A Arquitetura do DSP TMS320C67x

Davi de Andrade Lima Castro
RA: 107072
davi.castro@gmail.com

RESUMO

Este artigo descreve a arquitetura da plataforma C6000, mais especificamente a variação que possui unidades de execução de ponto flutuante, TMS320C67x, de Processadores de Sinais Digitais (*Digital Signal Processor* - DSP) da Texas Instruments.

Palavras-chave

DSP, TSM320, C6000, C67x, arquitetura, VLIW, VelociTI, ponto-flutuante, *embedded*, sistemas embarcados, processamento digital de sinais.

1. INTRODUÇÃO

Os DSPs TSM320C67x fazem parte da plataforma de alto desempenho C6000 juntamente com as outras variações: C62x, C64x(+), C67x+ e mais recentemente a C674x; sendo as duas primeiras focadas em operações de inteiros (e sem suporte a ponto-flutuante), a terceira apenas uma versão da C67x com aprimoramentos para melhorar o desempenho, e a quarta uma nova variação com compatibilidade binária com ambas as variações C67x(+) e C64x(+) e com maior foco em *low-power*[1].

A plataforma C6000 é atualmente a de maior desempenho de toda a família de DSPs da Texas sendo a primeira e única com a arquitetura VLIW chamada de VelociTI. É importante notar que todas as variações citadas acima possuem esta mesma arquitetura em comum.

2. ARQUITETURA

2.1 Características Gerais

- Arquitetura **VLIW**:
  - Instruções de 32-bits
  - *Fetch* de oito instruções em cada ciclo, sendo possível executar todas as oito no mesmo ciclo
  - Execução Serial ou Paralela, com controle individual para cada instrução
- Execução condicional de cada instrução
- Oito Unidades Funcionais: dois multiplicadores e seis unidades de lógica e aritmética (ALUs)
- 32 registradores de 32-bits
- *Datapath* de 32 bits, com possível extensão de mais 8 bits para precisão extra
- Operações em ponto-flutuante, suportando precisão simples (32-bits) e dupla (64-bits)
- Arquitetura do tipo *Harvard*
- Arquitetura do tipo *load-store*, somente certas instruções acessam a memória, enquanto todas as outras operam somente nos registradores
- Suporte a dados de 8/16/32-bits
- Além do endereçamento linear (comum a processadores de propósito geral) possui modo de endereçamento circular

2.2 Diagrama de Blocos

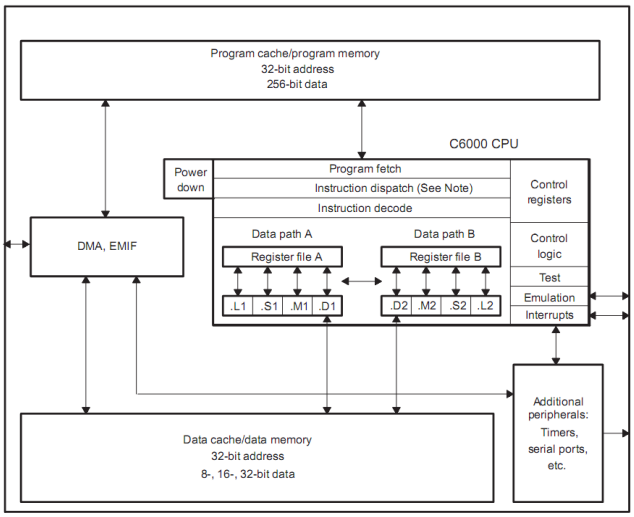


Figura 1 - Diagrama de Blocos do DSP TMS320C67x

O Diagrama de Blocos da arquitetura C67x pode está representado na Figura 1 (retirada de [2]). Nele podemos identificar todos os componentes principais que compõem a arquitetura:

Tem-se uma CPU da plataforma C6000. Como destaque pode-se citar a existência de dois *datapaths*, cada um contendo quatro unidades funcionais e seu próprio banco de registradores. As unidades de busca, despacho e decodificação são responsáveis por entregar até oito instruções, uma para cada unidade funcional, em um mesmo ciclo. Já os registradores e lógica de controle permitem configurar alguns aspectos da operação da CPU, como por exemplo, tipos diferentes de arredondamentos para as operações de ponto flutuante.

Por ser uma arquitetura do tipo *Harvard*, temos as memórias independentes para dados e programa. Todos os modelos possuem memórias internas (*on-chip*), embora o tamanho varie, chegando em até 7M bits [2]. Têm-se duas portas de 32-bits para o acesso à memória interna de dados. Já o acesso à memória interna de

programa é feito por uma porta de 256-bits, que é a largura de um só pacote de instruções.

Certos modelos possuem também suporte à utilização de memórias externas (*off-chip*), de diferentes tecnologias, SDRAM, SRAM, através de interfaces específicas. Em alguns modelos é possível inclusive operar a memória interna como um *cache* da memória externa [3].

2.3 Datapath

A Figura 2 (retirada de [2]) ilustra o *datapath* da arquitetura C67x e nas subseções seguintes serão descritos em maiores detalhes cada um dos componentes.

2.3.1 Bancos de Registradores

Cada um dos *datapaths* possui seu próprio banco de registradores e cada um dos dois bancos é composto de 16 registradores de 32-bits. É interessante notar que cada uma das quatro unidades de cada *datapath* possui sua própria porta de escrita no banco correspondente, isto significa que em um mesmo ciclo podem ocorrer um total de 8 escritas, 4 em cada um dos bancos, desde que sejam em registradores diferentes. Os registradores podem ser usados como dados, apontadores de endereço ou em testes de condição.

Existe o suporte em nível de registrador para tipos de dados menores que 32-bits, 8 e 16-bits, e também maiores, 40 e 64-bits. No primeiro caso um só registrador pode conter um pacote de dois dados de 16-bits ou quatro dados de 8-bits, e para cada caso existem variações de instruções que tratam cada dado separadamente. No segundo caso um par de registradores são usados para representar um dado de 40-bits (sendo desprezados 24 bits de um dos registradores) ou de 64-bits (neste último caso são necessários dois ciclos para a leitura e/ou escrita).

2.3.1.1 Caminhos de Cruzamento

De forma a permitir a leitura de algum registrador do banco do outro *datapath*, dois caminhos de cruzamento são fornecidos. É possível então a leitura de um, e somente um, registrador que se encontre no outro *datapath.*

2.3.2 Unidades Funcionais

As oito unidades funcionais podem ser divididas em dois grupos de quatro unidades: .L, .S, .M e .D. Cada um dos dois *datapaths* possui um destes grupos, e as unidades correspondentes entre eles são quase idênticas.

Cada unidade funcional possui duas portas de 32-bits para a leitura dos dados fontes (*src1* e *src2*). As unidades .L1, .S1, L2 e S2 possuem também 8-bits a mais para leitura e escrita, formando dados de 40-bits.

Cada um dos quatro tipos de unidades funcionais executa tipos diferentes de instruções, sendo que algumas instruções podem ser executadas por mais que uma unidade. Todas realizam algum tipo de operação tanto em ponto-fixo quanto em ponto-flutuante.

As unidades **.L** realizam operações de lógica, aritmética e de comparação em dados de ponto-fixo de 32-bits (também 40-bits no caso das aritméticas). Realizam também operações aritméticas em ponto-flutuante e operações de conversão de inteiro para ponto-flutuante e vice-versa.

As unidades **.S** se assemelham às unidades **.L** em relação às operações em ponto-fixo, porém são responsáveis também pelas instruções de desvios e por acessar os registradores de controle (no caso da unidade .S2).

As unidades **.M** realizam operações de multiplicação em ambas as representações numéricas.

Por último, as unidades **.D** realizam cálculos de endereços e por isso são responsáveis pelas instruções de *load* e *store*.

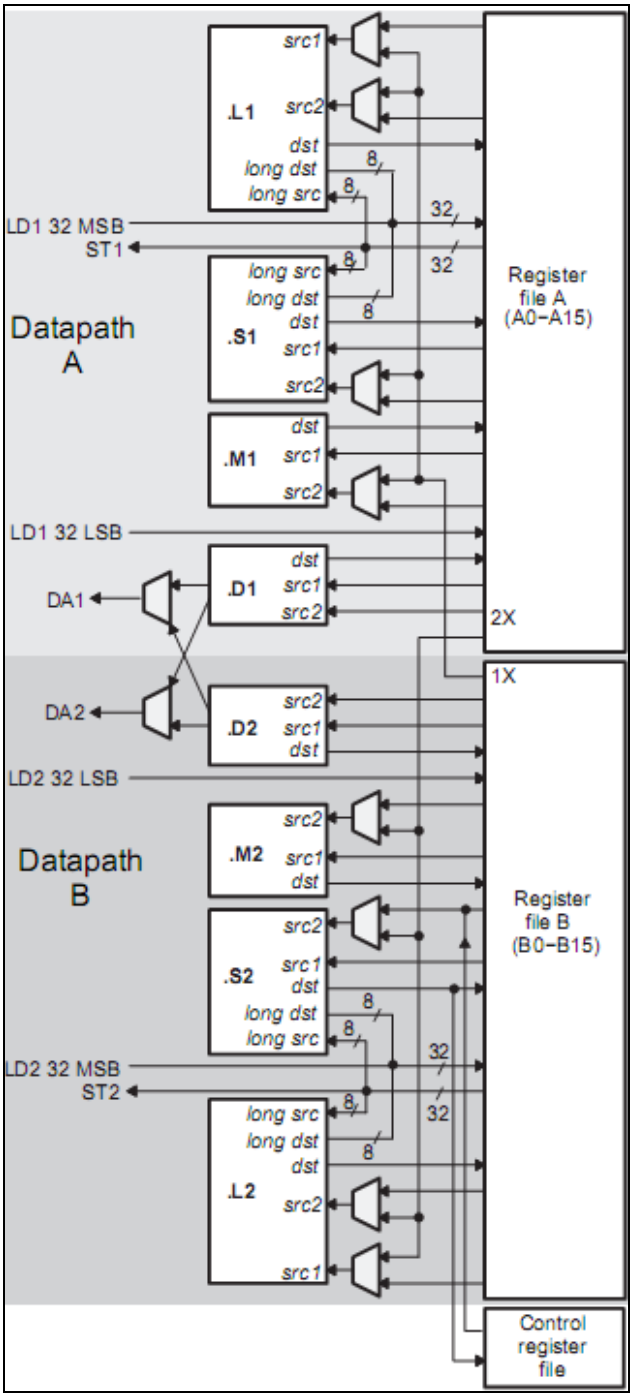


Figura 2 - Datapath do DSP TMS320C67x

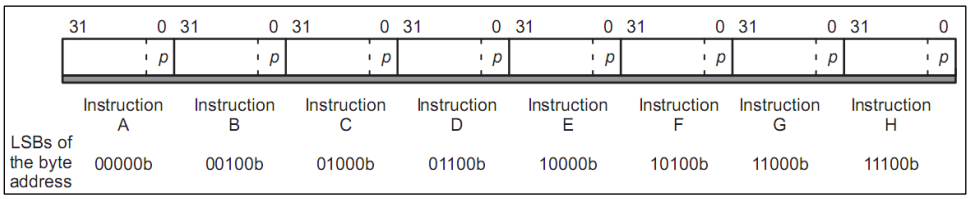


Figura 3 - Pacote de Busca (*Fetch Packet*)

2.3.3 Caminhos entre Memória e Registradores

Cada um dos *datapaths* possui dois caminhos de 32-bits para carregar valores da memória nos registradores, o que torna possível o carregamento em um único ciclo de um dado de 64-bits em um par de registradores, e um caminho de 32-bits para guardar valores dos registradores na memória.

2.3.4 Registradores de Controle

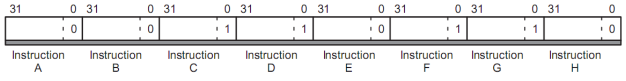
Os registradores de controle são compostos tanto de registros comuns de qualquer processador, tais como o Contador de Programa e tanto de registros que configuram algum modo específico de operação do C67x, por exemplo, se o modo de endereçamento é linear ou circular.

2.4 Pacote de Instruções

A cada busca um total de oito instruções de 32-bits são recebidas da memória de programa. A este pacote se dá o nome de **Pacote de Busca** (*Fetch Packet*) e a estrutura básica deste está mostrada na Figura 3 (retirada de [2]).

De forma a controlar quais instruções de um mesmo Pacote de Busca serão executadas em paralelo, cada instrução possui um "bit de paralelismo" (***p***) que indica se a próxima instrução (a do próximo endereço) pode ou não ser executada em conjunto.

A cada grupo de instruções de um mesmo Pacote de Busca que estão marcadas para serem executadas em paralelo dá-se o nome de **Pacote de Execução** (*Execute Packet*). Um só Pacote de Busca pode conter de um (totalmente paralelo) até oito (totalmente serial) Pacotes de Execução. A Figura 4 (retirada de [2]) exemplifica um Pacote de Busca que contém quatro Pacotes de Execução, cada um a ser executado em um ciclo diferente.



Cycle/Execute Packet	Instructions		
1	A		
2	B		
3	C	D	E
4	F	G	H

Figura 4 - Exemplo de um Pacote de Busca que contém quatro Pacotes de Execução

2.5 Pipeline

A Figura 5 (retirada de [2]) ilustra uma visão geral de todas as etapas do pipeline completo da arquitetura C67x. Como pode ser visto, este está dividido em três fases: Busca, Decodificação e Execução. Todas as instruções passam por todos os estágios de busca e decodificação, porém cada tipo de instrução precisa de um número diferente de estágios de execução.

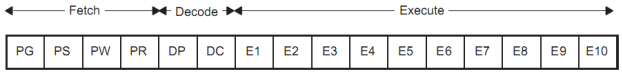


Figura 5 - Visão Geral do Pipeline Completo da Arquitetura C67x

2.5.1 Estágios de Busca

A fase de busca é dividida em quatro estágios:

- **PG:** é gerado o endereço de programa
- **PS:** o endereço de programa é enviado à memória
- **PW:** a leitura na memória de programa ocorre
- **PR:** o Pacote de Busca é recebido

É importante notar que todas as instruções de um mesmo Pacote de Busca passam por esses estágios em conjunto, visto que o Pacote de Busca é sempre lido por inteiro, sem importar a quantidade de Pacotes de Execução.

2.5.2 Estágios de Decodificação

A fase de decodificação é dividida em dois estágios:

- **DP:** despacho de instruções. É neste estágio que o Pacote de Busca é dividido em Pacotes de Execução. Cada instrução de um mesmo Pacote de Execução é então encaminhada para a unidade funcional apropriada.
- **DC:** neste estágio a informação sobre os registradores de destino e fonte e todos os caminhos associados é decodificada para então iniciar a fase de execução.

É importante notar que cada Pacote de Execução passa individualmente por estes estágios, por exemplo, se um dado Pacote de Busca conter três Pacotes de Execução, então ocorrerá uma fase de decodificação para cada um destes.

2.5.3 Estágios de Execução

Apesar de a fase de execução ser composta de 10 estágios, como mostrado na Figura 5, cada tipo de instrução precisa de um certo número destes estágios, por exemplo, instruções de um único ciclo somente precisam do estágio E1.

Para determinar o estado da CPU a cada ciclo, é preciso conhecer então por quantos estágios de execução cada tipo de instrução passa e o que ocorre em cada um destes estados. A descrição completa dos ciclos de execução de cada instrução pode ser encontrada em [2].

2.5.4 Exemplo de Operação do Pipeline

A Figura 6 (retirada de [2]) mostra o *datapath* completo e ilustra as fases e estágios do pipeline.

Na unidade de busca temos quatro Pacotes de Busca sendo processados, cada um em um dos estágios.

Na unidade de decodificação e no primeiro estágio, DP, temos um único Pacote de Execução visto que todas as instruções possuem uma atribuição para cada uma das oito unidades funcionais, representadas por setas. Já no estágio DC temos apenas uma unidade funcional a não ser utilizada, no caso a .D2, o restante possuem instruções que estão sendo decodificadas.

Na unidade de execução está representado apenas o uso das unidades funcionais para o estágio E1, sendo que neste estágio temos duas unidades funcionais sem instrução alguma, a .D2 e .S2.

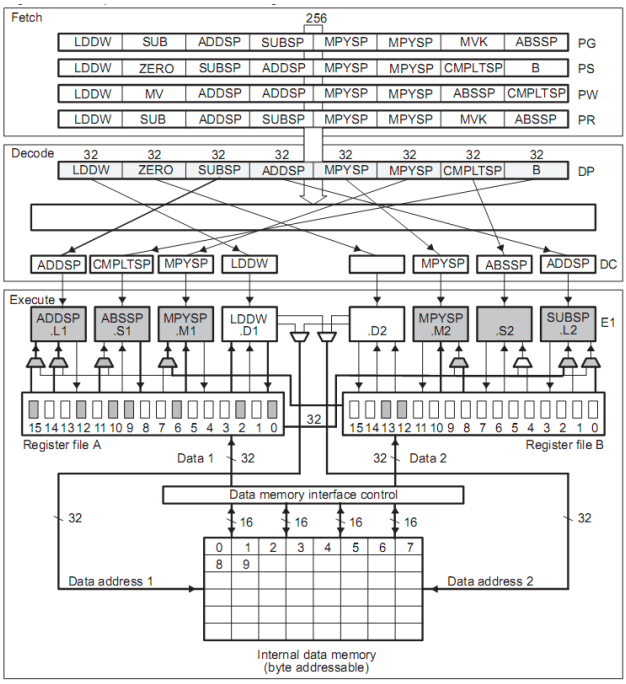


Figura 6 - Exemplo das Três Fases do Pipeline e seus Estágios

2.6 Considerações de Desempenho

2.6.1 Stalls

A Figura 7 (retirada de [2]) ilustra um conjunto ideal de Pacotes de Busca, onde todos possuem um único Pacote de Execução, ou seja, todas as instruções são executas em paralelo. O pipeline está sempre cheio e tem-se o desempenho máximo da arquitetura. Esta situação ideal raramente ocorrerá na prática

A Figura 8 (retirada de [2]) ilustra uma situação menos ideal, aonde agora o primeiro Pacote de Busca (n) possui três Pacotes de Execução (k, k+1 e k+2). Nota-se que agora temos um *stall* no pipeline causado pelo fato de que cada um dos Pacotes de Execução, k+1 e k+2, também precisam passar individualmente pela fase completa de decodificação e durante este tempo nenhuma nova busca de instrução pode ocorrer.

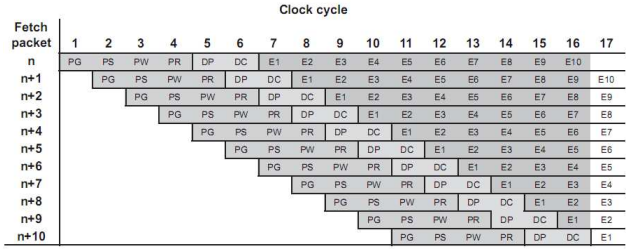


Figura 7 - Pipeline Ideal: sem stalls e todas as instruções sendo executadas em paralelo.

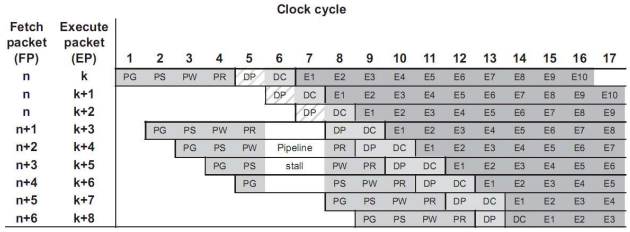


Figura 8 - Pipeline stalls devido a diferença de números de Pacotes de Execução

2.6.2 Penalidades de Desvios

Na arquitetura do C67x a instrução de desvio possui apenas um estágio de execução, o E1, e neste mesmo estágio já é computado o alvo do desvio e confirmado se a condição para desvio é satisfeita de forma que o primeiro estágio de busca, PG, já ocorre em paralelo, como pode ser visto na Figura 9 (retirada de [2]).

Nesta mesma figura pode ser visto também que a penalidade do desvio é de quatro ciclos (chamada em [2] de *delay slot*, e que significa apenas uma unidade de atraso) e esta é fixa para qualquer ocorrência de desvios que são tomados.

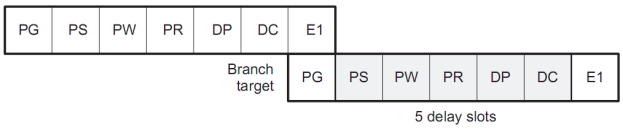


Figura 9 - Os estágios do pipeline para uma instrução de branch que foi tomada

3. DESENVOLVIMENTO DE CÓDIGO

É notável que a arquitetura em questão deixa a cargo do programador assembly e/ou compilador uma grande parte da responsabilidade pelo desempenho final da aplicação.

Por se tratar de uma arquitetura VLIW. Todo paralelismo deve ser explicitado diretamente no código assembly e/ou determinado a tempo de compilação. O trecho de código abaixo ilustra o assembly do C67x:

```
        LDDW    .D1     *A0--[4],B5:B4
||      ADDSP   .L1     A9,A10,A12
||      SUBSP   .L2X    B12,A2,B12
||      MPYSP   .M1X    A6,B13,A11
```

```
|| MPYSP    .M2     B5,B13,B11
|| ABSSP    .S1     A12,A15
```

Neste exemplo temos um Pacote de Execução com seis instruções. Nota-se que é necessário explicitar também a unidade funcional que a qual cada instrução está destinada.

Além de ter que explicitar o paralelismo, também é tarefa do programador/compilador fazer toda a checagem de dependência de dados; a arquitetura não realiza nenhum tipo de checagem dinâmica [2].

É importante notar que tais requerimentos sobrecarregam o programador de muitos detalhes da arquitetura, por exemplo, o programador deverá dominar bem quantos estágios de execução cada tipo de instrução usa a fim de garantir que a dependência de dados seja cumprida.

Dentre as ferramentas propostas para a plataforma C6000 em [4] tem-se: **Compilador C**, **Otimizador Assembly**, **Assembler**, **Linker** e **Ferramentas de Avaliação/*Debug*.**

Apesar de ser comum em aplicações DSP o uso direto do assembly [5], a proposta da Texas é que com o conjunto de ferramentas proposto o uso da linguagem C já irá atender à maioria das aplicações, restringindo o assembly apenas para otimizações em partes críticas da aplicação, se necessário.

De toda forma, a fim de facilitar a programação em assembly o Otimizador Assembly tem a função de abstrair detalhes demais da arquitetura ao programador, sendo possível até que o programador programe de forma linear e seqüencial que a ferramenta também extrai paralelismo, otimizando o código final [4].

4. PERIFÉRICOS

Uma parte importante que compõe a solução completa de um DSP são os periféricos *on-chip*. Diferentes modelos geralmente provêem diferentes periféricos *on-chip* de forma a atender diversas aplicações [6].

Dentre os periféricos comumente encontrados nos DSPs da plataforma C6000, incluindo as variações C67x, são: Controlador DMA, Host-Port Interface (porta de acesso a um processador *host*), EMIF (interface de memória externa, com suporte a diferentes tecnologias, SDRAM, SRAM, e outras), *Timers*, Gerenciador de Interrupções, Lógica de Power-Down, entre outros [4].

5. CONSIDERAÇÕES FINAIS

Foi apresentado a arquitetura do DSP TMS320C67x, que tem como base a arquitetura de toda a plataforma C6000. Através deste trabalho percebe-se claramente o *trade-off* das arquiteturas VLIW: melhor desempenho com hardware não tão complexo, porém com uma grande responsabilidade posta ao programador-assembly e/ou compilador, visto que a melhora no desempenho só virá com o uso de forma ótima dos recursos fornecidos pela arquitetura.

6. REFERÊNCIAS

[1] Internet – TI Embedded Processors Wiki: C674x/OMAPL1x Introductory Information: http://processors.wiki.ti.com/index.php/C674x/OMAPL1x_Introductory_Information (Último acesso: **13/junho/10**)

[2] Texas Instrument Technical Document: TMS320C67x/C67x+ DSP CPU and Instruction Set Reference Guide / Literature Number: SPRU733A / November 2006.

[3] Texas Instrument Technical Document: TMS320C6711, TMS320C6711B FLOATING-POINT DIGITAL SIGNAL PROCESSORS / Literature Number: SPRS088B / September 2001.

[4] Texas Instrument Technical Document: TMS320C6000 Technical Brief / Literature Number: SPRU197D / February 1999

[5] Choosing a DSP Processor, Berkeley Design Technology Inc. 2006. http://www.bdti.com/articles/choose_2000.pdf (Último acesso: **10/junho/10**)

[6] Internet – Texas Instrument DSP & ARM MPU Selection Tool: http://focus.ti.com/en/multimedia/flash/selection_tools/dsp/dsp.html (Último acesso: **13/junho/10**)