1. Publicação nº	2. Versão	3. Data	5. Distribuição
INPE-2992-TDL/153		*Jan., 1984*	☐ Interna ☒ Externa ☐ Restrita

4. Origem	Programa
DRH-DCA	*FRH/ECO*

6. Palavras chaves - selecionadas pelo(s) autor(es)

ARQUITETURAS ENCADEADAS (PIPE LINE)
PROCESSAMENTO EM TEMPO REAL

7. C.D.U.: *681.3.014*

8. Título *INPE-2992-TDL/153*

USO DE TÉCNICAS DE ENCADEAMENTO NA ARQUITETURA DE PROCESSADORES DIGITAIS DEDICADOS A APLICAÇÕES EM TEMPO REAL

10. Páginas: *137*

11. Última página: *B.8*

12. Revisada por

Eduardo W. Bergamini

13. Autorizada por

Nelson de Jesus Parada
Diretor Geral

9. Autoria *Almir Cavalcanti Lemos Filho*

Assinatura responsável

14. Resumo/Notas

É comum surgir a necessidade de submeter, em tempo real, um fluxo contínuo de dados digitais a um algoritmo que deve ser implementado com um processador digital síncrono. A medida que a vazão deste fluxo de dados é grande, alguma forma de paralelismo tem de ser inserida na arquitetura do processador, para que ele satisfaça esta demanda por processamento em alta velocidade. Este trabalho estuda o uso eficiente de arquiteturas endadeadas estaticamente configuradas, com uma alternativa interessante para a inserção de paralelismo na arquitetura de um processador digital. Uma metodologia de implementação de processadores com este tipo de arquitetura é desenvolvida. Esta metodologia impõe requisitos de viabilidade que influenciam no projeto inicial do processador, já que, se uma arquitetura encadeada satisfaz a esses requisitos, a metodologia, em uma fase posterior, é capaz de alterá-la de forma a fazer com que o algoritmo implementado possa ser executado a uma velocidade compatível com o fluxo de dados que chega ao processador. Um equalizador radiométrico de imagens para operar na taxa de 1 M"pixel"/s e um decodificador de Viterbi com 64 estados para operar na taxa de 32 kbits/s são usados como exemplos de aplicação da metodologia desenvolvida.

15. Observações

Dissertação de Mestrado em Eletrônica e Telecomunicações, aprovada em 22 de agosto de 1983.

Aprovada pela Banca Examinadora
em cumprimento a requisito exigido
para a obtenção do Título de Mestre
em Eletrônica e Telecomunicações

Dr.Gilberto F.Schleiniger

Presidente

Dr.Eduardo Whitaker Bergamini

Orientador

Dr.Wilson Ruggiero

Membro da Banca
-convidado-

Eng.Euclides Robert Filho, MSc.

Membro da Banca
-convidado-

Eng.Wilson Yamaguti, MSc.

Membro da Banca

Candidato: Almir Cavalcanti Lemos Filho

São José dos Campos, 22 de agosto de 1983

À minha esposa Berenice e à minha filha Isabel.

AGRADECIMENTOS

Aos orientadores, Dr. Eduardo W. Bergamini e Dr. Ricardo C.O. Martins, pela ajuda e orientação no curso deste trabalho.

Ao Dr. Gilberto F. Schleiniger e ao Eng. Wilson Yamaguti pelo incentivo e sugestões para elaboração do texto final.

ABSTRACT

It's common to arise the necessity to submit, in real time, a continuous digital data flow to an algorithm which has to be implemented with a synchronous digital processor. As this data flow increases, some kind of parallelism needs to be inserted in the processor architecture, as a way to satisfy this demand for high speed processing. This work studies the efficient use of statically configured pipelines as an interesting alternative to insert parallelism in a digital processor architecture. An implementation methodology of processors with this type of architecture is developed. This methodology imposes viability requirements that influence the initial stage of the processor design, since, if the proposed pipelined architecture satisfies these viability requirements, the methodology, in a later stage, is able to alter it in order to force the implemented algorithm to run in a compatible speed with the processor arriving data flow. To exemplify the application of the developed methodology an image radiometric equalizer to work with 1 Mpixel/s data rate and a 64 states Viterbi decoder to work with 32 kbits/s data rate are used.

SUMÁRIO

Pág.

LISTA DE FIGURAS

Pág.

LISTA DE TABELAS

CAPÍTULO 1

INTRODUÇÃO

É comum nas redes de comunicação de dados e na transmissão, recepção e manipulação de sinais e imagens de satélites, entre outras áreas de atuação do INPE, surgir a necessidade de submeter, em tempo real, um fluxo contínuo de dados digitais a um algoritmo que deve ser implementado com o auxílio de um processador digital síncrono (Figura 1.1).

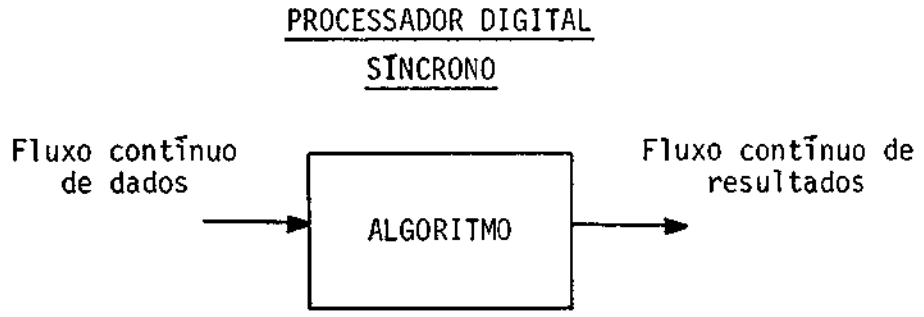


Fig. 1.1 - Processador digital dedicado a uma aplicação em tempo real (PDTR).

Por *fluxo contínuo de dados* entende-se que, em intervalos regulares de tempo, chega ao processador um conjunto de dados para ser manipulado pelo algoritmo implementado no processador a partir de seu passo inicial; e que a duração do fluxo de dados é suficientemente longa (ou indeterminada) para ser considerada infinita. Por sua vez, submeter em *tempo real* esse fluxo contínuo de chegada de dados ao algoritmo implementado no processador significa a existência (em regime), na saída do processador, de um fluxo também contínuo de resultados, na mesma taxa em que o algoritmo é iniciado no tempo. Ou seja, para cada conjunto de dados inicial do algoritmo que chega ao processador deve ser gerado, na sua saída, um conjunto de dados que é o resultado de uma execução completa do algoritmo, iniciada anteriormente.

Neste trabalho diz-se que a solução de um problema com essas características está a cargo de um *processador digital dedicado a uma aplicação em tempo real* - PDTR.

A necessidade de utilizar um PDTR pode surgir em várias situações. Uma delas, obviamente, é quando o problema em si exige o conhecimento dos resultados do algoritmo (após um atraso finito) a medida que seus dados iniciais vão chegando ao processador, como por exemplo na codificação e decodificação em canais de comunicação de voz. Uma outra situação é quando o volume de dados recebidos através do fluxo contínuo de dados, apesar de plenamente conhecido a priori, é muito grande e vai ser armazenado em uma memória de massa. Nestes casos é mais atrativo o armazenamento dos dados já manipulados pelo algoritmo, ao invés dos dados brutos recebidos. Como exemplo desse caso podem-se citar a decodificação, calibração e formatação de dados na recepção de imagens de satélites de observação da Terra.

Em uma arquitetura estritamente sequencial do PDTR, a medida que a vazão do fluxo contínuo de dados precisa ser maior, a obtenção de um processador compatível com essa vazão torna-se cada vez mais cara (não só em custo, mas também em peso, volume, energia consumida, etc.), quando não impossível. Para satisfazer essas demandas por processamento em alta velocidade requer-se então a adoção de alguma forma de paralelismo na arquitetura do PDTR.

A inserção de paralelismo em um PDTR pode ocorrer de duas maneiras básicas (Hayes, 1978). A primeira delas é com a replicação total dos recursos do processador em várias cópias distintas. Esse paralelismo, dito *paralelismo total*, apresenta um custo diretamente proporcional ao número de cópias do processador. Uma outra maneira, mais barata, é com o uso de *técnicas de encadeamento* (tradução do termo "pipeline" em inglês), em que o processador é desmembrado em uma série de segmentos de circuitos de propósito especial, capazes de operar concorrentemente e através dos quais os operandos fluem.

O uso de técnicas de encadeamento na arquitetura de um processador torna-se atrativo quando o processamento a ser efetuado envolve a execução de algoritmos (ou trechos de algoritmos) repetitivamente no tempo, bem quando a capacidade de processar continuamente, em tempo real, uma alta taxa de chegada de dados ao processador é mais importante do que o atraso em obter os resultados pertinentes (o que é o caso de um PDTR). Assim sendo, estas técnicas são utilizadas no processo de execução de instruções em Unidades Centrais de Processamento e na implementação de operações aritméticas complexas, como as que envolvem números em ponto flutuante. Exemplos desses dois tipos de aplicação são apresentados por Ramamoorthy e Li (1977).

Arquiteturas encadeadas apresentam-se como uma alternativa viável e barata para os casos em que processadores sequenciais baseados em microprocessadores, e até microprocessadores "bit-slice", revelam-se ineficazes na solução de certos problemas de processamento digital dedicado a uma aplicação em tempo real, principalmente quando, apesar da simplicidade dos algoritmos envolvidos, a vazão do fluxo contínuo de dados que deve chegar ao processador é alta.

O objetivo deste trabalho é o desenvolvimento de procedimentos de análise e aumento do desempenho que auxiliem na síntese de arquiteturas encadeadas, tendo em vista o seu uso eficiente no projeto de PDTRs que lidem com altas taxas de chegada de dados.

Inicialmente, no Capítulo 2, efetua-se uma caracterização mais precisa do funcionamento de um PDTR. A seguir discutem-se as técnicas básicas para inserção de paralelismo na arquitetura de um PDTR.

Um problema fundamental no uso de processadores com arquitetura encadeada, como será visto adiante neste trabalho, é a programação dos inícios das execuções do algoritmo implementado, de maneira a obter o maior número de execuções do algoritmo no tempo, evitando, porém, que diferentes execuções em paralelo desse algoritmo tentem utilizar ao mesmo tempo um mesmo recurso da arquitetura do processador, criando assim situações de conflito na alocação desses recursos. Objetivan

do atingir a solução desse problema, no Capítulo 3 introduzem-se as características e conceitos básicos de arquiteturas encadeadas e elaboram-se as ferramentas necessárias para a análise do seu funcionamento e uso eficiente. No Capítulo 4 são desenvolvidos procedimentos que levam ao *maior* número de execuções do algoritmo implementado, no tempo. E, no Capítulo 5, procede-se a um estudo para, através de alterações mínimas na arquitetura encadeada, fazer com que pelo menos um recurso do processador seja utilizado efetivamente 100% no tempo, quando se obtêm então procedimentos que levam ao número *máximo* de execuções do algoritmo implementado, no tempo. Todos esses procedimentos são apresentados sob a forma final de algoritmos, para deixar claros os métodos empregados.

Usando os procedimentos desenvolvidos nos Capítulos 4 e 5 a metodologia de implementação de PDTRs com arquiteturas encadeadas é abordada no Capítulo 6. Nesse mesmo capítulo, como exemplos, mostra-se o uso de técnicas de encadeamento na arquitetura de um processador para ser empregado na equalização dos "pixels" de imagens de satélites de observação da Terra, bem como, na arquitetura de um outro processador para ser usado como decodificador de Viterbi em canais de comunicação de voz.

CAPÍTULO 2

USO DE PARALELISMO EM UM PDTR

Neste capítulo discutem-se as duas maneiras básicas para a introdução de paralelismo na arquitetura de um PDTR: paralelismo total e paralelismo com uso de técnicas de encadeamento. Mas, antes, procede-se a uma caracterização mais precisa do funcionamento de um PDTR como base para esta discussão.

2.1 - CARACTERIZAÇÃO FUNCIONAL DE UM PDTR

A função primordial de um PDTR é efetuar uma transformação definida por um algoritmo, chamado *algoritmo implementado*, sobre os elementos de uma sequência (considerada infinita) de dados, para gerar os elementos de uma sequência de saída (Figura 2.1).

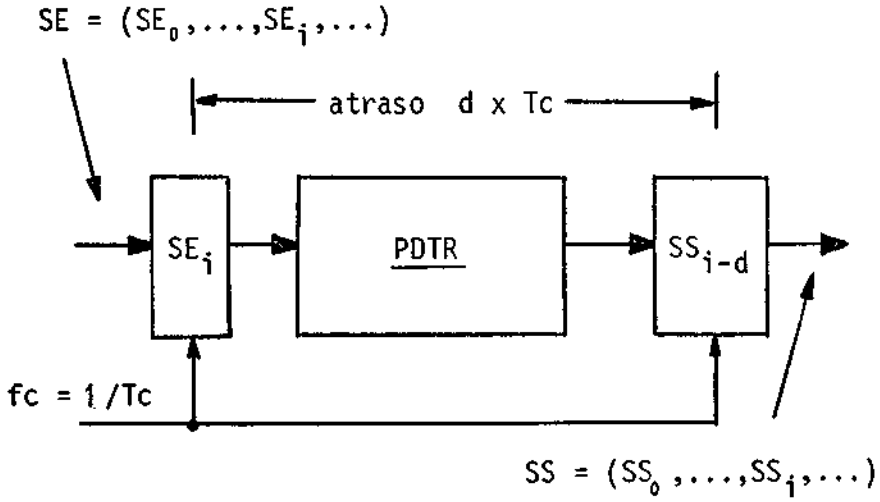


Fig. 2.1 - Sequências de dados de entrada e de saída de um PDTR.

Os elementos da sequência de entrada $SE = (SE_i)_{i=0}^{\infty}$ compõem-se dos valores assumidos por um número finito de variáveis de entrada definidas sobre domínios bem determinados e codificadas em uma

representação binária. Os elementos SE_i chegam à entrada do PDTR periodicamente no tempo e de forma ordenada. Para cada SE_i o algoritmo implementado no PDTR deve ser executado completamente desde o seu passo inicial até o seu passo final, gerando na saída do PDTR, como resultado, o elemento SS_i da sequência de saída $SS = (SS_i)_{i=0}^{\infty}$. SS_i é composto dos valores assumidos por um número finito de variáveis de saída, também definidas sobre domínios bem determinados e codificadas em binário.

Em um PDTR a capacidade de processamento dos elementos SE_i, na sua real taxa de chegada, é mais importante do que o tempo (finito) gasto na execução completa do algoritmo, para cada particular elemento da sequência de entrada SE. Ou seja, em regime, a chegada de um elemento SE_i deve sempre corresponder à colocação na saída do PDTR do elemento SS_{i-d}, com d fixo, inteiro positivo, finito, não necessariamente igual a um. Note-se que d representa o número de elementos da sequência de entrada processados ao mesmo tempo dentro do PDTR e que d x Tc é o atraso do sistema, onde Tc é o inverso da frequência fc de chegada dos elementos SE_i ao PDTR.

Sendo tm o tempo máximo gasto na execução completa do algoritmo implementado do PDTR, para quaisquer dados iniciais SE_i, então, necessariamente:

$$tm \leq d/fc. \qquad (2.1)$$

Na inequação acima, quando d > 1, o tempo máximo tm poderá ser maior do que o intervalo de tempo entre as chegadas dos elementos SE_i, e nessa situação haverá paralelismo no processamento dos elementos da sequência de entrada.

2.2 - INSERÇÃO DE PARALELISMO EM UM PDTR

Paralelismo em um PDTR pode ocorrer em vários níveis: no processamento simultâneo de vários elementos da sequência de entrada (quando d > 1); na execução de tarefas em paralelo, referentes ao pro

cessamento de um único elemento da sequência de entrada; ou até mesmo no mecanismo de busca e execução das instruções do PDTR. Algumas destas possibilidades de paralelismo estão intimamente ligadas ao algoritmo em si. Por exemplo, se o processamento a ser efetuado com o elemento SE_{i+1} depender dos resultados obtidos no processamento do elemento SE_i então o processamento em paralelo de elementos da sequência de entrada poderá não ser permitido.

Na discussão que se segue apenas a inserção de paralelismo, a nível de processamento simultâneo dos elementos da sequência de entrada do PDTR, será abordada. Adiante, neste trabalho, será evidenciado que as técnicas desenvolvidas para este nível de paralelismo podem ser aplicadas também aos outros níveis de paralelismo mencionados no parágrafo anterior.

Considere-se o seguinte problema hipotético. Seja fc a frequência de chegada dos elementos da sequência de entrada que devem ser submetidas, em tempo real, a um algoritmo A composto de n passos consecutivos. Considere-se também que um processador P, construído com uma certa tecnologia de componentes, seja capaz de executar o algoritmo A a uma velocidade tal que o tempo máximo te de processamento para cada elemento da sequência de entrada seja insatisfatório, com $te > 1/fc$.

Se a redução de te com a tecnologia de componentes utilizada é impraticável, a única alternativa para a solução do problema está na utilização de paralelismo no processamento dos elementos da sequência de entrada, considerando que esse nível de paralelismo é permitido pelo algoritmo A.

Como foi mencionado no Capítulo 1, existem duas maneiras básicas de introduzir paralelismo em um processador, e por conseguinte, em um PDTR. Uma primeira solução é o paralelismo total, apresentado na Figura 2.2, alcançado com a replicação dos recursos do processador P. Nela, determina-se o menor número inteiro d, tal que $te < d/fc$ e decompõe-se a sequência SE original em d sequências, como se segue:

sequência 1 - ..., SE_{i-d}, SE_i, SE_{i+d}, ...

sequência 2 - ..., SE_{i-d+1}, SE_{i+1}, SE_{i+d+1}, ...

.

.

.

sequência d - ..., SE_{i-1}, SE_{i+d-1}, SE_{i+2d-1}, ...

Cada uma dessas d sequências, assim obtidas, possui frequência de chegada fc/d e é entregue a um processador P a ela dedicado.

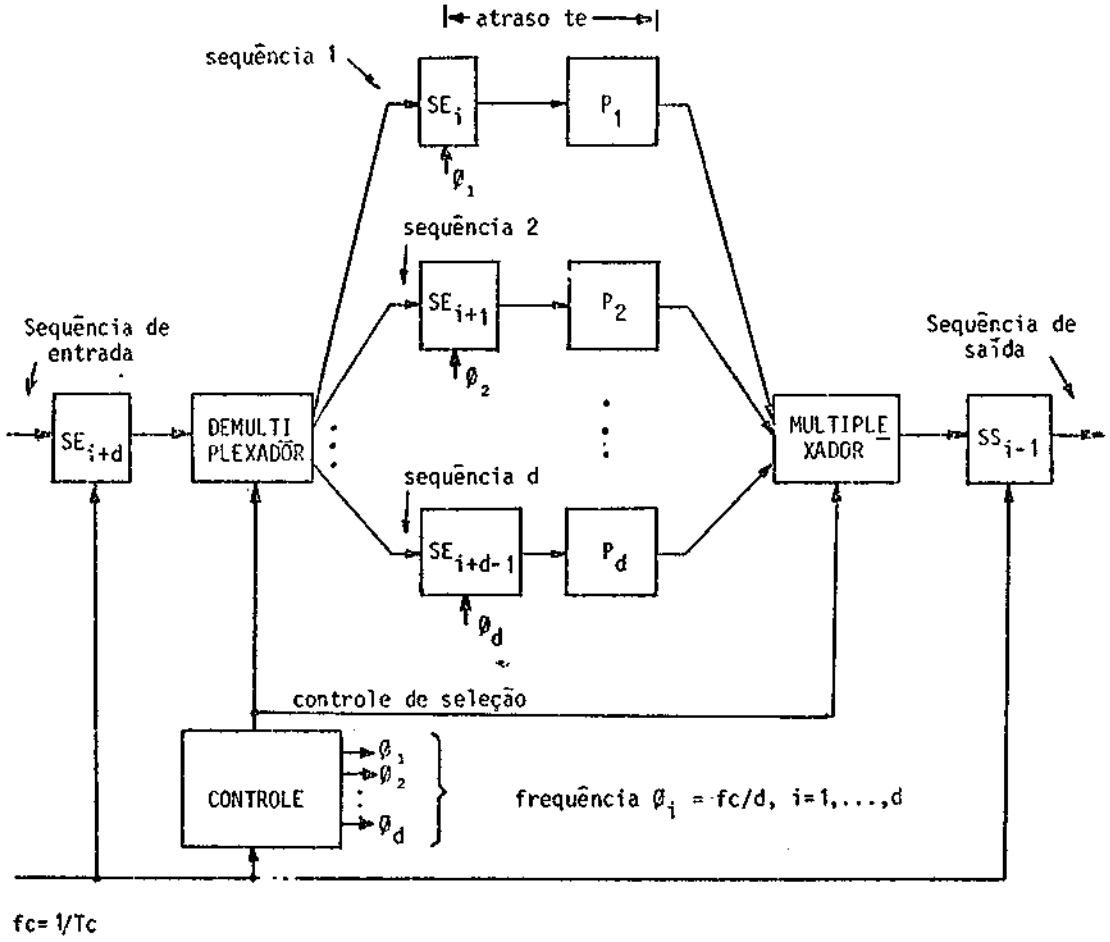


Fig. 2.2 - PDTR com paralelismo total.

Uma outra maneira de introduzir paralelismo em um PDTR é com o uso de técnicas de encadeamento. Agora, em vez de reproduzir totalmente os recursos do processador P, até se obter te < d/fc (arquitetura da Figura 2.2), partilha-se o algoritmo A em r *tarefas* (supostas consecutivas na Figura 2.3), estando cada tarefa sob a responsabilidade de um módulo especialmente dedicado que opera concorrentemente com os outros. Cada módulo executa o seu trecho do algoritmo A, com os dados iniciais existentes na sua entrada, e transfere os resultados para o módulo seguinte, sincronamente. Na entrada de cada módulo existe um elemento armazenador para os dados iniciais referentes à execução da tarefa pelo módulo. Um PDTR com essa arquitetura passa a funcionar analogamente a uma linha de montagem industrial. Sendo max (te_i) o tempo máximo de execução da tarefa que leva mais tempo para ser realizada, é necessário então que max (te_i) < 1/fc em vez de $\sum_{i=1}^{r} te_i < 1/fc$.

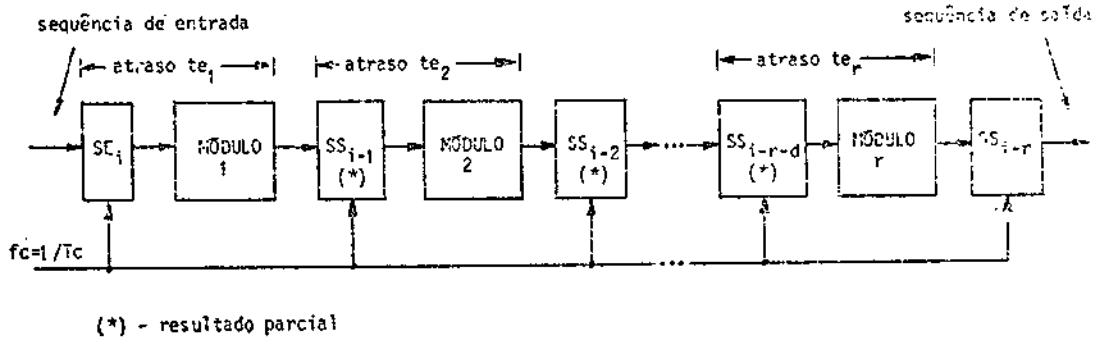


Fig. 2.3 - PDTR com arquitetura encadeada.

A grande vantagem da arquitetura encadeada sobre a arquitetura com paralelismo total é o tamanho do circuito do PDTR que tende a ser menor: como cada módulo é especializado na execução de apenas um trecho do algoritmo A (e não o algoritmo completo), é razoável supor que, além de executar a sua tarefa específica mais rapidamente do que o processador P, ele deva apresentar um circuito menor do que P. E uma redução no tamanho de um circuito acarreta também redução no seu cus-

to, peso, volume, consumo de energia, etc, tornando, portanto, atrativo o uso de técnicas de encadeamento no projeto de PDTRs que necessitem trabalhar com altas taxas de chegada de dados.

CAPÍTULO 3

CARACTERÍSTICAS E CONCEITOS BÁSICOS DE ARQUITETURAS ENCADEADAS

A *arquitetura encadeada linear* da Figura 2.3 é muito simples e de fácil análise. Cada módulo de circuito, doravante chamado *segmento*, executa a sua parte do algoritmo dentro de um tempo corrido, constante e fixo de Tc segundos, com todos os segmentos trabalhando de forma sincronizada. O tempo mínimo entre sucessivos inícios de execução do algoritmo implementado, chamado *latência*, é de Tc segundos, enquanto o tempo gasto para executar completamente o algoritmo implementado, do seu passo inicial até o seu passo final, chamado *atraso* do processador, é de r x Tc segundos.

Apesar de sua simplicidade, a arquitetura da Figura 2.3 é uma solução válida, em casos igualmente simples de processamento digital dedicado a uma aplicação em tempo real. Entretanto, a análise de casos mais complexos requer um modelamento mais apropriado do funcionamento de arquiteturas encadeadas. Estes casos mais complexos surgem quando um segmento é utilizado na execução de mais de uma tarefa (não-consecutivas) do algoritmo implementado, ou até mesmo quando são alocados tempos diferentes para os segmentos executarem suas tarefas. Como exemplos desses casos mais complexos podem-se citar: tarefas idênticas, localizadas em pontos distintos do algoritmo, sendo realizadas por um mesmo segmento; ou um segmento composto de um bloco de memória, cujo conteúdo é lido e modificado em vários pontos do algoritmo implementado.

Um grande problema emerge nesses casos mais complexos de arquiteturas encadeadas, qual seja o de obter, no tempo, o maior número de execuções do algoritmo implementado, mas evitando que execuções em paralelo desse algoritmo necessitem utilizar, ao mesmo tempo, um mesmo segmento do processador.

Tendo em vista atingir a solução deste problema, este capítulo tratará das características e conceitos básicos de arquiteturas encadeadas, necessários ao desenvolvimento posterior deste trabalho. As Seções 3.2, 3.3, 3.4 e 3.5 baseiam-se parcialmente em Davidson (1971), citado por Shar (1972), e Ramamoorthy e Li (1977). Os Capítulos 4 e 5 são dedicados ao desenvolvimento de procedimentos que permitirão a análise e uso eficiente de arquiteturas encadeadas durante a síntese de PDTRs com esse tipo de arquitetura.

3.1 - GRAFO DE FLUXO E TABELA DE TEMPOS DE EXECUÇÃO

O fluxo de operandos pelos segmentos de uma arquitetura encadeada (Figura 3.1a) pode ser caracterizado por um grafo de fluxo e uma tabela de tempos de execução.

O *grafo de fluxo* (Figura 3.1b) exprime a ordem de utilização dos segmentos a medida que os operandos fluem pelos vários segmentos da arquitetura encadeada. Os nós do grafo são os segmentos utilizados e os arcos orientados representam a passagem de operandos de um segmento para outro. Um segmento possui associado a ele tantos nós quantas vezes é utilizado pelo algoritmo implementado. No grafo de fluxo, se um arco vai do nó Na para o nó Nb então os resultados gerados pelo segmento associado ao nó Na são transferidos imediatamente após terem sido obtidos, para o segmento associado ao nó Nb. Nessa situação diz-se que o nó Nb é um nó *sucessor* do nó Na no grafo de fluxo. Em um grafo de fluxo não existe nenhuma trajetória que passe mais de uma vez por algum nó do grafo, o que significa que não existem ciclos ("loops") no grafo de fluxo.

Seja T uma *unidade de tempo* suficientemente pequena para que o tempo requerido por cada segmento na realização da sua tarefa possa ser expresso como um múltiplo inteiro de T. Na *tabela de tempos de execução* (Figura 3.1c) são então listados os múltiplos inteiros da unidade de tempo T que cada segmento da arquitetura encadeada gasta para realizar a sua tarefa, desde que todos os operandos necessários já estejam disponíveis.

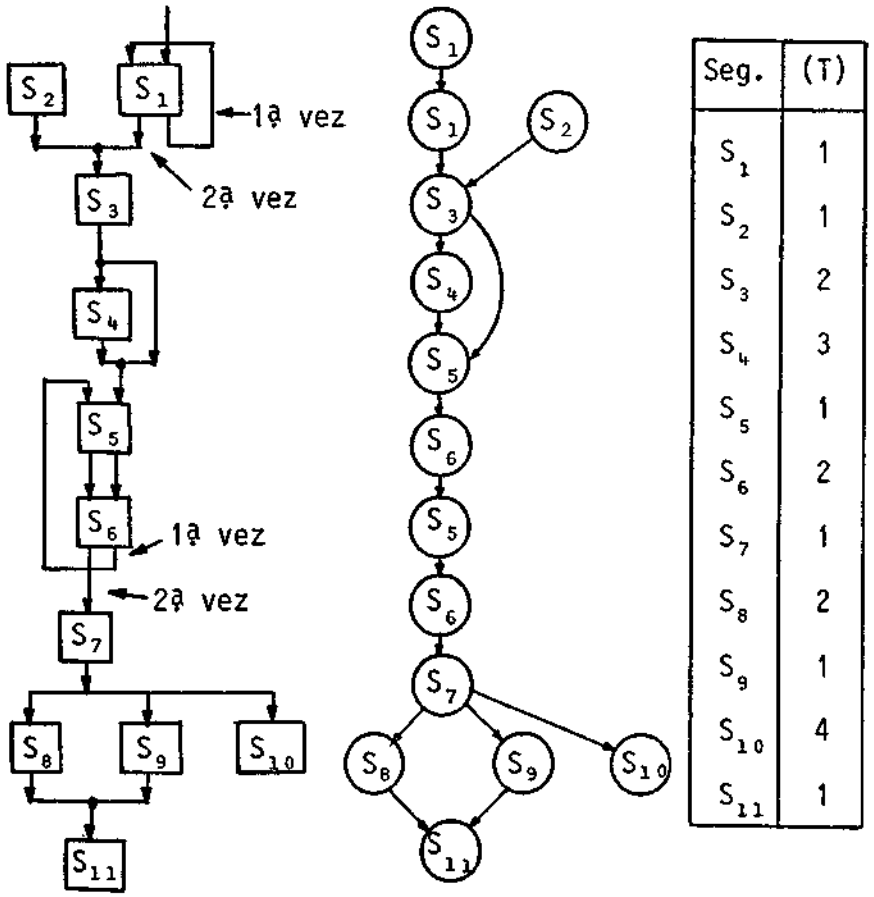


Fig. 3.1 - Arquitetura encadeada estaticamente configurada: a) fluxo de operandos pelos segmentos, b) grafo de fluxo, c) tabela de tempos de execução.

Diz-se que uma arquitetura encadeada, representada pelo seu grafo de fluxo e tabela de tempos de execução, é uma arquitetura encadeada *válida* se e somente se o fluxo de operandos, em uma única execução do algoritmo implementado, não leva nenhum segmento da arquitetura encadeada a ser utilizado ao mesmo tempo na execução de mais de uma tarefa. Se isso ocorrer diz-se que a arquitetura encadeada é *não-válida*. Neste trabalho, sempre que forem referenciadas arquiteturas encadeadas fica implícito que elas são válidas. Entretanto, vale ressaltar que arquiteturas encadeadas não-válidas podem ser transformadas em arquiteturas encadeadas válidas através de artifícios tais como:

a) passagem fictícia de operandos entre segmentos,

b) inserção de segmentos atrasadores (Capítulo 5).

Apesar de T = Tc na arquitetura da Figura 2.3, esta igualdade nem sempre prevalece no caso geral de PDTRs com arquiteturas encadeadas. Isso será facilmente verificado mais adiante nesta capítulo.

3.2 - TABELA DE RESERVA DE SEGMENTOS

Baseado no grafo de fluxo e na tabela de tempos de execução, a utilização dos segmentos de uma arquitetura encadeada pode ser representada por uma *tabela de reserva de segmentos*. Essa tabela mostra, exatamente, em quais intervalos de tempo T, após o processamento ter-se iniciado, cada segmento da arquitetura é utilizado pelo algoritmo implementado no processador. Na tabela de reserva de segmentos, os segmentos são listados na vertical, enquanto os intervalos de tempo T, numerados sequencialmente, são listados na horizontal. Um X é colocado na interseção da linha do segmento com as colunas correspondentes aos intervalos de tempo, nos quais, após a execução do algoritmo ter-se iniciado, o particular segmento é utilizado.

Na confecção da tabela de reserva de segmentos o *tempo de utilização* de um segmento é formado pela soma de duas parcelas. Uma delas é o próprio *tempo de execução*, obtido da tabela com o mesmo nome descrita na seção anterior. A outra parcela é o *tempo de espera ocupada* que, eventualmente, pode existir entre a chegada do primeiro operando ao segmento e o instante em que todos os operandos ficam disponíveis para o segmento efetivamente inciar a execução da sua tarefa.

A utilização de um segmento em espera ocupada aparece porque, em uma arquitetura encadeada, sempre que um segmento termina de executar a sua tarefa, ele transfere os resultados obtidos imediatamente para os segmentos a que esses resultados se destinam, o que pode causar a chegada, em instantes de tempo diferentes, dos operandos necessários para um segmento iniciar a execução da sua tarefa. Neste caso, uma vez recebido o primeiro dos operandos, o segmento já fica comprometido em aguardar a chegada dos operandos restantes, supondo-se que ainda falta chegar algum outro operando.

Em uma arquitetura encadeada não existe uma unidade de tempo em que todos os segmentos da arquitetura fiquem inoperantes. Também não existe um segmento que não seja utilizado em nenhum instante de tempo pelo algoritmo implementado. Portanto, em uma tabela de reserva de segmentos existe pelo menos um X em cada coluna e pelo menos um X em cada linha da tabela. E, caso haja mais de um X em uma mesma coluna da tabela então os segmentos correspondentes estarão sendo utilizados em paralelo.

O tempo total gasto em uma execução completa do algoritmo implementado, ou seja o atraso do processador, corresponde ao número de colunas (expresso em unidades de tempo T) da tabela de reserva de segmentos.

Na Tabela 3.1 é mostrado um exemplo de tabela de reserva de segmentos, referente ao fluxo de operandos pelos segmentos da arquitetura encadeada da Figura 3.1. Observe-se nesse exemplo que, apesar de

o tempo de execução do segmento S_5 ser de uma unidade de tempo, a primeira vez que ele é utilizado gastam-se três unidades de tempo em espera ocupada. Espera ocupada também ocorre na utilização dos segmentos S_3 e S_{11}. Observe-se também que na Tabela 3.1 não dá para distinguir que o segmento S_1 foi utilizado duas vezes consecutivas e o segmento S_8 apenas uma vez.

TABELA 3.1

TABELA DE RESERVA DE SEGMENTOS DA ARQUITETURA ENCADEADA DA FIGURA 3.1

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}	T_{16}	T_{17}	T_{18}
S_1	X	X																
S_2	X																	
S_3		X	X	X														
S_4					X	X	X											
S_5					X	X	X	X			X							
S_6									X	X		X	X					
S_7														X				
S_8															X	X		
S_9															X			
S_{10}															X	X	X	X
S_{11}																X	X	

Em um PDTR, o algoritmo implementado é fixo e previamente definido, implicando que, uma vez decidida a melhor política de distribuição e uso dos segmentos, a tabela de reserva de segmentos decorrente fique inalterada, o que caracteriza uma arquitetura encadeada *estaticamente configurada*. Observando apenas a tabela de reserva de segmentos de uma arquitetura encadeada estaticamente configurada é possível estudar o que ocorre quando várias execuções do algoritmo nela implementado se dão ao mesmo tempo, ou seja, em paralelo.

Uma arquitetura encadeada, caracterizada por um grafo de fluxo e uma tabela de tempos de execução, determina uma única tabela de reserva de segmentos. Entretanto, o oposto não é verdadeiro: uma tabela de reserva de segmentos pode ser gerada a partir de diferentes grafos de fluxo e tabelas de tempo de execução, o que é facilmente verificável porque em uma tabela de reserva de segmentos:

a) não existe distinção entre tempo gasto em espera ocupada e tempo gasto efetivamente na execução de uma tarefa por um segmento,

b) não é possível saber se vários X juntos em uma linha da tabela correspondem a uma única ou a várias utilizações consecutivas de um mesmo segmento pelo algoritmo implementado.

Portanto, o mapeamento existente entre o conjunto de grafos de fluxo e o conjunto de tabelas de tempos de execução, no conjunto de tabelas de reserva de segmentos, não é uma função biunívoca. Devido a isso, apesar de a tabela de reserva de segmentos ser uma ferramenta de muita utilidade na análise do funcionamento de uma arquitetura encadeada estaticamente configurada, quando várias execuções do algoritmo implementado se dão ao mesmo tempo, o estudo apenas da tabela de reserva de segmentos não é suficiente para esgotar a análise de uma arquitetura encadeada particular, representada por seu grafo de fluxo e sua tabela de tempos de execução.

3.3 - VETOR DE COLISÃO

Em uma arquitetura encadeada deve-se evitar que execuções em paralelo do algoritmo implementado necessitem usar simultaneamente um mesmo segmento. A essa situação dá-se o nome de *colisão*. Como então programar um novo início de execução do algoritmo implementado sem causar alguma colisão com execuções deste mesmo algoritmo já iniciados anteriormente?

Considere-se o caso em que a execução do algoritmo acabou de começar. Para determinar em que tempos futuros a execução do algoritmo pode novamente iniciar-se, sem causar colisão, basta consultar a tabela de reserva de segmentos. Uma maneira de verificar se o início de uma segunda execução do algoritmo pode dar-se ℓ unidades de tempo depois do início de uma primeira execução é sobrepondo uma cópia da tabela de reserva de segmentos sobre ela mesma deslocada de ℓ unidades de tempo. Se um X de uma tabela coincidir com um X da outra tabela, então uma colisão irá ocorrer naquele segmento e ℓ será uma *latência proibida*. Caso contrário, se nenhuma colisão é detetada, então ℓ será uma *latência permitida*. Portanto, verificando as distâncias (medidas em unidades de tempo T) entre todos os possíveis pares de X de cada uma das linhas, pode-se construir uma lista com todas as latências proibidas para uma tabela de reserva de segmentos particular. A essa lista dá-se o nome de *lista de latências proibidas*. Na Tabela 3.2 a lista de latências proibidas é: {1, 2, 5}.

TABELA 3.2

EXEMPLO DE TABELA DE RESERVA DE SEGMENTOS

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}
S_1	X										
S_2		X					X				
S_3			X					X			
S_4				X	X	X					
S_5									X		
S_6										X	
S_7										X	
S_8											X

De posse da lista de latências proibidas (na qual n é a latência maior) é possível construir o *vetor de colisão* C, que é um vetor binário de n bits (bit 1 mais a direita e bit n mais a esquerda)

com o seguinte significado: $bit_{\ell} = 1$ se e somente se ℓ é um número presente na lista de latências proibidas. Portanto, se um vetor de colisão é:

$$C = c_n\ c_{n-1}\ c_{n-2} \ldots c_2\ c_1,$$

$$\text{então } c_{\ell} = \begin{cases} 1, \text{ se } \ell \text{ é um elemento da lista de latências proibidas;} \\ 0, \text{ caso contrário.} \end{cases}$$

Por definição, note-se que $c_n = 1$ em qualquer vetor de colisão. Com isso, o exemplo simples da Figura 2.3 é um caso degenerado com n = 0 e vetor de colisão vazio. No exemplo da Tabela 3.2, o vetor de colisão é 10011.

É importante observar que uma tabela de reserva de segmentos determina um único vetor de colisão, mas o oposto não é verdadeiro. Dado um vetor de colisão, ele pode ser obtido a partir de infinitas tabelas de reserva de segmentos (veja-se o exemplo na Tabela 3.3), ou seja, o mapeamento entre o conjunto de tabelas de reserva de segmentos e o conjunto dos vetores de colisão também não é uma função biunívoca.

TABELA 3.3

EXEMPLO DE TABELAS DE RESERVA DE SEGMENTOS QUE DETERMINAM O MESMO VETOR DE COLISÃO: 10011

Seg.	T_1	T_2	T_3	T_4	T_5	T_6
S_1	X	X				
S_2			X			
S_3				X	X	
S_4	X		X			
S_5	X					X

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
S_1	X					X			
S_2		X	X						
S_3				X		X			
S_4					X		X		
S_5								X	
S_6									X

Seg.	T_1	T_2	T_3	T_4	T_5	T_6
S_1	X					X
S_2		X	X			
S_3				X		X
S_4					X	

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
S_1	X							
S_2		X					X	
S_3			X	X				
S_4					X			
S_5						X		X

3.4 - GRAFO DE LATÊNCIAS PERMITIDAS

Antes de iniciar uma nova execução do algoritmo implementado em uma arquitetura encadeada, é necessário prevenir a ocorrência de colisão com as execuções do algoritmo já em andamento. Isto significa verificar se nenhum dos tempos decorridos, medidos em unidades de tempo T, desde o início dessas execuções já em andamento não faz parte da lista de latências permitidas da arquitetura encadeada em estudo.

Essa verificação pode ser feita dinamicamente no tempo em cada unidade de tempo T. Para isto, basta memorizar a lista de todas as latências proibidas *medidas a partir do instante considerado*, e referentes às execuções em andamento do algoritmo implementado no processador.

Essa lista recebe o nome de *lista de latências proibidas futuras*. Por exemplo, considere-se o caso de uma tabela de reserva de segmentos com lista de latências proibidas {1, 4, 6} e que, inicialmente, não haja nenhuma execução do algoritmo em andamento no processador. No instante em que for iniciada a primeira execução do algoritmo, a lista de latências proibidas futuras será {1, 4, 6} que é a própria lista de latências proibidas da arquitetura encadeada. Se após duas unidades de tempo tem início a segunda execução do algoritmo, nesse novo instante de tempo a lista de latências proibidas será {1, 2, 4, 6} que resulta das latências proibidas futuras da segunda execução do algoritmo {1, 4, 6}, acrescida das latências proibidas futuras ainda remanescentes da primeira execução do algoritmo: 2 (=4-2) e 4 (=6-2). Agora, se passadas mais três unidades de tempo, tiver início a terceira execução do algoritmo, nesse novo instante de tempo a lista de latências proibidas futuras será {1, 3, 4, 6} que advêm das latências proibidas futuras desta terceira execução do algoritmo {1, 4, 6}, acrescida das latências proibidas futuras remanescentes da segunda execução do algoritmo 1 (=4-3) e 3 (=6-3), mais a latência futura ainda remanescente da primeira execução do algoritmo: 1 (=4-3). E assim, pode-se prosseguir indefinidamente no tempo.

Esse procedimento de prevenção dinâmica de colisão pode ser melhor visualizado com a utilização de um grafo, chamado *grafo de latências permitidas*, onde estão representadas todas as latências permitidas em todas as situações possíveis (veja-se a Figura 3.2). Os nós do grafo são as listas de latências proibidas futuras, sendo o *nó inicial* a própria lista de latências proibidas da arquitetura encadeada, (que é a lista de latências proibidas futuras, após o início da primeira execução do algoritmo). Cada nó possui, para cada latência permitida possível, um arco orientado saindo dele e rotulado com esta latência permitida. Um arco rotulado ℓ, deixando o nó Na, leva ao nó Nb que é uma lista de latências proibidas futuras construída com os elementos do nó inicial do grafo, acrescida de todos os elementos da lista do nó Na subtraídos de ℓ tomando-se apenas os resultados positivos. De to

dos os nós do grafo saem um arco rotulado $(n + 1)^+$ (onde n é o maior elemento da lista de latências proibidas), que vai para o nó inicial do grafo. Isto indica que, se mais de n unidades de tempo passarem desde o último início de execução do algoritmo, não mais existirá nenhuma influência das execuções do algoritmo anteriormente iniciadas, na lista de latências proibidas futuras.

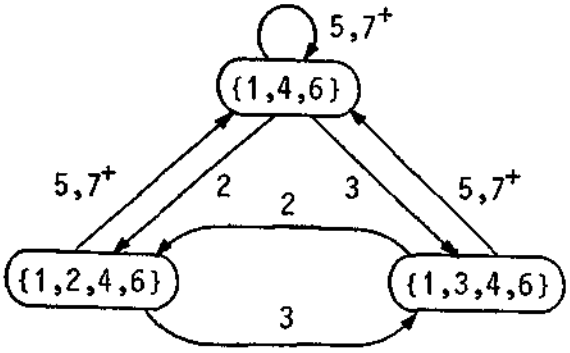


Fig. 3.2 - Exemplo de grafo de latências permitidas para o caso em que a lista de latências proibidas do problema é {1, 4, 6}.

Usando a notação de vetor de colisão introduzida na seção anterior também para as listas de latências proibidas futuras dos nós do grafo da Figura 3.2 a construção do grafo torna-se mais fácil (veja-se a Figura 3.3). Observe-se na Figura 3.3 que, se o arco orientado rotulado de ℓ vai do nó Na para o nó Nb, então o vetor de colisão do nó Nb será obtido através de um OU-lógico, bit a bit, entre o vetor de colisão do nó inicial com o vetor de colisão do nó Na deslocado de ℓ bits para a direita. Uma outra diferença entre o grafo da Figura 3.2 e o da Figura 3.3 é que neste último os arcos rotulados $(n + 1)^+$ são omitidos e representados apenas pelo fim de arco $(n + 1)^+$ entrando no nó inicial ficando subentendido que de todo nó do grafo sai um arco $(n + 1)^+$ que se destina ao nó inicial.

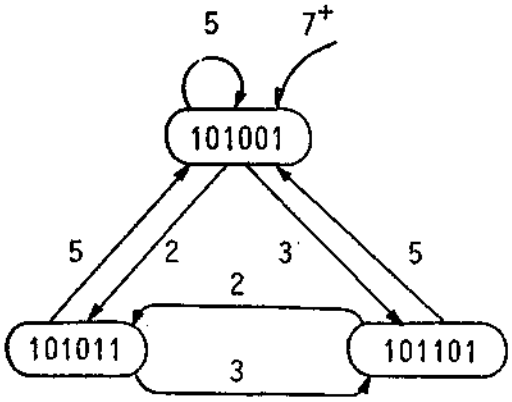


Fig. 3.3 - Exemplo de grafo de latências permitidas para o vetor de colisão 101001.

O grafo de latências permitidas, construído com a notação de vetor de colisão (como na Figura 3.3) é mais conhecido na literatura corrente sobre arquiteturas encadeadas (vejam-se referências bibliográficas deste trabalho) como diagrama de estado ou diagrama de estado modificado. Apesar de o grafo ser exatamente o mesmo, a diferença de nome surge apenas devido a maneira usada neste trabalho para construir o grafo, que é distinta da empregada nessas referências. O nome diagrama de estados advém do fato de o grafo de latências permitidas coincidir, nessas referências, com o diagrama de estados atingíveis de um controlador para arquiteturas encadeadas proposto por Davidson (1971) e construído com um registro de deslocamento.

O grafo de latências permitidas pode também ser apresentado na forma de uma *tabela de conexão*. Nesta tabela existe uma linha e uma coluna para cada nó do grafo de latências permitidas, e na intersecção da linha i com a coluna j são colocadas as latências dos arcos (normalmente apenas uma) que vão do nó i para o nó j. Na Tabela 3.4 é

mostrada a tabela de conexão do grafo da Figura 3.3. A tabela de conexão é muito útil para representar grafos de latências permitidas com grande número de nós ou que necessitem ser submetidos a algum processamento por computador.

TABELA 3.4

TABELA DE CONEXÃO PARA O GRAFO DA FIGURA 3.3

$Nó_i$ \ $Nó_j$	101001	101011	101101
101001	7^+,5	2	3
101011	7^+,5	-	3
101101	7^+,5	2	-

3.5 - CICLO ÓTIMO

Percorrendo os nós de um grafo de latências permititas (sempre a partir do nó inicial que representa a lista de latências proibidas futuras após o início da primeira execução do algoritmo implementado) com uma trajetória que passe pelos nós do grafo obtêm-se as latências que dão os instantes de tempo em que o algoritmo implementado no processador pode ter sua execução iniciada, livre do problema da colisão. Como o algoritmo implementado em um PDTR deve ser executado infinitas vezes e como o grafo de latências permitidas é um grafo fechado que possui um número finito de nós, uma trajetória infinita compor-se-á do percorrimento sucessivo de vários ciclos desse grafo.

Um *ciclo* no grafo de latências permitidas é completamente especificado pelos nós por onde passa e pelas latências dos arcos que vão de um nó a outro sequencialmente no ciclo. Na Figura 3.3 entre outros pode-se detetar o ciclo (101001, 101101) com latências (3, 5).

Cada ciclo tem uma *latência média*, que é a média aritmética das latências dos arcos que o constituem, ou seja, se as latências de um ciclo são $\ell_1, \ell_2, \ldots, \ell_m$, então a sua latência média LM será:

$$LM = \frac{\sum_{i=1}^{m} \ell_i}{m} \qquad (3.1)$$

O ciclo da Figura 3.3 mencionado anteriormente nesta seção apresenta latência média LM = 4.

Se as latências dos arcos que constituem um ciclo são todas iguais a ℓ, então diz-se que este ciclo apresenta *latência constante* ℓ. Na Figura 3.3, o ciclo em torno do nó inicial (101001) é um ciclo com latência constante igual a 5.

Diz-se que um ciclo que tem a menor latência média de todos os ciclos de um grafo de latências permitidas possui a *latência média mínima* (LMmin) e é chamado *ciclo ótimo*. Na Figura 3.3 o ciclo (101011, 101101) com latências (3, 2) é o ciclo ótimo e, neste caso, LMmin = 5/2.

Em um grafo de latências permitidas é possível mais de um ciclo possuir latência média igual a latência média mínima dos ciclos do grafo, ou seja, o ciclo ótimo pode não ser único. Isso é ilustrado com o grafo da Figura 3.4. Neste grafo, os seguintes ciclos possuem LM = LMmin = 2: (1000, 1100, 1110, 1111) com latências (1, 1, 1, 5) e (1001, 1010, 1101, 1011) com latências (2, 1, 2, 3).

Em um processador com arquitetura encadeada, se os inícios das execuções do algoritmo implementado forem programados para ocorrer de acordo com as latências de um ciclo ótimo, então será alcançado, no tempo, o maior número (médio) possível de execuções do algoritmo, sem que ocorra colisão. Nessa situação, supondo a unidade de

tempo T satisfatória, o inverso da frequência de chegada dos elementos da sequência de entrada do PDTR (Tc) será igual a LMmin unidades de tempo T, ou seja: TC = LMmin x T. Ressalte-se que se o processador apresentar uma unidade de tempo T' < T, o seu desempenho pode ser piorado de forma a se ter T' = T.

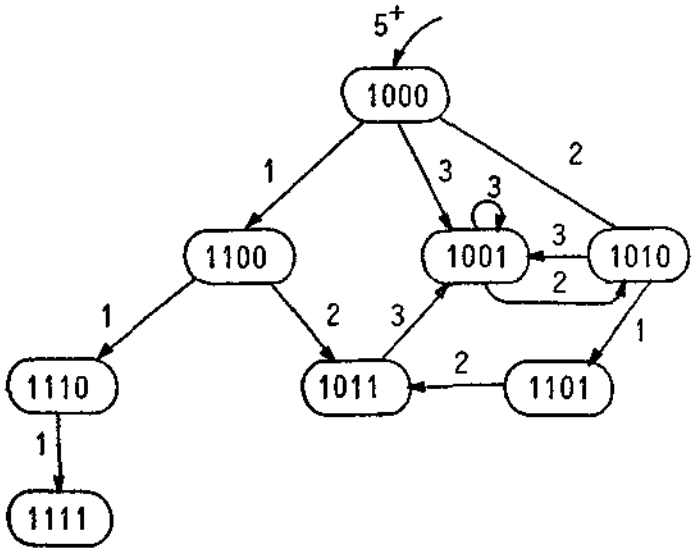


Fig. 3.4 - Ilustração de grafo de latências permitidas com vários ciclos sendo um ciclo ótimo.

A latência média, como o próprio nome indica, leva a um número médio de execuções do algoritmo no tempo. Como os elementos da sequência de entrada chegam ao PDTR sincronamente e as latências do ciclo ótimo podem ser diferentes (Figura 3.3), então pode ser necessário o uso de "buffers" na entrada e na saída do PDTR, para compatibilizar uma possível frequência irregular de iniciações do algoritmo com a vazão constante de chegada e de saída de dados do processador.

3.6 - FATOR DE UTILIZAÇÃO

Seja uma arquitetura encadeada com os inícios de execuções do algoritmo implementado programados para ocorrer o maior número de vezes no tempo, ou seja, de acordo com as latências de um ciclo ótimo. Nessa situação, a fração de tempo em que um particular segmento S_i da arquitetura encadeada é usado chama-se *fator de utilização* do segmento S_i, denotado por $FU(S_i)$.

Em um ciclo ótimo, uma execução do algoritmo inicia-se, em média, a cada LMmin unidades de tempo. Supondo que o segmento S_i seja utilizado U_i unidades de tempo em cada execução do algoritmo implementado, então:

$$FU(S_i) = \frac{U_i}{LMmin} \leq 1 \qquad (3.2)$$

Note-se que U_i é o número de X existentes na linha correspondente ao segmento S_i da tabela de reserva de segmentos da arquitetura encadeada em questão. Obviamente, o fator de utilização para qualquer segmento de uma arquitetura encadeada não pode nunca ser maior do que 100%, já que um segmento só pode ser utilizado por uma única execução do algoritmo implementado por vez.

O fator de utilização, como definido nesta seção, é um parâmetro importante na análise do funcionamento de uma arquitetura encadeada. Mas ele não dá uma idéia precisa do grau real de utilização de um segmento. Por exemplo, na Tabela 3.1 o maior fator de utilização é o do segmento S_5 ($U_5 = 5T$), enquanto, se forem desprezadas as unidades de tempo gastas com espera ocupada, os segmentos S_6 e S_{10} são os que realmente gastam mais tempo executando as suas tarefas (4T para S_6 e S_{10}, e 2T para S_5).

Portanto, faz-se necessário a introdução de um novo parâmetro, chamado *fator real de utilização*, denotado por $FRU(S_i)$ que é definido como a fração de tempo em que um particular segmento S_i da arquitetura encadeada é usado na execução da sua tarefa, desprezando o tempo gasto em espera ocupada e quando os inícios das execuções do algoritmo implementado se dão de acordo com as latências de um ciclo ótimo. Analogamente a $FU(S_i)$, $FRU(S_i)$ é dado por:

$$FRU(S_i) = \frac{UE_i}{LMmin} \leq 1, \qquad (3.3)$$

onde UE_i é o número de X existentes na linha da tabela de reserva de segmentos correspondente ao segmento S_i, desprezando-se todas as unidades de tempo gastas em espera ocupada durante a execução do algoritmo implementado. Da mesma maneira que $FU(S_i)$, $FRU(S_i)$ não pode ser maior do que 100%.

3.7 - TEMPO TOTAL DE PROCESSAMENTO

Nesta seção é introduzida a seguinte notação, válida no restante do trabalho: se $p = q \times s + r$, com p, q, r, s inteiros e $r < q$, então: $r = p \bmod q$ e $s = p \operatorname{div} q$.

Considere-se uma sequência finita $(SE = SE_0, SE_1, \ldots, SE_{n-1})$ com n conjuntos de dados iniciais para o algoritmo implementado em um processador com arquitetura encadeada. O tempo total de processamento dessa sequência de entrada SE será (Figura 3.5):

$$TP(n) = \sum_{i=1}^{n-1} \ell_i + d, \qquad (3.4)$$

onde ℓ_i é a latência (expressa em unidades de tempo T) existente entre os inícios de processamento dos elementos SE_{i-1} e SE_i, e d é o atraso do processador (tempo total fixo gasto no processamento de cada elemento SE_i).

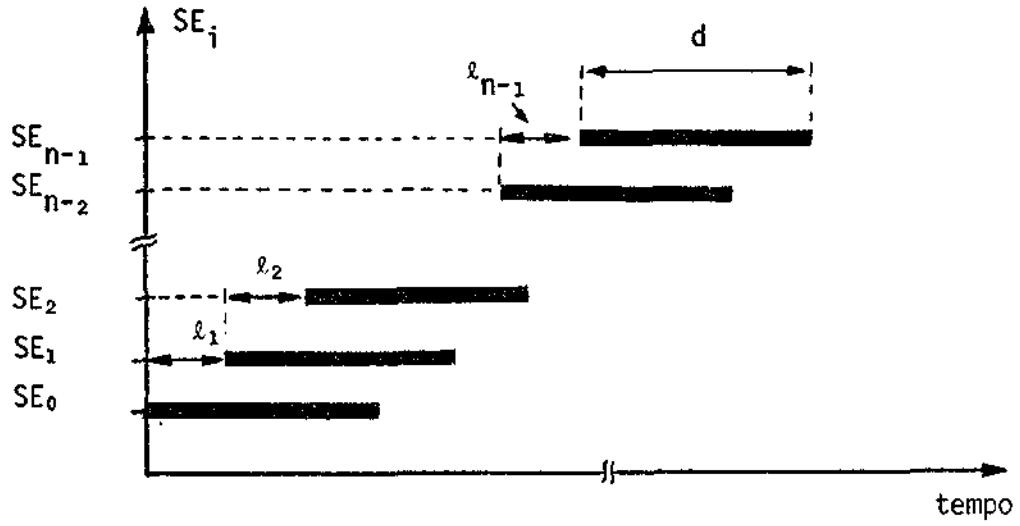


Fig. 3.5 - Distribuição no tempo de n execuções do algoritmo implementado.

Suponha-se, também, que os inícios de execuções do algoritmo estejam programados para ocorrer de acordo com as latências $\ell_1, \ell_2, \ldots, \ell_m$ de um ciclo do grafo de latências permitidas. Assim sendo

$$\ell_{i+m} = \ell_i, \text{ para } 1 \leq i < n - m$$

e,

$$TP(n) = [(n-1) \text{ div } m] \times \left(\sum_{i=1}^{m} \ell_i\right) + \sum_{i=1}^{f} \ell_i + d, \qquad (3.5)$$

onde $f = (n-1) \bmod m$.

Como $\left(\sum_{i=1}^{m} \ell_i\right)/m = LM$ (latência média), então a expressão final para o tempo total de processamento fica:

$$TP(n) = [(n-1) \text{ div } m] \times m \times LM + \sum_{i=1}^{f} \ell_i + d. \qquad (3.6)$$

Observe-se na Equação 3.6 que se n tende a infinito então TP (n)/n tende à latência média LM.

CAPÍTULO 4

OBTENÇÃO DO MAIOR NÚMERO DE EXECUÇÕES, NO TEMPO, DO ALGORITMO IMPLEMENTADO

Uma vez que se tenha realizado a distribuição das tarefas que compõem o algoritmo a ser implementado pelos vários segmentos de uma arquitetura encadeada, a questão que advém é: qual a maior frequência de chegada de dados ao processador que essa arquitetura encadeada é capaz de processar? A resposta a esta pergunta, como foi visto no Capítulo 3, é dada pelo ciclo ótimo de iniciações do algoritmo implementado e a respectiva latência média (mínima).

Neste capítulo são então desenvolvidos os procedimentos necessários para encontrar um ciclo ótimo de iniciações de uma arquitetura encadeada, conhecidos inicialmente apenas o seu grafo de fluxo de operandos e a tabela de tempos de execução dos segmentos que a compõem. À determinação da tabela de reserva de segmentos (Seção 4.1), gerada a partir do grafo de fluxo e da tabela de tempos de execução, segue-se a determinação do vetor de colisão (Seção 4.2) e do grafo de latências permitidas (Seção 4.3). Após essas ações intermediárias procede-se então à determinação de um ciclo ótimo de latências livre de colisão, o que é feito na Seção 4.4.

A análise de quanto os segmentos da arquitetura encadeada estão sendo efetivamente utilizados e como aumentar ainda mais a frequência de iniciações do algoritmo implementado, para além da obtida com o ciclo ótimo, é assunto do Capítulo 5.

Todos os procedimentos desenvolvidos neste e no próximo capítulo são apresentados na forma final de um algoritmo, com o objetivo primordial de deixar claros os métodos desenvolvidos. Devido a isso, a confecção desses algoritmos não envolveu maiores cuidados quanto a

uma redução do tempo de execução e dos tamanhos das estruturas de dados por eles manipuladas. Reduções estas que são facilmente atingíveis, por exemplo, com o uso de técnicas de armazenamento de matrizes esparsas, muito comuns no decorrer dos Capítulos 4 e 5.

No Apêndice A mostra-se que os algoritmos apresentados neste e no próximo capítulo terminam após a execução de um número finito de passos.

4.1 - DETERMINAÇÃO DA TABELA DE RESERVA DE SEGMENTOS

Nesta seção, baseado no grafo de fluxo e na tabela de tempos de execução de uma arquitetura encadeada, desenvolve-se um procedimento para a determinação da tabela de reserva de segmentos correspondente.

Considere-se uma arquitetura encadeada com s segmentos S_i, i = 1, ..., s, e tabela de tempos de execução representada na forma de um vetor:

$$E: \{1, \ldots, s\} \rightarrow I^+,$$

onde I^+ é o conjunto dos inteiros positivos e $E(i) = k \in I^+$, se e somente se k é o múltiplo inteiro da unidade de tempo T que o segmento S_i gasta para executar a sua tarefa, desde que todos os operandos necessários estejam disponíveis.

Considere-se uma relação biunívoca entre os nós do grafo de fluxo e o conjunto de números inteiros {1, 2, ..., f}, onde f é o número total de nós no grafo. O grafo de fluxo pode então ser representado na forma de duas matrizes binárias F e D:

$$F: \{1, 2, \ldots, f\} \times \{1, 2, \ldots, f\} \rightarrow \{0, 1\},$$

com F (i, j) = 1 se e somente se o nõ j sucede o nõ i no grafo de fluxo em questão;

$$D: \{1, 2, \ldots, f\} \times \{1, 2, \ldots, s\} \rightarrow \{0, 1\},$$

com D (i, j) = 1 se e somente se ao nõ i do grafo de fluxo está associado o segmento S_j da arquitetura encadeada.

A matriz F exprime a *conexão* entre os f nõs do grafo de fluxo e a matriz D exprime a *associação* existente entre os nõs do grafo de fluxo e os segmentos da arquitetura encadeada.

Note-se na matriz D que:

a) Como os segmentos da arquitetura encadeada são usados pelo menos uma vez na execução do algoritmo implementado, então não existe nunhuma coluna na matriz D com todos os elementos iguais a 0.

b) Como a cada nõ do grafo é associado um e somente um segmento, então em cada linha da matriz D existe um e apenas um elemento igual a 1.

Seja a matriz H preenchida inicialmente a partir do conteúdo das matrizes E, F e D:

$$H: \{1, 2, \ldots, f\} \times \{1, 2, \ldots, f + 1\} \rightarrow I,$$

onde I é o conjunto dos inteiros e

$$\text{para } j \neq f + 1, H(i, j) = \begin{cases} -1, \text{ se } F(i, j) = 0; \\ 1, \text{ se } F(i, j) = 1; \end{cases}$$

$$\text{para } j = f + 1, H(i, f + 1) = \sum_{k=1}^{s} (E(k) \times D(i, k)).$$

Note-se que na matriz H acima, se o elemento H (i, j) = 1, para j < f + 1, então o segmento associado ao nõ i do grafo de fluxo, após executar a sua tarefa por H (i, f + 1) unidades de tempo, transfere os resultados para o segmento associado ao nõ j do grafo de fluxo. A matriz H auxiliará na determinação, a cada unidade de tempo, de quais segmentos da arquitetura encadeada não estão executando a sua tarefa, estão em espera ocupada, estão efetivamente executando a sua tarefa ou já terminaram de executar a sua tarefa.

Seja qmax o número máximo de unidades de tempo, necessárias em uma execução completa do algoritmo implementado na arquitetura encadeada representada na forma das matrizes E, F e D descritas anteriormente. Considerando que não haja paralelismo na execução das respectivas tarefas pelos s segmentos da arquitetura encadeada e levando em conta o fato de que em cada unidade de tempo pelo menos um segmento está efetivamente executando a sua tarefa, então qmax será dado por:

$$qmax = \sum_{i=1}^{f} \sum_{j=1}^{s} (D(i, j) \times E(j)). \qquad (4.1)$$

Considere-se a matriz binária R com s linhas e qmax colunas:

$$R: \{1, 2, \ldots, s\} \times \{1, 2, \ldots, qmax\} \rightarrow \{0, 1\},$$

com R (i, j) = 1, se e somente se o segmento S_i é utilizado na unidade de tempo T_j.

Supondo inicialmente todos os elementos da matriz R iguais a zero, propõe-se o seguinte procedimento que, baseado no conteúdo das matrizes H e D, gera a tabela de reserva de segmentos nas primeiras colunas da matriz R:

a) Faça um ponteiro de tempo apontar para a primeira coluna da matriz R.

b) Pesquise as colunas da matriz H, com o índice variando de 1 a f, em busca daquelas que não têm nenhum elemento maior do que zero. Isto significa que os segmentos dos nós associados a essas colunas já têm todos os operandos disponíveis para executar a sua tarefa. Elimine dos resultados dessa pesquisa os nós cujos segmentos associados já terminaram de executar a sua tarefa, ou seja, aqueles que possuem linha na matriz H com elemento H (i, f + 1) = 0. Inicialmente, nenhum elemento da matriz H é igual a zero. Entretanto, no decorrer do procedimento, isto virá a acontecer (veja-se o passo e).

c) Pesquise as colunas da matriz H, com o índice variando de 1 a f, em busca daquelas que ao mesmo tempo têm pelo menos um elemento igual a zero e pelo menos um elemento maior do que zero. Isso significa que os segmentos associados aos nós dessas colunas já receberam pelo menos um operando, mas não todos os operandos necessários para executar a sua tarefa e estão, portanto, em espera ocupada.

d) Na coluna da matriz R, apontada pelo ponteiro de tempo, faça iguais a um todas as linhas cujos segmentos são associados aos nós encontrados nas pesquisas efetuadas nos passos b) e c). A determinação do segmento associado a um nó é feita com o auxílio da matriz D.

e) Decremente de um os elementos da coluna f + 1 da matriz H correspondentes às linhas dos nós encontrados apenas na pesquisa feita em b), e faça iguais a zero todos es elementos das linhas da matriz H que anteriormente eram iguais a 1 e cujo elemento na coluna f + 1 tenha recebido o valor zero após o decremento.

f) Se ainda existe algum elemento maior do que zero na coluna f + 1 da matriz H, avance o ponteiro de tempo para a coluna seguinte da matriz R e retorne para iniciar nova pesquisa em b). Caso contrário, a tabela de reserva de segmentos já se encontra armazenada na matriz R.

Este procedimento é detalhado no Algoritmo A apresentado a seguir.

Algoritmo A: determinação da tabela de reserva de segmentos. Baseado no grafo de fluxo e na tabela de tempos de execução representados na forma das matrizes E, F e D descritas nesta seção, este algoritmo gera, após o seu término, nas primeiras q colunas da matriz R (também descrita nesta seção), a tabela de reserva de segmentos correspondente:

A1. Crie um vetor binário Y = (y (1), ..., y (f)), com f bits (Y será usado para armazenar o resultado da pesquisa dos nós do grafo de fluxo cujos segmentos estão executando a sua tarefa).

A2. Crie um vetor binário Z = (z (1), ..., z (f)), com f bits (Z será usado para armazenar o resultado da pesquisa dos nós do grafo de fluxo cujos segmentos estão em espera ocupada).

A3. Crie uma matriz H com f linhas e f + 1 colunas, e elementos H (i, j), i = 1, ..., f e j = 1, ..., f + 1, sendo números inteiros (H será usada para determinar em cada unidade de tempo quais nós do grafo têm segmentos que não estão sendo utilizados, estão executando a sua tarefa ou estão em espera ocupada).

A4. Faça todos os elementos da matriz R iguais a zero: R (i, j) = 0, i = 1, ..., s e j = 1, ..., qmax (inicialmente nenhum segmento da arquitetura encadeada foi utilizado na execução do algoritmo implementado).

A5. Faça q = 1 (q é o ponteiro de tempo que auxiliará o preenchimento da matriz R).

A6. Preencha a matriz H com o seu conteúdo inicial: para i = 1, ..., f e j = 1, ..., f, faça H (i, j) = -1, se F (i, j) = 0; caso contrário, F (i, j) = 1, faça H (i, j) = 1. Para i = 1, ..., f, faça $H(i, f+1) = \sum_{k=1}^{s} (E(k) \times D(i, k))$.

A7. Zere todos os bits dos vetores Y e Z: para i = 1, ..., f, faça z (i) = 0 e y (i) = 0.

A8. Faça j = 1 (j será usado para varrer as colunas da matriz H na pesquisa dos nós do grafo que estão executando a sua tarefa).

A9. Se H (i, j) ≤ 0 para todo i = 1, ..., f, vá para o passo A10. Caso contrário, vá para o passo A11.

A10. Se H (j, f + 1) > 0, o segmento associado ao nó j está executando a sua tarefa, então faça y (j) = 1, Caso contrário não faça nada.

A11. Se j < f, faça j = j + 1 e retorne para o passo A9. Caso Contrário, j = f e todas as colunas da matriz H já foram pesquisadas, vá para o passo A12.

A12. Faça j = 1 (j será usado para varrer as colunas da matriz H na pesquisa dos nós do grafo cujos segmentos estão em espera ocupada).

A13. Faça i = 1 (i será usado para varrer as linhas da matriz H).

A14. Se H (i, j) < 0 e i < f, faça i = i + 1 e repita este passo. Se H (i, j) = 0 e i < f, faça i = i + 1 e vá para o passo A15. Se H (i, j) > 0 e i < f, faça i = i + 1, e vá para o A16. Caso contrário, i = f, vá para o passo A18.

A15. Se H (i, j) ≤ 0 e i < f, faça i = i + 1 e repita este passo. Se H (i, j) > 0, vá para o passo A17. Caso contrário, H (i, j) ≤ 0 e i = f, vá para o passo A18.

A16. Se H (i, j) ≠ 0 e i < f, faça i = i + 1 e repita este passo. Se H (i, j) = 0, vá para o passo A17. Caso contrário, H (i, j) ≠ 0 e i = f, vá para o passo A18.

A17. Faça z (j) = 1 (pelo menos um nó antecessor do nó j já terminou de executar a sua tarefa, mas não todos os nós antecessores).

A18. Se j < f, faça j = j + 1 e retorne para o passo A13. Caso contrário, j = f e todas as colunas da matriz H já foram pesquisadas, vá para o passo A19.

A19. Para i = 1, ..., f: se y (i) = 1 ou z (i) = 1, procure k tal que D (i, k) = 1 e faça R (k, q) = 1.

A20. Faça i = 1 (i será usado para varrer o vetor Y).

A21. Se y (i) = 1, faça H (i, f + 1) = H (i, f + 1) - 1 e vá para o passo A22. Caso contrário, vá para o passo A23.

A22. Se H (i, f + 1) = 0, então para j = 1, ..., f, se H (i, j) = 1, faça H (i, j) =0.

A23. Se i < f, faça i = i + 1 e retorne para o passo A21. Caso contrário, i = f e todo o vetor Y foi pesquisado, vá para o passo A24.

A24. Se existe i tal que H (i, f + 1) > 0, faça q = q + 1 e retorne para o passo A7. Caso contrário já se encontra armazenada nas primeiras q colunas da matriz R a tabela de reserva de segmentos procurada, e o algoritmo termina.

4.2 - DETERMINAÇÃO DO VETOR DE COLISÃO

Considere-se a tabela de reserva de segmentos de uma arquitetura encadeada com s segmentos e q unidades de tempo de atraso, representada na forma de uma matriz R, como gerada pelo Algoritmo A da seção anterior:

$$R: \{1, 2, \ldots, s\} \times \{1, 2, \ldots, q\} \rightarrow \{0, 1\},$$

com R (i, j) = 1 se e somente se o segmento S_i é utilizado na unidade de tempo T_j. Por definição, cada linha e cada coluna da matriz R possui pelo menos um elemento igual a 1.

Seja a *distância* d(r (i, j), R (i, k)), entre dois elementos R (i, j) e R (i, k) de uma mesma linha i da matriz R, definida como o módulo da diferença entre os índices j e k:

$$d\,(R\,(i, j), R\,(i, k)) = |j - k|. \qquad (4.2)$$

Seja o vetor binário C, com um número de bits igual a maior distância possível entre os elementos de uma mesma linha da matriz R. Como o limitante superior para d (R (i, j), R (i, k)) na matriz R é q - 1, então C possui q - 1 bits: C = (c (q - 1), ..., c (1)).

Considerando inicialmente todos os bits do vetor C iguais a zero, a determinação do vetor de colisão, como descrito na Seção 3.3, pode ser feita pesquisando as linhas da matriz R para calcular as distâncias entre todos os pares de elementos iguais a 1, existentes em uma mesma linha desta matriz. Se para cada distância d assim encontrada o bit correspondente no vetor C for feito igual a 1 (c (d) = 1), então após todas as linhas da matriz R terem sido pesquisadas, C armazenará o vetor de colisão procurado, nos bits c (n) a c (1), onde n é a maior distância encontrada.

Este procedimento simples é detalhado no Algoritmo B apresentado a seguir:

Algoritmo B: determinação do vetor de colisão. A matriz R, com s linhas e q colunas armazena a tabela de reserva de segmentos a ser analisada. No final do algoritmo, nos bits c (n) a c (1) do vetor binário C (com q - 1 bits) estará o vetor de colisão da matriz R.

B1. Faça o vetor C vazio: c (i) = 0, para i = 1, ..., q - 1 e n = 0 (n armazenará o número de bits final do vetor de colisão).

B2. Faça i = 1 (i será a linha da matriz sendo pesquisada).

B3. Faça j = 1 (j é usado para varrer a linha i em busca dos elementos R (i, j) iguais a 1).

B4. Se R (i, j) = 0 e j < q, incremente j de 1 e repita este passo. Se j = q vá para o passo B8. Caso contrário, R (i, j) = 1 e j < q, vá para o passo B5.

B5. Faça k = j + 1 (k é usado para varrer o restante da linha i a partir da coluna j em busca de elementos R (i, k) = 1).

B6. Se R (i, k) = 0 e k ≤ q, incremente k de 1 e repita este passo. Se k > q, faça j = j + 1 e retorne ao passo B4. Caso contrário, R (i, k) = 1, vá para o passo B7.

B7. Faça c (k - j) = 1. Se k - j > n faça n = k - j. Incremente k de 1 e retorne ao passo B6.

B8. Faça i = i + 1. Se i ≤ s, retorne para o passo B3. Caso contrário, o algoritmo termina.

4.3 - DETERMINAÇÃO DO GRAFO DE LATÊNCIAS PERMITIDAS

Considere-se um vetor de colisão armazenado nos bits c (n) a c (1) de um vetor C, como gerado pelo Algoritmo B da seção anterior. Por definição c (n) = 1.

Sendo g o número de nós do grafo de latências permitidas que tem o vetor de colisão C como nó inicial, denotado por GLP (C), considere-se a seguinte representação para a tabela de conexão do grafo GLP (C) na forma do vetor V e da matriz G:

V: {1, 2, ..., g} → {todos os vetores de colisão que são nó do grafo GLP (C)}.

Os elementos V (i) = (vi (n), ..., vi (1)) são vetores binários com n bits, onde n é o número de bits do vetor de colisão C e vi (ℓ) = 0 se e somente se ℓ é uma latência permitida para o vetor de colisão V (i). E

G: {1, 2, ..., g} x {1, 2, ..., g} → {todos os subconjuntos do conjunto de latências permitidas do vetor de colisão C}.

Representam-se os elementos G (i, j) dados pelos vetores binários (gij (n + 1), gij (n), ..., gij (1)) com n + 1 bits, onde n é o número de bits do vetor de colisão C; gij (ℓ) = 0 se e somente se V (j) sucede V (i) no grafo GLP (C) através do arco com latências $\ell < n$; e gij (n + 1) representa o arco $(n + 1)^+$ no grafo GLP (C).

Suponha-se que no grafo GLP (C), V (j) sucede V (i) através do arco com latências ℓ (gij (ℓ) = 0). De acordo com o exposto na Seção 3.3, V (j) pode ser obtido através do OU-lógico bit a bit entre o vetor de colisão C e o vetor V (i) deslocado de ℓ bits para a direita:

$$\text{para } \ell < n,\ vj\ (k) = \begin{cases} c\ (k)\ v\ vi\ (k+1),\ k = 1, \ldots, n - \ell; \\ c\ (k),\ k = n - \ell + 1, \ldots, n; \end{cases} \tag{4.3}$$

para $\ell = (n + 1)^{+}$, $vj\ (k) = c\ (k)$, $k = 1, \ldots, n$.

Portanto, o vetor de colisão de qualquer nó do grafo GLP (C) apresenta:

a) 1 pelo menos nas mesmas posições em que o vetor C;

b) o mesmo número n de bits do vetor C.

Seja então o vetor binário V = (v (r), ..., v (1)). Define-se V como um *vetor candidato* a ser um nó do grafo GLP (C) se e somente se:

a) V tem o mesmo número de bits que C: r = n;

b) V apresenta 1 pelo menos nas mesmas posições correspondentes aos 1 no vetor C: se c (i) = 1, então v (i) = 1.

Note-se que todos os vetores dos nós do grafo GLP (C) são vetores candidatos, mas o oposto pode não ser verdadeiro.

Considerando as dimensões iniciais de V e G como g = v, onde v é o número total de vetores condidatos a pertencer ao grafo GLP (C) (note-se que $v = 2^{z}$, onde z é o total de bits iguais a zero no vetor C) e considerando também que qualquer nó no grafo GLP (C) é sucessor de algum outro nó do mesmo grafo, então propõe-se o seguinte procedimento para a obtenção do grafo GLP (C):

a) Preencha V inicialmente com todos os v vetores candidatos a pertencer ao grafo GLP (C).

b) Preencha a matriz G inicialmente de forma que gij (ℓ) = 0 se e somente se os vetores condidatos V (j) e V (i) satisfazem à Equação 4.3, para a latência permitida ℓ. Após o preenchimento da matriz G, se algum elemento G (i, j) desta matriz for dado por um vetor com todos os bits iguais a 1, então V (j) não sucede V (i) no grafo GLP (C) e G (i, j) é dito ser um *elemento vazio*.

c) Elimine todos os vetores candidatos que não são sucessores de nenhum outro vetor candidato. Para isto, basta pesquisar as colunas da matriz G. Se G (i, k) é vazio para i = 1, ..., v, então elimine o vetor V (k) de V e a linha k e coluna k da matriz G. Repetindo este processo de eliminação até que não haja mais nenhuma coluna na matriz G com todos os elementos vazios, então V apenas conterá vetores que são nós do grafo GLP (C) e a matriz G representará a tabela de conexão do grafo de latências permitidas que tem o vetor C como nó inicial.

Este procedimento pode ser reduzido ao Algoritmo C que segue.

Algoritmo C: determinação do grafo de latências permitidas. Este algoritmo, baseado no vetor de colsião C = (c (n), ..., c (1)), gera, após o seu término, o grafo de latências permitidas que tem o vetor de colisão C como nó inicial. Este grafo será representado na forma de uma tabela de conexão que ficará armazenada nas primeiras g colunas e linhas das matrizes V e G descritas nesta seção. Originalmente V e G possuem v linhas e colunas, onde v é o número total de vetores candidatos a pertencer ao grafo GLP (C):

C1. Crie um vetor binário R = (r (n), ..., r (1)) com n bits (R será usado para gerar todos os vetores binários possíveis com n bits e bit n igual a 1).

C2. Crie um vetor binário T = (t (n), ..., t (1)) com n bits (T será usado na geração dos nós sucessores).

C3. Faça K = 1 (k apontará para as linhas da matriz V a medida que elas forem sendo preenchidas).

C4. Faça todos os bits do vetor R armazenar zero, exceto o bit n que armazenará um: para i = 1, ..., n - 1 faça r (i) = 0 e r (n) = 1.

C5. Verifique se R é um vetor candidato a pertencer ao grafo. Se r (i) $\wedge$ c (i) for igual a c (i), para i = 1, ..., n, R é vetor candidato, então vá para o passo C6. Caso contrário, vá para o passo C7.

C6. Faça V (k) = R. Se k < v, faça k = k + 1 e vá para o passo C7. Caso contrário, todos os vetores candidados já foram encontrados, vá para o passo C8. (Observe-se que a maneira de preenchimento da matriz V leva à seguinte ordenação entre os seus elementos: se $1 \leq p < q \leq v$, então $\sum_{i=1}^{n} 2^{vp(i)} < \sum_{i=1}^{n} 2^{vq(i)}$; e que V (1) é o próprio vetor de colisão C).

C7. Calcule o próximo vetor R. Encontre j tal que r (j) = 0 e se j > 1, r (i) = 1 para i = 1, ..., j - 1. Faça r (j) = 1 e se j > 1, faça também r (i) = 0 para i = 1, ..., j - 1. Retorne ao passo C5.

C8. Faça inicialmente todos os bits de todos os vetores que dão os elementos da matriz G iguais a 1: gij (ℓ) = 1, para i = 1, ..., v, j = 1, ..., v, ℓ = 1, ..., n + 1.

C9. Faça k = 1 (k apontará para as linhas da matriz V a medida que a matriz G for sendo preenchida).

C10. Faça o vetor R, criado no passo C4, igual ao vetor V (k) e $\ell = 1$ (R é utilizado na pesquisa das latências permitidas dos vetores da matriz V e ℓ armazenará as possíveis latências permitidas).

C11. Se r (1) = 0, faça o vetor T igual ao vetor sucessor de R com latência ℓ: t (i) = c (i) v r (i + 1), i = 1, ..., n - 1 e t (n) = c (n), e vá para o passo C12. Caso contrário, vá para o passo C13.

C12. Encontre j tal que V (j) = T e faça gkj (ℓ) =0.

C13. Se $\ell < n - 1$, faça $\ell = \ell + 1$; r (i) = r (i + 1), para i = 1, ..., n - 1; r (n) = 0; e retorne para o passo C11. Caso contrário, terminou a pesquisa dos sucessores de V (k), vá para o passo C14.

C14. Insira o arco $(n + 1)^+$ que vai para o nó inicial. Faça gk1 (n + 1) = 0.

C15. Se k < v, faça k = k + 1 e retorne para o passo C10. Caso contrário, todos os vetores da matriz V já foram pesquisados, vá para o passo C16.

C16. Faça g = v (g armazenará as dimensões finais das matrizes V e G).

C17. Faça j = 1 (j será agora usado para varrer as colunas da matriz G).

C18. Se todos os bits de todos os vetores da coluna j na matriz G forem iguais a 1, então vá para o passo C19. Caso contrário, vá para o passo C23.

C19. Se j < g, elimine o vetor candidato V (j) da matriz V: faça V (i) = V (i + 1), para i = j, ..., g - 1.

C20. Se j < g, elimine a linha j da matriz G: Faça G (i, k) = = G (i + 1, k) para k = 1, ..., g e i = j, ..., g - 1.

C21. Se j < g, elimine a coluna j da matriz G: Faça G (k, i) = = G (k, i + 1) para k = 1, ..., g e i = j, ..., g - 1.

C22. Faça g = g - 1 e retorne para o passo C17.

C23. Se j < g, faça j = j + 1 e retorne para o passo C18. Caso contrário, não há mais vetores condidatos a ser eliminados, o algoritmo termina.

4.4 - DETERMINAÇÃO DE UM CICLO ÓTIMO

A determinação de um ciclo ótimo existente no grafo de latências permitidas pode ser feita pesquisando as latências médias de todos os ciclos existentes nc grafo, em busca daquele que apresenta a latência média mínima. Mas como detectar todos os ciclos existentes em um grafo de latências permitidas?

Shar (1972) demonstrou as seguintes propriedades do grafo de latências permitidas:

Propriedade 1: Teorema 3.2 na referência citada. Um ciclo, especificado apenas pelas latências dos arcos que vão de um nó a outro sequencialmente no ciclo, passa através de um único conjunto de nós no grafo de latências permitidas. Ou seja, um ciclo pode ser univocamente especificado pela sequência de latências dos arcos por onde ele passa.

Propriedade 2: particularização do Teorema 3.4 da referência citada. Qualquer ciclo no grafo de latências permitidas pode ser alcançado, a partir do nó inicial do grafo, através de uma trajetória determinada por uma sequência de todas as latências do ciclo.

Acrescente-se a esta lista as seguintes propriedades do grafo de latências permitidas:

Propriedade 3. Todos os nós do grafo de latências permitidas possuem pelo menos um nó sucessor (qualseja o nó inicial, através do arco $(n + 1)^+$).

Propriedade 4. Sendo g o número de nós de um grafo de latências permitidas, qualquer ciclo do grafo passa, no máximo, por g nós.

Baseado nessas propriedades, propõe-se o seguinte procedimento para a determinação de todos os ciclos de um grafo de latências permitidas, que é feita de maneira exuastiva com o percorrimento de todas as trajetórias possíveis a partir do nó inicial do grafo. Sempre que alguma trajetória retornar a um nó já percorrido (e isto sempre ocorre de acordo com as propriedades 3 e 4) um ciclo é detectado. Este ciclo é especificado apenas pelas latências dos arcos que o compõem, de acordo com as propriedades 1 e 2.

Seja então o grafo de latências permitidas, representado, na forma de uma tabela de conexão, por matrizes V e G geradas pelo Algoritmo C da seção anterior. Na descrição que se segue *índice de um nó* é o índice correspondente ao vetor de colisão em questão, nas matrizes V e G. O percorrimento de todas as trajetórias a partir do nó inicial desse grafo, em busca dos ciclos existentes, pode ser feito com o auxílio de uma estrutura de dados tipo pilha:

a) Inicialmente, coloque na pilha o índice do nó inicial do grafo.

b) Pesquise o conteúdo da pilha para verificar se o índice existente no seu topo já foi empilhado anteriormente, o que caracteriza um ciclo.

b.1) Se não for detetado um ciclo, então empilhe o índice do primeiro nó que sucede ao nó com o índice no topo da pilha, percorrendo a matriz G a partir da coluna 1. E retorne para fazer nova pesquisa do conteúdo da pilha.

b.2) Se for detetado um ciclo, então obtenha da matriz G, para estudo, a sequência de latências do ciclo, já que os índices dos nós do ciclo estão empilhados, no sentido do fundo para o topo, na pilha. Após o que, desempilhe o índice i existente no topo da pilha e tente empilhar o índice do primeiro sucessor do novo nó com o índice no topo da pilha percorrendo a matriz G a partir da coluna i + 1, no sentido dos índices maiores. Se isto for possível, retorne para fazer nova pesquisa do conteúdo da pilha. Caso contrário, continue tentando desempilhar e empilhar o índice de um novo sucessor. Quando a pilha ficar vazia, todos os ciclos do grafo em questão terão sido detectados.

Este procedimento é detalhado no Algoritmo D apresentado a seguir.

Algoritmo D: determinação do ciclo ótimo. Este algoritmo considera o grafo de latências permitidas a ser pesquisado já armazenado em uma tabela de consexão, na forma das matrizes V e G geradas pelo Algoritmo C. As t latências do primeiro ciclo ótimo encontrado ficarão armazenadas no vetor L = (L (1), ..., L (g)) com g elementos inteiros (onde g é o número de nós do grafo de latências permitidas sendo analisado), sequencialmente de L (1) a L (t), e em LM ficará a latência média (mínima) desse ciclo.

D1. Crie um vetor P com g + 1 posições, sendo os elementos P (i), i = 1, ..., g+1, números inteiros e g é o número de nós do grafo (P armazenará a pilha usada na determinação dos ciclos do grafo).

D2. Crie um vetor Q com g elementos, sendo os elementos Q (i), i = 1, ..., g, números inteiros (Q armazenará as latências de cada ciclo encontrado no grafo).

D3. Faça p = 0 (p apontará para a última posição preenchida na pilha P, ou seja, para o topo da pilha).

D4. Faça LM = g x (n + 1) (LM armazenará a latência média do ciclo armazenado no vetor L).

D5. Faça p = p + 1 e P (p) = 1 (coloque na pilha o índice do nó inicial do grafo).

D6. Verifique se existe k inteiro, $1 \leq k < p$, tal que P (k) = P (p). Se não existe, faça j = 1 e vá para o passo D14. Caso contrário, foi detectado um ciclo, vá para o passo D7.

D7. Faça q = 0 (q auxiliará no preenchimento do vetor Q).

D8. Faça r = P (k); s = P (k + 1); e ℓ = 1.

D9. Se grs (ℓ) = 1 na matriz G, faça $\ell = \ell + 1$ e repita este passo. Caso contrário, grs (ℓ) = 0, vá para o passo D10 (como o nó com índice s é sucessor do nó com índice r, sempre vai ser possível achar ℓ tal que grs (ℓ) = 0 na matriz G).

D10. Faça q = q + 1 e Q (q) = ℓ. Se s ≠ P (p), faça k = k + 1 e retorne para o passo D8. Caso contrário, todas as latências do ciclo detectado já estão no vetor Q, vá para o passo D11.

D11. Se LM < $(\sum_{i=1}^{q} Q(i))/q$, então vá para o passo D12. Caso contrário, foi encontrado um ciclo com latência média menor, faça LM = $(\sum_{i=1}^{q} Q(i))/q$; L (i) = Q (i), i = 1, ..., q; t = q; e vá para o passo D12 (t guardará o último elemento preenchido no vetor L).

D12. Faça j = P (p) e p = p - 1 (desempilhe o índice do nó no topo da pilha).

D13. Faça j = j + 1.

D14. Faça i = P (p) (i é índice do nó que está atualmente no topo da pilha).

D15. Se j ≤ g, vá para o passo D16. Caso contrário, não há mais sucessor de i que já não tenha sido pesquisado, vá para o passo D18.

D16. Faça ℓ = 1 (ℓ será usado para verificar se o nó com índice j é sucessor do nó com índice i).

D17. Se gij (ℓ) = 1 e ℓ < n + 1, faça ℓ = ℓ + 1 e repita este passo. Se gij (ℓ) = 1 e ℓ = n + 1, faça j = j + 1 e retorne para o passo D15. Caso contrário, se o nó com índice j é sucessor do nó com índice i, faça p = p + 1, P (p) = j e retorne para o passo D6.

D18. Se p > 1, retorne para o passo D12. Caso contrário, a pilha seria esvaziada no passo D12, o algoritmo termina.

CAPÍTULO 5

OBTENÇÃO DO NÚMERO MÁXIMO DE EXECUÇÕES, NO TEMPO, DO ALGORITMO IMPLEMENTADO

Conhecidos o grafo de fluxo e a tabela de tempos de execução de uma arquitetura encadeada, a aplicação dos procedimentos desenvolvidos no capítulo anterior para a identificação de um ciclo ótimo de iniciações do algoritmo implementado leva ao maior número possível de execuções desse algoritmo no tempo. Mas, esse número não pode ser aumentado ainda mais com alterações mínimas na arquitetura encadeada? Na Seção 3.3, comentou-se que o mapeamento entre o conjunto de tabelas de reserva de segmentos e o conjunto de vetores de colisão não é uma função biunívoca. Com isso, a resposta a esta pergunta deve ser obtida analisando a própria tabela de reserva de segmentos, já que a análise do grafo de latências permitidas, gerado a partir do vetor de colisão, fornece apenas o ciclo ótimo de iniciações como a melhor política de escalonamento.

Considere-se o exemplo da Figura 5.1. Nele, piorando o desempenho de um particular segmento (aumento do tempo gasto pelo segmento S_4 para realizar a sua parte do algoritmo), consegue-se duplicar o número de execuções do algoritmo, determinado pelo ciclo de iniciações ótimo. O mesmo efeito também é obtido com o uso de um segmento (S_5) que apenas insere um atraso na passagem, para o segmento S_3, dos operandos gerados pelo segmento S_4.

Portanto, a resposta, à pergunta inicialmente formulada neste capítulo é de que é possível a obtenção de uma política de escalonamento, associada com alterações mínimas na arquitetura encadeada, que aumente o número de execuções do algoritmo implementado para além daquele inicialmente determinado por um ciclo de iniciações ótimo. Por *alterações mínimas* na arquitetura encadeada entende-se que podem ser tomadas apenas as seguintes providências:

a) piorar o desempenho de um segmento através do aumento do tempo gasto para ele realizar a sua tarefa,

b) inserir *segmentos atrasadores* que apenas transferem para a sua saída os operandos existentes na sua entrada.

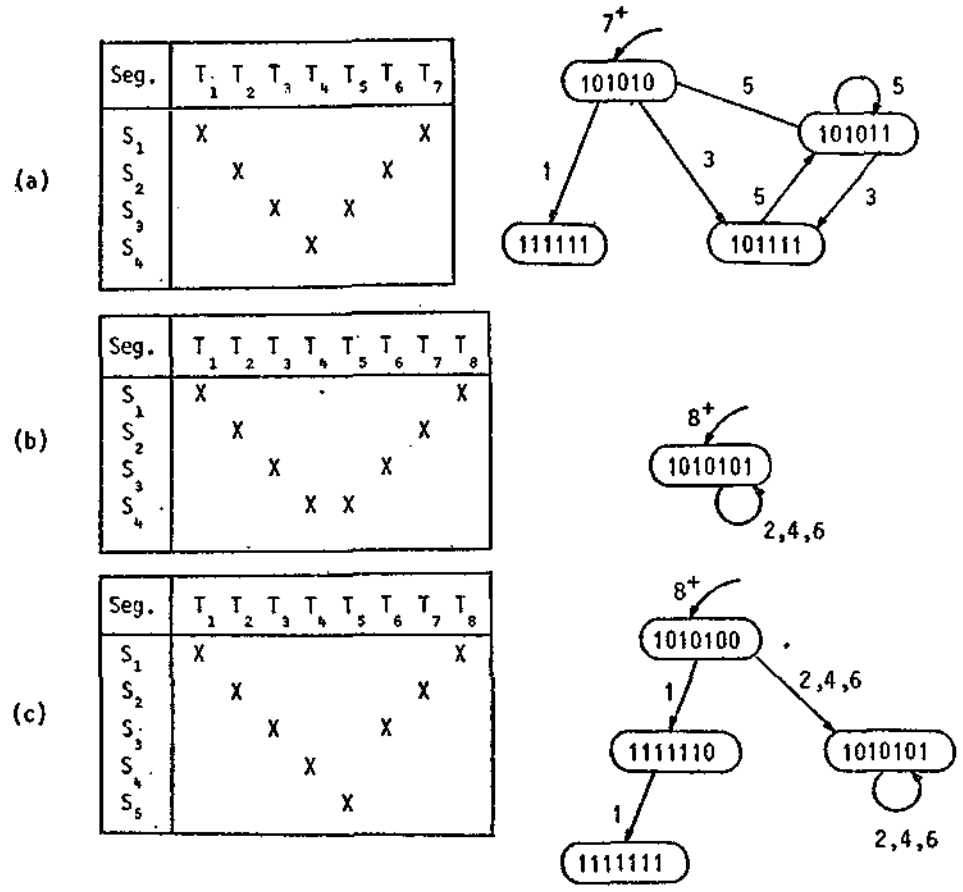


Fig. 5.1 - Exemplo de aumento do número de execuções do algoritmo implementado no tempo: a) LMmin = 4, b) LMmin = 2, c) LMmin = 2.

Na Seção 5.1 é encontrado o limitante inferior para a latência média do ciclo ótimo de uma arquitetura encadeada e na seção seguinte determina-se o menor valor possível que esse limitante inferior pode assumir. A seguir, nas Seções 5.3 a 5.7 são desenvolvidos procedimentos para, apenas efetuando essas alterações mínimas em uma arquite

tura encadeada, dita *arquitetura encadeada original*, levar os seus segmentos a ser efetivamente utilizados o máximo possível no tempo. E com isso maximiza-se o número de execuções concorrentes do algoritmo implementado. Na Seção 5.8 são tecidas considerações a respeito do efeito dos procedimentos desenvolvidos neste capítulo, sobre o tempo total de processamento.

Todos os procedimentos desenvolvidos neste capítulo são reduzidos a uma forma final do algoritmo, valendo as mesmas observações sobre os algoritmos desenvolvidos neste trabalho, feitas no início do Capítulo 4.

5.1 - DETERMINAÇÃO DO LIMITANTE INFERIOR PARA A LATÊNCIA MÉDIA DO CICLO ÓTIMO

Dada uma arquitetura encadeada, representada pelo seu grafo de fluxo e tabela de tempos de execução, qual o *limitante inferior para a latência média* dos ciclos do grafo de latências permitidas por ela determinado?

Esse limitante inferior, denotado por LMmin*, pode ser obtido do fator de utilização dos segmentos da arquitetura em estudo. O fator de utilização (veja-se a Seção 3.6) de qualquer segmento de uma arquitetura encadeada não pode ser maior do que 100%, o que resulta da Equação 3.2 em $LMmin^* \geq U_i$, onde U_i é o número de unidades de tempo que o segmento S_i é usado em uma execução do algoritmo implementado. Como esta inequação tem de valer para todos os segmentos da arquitetura encadeada, então:

$$LMmin \geq \max (U_i) = LMmin^* \qquad (5.1)$$

Portanto, o limitante inferior para a latência média de qualquer ciclo (até mesmo de um ciclo ótimo) é o número máximo de X existente em uma linha da tabela de reserva de segmentos, gerada a partir do grafo de fluxo e da tabela de tempos de execução da arquitetura encadeada.

Baseado nessa propriedade (que resulta na Equação 5.1) o Algoritmo E, apresentado a seguir, obtém o limitante inferior para a latência média do ciclo ótimo.

Algoritmo E: determinação do limitante inferior para a latência média do ciclo ótimo. Este algoritmo, analisando a tabela de reserva de segmentos, representada na forma de uma matriz R, definida na Seção 4.1, gera após o seu término o limitante inferior para a latência média do ciclo ótimo: LMmin*:

E1. Faça LMmin* = 0.

E2. Faça i = 1 (i será utilizado para varrer as p linhas da matriz R).

E3. Faça $t = \sum_{j=1}^{q} R(i, j)$.

E4. Se t > LMmin*, faça LMmin* = t.

E5. Se i < s, faça i = i + 1 e retorne para o passo E3. Caso contrário, todas as linhas da matriz R já foram pesquisadas, o algoritmo termina.

5.2 - DETERMINAÇÃO DO LIMITANTE INFERIOR REAL PARA A LATÊNCIA MÉDIA DO CICLO ÓTIMO

Na obtenção do limitante inferior LMmin* (veja-se Equação 5.1) entram todas as unidades de tempo em que os segmentos da arquitetura encadeada são utlizados, até mesmo as referentes à espera ocupada. Supondo que seja eliminado da arquitetura encadeada o uso de segmentos em espera ocupada (o que é abordado na Seção 5.3) vários X da tabela de reserva de segmentos poderão ser removidos. Com isto, o limitante inferior para a latência média de um ciclo ótimo do grafo de latên

cias permitidas seria reduzido ao seu menor valor possível. Esse valor, chamado *limitante inferior real* e denotado por LIR*, pode ser calculado através do fator real de utilização dos segmentos da arquitetura encadeada e resulta da Equação 3.3 em:

$$LMmin \geq \max (UE_i) = LIR^*, \tag{5.2}$$

onde $\max (UE_i)$ é o número máximo de X existentes em uma linha da tabela de reserva de segmentos, desprezando-se todas as unidades de tempo gastas em espera ocupada, durante a execução do algoritmo implementado. UE_i pode também ser obtido do conteúdo das matrizes E e D descritas na Seção 4.1, o que resulta em:

$$LIR^* = \max \left[\sum_{i=1}^{f} (D(i,j) \times E(j)) \right], \text{ para } j = 1, \ldots, s. \tag{5.3}$$

É importante ressaltar que dada uma arquitetura encadeada, representada pelo seu grafo de fluxo e tabela de tempos de execução, a tabela de reserva de segmentos gerada é única e determina um limitante inferior LMmin*. Já o limitante inferior real é apenas o menor valor que o limitante inferior da latência média dos ciclos do grafo de latências permitidas pode assumir:

$$LMmin \geq LMmin^* \geq LIR^* \tag{5.4}$$

Como será visto na Seção 5.3, se LMmin* ≠ LIR*, então a arquitetura encadeada tem de sofrer alterações para que esta igualdade ocorra.

Usando a Equação 5.3 a determinação de LIR* pode ser feita com o Algoritmo F, apresentado a seguir.

Algoritmo F: determinação do limitante inferior real para a latência média do ciclo ótimo. Este algoritmo, baseado no conteúdo das matrizes D e E descritas na Seção 4.1, obtém, após o seu término, o limitante inferior real LIR* para a latência média do ciclo ótimo do grafo de latências permitidas:

F1. Faça LIR* = 0.

F2. Faça j = 1 (j será usado para varrer as colunas da matriz D e as linhas da matriz E).

F3. Faça $t = \sum_{i=1}^{f} (D(i, j) \times E(j))$.

F4. Se t > LIR*, faça LIR* = t.

F5. Se j < s, faça j = j + 1 e retorne para o passo F3. Caso contrário, os tempos de utilização efetiva de todos os segmentos já foram pesquisados, o algoritmo termina.

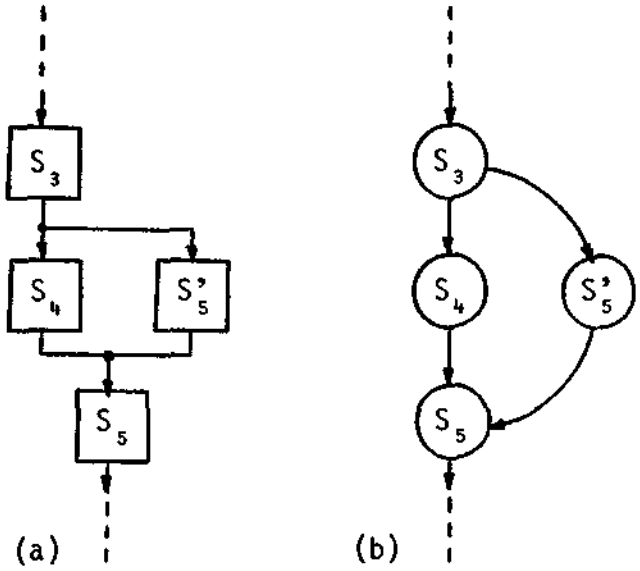
5.3 - ALTERAÇÃO DA ARQUITETURA ENCADEADA PARA A ELIMINAÇÃO DO USO DE SEGMENTOS EM ESPERA OCUPADA

A eliminação de espera ocupada na utilização dos segmentos de uma arquitetura encadeada é importante, não só para fazer o limitante inferior LMmin* se igualar ao limitante inferior real LIR* (veja-se a Seção 5.2), mas, como será visto na Seção 5.4, principalmente para alterar o grafo de fluxo de forma a fazer surgir um ciclo ótimo com latência constante e igual ao limitante inferior real LIR*.

O procedimento proposto para eliminar a espera ocupada consiste na inserção de segmentos atrasadores (mencionados no início deste capítulo) na arquitetura encadeada. Esses segmentos atrasadores funcionarão como elementos armazenadores temporários dos operandos gerados por um segmento, mas que não devem ainda ser transferidos para um

outro segmento da arquitetura encadeada, porque essa ação causaria a utilização desse último na condição de espera ocupada. Assim os segmentos originais da arquitetura passarão a ser utilizados apenas quando todos os operandos necessários para eles executarem as respectivas tarefas estiverem disponíveis. Por exemplo, na Figura 3.1 se for colocado um segmento atrasador S_5' na passagem de operandos do segmento S_3 para o segmento S_5, o tempo gasto por S_5 em espera ocupada pode ser eliminado, o que é feito na Figura 5.2.

O número de segmentos atrasadores necessários nesse processo de eliminação da espera ocupada de segmentos é f: um segmento atrasador colocado na entrada dos segmentos originais da arquitetura encadeada, para cada vez que cada segmento é utilizado. Por isso, o Algoritmo G apresentado a seguir, inicialmente cria f nós do grafo de fluxo associados a f segmentos atrasadores. O nó i + f é conectado ao grafo de fluxo, como antecessor do nó i do grafo, apenas se o segmento associado ao nó i é utilizado na condição de espera ocupada na arquitetura encadeada original. O procedimento usado no Algoritmo G é análogo ao procedimento de determinação da tabela de reserva de segmentos da Seção 4.1, com modificação apenas no tratamento de utlização de segmentos originais da arquitetura encadeada em espera ocupada, quando é feita a inserção de um segmento atrasador.



Seg.	(T)
S_1	1
⋮	⋮
S_3	2
S_4	3
S_5	1
⋮	⋮
S'_5	3

(c)

Seg.	T_1 ... T_4 T_5 T_6 T_7 T_8 ...
S_1	
⋮	
S_3	X
S_4	X X X
S_5	X
⋮	
S'_5	X X X

(d)

Fig. 5.2 - Eliminação do uso em espera ocupada o segmento S_5 da Figura 3.1, com o emprego do segmento atrasador S_5^2: a) fluxo de operandos pelos segmentos, b) grafo de fluxo, c) tabela de tempos de execução, d) tabela de reserva de segmentos.

Algoritmo G: alteração da arquitetura encadeada para a eliminação do uso de segmentos em espera ocupada. Baseado no grafo de fluxo e na tabela de tempos de execução representados na forma das matrizes E, F e D descritas na Seção 4.1, este algoritmo gera, após o seu término, o grafo de fluxo e tabela de tempos de execução modificados para que nenhum segmento original da arquitetura encadeada seja utilizado em espera ocupada. O grafo de fluxo e tabela de tempos de execução modificados serão representados na forma das matrizes E', F' e D', análogas às matrizes E, F e D originais, descritas a seguir:

Matriz E': $\{1, 2, \ldots, s + f\} \rightarrow I$, onde I é o conjunto dos números inteiros e $E'(i) = k \in I$ se e somente se k é o múltiplo inteiro da unidade de tempo T que o segmento S_i gasta para executar a sua tarefa. Os segmentos S_1 a S_s são os segmentos originais da arquitetura encadeada e os segmentos S_{s+1} a S_{s+f} são segmentos atrasadores.

Matriz F': $\{1, 2, \ldots, 2f\} \times \{1, 2, \ldots, 2f\} \rightarrow \{0, 1\}$, com elementos $F'(i, j) = 1$ se e somente se o nó j sucede o nó i no grafo de fluxo modificado. Os nós 1 a f são os próprios nós do grafo de fluxo original, e os nós f + 1 a 2f são os nós associados aos segmentos atrasadores S_{s+1} a S_{s+f}.

Matriz D': $\{1, 2, \ldots 2f\} \times \{1, 2, \ldots, s + f\} \rightarrow \{0, 1\}$, com elementos $D'(i, j) = 1$ se e somente se ao nó i do grafo de fluxo está associado o segmento S_j. Caso o nó $i + f$, $1 \leq i \leq f$, venha a ser inserido no grafo, o segmento atrasador S_{i+s} será o segmento associado a esse nó.

G1. Crie um vetor binário $Y = (y(f), \ldots, y(1))$, com f bits (Y será usado para armazenar o resultado da pesquisa dos nós do grafo de fluxo original que estão executando a sua tarefa).

G2. Crie um vetor binário Z = (z (f), ..., z (1)), com f bits (Z será usado para armazenar o resultado da pesquisa dos nós do grafo de fluxo original que ficam em espera ocupada).

G3. Crie uma matriz H com f linhas e f + 1 colunas, sendo os elementos H (i, j), i = 1, ..., f, j = 1, ..., f + 1, números inteiros (H será usada para determinar, em cada unidade de tempo T, quais nós do grafo original estão executando a sua tarefa, quais não estão sendo utilizados e quais estão em espera ocupada).

G4. Preencha a matriz E' inicialmente com zero em todos os seus elementos: E' (i) = 0, para i = 1, ..., s + f.

G5. Transfira para a matriz E' o conteúdo da matriz E: E' (i) = E (i) para i = 1, ..., s.

G6. Preencha a matriz F' inicialmente com zero em todos os seus elementos: F' (i, j) = 0, para i = 1, ..., 2f e j = 1, ..., 2f.

G7. Transfira para a matriz F' o conteúdo da matriz F: F' (i, j) = = F (i, j), para i = 1, ..., f e j = 1, ..., f.

G8. Preencha a matriz D' inicialmente com zero em todos os seus elementos: D' (i, j) = 0, para i = 1, ..., 2f e j = 1, ..., s+f.

G9. Transfira para a matriz D' o conteúdo da matriz D: D' (i, j) = = D (i, j), para i = 1, ..., f e j = 1, ..., s.

G10. Preencha a matriz H com o seu conteúdo inicial: para i = 1, ..., f, j = 1, ..., f, faça H (i, j) = -1, se F (i, j) = 0; caso contrário, F (i, j) = 1, faça H (i, j) = 1. Para i = 1, ..., f, faça $H(i, f+1) = \sum_{k=1}^{s} (E(k) \times D(i, k))$.

G11. Zere todos os bits dos vetores Y e Z: para i = 1, ..., f, faça z (i) = 0 e y (i) = 0.

G12. Faça j = 1 (j será usado para varrer as colunas da matriz H, na pesquisa dos nós do grafo que estão executando a sua tarefa).

G13. Se H (i, j) ≤ 0 para i = 1, ..., f, vá para o passo G14. Caso contrário, vá para o passo G15.

G14. Se H (j, f + 1) > 0, o segmento associado ao nó j está executando a sua tarefa, então faça y (j) = 1. Caso contrário, não faça nada.

G15. Se j < f, faça j = j + 1 e retorne para o passo G13. Caso contrário, j = f e todas as colunas da matriz H já foram pesquisadas, vá para o passo G16.

G16. Faça j = 1 (j será usado para varrer as colunas da matriz H na pesquisa dos nós do grafo cujos segmentos estão em espera ocupada).

G17. Faça i = 1 (i será usado para varrer as linhas da matriz H).

G18. Se H (i, j) < 0 e i < f, faça i = i + 1 e repita este passo. Se H (i, j) = 0 e i < f, faça i = i + 1 e vá para o passo G19. Se H (i, j) > 0 e i < f, faça i = i + 1 e vá para o passo G20. Caso contrário, i = f, vá para o passo G.22.

G19. Se H (i, j) ≤ 0 e i < f, faça i = i + 1 e repita este passo. Se H (i, j) > 0, vá para o passo G21. Caso contrário, H (i, j) ≤ 0 e i = f, vá para o passo G22.

G20. Se H (i, j) ≠ 0 e i < f, faça i = i + 1 e repita este passo. Se H (i, j) = 0, vá para o passo G21. Caso contrário, H (i, j) ≠ 0 e i = f, vá para o passo G22.

G21. Faça z (j) = 1 (pelo menos um nó antecessor do nó j já terminou de executar a sua tarefa, mas não todos os nós antecessores).

G22. Se j < f, faça j = j + 1 e retorne para o passo G17. Caso contrário, j = f e todas as colunas da matriz H já foram pesquisadas, vá para o passo G23.

G23. Para j = 1, ..., f: se z (j) = 1, então faça E' (s + j) = = E' (s + j) + 1 e, para i = 1, ..., f, se H (i, j) = 0, então faça F' (i, j) = 0, F' (i, j + f) = 1, F' (j + f, j) = 1, D' (j + f, s + j) = 1 (insira o segmento atrasador antes do nó j).

G24. Faça i = 1 (i será usado para varrer o vetor Y).

G25. Se y (i) = 1, faça H (i, f + 1) = H (i, f + 1) -1 e vá para o passo G26. Caso contrário, vá para o passo G27.

G26. Se H (i, f + 1) = 0, então para j = 1, ..., f se H (i, j) = 1, então faça H (i, j) = 0.

G27. Se i < f, faça i = i + 1 e retorne para o passo G25. Caso contrário, se i = f e todo o vetor Y foi pesquisado, vá para o passo G28.

G28. Se existe i tal que H (i, f + 1) > 0, retorne para o passo G11. Caso contrário, o algoritmo termina.

5.4 - DETERMINAÇÃO DA TABELA DE RESERVA DE SEGMENTOS MODIFICADA

A determinação da tabela de reserva de segmentos R', a partir do grafo de fluxo e da tabela de tempos de execução modificados (matrizes E', F' e D' geradas pelo Algoritmo G da seção anterior), pode ser feita usando um algoritmo análogo ao Algoritmo A descrito na Seção 4.1 (Algoritmo H descrito a seguir), mas com as seguintes modificações importantes:

a) O número máximo de colunas da matriz R' (qmax') deve ser calculado para os domínios dos índices das matrizes F' e D':

$$qmax' = \sum_{i=1}^{2f} \sum_{j=1}^{s+f} (D'(i, j) \times E'(j)). \qquad (5.5)$$

b) Diferentemente dos segmentos originais da arquitetura encadeada, os segmentos atrasadores inseridos pelo Algoritmo G (Seção 5.3) iniciam a "executar a sua tarefa" logo que eles recebam o primeiro operando.

Algoritmo H: determinação da tabela de reserva de segmentos modificada. Baseado na grafo de fluxo e na tabela de tempos de execução modificados pelo Algoritmo G e representados na forma das matrizes E', F' e D', este algoritmo gera, após o seu término, nas primeiras q' colunas da matriz R' (com s + f linhas e qmax' colunas) análoga à matriz R descrita na Seção 4.1, a tabela de reserva de segmentos (modificada) correspondente:

H1. Crie um vetor binário Y = (y (2f), ..., y (1)), com 2f bits (Y será usado para armazenar o resultado da pesquisa dos nós originais do grafo de fluxo modificado que estão executando a sua tarefa e dos nós referentes aos segmentos atrasadores que estão em espera ocupada).

H2. Crie um vetor binário Z = (z (f), ..., z (1)), com f bits (Z será usado para armazenar o resultado dos nós orginais do grafo de fluxo modificado que estão em espera ocupada).

H3. Crie uma matriz H' com 2f linhas e 2f + 1 colunas, sendo os elementos H' (i, j), i = 1, ..., 2f, j = 1, ..., 2f + 1, números inteiros (H' será usada para determinar em cada unidade de tempo quais nós do grafo não estão sendo utilizados, quais estão executando a sua tarefa e quais estão em espera ocupada).

H4. Faça todos os elementos da matriz R' iguais a zero: R' (i, j)=0, i = 1, ..., s + f, j = 1, ..., qmax' (inicialmente nenhum segmento da arquitetura encadeada foi utilizado).

H5. Faça q' = 1 (q' é o ponteiro de tempo que auxiliará no preenchimento da matriz R').

H6. Preencha a matriz H' com o seu conteúdo inicial: para i = 1, ..., 2f e j = 1, ..., 2f faça H' (i, j) = -1 se F' (i, j) = 0; caso contrário, F' (i, j) = 1, faça H' (i, j) = 1. Para i = 1, ..., 2f, faça $H'(i, 2f+1) = \sum_{k=1}^{s+f} (E'(k) \times D'(i, k))$.

H7. Zere todos os bits dos vetores Y e Z: para i = 1, ..., f, faça y (i) = z (i) = 0; para i = 1 + f, ..., 2f faça y (i) = 0.

H8. Faça j = 1 (j será usado para varrer as colunas da matriz H' na pesquisa dos nós do grafo modificado que estão executando a sua tarefa).

H9. Se H' (i, j) ≤ 0 para todo i = 1, ..., 2f, vá para o passo H10. Caso contrário, vá para o passo H11.

H10. Se H' (j, 2f + 1) > 0, o segmento associado ao nó j está executando a sua tarefa, então faça y (j) = 1.

H11. Se $j < f$, faça $j = j + 1$ e retorne para o passo H9. Caso contrário, $j = f$ e todos os segmentos originais já foram pesquisados, vá para o passo H12.

H12. Faça $j = 1$ (j será usado para varrer as colunas da matriz H na pesquisa dos nós do grafo cujos segmentos estão em espera ocupada).

H13. Faça $i = 1$ (i será usado para varrer as linhas da matriz H).

H14. Se $H'(i, j) < 0$ e $i < 2f$, faça $i = i + 1$ e repita este passo. Se $H'(i, j) = 0$ e $i < 2f$, faça $i = i + 1$ e vá para o passo H15. Se $H'(i, j) > 0$ e $i < 2f$, faça $i = i + 1$ e vá para o H16. Caso contrário, $i = 2f$, vá para o passo H18.

H15. Se $H'(i, j) \leq 0$ e $i < 2f$, faça $i = i + 1$ e repita este passo. Se $H'(i, j) > 0$, vá para o passo H17. Caso contrário, $H'(i, j) \leq 0$ e $i = 2f$, vá para o passo H18.

H16. Se $H'(i, j) \neq 0$ e $i < 2f$, faça $i = i + 1$ e repita este passo. Se $H'(i, j) = 0$, vá para o passo H17. Caso contrário, $H'(i, j) \neq 0$ e $i = 2f$, vá para o passo H18.

H17. Faça $z(j) = 1$ (pelo menos um nó antecessor do nó j já terminou de executar a sua tarefa, mas não todos os nós antecessores).

H18. Se $j < f$, faça $j = j + 1$ e retorne para o passo H13. Caso contrário, $j = f$ e todas as colunas da matriz H já foram pesquisadas, vá para o passo H19.

H19. Faça $j = f + 1$ (j será usado para varrer as colunas da matriz H' na pesquisa dos nós do grafo em espera ocupada).

H20. Se existe i tal que H' (i, j) = 0 e H' (j, 2f+1) > 0, então faça y (j) = 1 (o segmento atrasador associado ao nó j está "executando a sua tarefa").

H21. Se j < 2f, faça j = j + 1 e retorne para o passo H20. Caso contrário, j = 2f e todas as colunas da matriz H' (i, j) já foram pesquisadas, vá para o passo H22.

H22. Para i = 1, ..., f, se y (i) = 1 ou z (i) = 1 procure k tal que D' (i, k) = 1 e faça R' (k, q') = 1. Para i = 1 + f, ..., 2f, se y (i) =1, procure k tal que D'(i, k) = 1 e faça R'(k, q') = 1

H23. Para i = 1, ..., 2f, se y (i) = 1 faça H' (i, 2f + 1) = = H' (i, 2f + 1) - 1; se H' (i, 2f + 1) recebeu o valor zero, então para j = 1, ..., 2f, se H'(i, j) > 0 faça H'(i, j) = 0.

H24. Se existe i tal que H'(i, 2f + 1) > 0, faça q' = q' + 1 e retorne para o passo H7. Caso contrário, já se encontra armazenada na matriz R' a tabela de reserva de segmentos procurada, o algoritmo termina.

5.5 - ALTERAÇÃO DA ARQUITETURA ENCADEADA PARA UM CICLO ÓTIMO APRESENTAR LATÊNCIA CONSTANTE IGUAL AO LIMITANTE INFERIOR REAL

Se a latência média do ciclo ótimo for igual ao seu limitante inferior real, então pelo menos um segmento da arquitetura encadeada estará sendo utilizado efetivamente (sem espera ocupada) 100% no tempo. Neste caso, o aumento do número de execuções do algoritmo, no tempo, só poderá ser obtido através da duplicação dos segmentos com fator de utlização igual a 100%; ou tornando esses segmentos mais rápidos na execução das suas tarefas; ou usando técnicas de encadeamento, particionando as tarefas desses segmentos em várias subtarefas, possuindo cada nova subtarefa um segmento a ela dedicado.

Porém, geralmente, uma primeira proposta de arquitetura encadeada não apresenta a latência média do ciclo ótimo igual ao seu limitante inferior, e muito menos, esse limitante inferior igual ao limitante inferior real. Como então, realizando apenas as alterações mínimas descritas no início deste capítulo, modificar uma arquitetura encadeada, caracterizada apenas pelo seu grafo de fluxo e tabela de tempos de execução, de forma a fazer com que os seus segmentos tenham o maior fator de utilização possível e, com isso, seja atingido o número máximo de execuções concorrentes do algoritmo?

Shar (1972) demonstrou que a latência média mínima de uma arquitetura encadeada pode ser reduzida ao seu limitante inferior, com o uso de segmentos atrasadores (Teorema 3.10 da referência citada). Essa demonstração foi feita provando que é sempre possível modificar a tabela de reserva de segmentos, com a inserção de segmentos atrasadores, de forma a que um ciclo tenha latência constante ℓc igual ao limitante inferior: $\ell c = LMmin^*$. A idéia básica resume-se em forçar o vetor de colisão C, determinado pela tabela de reserva de segmentos da arquitetura encadeada modificada, a ter zeros (ou seja, latências permitidas) nos bits ℓc, $2\ell c$, $3\ell c$, ... Para isso, a distência entre quaisquer dois X de uma mesma linha da tabela de reserva de segmentos modificada não pode ser nunca múltiplo inteiro de ℓc. E o ciclo (ótimo, porque $\ell c = LMmin^*$) com latência constante igual a ℓc existirá no grafo de latências permitidas gerado pela tabela. O procedimento para modificar a tabela de reserva de segmentos proposto por Shar (1972) é o seguinte:

a) Remova qualquer uso paralelo de segmentos, com a inserção de segmentos atrasadores, para que apenas um segmento original seja usado em cada unidade de tempo.

b) Percorrendo a tabela de reserva de segmentos, da esquerda para a direira, verifique se os X existentes, à esquerda do X da coluna sendo analisada, não estão localizados a distâncias múltiplos inteiros de LMmin*. Se isso ocorrer, desloque toda a par

te direita da tabela, a partir da coluna sendo analisada, de uma unidade de tempo para a direita e insira naquele ponto um segmento atrasador. No final do percorrimento da tabela não haverá mais nenhum par de X distanciados de ℓc, $2\ell c$, $3\ell c$, ...

Entretanto, por se basear apenas na informação contida na tabela de reserva de segmentos e não no grafo de fluxo e na tabela de tempos de execução da arquitetura encadeada, Shar (1972) não levou em conta no seu procedimento que (veja-se Seção 3.2):

a) Alguns X da tabela de reserva de segmentos podem estar relacionados com a utilização de segmentos em espera ocupada. Isso pode fazer com que um ciclo ótimo possa apresentar latência constante igual ao seu limitante inferior, mas não ao seu limitante inferior real (veja-se a Seção 5.2).

b) Alguns X juntos em uma mesma linha da tabela de reserva de segmentos podem estar relacionados com um único uso de segmento, o que torna esses X inseparáveis, porque em uma arquitetura encadeada, uma vez iniciada a execução da sua tarefa, um segmento não pode ser interrompido.

Devido a isso, será então proposto a seguir um procedimento que altera o grafo de fluxo de uma arquitetura encadeada, de forma a criar um ciclo ótimo com latência constante igual ao limitante inferior real LIR*, usando para tal apenas a inserção de segmentos atrasadores e respeitando a continuidade dos tempos de execução dos segmentos.

Considere-se que nenhum segmento da arquitetura encadeada seja utilizado em espera ocupada. Com isso, o limitante inferior para a latência média do ciclo ótimo é igual a LIR*. Para que um ciclo (no caso ótimo) com latência constante ℓc = LIR* exista no grafo de latências permitidas dessa arquitetura encadeada, basta que o vetor de colisão gerado pela sua tabela de reserva de segmentos apresente zeros

nas posições ℓc, $2\ell c$, ... Isto pode ser obtido se, ao construir a tabela de reserva de segmentos a partir do grafo de fluxo e da tabela de tempos de execução da arquitetura encadeada, for verificado, antes da colocação dos X consecutivos em uma linha da tabela, referentes à execução completa da tarefa de um segmento, a distância entre algum desses X com os X já existentes nessa linha da tabela é múltiplo inteiro de ℓc. Se isto ocorrer, basta então inserir um segmento atrasador, antes do segmento em questão, de forma a evitar esta situação.

Mas, é sempre possível, com a inserção de segmentos atrasadores, forçar que a distância entre dois X quaisquer da tabela de reserva de segmentos de uma arquitetura encadeada não seja nunca múltiplo inteiro de ℓc = LIR*?

Considere-se que o segmento S_r da arquitetura encadeada gaste p unidades de tempo para executar a sua tarefa (esta informação é obtida da tabela de tempos de execução). Ao utilizar o segmento S_r, isto implica a colocação de p X consecutivos na linha r da tabela de reserva de segmentos. Como $p \leq \ell c$ (de acordo com a definição de LIR*) então a distância entre quaisquer desses p X consecutivos é sempre menor do que ℓc e, portanto, diferente de ℓc. Resta estudar as distâncias entre os X referentes a mais de uma utilização do segmento S_r.

Considere-se que a primeira utilização do segmento S_r ocorra a partir da unidade de tempo T_{q_1}. Com isto, se não é desejado que a distância entre quaisquer dois X da linha r da tabela de reserva de segmentos seja múltiplo de ℓc, fica então proibido um outro uso do segmento S_r nas unidades de tempo T_i tais que $i > 0$ $(i \bmod \ell c) \in Q_1 = \{(q_1 + k) \bmod \ell c$, com $0 \leq k \leq (p - 1)\}$. Considere-se agora que a segunda utilização do segmento S_r devesse se iniciar na unidade de tempo T_{q_2} $(q_2 > q_1)$. Inserindo um atraso de ℓ_2 unidades de tempo é possível fazer essa segunda utilização de segmento S_r dar-se a partir da unidade de tempo T_{q_2}, com $q_2 + \ell_2 = q_1 + p + (n \times \ell c)$. Com isto, fica também

proibido o uso do segmento S_r nas unidades de tempo T_i tais que $i > 0$ e $(i \bmod \ell c) \in Q_2 = \{(q_2 + \ell_2 + k) \bmod \ell c$, com $0 \leq k \leq (p - 1)\}$. Substituindo $q_2 + \ell_2$ por $q_1 + p + (n \times \ell c)$ na definição de Q_2 acima, obtém-se $Q_2 = \{(q_2 + k) \bmod \ell c$ tal que $p \leq k \leq (2p - i)\}$.

Como por definição $2p \leq \ell c$, então a interseção entre os conjuntos Q_1 e Q_2 é um conjunto vazio.

Após a segunda utilização do segmento S_r fica então proibido o uso do segmento S_r nas unidades de tempo T_i tais que $i > 0$ e $(i \bmod \ell c) \in Q_2' = \{(q_1 + r) \bmod \ell c$, com $0 \leq k \leq (2p - 1)\}$.

Seja n o número de vezes que o segmento S_r é utilizado na execução do algoritmo implementado. Como $n \times p \leq \ell c = LIR^*$, então sempre vão existir latências $\ell_2, \ell_3, \ldots, \ell_n$ que geram um conjunto $Q_n' = \{(q_1 + k) \bmod \ell c$, com $0 \leq k \leq (np - 1)\}$ que possui np elementos diferentes.

Como, aplicando o Algoritmo G descrito na Seção 5.3, pôde-se eliminar o uso de segmentos de uma arquitetura encadeada na condição de espera ocupada, pode-se então generalizar e concluir que é sempre possível, apenas com a utilização de segmentos atrasadores, forçar que a distância entre dois X quaisquer da tabela de reserva de segmentos de uma arquitetura encadeada não seja nunca múltiplo inteiro de $\ell c = LIR^*$. E, com isto, pode-se fazer surgir um ciclo ótimo igual ao limitante inferior real LIR^*.

O procedimento descrito anteriormente nesta seção é apresentado na forma do Algoritmo I. Esse algoritmo considera como dados de entrada o grafo de fluxo e a tabela de reserva de execução modificados, já armazenados nas matrizes E', F' e D' como geradas pelo Algoritmo G da Seção 5.3. Ele é análogo ao procedimento para determinação da tabela de reserva de segmentos da Seção 5.4, com modificação apenas no tratamento da utilização de um segmento original da arquitetura encadeada, pelo algoritmo implementado.

Algoritmo I: alteração da arquitetura encadeada para um ciclo ótimo apresentar latência constante igual ao limitante inferior real. Este algoritmo modifica o conteúdo das matrizes E', F' e D' (geradas pelo Algoritmo G da Seção 5.3) para que nenhum par de X da tabela de reserva de segmentos, referentes ao uso de segmentos originais da arquitetura encadeada, fiquem distanciados de $\ell c = LIR^*$ unidades de tempo. Note-se que LIR* é obtido com o Algoritmo E da Seção 5.2. Além do que este algoritmo também deixa armazenadas na matriz Q, descrita a seguir, as unidades de tempo que foram proibidas para o uso dos segmentos originais da arquitetura encadeada.

Matriz Q: $\{1, 2, \ldots, s\} \times \{0, 1, \ldots, lc - 1\} \rightarrow \{0, 1\}$ com elementos $Q(i, j) = 1$ se e somente se nas unidades de tempo T_k, $k > 0$ e $(k \bmod \ell c) = j$, for proibido usar o segmento S_i. Q será usada para determinar quando se pode ou não utilizar novamente um segmento da arquitetura encadeada original.

I1. Crie um vetor binário $Y = (y(2f), \ldots, y(1))$, com 2f bits (Y será usado para armazenar o resultado da pesquisa dos nós do grafo de fluxo modificado que estarão executando a sua tarefa ou que estão em espera ocupada).

I2. Crie uma matriz H' com 2f linhas e 2f + 1 colunas, sendo os elementos $H'(i, j)$, $i = 1, \ldots, 2f$, $j = 1, \ldots, 2f + 1$, números inteiros (H' será usada para determinar em cada unidade de tempo quais nós do grafo não estão sendo utilizados, quais estão executando a sua tarefa e quais estão em espera ocupada.

I3. Faça inicialmente todos os elementos da matriz Q iguais a zero: $Q(i, j) = 0$, $i = 1, \ldots, s$, $j = 0, \ldots, \ell c - 1$.

I4. Preencha a matriz H' com o seu conteúdo inicial: para $i = 1, \ldots, 2f$ e $j = 1, \ldots, 2f$ faça $H'(i, j) = -1$, se $F'(i, j) = 0$; caso contrário, $F'(i, j) = 1$, faça $H'(i, j) = 1$. Para $i = 1. \ldots, 2f$, faça $H'(i, 2f + 1) = \sum_{k=1}^{s+f} (E'(k) \times D'(i, k))$.

I5. Faça q = 1 (q será usado como ponteiro de tempo).

I6. Zere todos os bits do vetor Y: para i = 1, ..., 2f, faça y (i) =0.

I7. Faça j = 1 (j será usado para varrer as colunas da matriz H' na pesquisa dos nós do grafo modificado que estão executando a sua tarefa).

I8. Se H' (i, j) ≤ 0 para i = 1, ..., 2f, vá para o passo I9. Caso contrário, vá para o passo I10.

I9. Se H' (j, 2f + 1) > 0, o segmento associado ao nó j está executando a sua tarefa, então faça y (j) = 1.

I10. Se j < f, faça j = j + 1 e retorne para o passo I8. Caso contrário, j = f e todos os segmentos originais já foram pesquisados, vá para o passo I11.

I11. Faça j = f + 1 (j será usado para varrer as colunas da matriz H' na pesquisa dos nós do grafo em espera ocupada).

I12. Se existe i tal que H' (i, j) = 0 e H' (j, 2f + 1) > 0, então faça y (j) = 1 (o segmento atrasador associado ao nó j está "executando a sua tarefa").

I13. Se j < 2f, faça j = j + 1 e retorne para o passo I12. Caso contrário, j = 2f e todas as colunas da matriz H' já foram pesquisadas, vá para o passo I14.

I14. Faça i = j (i será usado para varrer o vetor Y).

I15. Se y (i) = 1, faça p = H'(i, 2f + 1), procure k tal que D' (i,k) = 1 e vá para o passo I16. Caso contrário, vá para o passo I19.

I16. Se Q (k, ℓ) = 0, para ℓ = (q + r) mod ℓc, com r = 0, ..., p - 1, então vá para o passo I17. Caso contrário, o uso do segmento S_k está proibido nas p unidades de tempo a partir de T_q, vá para o passo I18.

I17. Faça Q (k, ℓ) = 1, para ℓ = q mod ℓc e H' (i, 2f + 1) = = H' (i, 2f + 1) - 1. Se H' (i, 2f + 1) recebeu o valor zero, então, para j = 1, ..., 2f, se H' (i, j) > 0 faça H' (i, j) =0. Vá para o passo I19.

I19. Para j = 1, ..., f, se H' (j, i) = 0, faça F' (j, i) = 0; F' (j, i + f) = 1; F' (i + f, i) = 1; D' (i + f, s + i) = 1; E' (s + i) = E' (s + i) + 1.

I19. Se i < f, faça i = i + 1 e retorne para o passo I15. Caso contrário, vá para o passo I20.

I20. Para i = f + 1, ..., 2f, se y (i) = 1, então faça H' (i, 2f + 1) = H' (i, 2f + 1) - 1. Se H' (i, 2f + 1) recebeu o valor zero, então, para j = 1, ..., 2f, se H' (i, j) > 0 faça H' (i, j) = 0.

I21. Se existe i tal que H' (i, 2f + 1) > 0, faça q = q + 1 e retorne para o passo I6. Caso contrário, o algoritmo termina.

5.6 - ELIMINAÇÃO DOS SEGMENTOS ATRASADORES DESNECESSÁRIOS

O procedimento desenvolvido na seção anterior requer, para sua aplicação, que nenhum segmento original da arquitetura encadeada seja utilizado em espera ocupada. Entretanto, uma análise posterior da arquitetura encadeada modificada, gerada pelos Algoritmos G e I (Seções 5.4 e 5.5), pode revelar que alguns dos segmentos atrasadores inseridos por esses algoritmos são perfeitamente dispensáveis. Os segmentos atrasadores dispensáveis (portanto, desnecessários) são aqueles

que, uma vez desconectados da arquitetura encadeada modificada, levarão ao uso de segmentos originais em espera ocupada, mas em unidades de tempo tais que não causarão o desaparecimento do ciclo ótimo (de iniciações do algoritmo implementado) com latência constante igual ao limitante inferior real.

Considere-se a matriz de reserva de segmentos R' com s + f linhas e qmax' colunas gerada pelo Algoritmo H (Seção 5.4) a partir da arquitetura encadeada modificada representada pelas matrizes E', F' e D' geradas pelo Algoritmo I da seção anterior.

Suponha-se que em R' o segmento atrasador S_{s+a} ($f \leq a \leq f$) seja utilizado nas unidades de tempo T_q, T_{q+1}, ..., T_{q+k-1} ($k > 0$) e que S_{s+a} esteja associado a um nó do grafo de fluxo modificado, cujo nó sucessor está associado ao segmento original S_b. Se, observando a matriz Q (também gerada pelo Algoritmo I da seção anterior), T_q, T_{q+1}, ..., T_{q+k-1} forem unidades de tempo não-proibidas para o uso de segmento S_b, então o segmento atrasador S_{s+a} pode ser desconectado da arquitetura encadeada modificada, já que o ciclo ótimo com latência constante ℓc, igual ao seu limitante inferior real continuará existindo. Obviamente, as unidades de tempo T_i tais que $i > 0$ e $(i \bmod \ell c) \in \{(q + j) \bmod \ell c$, com $0 \leq j \leq k - 1\}$, passam a ficar proibidad para um novo uso de segmento S_b.

Esse procedimento é descrito a seguir.

Algoritmo J: eliminação dos segmentos atrasadores desnecessários. Baseado na matriz de reserva de segmentos R' da arquitetura encadeada modificada gerada pelo Algoritmo I e representada na forma das matrizes E', F' e D', e na matriz Q também gerada pelo Algoritmo I, este algoritmo elimina (desconecta) da arquitetura encadeada os segmentos atradadores desnecessários:

J.1 Faça i = 1 (i será usado para varres as linhas da matriz E' em busca dos segmentos atrasadores conectados à arquitetura encadeada).

J2. Se E' (s + i) = 0, vá para o passo J9. Caso contrário, E' (s + i) > 0 e o segmento atrasador S_{s+i} está conectado à arquitetura encadeada, vá para o passo J3.

J3. Faça k = E' (s + i) (k é o número de unidades de tempo que o segmento atrasador S_{s+i} é utilizado).

J4. Procure o menor número q tal que R'(s + i, q) = 1 (q é a unidade de tempo a partir da qual o segmento atrasador S_{s+i} é utilizado).

J5. Procure j tal que D'(i, j) = 1 (o segmento original S_j é quem recebe os "resultados" do segmento atrasador S_{s+i}).

J6. Se Q (j, ℓ) = 0, para ℓ = (q + r) mod ℓc, com r = 0, ..., k - 1, vá para o passo J7. Caso contrário, o uso do segmento S_j está proibido nas k unidades de tempo a partir de T_q, vá para o passo J9.

J7. Faça: E'(s + i) = 0; D'(i + f, s + i) = 0; para ℓ = 1, ..., 2f, se F' (ℓ, f + i) = 1 então faça: F'(ℓ, f + i) = 0, F'(f + i, i) = 0 e F'(ℓ, i) = 1.

J8. Faça Q (j, ℓ) = 1 para ℓ = (q + r) mod ℓc, com r = 0, ..., k - 1.

J9. Se i < f, faça i = i + 1 e retorne para o passo J2. Caso contrário, todos os segmentos atrasadores já foram examinados, o algoritmo termina.

5.7 - OBSERVAÇÕES SOBRE OS PROCEDIMENTOS DESENVOLVIDOS

O Algoritmo I (Seção 5.5) modifica uma arquitetura encadeada para que apareça um ciclo ótimo com latência constante igual ao limitante inferior LMmin* (feito igual ao limitante inferior real LIR* usando anteriormente o Algoritmo G), mas às custas de um aumento do

atraso da arquitetura encadeada, que é o tempo gasto em uma execução completa do algoritmo implementado, do seu passo inicial até o seu passo final. Uma característica funcional de um PDTR (veja-se a Seção 2.1) é de que a sua capacidade de processar os elementos de sequência de entrada, na sua real taxa de chegada, é mais importante do que o atraso fixo existente no processador. Sob esse aspecto, os métodos empregados no Algoritmo I justificam plenamente o objetivo atingido. Note-se ainda que um ciclo constante de iniciações do algoritmo implementado traz consigo vantagens na implementação do PDTR, pois, torna desnecessário o uso de "buffers" (veja-se a Seção 3.5) na entrada e saída do PDTR; "buffers" esses que seriam empregados para compatibilizar uma possível frequência irregular de iniciações do algoritmo, com o fluxo constante de chegada e de saída de dados do processador.

Após a modificação da arquitetura encadeada pelos Algoritmos G, I e J (Seções 5.3, 5.5 e 5.6) pode resultar que alguns segmentos atrasadores inseridos na arquitetura encadeada original apresentem tempo de execução maior do que o limitante inferior real LIR*. Como nenhum segmento da arquitetura encadeada modificada final pode apresentar tempo de execução maior do que LIR*, e os segmentos atrasadores são utilizados apenas uma vez na execução do algoritmo implementado, então a solução deste problema é simples e consiste na substituição de cada um desses segmentos por tantos segmentos atrasadores quanto necessários, com tempo de execução menor ou igual a LIR* e colocados em cascata na arquitetura encadeada.

A eliminação dos segmentos atrasadores efetuada pelo Algoritmo J foi realizada pesquisando os segmentos atrasadores sequencialmente, sem considerar objetivos de eliminação do número máximo de segmentos atrasadores da arquitetura encadeada modificada. Uma abordagem do problema com esse enfoque teria de levar em conta qual a ordem mais conveniente para a eliminação de segmentos atrasadores, além de considerar a lateração mínima citada no início do Capítulo 5, mas não utilizada nos procedimentos desenvolvidos neste trabalho, que é a piora no desempenho de um segmento com o aumento do tempo de execução da sua tare

fa. Isto corresponderia a retardar a passagem dos resultados obtidos para o segmento a que eles se destinam. Entretanto é importante observar que isto poderia levar a tempos de execução diferentes para cada utilização do segmento em questão, o que exigiria uma modelagem do fluxo de operandos, através dos segmentos de uma arquitetura encadeada, distinta da realizada inicialmente na Seção 3.1.

Vale notar que alguns dos f segmentos atrasadores criados pelo Algoritmo G (Seção 5.3), após a aplicação do Algoritmo J (Seção 5.6), ainda se encontram desconectados da arquitetura endadeada. Estes segmentos correspondem a linhas na matriz E' com valores iguais a zero (tempo de execução nulo); linhas e colunas na matriz F' com todos os elementos iguais a zero (nó do grafo de fluxo sem antecessor ou sucessor); e linhas e colunas da matriz D' com todos os elementos iguais a zero (o nó permanece sem nenhum segmento a ele associado). Devido a isso, as matrizes E', F' e D' que representam a arquitetura encadeada modificada, divergem um pouco da definição das matrizes E, F e D originais feita na Seção 4.1, no que diz respeito apenas aos segmentos atrasadores. Mas, com a remoção das linhas e das colunas referentes aos segmentos atrasadores não-utilizados das matrizes E', F' e D', essa divergência seria eliminada.

5.8 - CONSIDERAÇÕES PARA O CASO DE A SEQUÊNCIA DE ENTRADA NÃO SER INFINITA

Na Seção 3.7 deduziu-se que o tempo total de processamento, TP (n), para n execuções do algoritmo implementado em uma arquitetura encadeada, com os inícios de execução do algoritmo programados para ocorrer de acordo com um ciclo ótimo com latências $\ell_1, \ell_2, \ldots, \ell_3$, é:

$$TP(n) = [(n - 1) \text{ div } m] \times m \times LM + \sum_{i=1}^{f} \ell_i + d, \qquad (3.6)$$

onde $f = (n - 1) \bmod m$ e LM é a latência média do ciclo $\ell_1, \ell_2, \ldots, \ell m$.

Consisdere-se como exemplo um ciclo com latência constante ℓc (m = 1 na Equação 3.6), o que leva a:

$$TP(n) = (n - 1) \times \ell c + d. \qquad (5.6)$$

Observe-se, na Figura 5.3, o que ocorre com o tempo total de processamento (expresso pela Equação 5.6), se a latência ℓc for diminuída às custas de um aumento do atraso d. Caso n não seja suficientemente grande ($> n_c$), a redução de TP (n) devido a um maior número de iniciações do algoritmo implementado no tempo pode ser comprometida com o aumento no tempo total de execuções do algoritmo.

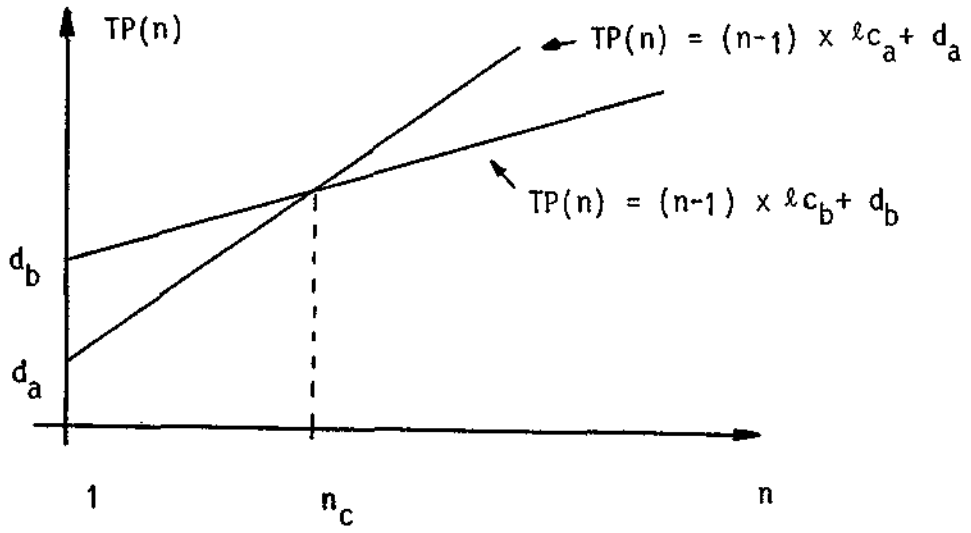


Fig. 5.3 - Efeito no tempo total de processamento quando $\ell c_b < \ell c_a$, $d_b > d_a$.

Portanto, se o número de elementos da sequência de entrada a ser submetida a um PDTR construído com arquitetura encadeada não for suficientemente grande (para ser considerado infinito), o uso do procedimento apresentado neste capítulo deve ser acompanhado de uma análise sobre os seus efeitos no tempo total de processamento da sequência

de entrada. Se o número n de elementos dessa sequência for fixo a análise é simples e basta calcular TP (n) para a arquitetura original e para a arquitetura encadeada modificada. No caso de n ser indeterminado, sugere-se uma análise com base no valor esperado para o número n determinado por processos estocásticos.

CAPÍTULO 6

IMPLEMENTAÇÃO DE PDTRs COM ARQUITETURA ENCADEADA

Neste capítulo discute-se inicialmente a organização interna de um PDTR (Seção 6.1) e a seguir, baseado nos procedimentos desenvolvidos nos Capítulos 4 e 5, aborda-se uma metodologia de implementação de PDTRs com arquitetura encadeada (Seção 6.2). O problema do equalizador radiométrico de imagens e o do decodificador de Viterbi, descritos no Apêndice B, são utilizados, na Seção 6.3, como exemplo de aplicação da metodologia de implementação da Seção 6.2.

6.1 - ORGANIZAÇÃO INTERNA DO PROCESSADOR

Um PDTR é um sistema que implementa algoritmos e como tal, a sua organização interna pode ser decomposta em duas partes (Figura 6.1): uma parte de controle, representada por um autômato de Moore, e uma parte operativa, representada por um autômato de Mealy (Gluskhov and Letchevskii, 1969).

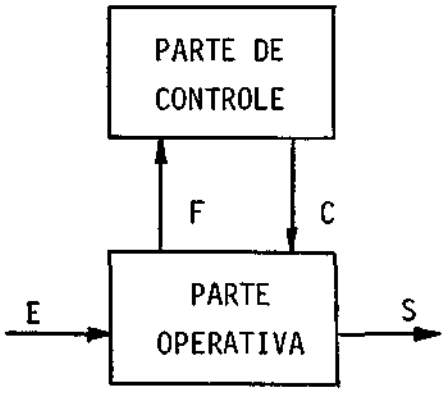
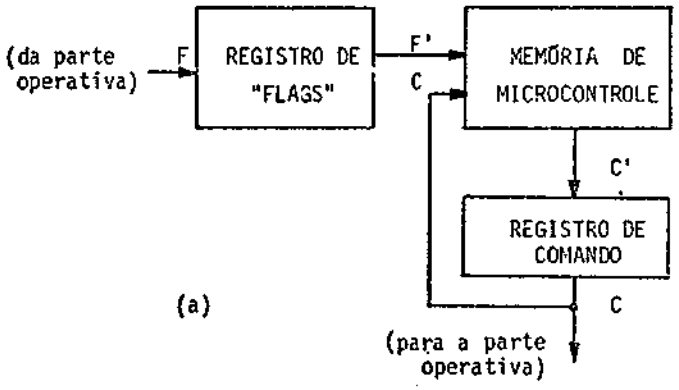


Fig. 6.1 - Partes que compõem a orgranização interna de um PDTR.

A parte de controle é responsável pela geração, na sequência correta, dos comandos (C) enviados para a parte operativa e necessários para a execução do algoritmo implementado no PDTR. Como o uso dos métodos desenvolvidos nos Capítulos 4 e 5 encontram terreno mais fértil para aplicação na parte operativa, então a arquitetura simples da Figura 6.2a será utilizado na parte de controle dos exemplos. Nela, o próximo comando é determinado pelo comando atualmente sendo executado (armazenado no registro de comando) e pelos "flags" (F) oriundos da parte operativa na execução do comando anterior (armazenados no registro de "flags"). A lógica de determinação do próximo comando é realizada por uma memória de microcontrole, o que caracteriza uma parte de controle microprogramada. Maiores detalhes sobre esse tipo de arquitetura podem ser obtidos em Lemos Filho (1982) e White (1981).



(a)

RECURSO (SEGMENTO) UTILIZADO	T_1	T_2
Memória de micro controle	X	
Registro de comando		X

(b)

Fig. 6.2 - Parte de controle de um PDTR: a) arquitetura, b) tabela de reserva de segmentos para a busca e execução de um comando.

Ressalte-se que a arquitetura da Figura 6.2a é uma arquitetura encadeada linear (veja-se a Figura 6.2b), dando-se o acesso à memoria de microcontrole, na busca do próximo comando a ser executado, em paralelo com a execução do comando atual, armazenado no registro de comando. Esta arquitetura serve, também, como exemplo do uso de técnicas de encadeamento no mecanismo de busca e execução de instruções (comandos) em um PDTR, que é um dos níveis de paralelismo mencionado na Seção 2.2.

Um comando fica válido para ser executado pela parte operativa durante uma janela de tempo T igual à unidade de tempo definida na Seção 3.1. Os comandos possuem formato único, composto de s campos C_i, i = 1, ..., s, onde s é o número de segmentos da arquitetura encadeada da parte operativa do processador (veja-se a Figura 6.3). A fatia do comando C, referente ao campo C_i, é chamada microoperação e deve ser executada pelo segmento S_i na janela de tempo T.

O campo C_i compõe-se de dois tipos de subcampos. O primeiro deles (C_{ia} e C_{ib} na Figura 6.3) atua sobre os elementos que guardam os operandos recebidos dos outros segmentos da arquitetura encadeada. O outro tipo de subcampo (C_{ic} na Figura 6.3) informa ao elemento transformador que ele deve iniciar a executar a sua tarefa e, se for necessária mais de uma unidade de tempo para isso, ele deve continuar a executar a sua tarefa, além de eventualmente transferir alguma informação necessária para a execução da tarefa pelo elemento transformador.

A chegada de pelo menos um operando faz o segmento aguardar a chegada dos outros operandos restantes (supondo que isto ocorra), o que já o compromete na execução futura da tarefa e gera sua utilização na situação de espera ocupada mencionada na Seção 3.2. Entretanto, não é raro que um segmento, para guardar os operandos recebidos e executar completamente a sua tarefa, necessite de apenas uma unidade de tempo.

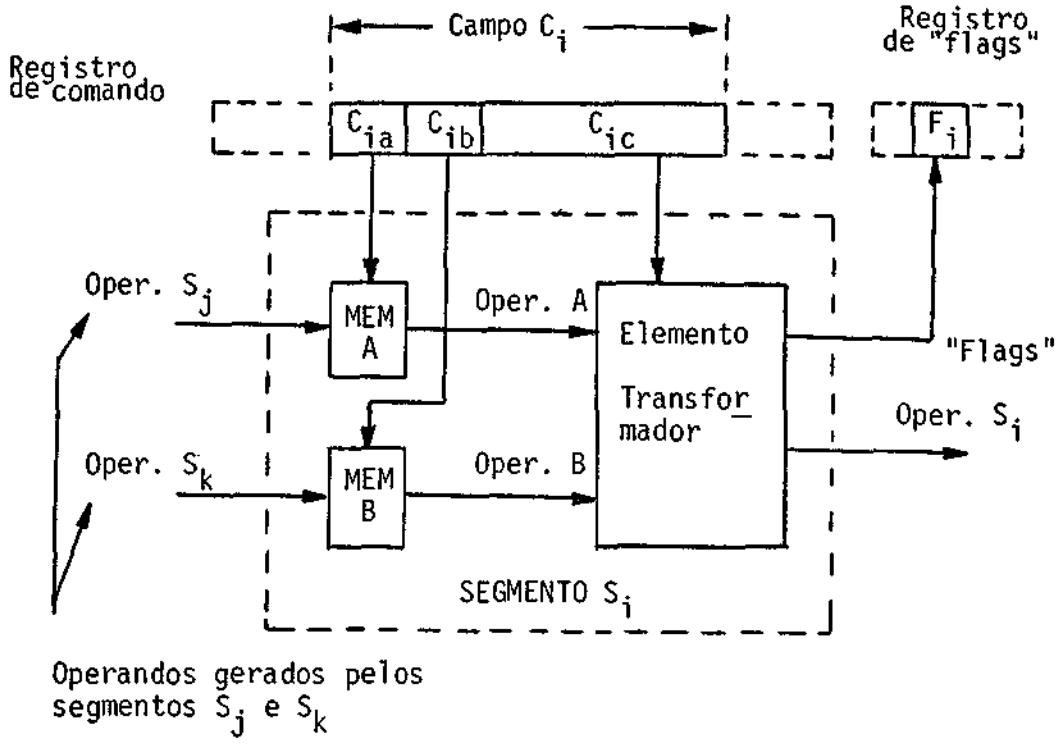


Fig. 6.3 - Parte operativa: representação de um segmento.

Os casos práticos onde o elemento transformador começa a executar a sua tarefa apenas com parte do total de operandos para ele realizá-la completamento podem ser reduzidos ao modelo exposto na Figura 6.3.

Após enviar para um segmento a microoperação (ou sequência de microoperações) referentes a sua utilização, a parte de controle considera que os operandos (resultados) gerados pelo segmento encontram-se válidos e disponíveis na sua saída, isto sem nenhum envio de sinal do segmento para a parte de controle. Note-se que os "flags" eventualmente gerados pelos segmentos referem-se exclusivamente aos resultados obtidos pelo elemento transformador e não ao término de execução de uma tarefa.

As alterações mínimas mencionadas no Capítulo 5 correspondem na Figura 6.3 a:

a) segmento atrasador: segmento sem elemento transformador, onde os operandos ficam disponíveis na sua saída, imediatamente após terem sido recebidos, já que não existe nenhuma transformação sobre esses operandos;

b) Piora no desempenho de um segmento: envio, pela parte de controle, de mais microoperações do tipo "continue a executar a sua tarefa", do que o efetivamente necessário para tal.

6.2 - METODOLOGIA DE IMPLEMENTAÇÃO

Nessa seção, aborda-se uma metodologia de implementação da parte operativa de um PDTR com o uso de técnicas de encadeamento, de forma que ela apresenta um ciclo de iniciações do algoritmo implementado com latência compatível com a frequência de chegada de dados ao PDTR. A parte de controle do PDTR supõe-se ser a apresentada na seção anterior.

Essa metodologia utiliza os resultados obtidos nos Capítulos 4 e 5 e compõe-se de quatro fases, a saber:

1) Fase inicial de projeto.

2) Fase de análise.

3) Fase de melhora do desempenho.

4) Fase de crítica.

Para as fases de análise e de melhora do desempenho a arquitetura encadeada deve ser válida (ver Seção 3.1) e representada por um grafo de fluxo e tabela de tempos de execução.

1) Fase inicial de projeto

Nesta fase, a partir da frequência de chegada dos elementos da sequência de entrada ao PDTR, do algoritmo a ser implementado e dos componentes eletrônicos disponíveis para a construção do processador agrupam-se os passos do algoritmo a ser implementado em tarefas e, obedecendo a precedência dessas tarefas, aloca-se a execução delas por vários segmentos de circuito. Com isso, obtém-se uma *arquitetura encadeada proposta* para solucionar o problema. Esta arquitetura encadeada proposta deve estar sujeita, necessariamente, aos seguintes *requisitos de viabilidade:*

a) A unidade de tempo T, definida na Seção 3.1, além de ser suficientemente pequena para que o tempo requerido por cada segmento na realização da sua tarefa possa ser expresso como múltiplo inteiro de T, deve ser compatível com o funcionamento da parte de controle, já que a cada unidade de tempo T um novo comando deve estar disponível no registro de comando da parte de controle (veja-se a Seção 6.1).

b) O fator real de utilização (Seção 3.6) de qualquer segmento da arquitetura encadeada obtida nessa fase inicial não pode ser maior do que Tc, onde Tc é o inverso da frequência de chegada ao PDTR dos elementos da sequência de entrada; já que, se o limitante inferior real LIR* para esta arquitetura encadeada proposta for maior do que Tc, então será impossível obter um ciclo (inclusive o ótimo) de iniciações do algoritmo implementado com latência média menor do que Tc, satisfatória para solucionar o problema (veja-se a Seção 5.2).

2) Fase de análise

Nesta fase procede-se a uma análise da arquitetura encadeada proposta. Essa análise compõe-se de duas partes. Na primeira delas (Figura 6.4a) obtem-se o limitante inferior para a latência média do ciclo ótimo: LMmin*. Se o LMmin* encontrado não for satisfatório, ou

seja, maior do que Tc, onde Tc é o inverso da frequência fc de chegada dos elementos da sequência de entrada ao PDTR, então parte-se direto para a fase de melhora do desempenho da arquitetura encadeada proposta. Isto porque nenhum ciclo do grafo de latências permitidas dessa arquitetura encadeada irá apresentar latência média menor do que Tc.

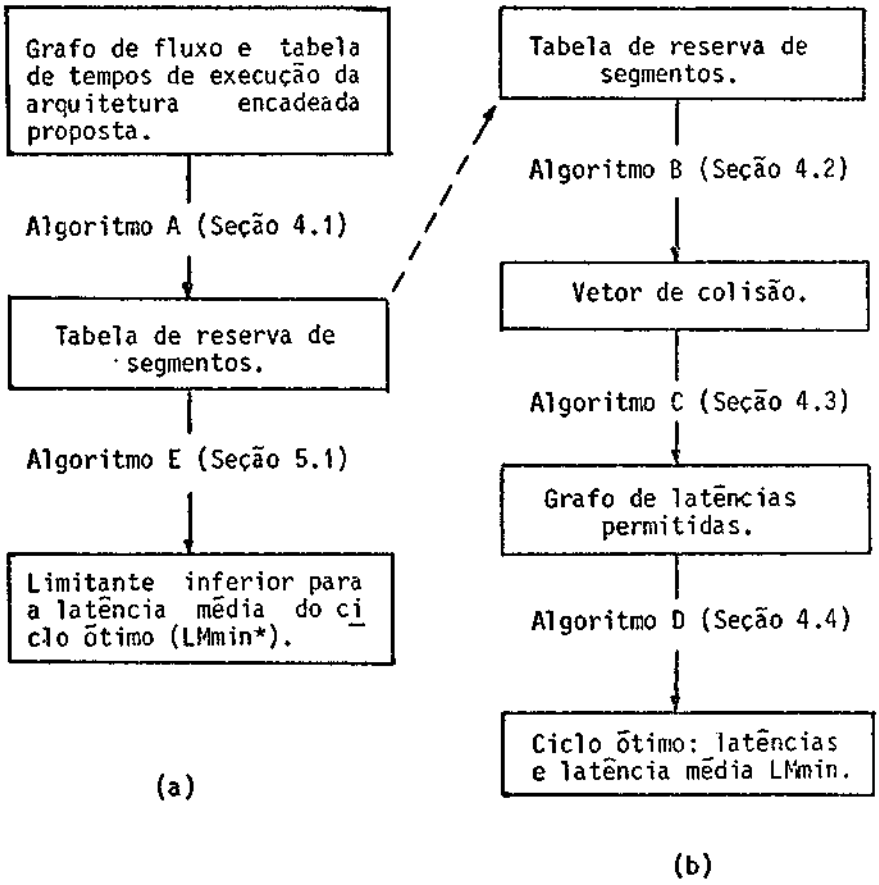


Fig. 6.4 - Procedimentos usados na fase de análise de uma arquitetura encadeada: a) primeira parte, b) segunda parte.

Caso contrário, existe a possibilidade de um ciclo do grafo de latências permitidas da arquitetura encadeada porposta ter latência média menor do que Tc. Na segunda parte desta fase (Figura

6.4b) pesquisa-se o grafo de latências permitidas da arquitetura encadeada em busca das latências de um ciclo ótimo de iniciações do algoritmo implementado e a respectiva latência média (que é mínima) LMmin.

Se essa latência média LMmin encontrada for inferior a Tc, a arquitetura encadeada proposta é uma solução satisfatória e restaria apenas verificar se o ciclo ótimo encontrado é um ciclo com latência constante. Se não for, verificar qual a influência disso na necessidade de utilizar "buffers" na entrada e saída do PDTR para compatibilizar uma possível frequência irregular de iniciações do algoritmo com a frequência de chegada e saída de dados no PDTR (veja-se a Seção 3.5 e Capítulo 5) e ir direto para a fase de crítica.

Entretanto, o que geralmente ocorre nessa fase é a latência média LMmin encontrada não ser satisfatória (é maior do que Tc). Neste caso, deve-se submeter a arquitetura encadeada proposta à fase de melhora do desempenho.

3) Fase de melhora do desempenho

A fase de melhora de desempenho é ativada apenas se a arquitetura encadeada proposta não apresenta um ciclo de iniciações do algoritmo implementado compatível com o fluxo de dados que o PDTR deve receber, ou, se esse ciclo, mesmo existindo, por não apresentar latência constante (o que torna necessário o uso dos "buffers" mencionados na fase anterior) traga problemas para a construção do PDTR.

Nela, usando os procedimentos desenvolvidos no Capítulo 5 (veja-se a Figura 6.5), a arquitetura encadeada proposta é modificada (inserção de segmentos atrasadores) de forma tal que a arquitetura encadeada resultante apresente um ciclo ótimo com latência constante igual ao limitante inferior real LIR*. Esta arquitetura certamente é solução para o problema em questão, já que a arquitetura encadeada proposta possui LIR* $\leq$ Tc, que é um dos requisitos de viabilidade da fase inicial.

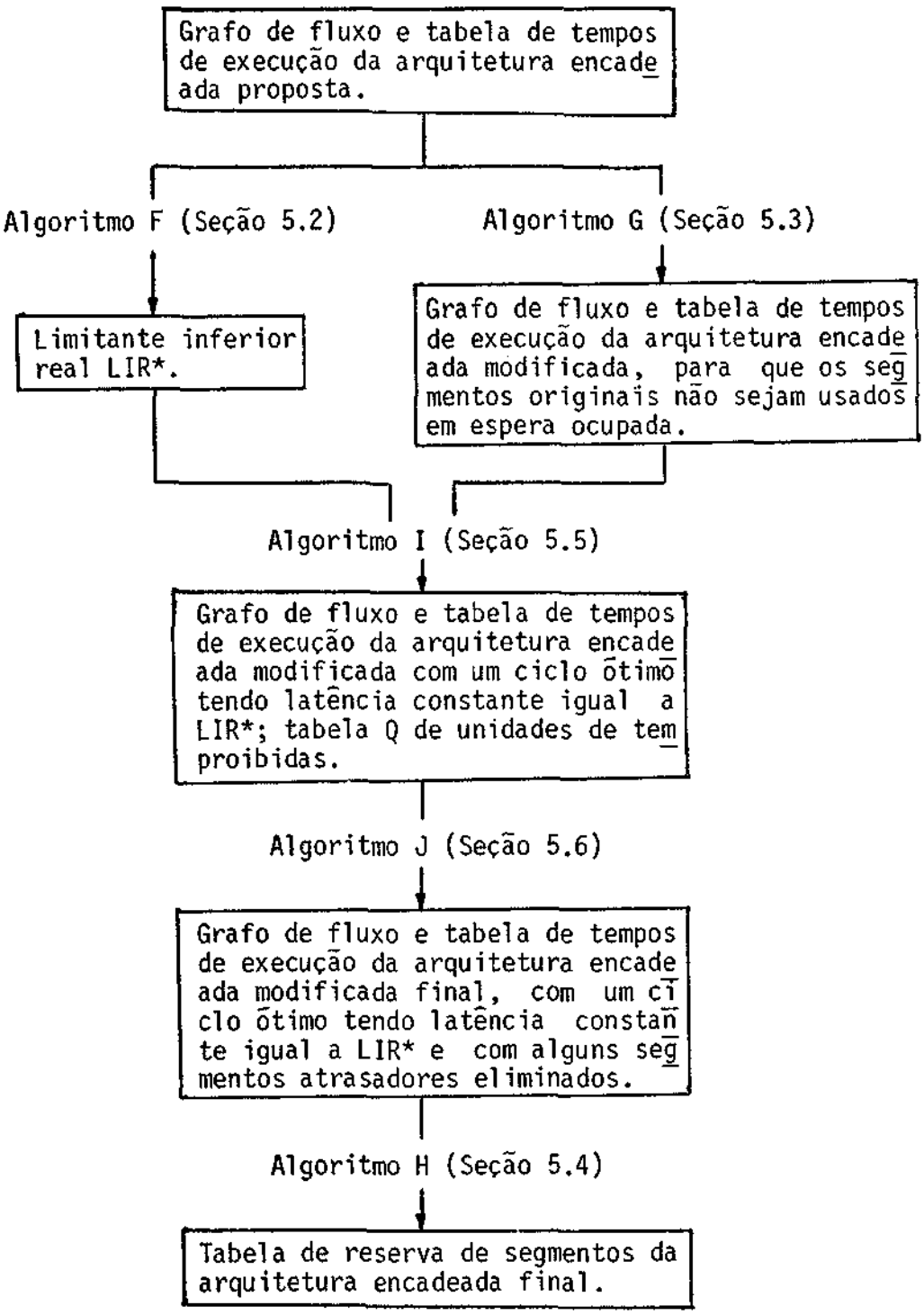


Fig. 6.5 - Procedimentos para a melhora do desempenho de uma arquitetura encadeada.

Além da arquitetura encadeada modificada final, esta fase também gera a tabela de reserva de segmentos correspondente, que deverá ser usada em uma segunda ativação da fase de análise apenas a título de verificação (veja-se a Figura 6.4b).

4) Fase de crítica

Uma vez que se tenha atingido o objetivo primeiro, que é o da obtenção de um processador com arquitetura encadeada que apresente um ciclo de iniciações do algoritmo implementado compatível com o fluxo de chegada de dados do problema a ser resolvido, convém então que se proceda a uma fase de crítica. Baseado no fator de utilização dos segmentos verifica-se a possibilidade de um rearranjo dos segmentos da arquitetura encadeada final para se obter um circuito menor ou uma frequência ainda maior de iniciações do algoritmo implementado. Também nessa fase de crítica pode-se verificar se a piora no desempenho dos segmentos não poderia causar a eliminação de segmentos atrasadores da arquitetura encadeada modificada final obtida. É importante lembrar que qualquer alteração efetuada na arquitetura encadeada, nessa fase, não pode causar a distância entre dois pares de X da tabela de reserva de segmentos a ser múltiplo inteiro de LIR*, porque isso acarretaria o desaparecimento do ciclo ótimo com latência constante LIR*.

Um outro enfoque de implementação poderia ser o do uso iterativo da metodologia, composto dos seguintes passos:

a) Propõe-se uma arquitetura encadeada para solucionar o problema.

b) Submete-se a arquitetura às fases de análise e melhora de desempenho.

c) Analise-se a arquitetura encadeada modificada final assim obtida. Nesta análise pode-se alterar a arquitetura encadeada proposta e retornar para o passo b). Caso contrário, foi encontrada uma arquitetura encadeada satisfatória para solucionar o problema em questão.

6.3 - EXEMPLOS DE IMPLEMENTAÇÃO

Nesta seção, a título de exemplo, discute-se a implementação de dois PDTRs com arquiteturas encadeadas, usando a metodologia da seção anterior. Os problemas exemplos abordados encontram-se descritos no Apêndice B. O primeiro deles (equalizador radiométrico de imagens) exemplificará o uso de arquiteturas encadeadas para baratear o custo do processador. O outro exemplo (decodificador de Viterbi) mostrará como o uso de técnicas de encadeamento pode viabilizar a construção de um processador rápido com os componentes eletrônicos comuns, disponíveis para tal.

6.3.1 - EQUALIZADOR RADIOMÉTRICO DE IMAGENS

Este é o problema exemplo A descrito na Seção 3.2 do Apêndice B. Nele, se não houver paralelismo na equalização dos "pixels" da linha de uma imagem (o que é permitido pelo algoritmo em questão), o processador deve, em 1 μs, efetuar três acessos à memória para a obtenção dos dados de calibração, uma subtração, um produto e uma divisão. Entretanto, inserindo paralelismo no processamento dos "pixels", o processador passará a dispor de 1 μs para efetuar cada uma dessas operações, o que reduz significativamente o custo dos componentes necessários para construir o processador. Isto é atingido aplicando a metodologia da Seção 6.2 no projeto do PDTR, como será mostrado a seguir.

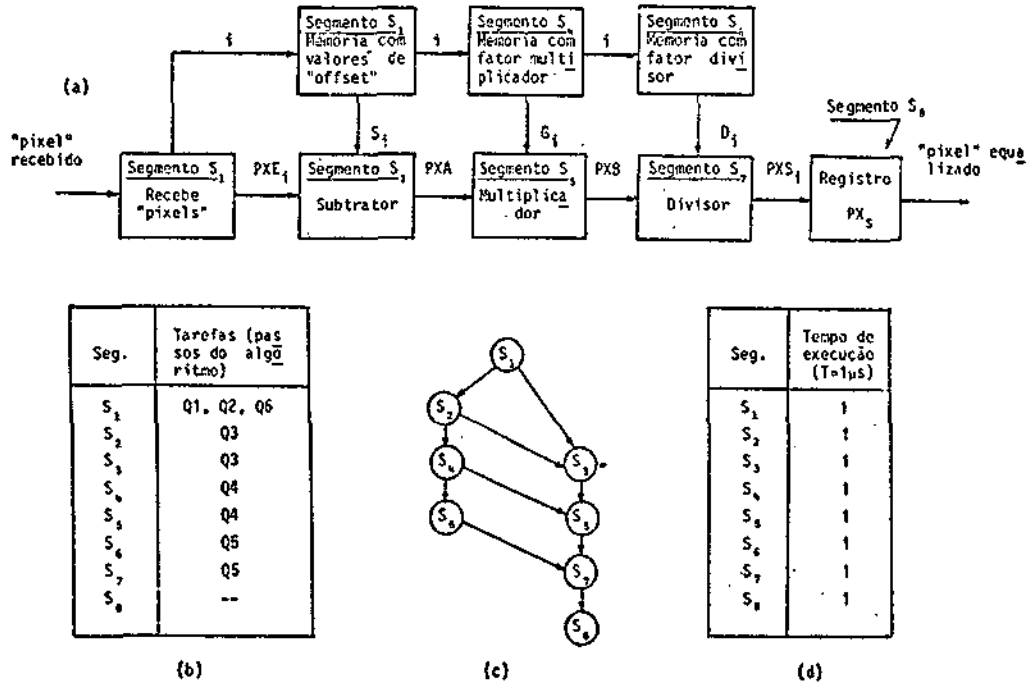
Na fase inicial do projeto, a partir dos seguintes dados iniciais:

a) frequência de chegada ao PDTR dos elementos da sequência de entrada: fc = 1 M "pixels"/s (→ Tc = 1 μs);

b) algoritmo a ser implementado: algoritmo Q descrito na Seção B.2 do Apêndice B.2

c) componentes disponíveis para a construção do processador: circuitos integrados TTL ("standard, schottky e low-power schottky") e memórias de acesso aleatório MOS;

obtem-se a arquitetura encadeada proposta da Figura 6.6a, que resultou da alocação de tarefas da Figura 6.6b. O fluxo de operandos pela arquitetura encadeada leva ao grafo de fluxo da Figura 6.6c, enquanto o tempo de execução das tarefas, por simplicidade, é o mesmo para todos os segmentos e igual a 1 μs (veja-se a Figura 6.6d).



Seg.	Tarefas (passos do algoritmo)
S_1	Q1, Q2, Q6
S_2	Q3
S_3	Q3
S_4	Q4
S_5	Q4
S_6	Q5
S_7	Q5
S_8	--

(b)

(c)

Seg.	Tempo de execução (T=1μs)
S_1	1
S_2	1
S_3	1
S_4	1
S_5	1
S_6	1
S_7	1
S_8	1

(d)

Fig. 6.6 - Equalizador radiométrico de imagens (proposto): a) arquitetura encadeada; b) alocação de tarefas; c) grafo de fluxo; d) tabela de tempos de execução.

Os requisitos de viabilidade estão atendidos por esta arquitetura encadeada proposta, pois 1 μs (unidade de tempo) é suficiente para a construção da parte de controle com os componentes eletrônicos disponíveis, e cada segmento é utilizado apenas uma vez na execução do algoritmo implementado, o que faz o fator real de utilização de todos os segmentos ser igual a uma unidade de tempo e, portanto, não maior do que Tc.

A tabela de reserva de segmentos obtida na fase de análise (Tabela 6.1) fornece um limitante inferior LMmin* para a latência média do ciclo de iniciações do algoritmo implementado igual a duas unidades de tempo, ou seja, 2 μs, que é maior do que Tc, sendo, portanto, insatisfatória. Assim, torna-se imperativa uma melhora do desempenho da arquitetura encadeada proposta.

TABELA 6.1

EQUALIZADOR RADIOMÉTRICO DE IMAGENS: TABELA DE RESERVA DE SEGMENTOS DA ARQUITETURA ENCADEADA PROPOSTA; LMmin* = 2

Seg.	T_1	T_2	T_3	T_4	T_5	T_6
S_1	X					
S_2		X				
S_3		X	X			
S_4			X			
S_5				X		
S_6				X		
S_7					X	
S_8						X

A fase de melhora do desempenho, realizada de acordo com o exposto na Figura 6.5, leva à inserção de um segmento atrasador (S_{11}) na passagem do operando PXE_i do segmento S_1 para o segmento S_3, ge

rando a arquitetura encadeada modificada da Figura 6.7, que, por sua vez, apresenta vetor de colisão vazio e um ciclo de iniciações do algoritmo implementado com latência constante igual a 1 μs, satisfatório para resolver o problema.

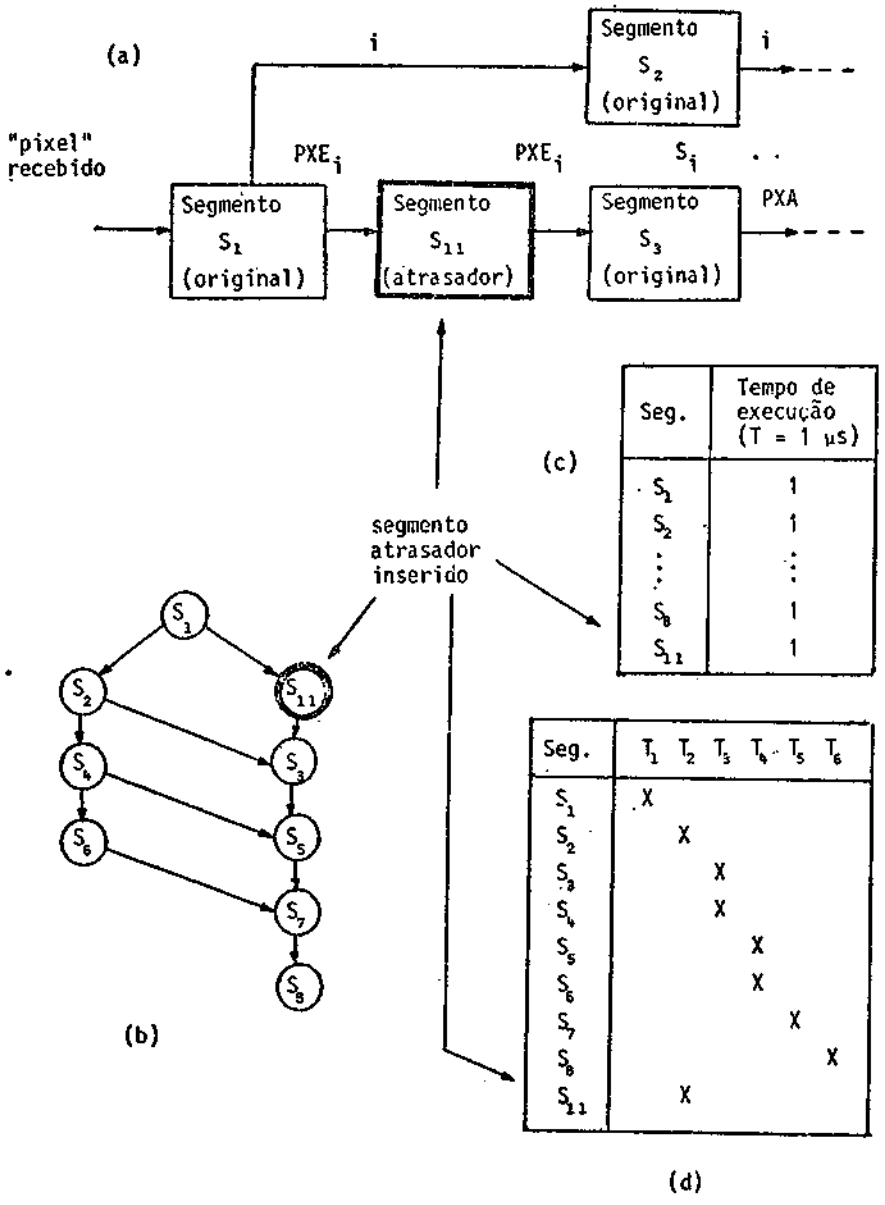


Fig. 6.7 - Equalizador radiométrico de imagens (modificado): a) arquitetura encadeada, b) grafo de fluxo, c) tabela de tempos de execução, d) tabela de reserva de segmentos.

6.3.2 - DECODIFICADOR DE VITERBI

No decodificador de Viterbi (problema - exemplo B descrito na Seção B.3 do Apêndice B), cada elemento da sequência de entrada compõe-se dos valores assumidos pelo par de bits quantizados y_1 e y_2. Como o processamento de cada par de bits depende dos valores determinados para as métricas de estado (S_i) e trajetória (P_{ij}), no processamento do par de bits anterior, então não pode haver paralelismo a nível de processamento dos elementos da sequência de entrada do PDTR. Isto faz com que o algoritmo do decodificador de Viterbi, no regime, deva ser executado do passo R5 a R21 (onde se inclui a execução dos passos R10 a R19 por 64 vezes) em 62,5 μs no máximo, o que daria menos de 1 μ para os 6 acessos à memória, 4 somas, 2 deslocamentos de 24 bits, 4 comparações e demais transferências de dados, envolvidos em uma execução dos passos R10 a R19 deste algoritmo.

Entretanto, o algoritmo em questão permite que cada uma das 64 execuções dos passos R10 a R19 possa se dar em paralelo com as outras execuções desses passos, o que abre uma possibilidade de dilatar o tempo de 1 μs, mencionado no parágrafo anterior, e caracteriza um paralelismo a nível de execução de tarefas referentes ao processamento de um único elemento da sequência de entrada do PDTR (reporte-se à Seção 2.2). Ressalte-se que dilatar o tempo para a execução de um processamento implica o fato de poder construir o processador com componentes mais lentos.

Para a inserção deste nível de paralelismo com o uso de técnicas de encadeamento (metodologia da Seção 6.2) os dados iniciais do problema são:

a) Supondo que 10% do tempo disponível para o processamento de um par de bits y_1 e y_2 sejam gastos na execução dos passos R5 a R9, R20 e R21, restaria 56 μs (0,9 x 62,5 μs) para as 64 execu

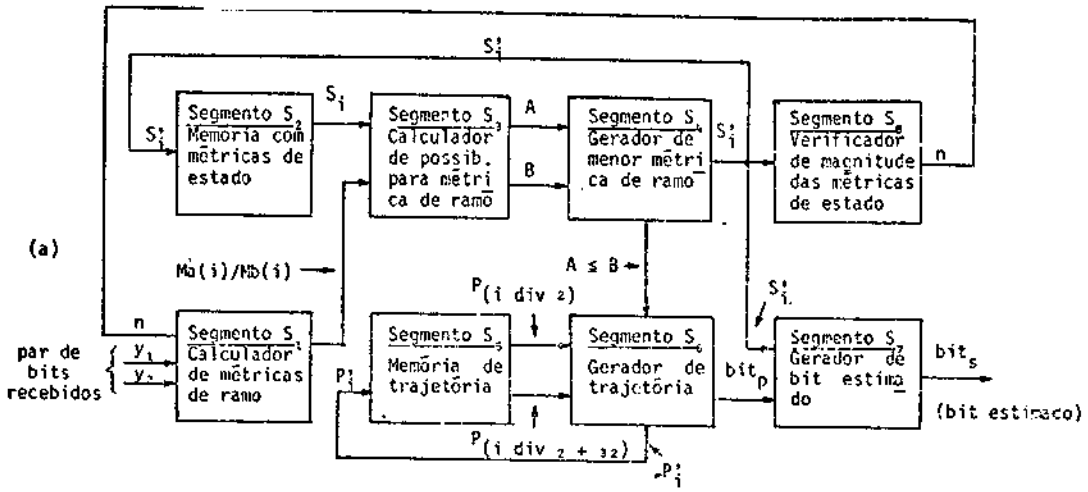
ções dos passos R10 a R19, o que daria uma "frequência de chegada" (inícios de execução dos passos R10 a R19) da ordem de $1{,}2 \times 10^6\ s^{-1}$ (= 1/(56 μs/64)) e Tc = 830 ns.

b) O algoritmo a ser implementado (no nível de paralelismo considerado) corresponde aos passos R10 a R19 do Algoritmo R descrito na Seção 3 do Apêndice B.

c) Os componentes disponíveis para a construção do processador são circuitos integrados TTL e memórias de acesso aleatório MOS.

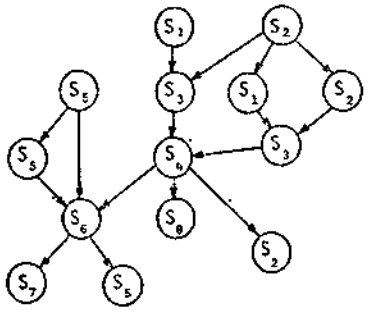
Desses dados, chega-se à arquitetura encadeada proposta da Figura 6.8, que atende aos requisitos de viabilidade: a unidade de tempo T de 120 ns não impossibilita a construção da unidade de controle com os componentes eletrônicos disponíveis e o maior fator de utilização real (segmentos S_2 e S_5) é 6T = 720 ns < Tc = 830 ns.

A tabela de reserva de segmentos da arquitetura encadeada proposta (Figura 6.9), obtida na fase de análise, fornece um limitante inferior LMmim*, para a latência média do ciclo ótimo de iniciações do algoritmo implementado, satisfatório e igual a 6 T = 720 ns. Mas, a segunda parte da fase de análise indica um único ciclo no grafo de latências permitidas, que tem latência constante insatisfatória de 9T = 1080 ns > Tc = 830 ns, sendo necessário submeter a arquitetura encadeada proposta a uma fase de melhora de desempenho.



Seg.	Tarefas (passos do algoritmo)
S_1	R10, R11
S_2	R12, R13, R16
S_3	R12, R13
S_4	R16
S_5	R14, R15, R16
S_6	R14, R15, R16
S_7	R18
S_8	R17

(b)



(c)

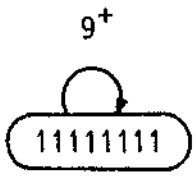
Seg.	Tempo de execução (T=120 ns)
S_1	1
S_2	2
S_3	1
S_4	1
S_5	2
S_6	1
S_7	1
S_8	1

(d)

Fig. 6.8 - Decodificador de Viterbi (proposto): a) arquitetura encadeada, b) alocação de tarefas, c) grafo de fluxo, d) tabela de tempos de execução.

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
S_1	X		X						
S_2	X	X	X	X			X	X	
S_3		X	X	X	X				
S_4				X	X	X			
S_5	X	X	X	X				X	X
S_6			X	X	X	X	X		
S_7								X	
S_8							X		

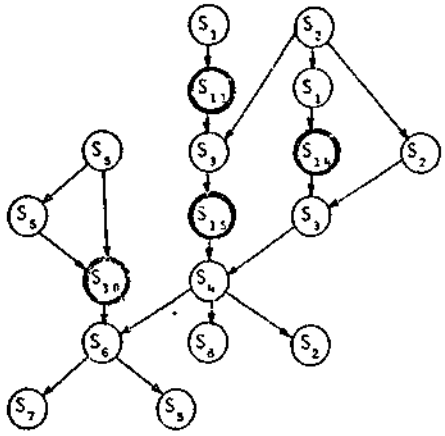
(a)



(b)

Fig. 6.9 - Decodificador de Viterbi: a) tabela de reserva de segmentos da arquitetura encadeada proposta, b) grafo de latências permitidas.

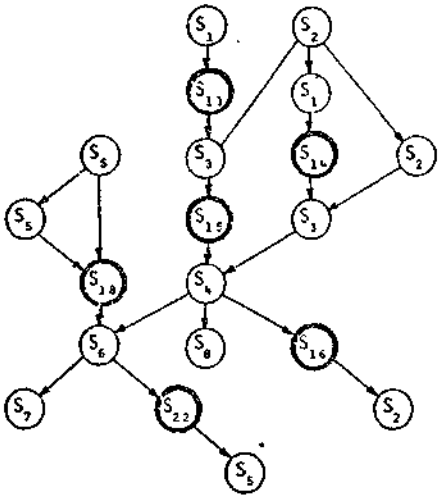
A fase de melhora do desempenho fornece os resultados intermediários da Figura 6.10, e a arquitetura encadeada modificada final da Figura 6.11, onde, através da inserção de segmentos atrasadores na passagem dos operandos do segmento S_4 para o segmento S_2 (S_i') e do segmento S_6 para o segmento S_5 (P_{ij}'), forçou-se o aparecimento de um ciclo ótimo com latência constante igual a $6T = 720$ ns. Utilizando a Equação 5.6, o tempo total de processamento dos passos R10 a R19, por 64 vezes, na arquitetura encadeada modificada final será de 63 x (6 x 120 ns) + (12 x 120 ns) = 46,8 μs que é de 75% do tempo disponível (62,5 μs) para o processamento de um par de bits y_1 e y_2. Enquanto, para a arquitetura encadeada proposta esse tempo é de 63 x (9 x 120 ns) + (9 x 120 ns) = 69 μs, superior a 62,5 μs.



Seg.	nT
S_1	1
S_2	2
S_3	1
S_4	1
S_5	2
S_6	1
S_7	1
S_8	1
S_{11}	1
S_{14}	1
S_{15}	2
S_{16}	4

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9
S_1	X		X						
S_2	X	X	X	X			X	X	
S_3			X		X				
S_4						X			
S_5	X	X	X	X				X	X
S_6							X		
S_7								X	
S_8							X		
S_{11}		X							
S_{14}				X					
S_{15}				X	X				
S_{16}			X	X	X	X			

(a)

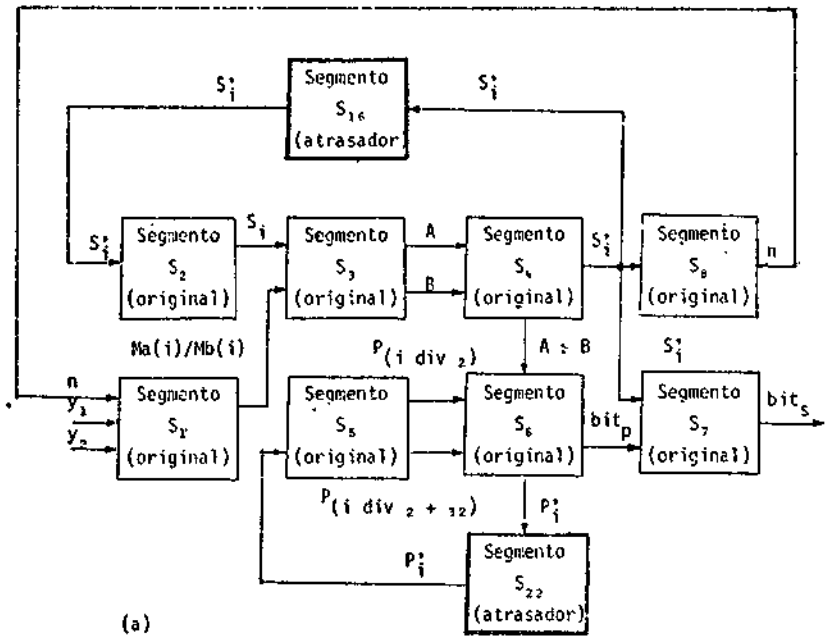


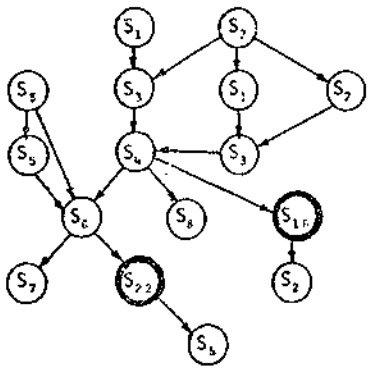
Seg.	nT
S_1	1
S_2	2
S_3	1
S_4	1
S_5	2
S_6	1
S_7	1
S_8	1
S_{11}	1
S_{14}	1
S_{15}	2
S_{16}	4
S_{18}	4
S_{22}	3

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}
S_1	X		X									
S_2	X	X	X	X							X	X
S_3			X		X							
S_4						X						
S_5	X	X	X	X							X	X
S_6							X					
S_7								X				
S_8							X					
S_{11}		X										
S_{14}				X								
S_{15}				X	X							
S_{16}							X	X	X	X		
S_{18}			X	X	X	X						
S_{22}								X	X	X		

(b)

Fig. 6.10 - Decodificador de Viterbi (modificado) - grafo de fluxo, tabela de tempos de execução e tabela de reserva de segmentos: a) arquitetura encadeada modificada, sem espera ocupada no uso dos segmentos originais; b) arquitetura encadeada modificada, sem espera ocupada e com ciclo ótimo = LIR*.





Seg.	nT
S_1	1
S_2	2
S_3	1
S_4	1
S_5	2
S_6	1
S_7	1
S_8	1
S_{16}	4
S_{22}	3

(c)

Seg.	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{32}
S_1	X		X									
S_2	X	X	X	X							X	X
S_3		X	X	X	X							
S_4				X	X	X						
S_5	X	X	X	X							X	X
S_6			X	X	X	X	X					
S_7								X				
S_8							X					
S_{16}							X	X	X	X		
S_{22}								X	X	X		

(d)

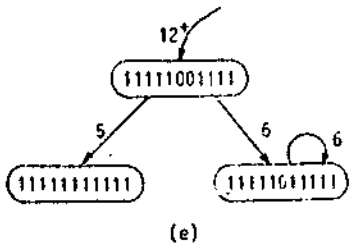


Fig. 6.11 - Decodificador de Viterbi (modificado final): a) arquitetura encadeada, b) grafo de fluxo, c) tabela de tempos de execução, d) tabela de reserva de segmentos , e) grafo de latências permitidas.

CAPÍTULO 7

CONCLUSÕES

O funcionamento de arquiteturas encadeadas, estaticamente configuradas, foi estudado quando várias execuções do algoritmo implementado se dão ao mesmo tempo. Deste estudo resultaram dois limitantes para a latência média de um ciclo ótimo de iniciações do algoritmo implementado. Um deles, limitante inferior LMmin*, obtido da tabela de reserva de segmentos, é importante porque ele é capaz de informar, antes da construção do grafo de latências permitidas e da pesquisa neste grafo do ciclo (ótimo) com menor latência média (LMmin), se existe a possibilidade de LMmin ser menor ou igual a um valor desejado. O outro limitante, limitante inferior real LIR*, obtido diretamente do grafo de fluxo e da tabela de tempos de execução da arquitetura encadeada, indica o menor valor possível para LMmin; isto acontece quando os segmentos da arquitetura encadeada estão sendo utilizados efetivamente (sem espera ocupada) o máximo possível, e pelo menos um deles 100% no tempo. Deste estudo também resultaram procedimentos que permitem a análise e melhora do desempenho de arquiteturas encadeadas com um aumento do fator de utilização dos segmentos até o seu limite máximo.

Uma metodologia de implementação de processadores com arquiteturas encadeadas, que se utiliza dos procedimentos desenvolvidos, foi abordada. Com ela, atingiu-se o objetivo primeiro do trabalho que foi desenvolver meios de análise e uso eficiente de arquiteturas encadeadas no projeto de processadores digitais dedicados a aplicações em tempo real, que lidem com altas taxas de dados. Essa metodologia fundamenta-se em duas características importantes:

1) Dado o problema a ser resolvido com um PDTR, ela fornece os *requisitos de viabilidade* que qualquer arquitetura encadeada deve satisfazer para resolver esse problema, ou seja, fornece parâmetros que influenciam no projeto inicial dos segmentos que comporão a arquitetura encadeada.

2) Se uma arquitetura encadeada proposta não é capaz de processar o fluxo contínuo de dados que chega ao processador, mas satisfaz os requisitos de viabilidade mencionados no item 1, então a metodologia altera essa arquitetura encadeada proposta para torná-la satisfatória na solução do problema.

Exemplos de emprego de arquiteturas encadeadas em três níveis de paralelismo foram apresentados: no processamento ao mesmo tempo de vários elementos da sequência de entrada que chega ao PDTR (equalizador radiométrico de imagens); na execução de tarefas em paralelo referentes ao processamento de um único elemento da sequência de entrada do PDTR (decodificador de Viterbi); e no mecanismo de busca e execução dos comandos na parte de controle de um PDTR. Esses exemplos também evidenciaram duas situações onde o uso de técnicas de encadeamento são úteis: na diminuição do custo do circuito de um PDTR (equalizador radiométrico de imagens) e em como atingir uma alta velocidade de execução de um algoritmo com o uso de componentes eletrônicos TTL e memórias MOS (decodificador de Viterbi). Apesar de a metodologia desenvolvida ser aplicável a qualquer sistema que envolva a execução em paralelo de um algoritmo repetitivamente no tempo, os exemplos empregados também mostram que, além de gerar processadores mais baratos e rápidos, a metodologia pode ser vista como uma maneira de projetar circuitos organizados, com blocos funcionais bem definidos e, portanto, mais fáceis de ser construídos e testados.

Os algoritmos apresentados nos Capítulos 4 e 5 foram codificados em linguagem ALGOL e os programas decorrentes foram executados no computador B6800 do INPE de São José dos Campos. O objetivo dessa implementação foi unicamente testar o funcionamento correto dos procedimentos (reduzidos a esses algoritmos) e não envolveu maiores cuidados quanto a uma redução das estruturas de dados manipulados, bem como do tempo de execução deles. Fica, portanto, como sugestão para trabalhos futuros um estudo mais apurado para a geração de versões otimizadas desses algoritmos.

No Capítulo 5 foram desenvolvidos procedimentos que, apenas com a inserção de segmentos atrasadores, alteram uma arquitetura encadeada de forma a fazer com que apareça um ciclo ótimo de iniciações do algoritmo implementado com latência constante igual ao limitante inferior real LIR*. Sob esse aspecto, um objetivo do trabalho foi atingido. Entretanto, apesar de alguns segmentos atrasadores serem eliminados da arquitetura encadeada modificada final (veja-se a Seção 5.6), não se considerou a eliminação de um número máximo (que equivale à inserção do número mínimo) de segmentos atrasadores. Isso também fica como sugestão para trabalhos futuros que poderão considerar a alteração mínima permitida, mencionada, mas não utilizada de "piorar" o desempenho de um segmento da arquitetura encadeada original.

REFERÊNCIAS BIBLIOGRÁFICAS

DAVIDSON, E.S. The design and control of pipelined function generators, In: International IEEE Conference of Systems, Networks and Computers, Oaxtepec, Mexico, 1971. *Proceedings*. p. 19-21.

GLUSKHOV, V.M.; LETCHEVSKII, A.A. Theory of algorithms and discrete processors. In: Tou, J.T. *Advances for information systems science*, New York, Plenum, 1969. v.1 cap. 1, p. 1-58.

HAYES, J.P. Parallel processing. In: —— *Computer architecture and organization*. Tokyo, Macgrow-Hill, c1978. Seção 3.4, p. 209-236.

KONO, J. *Decodificador de máxima verossimilhança para código convolucional binário*. São José dos Campos, INPE, 1977, 53p. (INPE-1051-NTE/090).

LEMOS FILHO, A.C. *Idéias básicas para a implementação de controles microprogramados*. São José dos Campos, INPE, jul 1982. 36p. (INPE-2479-NTE/189).

RAMAMOORTHY, C.V.; LI, H.F. Pipeline architecture. ACM Computing Surverys, 9 (1): 61-102, Mar. 1977.

SHAR, L.E. *Design and scheduling of statically configured pipelines*. Stanford, Stanford University, 1972.

SPILKER, J.J. Viterbi decoding of convolutional codes. In: —— *Digital communications by satellite*. New Jersey, Prentice-Hall, c1977. cap. 14, p. 455-472.

SPOT TECHNOLOGY, *Spot Newsletter*, n. 2, Aug. 1982.

THOMPSON, L.L.; TRACY, R.A.; FRANKEL, D.G. On-board radiometric preprocessing for multispectral linear arrays (MLA). In SPIE, *Smart Sensors*, Washington, 1979. p. 17-18 (SPIE Proceedings v.178).

WHITE, D.E. *Bit-slice design: controllers and ALUS*. New York, Garland STPM Press, c1981.

BIBLIOGRAFIA COMPLEMENTAR

BAER, J.L. *Computer systems architecture.* Maryland, Computer Science Press, c1980.

CHEN, T.C. Overlap and pipeline processor. In: Stone, H.S. *Introduction to computer architecture.* Chicago, SRA, c1975. cap. 9, p. 375-429.

KELLER, R.M. Look-ahead processors. *ACM Computing Surveys*, 7 (4): 177-195, Dec. 1975.

KUCK, J.D.; LAWRIE, D.H.; SAMEH, A.H. *High speed computer and algorithm organization.* New York, Academic Press, c1977.

STONE, H.S. Parallel computers. In: ——— *Introduction to computer architecture.* Chicago, SRA, c1975. cap. 8, p. 318-374.

APÊNDICE A

CONSIDERAÇÕES SOBRE A EXECUÇÃO COMPLETA DOS ALGORITMOS

Neste apêndice verifica-se que os Algoritmos A e J dos Capítulos 4 a 5 terminam sempre após um número finito de passos executados.

A.1 - ALGORITMO A DA SEÇÃO 4.1

É facil de verificar que se o passo inicial A1 for executado, o passo A7 também o será. E o passo A24 será executado se o passo A7 o for.

No passo A24 é testado se ainda existe algum elemento da coluna f + 1 da matriz H maior do que zero. Se o teste for verdadeiro ocorre um retorno para o passo A7; caso contrário o Algoritmo A termina.

Ora, o grafo de fluxo de uma arquitetura encadeada válida não apresenta ciclos, o que faz com que sempre pelo menos um segmento da arquitetura encadeada esteja executando a sua tarefa, ou seja, para cada vez que o passo A7 for executado, pelo menos um elemento do vetor Y é feito igual a um no passo A10, o que causa pelo menos um elemento da coluna f + 1 da matriz H a ser decrementado de um no passo A21.

Portanto, após no máximo qmax (veja-se a Seção 4.1) execuções do ciclo A7-A24 (que é atingido após a execução do passo inicial A1), nenhum elemento da matriz H será maior do que zero, o teste do passo A24 será falso e o Algoritmo A terminará.

A.2 - ALGORITMO B DA SEÇÃO 4.2

Se o passo B5 for executado, os passos B6 e B7 também o serão, o que faz j ser incrementado de um e ocorrer um retorno para o passo B4. Se B3 for executado, B4 será executado inicialmente com j = 1. E, para cada vez que B4 é executado, j é incrementado de um (executando-se ou não o passo B5), o que faz com que B4 seja executado q vezes até j atingir o valor q e ocorrer uma execução do passo B8. Finalmente, se o passo inicial B1 for executado, B2 também o será, o que faz B3 ser executado inicialmente com i = 1. Mas, já foi mostrado que se B3 é executado, B8 também o é, o que faz o índice i ser incrementado cada vez que B3 é executado. Com isso, o índice i ultrapassará o valor s e o teste do passo B8 fará o Algoritmo B terminar após s execuções do ciclo B3-B8.

A.3 - ALGORITMO C DA SEÇÃO 4.3

Após a execução dos passos iniciais C1 e C2, os passos C3 a C7 armazenam, na matriz V, todos os v vetores candidatos a pertencer ao grafo de latências permitidas procurado. Iniciando com k = 1, para cada vetor candidato encontrado, k é incrementado de um até ultrapassar o valor v, quando então o passo C8 é executado.

Os passos C8 a C15, para cada um dos v vetores candidatos encontrados e para cada latência $\ell = 1, \ldots, n + \ell$ verificam se o vetor sucessor é também vetor candidato. Com isso, esta parte do algoritmo preenche a matriz G em um número finito de passos e vai para o passo C16.

Os passos C16 a C23 eliminam das matrizes V e G as linhas e colunas referentes a vetores candidatos que não pertencem ao grafo . Nesta parte, se algum vetor candidato é eliminado, g é decrementado de um e j é feito novamente igual a um no passo C17. Caso contrário, j é sucessivamente incrementado no passo C23 até atingir o valor g, quando então o Algoritmo C termina.

A.4 - ALGORITMO D DA SEÇÃO 4.4

Seja o custo de uma trajetória armazenada no vetor $P (= P_1, \ldots, P_p, \ldots, P_{n+1})$ definida como $\sum_{i=1}^{n+1} (g + 1)^{n+1-i} \times a_i$, onde: g é o número de nós do grafo sendo pesquisado; n é o número de bits do vetor de colisão que é o nó inicial do grafo; $a_i = P_i$, para $i = 1, \ldots, p$ e $a_i = 0$, para $i = p + 1, \ldots, n + 1$; e p é a última posição preenchida na pilha P.

Pode-se verificar que sempre que o índice de um nó é empilhado na pilha P, o custo da nova trajetória é sempre maior do que o das anteriores. Dado que $a_i \leq g_i$ para qualquer i e, portanto, que o custo possui um valor máximo $(g + 1)^{n+1} - 1$, então o Algoritmo D terminará após um número finito de passos executados.

A.5 - ALGORITMO E DA SEÇÃO 5.1

Após a execução dos passos iniciais E1 e E2, os passos E3 a E5 são execuatos s vezes, para $i = 1, \ldots, s$, quando então o Algoritmo E termina.

A.6 - ALGORITMO F DA SEÇÃO 5.2

Após a execução dos passos iniciais F1 e F2, os passos F3 a F5 são executados s vezes, para $j = 1, \ldots, s$, quando então o Algoritmo F termina.

A.7 - ALGORITMO G DA SEÇÃO 5.3

O Algoritmo G é análogo ao Algoritmo A. Portanto, de forma idêntica, pode-se mostrar que o Algoritmo G termina após a execução de um número finito de passos.

A.8 - ALGORITMO H DA SEÇÃO 5.4

O Algoritmo H também é análogo ao Algoritmo A. Portanto, de forma idêntica, pode-se mostrar que o Algoritmo H termina após executar um número finito de passos.

A.9 - ALGORITMO I DA SEÇÃO 5.5

É fácil verificar que se o passo inicial I1 for executado, o passo I6 também o será, e se o passo I6 for executado, o passo I21 também o será.

No passo I21 é testado se ainda existe algum elemento da coluna 2f + 1 da matriz H' maior do que zero. Se o teste for verdadeiro, ocorre um retorno para o passo I6. Caso contrário, o Algoritmo I termina.

O grafo de fluxo da arquitetura encadeada modificada não apresenta ciclos, o que faz com que pelo menos um segmento da arquitetura encadeada modificada (original ou atrasador) deve estar executando a sua tarefa. Assim, para cada vez que o passo I6 é executado, pelo menos um elemento do vetor Y é feito igual a um. Se este elemento corresponder a um segmento atrasador ou a um segmento original com utilização permitida, um elemento da coluna 2f + 1 da matriz H' é decrementado de um. Se o elemento corresponder a um segmento original com uso proibido pode ser que nenhum elemento da coluna 2f + 1 da matriz H' venha a ser decrementado. Entretanto, nesse caso, o ponteiro de tempo q é sempre incrementado no passo I21, o que fatalmente tornará o uso do segmento em questão permitido em uma execução futura do ciclo I6-I21, o que leva, após um número finito de execuções do ciclo I6-I21, nenhum elemento da coluna 2f + 1 da matriz H' a ser maior do que zero, causando o término da execução do Algoritmo I.

A.10 - ALGORITMO J DA SEÇÃO 5.6

Após a execução do passo J1 que faz i = 1, o ciclo J2-J9 é executado f vezes, para i = 1, ..., f, quando então o Algoritmo J termina. Vale notar que uma execução do ciclo J2-J9 pode incluir ou não a execução sequencial por uma vez dos passos J3 a J8.

APÊNDICE B

DESCRIÇÃO DOS PROBLEMAS EXEMPLOS

B.1 - INTRODUÇÃO

Neste apêndice estão descritos dois problemas que requerem um processamento digital, em tempo real, sobre um fluxo contínuo de dados: equalizador radiométrico de imagens (problema A) e decodificador de Viterbi (problema B). Eles são usados no Capítulo 6 como exemplos de implementação de PDTRs com arquiteturas encadeadas. Como o interesse do trabalho restringe-se apenas aos algoritmos envovildos e a taxa de chegada de dados ao processador, a abordagem dos problemas exemplos é feita de forma resumida. Maiores detalhes podem ser obtidos nas referências citadas.

B.2 - PROBLEMA A: EQUALIZADOR RADIOMÉTRICO DE IMAGENS

As redes lineares com fotodetetores de tecnologia CCD ("charge coupled device"), ou redes CCD, usadas nos satélites de observação da Terra, apresentam variações na resposta de um sensor para outro (Figura B.1). Estas variações acarretam distorções radiométricas que devem ser removidas da imagem obtida, com uma equalização da resposta dos fotodetetores da rede (ou redes) CCD utilizada.

Essa equalização envolve a correção do "offset" e a correção do ganho, para cada sensor separamente:

a) Correção de "offset": consiste em subtrair dos "pixels" de imagem gerados pelo sensor o valor correspondente ao escuro.

b) Correção de ganho: consiste em normalizar as variações de ganho existentes entre os vários sensores da rede CCD.

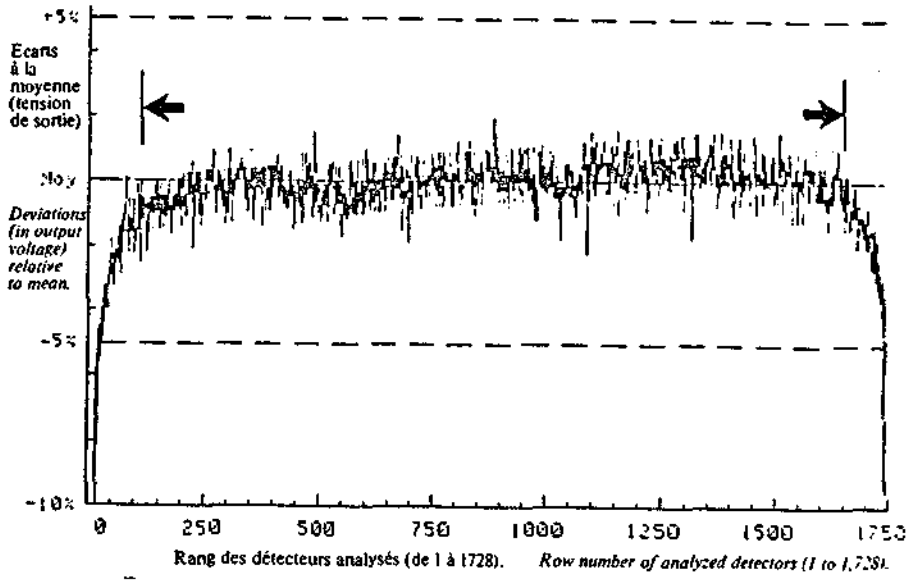


Fig. B. 1 - Variação na resposta dos 1728 sensores de uma rede CCD, exposta a uma iluminação espacial uniforme.

FONTE: Spot Technology, 1982.

Como o volume de dados gerados por cada passagem de um satélite de observação da Terra é alto (da ordem de centenas de M "pixel") e deve ser gravado em uma memória de massa, torna-se atrativo, durante o processo de recepção da imagem, armazenar os "pixels" já equalizados.

O equalizador (Figura B.2) recebe então os "pixels" da imagem junto com o pulso de sincronismo e, para realizar a sua função a contento, ele necessita ser calibrado com os valores referentes à correção do "offset" e ganho obtidos anteriormente. Devido à degradação da resposta dos sensores com o tempo, é conveniente que os dados para calibração sejam atualizados periodicamente.

Fig. B.2 - Equalizador radiométrico de imagens: entradas e saídas.

No problema considerado para exemplo neste trabalho, cada linha da imagem possui n "pixels"; a taxa de chegada de "pixels" é 1 M"pixel"/s e o algoritmo de equalização empregado é o sugerido por Thompson et alii (1979), que, além de manipular apenas números binários puros, apresenta uma divisão por um número que é sempre potência de 2. Esse algoritmo é descrito a seguir:

Algoritmo Q: equalização de imagem de satélites de observação da terra. Baseado nos valores de calibração de "offset" (S_i, $1 \leq i \leq n$) e ganho (fator multiplicador G_i, $1 \leq i \leq n$ e fator divisor D_i, $1 \leq i \leq n$) este algoritmo equaliza, sequencialmente os n "pixels" de uma linha de imagem gerada por um satélite de observação da Terra, obtida com redes CCDs.

Q1. Faça i = 1 (i será usado para determinar a ordem do "pixel" a ser equalizado e, portanto, quais dados de calibração devem ser usados).

Q2. Obtenha o "pixel" PXE_i.

Q3. Faça $PXA = PXE_i - S_i$.

Q4. Faça $PXB = PXA \times G_i$.

Q5. Faça PXS = PXB/D_i.

Q6. Se i < n, faça i = i + 1 e retorne para o passo Q2. Caso contrário, toda a linha já foi recebida e equalizada, o algoritmo termina.

B.3 - PROBLEMA B: DECODIFICADOR DE VITERBI

O texto a seguir baseia-se em Spilker (1977) e Kono(1977).

Nos canais de comunicação espacial, normalmente há largura de faixa suficiente para fazer uma expansão do canal, e o ruído pode ser considerado aditivo branco e gaussiano (AWGN) com boa aproximação. O aumento da eficiência do canal - geralmente medida pela razão Eb/No, onde Eb é a energia recebida por bit e No é a densidade espectral de potência unilateral do AWGN -, é obtido com menor custo através da codificação da informação do que com o aumento da relação sinal/ruído.

Como a comunicação por satélite envolve um tempo substancial de atraso (> 0,25 s) e, geralmente, altas taxas de dados, o uso de códigos convolucionais se torna atraente, já que eles realizam a correção a medida que os dados fluem pelo canal de comunicação sem necessidade de retransmissão.

Um codificador convolucional é um registro de deslocamento ("shift register") com k estágios e n geradores de funções algébricas lineares (um para cada saída), onde os bits de informação são deslocados de b em b bits. A Figura B.3 mostra um exemplo de codificador convolucional para k = 3, n = 2 e b = 1.

No caso de canais de comunicação de voz digitalizada por modulação delta e codificada com um código convolucional, o método de decodificação mais eficiente, entre os vários existentes, é o de máxima verossimilhança, ou de Viterbi.

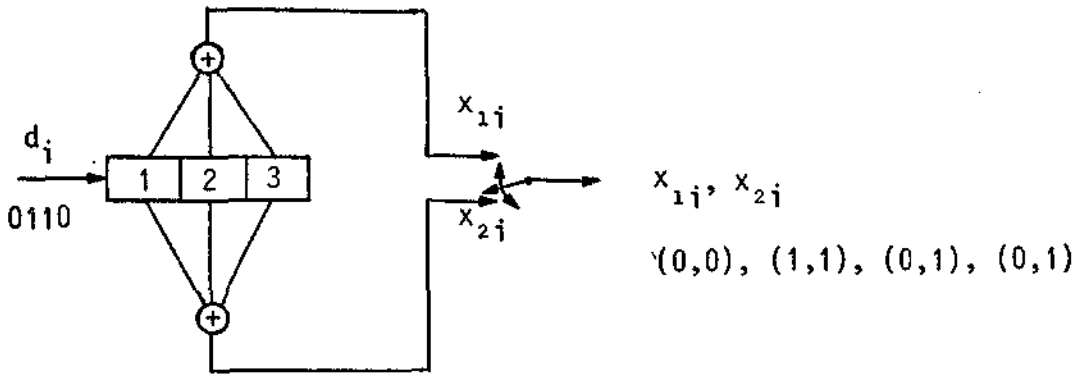


Fig. B.3 - Codificador convolucional para k = 3, n = 2 e b = 1.

FONTE: Spilker (1977), p. 456.

Na Figura B.4 são mostradas as entradas e saídas do decodificador convolucional, objeto de estudo neste trabalho.

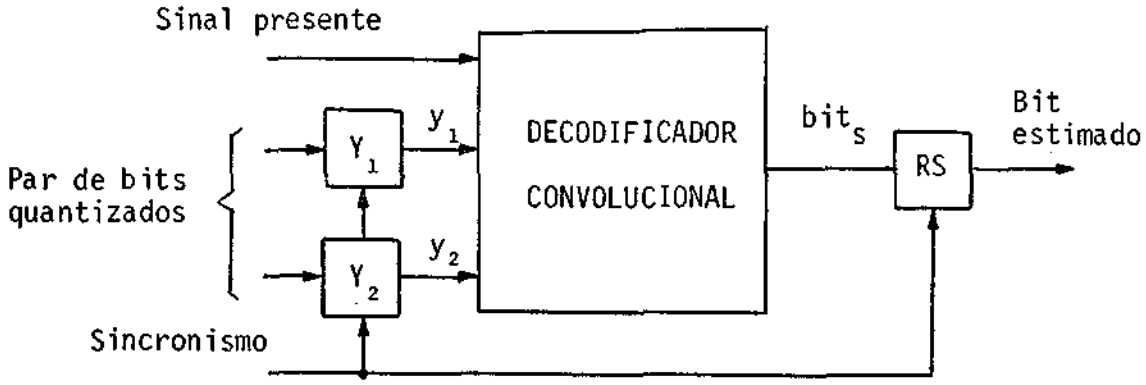


Fig. B.4 - Decodificador convolucional: entradas e saídas.

Uma vez que haja sinal presente no canal, o decodificador aguarda o armazenamento nos registros de entrada (Y1 e Y2) dos bits quantizados em 8 níveis y1 e y2 pelo pulso de sincronismo. Baseado nesses dois bits e no seu estado interno, o decodificador estima um bit (bit_s) que é colocada na entrada do registro de saída RS. No próximo pulso de sincronismo, o bit estimado é colocado na saída, enquanto dois novos bits y1 e y2 entram no decodificador. Ressalte-se que para cada par de bits que chega ao decodificador, é estimado apenas um bit

na sua saída. A taxa de transferência do canal é de 32 kbits/s e obviamente a decodificação deve ser realizada em tempo real. Tomando como referência Kono (1977), um decodificador de Viterbi com 64 estados requer o seguinte algoritmo de decodificação.

Algoritmo R: decodificador de Viterbi. Baseado no conteúdo das 64 métricas de estado (S_i, $0 \leq i \leq 63$), das 64 trajetórias de 24 bits (P_{ij}, $0 \leq i \leq 63$, $1 \leq j \leq 24$) e dos pares de bits quantizados (y1 e y2) este algoritmo estima os bits mais prováveis correspondentes a cada par de bits recebido.

R1. Se há sinal presente no canal, vá para o passo R2. Caso contrário, repita este passo.

R2. Zere as métricas de estado: faça $S_i = 0$, $0 \leq i \leq 63$.

R3. Zere todos os bits das trajetórias: faça $P_{ij} = 0$, $0 \leq i \leq 63$, $i \leq j \leq 24$.

R4. Faça n = 0 (n é usado para determinar se todas as métricas de estado calculadas são maiores do que 7).

R5. Se há sinal presente no canal, vá para o passo R6. Caso contrário, retorne para o passo R1.

R6. Se chegou o par de bits y1 e y2, vá para o passo R7. Caso contrário, retorne para o passo R5.

R7. Faça MIN = 127.

R8. Faça i = 0.

R9. Faça m = 1 (m será usado para verificar se todas as métricas de estado são maiores do que 7).

R10. Calcule a primeira possibilidade para a métrica de ramo Ma (i). Ma (i) é obtida com uma das seguintes quatro fórmulas, de acordo com o código convolucional empregado:

$$Ma\ (i) = \begin{cases} y1 + y1 - (n \times 7), \text{ ou} \\ 7 + y1 - y2 - (n \times 7), \text{ ou} \\ 7 + y2 - y1 - (n \times 7), \text{ ou} \\ 14 - y1 - y2 - (n \times 7). \end{cases}$$

R11. Calcule a segunda possibilidade para a métrica de ramo Mb (i). Mb (i) é também obtida com uma das quatro fórmulas do passo anterior e de acordo com o código convolucional empregado.

R12. Calcule a primeira possibilidade para a métrica de estado: faça $A = S_{(i \text{ div } 2)} + Ma\ (i)$.

R13. Calcule a segunda possibilidade para a métrica de estado: faça $B = S_{32+(i \text{ div } 2)} + Mb\ (i)$.

R14. Calcule a primeira possibilidade para a trajetória. Faça: $bit_a = P_{(i \text{ div } 2),\ 24}$; para $j = 24, j = 23, \ldots, j = 1$, $P_{aj} = P_{(i \text{ div } 2),\ (j-1)}$; $Pa_1 = 0$.

R15. Calcule a segunda possibilidade para a trajetória. Faça: $bit_b = P_{(32 + i \text{ div } 2),\ 24}$; para $j = 24, j = 23, \ldots, j = 1$, $Pb_j = P_{(32 + i \text{ div } 2),\ (j-1)}$; $Pb_1 = 1$.

R16. Se $A \leq B$, faça: $S_i' = A$; $P_{ij}' = Paj$, $0 \leq j \leq 63$; $bit_p = bit_a$. Caso contrário, faça $S_i' = B$; $P_{ij}' = Pb_j$, $0 \leq j \leq 63$; $bit_p = bit_b$.

R17. Se $S_i' < MIN$, faça $MIN = S_i'$ e $bit_s = bit_p$ (após as 64 interações MIN guardará a menor métrica de estado obtida e bit_s será o bit mais significativo da trajetória correspondente).

R18. Se $S'_i < 7$, faça m = 0 (após as 64 teraçōes, se todas as métricas de estado forem maiores do que 7, m permanecerá com o valor 1).

R19. Se i < 63, faça i = i + 1 e retorne para o passo R10. Caso contrário, em bit será o bit estimado, vá para o passo R20.

R20. Faça n = m e, para $0 \leq i \leq 63$, faça $S_i = S'_i$.

R21. Para $0 \leq i \leq 63$, $1 \leq j \leq 24$, faça $P_{ij} = P'_{ij}$, e retorne para o passo R5.