de gerada a exceção de data abort, a CPU carrega o registrador LR (R14_abt) com o endereço da instrução abortada mais 8.

O conteúdo de CPSR é salvo em SPSR_abt e em seguida o bit de seleção de modo do CPSR é setado para o modo abort (Quadro 2), zera o bit T do CPSR para o modo ARM, e o bit I para desabilitar a interrupção IRQ. O registrador PC é carregado com o vetor data abort (endereço 0x00000010) e o programa é desviado para este endereço.

Os efeitos resultantes da execução parcial da instrução abortada variam conforme a implementação da arquitetura, no caso da família ARM7, implementava-se Base Updated Data Abort Model, enquanto na família ARM9 e em seus posteriores, implementa-se o Base Restored Abort Model, de acordo com ARM (2000b).

Segundo ARM (2000b), a diferença entre os modelos consiste em apenas numa pequena sessão de operação do código do sistema. Quando a exceção do data abort é gerado enquanto esta sendo

executado um acesso de memória à instrução, no modelo mais recente, o registrador base é sempre restaurado pelo processador para o valor que este registrador possuía antes da instrução ter sido executada, removendo assim a necessidade de a rotina de tratamento desse abort atualizar o registrador base presente no modo anterior.

Além disso, neste modelo, a maioria das instruções não produz efeito nenhum, mantendo os registradores inalterados, porém as instruções load/store múltiplo (LDM/STM) que causam aborts precisam de uma atenção especial.

No caso especifico citado acima, se a instrução fizer uso do modo write-back (alteração do registrador base), ele pode ser alterado caso um abort ocorra durante a operação. Após ocorrer à exceção, todas as demais transferências que deveriam ocorrer dentro da instrução são canceladas, evitando produzirem o mesmo efeito (mesmo que o registrador base esteja especificado dentre os que serão transferidos).

Um componente importante que é utilizado para verificar a ocorrência deste tipo de exceção é a MMU. Quando ocorre a tentativa da CPU em acessar o endereço de memória não implementado fisicamente ou não permitido, a MMU gera um sinal indicando o abort. Neste caso, a rotina de tratamento verifica o motivo da indisponibilidade do endereço especificado e trata-o corretamente.

Existem duas maneiras de realizar o retorno de uma exceção data abort, dependendo da ação de tratamento escolhida:

- no caso de não houver a intenção de uma nova tentativa de execução da instrução geradora da exceção, o retorno deverá ser realizado através da instrução SUBS PC, R14,#4 (subtrai 4 de R14_abt e salva-o em PC). Com isso, o PC apontará para a instrução posterior a geradora da exceção;
- no caso da existência de intenção de executar a instrução geradora da exceção novamente, o retorno deverá ser realizado através da instrução SUBS PC, R14,#8 (subtrai 8 do R14_abt e salva-o em PC). Com isso, o PC irá apontar para a instrução geradora da exceção novamente.

Após o retorno da exceção, o conteúdo de CPSR é restaurado pelo valor armazenado em SPSR_abt, fazendo com que a CPU retorne para o estado e modo anteriores.

Lembrando que, semelhante ao modo Supervisor, o modo Abort possui seus registradores R13_abt e R14_abt fisicamente independentes dos registradores R13 e R14 do modo User, além do seu próprio registrador SPSR, chamado de SPSR_abt.

3.6.5 MODO UNDEFINED (UND)

O modo Undefined também é um modo privilegiado que é "[...] invocado quando uma instrução não definida atinge a unidade de exceção (o final do pipeline)." (PEREIRA, 2007, p. 78)

Na ocorrência deste fato, a linha interna da CPU, a nCPI, é ativada (nível lógico 0), para o caso de existir um co-processador e possa reconhecer essa instrução. Existindo o co- processador e este reconhecendo a instrução, ele sinaliza para a CPU através da linha interna CPA. Com isso, a CPU irá aguardar a execução da instrução pelo co-processador, que sinaliza a finalização da execução através da linha interna CPB. Sendo assim, nenhuma exceção é gerada, apenas no caso de não existir o co-processador responsável pela execução desta instrução.

Uma solução interessante caso ocorra à exceção é a emulação do co-processador por software, ou seja, "[...] a CPU executa um código equivalente a operação que a instrução do co-processador deveria realizar" (PEREIRA, 2007, p. 79), permitindo assim que um programa que necessite de um co-processador matemático funcione sem a existência dele.

> Outra aplicação dessa exceção é a extensão do conjunto de instruções da CPU. É possível inserir *opcodes* não-implementados no programa, cuja tentativa de execução irá provocar essa exceção. A rotina de tratamento da exceção verifica o *opcode* que causou a exceção, e realiza a operação cabível. (PEREIRA, 2007, p. 79)

Quando o modo Undefined é acionado, o R14_und é carregado com o endereço da instrução seguinte a causadora da exceção, o registrador SPSR_und é carregado com o conteúdo do CPSR, e após seu conteúdo ser salvo, ele é modificado. Os bits de modo de processamento são alterados (vide Quadro 2), o bit T é igualado a 0 (estado ARM), bit I é setado (interrupções IRQ desabilitadas), e o PC é carregado com o endereço 0x00000004.

O retorno pode ser efetuado através da instrução MOVS PC, R14, ou seja, copiar o conteúdo de R14_und para o PC, além de copiar o conteúdo de copiar o conteúdo de SPSR_und para CPSR, forçando a CPU a retorna para o estado e modo de processamento anteriores a exceção.

Como pode ser verificado na Figura 7, o modo Undefined também possui seus próprios registradores R13 e R14, que são os R13_und e R14_und, respectivamente, além do SPSR_und, semelhante ao que ocorre aos modos Supervisor e Abort.

3.6.6 MODO IRQ (IRQ)

O modo IRQ é um modo de processamento privilegiado responsável pelo tratamento de interrupções, onde "[...] o tempo de atendimento da interrupção é da ordem de 416 ns rodando a 60 MHz." (SOUSA, 2006, p. 47). Este é acionado quando a linha de interrupção é posta em 0 e o bit I do CPSR encontra-se em nível 0 (interrupção IRQ habilitada).

Ocorrida à exceção, a CPU finaliza a execução da instrução corrente e executa as seguintes etapas:

1. o endereço da instrução posterior mais 4 é salvo no registrador R14_irq;
2. CPSR é armazenado em SPSR_irq;
3. CPSR é alterado, setando os bits de seleção de modo para o modo IRQ (Quadro 2); selecionado o estado ARM, ou seja, bit T=0, e a interrupção IRQ é desabilitada, ou seja, bit I=1;
4. PC é carregado com o endereço 0x00000018, desviando o programa para este endereço.

Em seguida, a rotina de tratamento de interrupção (RTI) deve verificar o periférico externo que originou a interrupção e tomar as decisões necessárias.

O retorno da exceção pode ser realizado através da execução da instrução SUBS PC, R14, #4, que subtrai 4 do conteúdo do R14_irq e armazena o resultado no PC, além de copiar o conteúdo de SPSR_irq para o CPSR, fazendo com que a CPU retorne para o modo e estado anteriores.

Similar aos modos privilegiados anteriores, ele possui o registradores R13 e R14 particulares, os R13_irq e R14_irq, respectivamente, além do seu próprio SPSR, o SPSR_irq.

3.6.7 MODO FIQ (FIQ)

O modo FIQ é o outro modo de processamento privilegiado gerado por interrupção, com "[...] o tempo de atendimento da interrupção [...] da ordem de 200 ns rodando a 60 MHz." (SOUSA, 2006, p. 45). E de acordo com PEREIRA (2007), este é similar ao anterior, porém com algumas diferenças importantes, como:

- maior prioridade: a interrupção FIQ possui maior prioridade em relação à interrupção IRQ. Desta forma, se duas interrupções (uma FIQ e outra IRQ) acontecerem simultaneamente, a FIQ será tratada primeiramente;
- conjunto de registradores especiais: o modo FIQ dispõe de um conjunto de registradores R8 a R12 diferenciado do modo IRQ e dos demais modos de operação da CPU. Esses registradores

diferenciados permitem que a rotina de tratamento da interrupção FIQ responda muito mais rapidamente ao evento de interrupção, já que esses registradores podem ser empregados localmente sem a necessidade de preservação do seu conteúdo.

Depois de detectada a interrupção, a CPU executa a instrução corrente e realiza o seguinte:

1. o registrador R14_fiq armazena o endereço da próxima instrução adicionado de 4;
2. o CPSR é salvo no registrador SPSR_fiq;
3. o registrador CPSR é alterado. O modo FIQ é setado nos bits de modo (Quadro 2), o estado ARM é selecionado (bit T=0), e as interrupções IRQ e FIQ são desabilitadas (bit I=1 e bit F=1);
4. o PC é carregado com o endereço 0x0000001C, para onde o programa é desviado.

Semelhante ao modo IRQ, a RTI verifica o periférico que originou a interrupção e toma as decisões necessárias. Como a intenção do FIQ é ser uma interrupção mais rápida, normalmente é atribuído apenas um periférico ao FIQ, dispensando a verificação do causador, acelerando o processo.

O retorno pode ser realizado através da realização da instrução SUBS PC, R14,#4, igualmente ao retorno do modo IRQ.

3.7 EXCEÇÕES, INTERRUPÇÕES E TABELA DE VETORES

Quando uma exceção ou interrupção é gerada por diversos motivos como o surgimento de uma instrução desconhecida pela CPU, ou por uma simples interrupção externa acionado por um periférico, o processador armazena no registrador PC o endereço específico de acordo com o tipo de exceção/interrupção, deslocando assim o programa para esta área onde executa a rotina de tratamento.

O range de endereços específicos para tais rotinas de tratamento é chamado de tabela de vetores. A tabela de vetores possui os seguintes vetores, de acordo com as exceções/interrupções:

- Vetor Reset – é a localização da primeira instrução a ser executada pelo processador quando este é ligado;
- Vetor Undefined Instruction – é usado quando a CPU não pode decodificar a instrução a ser executada;

- Vetor Software Interrupt – é utilizado quando a CPU executa uma instrução SWI. Esse tipo de instrução é freqüentemente utilizado como mecanismo de chamada para rotinas de sistemas operacionais;
- Vetor Prefetch Abort - ocorre quando se tenta executar uma instrução buscada em uma posição de memória inválida ou não permitida;
- Vetor Data Abort- é semelhante ao anterior, porém com a diferença de que este é provocado pela tentativa de escrever ou ler um dado de uma posição de memória não permitida ou não implementada;
- Vetor IRQ – é utilizado por um periférico externo para interromper a execução normal do código. Só pode ser usado caso o registrador CPSR seja devidamente configurado (bit I=0);
- Vetor FIQ – é semelhante ao anterior, porém é utilizado quando o periférico necessita de uma rotina de interrupção de resposta mais rápida. Novamente, este só pode ser utilizado com a correta configuração do registrador CPSR (bit F=0).

Exceção/Interrupção	Abreviação	Endereço	Último Endereço
Reset	RESET	0x00000000	0xFFFF0000
Undefined Instruction	UNDEF	0x00000004	0xFFFF0004
Software Interrupt	SWI	0x00000008	0xFFFF0008
Prefetch Abort	PABT	0x0000000C	0xFFFF000C
Data Abort	DABT	0x00000010	0xFFFF0010
Reservado	-	0x00000014	0xFFFF0014
Interrupt Request	IRQ	0x00000018	0xFFFF0018
Fast Interrupt Request	FIQ	0x0000001C	0xFFFF001C

Quadro 3 – Tabela de Vetores de Exceções/Interrupções. Fonte: ARM, 2000a.

Prioridade	Exceção/Interrupção
1	*Reset*
2	*Data Abort*
3	FIQ
4	IRQ
5	*Prefetch Abort*
6	*Undefined Instruction*/SWI

Quadro 4 – Prioridades das Exceções/Interrupções. Fonte: Pereira, 2007.

No Quadro 3 pode ser visualizado os respectivos endereços iniciais e finais de cada exceção/interrupção, enquanto que no Quadro 4 pode-se observar a existência de diferentes níveis de prioridade, de acordo com a natureza da exceção/prioridade, onde o de maior prioridade é o reset e o de última prioridade é o Undefined Instruction e o SWI.

4 MEMÓRIA E ARQUITETURA DE SISTEMAS

4.1 ORGANIZAÇÃO DA MEMÓRIA CACHE

Como foi descrito anteriormente no início do capítulo 3, a arquitetura ARM possui dois tipos diferentes de arquiteturas de acesso a memória, a Von Neumann, utilizada até o ARM7, e a arquitetura Harvard, utilizada nos ARM9 e posteriores. Essa diferença reflete diretamente no projeto da cache, para que suporte ambas as arquiteturas.

“No núcleo do processador usando a arquitetura Von Neumann, esta uma cache única utilizada para instruções e para dados. Esse tipo de cache é conhecido como unified cache. A memória unified cache possui tanto instruções quanto valores de dados.” (SLOSS et al., 2004, p. 408)

E diferentemente do projeto anterior, de acordo com Sloss, Symes e Wright (2004), a cache do tipo split cache, é dividida em I-cache (Instruction Cache) e a D-cache (Data Cache), e é utilizada nos processadores com a arquitetura Harvard.

4.1.1 I-CACHE E D-CACHE

Como descrito logo acima, numa split cache, as instruções são armazenas separadamente dos dados, na I-cache, e no caso dos processadores ARM que possuem a I- cache, esta pode ser devidamente alocada, escolhendo assim um dos tamanhos de caches suportados, que podem variar de 0KB a 1MB.

Na arquitetura ARM, é possível habilitar/desabilitar a I-cache, o que pode ser feito através do bit 12 do registrador de controle CP15, onde o valor 1 habilitará a I-cache, caso a protection unit já estiver habilitada ou ser habilitada simultaneamente, e 0 desabilitará o mesmo. Porém, uma importante observação com relação à habilitação/desabilitação da I-

cache é a seguinte:

“Você deverá desabilitar a cache se sua implementação é configurada com a I-cache de 0KB de tamanho. Habilitando a instruction cache quando não houver cache presente poderá resultar em um comportamento imprevisível.” (ARM, 2003, p. 3-6)

Igualmente a I-cache, o D-cache, possui a possibilidade de configurar o seu tamanho de acordo com os diversos tamanhos de caches suportados, além também da possibilidade de habilitação/desabilitação através do bit 2 do CP15.

E similar ao I-cache, ele também deverá ser apenas habilitado se a protection unit já estiver habilitada ou se esta for habilitada simultaneamente, e o mesmo comportamento será verificado no caso de habilitação da D-cache com seu tamanho configurado com 0 KB.

4.1.2 SET-ASSOCIATIVITY

Ao longo dos anos, houve uma procura intensa sobre qual seria a política que tivesse o melhor desempenho para as caches, a chamada procura pela cache “perfeita”, de acordo com Furber (2000).

Seguindo essa tendência da indústria de processadores, a ARM optou pela utilização da cache do tipo set associative, onde a cache possui diferentes blocos alocados separadamente, dizendo-se que ele possui N-ways, onde N é o número de subconjuntos da cache.

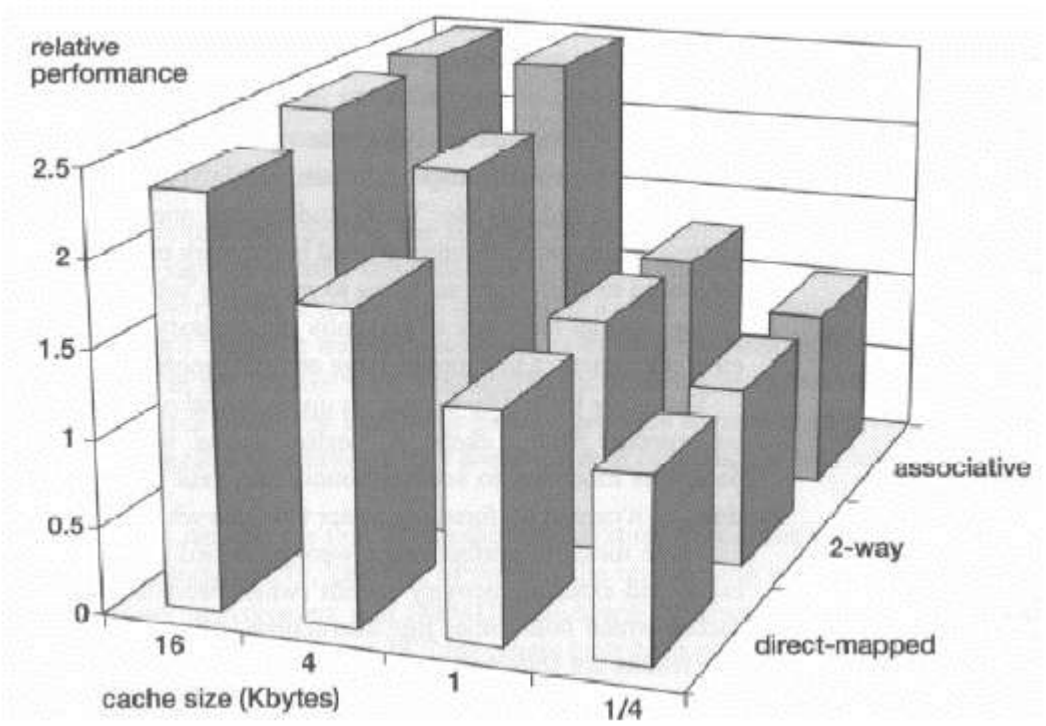

Figura 8 – Desempenho das diferentes políticas de *cache*. Fonte: Furber, 2000.

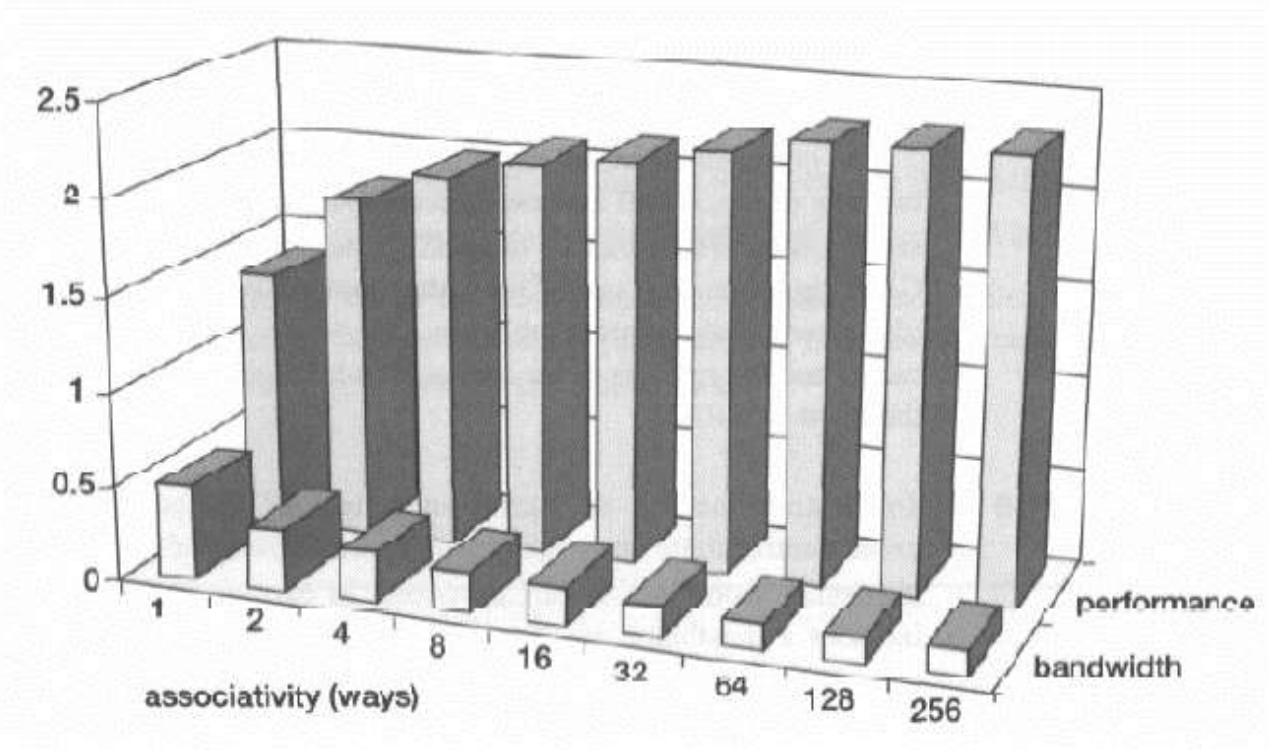

Figura 9 – Desempenho de acordo com a quantidade ways.

Fonte: Furber, 2000.

4.1.3 TAMANHO DA CACHE

> Geralmente, quando o tamanho da cache aumenta, uma grande percentagem de acesso a memória são os cache hits. Isto reduz o tempo médio por acesso de memória e aumenta o desempenho. Entretanto, uma grande cache tipicamente utiliza uma significante quantidade de área de silício e de energia. (ARM, 2005b, p. B6-4)

Logo, devido a esse fato e seguindo com seu objetivo de aumentar ao máximo o desempenho de seus processadores e reduzir o consumo de energia, a ARM tornou possível a configuração do tamanho da cache para atender a "[...] importância relativa do desempenho, da área de silício e do consumo de energia." (ARM, 2005b, p. B6-4)

Segundo ARM (2005b), a cache pode ser quebrada através do produto de três fatores:

- o tamanho da linha da cache LINELEN, mensurados em bytes;
- a set-associativity ASSOCIATIVITY. A cache consiste de ASSOCIATIVITY linhas de cache, logo o tamanho da cache é ASSOCIATIVITY x LINELEN;
- o número NSETS de set caches que pertencem a cache.

Uma observação importante é no caso do uso de I-cache e D-cache, diferentes valores para seus parâmetros podem ser escolhidos, resultando que seus tamanhos podem ser diferentes.

No Quadro 5, podem-se observar as diferentes caches para os diferentes chips da arquitetura ARM, onde se pode visualizar a mudança de suas configurações ao longo das gerações.

Núcleo	Tipo de *cache*	Tamanho da *cache* (Kbytes)	Tamanho da linha de *cache* (*words*)	*Associativity*	Localização	Suporte a *cache lockdown*	Tamanho do *write buffer* (*words*)
ARM720T	unified	8	4	4 - way	Lógico	Não	8
ARM740T	unified	4 ou 8	4	4 - way		Sim 1/4	8
ARM920T	split	16/16 D + I	8	64 - way	Lógico	Sim 1/64	16
ARM922T	split	8/8 D + I	8	64 - way	Lógico	Sim 1/64	16
ARM940T	split	4/4 D + I	4	64 - way		Sim 1/64	8
ARM96EJ-S	split	4 - 128/4 - 128 D + I	8	4 - way	Lógico	Sim 1/4	16
ARM946E-S	split	4 - 128/4 - 128 D + I	4	4 - way		Sim 1/4	4
ARM1022E	split	16/16 D + I	8	64 - way	Lógico	Sim 1/64	16
ARM1026EJ-S	split	4 - 128/4 - 128 D + I	8	4 - way	Lógico	Sim 1/4	8
Intel StrongARM	split	16/16 D + I	4	32 - way	Lógico	Não	32
Intel Xscale	split	32/32 D + I	8	32 - way	Lógico	Sim 1/32	32

Quadro 5 – Cache dos núcleos ARM. Fonte: Sloss et al., 2004.

4.1.4 CACHE LOCKDOWN

Um problema comum com caches é que quando eles normalmente elevam a média de acessos para dado e para instruções, eles usualmente fazem com que o pior caso de tempo de acesso piore. Como umas das principais aplicações do processador ARM são as aplicações de tempo real, a solução deste problema torna-se importante, já que "[...] o incremento do pior caso de tempo de acesso pode ser muito significativo." (ARM, 2000a, p. B5-18)

Uns dos diversos motivos para a ocorrência deste problema, afirmados por ARM (2000a), são:

- existe um delay antes que o sistema detecte que ocorreu cache miss e comece o acesso a memória principal;
- se o write-back cache esta sendo utilizado, poderá prolongar-se um delay até que seja necessário o armazenamento do conteúdo da linha de cache que esta sendo realocado;
- quando uma parte da linha de cache é carregada da memória principal, não somente o dado requisitado pelo processador ARM.

Para resolver esse problema, alguns sistemas de memória da arquitetura ARM possuem o cache lockdown. "Este permite que códigos críticos e dados (como rotinas de interrupções de alta prioridade e os dados que estes acessam) sejam carregados para a cache de modo que as linhas de cache que possuem tais não sejam subseqüentemente realocadas." (ARM, 2000a, p. B5-18)

> O propósito da informação *lockdown* na cache é de evitar a penalidade por *cache miss*. Entretanto, visto que qualquer memória *cache* utilizada por *lockdown é* indisponível para armazenamento de qualquer outra parte da memória principal, o tamanho da *cache* disponível torna-se reduzido. (SLOSS et al., 2004, p. 443)

O processador ARM aloca unidades fixas da cache para o lockdown. O tamanho dessas unidades que o ARM aloca é um way. Por exemplo, no caso do ARM9 que possui "[...] uma cache four-way set associative, é permitido que o código ou os dados armazenados na área de lockdown sejam 1/4 do tamanho da cache." (SLOSS et al., 2004, p. 443)

Porém, para ter um resultado significativo na utilização de cache lockdown, Sloss, Symes e Wright (2004) recomendam que sejam armazenados nas áreas da cache com lockdown a tabela de vetores de interrupções, ou o serviço de rotinas de interrupções ou algum código crítico que o sistema use extensivamente, além de, se possível, o armazenamento de variáveis globais.

Além disso, uma observação importante com relação ao cache lockdown para o bom desempenho deste, é que apesar dos dados e códigos salvos com ele serem imunes ao remanejamento, quando ocorre flush na cache, toda e qualquer informação nesses espaços também são perdidos. Como estas áreas são indisponíveis como memórias caches normais, tais informações perdidas devem ser devidamente restauradas com uma rotina de lockdown, de acordo com Sloss, Symes e Wright (2004).

E apesar do princípio do funcionamento do cache lockdown ser apenas um, existem certa variedade de maneiras alternativas de se conseguir implementá-lo. Umas das novas maneiras de implementar o cache lockdown no ARM9 e em seus posteriores são através do incremento do way index e através do uso dos lock bits, que serão detalhados a seguir.

4.1.4.1 CACHE LOCKDOWN POR INCREMENTO DO WAY INDEX

Sloss, Symes e Wright (2004) descrevem a implementação da cache lockdown por incremento do way index como basicamente a alteração da contagem dos valores de ways disponíveis na cache, alterando-se o seu valor inicial ou valor de reset deste.

Primeiramente, antes de qualquer procedimento de lockdown, é necessário alterar-se os registradores 9 e 0 do CP15, escolhendo qual linha de cache de qual way das duas caches será alvo do lockdown. A seguir é necessário escrever no registrador 7 do CP15, setando o victim reset value, que é o valor que o contador de ways é “[...] resetado quando este incrementa além do número de ways do núcleo.” (SLOSS et al., 2004, p. 445)

No caso de leitura e escrita deste registrador, faz se necessário a utilização das instruções MRC e MCR, respectivamente. Porém, para isso é necessário a utilização correta do formato do registrador Rd, que varia rapidamente de processador para processador, como pode ser visto na Figura 10.

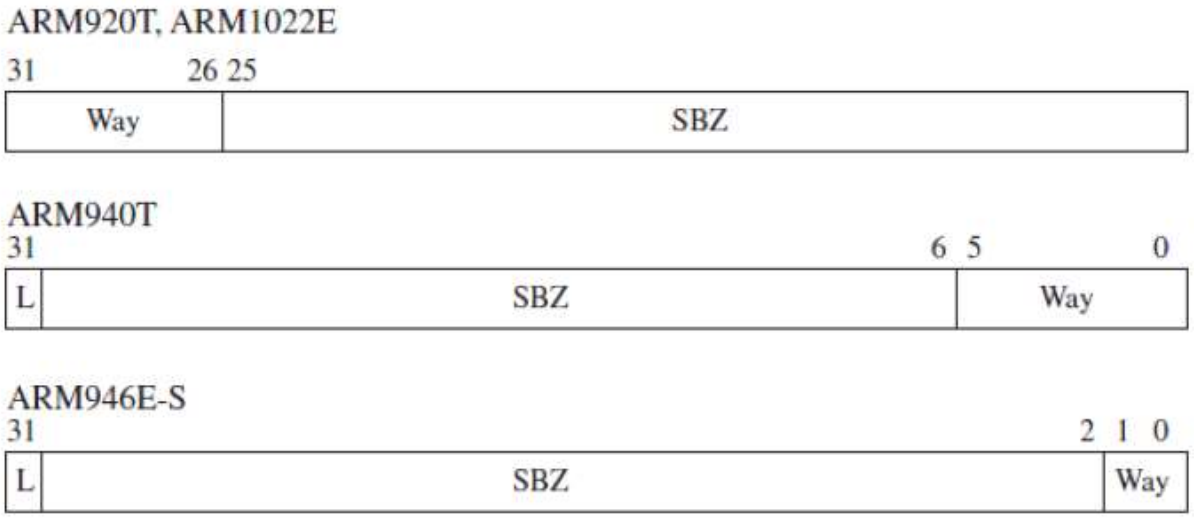

Figura 10 – Formato do registrador Rd nos diversos processadores. Fonte: Sloss et al, 2004.

Também se faz necessário nesse caso um comando especial de load para ler quando necessário as instruções para área de lockdown. Este comando deverá copiar por blocos do tamanho da linha de cache da memória principal para a I-cache, lembrando que este comando também necessita da formatação correta do Rd, de acordo com Sloss, Symes e Wright (2004).

4.1.4.2 CACHE LOCKDOWN UTILIZANDO OS LOCK BITS

Como foi dito no início deste tópico, no ARM9 e em seus posteriores, há também a possibilidade de realizar o cache lockdown através da utilização dos lock bits, porém existe diferenças de implementação deste entre a I-cache e a D-cache, como será visto a seguir.

Primeiramente, segundo ARM (2003), é necessário realizar os seguintes passos em ambos as caches:

1. colocar a cache em modo lockdown através da configuração do registrador 9;
2. forçar um preenchimento de linha;
3. travar o dado correspondente na cache.
4. Em seguida, no caso de utilização do lockdown em uma D-cache, segundo ARM (2003), os passos a seguir são necessários:
5. escrever no registrador 9 no CP15, modificando o valor de DL para 1 (DL é o bit 31, chamado load bit) e Dindex para 0 (Dindex são os bits 0 e 1, chamados de bits de segmentos de cache);
6. inicializar o ponteiro para a primeira word a ser travada na cache;

7. executar a LDR desta localização. Isto força o preenchimento da linha e resulta que oito words são capturadas na cache;
8. incrementar o ponteiro por 32 (número de bytes numa linha de cache);
9. executar a LDR desta localização. O resultado será que a linha preenchida será capturada na cache;
10. repetem-se os dois passos anteriores até que todas as palavras sejam colocadas na
11. cache ou ¼ da cache já estiver sido carregada;
12. escreva no registrador 9 no CP15, modificando o valor de DL para 0 e Dindex para 1.
13. Já com relação ao I-cache, segundo ARM (2003), os seguintes passos a seguir deverão ser efetuados:

4. escrever no registrador 9 no CP15, modificando o valor de IL para 1 (DL é o bit 31, chamado load bit) e Iindex para 0 (Dindex são os bits 0 e 1, chamados de bits de segmentos de cache);
5. inicializar o ponteiro para a primeira word a ser travada na cache;
6. forçar o preenchimento da linha desta localização escrevendo o registrador 7 do CP15 (instrução da cache pré carregada);
7. incrementar o ponteiro por 32 (número de bytes numa linha de cache);
8. forçar o preenchimento da linha desta localização escrevendo o registrador 7 do CP15. O resultado será que a linha preenchida será capturada na cache;
9. repetem-se os dois passos anteriores até que todas as palavras sejam colocadas na

cache ou ¼ da cache já estiver sido carregada;

10. escreva no registrador 9 no CP15, modificando o valor de IL para 0 e Iindex para 1.

Lembrando que a limitação de ¼ da cache dito acima é referente ao ARM9 que possui four-ways, ou seja, ¼ da cache significa um way para este. O real limite do cache lockdown é de um way por utilização, permitindo então neste caso a sua implementação por mais três vezes, já que o mínimo é de um way para utilização normal da cache, de acordo com Sloss, Symes e Wright (2004).

4.2 VIRTUAL MEMORY SYSTEM ARCHITECTURE (VMSA)

Com o intuito de tornar os processadores ARM capazes de suportar sistemas operacionais complexos, que "[...] tipicamente utilizam o sistema virtual de memória para prover a separação e a proteção dos espaços de memória por diferentes processos [...]" (ARM, 2005b, p. 4-2), a ARM criou a VMSA, baseada principalmente no MMU.

Outra observação sobre a importância da implementação do VMSA na arquitetura ARM é feito por Pereira (2007). De acordo com ele, a criação desse sistema de memória virtual tornou possível a implementação de sistemas operacionais que suportam melhor a multi-tasking, ou seja, a criação de threads, além do já citado proteção de memória, fundamental para impedir falhas onde um processo interfere no outro ou mesmo no kernel.

No VMSA, segundo ARM (2005b), os processos dinamicamente alocados na memória e em outras fontes de sistemas de memória mapeadas tornam-se sobre o controle da MMU. Tal controle pela MMU é de granularidade fina, o que é possível graças à transformação dos endereços de memória virtuais em físicos, mapeando e associando as propriedades de memória armazenadas em uma ou mais estruturas chamadas Translation Lookaside Buffers (TLBs).

Por sua vez, ARM (2005b) afirma que os conteúdos das TLBs são gerenciados através das pesquisas de traduções do hardware a partir de um conjunto de tabelas de conversão mantida na memória principal. E o processo de pesquisa completa na TLB é chamado de translation table walk, que é realizado automaticamente pelo hardware, e possui um significante tempo de execução devido a um, e muitas vezes dois, acessos a memória principal.

Apesar do tempo significativo, as TLBs conseguem reduzir o custo médio de acessos a memória principal pelo armazenamento de translation table walks na cache, onde tanto a arquitetura Von Neumann, utilizando-se apenas uma única TLB, quanto à arquitetura Harvard, utilizando-se TLBs de instruções e de dados separadamente, são suportados.

Segundo ARM (2005b), no VMSA é permitido um alto nível de controle através do uso dos registradores do CP15, que provê o controle da locação das TLBs e o fornecimento da informação de status sobre os aborts de memória para os processadores.

Uma grande vantagem da utilização da VMSA em sistemas de tempo real é a possibilidade do travamento das entradas na TLB, que com isso garante que estes acessos associados a áreas de memória não sejam requeridos a memória principal pela translation table walk, permitindo que o pior

caso de tempo de acesso para códigos e dados para rotinas de tempo real seja minimizado e determinístico.

E novamente, existe certas modificações significantes com relação ao VMSA, que originalmente foi implementado no ARM720T para os implementados na família ARM9 e em seus posteriores. Uma dessas modificações é que esta nova versão prevê a necessidade de invalidações da TLB na mudança de contexto, onde cada mapeamento de endereço virtual para endereço físico pode ser marcado como associado a um específico espaço de aplicação, ou como global para todos os espaços de aplicações.

Segundo ARM (2005b), somente os mapeados globalmente e seus endereços de aplicações correntes podem ser habilitados a qualquer momento. E pela modificação do chamado Application Space IDentifier (ASID), permiti-se a alteração do conjunto de mapeamentos de endereços virtuais para físicos.

Outra modificação foi à incorporação de definições de diferentes tipos de memória e de outros atributos, onde para compatibilidade com versões anteriores, está no bit de controle XP do registrador 1 do CP15.

São incluídas entre o conjunto de propriedades de memória que podem ser associados com cada TLB, de acordo com ARM (2005b), a seguir:

- permissão de controle de acesso a memória: este controla se o programa possui nenhuma permissão de acesso, ou acesso apenas para leitura, ou acesso para leitura e escrita por área de memória. Quando o acesso não é permitido, um abort de memória é sinalizado ao processador. O nível de acesso permitido pode ser afetado caso o programa esteja sendo executado no modo User, ou em um modo privilegiado, e através de usos de domínios;
- atributos de regiões de memória: Este descreve as propriedades da região da memória, por exemplo, non-cacheable, write-through, e write-back;
- mapeamento de endereços virtuais para físicos: Um endereço gerado pelo processador ARM é chamado de endereço virtual. A MMU permite que estes endereços possam ser mapeados para diferentes endereços físicos. O endereço físico identifica que local da memória principal esta sendo acessado. Isto pode ser utilizado para gerenciar a alocação de memória física por diferentes maneiras. Por exemplo, isto pode ser usado para alocar memória para diferentes processos com potencias conflitos de mapas de endereços, ou permitir que a aplicação com enorme mapa de endereços utilize uma continua região de memória física.

Além dessas duas modificações descritas anteriormente em relação ao VMSA, existem outras, como por exemplo, a existência de atributo de regiões de memória para marcar páginas que possam ser compartilhadas por múltiplos processadores e o desuso das páginas pequenas e do formato de segundo nível da tabela de páginas, os tornando obsoletos.

Em seguida é descrito o papel da MMU no VMSA, dando maior importância na sua habilitação, desabilitação, a seqüência de acesso a memória e a configuração das permissões de acesso e de domínios.

4.2.1 MEMORY MANAGEMENT UNIT (MMU)

A MMU, como já foi dito, é peça fundamental, responsável pela implementação da VMSA na arquitetura ARM. O seu papel na VMSA é observado durante a seqüência de acesso a memória, que segundo ARM (2005b), quando a CPU gera o acesso a memória, então a MMU realiza uma pesquisa para o mapeamento do endereço virtual requisitado na TLB, lembrando que tal implementação pode ser feita tanto na arquitetura Harvard, quanto na Von Neumann.

No caso da entrada da TLB for achada, a informação encontrada é utilizada, de acordo com ARM (2005b), da seguinte maneira:

1. o bit de permissão de acesso e o domínio são usados para determinar se o acesso é permitido ou não. Se o acesso não for permitido pela MMU, um sinal de abort será sinalizado por este. Senão, no caso de ser permitido, o acesso segue os demais passos a seguir;
2. os atributos da região de memória são usados para controlar:
    a) a cache e o write buffer;
    b) se o acesso é cacheable ou não;
    c) o tipo de memória de destino;
    d) se o tipo de memória destino é dividido ou não;
3. o endereço físico é utilizado para qualquer acesso externo ou Tightly Coupled Memory (TCM).

Entretanto, a MMU é não é um recurso obrigatório na arquitetura ARM, sendo possível habilitar e desabilitar este através do bit M (bit 0) do registrador 1 do CP15, ressaltando que quando ocorre o reset do processador, automaticamente este bit é zerado, desabilitando a MMU.

Estando a MMU desabilitada, de acordo com ARM (2005b), o acesso a memória é tratada da seguinte maneira:

- todos os acessos a dados são tratados como non-cacheable e fortemente ordenado. O comportamento de uma inesperada data cache hit é definido por implementação;
- se a cache for do tipo de Harvard, é utilizado como que todos acessos a instruções sejam cacheable, não divididos e como memória normal, se o bit I (bit 12) do registrador 1 do CP15 estiver setado e como non-cacheable, não divididos e como memória normal se o bit I estiver zerado. Os outros atributos relativos à memória cache são definidos por implementação;
- se a cache for do tipo Von Neumann, todos os acessos são tratados como não divididos, normais e non-cacheable;
- todos os acessos explícitos são fortemente ordenados e o valor do bit W (bit 3, write buffer) do registrador 1 do CP15 é ignorado;
- nenhuma permissão de acesso de memória é verificada e nenhuma abort é gerada pela MMU;
- os endereços físicos para todos os acessos são iguais aos seus endereços virtuais.

Lembrando que de acordo com ARM (2005b), antes de habilitar o MMU, faz se necessário programar todos os registradores relevantes do CP15, além de desabilitar e invalidar a I-cache, porém este pode ser novamente habilitado no mesmo momento que habilitar a MMU.

Com relação ao controle da memória de acesso, este é controlado através dos bits de permissão e dos bits de domínios nas entradas da TLB. No caso primeiro caso, os bits de permissão de acesso controlam o acesso correspondente a região da memória, ou seja, no caso de o acesso esta sendo feito na área de memória sem permissão requerida, um Permission Fault é gerado, ademais a permissão de acesso é determinada pela combinação dos bits AP e APX nas TLBs, e pelos bits S e R do registrador 1 do CP15.

Já em relação ao segundo caso, é necessário o entendimento do conceito de domínio, que segundo ARM (2005b), é a coleção de regiões de memória, lembrando que a arquitetura ARM oferece suporte para 16 diferentes tipos de domínios. Com isso, cada entrada da TLB possui um campo que especifica que domínio ela esta situada.

O acesso a cada domínio é controlado por dois campos de bits do registrador Domain Access Control e como cada campo permite que o acesso ao domínio das entradas seja habilitado ou desabilitado muito rapidamente, faz com que esses espaços de memória possam ser trocados na memória virtual

muito eficientemente. Abaixo pode se verificar, segundo ARM (2005b), dois exemplos de domínios de acesso suportados pela arquitetura ARM:

- clients: o domínio dos usuários (executa programas e acessa dados), protegidos pela permissão de acesso da entrada da TLB para este domínio;
- managers: controla o comportamento do domínio (as secções correntes , as páginas no domínio e o domínio de acesso), e não são protegidos pela permissão de acesso das entradas de TLB para este domínio.

Logo, um mesmo "[...] programa pode ser client para alguns domínios, e manager para outros, e possuir nenhum acesso remanescente a domínio, permitindo uma grande flexibilidade na proteção de memória dos programas e acesso a diferentes tipos de memória." (ARM, 2005b, p. B4-10)

4.3 PROTECTED MEMORY SYSTEM ARCHITECTURE (PMSA)

A PMSA é baseada na MPU para provê uma considerável, porém mais simples esquema de memória protegida do que o fornecido pela MMU, em alguns processadores ARM7 e posteriores. "A simplificação aplica-se tanto no hardware quanto no software [...]" (ARM, 2005b, p. 5-2), como será verificador a seguir.

A principal simplificação deste com relação à MMU é que este não utiliza a tradução de tabelas. "Em vez disso, os registradores do CP15 são utilizados totalmente para definir as protection regions, eliminando a necessidade de o hardware traduzir as tabelas caminhos, e para o software a configuração e a manutenção da tradução de tabelas." (ARM, 2005b, p. 5-2)

Com esta simplificação, ARM (2005b) relata que traz o benefício de fazer a memória ser checada totalmente deterministicamente, porém, em contrapartida, faz com que o nível de controle ser consideravelmente de menor granularidade fina.

Outra simplificação é que a PMSA não suporta o mapeamento de endereços virtuais para físicos. O endereço de memória físico é sempre o mesmo do endereço da memória virtual gerada pelo processador. Além dessas duas características principais, a seguir são descritas outras comuns a todos os projetos de PMSA:

- a memória é dividida em regiões. Os registradores do CP15 são utilizados para definir estas regiões, o endereço base, e os atributos da memória, como por exemplo: a cacheability, bufferability e a permissão de acesso destas regiões;

- o controle da região de memória (acesso de leitura e escrita) é permitido somente pelos modos privilegiados;
- se um endereço for definido para múltiplas regiões, um esquema de prioridade fixa é usado para definir a propriedade dos endereços que estão sendo acessados;
- um acesso a um endereço que não esteja definido em qualquer região causará um
- abort de memória;
- todos os endereços são endereços físicos; endereços traduzidos não são suportados;
- suporta a cache do tipo unified (Von Neumann) e do tipo separate (Harvard), com espaços de endereços para instruções e para dados.

Lembrando que no caso dos alguns processadores ARM9 e posteriores, de acordo com o ARM (2005b), o PMSA foi revisado e modificado para melhor atendê-lo, sendo agora possível implementar mais do que oito regiões protegidas, como era anteriormente, sendo que o número de regiões suportadas e o método de acesso delas são associadas aos registradores do CP15, sendo similar ao esquema empregado pelo TCM.

Outra modificação é referente aos atributos e as permissões de acessos, que foram estendidas para suportar adicionalmente características dos processadores ARM9 e posteriores. Entre elas, inclui-se a expansão do acesso de memória, permitindo tanto a somente leitura para modos privilegiados, quanto à somente leitura dos modos privilegiados e usuário suportado simultaneamente.

Em relação aos mecanismos de abort, passou a ser usado o CP15, definindo os registradores Fault Status e Fault Address para reportar a origem do abort e o endereço deste. Essa modificação foi significativa, porque ARM (2005b) relata que anteriormente um abort no PMSA era considerado catastrófico, com nenhum mecanismo de recuperação arquitetado.

A seguir é descrito o papel da MPU no PMSA, dando um foque na sua habilitação, desabilitação, o acesso a memória protegida e a configuração das permissões de acesso.

4.3.1 MEMORY PROTECTION UNIT (MPU)

Segundo ARM (2005b), a principal responsabilidade da MPU na PMSA refere-se à comparação do endereço de memória com as regiões programadas de memória, quando o processador gera o acesso de memória.

Ocorrendo essa comparação, no caso da região de memória não ser achado, um sinal de abort de memória é enviada ao processador, no outro caso, a informação da região é usada da seguinte maneira:

> 1. os bits de permissão de acesso são utilizados para determinar quando o acesso é permitido. Se o acesso não for permitido, a MPU sinalizará um abort de memória. No outro caso, o acesso é permitido e procede como a seguir;
> 2. os atributos da região de memória são utilizados para determinar os atributos do acesso, por exemplo, cacheable ou non-cacheable. (ARM, 2005b, p. 5-4)

Porém, é possível a escolha da habitação e desabilitação da MPU, que ocorre através da escrita no bit M, vulgo bit 0, do registrador CP15. Lembrando que quando ocorre o reset, o bit M é zerado, desabilitando a MPU, forçando sua reabilitação por software, se necessário.

Antes de habilitar a MPU, todos os registradores relevantes do CP15 devem estar programados, incluindo a habilitação de ao menos uma região de memória. Outro requisito anterior a habilitação da MPU é que a I-cache deve esta desabilitada e invalidada, e a D-cache deve esta igualmente desabilitada e invalidada, além de limpa.

Com relação ao comportamento quando o MPU é desabilitado, houve também mudanças significativas, já que anteriormente, quando isto ocorria, todas as regiões de memória eram tratadas como memórias non-cacheable, não-bufferizáveis e não-abortivas.

ARM (2005b) afirma sobre outra mudança a partir do ARM9, a desabilitação da MPU, tornando o acesso à memória a ser tratada da seguinte maneira:

- nenhuma checagem de permissão de acesso de memória é realizada, e nenhuma abort é gerada por MPU;
- acessos de dados é o default do mapa de memória; Os acessos de dados menores do que 2GB são tratados como cacheables se o D-cache (ou unified) estiver habilitado pelo bit C (bit 2) do CP15. Os acessos de dados maior que 2GB são tratados como non-cacheables;
- se a cache for arranjada no modelo de Harvard esta sendo usada, todos os acessos a instruções menores de 2GB são tratados como normais, não-compartilhados e cacheables. Os acessos de instruções são tratados como cacheables se o bit (bit 12) do CP15 estiver setado. Os acessos a instruções maiores de 2GB são tratados como non- cacheable;

- se o unified cache é utilizado, todas as instruções menores do que 2GB são tratados como cacheables se o bit C (bit 2) do CP15 estiver setado. Os acessos maiores de 2GB são tratados como non-cacheables;
- a predicação do programa corrente funcionam normalmente, controlados pelo estado do bit Z (bit 11) do CP15 do registrador 1;
- todas as operações do CP15 em relação ao MPU e cache funcionam normalmente quando o MPU é desabilitado;
- os acessos ao TCM funcionam normalmente se o TCM estiver habilitado;
- a cache L2 possui os mesmo atributos de memória da cache L1.

Outra modificação com relação ao uso do MPU esta relacionada ao controle de acesso a memória, tendo sido modificada a programação dos bits de acesso de controle da MPU. Anteriormente, eram suportadas apenas oito regiões diferentes, que eram programados nos bits de um único registrador, o registrador AP, porém com a mudança da quantidade de regiões, a configuração foi alterada, no caso de regiões de dados, a alteração pode ser vista abaixo.

AP	Permissão dos modos privilegiados	Permissão do modo User
0b00	Sem permissão	Sem permissão
0b01	Leitura/Escrita	Sem permissão
0b10	Leitura/Escrita	Somente leitura
0b11	Leitura/Escrita	Leitura/Escrita

Quadro 6 – A configuração anterior do controle de permissões de acesso do MPU. Fonte: ARM, 2005b.

AP [2:0]	Permissão dos modos privilegiados	Permissão do modo *User*	Descrição
000	Sem permissão	Sem permissão	Todos os acessos geram falha de permissão
001	Leitura/Escrita	Sem permissão	Somente acesso privilegiado
010	Leitura/Escrita	Somente leitura	Escrita no modo *User* gera uma falha de permissão
011	Leitura/Escrita	Leitura/Escrita	Acesso total
100	Indefinido	Indefinido	Reservado
101	Somente leitura	Sem permissão	Somente leitura privilegiada
110	Somente leitura	Somente leitura	Somente leitura para ambos os modos
111	Indefinido	Indefinido	Reservado

Quadro 7 – A atual configuração do controle de permissões de acesso do MPU. Fonte: ARM, 2005b.

5 FUTURO DA ARQUITETURA ARM

Neste capítulo é mostrado às tendências futuras da arquitetura ARM, através de um overview da mais nova família, a família Cortex, descrevendo alguns dos novos recursos implementados, demonstrando assim, a preocupação constante da ARM em dar suporte as necessidades do mercado. Além desse overview, no final deste capítulo é demonstrada uma comparação da família Cortex com os processadores ARM7 e ARM9, mostrando suas diferenças e o quanto os processadores ARM evoluíram ao longo das gerações.

5.1 FAMÍLIA CORTEX

- A família Cortex utiliza a mais recente geração da arquitetura ARM, a sétima geração, permitindo atingir níveis de desempenho muito superiores as versões anteriores. Entre as principais características disponíveis na família Cortex, segundo Pereira (2007), destacam-se:
- a arquitetura superescalar (execução de duas instruções simultaneamente) com pipeline de até 13 estágios;
- a predicação dinâmica de desvios;
- memória cache L1 (primária) de 16 ou 32 KB e L2 (secundária) de 64 KB a 2 MB;
- novo conjunto de instruções, o Thumb-2, e seu derivado, o Thumb-2EE (Thumb-2
- Execution Environment);
- co-processador de mídia NEON com instruções SIMD (Single Instruction, Multiple Data) de 64/128 bits;
- sistema de segurança integrado a arquitetura, o TrustZone.

Com essas características, os níveis de desempenho podem atingir a marca de 2000 MIPS a 1GHz. Porém, devido à enorme variedade de desempenho e de implementação de recursos na família Cortex, esta acabou se subdividindo em três famílias, de acordo com seus desempenhos e aplicações alvos, que são:

- Cortex-A: direcionada para aplicações de alto desempenho baseadas em sistemas operacionais que utilizem sistemas de memória virtual. Sendo o Cortex-A, a subdivisão da família Cortex de melhor desempenho, implementando todas as características descritas acima, e de acordo com Pereira (2007), projetado para substituir futuramente o ARM11;
- Cortex-R: focada para aplicações de alto desempenho e tempo real, porém, com desempenho inferior ao Cortex-A, não implementando todas as características desse, e, segundo Pereira (2007), devendo ser o substituto do ARM9;
- Cortex-M: é a versão extremamente simples da família Cortex, e direcionada para as aplicações de baixo custo. Um exemplo dessa família é o Cortex-M3, que implementa algumas das características acima, e outras como um controlador de interrupções integrado que facilita a vetorização, priorização, tail chaining (que evita o desempilhamento/reempilhamento de registradores quando ocorre o tratamento de múltiplas interrupções simultaneamente).

Devido a essas características simples e pelo baixo custo, de acordo com Pereira (2007), a família Cortex-M deve no futuro substituir o ARM7.

5.2 NOVOS RECURSOS

A família Cortex de processadores não apenas se destaca pelo melhor desempenho de seus processadores devido a recursos com aumento da freqüência do clock ou maior quantidade de DMIPS/MHz, mas também por uma nova gama de recursos que incrementam ainda mais seu desempenho, além melhorarem outros aspectos como uso da memória, custo do projeto, tamanho do hardware e segurança.

E três recursos específicos, o Thumb-2 e sua derivação Thumb-2EE, o NEON e o TrustZone, provavelmente serviram de base para os novos processadores ARM devido ao seus incremento significativo no desempenho e nos outros aspectos, que são melhor detalhados a seguir.

5.2.1 THUMB-2

O Thumb-2 é um novo conjunto de instruções altamente eficiente e poderosa que oferece vantagens significantes em termos de facilidade de uso, densidade de código e desempenho. O conjunto de instruções Thumb-2 é um superconjunto que abrange tanto o conjunto de instruções Thumb de 16 bits, quanto um subconjunto de instruções ARM de 32 bits, além de uma nova variedade de instruções.

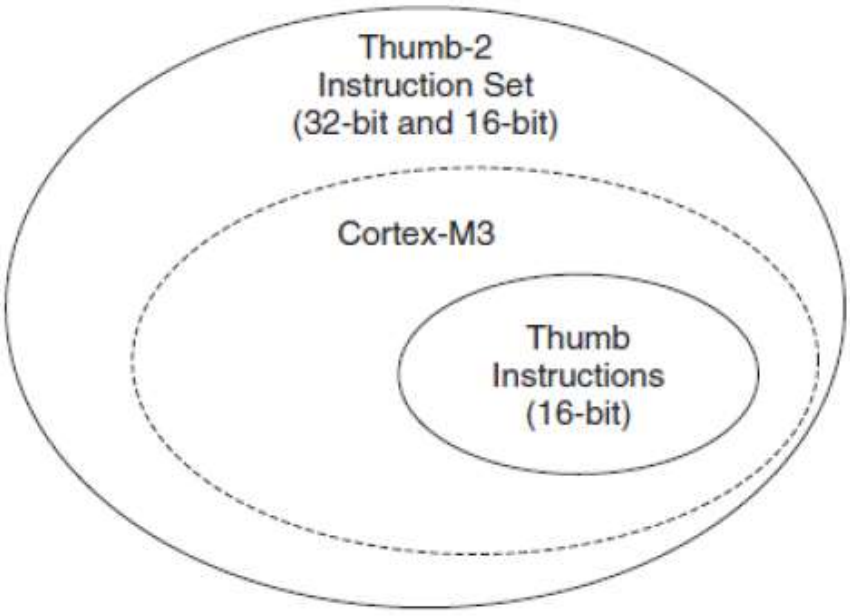

Figura 11 – Relação entre os conjuntos de instruções no Cortex-M3. Fonte: Yiu, 2007.

Essa característica de superconjunto do Thumb-2 proporciona uma maior eficiência e facilidade de uso, já que se faz desnecessário a troca constante entre o estado ARM e o estado Thumb, devido ou pela necessidade de uso das instruções complexas ARM, ou pela redução do tamanho do código.

Vale-se ressaltar, que devido o objetivo de minimizar ao máximo o tamanho do chip Cortex-M3, este resultou sendo um exemplo de processador ARM que não possui o conjunto de instruções ARM padrão, como pode ser visualizado na Figura 11, inviabilizando a compatibilidade de códigos escritos para o ARM7 ou outros processadores ARM que utilizem esse conjunto de instruções

Outra facilidade, segundo ARM (2006), é que as instruções Thumb-2 são acessíveis da mesma maneira que as instruções Thumb são acessadas, colocando-se o processador no estado Thumb, ou seja, fazendo-se o bit T do CPSR ser 1 e o bit J do mesmo ser 0.

Além da maior eficiência proporcionada e da facilidade de uso, houve outras melhorias, segundo ARM (2006), como por exemplo:

- adição de novas instruções Thumb de 16 bits para melhorar o fluxo do programa;
- adição de instruções de 32 bits ao conjunto de instruções Thumb para:

➢ fornecer suporte ao tratamento de exceções no estado Thumb;
➢ facilitar acesso aos co-processadores;
➢ incluir instruções de PDS e de mídia;
➢ melhorar o desempenho nos casos em que uma única instrução de 16 bits limita as funções disponíveis para o compilador;

- adição de uma instrução de IT (if-then), que permite ¼ das instruções Thumb serem condicionais;
- além de um CBZ (Compare with Zero and Branch) de 16 bits que permite aumentar a densidade de código, substituindo uma seqüência de duas instruções por apenas uma instrução.

As instruções ARM no Thumb-2 são adicionadas no espaço ocupado pelas instruções Thumb BL e BLX, resultando no seguinte formato:

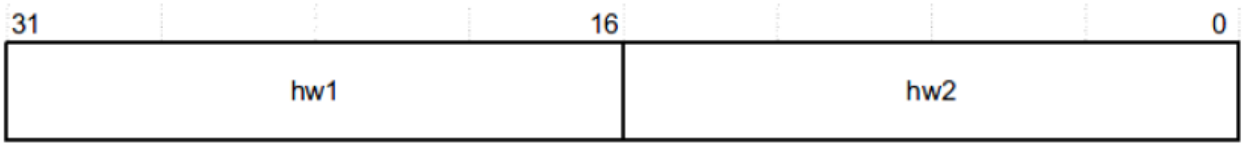

Figura 12 – Formato da instrução 32 bits ARM no Thumb-2. Fonte: ARM, 2006.

Na Figura 12, pode-se visualizar dois campos distintos de 16 bits, o hw1 (halfword 1), do bit 31 ao 16, e o hw2 (halfword 2), do 15 bit ao bit 0. O campo denominado como hw1 determina o comprimento da instrução e sua funcionalidade. O segundo campo só será verificado pelo processador no caso de se verificar que a instrução é de 32 bits, fazendo-se necessário a busca pelo halfword complementar da instrução.

E segundo Phelan (2003), todas essas melhorias fizeram com que a tecnologia de núcleo Thumb-2 pode-se proporcionar:

- desempenho em torno de 98% do código em C desenvolvido utilizando apenas o conjunto de instruções ARM;
- memória gerada com aplicações envolvendo checagem de digitais humanas com apenas 74% do tamanho da mesma gerada por implementação com instruções ARM equivalentes;
- possibilidade de ser até 35% menor que o código gerado em ARM.

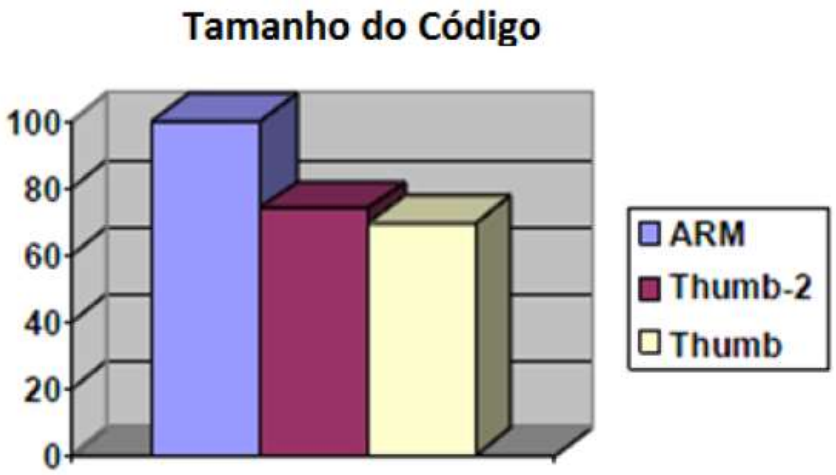

Figura 13 – Relação da densidade de código. Fonte: Phelan, 2003.

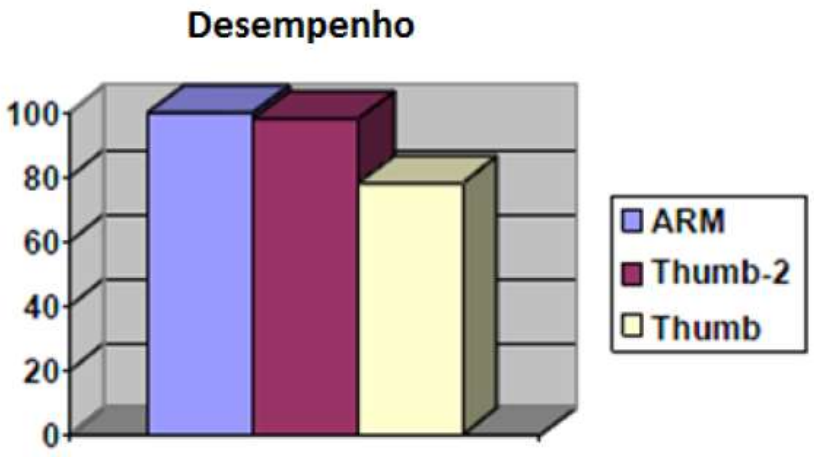

Figura 14 – Relação do desempenho. Fonte: Phelan, 2003.

Para uma melhor visualização do significado dos gráficos acima, é demonstrado um exemplo comparativo, que demonstra três pedaços de códigos similares, cada um em um estado diferente, demonstrando assim o desempenho e a densidade de código de cada estado.

Código em ARM	Código em *Thumb*	Código em *Thumb*-2
LDREQ r0, [r1]	*BNE L1*	*ITETE EQ*
LDRNE r0, [r2]	*LDR r0, [r1]*	*LDR r0, [r1]*
ADDEQ r0, r3, r0	*ADD r0, r3, r0*	*LDR r0, [r2]*
ADDNE r0, r4, r0	*B L2*	*ADD r0, r3, r0*
	L1	*ADD r0, r4, r0*
	LDR r0, [r2]	
	ADD r0, r4, r0	
	L2	

No exemplo acima, segundo Phelan (2003), em relação à densidade de código, o código gerado em ARM ocupa um tamanho de 16 bytes, enquanto o código em Thumb ocupa 12 bytes e o em Thumb-2 ocupa apenas 10 bytes. Já em relação ao desempenho, o código gerado em ARM será executado em 4 ciclos do clock, diferentemente do código em Thumb que poderá levar 4 à 20 ciclos para ser executado, e enquanto isso, o código em Thumb-2 levará 4 ou 5 ciclos do clock.

A grande variação da contagem dos ciclos de clock do código em Thumb depende da predicação exata dos desvios. No caso do Thumb-2, duas instruções podem ser substituídas por uma similar, reduzindo de 5 para 4 ciclos do clock.

Com isso, todas essas novas características implementadas pelo Thumb-2, em conjunto com o bom desempenho e a boa densidade de código, fará com que "[...] a tecnologia de núcleo Thumb-2 se torne o conjunto de instruções default para a maioria das aplicações [...]." (PHELAN, 2003, p. 4)

5.2.1.1 THUMB-2EE

Entretanto, com o objetivo de aumentar ainda mais o desempenho dos microcontroladores ARM, e de aumentar cada vez mais a densidade de código, a ARM desenvolveu uma variante da tecnologia Thumb-2, o Thumb-2EE, utilizado em chips da família Cortex de alto desempenho, como o Cortex-A8.

Segundo ARM (2006), o código em Thumb-2EE é compilado dinamicamente, ou seja, ele é compilado um pouco antes de sua execução ou até mesmo durante sua execução feita a partir um bytecode portátil, ou de outras linguagens intermediárias, como por exemplo: Java, C#, Perl e Python, ou da nativa.

Com essa característica, o Thumb-2EE consegue um aumento da densidade de código comparado ao código binário compilado pelo Thumb ou pelo Thumb-2, além de ter um desempenho melhor do que ambos, se igualando a perfomace do conjunto de instruções ARM, como podem ser visto na Figura 15 e 16.

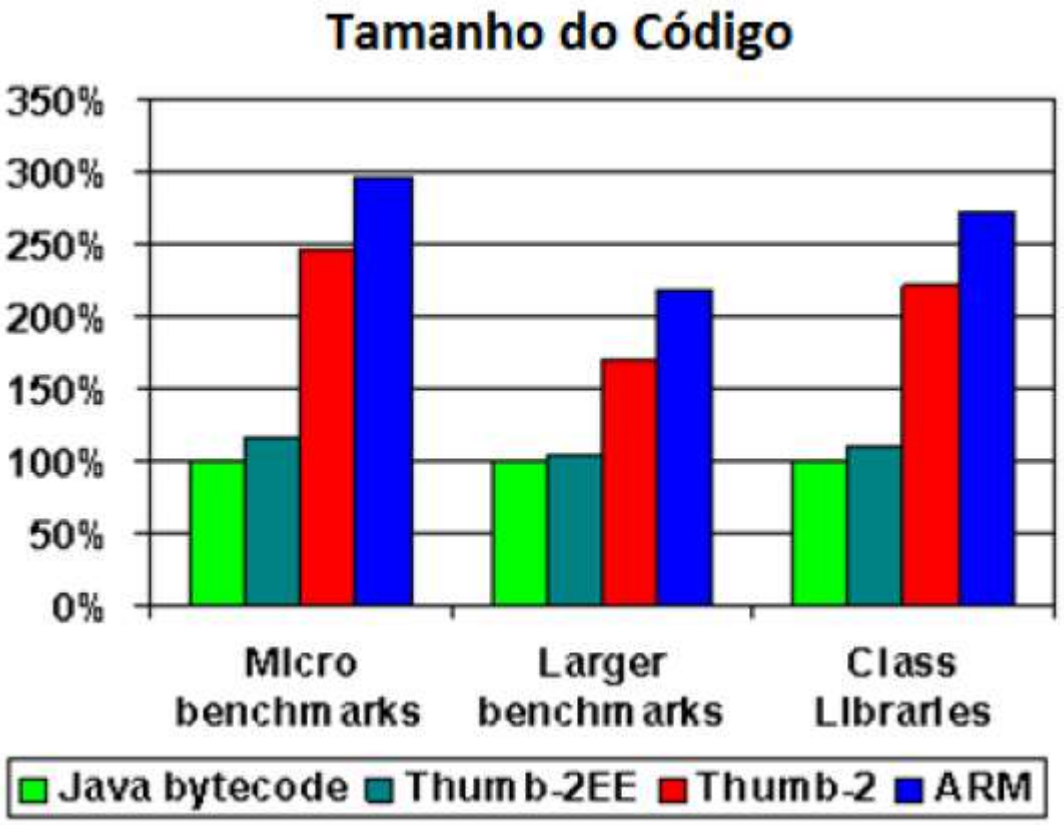

Figura 15 – Comparação da densidade de código do Thumb-2EE com os demais.

Fonte: Whealton, 2008.

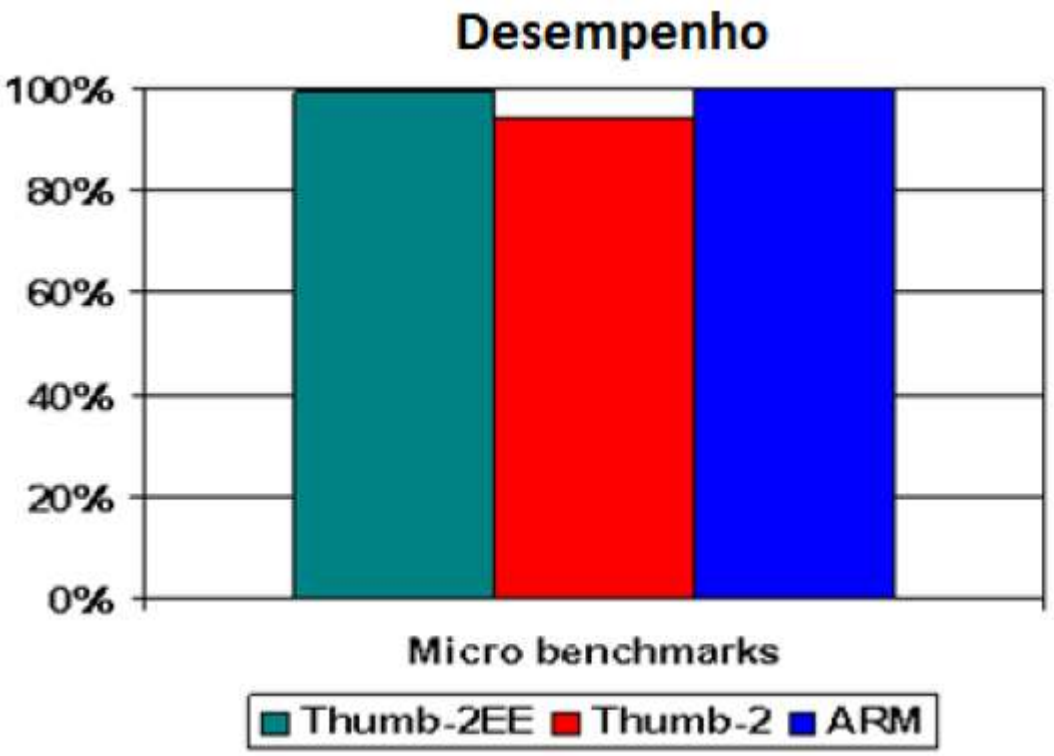

Figura 16 – Comparação do desempenho do *Thumb*-2EE com os demais.

Fonte: Whealton, 2008.

Ademais, o Thumb-2EE introduz um novo estado de processamento, o estado ThumbEE, que pode ser ativado fazendo-se o bits T e J do CPSR terem valor igual a 1. Habilitando esse novo estado, poder-se-á então usufruir das qualidades do Thumb-2EE, de acordo com ARM (2006), como também das novas instruções adicionadas às instruções do Thumb-2 e algumas modificações das instruções já existentes, como por exemplo:

- instrução adicional de mudança de estado, tanto no estado Thumb, quanto no estado ThumbEE;
- novas instruções de desvio;
- verificação de ponteiros nulos no load/store;
- adição de uma instrução suplementar de verificação de limites de matrizes;
- algumas modificações em instruções de load/store e desvio.

Essas novas instruções, em conjunto com o bom desempenho e densidade de código, fez com que o Thumb-2EE pudesse contribuir com que o Cortex-A8 e o Cortex-A9 sejam os chips ARM de melhor desempenho, facilitando seu uso em diversas aplicações como games, GPS (Global Positioning System), e outros dispositivos eletrônicos de alto desempenho, além de servir de referência para os novos chips ARMs que estão por vim.

5.2.2 NEON

No entanto, observou-se nos processadores ARM um baixo desempenho resultante do processamento de codecs de mídia ou de aceleradores gráficos, que utilizam uma grande quantidade de informação menor que o word (32 bits), como dados de 16 bits, comuns em aplicações de áudio, e de 8 bits, comuns em gráficos e vídeo, que entram em “conflito” com a natureza do processamento em 32 bits, assim, tornando ocioso parte do processador durante sua execução e utilizando mais energia com este.

Com a finalidade de resolver esse problema, a ARM tornou possível o uso da tecnologia SIMD através da expansão NEON nos seus processadores, sendo assim, segundo ARM (2009e), é possível processar múltiplos dados de tamanhos menores ao mesmo tempo, aumentando assim a eficiência do processador para essas aplicações.

As instruções do NEON são executadas como parte do conjunto de instruções Thumb ou ARM., simplificando o desenvolvimento, depuração e integração. Abaixo, segundo ARM (2009e), as funções que as instruções NEON podem desempenhar:

- acesso a memória;
- copiar os dados dos registradores de propósito geral para os registradores do NEON;
- converter os diferentes tipos de dados;
- processamento de dados.

Para dar suporte ao processamento SIMD, o NEON implementa um banco de 32 registradores exclusivos, que só podem ser acessados pelo Advanced SIMD ou pelo VFPv3, de 64 bits, podendo esses serem visualizados também como 16 registradores de 128 bits, a depender da necessidade de processamento da aplicação. Outra implementação importante que auxilia as funções das instruções do NEON é a existência de duas ULA de 64 bits em paralelo, possibilitando a manipulação 128 bits em um único ciclo do clock.

Assim, as instruções NEON suportam dados de 8, 16, 32, 64 bits signed e inteiros unsigned. Além disso, o NEON também pode suportar dados com 32 bits de ponto flutuante de precisão simples, e de 8 e 16 bits polinomiais, podendo também converter com a instrução VCVT elementos de ponto flutuante de precisão simples em inteiros de 32 bits, pontos fixos, ou em ponto flutuante de meia precisão, caso esteja disponível no processador.

Com todas essas características fornecidas pela tecnologia NEON, a ARM resolveu uma deficiência de seus processadores, dando-lhes um ganho significativo de desempenho nas aplicações de áudio, vídeo e de gráficos 3D, assim, ampliando ainda mais a gama de suas aplicações.

5.2.3 SISTEMA DE SEGURANÇA

Outro aspecto que se mostrou falho na arquitetura ARM até então, e que vem se tornando uma preocupação importante devido, segundo ARM (2009f), ao aumento do uso de processadores ARM em aplicações envolvendo sistemas de crédito bancário, é a ausência de um sistema de segurança.

O conjunto de um processador sem um auxílio de um sistema de segurança e a utilização de plataformas de softwares abertos proporcionam uma grande vulnerabilidade nesse tipo de aplicação, e como o volume de dinheiro envolvido nessas aplicações é geralmente significativo, faz-se valer a pena para o invasor investir em quebrar esses sistemas e invadi-los.

Em resposta a essa falha importante, existem no mercado basicamente três tipos de sistemas de seguranças que podem ser implementados em conjunto com um processador, provendo a ele certo nível de segurança, que são: módulo externo de segurança, módulo interno de segurança e software de virtualização.

O módulo externo de segurança, por exemplo, os SIM cards e os sistemas de acesso condicional de smartcard, comuns em set-top box, possuem a vantagem de usar um hardware totalmente separado do processador, possibilitando o uso de técnicas e processos de silício que resultam em um alto nível de resistência à adulteração e a segurança física. Além disso, outra vantagem é a de possuir sistemas de certificação dos processos de fabricação, tornando- os assegurados para uso adequado em aplicações que necessitam confiabilidade, como cartões de crédito.

` Apesar dessas vantagens, esse tipo de sistema complementar possui uma enorme variedade de desvantagens, segundo ARM (2009f), que inviabilizam seu uso em projetos padrões, como por exemplo, a baixa eficiência energética, o baixo desempenho do dispositivo devido seu método de fabricação, em torno de 20 a 50 MHz, pouca memória RAM (Random Access Memory) disponível, a dependência de uso de software não tão seguro para acesso as suas funções, como o acesso através de PIN (Personal Identification Number), e o mais importante, o alto custo comercial.

O segundo sistema de segurança amplamente utilizado é o módulo interno de segurança, que, de acordo com ARM (2009f), possui dois tipos dominantes: o primeiro consiste de um bloco de hardware que gerencia as operações de criptografia e armazenamento de chaves, enquanto o segundo é um

mecanismo de processamento de uso geral, que é colocado ao lado do processador e que utiliza uma lógica de hardware personalizada para impedir o acesso não autorizado a recursos sensíveis.

Esse tipo de sistema de segurança sacrifica boa parte do nível de segurança proporcionado pelos smartcards, para trazer o benefício de um menor custo, a conveniência de já estar integrado ao sistema computacional. Além disso, os sistemas de segurança implementados através de processadores dedicados de propósito geral conseguem ter um bom desempenho, podendo ser comparado ao desempenho proporcionado pelo sistema TrustZone.

Apesar dessa equivalência de desempenho, uns dos problemas desse tipo de implementação é a necessidade de um processador de segurança física separado, que geralmente é menos poderoso que o processador de aplicativos principais e consome uma área de silício significativa, possui alto consumo energético, e necessita que a comunicação entre os processadores seja proveniente do uso de uma memória externa como intermediadora.

Já o sistema de segurança por software de virtualização, é um mecanismo que consiste basicamente da utilização de uma camada de gerenciamento altamente confiável, conhecido como hypervisor, executado em modo privilegiado de processador de propósito geral. Esta camada separa múltiplas plataformas de softwares independentes que rodam em cima dele usando a MMU, que coloca cada uma dentro de uma máquina virtual controlada pelo software hypervisor.

Esse tipo de sistema de segurança possibilita uma fácil implementação, já que qualquer processador que possui uma MMU pode implementá-lo. Porém, devido à isolação do processador através da execução do hypervisor, outros sistemas, como os processadores gráficos, podem fazer um bypass no hypervisor, diminuindo a segurança.

O suporte que o hypervisor pode fornecer a esses outros sistemas é de difícil implementação, e prejudica muito o desempenho do sistema. Além disso, a virtualização ignora os ataques através de hardware, como ameaças através de depuração ou infra-estrutura de teste, trazendo assim, uma grande vulnerabilidade a aplicação.

Logo, devido as grandes desvantagens e limitações dessas soluções comuns no mercado, a grande necessidade de resolver adequadamente esse problema e com o objetivo de sempre melhorar seus processadores, fazendo-os se destacarem perante os demais, a ARM desenvolveu a tecnologia TrustZone, que no momento esta sendo implementado nos Cortex- A8 e Cortex-A9.

5.2.3.1 TRUSTZONE

O objetivo primário, de acordo com ARM (2009f), dessa arquitetura de segurança é extremamente simples, a de permitir a construção de um ambiente que permita a confiabilidade e integridade a qualquer dispositivo contra ataques específicos. Uma plataforma com essas características se torna apta a construir um amplo conjunto de soluções de segurança que não são rentáveis com os métodos tradicionais.

Logo, diferentemente das demais soluções, segundo a ARM (2009f), o TrustZone não só protege uma parte restrita do sistema, ele permite a proteção de qualquer parte do sistema, permitindo uma solução de ponta a ponta, devido a implementação de três aspectos cruciais: a divisão dos recursos de hardware e software em dois mundos, o mundo Normal e o mundo Seguro, as extensões TrustZone implementadas no núcleo do processador e a infra-estrutura de depuração.

Primeiramente, a separação dos recursos em dois mundos distintos é devido a uma série de modificações em toda a arquitetura do processador, como por exemplo, uma modificação importante no sistema de barramento AMBA3 AXI, que inclui um sinal de controle extra para cada leitura e escrita dos canais do sistema de barramento principal.

Esses bits extras implementados são chamados de NS bits (Non-Secure bits), padronizando para 0 como mundo Seguro, e 1 para o Não-seguro, ou Normal. Com isso, todos os barramentos mestres mandam esse sinal quando eles fazem uma nova transação, e o barramento ou o decodificador lógico escravo deve interpretar eles e assegurar que a separação da segurança requerida não seja violada.

Devido a essa modificação, é possível utilizar periféricos seguros, como interrupções e timers seguros, que permitam o uso de um sistema de monitoramento, ou que um teclado seguro seja acoplado ao sistema, permitindo a entrada segura de usuários e suas senhas, e dispositivos de I/O (Input/Output), expandindo o ambiente seguro além dos dados, normalmente salvaguardados pelos sistemas de segurança tradicionais.

Outra modificação que auxilia na separação de mundos do processador, de acordo com ARM (2009f), é a adição de um bit de endereço no sistema de memória, fazendo-se assim que tenha disponível 32 bits de endereço físico para as transações do mundo Seguro, e outros 32 bits de endereço para o mundo Normal, acarretando em uma série de modificações no acesso da memória, e por conseqüência, no funcionamento da MMU.

O segundo aspecto, a inclusão das extensões do TrustZone no núcleo do processador, segundo a ARM (2009f), permite que um único processador físico execute os códigos de maneira segura e eficiente em ambos mundos. Essa implementação elimina a necessidade de um núcleo dedicado para a segurança do processador, economizando energia e área de silício, além de permitir que o software de segurança de alto desempenho rode ao lado do ambiente de operação normal.

O único processador implementa dois processadores virtuais, um para cada mundo, e um mecanismo robusto de troca entre eles, chamado de módulo monitor, como pode ser visualizado na Figura 17.

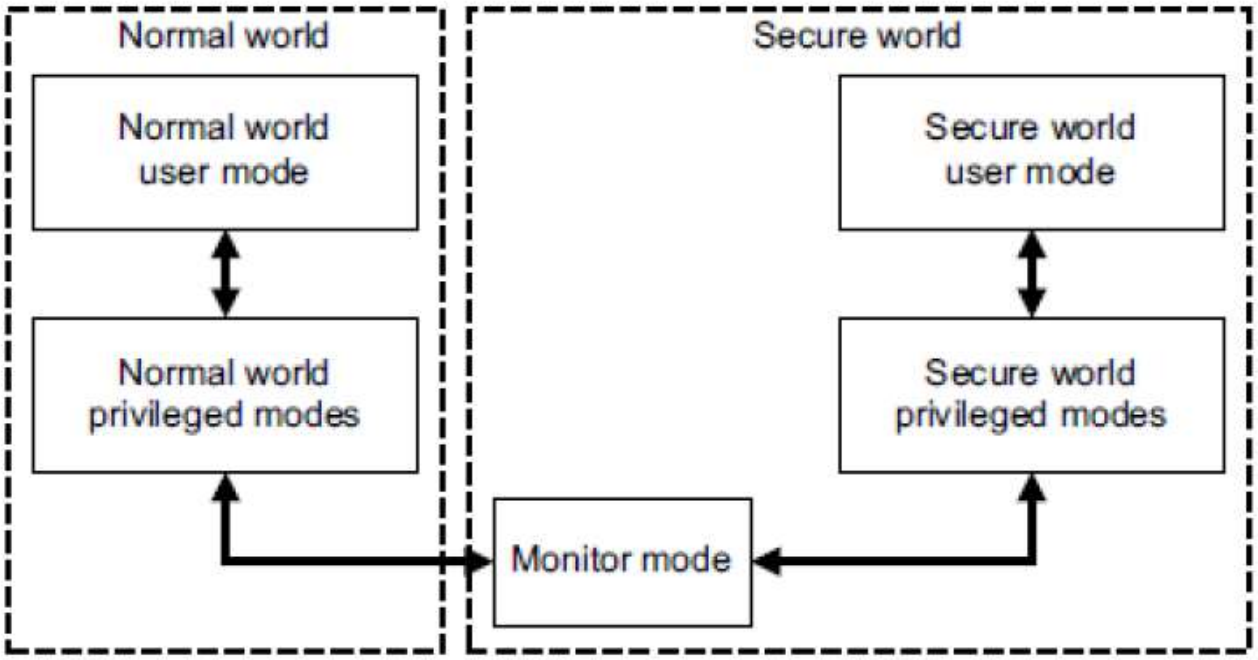

Figura 17 – Os dois mundos, ou processadores virtuais implementados pelo TrustZone.

Fonte: ARM, 2009f.

Mais uma modificação do processador com TrustZone é existência de dois MMUs virtuais dentro da MMU física, um para cada processador virtual, assim, possibilitando que cada mundo possua suas próprias TLBs, dando independência no controle do mapeamento dos endereços virtuais em físicos

Cada mundo possuindo suas próprias TLBs, torna-se necessário uma maneira de distinção das TLBs de acordo com seu mundo de origem, logo, acrescentaram-se o campo NS

para fazer essa distinção, associadas ao endereço físico na tabela de descritores. Essa fácil distinção entre as TLBs e seus mundos de origem possibilita a coexistência de ambos na cache, fazendo-se desnecessário realizar um flush na cache quando se muda de mundo, evitando assim perda de desempenho.

Com essas modificações nas TLBs, segundo ARM (2009f), o acesso a memória ocorre da seguinte maneira:

1. o núcleo lógico de processamento tenta carregar os dados, armazenar os dados, ou executar uma instrução. O hardware passa o endereço virtual (VA) e o mundo atual, através do NSTID (Non-Secure Table Identifier) para que a TLB possa realizar a tradução de endereços.
2. a TLB carrega o endereço físico (PA) e o NS bit associado com os VA e o NSTID que foram passados, realizando um caminho da tabela de páginas e forçando o NS =1 e o NSTID = 1, se necessário. O TLB passa essa informação para a cache executar os dados reais ou o acesso a instrução.
3. a cache tenta, a partir do PA e do bit NS do TLB, carregar a linha da cache existente. Se tudo der certo, ele irá retornar os dados dessa linha, caso contrário, ele irá carregar a linha de cache do sistema de memória externo.

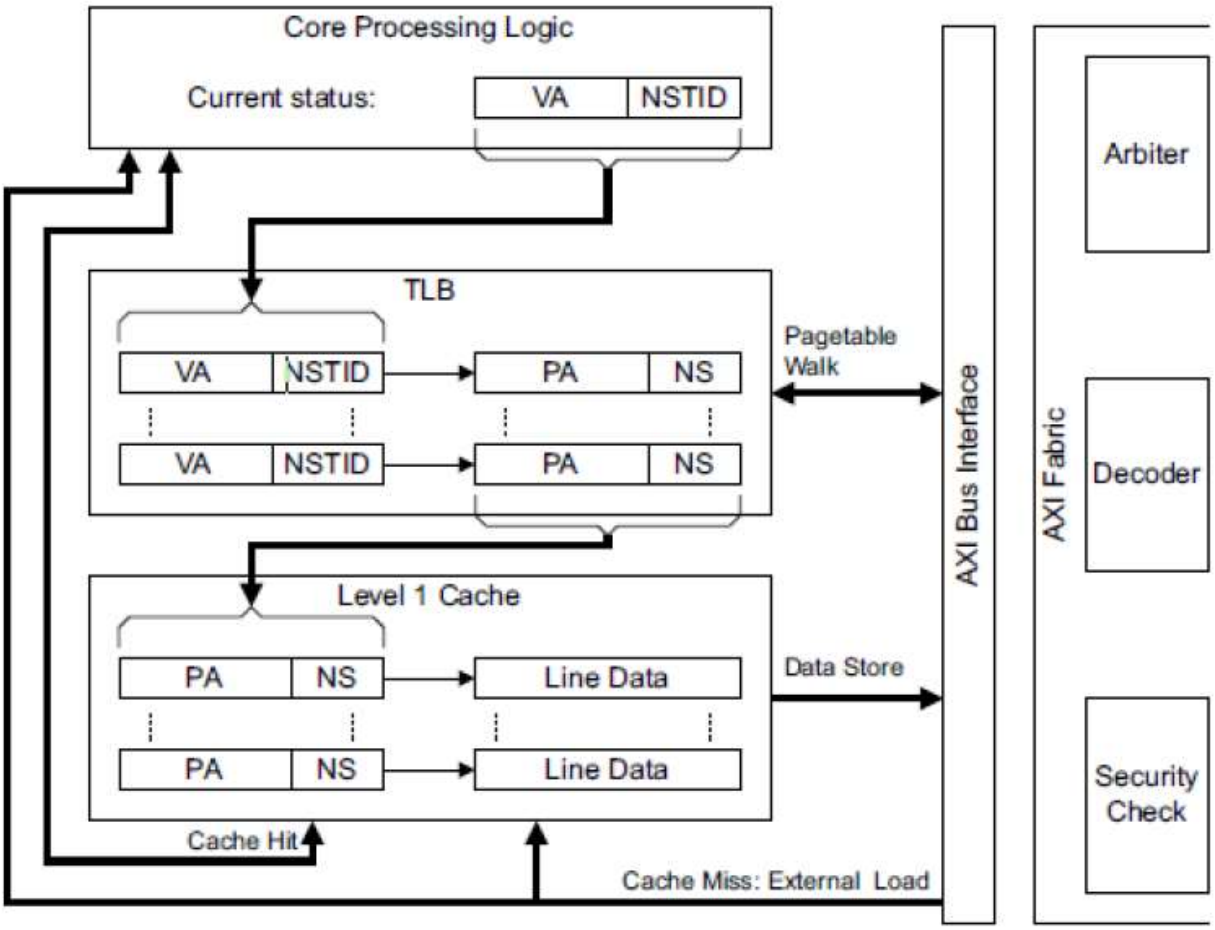

Figura 18 – Acesso a memória com o TrustZone. Fonte: ARM, 2009f.

Em relação às interrupções com o TrustZone, de acordo com ARM (2009f), o IRQ e o FIQ são interpretadas diretamente no módulo monitor, e uma vez chegando nele, o software confiável encaminha o pedido de interrupção em conformidade, sem a intervenção de código de qualquer dos mundos, permitindo a criação de um modelo flexível de interrupções de fontes seguras.

E no intuito de facilitar o uso das interrupções com essas características, a ARM recomendou um modelo de uso, onde se deve usar o IRQ como uma fonte do mundo Normal, e o FIQ como uma fonte provinda do mundo Seguro. Esse conselho se deve para facilitar a portabilidade de códigos antigos, já que o IRQ é mais comumente utilizado nos ambientes operacionais, e o FIQ seria usado como uma interrupção segura de menor uso, assim, alterando-se menos os códigos.

Assim, no momento que é gerado a interrupção, caso o processador já esteja no mundo de origem desta, não se faz necessário a mudança para o monitor, também gerando uma melhor eficiência, já que no caso contrário, é necessário que o monitor salve as configurações do mundo, mude para o

outro mundo para tratar a interrupção, e posteriormente fazer o processador retornar ao mundo anterior.

Já em relação ao CP15, a maioria das suas configurações só pode ser executada pelo mundo Seguro, já que tais resultam em modificações de resultados globais em todo o sistema, impactando em ambos os mundos. Logo, resta ao mundo Normal apenas a leitura de poucas configurações, sem nenhuma possibilidade de escrita neles.

Além disso, tudo, uma característica importante no TrustZone é o suporte de segurança a sistemas multiprocessadores, já que é possível utilizar um sistema com até 4 processadores ARM, fazendo-se cada processador possuir os dois mundos, e suportando a comunicação entre eles. Esse suporte é independente da configuração do sistema, assim, ele pode processar tanto no modo SMP (Symmetric Multi-Processing), quanto no modo AMP (Asymmetric Multi-Processing).

O aspecto final das modificações implementadas com o TrustZone é a infra-estrutura de segurança com reconhecimento de depuração, permitindo o controle sobre o acesso a depuração do mundo Seguro, sem prejudicar a visibilidade da depuração no mundo Normal. Além disso, fornecendo suporte a diversos recursos de análise de desempenho, como a verificação do tempo de execução do código, a contagem de eventos no processador em tempo real e a quantidade acessos a memória principal, encontrados no CP15.

Assim, devido a toda essas modificações na arquitetura ARM, o TrustZone possibilitou solucionar o problema da segurança nos processadores onde ele é implementado, como o Cortex-A8 e o Cortex-A9, viabilizando o seu uso nos mais diversos projetos de segurança antes não possibilitados, como o sistema bancário.

5.3 COMPARAÇÕES

Com o intuito de facilitar a visualização do futuro da arquitetura ARM, o quanto ela já evolui e assim, poder imaginar o que está por vim, foi feito um estudo comparativo entre os processadores ARM mais utilizados no momento, segundo Information Quartely (2005), os ARM7 e ARM9, e seus respectivos futuros substitutos, segundo Pereira (2007), os recentes processadores da família Cortex-M e Cortex-R.

E com o intuito de facilitar a comparação entre as famílias, foi escolhido como exemplares, o ARM7TDMI-S e o Cortex-M3, para a primeira comparação, e como exemplares para a segunda comparação, o ARM946E-S e o Cortex-R4.

5.3.1 ARM7 VS CORTEX-M

O processador Corte-M3, assim como todo membro da família Cortex, introduz uma grande variedade de modificações em toda sua arquitetura, se essa for comparada com a arquitetura do popular ARM7TDMI-S, demonstrando um ganho significativo de desempenho e ajudando a estabeler a sua aptidão para o uso em sistemas embarcados sensíveis ao custo e que exigem um comportamento determinístico, segundo Nagel (2008).

Uma das diferenças básicas é que, similar ao ARM9, o Cortex-M3 também implementa a arquitetura Havard, possuindo assim duas caches separadas, a I-cache e a D- cache, com todas as características descritas no capítulo 4, enquanto o ARM7TDMI-S implementa a obsoleta arquitetura Von Neumann.

Apesar dessa diferença básica em relação a arquitetura do processador, existe uma similaridade em relação ao pipeline, onde ambos possuem um pipeline de 3 estágios, porém com uma “leve” diferença: o Cortex-M3 possui seu pipeline com execução especulativa. Assim, quando uma instrução de desvio é encontrada, com a execução especulativa o segundo estágio do pipeline, a decodificação, também possui a função de fetch especulativo. Esse fetch especulativo busca a instrução do ramo destino do desvio, para que quando esse seja executado, caso o código se desvie a próxima instrução já esta disponível, diminuindo

assim o tempo ocioso para um ciclo, dando um ganho no desempenho.

Outra diferença é a quantidade de registradores disponíveis, que enquanto no ARM7TDMI-S existia ao todo 37 registradores, no Cortex-M3 só se encontram 13 registradores de propósito geral (R0 à R12) de 32 bits, registradores adicionais como os 2 registradores SP (Stack Pointer), o LR, e o PC, e outros registradores especiais como o APSR (Application Program Status Register), o IPSR (Interrupt Program Status Register) e o EPSR (Execution Program Status Register), segundo ARM (2008).

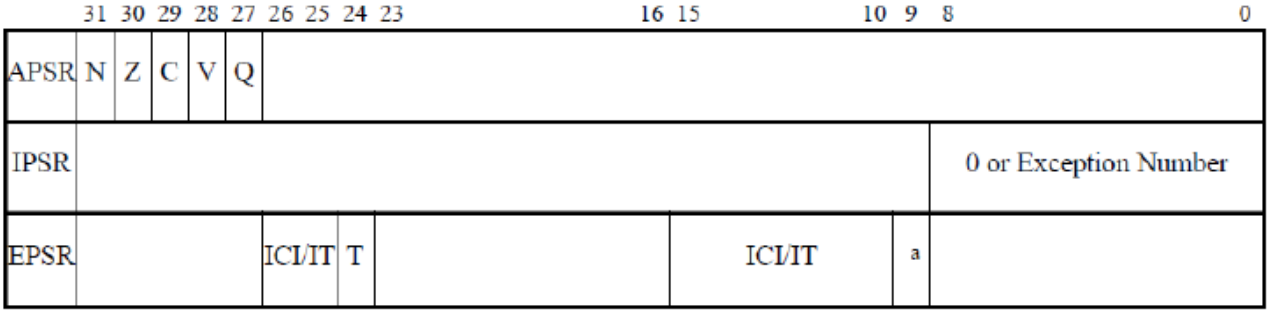

Figura 19 – Os bits do APSR, IPSR e do EPSR. Fonte: ARM, 2008.

Como pode ser visualizado na Figura 19, o registrador APSR possui o bits N, Z, C, V e Q, similares aos respectivos bits do registrador CPSR do ARM7TDMI-S, tendo seus demais bits reservados para futuras implementações. Já o registrador IPSR é modificado com a saída ou entrada de uma exceção, usando o campo de número da exceção para determinar a exceção corrente e o vetor de entrada a ser utilizado.

Enquanto isso, o registrador ESPR contém o bit T, que determina o estado da CPU, que se deve estar sempre configurado com valor 1 (valor de reset) devido a ausência do modo ARM. Existem também no ESPR os bits de estado de execução ICI/IT que dão suporte as instruções IT (interrupt-continue load/store), salvando o estado IT ou salvando as exceções geradas pelo IT.

Outro aspecto importante é que no processador Cortex-M3, diferentemente do ARM7TDMI-S que possui 7 modos diferentes de processamento, possui apenas dois modos de processamento, o modo Thread e o modo Handler e dois níveis de acesso de código, o privilegiado e o não-privilegiado.

O acesso não-privilegiado possui limites de execução e exclui o acesso a alguns recursos como certas instruções e localizações específicas na memória, enquanto que o acesso privilegiado tem acesso total a todos os recursos, assim possibilitando projetos de sistemas complexos e abertos, sem sacrificar a segurança do aplicativo.

O modo Thread é o modo padrão de operação, e suporta ambos os tipos de código, enquanto que o modo Handler entra quando ocorre algum tipo de exceção, sendo esse todo de código privilegiado. Além disso, apresenta dois estados, o estado Thumb, padrão, e um segundo estado diferente do tradicional ARM, o estado Debug para depuração de atividades.

Pode-se também observar no Quadro 8 a modificação em relação ao suporte ISA (Instruction Set Architecture), o suporte aos diferentes tipos de conjuntos de instruções, onde o Cortex-M3 não suporta o conjunto de instruções ARM, suportado pelo ARM7TDMI-S. Ao invés dele, o Cortex-M3 suporta o conjunto de instruções Thumb-2, onde pode-se encontrar algumas instruções de 32 bits e outros recursos, como descrito no tópico 5.2.1, ajudando-o, de acordo com Sadasivan (2006), a ser 70% mais eficiente por MHz do que o ARM7TDMI-S no modo Thumb, e 35% mais eficiente em relação ao tamanho do código do que o ARM7TDMI- S no modo ARM.

Em relação às interrupções, diferentemente do ARM7TDMI-S que implementava o FIQ e o IRQ, com latência de 24 a 42 ciclos, que eram tratados diretamente pelo núcleo do processador, o Cortex-M3

possui um controlador de interrupções altamente configurável chamado NVIC (Nested Vectored Interrupt Controller).

Características	ARM7TDMI-S	Cortex-M3
Arquitetura	Von Neumann	Havard
Suporte ISA	*Thumb*/ARM	*Thumb*/*Thumb*-2
Pipeline	3 Estágios	3 Estágios com execução especulativa
Interrupções	FIQ/IRQ	NMI + 1 a 240 Interrupções físicas
Latência das Interrupções	24-42 ciclos	12-ciclos
Modo *Sleep*	Nenhum	Integrado
Proteção de Memória	Nenhum	8 regiões com MPU
Dhrystone	0,9 DMPS/MHz (Modo ARM)	1,25 DMPS/MHz
Consumo energético	0,28mW/MHz	0,19mW/MHz
Área	0,6mm^2 (somente o núcleo)	0,86mm^2 (núcleo e periféricos)

Quadro 8 – Comparação entre o ARM7TDMI-S e o Cortex-M3.

Fonte: Sadasivan, 2006.

Segundo Sadasivan (2006), o NVIC fornece como padrão um NMI (Non-Maskable Interrupt) e 32 interrupções físicas de propósito geral de 12 ciclos de latência, com 8 níveis de prioridade, podendo ser configurado para possuir uma faixa entre 1 a 240 interrupções físicas com até 256 níveis de prioridade, trazendo assim uma excelente habilidade de manipulação de interrupções.

Além de ser um controlador de interrupções, o NVIC possui outras funcionalidades como, segundo Nagel (2008) e Sadasivan (2006), a implementação do System Tick (SysTick), um timer de 24 bits, que pode ser configurado para gerar uma interrupção de intervalos de tempos regulares, ideal para uso

por RTOS ou por tarefas agendadas, lembrando que para se usar um timer no ARM7TDMI-S, é necessário implementá-lo externamente.

Outra funcionalidade do NVIC é a implementação do sistema de gerenciamento de energia do processador que suporta o modo Sleep, diferentemente do ARM7TDMI-S que não possui tal. O modo Sleep é invocado por qualquer WFI (Wait For Interrupt), ou por qualquer WFE (Wait For Event), colocando o núcleo imediatamente em um baixo consumo de energia na pendência da exceção.

Além do sistema de gerenciamento de energia do NVIC, o bit SLEEPDEEP do CP15 pode ser definido para setar o valor do clock do processador, fazendo o processador poupar mais energia, ultrapassando assim, de acordo com Sadasivan (2006), a marca dos 30% de economia de energia que o Cortex-M3 possui em relação ao ARM7TDMI-S.

Outra significante diferenciação entre os processadores é a existência de uma MPU no Cortex-M3, que separa a memória em até 8 regiões, onde cada uma se subdivide em 8 sub- regiões. Os tamanhos de região suportados começam de 32 bytes e aumentam por fator de 2, indo até 4GB de memória, tendo seu funcionamento parecido com a MPU descrita no capítulo 4, com suas modificações e algumas a mais.

Com relação ao desempenho, observa-se no Quadro 8 a diferença significante entre ambos, onde o ARM7TDMI-S possui o desempenho de 0,9 DMPS/MHz no seu modo ARM, enquanto o seu substituto, o Cortex-M3 possui os incríveis 1,25 DMPS/MHz. Porém, mesmo com o melhor desempenho do Cortex-M3, para alguns tipos de aplicações onde o tamanho reduzido é o objetivo principal e não há a necessidade de periféricos, o ARM7TDMI-S tem a vantagem de ter 0,6mm2 de núcleo, enquanto o Cortex-M3 possui 0,86mm2 de núcleo e periféricos.

Outro aspecto em relação ao desempenho, segundo Sadasivan (2006), é que o Cortex- M3 possui a possibilidade, até então não existente, de poder escolher duas formas de funcionar, uma maneira hard e outra inteligente.

Na maneira hard, eleva-se a freqüência do clock do processador, fazendo-o aumentar seu desempenho, porém acompanhado com um maior consumo de energia e de complexidade de projeto. Por outro lado, na maneira inteligente, o processador trabalha com uma menor freqüência no clock, aumentando sua eficiência computacional, diminuindo a complexidade do projeto e consumo energético, e podendo executar as mesmas tarefas da maneira anterior.

Com todos esses recursos e desempenho superior, o Cortex-M3 possui a tendência de ser o substituto do ARM7TDMI-S meramente devido a sua simplicidade frente aos outros processadores da família Cortex, e pela similaridade de aplicações alvo, os de baixo custo.

5.3.2 CORTEX-R VS ARM9

Similar a comparação anterior entre o Cortex-M3 e o ARM7TDMI-S, existe também uma grande diferença, não só no desempenho, mas também de características da arquitetura e de recursos entre o Cortex-R4 e o ARM946E-S, como o pipeline, registradores, uso da memória RAM, a MPU, entre outros.

A arquitetura empregada em ambos processadores é a mesma, a arquitetura Havard, possuindo assim dois barramentos separados, uma para instruções e outra para dados, além da divisão da cache em duas, I-cache e D-cache, com todas as possibilidades de configuração e recursos, como a cache lockdown, descritos no capítulo 4.

Além dessa similaridade em relação à cache, ambos processadores possuem uma cache 4 way associative, porém com a diferença que no ARM946E-S, a cache possui uma linha de cache de 4 words, enquanto que no Cortex-R4 possui sua linha de cache com 8 words, além da configuração do tamanho da cache, onde no primeiro varia de 4 a 128 KB, enquanto que no segundo varia de 4 a 64 KB.

Em relação ao pipeline, há significativas diferenças, desde o aumento da extensão para 8 estágios pelo Cortex-R4, diferentemente do ARM946E-S que possui um pipeline 5 estágios, como também a divisão dos últimos estágios e a implementação de branch prediction.

Com o aumento da extensão do pipeline, houve a possibilidade de incrementar a freqüência do clock, resultando num melhor desempenho do processador. Observa-se também que seus últimos estágios foram divididos em 4 pipelines em paralelo, cada um com um tratamento para um tipo específico de instrução, possibilitando a execução de instruções em paralelo nesse trecho do pipeline, diferentemente do ARM946E-S que só pode executar uma única instrução por vez. As divisões dos últimos estágios do pipeline, segundo Lewin (2006), são:

- load/store: parte do pipeline responsável por processar todos os acessos a memória divididos em dois estágios, permitindo vários acessos a RAM sem perda de bandwidth;
- MAC: operações de multiplicação são divididos em 3 estágios de pipeline, onde a etapa final atualiza o banco de registradores;

- ULA: operações aritméticas que usam uma etapa de pre-shift, e em seguida, opcionalmente, saturar antes de atualizar o banco de registradores;
- divisor: um divisor que utiliza o algoritmo Radix-4 para uma divisão típica de 32 bits, com duração de 6 ciclos do clock para um único estágio do pipeline.

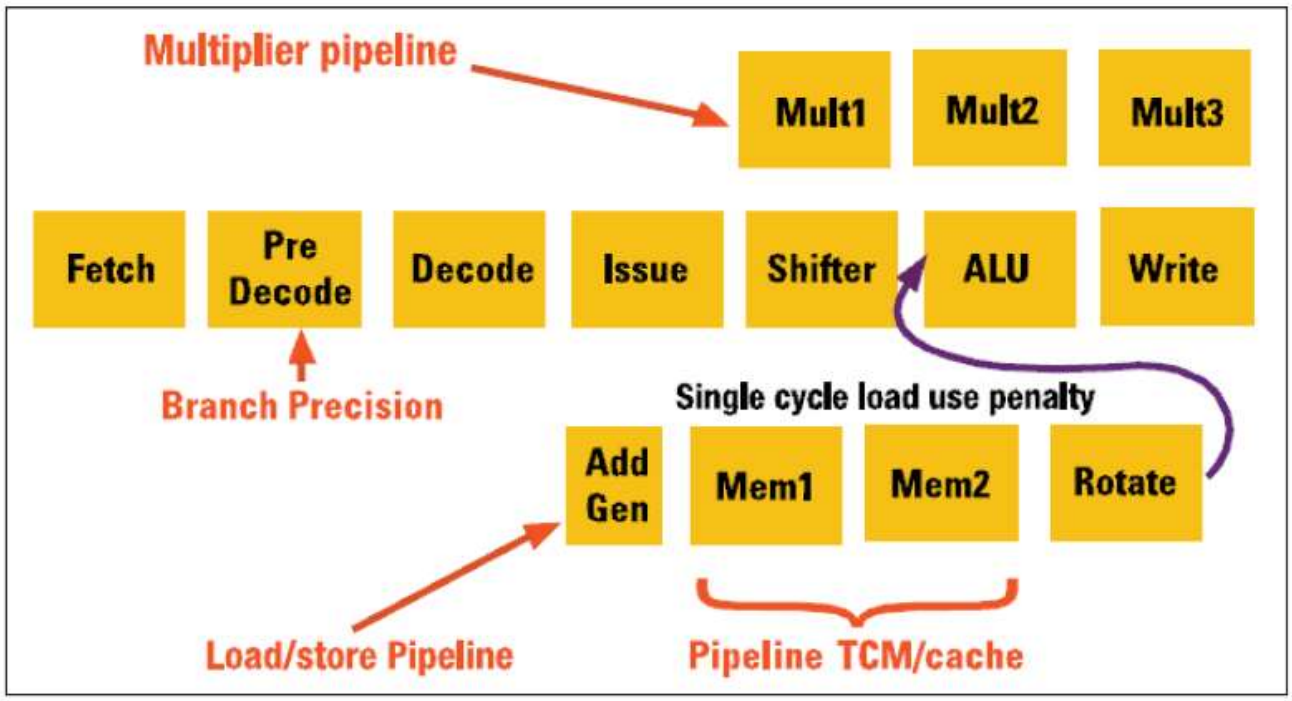

Figura 20 – Pipeline do ARM Cortex-R4. Fonte: Penton, 2006.

A outra modificação no pipeline foi a implementação do branch prediction com uma PFU (Prefetch Unit), que detecta os desvios no Pd-stage do pipeline, predicando ou não o desvio que será realizado, e determina ou predica os endereços e os desvios a serem utilizados. Assim, o processador tem um ganho adicional de desempenho já que a maioria dos desvios são predicados corretamente, diminuindo a ociosidade do processador e a necessidade de flush no pipeline, necessário no caso de erro de predicação.

- Ademais, o Cortex-R4 possui os mesmos 37 registradores que o ARM946E-S possui, os 31 registradores de propósito geral de 32 bits, e os 6 registradores de status, também de 32 bits. Porém, a algumas diferenças em relação aos bits do CPSR, utilizando alguns dos bits não especificados até então pelo ARM946E-S, segundo ARM (2009b), como os bits:
- J: chamado de Java State bit. Retorna 0 quando é lido e não se consegue utilizar a instrução MSR para modificá-lo;

- DNM: chamados de Do Not Modify bits. Este campo não pode ser modificado via software. Eles podem ser:

➢ lidos: para preservar o setado do processador, por exemplo, durante um processo de mudança de contexto;

➢ escritos: para fazer o processador restaurar seu estado;

- GE[3:0]: chamados de Greater than or equal to bits. Utilizado por algumas instruções SIMD, que modificam GE [3:0] de acordo com os halfwords ou os bytes dos resultados de suas operações;
- IT[7:2]: auxiliam na execução dos blocos IT, resultando num menor número de instruções a serem executadas;
- E: chamado de Data endianness bit, controla o load/store endianness, podendo ser modificado com instruções ARM ou Thumb;
- A: chamado de Imprecise abort disable bit. Automaticamente modificado, desabilita data aborts imprecisos.

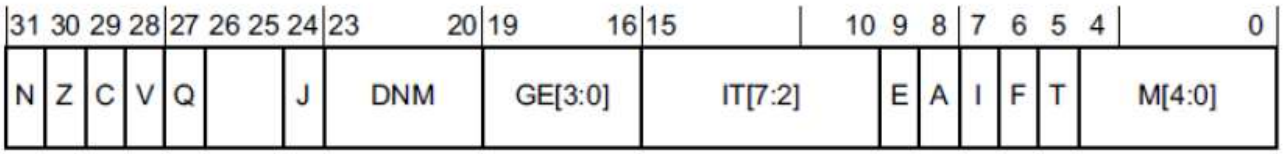

Figura 21 – Current Program Status Register no Cortex-R4. Fonte: ARM, 2009b.

Acima, se visualiza a configuração dos bits do CPSR no Cortex-R4, podendo-se observar não só os novos bits acrescidos, mas também a continuidade da existência dos bits anteriores, como os bits M[4:0], que indicam a continuidade da existência dos 7 modos de processamento, similares aos do ARM946E-S, incluindo o uso do mesmo modelo de tratamento dos data aborts, o Base Restored Data Abort model.

Outra diferença entre o Cortex-R4 e seus antecessor, o ARM946E-S, é o suporte a dados, adicionando-se o suporte a doubleword (64 bits) aos já existentes word (32 bits), halfword (16 bits) e bytes (8 bits). Entretanto, para um melhor desempenho do processador, é sugerida por ARM (2009b) a utilização de dados como aligned, com a exceção dos bytes.

O Cortex-R4 suporta os dois conjuntos de instruções suportados pelo ARM946E-S, o conjunto ARM e o Thumb, além do mais recente Thumb-2, fornecendo uma vantagem significante em relação ao

desempenho e a densidade de código, descritos detalhadamente no tópico 5.2.1, tornando as aplicações desenvolvidas com o Cortex-R4 potencialmente mais baratos do que o mesmo com o ARM946E-S, devido à menor necessidade de memória.

Outra diferença é em relação ao uso da memória RAM. O ARM946E-S necessita de 40% do ciclo do clock para acessar a RAM, ou seja, a 200 MHz (ciclo de 5ns), o tempo de resposta é de 2ns. Enquanto isso, o Cortex-R4, utilizando-o com o dobro da freqüência do clock, a 400 MHz, o seu tempo de resposta é de 2.5 ns, possibilitando que o Cortex-R4 utilize biblioteca de memória de baixa velocidade, o que reduz drasticamente a área de silício a ser utilizada e o consumo de energia.

Um exemplo, demonstrado em Lewin (2006) para melhor visualizar essa diferença significante, é se associarmos um Cortex-R4 com uma memória RAM Artisan Metro, que tem seu uso possibilitado devido às características do Cortex-R4 e melhor do que o normalmente utilizado Artisan Advantage pelo ARM946E-S, possibilitando ao projeto ter 35 % de área reduzida e uma redução de 54% no consumo energético, diferença vital em sistemas embarcados.

Já na interface de memória, o Cortex-R4 implementa um AMBA3 AXI de 64 bits, o dobro do tamanho do AHB do ARM946E-S. O ganho de desempenho no uso do AXI não se deve apenas a maior quantidade de bits em um único acesso, mas também por poder acessar mais endereços de uma só vez e acessar dados fora de ordem. Com isso, segundo Lewin (2006), essas características dão significativas vantagens ao Cortex-R4 frente que muitas aplicações possuem uma memória lenta ou periféricos que bloqueiam o barramento durante seu tempo de acesso.

E há certa similaridade entre os processadores no aspecto de proteção de memória, onde ambos implementam uma MPU com todas as características e modificações já descritas no capítulo 4. Há apenas pequenas modificações, entre elas estão à possibilidade de no Cortex-R4 possui não apenas 8 regiões, mas também a opção de possuir 12 regiões administradas pela MPU, permitindo uma maior flexibilidade se requerida, e o tamanho mínimo da região de 32 bytes, permitindo seu fino controle e reduzindo o desperdício de memória.

Outra similaridade é a continuidade do uso das interrupções IRQ e FIQ pelo Cortex- R4, apenas com a modificação em relação ao pior tempo acesso, onde no ARM946E-S, de acordo com Lewin (2006), possui seu pior tempo com 118 ciclos do clock, enquanto que agora no Cortex-R4 possui seu pior tempo com apenas 20 ciclos do clock, dando assim uma enorme vantagem para uso do Cortex-R4 em sistemas de tempo real que necessitam de uma rápida resposta a suas interrupções.

E um recurso adicionado ao Cortex-R4 e, segundo Penton e Jalloq (2006), de importância significativa para sistemas críticos de segurança, é o suporte de detecção por paridade e ECC (Error Correction Codes), que possuem o intuito de detectar e corrigir os erros gerados pela interferência de nêutrons ou de partículas alfa na memória devido a fragilidade gerada pelo seu tamanho reduzido.

Quando a detecção por paridade é habilitada, um bit extra é adicionado nos dados armazenados na RAM para futuras verificações, aplicando o algoritmo de bit paridade. Verificando-se o erro pela detecção por bit paridade, existem dois mecanismos de solução:

- correção por hardware: o processador pode ser configurado para invalidar a linha de cache que possui a falha, forçando um write-through, e assim a instrução correspondente é re-executada, ou a utilização da memória externa no caso de falta de dados corretos na cache;
- notificação por software: um abort pode ser gerado pela instrução ou pelo acesso de dados, permitindo que o software corrija o erro, podendo assim a levar a terminação do processo em circunstâncias fatais. O DFSR (Data Fault Status Register) permite que se identifique o tipo de abort gerado e o DFAR (Data Fault Address Register) fornece o endereço que gerou o abort. Assim, no caso que a linha da cache não esteja suja, o handler do abort pode invalidá-lo e acessar a memória externa para adquirir o dado livre de erro. Caso contrário, o erro detectado pelo bit paridade será irrecuperável.

Com relação ao desempenho, segundo Peyton e Jalloq (2006), a diferença é significante entre ambos, pois o Cortex-R4 possui um incremento de 40% em DMIPS/MHz em relação ao ARM946E-S, com 1.6 DMIPS/MHz e 1.14 DMIPS/MHz, respectivamente, além de um incremento de 30% na freqüência máxima do clock, com 280 MHz e 210 MHz, respectivamente.

Assim, devido todas essas características, novas ou melhoradas, e em conjunto com o melhor desempenho, o Cortex-R4 se consagra o futuro substituto do ARM946E-S, necessitando de apenas uma maior quantidade de fornecedores e menores preços para que isso ocorra, segundo Pereira (2007), além de ajudar a firmar a família Cortex-R como a preferencial em algumas aplicações de alto desempenho e de sistemas de tempo real.

6 CONCLUSÃO

Como pode ser visto ao longo de todo este trabalho, a arquitetura ARM de microcontroladores de 32 bits implementa um série de recursos adicionais ao projeto tradicional de processadores RISC,

auxiliando-o na redução do consumo energético, diminuição da área do processador, diminuição do custo do projeto, além do alto desempenho, vitais para a sua utilização em sistemas embarcados.

Esses recursos vão desde aumento do pipeline com a finalidade de aumentar a freqüência máxima do clock, e o conjunto de instruções Thumb para a redução do tamanho do código, e assim, diminuir a quantidade de memória necessária, reduzindo o custo do projeto, além da implementação da VMSA e da PMSA, que possibilitam uma maior proteção no sistema de memória e a implementação de sistemas operacionais complexos.

E assim, pode ser observado à constante evolução da arquitetura ARM, introduzindo cada vez mais esses tipos de recursos auxiliares, desde a diferenciação significativa entre o ARM7 e o ARM9, com a implementação da arquitetura Havard e a confirmação da tendência de uso das MMUs e MPUs, até a mais recente família, a família Cortex, com a introdução do Thumb-2, NEON e TrustZone.

Com isso, é possível vislumbrar a predominância dos processadores ARM no mercado, não somente entre os sistemas embarcados, mas em outros novos nichos de aplicação, como os sistemas bancários e os de games portáteis, e a evolução de sua arquitetura, objetivando sempre o baixo custo, baixo consumo energético e alto desempenho, os três pilares da filosofia de projetos de processadores ARM.

7 REFERÊNCIAS

ANDREWS, Jason R. Co-verification of hardware and software for ARM SoC design.1. ed. Burlington, United States of America: Elsevier, 2005. 260 p.

ARM. Architecture and Implementation of the ARM Cortex-A8 Microprocessor. UK, 2005a.

_____. ARM7 Processor Family. Disponível em: <http://www.arm.com/products/processors/classic/arm7/index.php>. Acesso em: 20 de janeiro de 2010a.

_____. DDI 0084E: ARM7TDMI-S: Technical Reference Manual. UK, 1999. 218 p.

_____. DDI 0100E: ARM Architecture Reference Manual. UK, 2000a. 811 p.

_____. DDI 0100I: ARM Architecture Reference Manual. UK, 2005b. 1138 p.

_____. DDI 0151C: ARM920T: Technical Reference Manual. UK, 2001. 340 p.

_____. DDI 0180A: ARM9TDMI: Technical Reference Manual. UK, 2000b. 158 p.

_____. DDI 0186E: ARM966E-S: Technical Reference Manual. UK, 2000c. 192 p.

_____. DDI 0201C: ARM946E-S: Technical Reference Manual. UK, 2003. 232 p.

_____. DDI 0213E: ARM966E-S: Technical Reference Manual. UK, 2004a. 188 p.

_____. DDI 0235C: MOVE Coprocessor: Technical Reference Manual. UK, 2004b. 48 p.

_____. DDI 0240B: ARM9E-S Core: Technical Reference Manual. UK, 2004c.

_____. DDI 0344B: Cortex-A8: Technical Reference Manual. UK, 2006. 730 p.

_____. DDI 0363E: Cortex-R4 and Cortex-R4F: Technical Reference Manual. UK, 2009a. 456 p.

_____. DDI 0403C: ARMv7-M Architecture Reference Manual. UK, 2010b. 716 p.

_____. DDI 0406B: ARM Architecture Reference Manual: ARMv7-A and ARMv7-R. UK, 2009b. 2158 p.

_____. DDI 0407F: Cortex-A9 MPCore: Technical Reference Manual. UK, 2010c. 168 p.

_____. DDI 0413C: Cortex-M: Technical Reference Manual. UK, 2008. 234 p.

_____. DDI 0450A: Cortex-A5 NEON Media Processing Engine: Technical Reference Manual. UK, 2009c. 32 p.

_____. DHT 0001A: Architectures, Processors, and Devices: Development Article. UK, 2009d. 12 p.

_____. DHT 0002A: Introducing NEON: Development Article. UK, 2009e. 18 p.

_____. DUI 0021A: Programming Techniques. UK, 1995. 258 p.

_____. PRD29-GENC-009492C: ARM Security Technology: Building a Secure System using TrustZone Technology. UK, 2009f. 108 p.

BRASH, David. The ARM Architecture Version 6 (ARMv6). UK, 2002. 15 p. CORMIE, D. The ARM11 Microarchitecture. Advanced RISC Machine, 2002.9 p.

DANDAMUDI, Sivarama P. Guide to RISC Processors for Programmers and Engineers. 1. ed. United States of America: Springer, 2005. 387 p.

FRANCIS, H. ARM DSP-Enhanced Extensions. Advanced RISC Machine, 2002. 7 p.

FURBER, Steve. ARM system-on-chip architecture. 2. ed. Harlow, United Kingdom: Pearson, 2000. 418 p.

GARDNER, J. Scott. ARM Cortex: M3 Challenges the Economics of 8-bit Microcontrollers.

Information Quartely Magazine, volume 5, número 3, 2006. Technology In-Depth, p. 28.

HAMACHER, V.Carl.; VRANESIC, Zvonko G; ZAKY, Safwat G. Computer organization. 4th ed. New York: McGraw-Hill, c1996. 555 p.

HIXON, Todd. Migrating ARM7 code to a Cortex-M3 MCU. Embedded.com, 30 de outubro de 2009. Disponível em: <http://www.embedded.com/design/testissue/ 220900313?_requestid=66731>. Acesso em: 11 de junho de 2010.

INFORMATION QUARTELY. ARM Leads Embedded Systems Development in China.

Information Quartely Magazine, volume 4, número 3, 2005. Intelligence, p.12.

KEIL, Reinhard. ARM provides the microcontroller solution. Embedded.com, 18 de fevereiro de 2008. Disponível em: <http://www.embedded.com/columns/ technicalinsights/206800805?pgno=1>. Acesso em: 11 de junho de 2010.

LEMIEUX, Joe. Introduction to ARM thumb. Embedded.com, 24 de setembro de 2003. Disponível em: <http://www.embedded.com/shared/ printableArticle.jhtml?articleID=15200241>. Acesso em: 11 de junho de 2010.

LEWIN, Peter. Cortex-R4: A Comparison with the ARM9E Processor Family. Information Quartely Magazine, volume 5, número 3, 2006. Technology In-Depth, p. 24.

MARTIN, Trevor. The Insider's Guide to the Philips ARM7: Based Microcontrollers. Coventry,United Kingdom: Hitex Ltd, 2005. 198 p.

MARTIN, Trevor; BEACH, M. The Insider's Guide to the STR91x ARM9. Coventry,United Kingdom: Hitex Ltd, 2006.

NAGEL, Brian. Advantages of the Cortex-M3. Information Quartely Magazine, volume 7, número 4, 2008. Technology In-Depth, p. 27.

PENTON, John; JALLOQ, Shareef. Cortex-R4: A Mid-range Processor for Deeply-embedded Applications. Information Quartely Magazine, volume 5, número 3, 2006. Technology In- Depth, p. 14.

PEREIRA, Fábio. Tecnologia ARM: microcontroladores de 32 bits. São Paulo: Érica, 2007. 448 p.

PHELAN, R. Improving ARM Code Densiy and Performance (Thumb-2). Advanced RISC Machine, 2003. 8 p.

SADASIVAN, Shyam. An Introduction to the ARM Cortex-M3 Processor. Advanced RISC Machine, 2006. 17 p.

SCOTT, Graham. Encounter Express: Simplifying ARM Cortex-A8 Implementation. Information Quartely Magazine, volume 6, número 1, 2008. Technology In-Depth, p. 18.

SLOSS, Andrew; SYMES, Dominic; WRIGHT, Chris. ARM systems developer's guide: designing and optimizing system software. California: Morgan Kaufmann, 2004. 689 p.

SOUSA, D. R. Microcontrolador ARM7 (Philips,Família LPC213X): o poder dos 32 bits: teoria e prática. 1. ed. São Paulo: Érica, 2006. 278 p. 689 p.

VERMA, Manish; MARWEDEL Peter. Advanced Memory Optimization Techniques for Low-Power Embedded Processors. Dordrecht, the Netherlands: Springer, 2007. 188 p.

WHEALTON, Karl. Designing Solutions for the Home with the Cortex-A8. Information Quartely Magazine, volume 7, número 2, 2008. The ARM-Enabled Home, p. 42.

YIU, Joseph. The Definitive Guide to the ARM Cortex-M3. 1. ed. Oxford, UK: Elsevier, 2007. 359 p.

_____. What next for microcontrollers?. Embedded.com, 05 de janeiro de 2010. Disponível em: <http://www.embedded.com/design/222200229?_requestid=57508>. Acesso em: 11 de junho de 2010.

Wild Freitas da Silva Santos

conhecimentolivre.org/home
contato@conhecimentolivre.org
editoraconhecimentolivre

ESTUDO DA ARQUITETURA ARM

EDITORA CONHECIMENTO LIVRE