UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA ELÉTRICA
DEPTO DE TELEMÁTICA

Este Exemplar corresponde a redação final da tese defendida por Ildeberto de Genova Bugatti e aprovada pela comissão julgadora em 30/10/86.

Shusa Motoyama

COMPUTADORES A FLUXO DE DADOS:

APLICAÇÃO EM CENTRAL DE COMUTAÇÃO TELEFÔNICA

ILDEBERTO DE GENOVA BUGATTI
Orientador: Prof. Dr. SHUSABURO MOTOYAMA

054/86

Tese apresentada à Faculdade de Engenharia Elétrica, da Universidade Estadual de Campinas - UNICAMP - como parte dos requisitos exigidos para obtenção do título de MESTRE EM ENGENHARIA ELÉTRICA.

OUTUBRO 1986
UNICAMP

AGRADECIMENTOS

- Ao Prof. Dr.Shusaburo Motoyama, que dedicadamente, foi orientador e amigo.

- À Coordenação de Aperfeiçoamento de Pessoal de Nível Superior, ao Departamento de Computação e Estatística da UFSCar e ao setor Telemática do DEE-FEC-UNICAMP, pelo apoio logístico.

- À Ana Sígoli Fernandes Matheus, pelos desenhos.

- À Elaine Miguel, pela digitação.

- À Ivani Aparecida da Silva Bugatti, pelas referências bibliográficas.

- À todos os colegas que direta ou indiretamente colaboraram para a concretização deste trabalho.

RESUMO

Neste trabalho é proposta uma central de comutação controlada por um computador a fluxo de dados, com o objetivo de eliminar-se o problema de estrangulamento de software hoje existentes nos grandes sistemas de comutação eletrônica comerciais.

No fluxo de dados o mecanismo de execução de instruções é semelhante ao mecanismo de controle de uma central de comutação convencional. Isto permite realizar eficientemente o processamento de uma chamada telefônica.

Foi proposta uma arquitetura a fluxo de dados organizada em forma de anel. Esta organização permite flexibilidade para expansões, confiabilidade e grande eficiência na execução de instruções.

Avaliações preliminares mostram que esse tipo de arquitetura oferece boas perspectivas na obtenção de centrais telefonicas com alto desempenho.

INDICE

1 - INTRODUÇÃO 5

2 - FLUXO DE DADOS : CONCEITOS E APLICAÇÕES 10

2.1 - INTRODUÇÃO 11
2.2 - CONCEITOS DE FLUXO DE DADOS 11
2.2.1 - PACOTES DE DADOS 14
2.2.2 - OPERAÇÕES 14
2.2.3 - PROCEDIMENTOS PARA TRANSFORMAÇÃO DE UM ALGORITMO SEQUENCIAL PARA A NOTAÇÃO FLUXO DE DADOS 15
2.2.4 - OPERADORES DE CONTROLE 18
2.2.5 - REPRESENTAÇÃO DE UM PROGRAMA FLUXO DE DADOS NUMA LINGUAGEM DE MÁQUINA 23
2.3 - MECANISMO BÁSICO DE EXECUÇÃO DE INSTRUÇÃO NUMA MÁQUINA FLUXO DE DADOS 26
2.4 - ESTÁGIO ATUAL NO ASSUNTO 28
2.4.1 - COMPUTADOR FLUXO DE DADOS DO INSTITUTO DE TECNOLOGIA DE MASSACHUSETTS 29
2.4.2 - PROCESSADOR DE DADOS DISTRIBUIDOS DA TEXAS INSTRUMENTS 30
2.4.3 - MÁQUINA DIRIGIDA POR DADOS DE UTAH 32
2.4.4 - MÁQUINA FLUXO DE DADOS DE IRVINE 33
2.4.5 - MÁQUINA FLUXO DE DADOS DE MANCHESTER 34
2.4.6 - SISTEMA LAU DE TOULOUSE 35
2.4.7 - COMPUTADOR FLUXO DE DADOS DE NEWCASTLE 36
2.5 - UM EXEMPLO DE CENTRAL A FLUXO DE DADOS 37
2.6 - CONCLUSÃO 1 39

3 - ARQUITETURA PROPOSTA 41

3.1 - INTRODUÇÃO 42
3.2 - ORGANIZAÇÃO BÁSICA DA ESTRUTURA DE CONTROLE 42
3.3 - FORMATO DE INSTRUÇÃO E DADOS 45
3.4 - DESCRIÇÃO E IMPLEMENTAÇÃO DE BLOCOS BÁSICOS 49
3.4.1 - COMUTADOR DE ENTRADA E SAÍDA 50
3.4.2 - UNIDADE DE ATIVAÇÃO 52
3.4.2.1 - O MECANISMO DE DISPARO DE INSTRUÇÕES EXECUTÁVEIS 54
3.4.3 - UNIDADE DE INSTRUÇÕES EXECUTÁVEIS 56
3.4.4 - UNIDADE DE PROCESSAMENTO 60
3.4.5 - COMUNICAÇÃO ENTRE OS BLOCOS BÁSICOS DO SISTEMA EM ANEL 63
3.5 - ARQUITETURA EXPANDIDA 65
3.6 - TOLERÂNCIA A FALHAS 70
3.7 - CONCLUSÃO 2 71

4 - ANÁLISE DE DESEMPENHO 73

4.1 - INTRODUÇÃO 74
4.2 - DEFINIÇÕES 74
4.2.1 - COMPRIMENTO DE UM PROGRAMA 74
4.2.2 - SISTEMA DE EXECUÇÃO PERFEITA 75
4.2.3 - PARALELISMO MÉDIO DE UM PROGRAMA FLUXO DE DADOS 75
4.2.4 - EFICIÊNCIA DE UM SISTEMA COM "J" PROCESSADORES 76
4.3 - PARALELISMO NA ARQUITETURA EM ANEL 77
4.4 - TEMPO DE ATRASO NOS BLOCOS 83
4.4.1 - TEMPO DE CICLO DO COMUTADOR DE ENTRADA E SAIDA 83
4.4.2 - TEMPO DE CICLO DA UNIDADE DE PROCESSAMENTO. 84
4.4.3 - TEMPO DE CICLO DA UNIDADE DE INSTRUÇÕES EXECUTAVEIS 85
4.4.4 - TEMPO DE CICLO DA UNIDADE DE ATIVAÇÃO 85
4.5 - DIMENSIONAMENTO DA CENTRAL DE COMUTAÇÃO 91
4.6 - EXEMPLO DE DIMENSIONAMENTO 94
4.7 - CONCLUSÃO 3 96

5 - CONCLUSÃO GERAL 97

6 - REFERENCIAS 101

7 - APENDICES 104

7.1 - SÍNTESE DO CIRCUITO DE CONTROLE DO COMUTADOR DE ENTRADA E SAÍDA DA ARQUITEURA EXPANDIDA 105
7.2 - TEOREMA - "EFICIÊNCIA DA ARQUITETURA EM ANEL" 110
7.3 - EQUIVALENCIA ENTRE UM SISTEMA PERFEITO E O SISTEMA EM ANEL 112
7.4 - RELAÇÃO DE MNEMONICOS 114

CAPÍTULO 1

INTRODUÇÃO

Os princípios utilizados nos projetos de arquitetura de computadores permaneceram durante muito tempo inalterados, baseados na organização conceitual de Von Neumann (1945). Entretanto, as máquinas baseadas nos conceitos de Von Neumann possuem uma limitação intrínseca que é a busca e execução sequencial das instruções. Desse modo para aumentar a velocidade de processamento dessas máquinas é necessário aumentar a velocidade dos componentes eletrônicos que compõe o sistema. Porém, a velocidade desses componentes já está chegando a um patamar teoricamente possível de se alcançar. Mas, a exigência de computação mais rápida continua ainda crescendo, principalmente em aplicações como previsão de tempo, por exemplo.

Desse modo surgiu na década passada o conceito de processamento paralelo, em que as instruções seriam executadas na forma paralela, eliminando, assim o principal ponto de estrangulamento das máquinas tipo Von Neumann.

Atualmente todos os grandes computadores comerciais e experimentais reunem inovações de arquiterura dirigidas para aumento de desempenho. Essas inovações são sempre baseadas em alguma forma de obter a capacidade de executar um grande número de atividades em paralelo (concorrentemente).

Essas inovações na área de arquitetura de computadores tiveram consequências imediatas em atividades que fazem uso intensivo de computadores, controle de processos em tempo real por exemplo. E em particular houve um avanço quase que em paralelo no desenvolvimento de novas centrais de comutação telefônica eletrônicas controladas por computador (SPC).

O controle da maioria das centrais telefônicas eletrônicas

(SPC) atualmente comercializadas é centralizado, utilizando essencialmente um computador baseado no conceito de Von Neumann. Como este computador utiliza um processamento sequencial das instruções, existe uma limitação na velocidade com que estas instruções são executadas, conduzindo ao que é chamado estrangulamento (engarrafamento) de "software".

Para superar esta limitação, estão surgindo centrais telefônicas com controle distribuido e com processamento paralelo aproveitando, exatamente a evolução nesse campo. Nestas centrais, o controle é feito através de vários microprocessadores interligados por uma rede de comunicação, e cada um com funções específicas. Isto possibilita a construção de centrais com maior confiabilidade. A comutação por controle distribuido poderá operar mesmo que haja uma falha numa parte do sistema de controle.

Das principais arquiteturas que possibilitam o processamento paralelo que surgiram voltamos nosso interesse para a arquitetura a Fluxo de Dados, com o objetivo de obter-se uma central de comutação que tenha flexibilidade de expansão e alto desempenho.

Fluxo de Dados é um conceito utilizado em sistemas de computação que possuem operações concorrentes e linguagem de programação representada de uma forma paralela. A execução do fluxo de dados é conduzida por dados (data-driven), isto é, cada instrução é habilitada a execução, logo após o fornecimento do operando pela instrução predecessora. Isto possibilita um processamento altamente concorrente, divergindo da máquina tipo Von Neumann que possui um contador que sequencia a execução da instrução.

Como visto acima, uma máquina a fluxo de dados é baseada num sistema de multi-processadores com funções partilhadas; onde instruções são executadas de acordo com "regras conduzidas por dados", pelas quais, "cada instrução torna-se executável quando e somente quando todos os dados necessários estão disponíveis". Portanto o mecanismo de funcionamento de uma máquina a fluxo de dados é semelhante à seção de controle de um sistema de comutação convencional. Pois num sistema de comutação, as tarefas serão executadas (disparadas) somente quando estiverem disponíveis todos os recursos e dados necessários para realizá-la. Por exemplo, a procura de caminho na matriz de comutação só pode ser realizado se forem previamente fornecidos (obtidos) os números do assinante (usuário) chamador, assinante chamado e a ligação será efetuada somente se houver caminho disponível na matriz de comutação e o assinante chamado estiver no estado não ocupado.

Assim um sistema de comutação controlado por um computador a fluxo de dados que explore as semelhanças inerentes aos dois sistemas (controle da central e fluxo de dados) possui vantagens tais como:

Simplificação de software:- não é necessária nenhuma técnica sofisticada de multiprogramação. Possibilita-se a construção de programas [Ac82] (numa linguagewm de fluxo de dados) simplificados e legíveis, onde a sequência de execução de instruções estará bem próxima da sequência lógica dos sinais de controle necessários ao processamento de uma chamada telefônica.

Concorrencia na execução de instruções:- utilizando o paralelismo do sistema a fluxo de dados é possível reduzir o

tempo de processamento viabilizando a realização de um processamento de chamada eficiente.

Existem várias centrais com processamento paralelo propostas na literatura [CPqD85,In81,Sp77]. Existe porem uma única arquitetura de controle de central proposta na literatura,baseada no conceito de fluxo de dados [AYM84]. A arquitetura da central de comutação baseada em fluxo de dados proposta em [AYM84] não permite flexibilidade de expansão. Porque um aumento do número de processadores na organização em barramento pode acarretar atrasos consideráveis na comunicação entre eles.

O objetivo desse trabalho é propor uma arquitetura de uma central telefônica baseada no conceito de fluxo de dados, que tenha flexibilidade de expansão, além de alto grau de paralelismo.

No capítulo 2 são introduzidos os principais conceitos da teoria de fluxo de dados e o estágio atual da arte. No capítulo 3 é proposta uma arquitetura a fluxo de dados para realizar o controle de uma central de comutação. Alguns blocos desta arquitetura foram projetados a níveis de portas lógicas visando uma futura implementação. No capítulo 4 é feita uma análise de desempenho da arquitetura em anel proposta no capítulo 3 em termos de instruções por segundo e, também é realizado um exemplo de dimensionamento do número de processadores necessários à central para que esta atenda a um determinado tráfego telefônico medido em Erlangs. Finalmente, o capítulo 5 mostra as principais conclusões do trabalho.

CAPÍTULO 2

FLUXO DE DADOS : CONCEITOS E APLICAÇÕES

2.1 - Introdução

O termo **fluxo de dados** é utilizado em sistemas de computação que possuem operações concorrentes e a linguagem de programação é representada de uma forma paralela. A execução de instruções em computadores a fluxo de dados é conduzida por dados, isto é, cada instrução é habilitada à execução, logo após o fornecimento dos operandos pelas instruções predecessoras.

A programação das máquinas de Fluxo de Dados, para a execução das instruções de forma concorrente, é feita através das linguagens de Fluxo de Dados. As liguagens de Fluxo de Dados encontradas na bibliografia [De74,DM75,Ru77,GW80,Ac82,TKH82] são notações de programação nas quais a dependência de dados entre instruções sucessoras são expressas pela estrutura do programa.

Neste capítulo serão introduzidos alguns conceitos fluxo de dados necessários para o desenvolvimento do assunto.

2.2 - Conceitos de Fluxo de Dados

Um programa numa linguagem Fluxo de Dados elementar é um grafo dirigido no qual os nós são operadores conectados por enlaces através dos quais podem fluir valores. Um operador é habilitado quando valores estiverem presentes em todos os seus enlaces de entrada. O operador habilitado pode disparar (iniciar a execução) a qualquer instante, removendo os valores de seus enlaces de entrada, computando esses valores e enviando um resultado para o seu enlace de saída. Um resultado pode ser enviado a mais de um destino através de um operador copiador, que remove um valor de seu enlace de entrada e coloca o valor em seus

enlaces de saída, produzindo assim, cópias dos valores de entrada. Um operador não pode disparar caso exista um dado em algum arco de saída do operador. Um programa representado em Fluxo de Dados é fruto de estudos sobre operação concorrente em sistemas de computadores.

Cada tipo de operador possui enlaces de entrada e saída e especifica uma função (tais como: adição, multiplicação ou comparação) entre os valores dos dados nos enlaces de entrada para os valores dos dados nos enlaces de saída (figura 2.1). Todas as operações são locais para os enlaces de um único operador; operações não possuem efeitos secundários. Cada enlace de dados conecta a saída de um operador a entrada de outro operador; enlaces especificam as dependências de dados num programa.

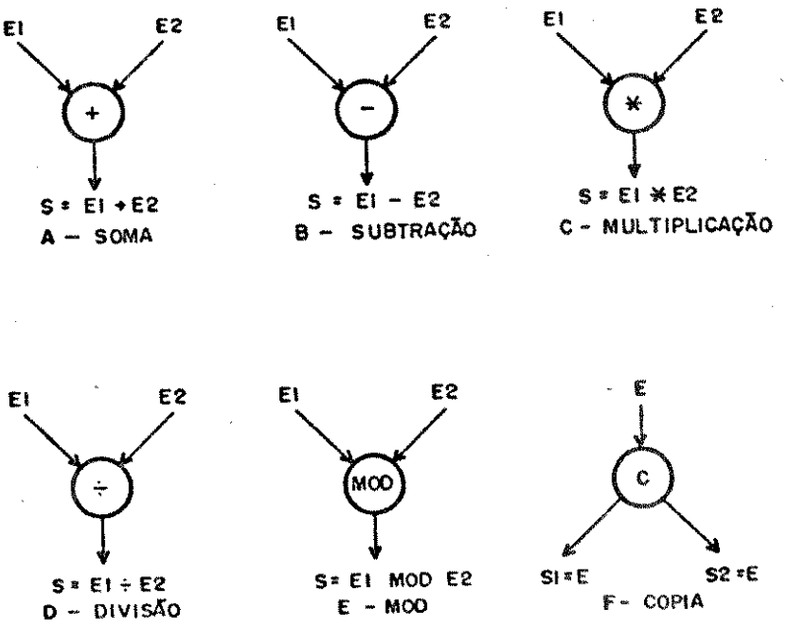
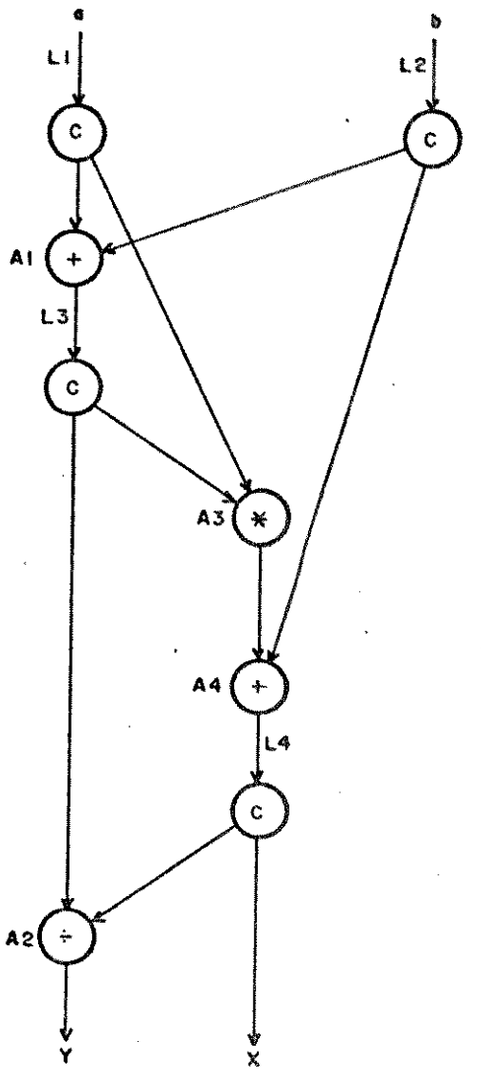


FIGURA 2.1 – ALGUNS OPERADORES FLUXO DE DADOS.

Um exemplo de um programa na linguagem de fluxo de dados é mostrado na figura 2.2 e representa a seguinte computação:

```
" Input a,b;
   x:= (a*(a+b))+b;
   y:= (a+b)/x;
  Output x,y "
```

(a)



(b)

FIGURA 2.2 - O PROGRAMA FLUXO DE DADOS MOSTRADO EM b REPRESENTA O TRECHO DE PROGRAMA CONVENCIONAL MOSTRADO EM a

No programa da figura 2.2 os enlaces L1 e L2 estão inicialmente ativados. O disparo de L1 gera cópias do valor "a" disponível para os operadores A1 e A3, o disparo de L2 apresenta os valores de "b" para os operadores A1 e A4. Uma vez que L1 e L2 foram disparados (numa ordem qualquer), o operador A1 é

ativado porque ele possui valores em cada um de seus arcos de entrada. Após o disparo de A1 (completando a computação "a+b"), o enlace L3 torna-se ativo. O disparo de L3 permitirá o disparo de A2 e A3 concorrentemente, e assim por diante.

2.2.1 - Pacotes de dados

Durante a execução de um programa, valores de dados residem em certos enlaces de dados. Um valor num enlace, chamado **pacote de dado**, representa o resultado intermediário de uma computação ainda não usado .Tipos de dados primitivos consistem de inteiros, ponto flutuante, booleanos e cadeias. Pacotes de dados podem também conter estruturas de valores (descritas a seguir). Cada valor ("token") é completo por si só. Não existem endereços de memória, referências a células compartilhadas ou apontadores para outros pacotes de dados.

2.2.2 - Operações

As operações primitivas de uma linguagem de fluxo de dados são em sua maioria equivalentes as operações primitivas de uma linguagem sequencial: soma, subtração, multiplicação, negação, E, OU, comparação, cópia, etc.. Pode ser usado algum conjunto conveniente de operações universais.

Um operador fluxo de dados ativado (quando todos os enlaces de entrada possuem pacotes de dados), remove os valores de seus enlaces de entrada e computa os valores de saída como função dos valores de entrada. Os valores de saída são emitidos para os enlaces de saída e o operador volta ao estado inativo (figura 2.3). A execução de um operador depende somente da informação local para o operador; não existem variáveis globais

ou efeitos secundarios. Operadores não possuem memória interna entre execuções.

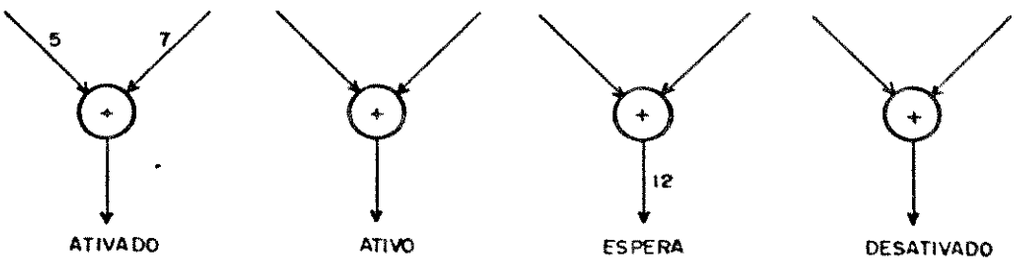


FIGURA 2.3 - EXECUÇÃO DA OPERAÇÃO SOMA

As principais diferenças entre notação fluxo de dados e notação sequencial são:

i - um programa fluxo de dados tem muita informação de controle.

ii - um operador fluxo de dados torna-se ativo em estagios, com a chegada dos operandos.

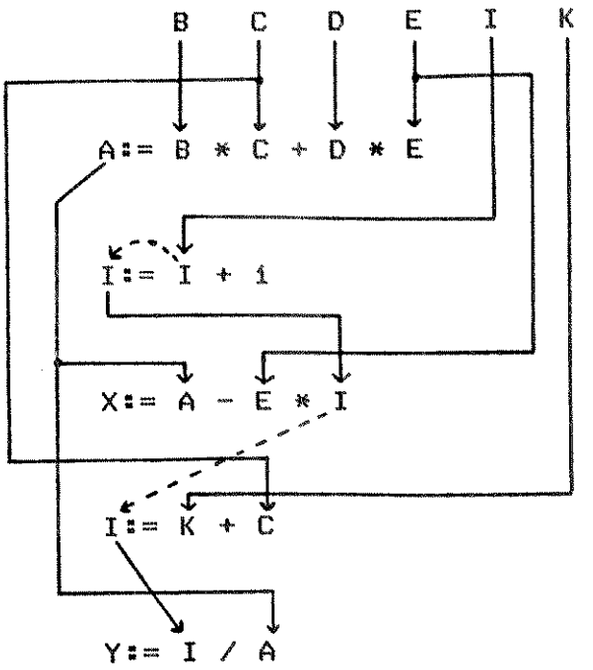
iii - o controle de fluxo e o controle de dados são especificados no mesmo passo de um programa fluxo de dados.

2.2.3 - Procedimento para Transformação de um algoritmo sequencial para a notação fluxo de dados

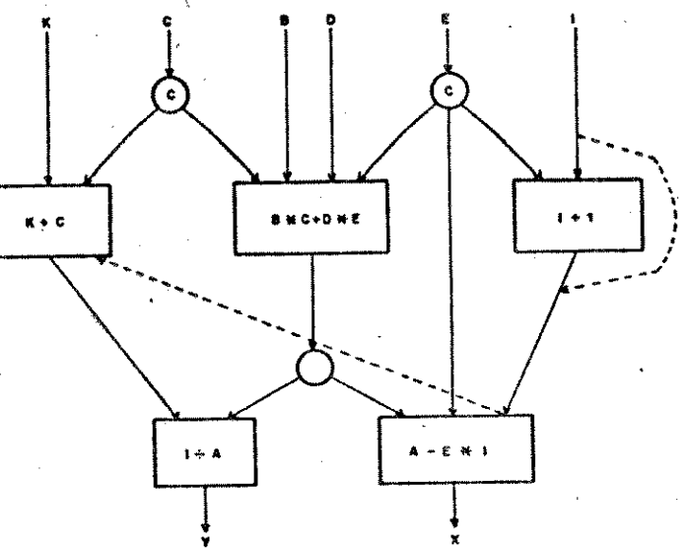
Numa estrutura altamente paralela, o uso de um algoritmo totalmente sequencial é desastroso do ponto de vista de utilização dos recursos disponiveis. Portanto criou-se métodos alternativos de transformação de um algoritmo sequencial para uma forma onde é possivel visualizar-se melhor as interdependências (paralelismo) de operações [GW80].

```
" Begin
    A:= B * C + D * E;
    I:= I + 1;
    X:= A + E * I;
    I:= K + C,
    Y:= I / A
  End "
```

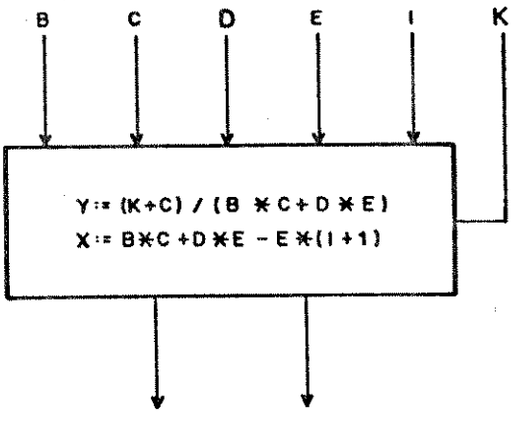
(a)



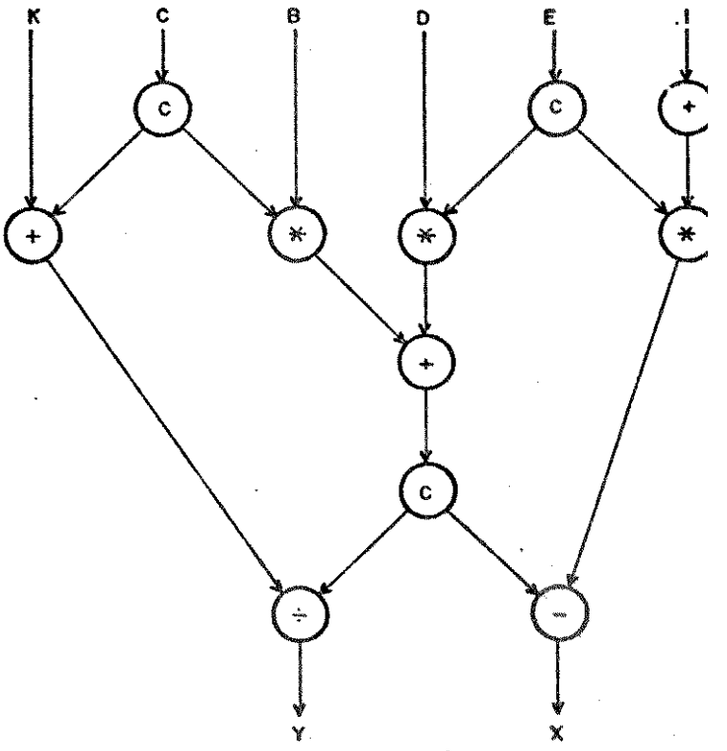
(b)



(c)



(e)



(d)

FIGURA 2.4 - TRANSFORMAÇÃO DE UM ALGORITMO SEQUENCIAL PARA A FORMA PARALELA

Na figura 2.4 temos um exemplo onde tentaremos ilustrar um conceito alternativo de programação. O objetivo maior deste exemplo é mostrar que o uso de um contador de programa é inapropriado para a execução paralela.

No segmento de programa convencional mostrado na figura 2.4a um programa é visto normalmente como uma sequência de avaliações de expressões e atribuições para locações de armazenamento apropriadas.

Um modo alternativo de representação é visto na figura 2.4b onde o conteúdo é agora definido por arcos sobrepostos que indicam certa dependência de tempo entre avaliação de expressões e atribuições. Esta representação é chamada de dependência de dados. Arcos sólidos indicam que dados antes devem ser gerados para depois serem consumidos. Arcos tracejados indicam reutilização de uma locação de memória cujo dado deve ser antes consumido para sua reutilização.

Estes dois modos de representar o programa são equivalentes, pois, os resultados gerados são os mesmos. Entretanto na figura 2.4b (bidimensional) torna-se fácil a identificação de atividades que podem ser executadas em paralelo.

Este simples passo para reinterpretação do significado do programa é tudo o requerido para a transição da computação na forma tradicional para uma computação dirigida por dados (DATA DRIVEN).

A figura 2.4c é apenas a reorganização da figura 2.4b, onde as expressões são avaliadas dentro dos nós. Para permitirmos uma quantidade máxima de atividades paralelas omitiremos todos os arcos tracejados formando a figura 2.4d. Com esta simples mas

importantíssima mudança nós abandonamos o conceito tradicional entre variáveis e locação de armazenamento. Esta é uma consequência necessária para a execução paralela de um programa.

Finalmente na figura 2.4e nós temos o programa principal visto como um nó que consome seus valores de entrada e produz dois valores de saida.

É claro que nenhum programa prático consiste somente de atribuições para variáveis escalares. Portanto estruturas de controle tais como: condicionais, iterações e recursão serão oportunamente modeladas.

2.2.4 - Operadores de Controle

A representação de condicionais e iterações na forma fluxo de dados requer tipos adicionais de operadores primitivos.

O primeiro refinamento para habilitar o computador a fluxo de dados a realizar processamentos condicionais é a definição do operador de comutação mostrado na figura 2.5, que seleciona um dos dois enlaces de saida para colocar o dado no seu enlace de entrada, de acordo com o estado de um segundo enlace de entrada lógico (booleano).

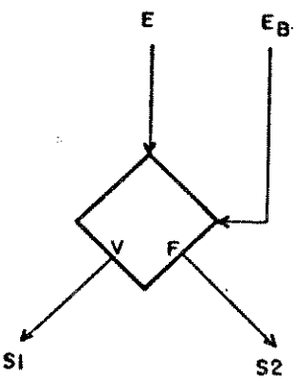


FIGURA 2.5 - OPERADOR COMUTAÇÃO

O segundo refinamento é a definição do operador união que tem dois enlaces de entrada e um enlace de saida (figura 2.6).

Quando um valor (token) aparece num de seus arcos de entrada ele transfere esse valor para seu enlace de saida.

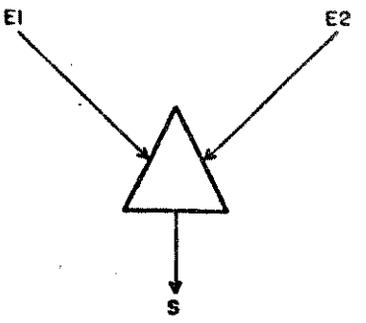


FIGURA 2.6 - OPERADOR UNIÃO

Outros operadores necessários para a construção de condicionais e iterações são mostrados na figura 2.7, onde temos os operadores booleanos e relacionais.

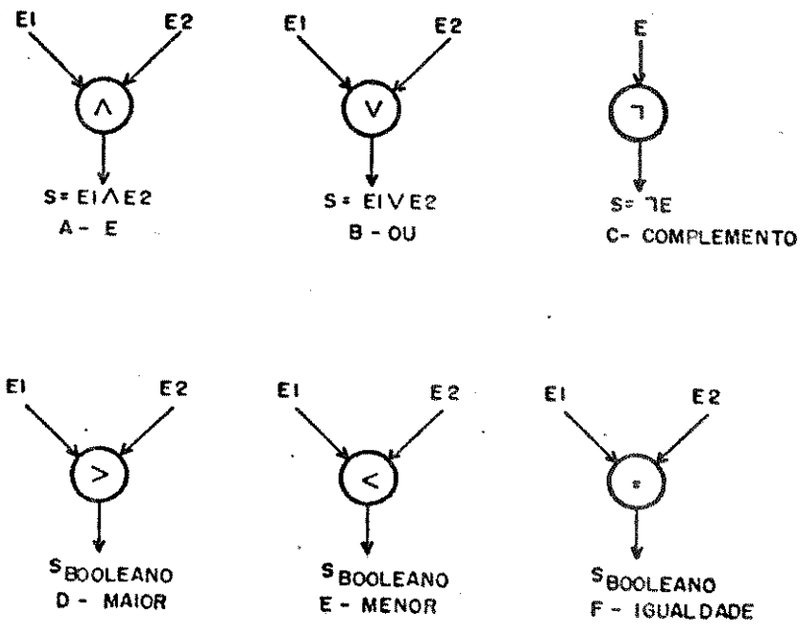


FIGURA 2.7 - OPERADORES BOOLEANOS E RELACIONAIS

Um exemplo de uma expressão condicional escrita numa linguagem de alto nivel, que calcula o valor absoluto de A, é

mostrado na figura 2.8.

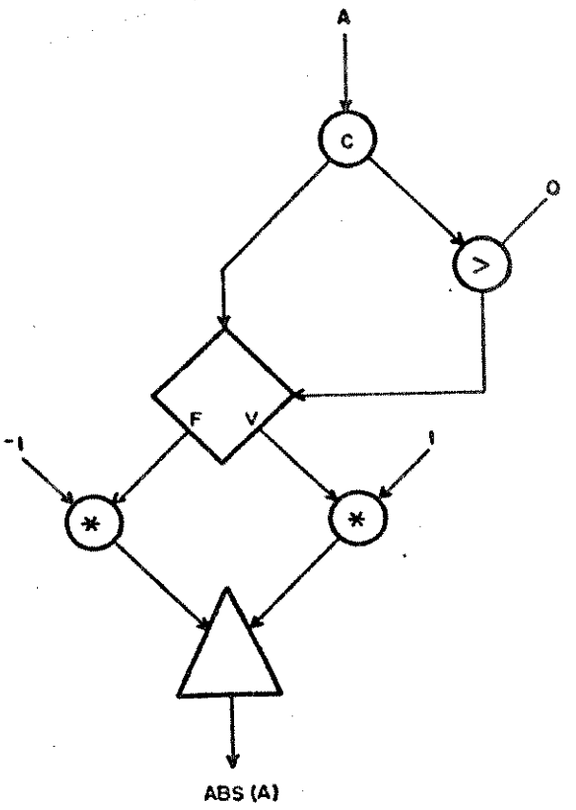


FIGURA 2.8 - CALCULO DO VALOR ABSOLUTO DE "A"

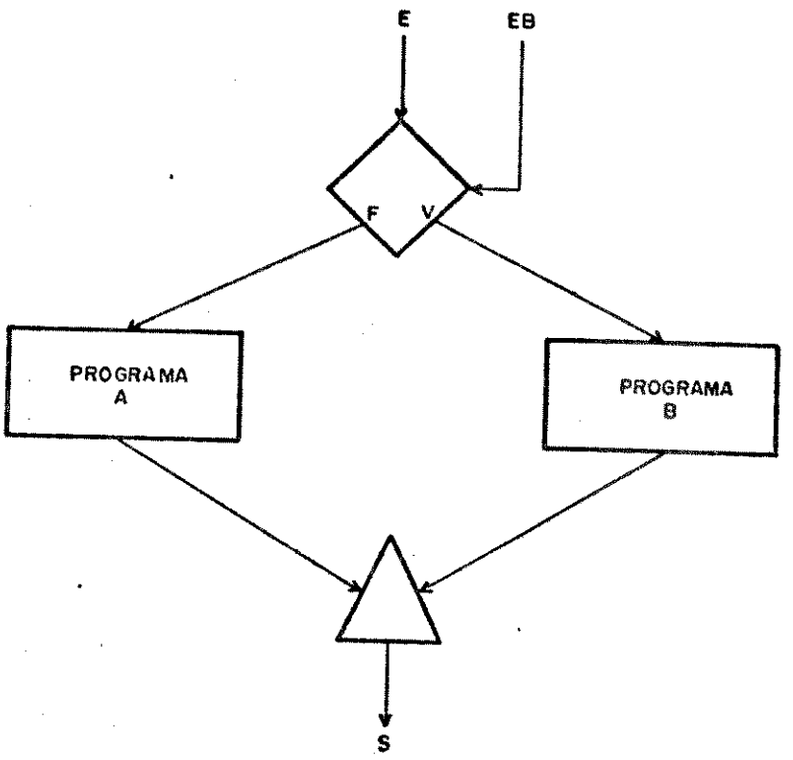


FIGURA 2.9 - CONDICIONAL

Podemos ver então que uma condicional possui uma comutação no seu inicio, dois braços e uma união no final. Cada braço deve ser um programa fluxo de dados disjunto do outro braço e do resto do programa (figura 2.9). Todas as entradas paras os braços entram através do operador comutação e os resultados saem através do operador união.

Uma iteração (loop) mostrado na figura 2.10, possui uma união no início e um comutador no final. Os corpos da iteração devem ser disjuntos entre si e do resto do programa.

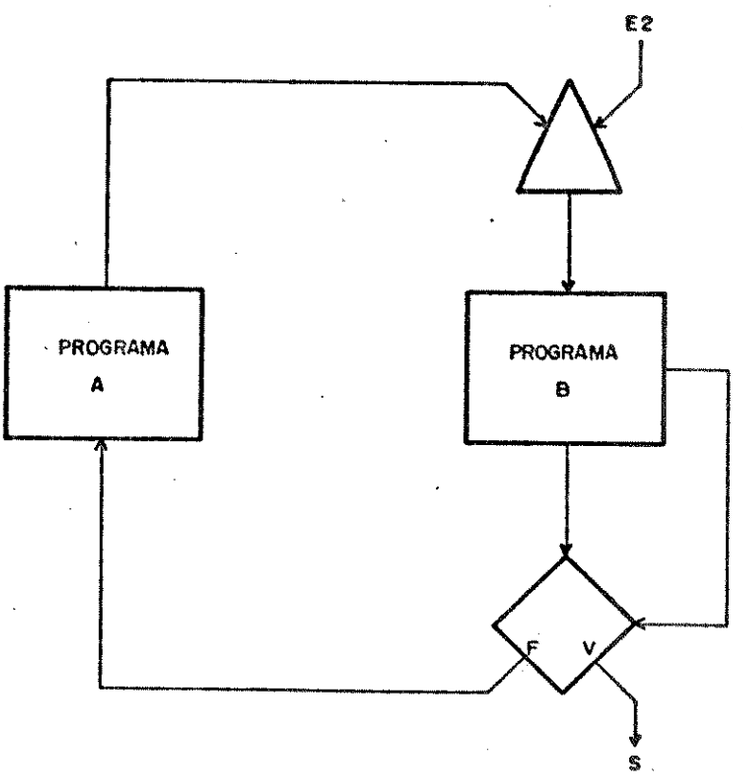


FIGURA 2.10 - ITERAÇÃO

Em um programa sintaticamente correto, valores adicionais não podem entrar em condicionais ou iterações durante sua execução. O comportamento externo de condicionais ou iterações portanto assemelham-se a um operador funcional (primitivo).

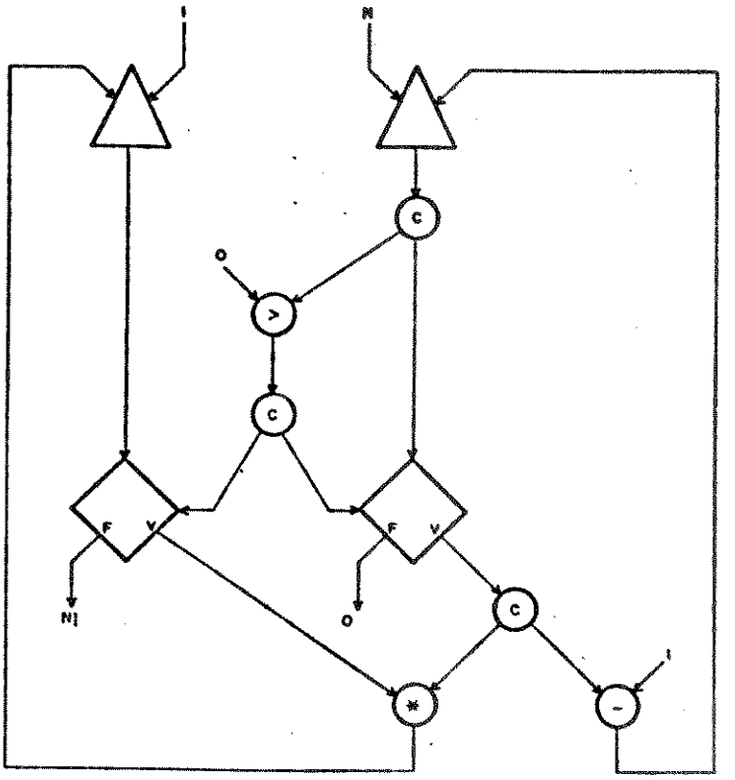
Já vimos que usando operadores primitivos de uma linguagem

fluxo de dados básica, condicionais e iterações podem ser representadas facilmente. Como ilustração mostraremos na figura 2.11b, o programa fluxo da dados que avalia o fatorial de um número N (N!).

Numa liguagem de programação convencional o programa pode ser expresso como mostra a figura 2.11a.

```
" Begin
   X:= N;
   Y:= 1;
   While X>0 do
    Begin
      Y:= Y*X;
      X:= X-1
    End
  End "
```

(a)



(b)

FIGURA 2.11 - COMPUTAÇÃO ITERATIVA DO FATORIAL DE UM NÚMERO N.

Entretanto este exemplo simples demonstra um dos maiores problemas associados aos grafos cíclicos. O problema surge porque um dos valores usados "y" não é usado para avaliar a condição de controle da iteração (loop). Se, por exemplo, a operação de multiplicação demorar mais quando comparada com outros operadores primitivos, a iteração irá gerar muitos v alores de "x" assincronamente; para superar esta dificuldade podemos recorrer a

pilhas para empilhar os pacotes de dados nos enlaces L,X e M; enquanto esperam a criação do correspondente pacote de dado y.

2.2.5 - Representação de um Programa Fluxo de Dados numa Linguagem de Máquina.

Uma representação de um programa fluxo de dados mais proxima de uma linguagem de máquina usada em protótipos fluxo de dados é importante para o entendimento do funcionamento destas máquinas [De80]. Neste esquema, um programa fluxo de dados é uma coleção de células de instruções.

Uma célula de instrução correspondendo ao operador soma é mostrada na figura 2.12, onde existem quatro campos:

Código de Operação - especifica a operação a ser realizada

2 Receptores - para conter os operandos (pacotes de dados).

Destino - especifica a instrução sucessora.

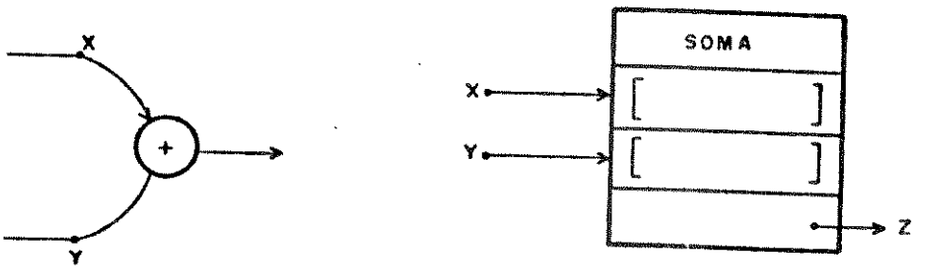


FIGURA 2.12 - ATIVIDADE RELATIVA AO OPERADOR SOMA

Se a instrução a ser executada possuir mais que uma célula de instrução sucessora nós usaremos operadores cópia em um número suficiente para suprir todas as instruções sucessoras .

A figura 2.13b mostra como as células de instruções são

unidas para representar o programa fluxo de dados especificado na figura 2.13a.

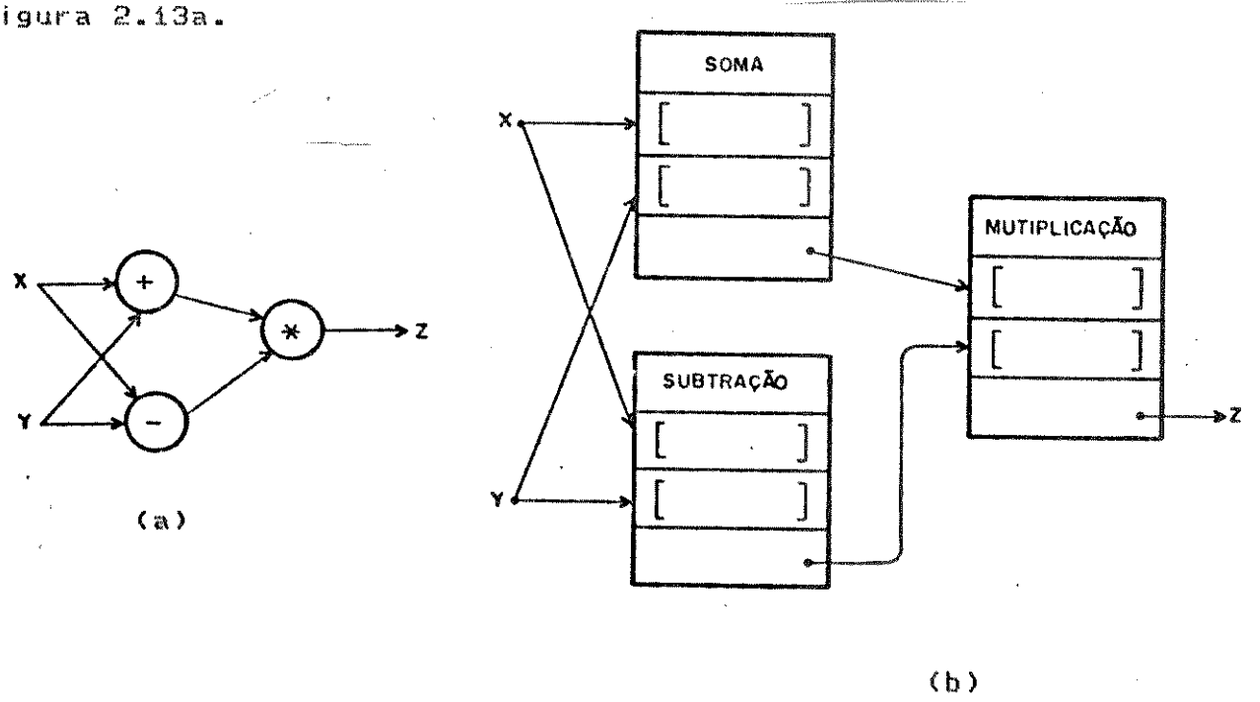


FIGURA 2.13 - EXEMPLO DE REPRESENTAÇÃO DE UM PROGRAMA FLUXO DE DADOS NA MEMÓRIA.

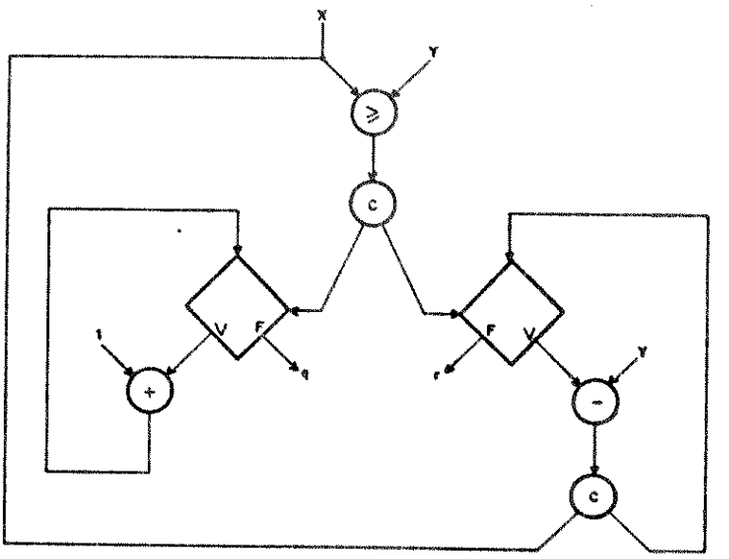
Cada campo destino especifica um campo receptor dentro de alguma atividade sucessora, isto é:

DESTINO:=<ENDEREÇO,RECEPTOR>

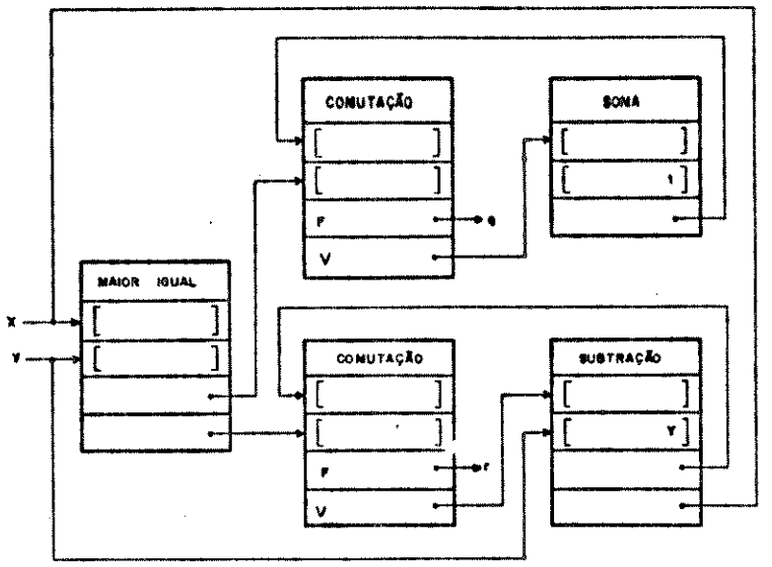
A firura 2.14a mostra um trecho de programa em linguagem sequencial convencional que realiza a divisão inteira de "x" por "y", a figura 2.14b mostra este mesmo programa na linguagem fluxo de dados. A figura 2.14c mostra a representação do programa na memória de um computador à fluxo de dados. As células de instruções que formam o programa representam operadores fluxo de dados que podem apontar para até duas instruções sucessoras. A representação da instrução na memória é, em geral, formada pelos campos: código de operação, operandos e destinos.

```
" Begin
   q:= 0;
   r:= x;
   While r>=y do
    Begin
     q:= q+1;
     r:= r-y
    End
  End "
```

(a)



(b)



(c)

FIGURA 2.14 - ESQUEMA DE UMA ATIVIDADE ITERATIVA QUE REALIZA A DIVISÃO DE X POR Y E SUA REPRESENTAÇÃO NA MEMORIA

A execução de um programa na forma de linguagem de máquina consistindo de células de instruções é visto como segue. O conteudo de uma célula de instrução com presença dos operandos nos recptores é um pacote de operação da forma:

PACOTE DE OPERAÇÃO:=<CÓDIGO OPERAÇÃO,OPERANDO,DESTINOS>

Um pacote que especifica um resultado tem a forma:

PACOTE DE DADO::=<VALOR,DESTINO>

2.3 - MECANISMO BÁSICO DE EXECUÇÃO DE INSTRUÇÃO DE UMA MÁQUINA FLUXO DE DADOS

O mecanismo básico de execução de instrução fluxo de dados é mostrado na figura 2.15, e nas referências [DM75,De80,GKW85,Ru77, TBH82].

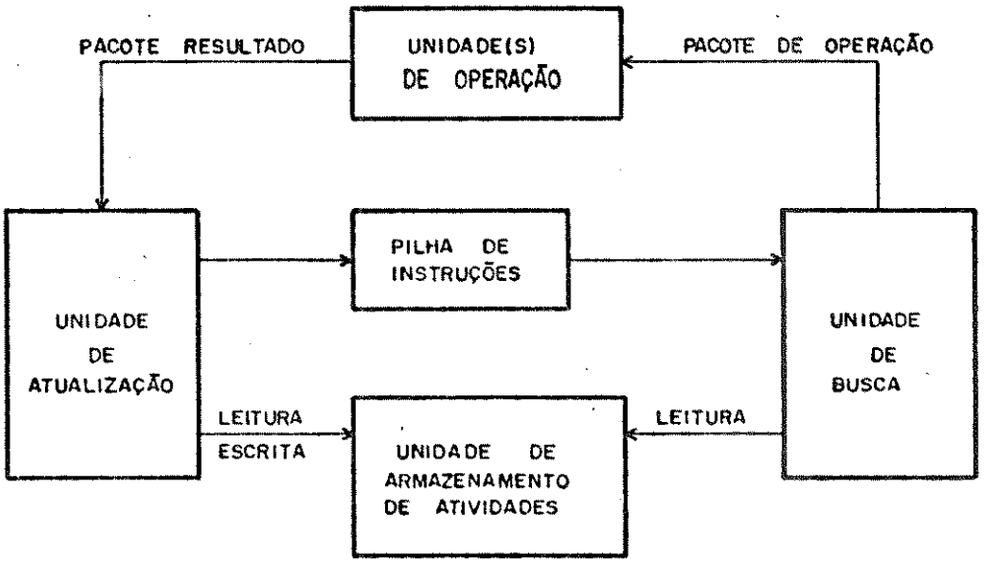


FIGURA 2.15 - MECANISMO BÁSICO DE EXECUÇÃO DE INSTRUÇÃO

Um programa fluxo de dados que descreve uma computação a ser realizada é uma coleção de atividades contidas na unidade de armazenamento de atividades. Cada atividade possui um único endereço que é colocado na pilha de instrução (FIFO), quando a instrução estiver pronta para a execução (com todos os seus operandos presentes nos campos receptores). A unidade de busca

pega o endereço da instrução na pilha de instrução e le a atividade a ser realizada na unidade de armazenamento de atividades, formando um pacote de operação. Este pacote de operação é enviado para a unidade de operação. A unidade de operação realiza a atividade especificada pelo código de operação sobre os operandos (valores), gerando um pacote de dado para cada campo do destino do pacote de operação. A unidade de atualização recebe o(s) pacote(s) resultado(s) e introduz o(s) valor(es) contido(s) nele(s) no(s) campo(s) de operando(s) (campos recptores) da atividade especificada pelo seu campo de destino. A unidade de atualização também testa e reconhece se todos os operandos requeridos para ativar a instrução foram recebidos e, se isto aconteceu, ela introduz o endereço da instrução na pilha de instrução.

Durante a execução do programa o número de entradas na pilha de instrução mede o grau de concorrência presente no programa. O mecanismo da figura 2.15 pode utilizar este potencial num grau limitado mas significante; uma vez que a unidade de busca enviou um pacote para a unidade de operação, ela pode imediatamente ler outro endereço na pilha de instrução sem esperar o término do processamento da instrução previamente enviada. Então uma cadeia contínua de pacotes de operação podem fluir da unidade de busca para a unidade de operação enquanto a pilha de instrução contiver endereços (não estiver vazia).

Vários pacotes podem estar fluindo simultaneamente em diferentes partes do anel e portanto o anel executa concorrentemente instruções diferentes. Então o anel opera como um sistema "pipeline", com todas suas unidades processando ativamente um pa-

cote por vez. O grau de concorrência possivel é limitado pelo número de unidades do anel e do grau "paralelismo pipeline" dentro de cada unidade. O aumento de concorrência pode ser explorado desdobrando-se alguma unidade do anel em várias unidades que podem ser alocadas para atividades concorrentes.

O grau de concorrência pode aumentar enormemente conectando-se vários elementos de processamento (PE) em paralelo de modo a realizarmos um sistema de multiprocessamento fluxo de dados, como mostra a figura 2.16.

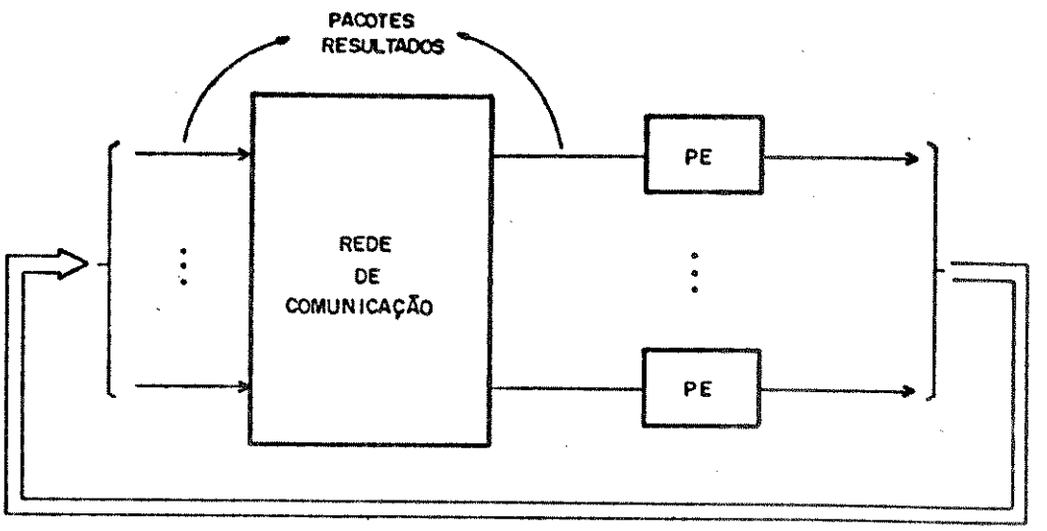


FIGURA 2.16 - MULTIPROCESSADOR FLUXO DE DADOS

2.4 - Estágio Atual no Assunto

Devido aos motivos realçados anteriormente [DM75,Ru77] surgiram recentemente um grande número de trabalhos sobre computadores com arquitetura fluxo de dados. Tentaremos agora descrever algumas implementações de arquiteturas elaboradas por diferentes grupos de pesquisadores [TBH82].

2.4.1 - Computador Fluxo de Dados do Instituto de Tecnologia de Massachusets (MIT).

A contribuição do projeto MIT para as pesquisas de computatores em arquitetura em fluxo de dados foi significativa, formando a base de muitos outros trabalhos fluxo de dados.

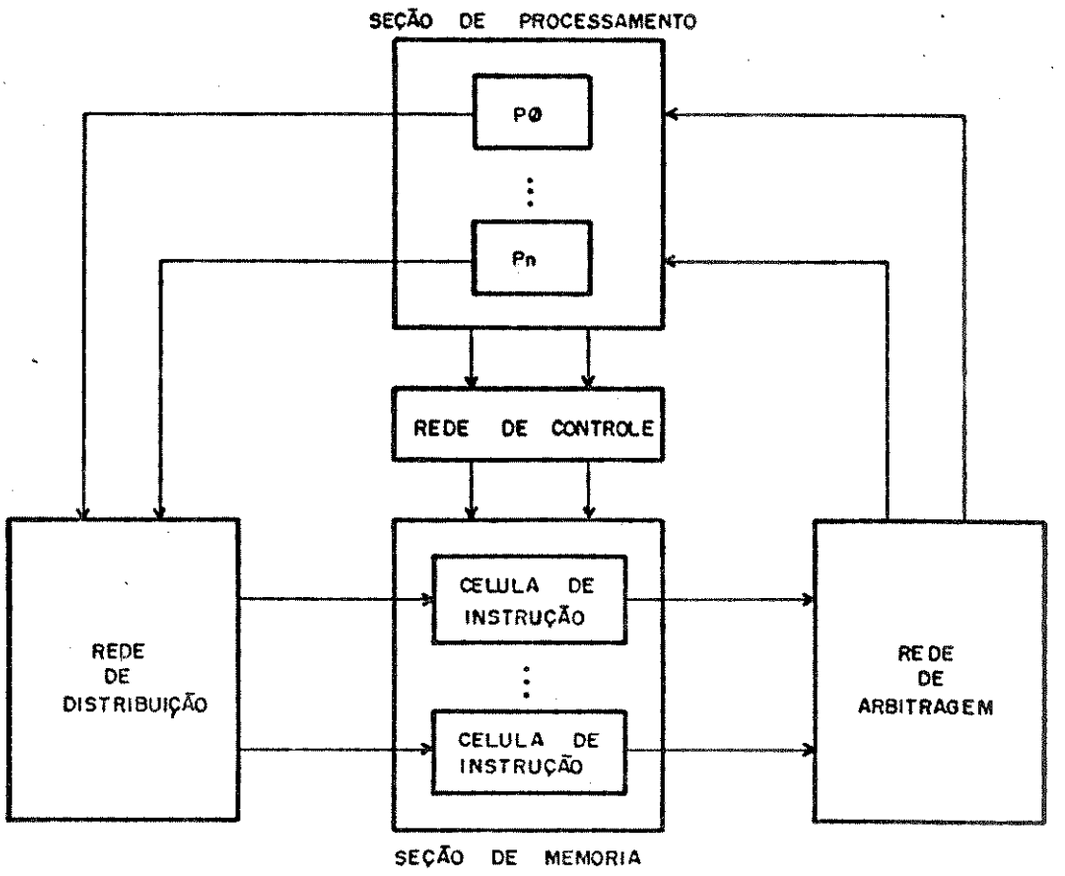


FIGURA 2.17 - COMPUTADOR FLUXO DE DADOS DE M.I.T.

A organização da máquina fluxo de dados MIT é apresentada na figura 2.17. Ela consiste de unidades conectadas por canais através dos quais, pacotes de informação caminham de modo assíncrono. Estas unidades são:

a) **Seção de Memória**- consistindo de Células de Instrução que

contem instruções e seus operandos.

b) Seção de Processamento - consistindo de elementos processadores especiais, que realizam operações sobre pacotes de dados.

c) **Rede de Arbitragem**- distribui pacotes de instruções executáveis da Seção de Memória para a Seção de Processamento.

d) **Rede de Controle** - distribui pacotes de controle da Seção de Processamento para a Seção de Memória.

e) **Rede de Distribuição**- distribui pacotes de dados da Seção de Processamento para a Seção de Memória.

2.4.2 - Processador de Dados Distribudos da TEXAS INSTRUMENTS

O Processador de Dados Distribuidos (DDP) é um sistema projetado pela Texas Instruments para investigar o potencial do fluxo de dados como base para computadores de alto desempenho.

A organização do processador DDP é mostrada no diagrama de blocos da figura 2.18. Ele consiste de cinco elementos de computação independentes; quatro computadores fluxo de dados idênticos executam a computação e, um computador 990/10 da Texas Instruments atua como processador de entrada e saída. Estes cinco elemento de computação do DDP são conectados por um registrador deslocador circular formalmente conhecido como um anel DCLN. Este registrador deslocador está ligado a cada elemento, portanto ele pode enviar pacotes de comprimento variável para qualquer elemento do anel.

Cada computador fluxo de dados consiste de quatro unidades especiais que são:

a) Unidade Aritmética- processa instruções executáveis.

b) Memória de Programa- constituida por uma memória de acesso aleatório (RAM) que contem instruções de fluxo.

c) Controlador Atualizador- atualiza as instruções com pacotes de dados.

d) Pilha de Instruções Pedentes- contendo instruções executáveis esperando para serem disparadas.

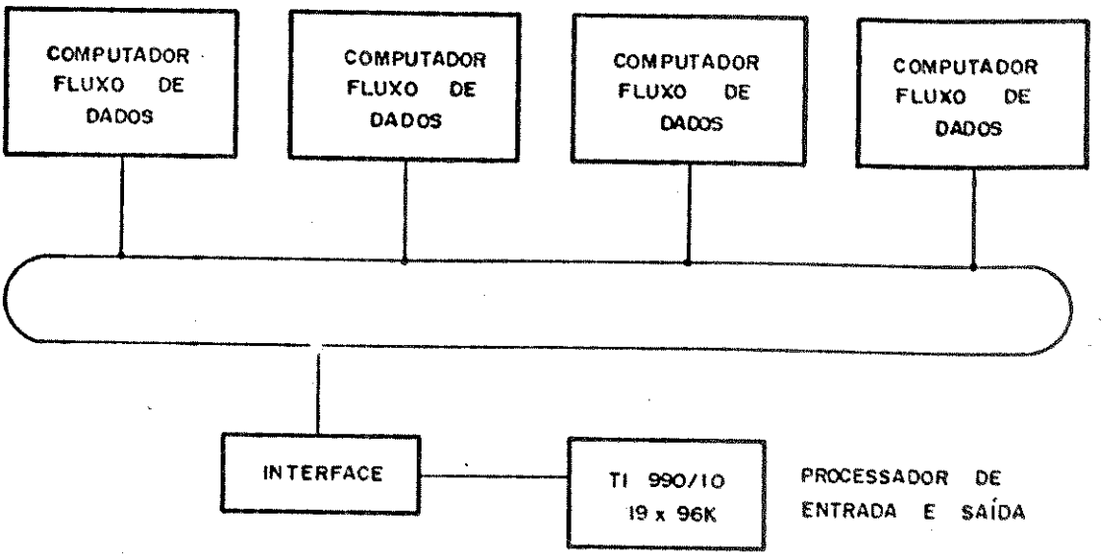


FIGURA 2.18 - PROCESSADOR DE DADOS DA TEXAS INSTRUMENTS

Instruções executáveis são removidas da Pilha de Instruções Pendentes pela unidade aritmética e processadas. Quando termina a execução da instrução, uma série de pacotes de dados (tokens) são enviados para o controlador atualizador. Usando a região de endereço do pacote, o controlador atualizador armazena o operando na instrução endereçada e decrementa de "um" o contador de operandos desta instrução. Se este contador torna-se zero a instrucao está pronta para ser executada, então uma cópia é colocada na pilha de instruções pendentes e o processo é

reinicializado.

2.4.3 - Máquina Dirigida por Dados de Utah

A Máquina Dirigida por Dados 1 (DDM1) é um elemento de computação com uma arquitetura a fluxo de dados estruturada recursivamente.

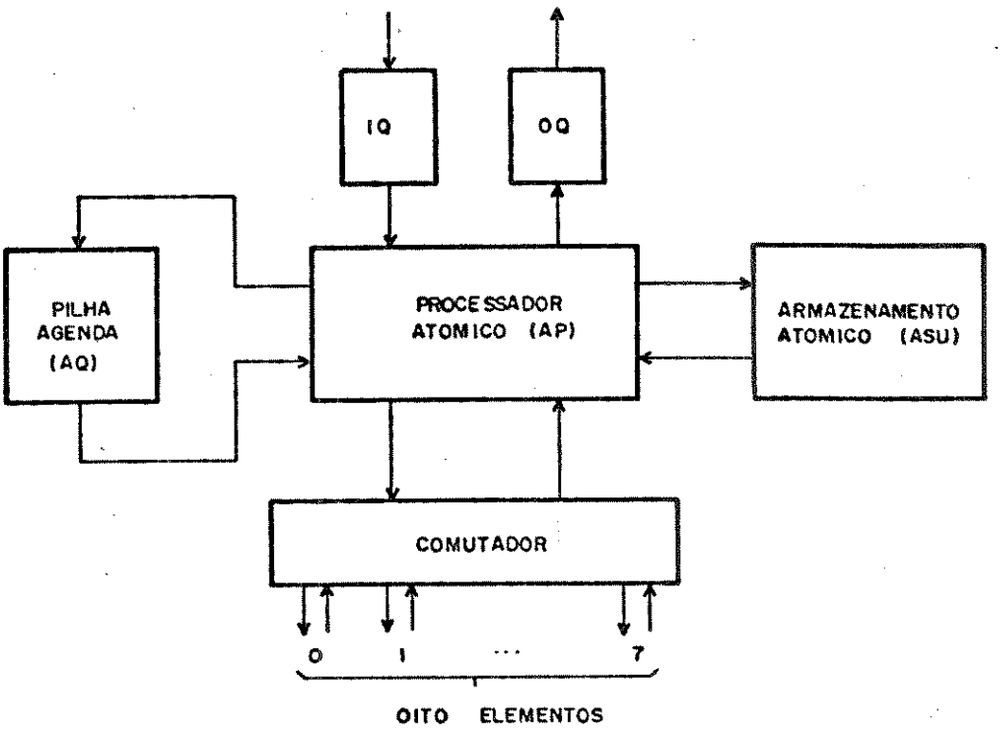


FIGURA 2.19 - MAQUINA DIRIGIDA POR DADOS DE UTAH

A figura 2.19 mostra o elemento de processamento DDM1 que consiste de seis unidades principais:

a) **Unidade de Armazenamento Atômico (ASU)**- contendo a memória do programa.

b) **Processador Atômico (AP)**- executa instruções.

c) **Pilha Agenda(AQ)**- armazena mensagens da unidade de armazenamento atômico.

d) Pilha de Entrada (IQ)- são buffers de entrada de mensagens do elemento de computação superior.

e) Pilha de Saida (OQ)- são buffers de mensagem para o elemento superior.

f) Comutador- conecta os elementos de computação com os oito elementos inferiores.

2.4.4 - Máquinas Fluxo de Dados de Irvine

A Máquina de Fluxo de Dados de Irvine foi motivada pelo desejo de explorar o potencial da tecnologia VLSI e a realização de programas altamente concorrentes.

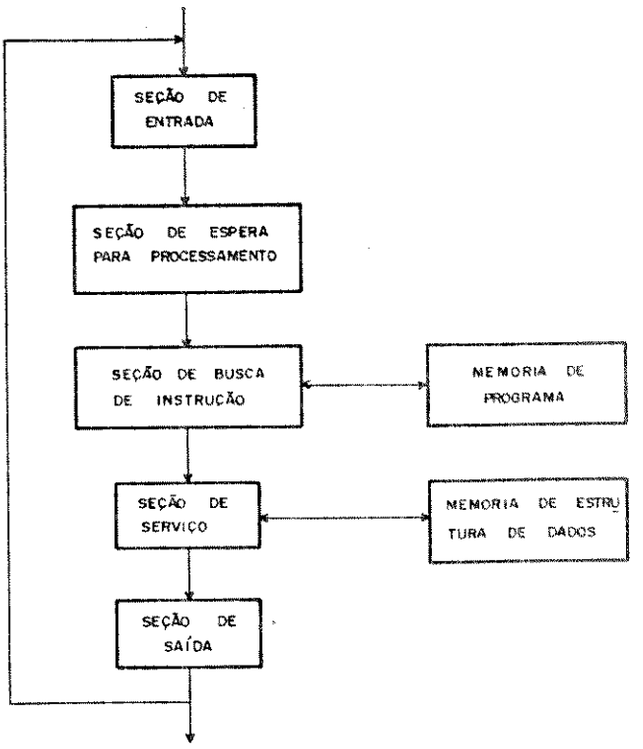


FIGURA 2.20 - ELEMENTO DE PROCESSAMENTO DE IRVINE

A figura 2.20 ilustra o elemento de processamento proposto na Máquina Fluxo de Dados de Irvine. Cada elemento é

essencialmente um computador completo com um cojunto de instruções com 16K palavras para armazenamento de programas e estruturas de dados.

Estes elementos especiais incluem:

a) **Seção de Entrada**- aceita mensagens provenientes de outros elementos de processamento.

b) **Seção de Espera para Processamento**- contem pacotes de dados para as instruções consumidoras (sucessoras).

c) **Seção de Serviço**- Unidade Lógica Aritmética com ponto flutuante.

d) **Seção de Saída**- envia pacotes de resultados (dados) para outros elementos de processamento.

2.4.5 - Computadores Fluxo de Dados de Manchester

O projeto fluxo de dados da Universidade de Manchester, como muitos outros projetos, foi idealizado com o principal objetivo de obter um computador de alto desempenho.

A figura 2.21 mostra o diagrama de blocos do computador fluxo de dados de Manchester. Ele é composto das seguintes principais unidades:

a) **Comutador**- realiza as funções de entrada e saída

b) **Pilha Pacote de Dado**- realiza armazenamento temporário de um pacote de dado.

c) **Unidade de Disparo**- armazena instruções executáveis.

d) **Armazenamento de Instruções**- memória contendo programas de fluxo de dados.

e) **Unidade de Processamento**- consistindo de vários elementos processadores idênticos, executa instruções.

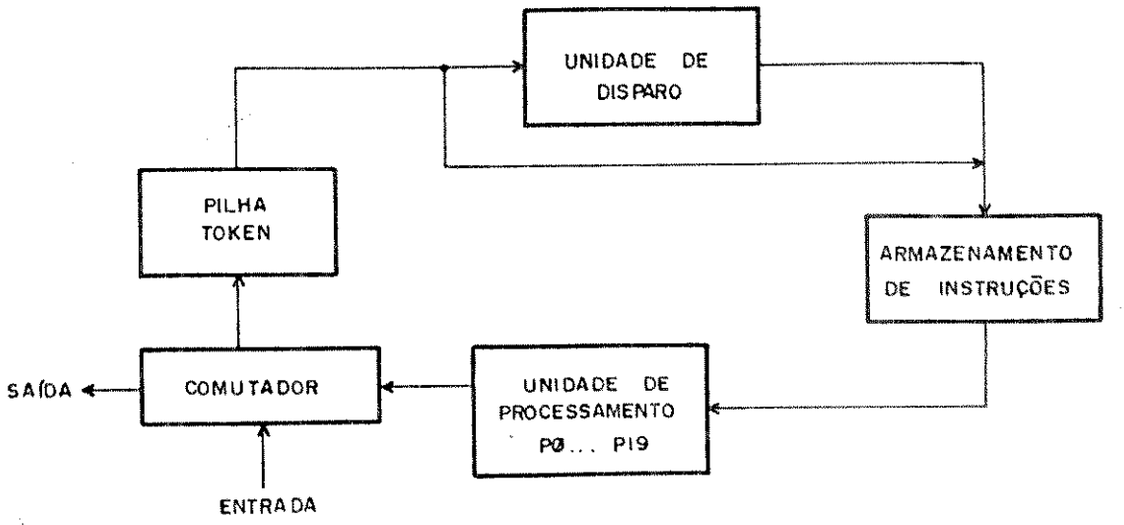


FIGURA 2.21 - COMPUTADOR FLUXO DE DADOS DE MANCHESTER

O comutador é usado para realizar a comunicação da máquina com o exterior (periféricos ou outros computadores). Para inicializar a execução de um fragmento de programa, pacotes de dados de inicialização são inseridos através do comutador para inicializar a computação. No final do programa são inseridos instruções especiais que geram pacotes de dados de saida.

2.4.6 - Sistema LAU de Toulouse

O sistema LAU é um computador dirigido por dados ,projetado para executar a "Linguagem de Atribuição Única" (LAU), a qual foi usada para programar um grande número de problemas usando o conceito de fluxo de dados. Uma breve descrição do computador LAU é feita na figura 2.22. Ele compreende:

a) **Unidade de Memória**- realiza o armazenamento de instruções de dados.

b) **Unidade de Controle**- realiza o controle da memória.

c) **Unidade de Processamento**- consiste de 32 elementos de processamento idênticos.

A parte mais interessante do projeto é a unidade de controle onde o contador de programa de Von Neumann é trocado por duas memórias:

- memória de irstrução de controle (ICM)
- memória de controle de dados (DCM)

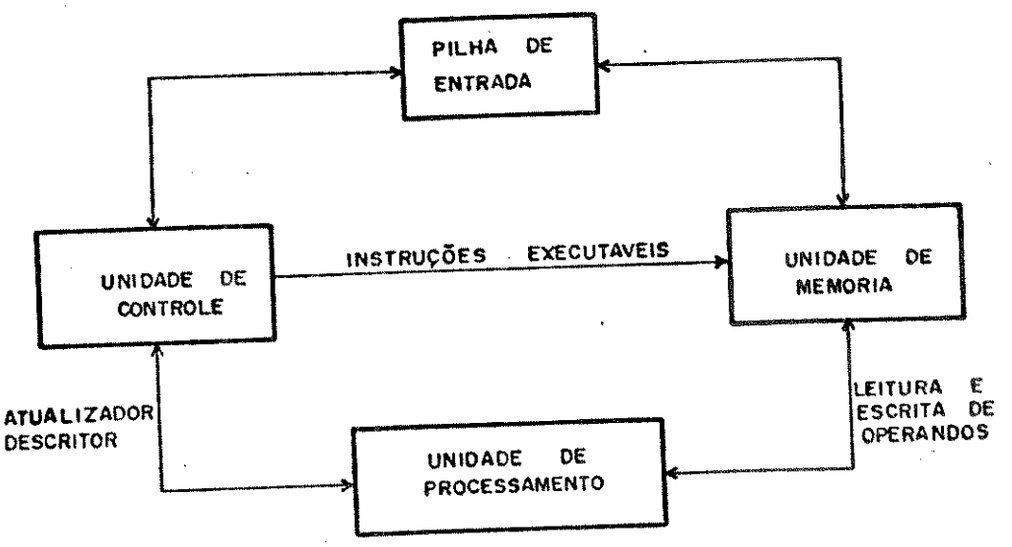


FIGURA 2.22 - SISTEMA LAU DE TOULOUSE

2.4.7 - Computador Fluxo de Dados de Newcastle

O computador descentralizado de propósitos gerais de Newcastle é o resultado da combinação de várias organizações fluxo de dados num único computador.

O diagrama de blocos da figura 2.23 descreve o computador fluxo de dados de Newcastle. Ele consiste de tres blocos básicos:

a) **Unidade de Disparo**- controla a ativação de instruções.

b) **Unidade de Memória**- armazena instruções e dados.

c) Unidade de Processamento- executa instruções e distribui resultados.

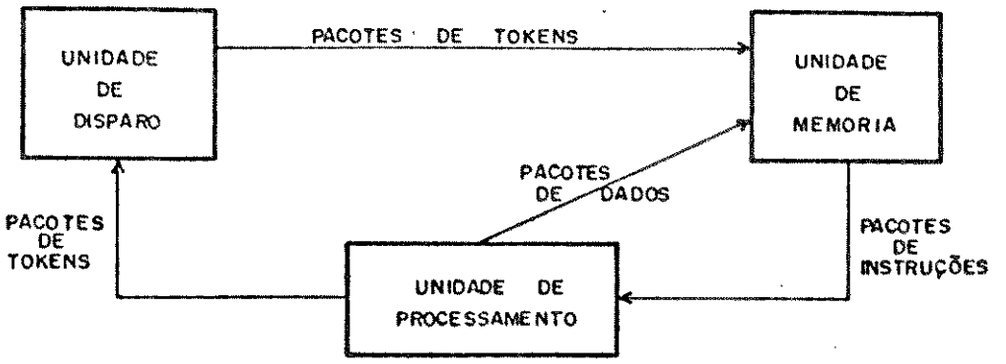


FIGURA 2.23 - COMPUTADOR FLUXO DE DADOS DE NEWCASTLE

2.5 - Um Exemplo de Central à Fluxo de Dados

Um sistema de comutação controlado por computador Fluxo de Dados foi construido no Departamento de Engenharia Elétrica da Universidade de Tokyo [AYM84]. O protótipo experimental da Central Fluxo de Dados foi denominado "DATAFLEX-1". A figura 2.24 mostra a organização do sistema de controle DATAFLEX-1.

As funções dos subsistemas contidos na figura 2.24 são as seguintes:

Elementos Processadores (PEs) - realizam as funções de processamento de chamadas telefônicas de maneira compartilhada.

Memória Celular (MC) - armazena programas fluxo de dados e dados de controle. O comprimento da palavra é de 32 bits.

Rede de Arbitragem (ANET) - distribui pacotes de instruções para os devidos PEs; procura a célula de instrução executável cujo endereço é apontado pelo topo de uma fila. Lendo o código de operação da célula , a ANET envia um

pacote de instrução para o PE designado, via "Bus de Endereço" (AAB).

Rede de Destinação (DNET) - recebe o resultado da operação na forma de um pacote de dado do PE, via "Bus de Resposta" (AWB) e, armazena dados na próxima célula de instrução da MC. A DNET tambem detecta a condição de disparo da célula.

Barramento de Endereço e Barramento de Resposta (AAB e AWB)- são barramentos unidirecionais de 32 bits que transportam pacotes de instruções e pacotes de dados.

Rede de Memória (MNET) - é um elemento processador, com a função de controlar a MC.

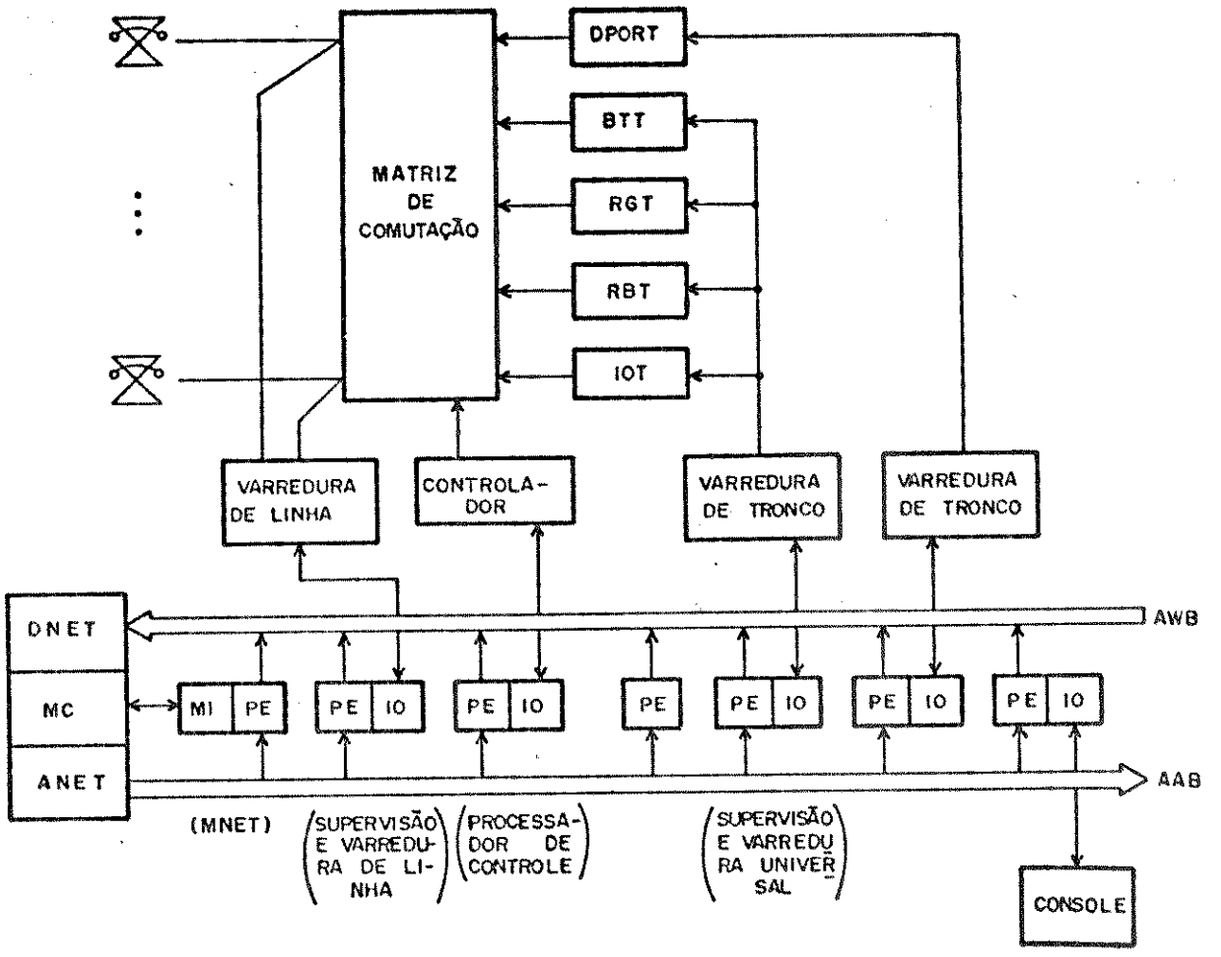


FIGURA 2.24 - ORGANIZAÇÃO DO SISTEMA DE CONTROLE DO DATAFLEX-1

Uma característica excepcional do controle fluxo de dados é a independência de cada primitiva, que é a função a ser distribuida para cada elemento processador. Portanto a liberdade para transferência de funções é relativamente alta. Considerando o consumo de tempo gasto (overhead) para transmitir pacotes de dados e a eficiência devido a operação concorrente, o DATAFLEX-1 adotou funções partilhadas a nível de macro-tarefas.

Com respeito ao controle de recursos comuns, tais como seleção de troncos, união e sinalização de equipamentos, o DATAFLEX-1 adotou o método de controle distribuido para simplificar o hardware. Cada elemento processador toma uma grande parte do controle dos recursos e funções congêneres tais como escrutar, contagem de pulsos discados ou controle de comutação. O sistema foi equipado com uma rede de divisão espacial de voz. O controlador fluxo de dados tem 7 elementos processadores Z80A comunicando-se através de barramento. Foi implementada a comutação interna de um escritório.

2.6 - Conclusão 1

Neste capítulo foram introduzidos os conceitos da teoria de fluxo de dados necessários para dar suporte ao desenvolvimento do trabalho. Em particular é necessário salientar que no fluxo de dados as instruções são executadas de acordo com a disponibilidade de operandos, ou seja, a execução é dirigida por dados ("data-driven"). Onde não é mais necessário o contador de programa de Von Neumann e, o programador não tem mais controle sobre a sequencia de execução de instruções. Mas, permite uma

execução altamente concorrente.

Foram também descritas sucintamente algumas arquiteturas a fluxo de dados estudadas e implementadas por vários grupos de pesquisa, que mostram o estagio atual no assunto, e finalmente foi abordada uma central de comutação experimental onde o controle é feito por uma máquina a fluxo de dados; sendo que os processadores presentes na arquitetura de controle comunicam-se através de barramento.

CAPÍTULO 3

ARQUITETURA PROPOSTA

3.1 - Introdução

O sistema de comutação a fluxo de dados discutido no item 2.5 utiliza uma configuração em forma de barramento para a comunicação entre os processadores. Esse tipo de configuração pode trazer incovenientes quando se pensa na expansão do sistema de comutação. A expansão do sistema significa em colocar mais processadores em paralelo no barramento. Entretanto, um barramento longo ocasiona atrasos consideráveis na transmissão e pode limitar o número de processadores.

A arquitetura do sistema de comutação à fluxo de dados aqui proposta é mais conveniente para expansão da central e mantém todas as vantagens de uma central com processamento paralelo.

A estrutura proposta possui a forma de anel. Essa estrutura possibilita maior velocidade de processamento, pois além do paralelismo dos processadores, utiliza processamento concorrente no anel (paralelismo pipeline). Ela permite facilidade de expansão, pois os anéis podem ser colocados em paralelo, e a interligação entre anéis é feita através de um módulo de comutação [BMb86].

3.2 - Organização Básica da Estrutura de Controle

A figura 3.1 mostra o diagrama de blocos da arquitetura do computador a fluxo de dados que realiza a função de controle da central de comutação.

A estrutura básica do sistema fluxo de dados é formada por um anel com quatro blocos básicos:

- comutador de entrada e saída,

- unidade de ativação,
- unidade de instruções executáveis e,
- unidade de processamento.

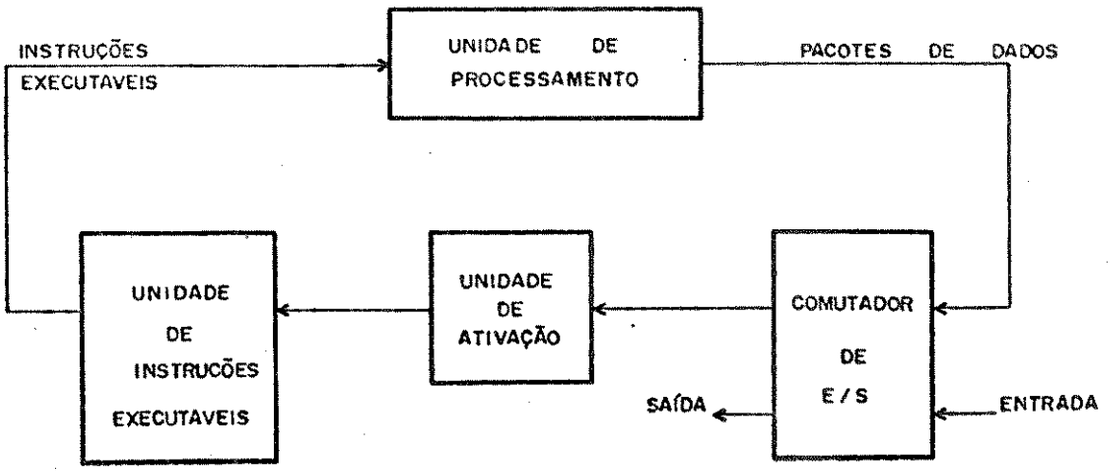


FIGURA 3.1 - DIAGRAMA DE BLOCOS DA ARQUITETURA

As principais atividades exercidas em cada bloco básico são:

COMUTADOR DE ENTRADA E SAÍDA- Permite realizar a comunicação do anel com o exterior (periféricos e outros anéis) permitindo a expansão da arquitetura. Programas são introduzidos no anel através do Comutador de Entrada e Saída.

UNIDADE DE ATIVAÇÃO- Ela contém o programa fluxo de dados e tem a função de verificar a disponibilidade de operandos para o processamento (execução) de uma instrução. Para isto ela recebe pacotes de dados provenientes de uma instrução (nó) predecessora, processada na Unidade de Processamento ou do exterior através do Comutador de Entrada e Saída, analisa

O bloco sistema de arbitragem é formado por um árbitro síncrono e um multiplexador. O árbitro será simplesmente um contador sequencial módulo 16 (onde serão utilizadas apenas os 3 bits menos significativos) que varrerá sequencialmente as filas Transmissoras dos 8 Processadores Elementares. O multiplexador terá a função de transferir o conteúdo da fila apontada pelo árbitro para a via de pacotes de dados.

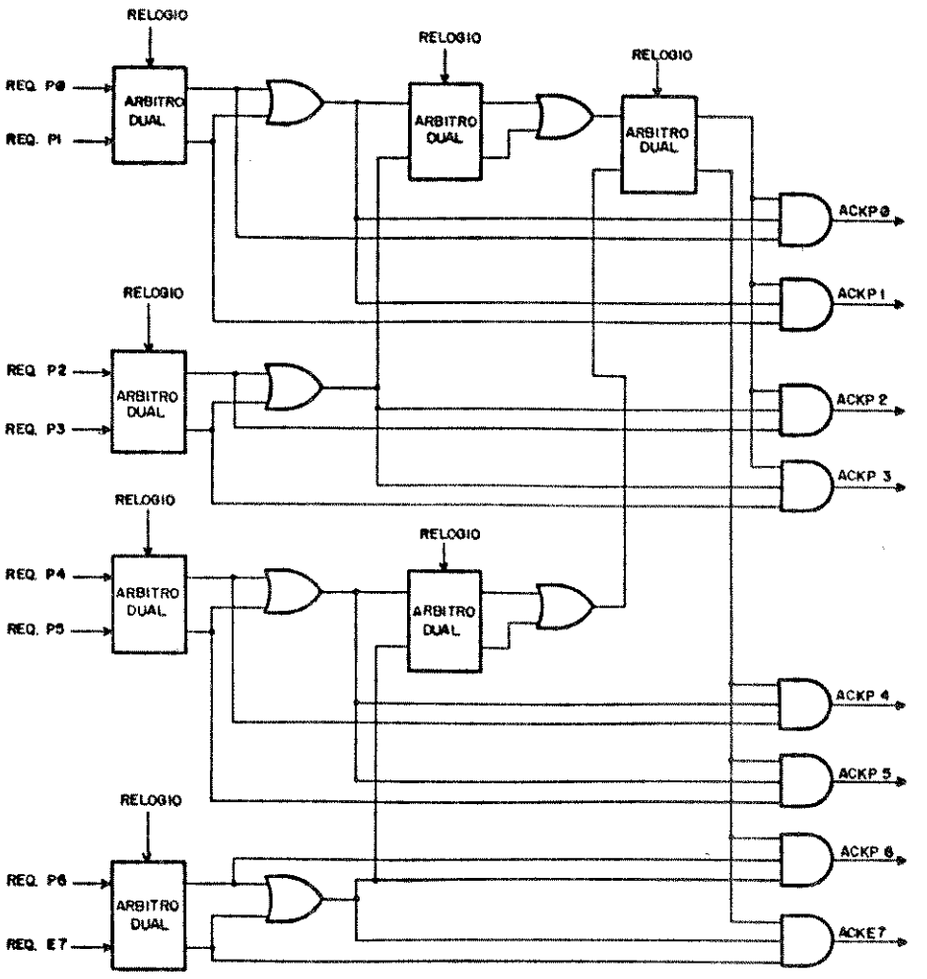


FIGURA 3.16 - SISTEMA DE ARBITRAGEM DA UNIDADE DE PROCESSAMENTO

Uma outra implementação do árbitro pode ser um sistema assíncrono de arbitragem mostrado na figura 3.16, onde a prioridade é dada a primeira requisição de algum processador

a instrução correspondente ao pacote de dado que chegou e quando operandos suficientes para uma dada instrução estão presentes na Unidade de Ativação, esta é enviada para a Unidade de Instruções Executáveis. Senão o pacote de dado é armazenado na Unidade de Ativação a espera de outro pacote de dado.

UNIDADE DE INSTRUÇÕES EXECUTÁVEIS- Desempenha uma atividade moderadora de tráfego. Ela contém uma fila (FIFO) com a função de armazenar instruções executáveis provenientes da Unidade de Ativação quando estas encontram a Unidade de Processamento sobrecarregada.

UNIDADE DE PROCESSAMENTO- Distribui as instruções provenientes da Unidade de Instruções Executáveis para os processadores elementares apropriados de acordo com o código de operação. O processador elementar (microprogramado) processa a instrução e envia o resultado na forma de pacotes de dados para as instruções sucessoras, contidas na Unidade de Ativação, via Comutador de Entrada e Saída; completando o circuito (anel).

A figura 3.2 mostra toda a potencialidade de paralelismo disponível na arquitetura proposta.

Podemos notar na estrutura da figura 3.2 dois níveis de paralelismo. Um na unidade de processamento utilizando-se vários processadores elementares em paralelo. Outro nos quatro blocos do anel (Unidade de Processamento, Unidade de Entrada e Saída, Unidade de Ativação e Unidade de Instruções Executáveis) que trabalham com atividades sobrepostas (paralelismo pipeline).

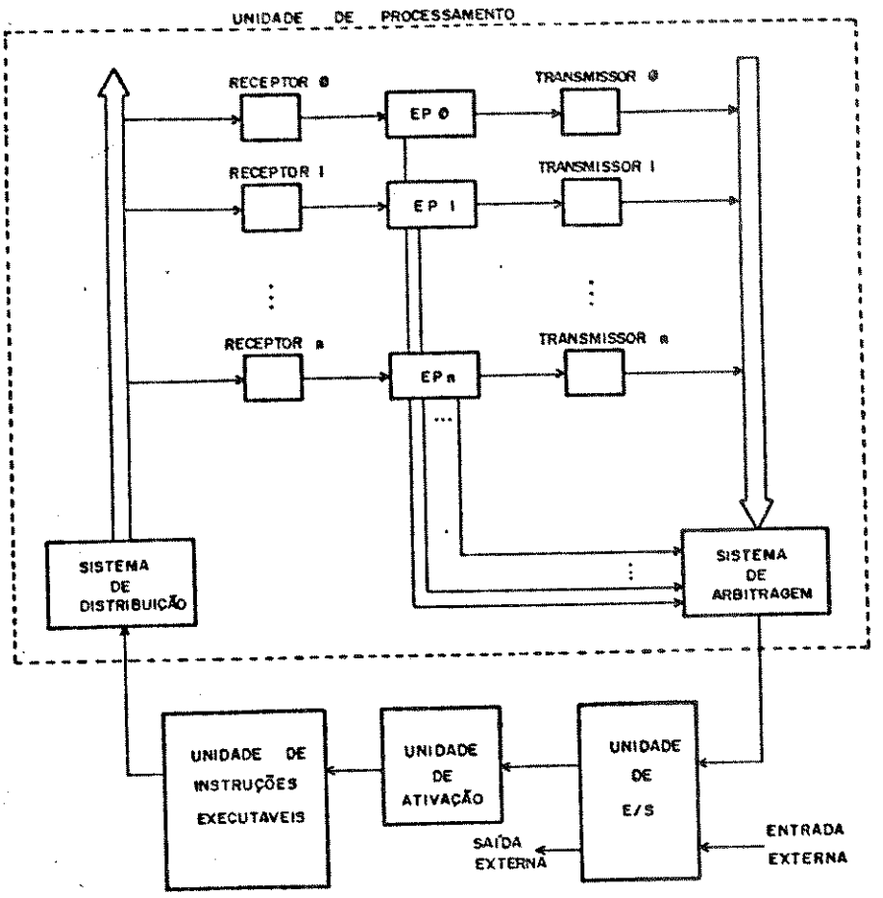


FIGURA 3.2 - ORGANIZAÇÃO DO SISTEMA DE CONTROLE

3.3 - Formatos de Instruções e Dados

Um programa fluxo de dados é um conjunto de instruções contidas na unidade de Ativação. Cada instrução contém a função a ser executada sobre os operandos e os endereços das instruções sucessoras. Assim sendo cada instrução é a descrição de um nó do programa fluxo de dados; onde os operandos estão contidos nos arcos de entrada e o resultado é colocado em seus arcos de saída (instruções sucessoras).

Para uma descrição mais detalhada da arquitetura começaremos definindo o formato das Instruções (figura 3.3) contidas na

Unidade de Ativação e o formato dos pacotes de dados (figura 3.6) gerados pela Unidade de Processamento.

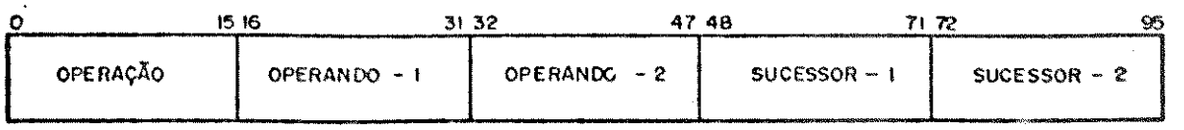


FIGURA 3.3 - FORMATO DE UMA PALAVRA DE INSTRUÇÃO FLUXO DE DADOS

Sendo:

Sucessor-1 e **Sucessor-2** os endereços das instruções sucessoras, que foram limitadas no máximo a duas, com o objetivo de encontrarmos facilidades na implementação da arquitetura (largura das vias de comunicação e da Unidade de Ativação).

Operando-1 e **Operando-2** respectivamente, os operandos da esquerda e da direita para instruções binárias, e sempre o operando da esquerda para as instruções unárias.

O campo de operação exemplificado na figura 3.4 é dado por:

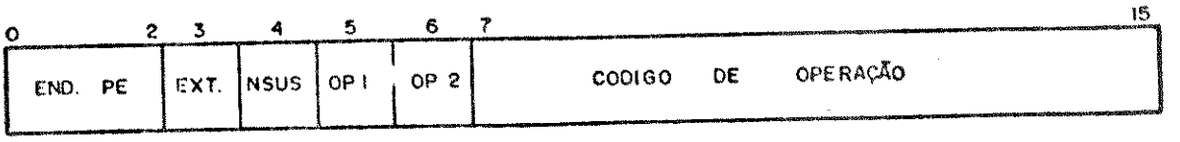


FIGURA 3.4 - CAMPO DE OPERAÇÃO DA PALAVRA DE INSTRUÇÃO

Sendo:

End. PE - três bits que selecionam o processador elementar conveniente para a execução da instrução contida no código de operação.

EXT - bit reservado para uma possível expansão do número de processadores elementares contidos na Unidade de Ativação.

NSUS - número de instruções sucessoras contidas na instrução.

nsus = 0 indica uma instrução sucessora.

nsus = 1 indica duas instruções sucessoras.

OP1 e OP2 - dois bits que indicam a existência dos operandos na instrução (um bit para cada operando).

0 indica inexistência do operando.

1 indica existência de operando.

Exemplo - o conteúdo "0" no campo OP1 e "1" no campo "OP2" indicam a inexistência do operando da esquerda e a existência do operando da direita.

COD.OPERAÇÃO - indica a operação a ser realizada sobre os operandos.

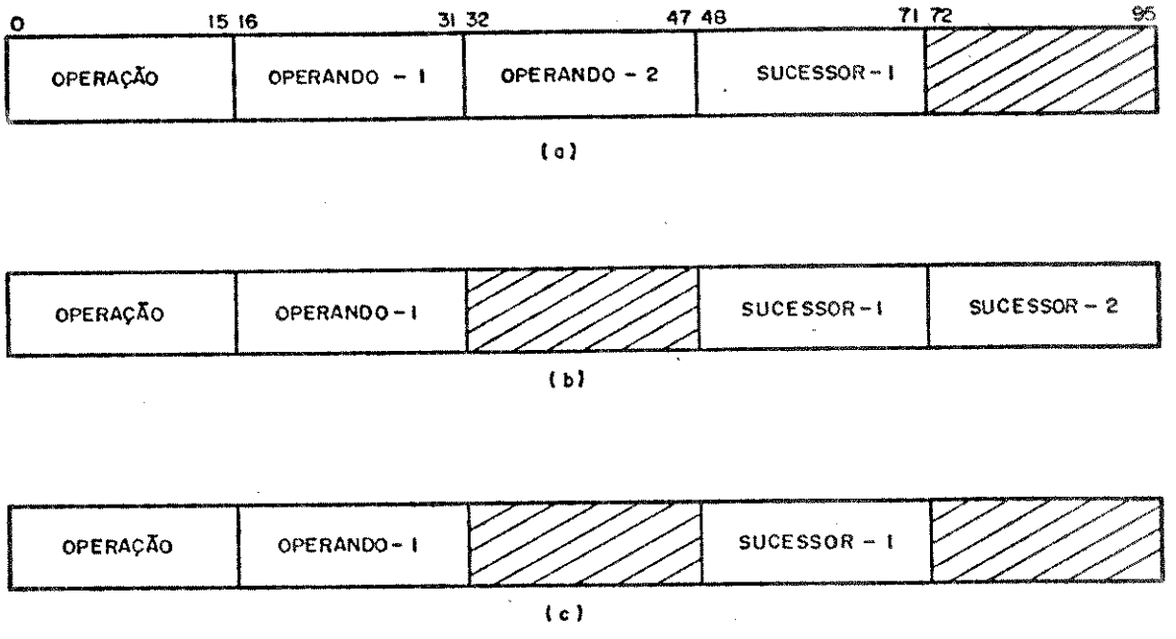


FIGURA 3.5 - VARIANTES DA PALAVRA DE INSTRUÇÃO

Pela interpretação dos bits NSUS, OP1 e OP2 podem existir as seguintes variantes no formato de instrução:

- instrução binária com duas sucessoras (figura 3.3).
- instrução binária com uma sucessora (figura 3.5a).
- instrução unária com duas sucessoras (figura 3.5b).
- instrução unária com uma sucessora (figura 3.5c).

O formato de um pacote de dados gerado pela Unidade de Processamento é mostrado na figura 3.6.

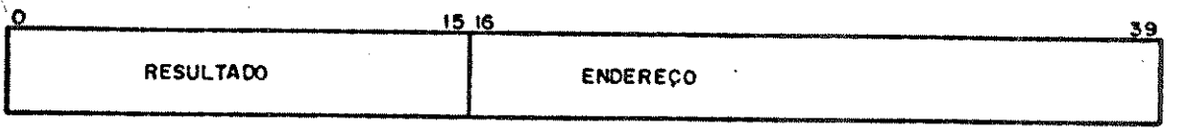


FIGURA 3.6 - FORMATO DE UM PACOTE DE DADO

Sendo:

RESULTADO o resultado da operação realizada pela instrução anterior.

ENDEREÇO- será uma cópia de sucessor1 ou sucessor2 de acordo com a interpretação do bit "nsus" da palavra de instrução. Se nsus = 1 serão gerados dois dados, um para cada instrução sucessora.

FIGURA 3.7 - FORMATO DO CAMPO DE ENDEREÇO DO PACOTE DE DADO

A figura 3.7 mostra o campo de endereço do pacote de dado em

maiores detalhes; observando que o campo de endereço do pacote de dados é idêntico ao formato de um sucessor na palavra de instrução.

Sendo:

END.INST - indica a que instrução pertence o resultado; e o endereço da instrução dentro do bloco (dentro da Unidade de Ativação).

BLOCO - três bits reservados para expansão do número de anéis básicos contidos na arquitetura.

POS - indica a posição do operando na instrução:

1 indica operando da direita;

0 indica operando da esquerda.

NOP - indica o número de operandos necessários para a instrução ser executada :

0 indica 0 ou 1 operando;

1 indica 2 operandos.

3.4 - Descrição e Implementação de Blocos Básicos (Funcionais)

A figura 3.8 mostra com maiores detalhes a organização do sistema de controle da central de comutação proposta. Nela já podemos identificar algumas atividades inerentes a cada Processador Elementar. Este compromisso entre atividades a serem desenvolvidas em cada um dos Processadores Elementares é uma característica desse sistema onde iremos identificar as instruções a serem executadas por um determinado Processador Elementar; o que facilita a tarefa de microprogramá-los. Esta característica diferencia este processador de um processador fluxo de dados de uso geral, onde as instruções são enviadas a

qualquer Processador Elementar livre (não processando).

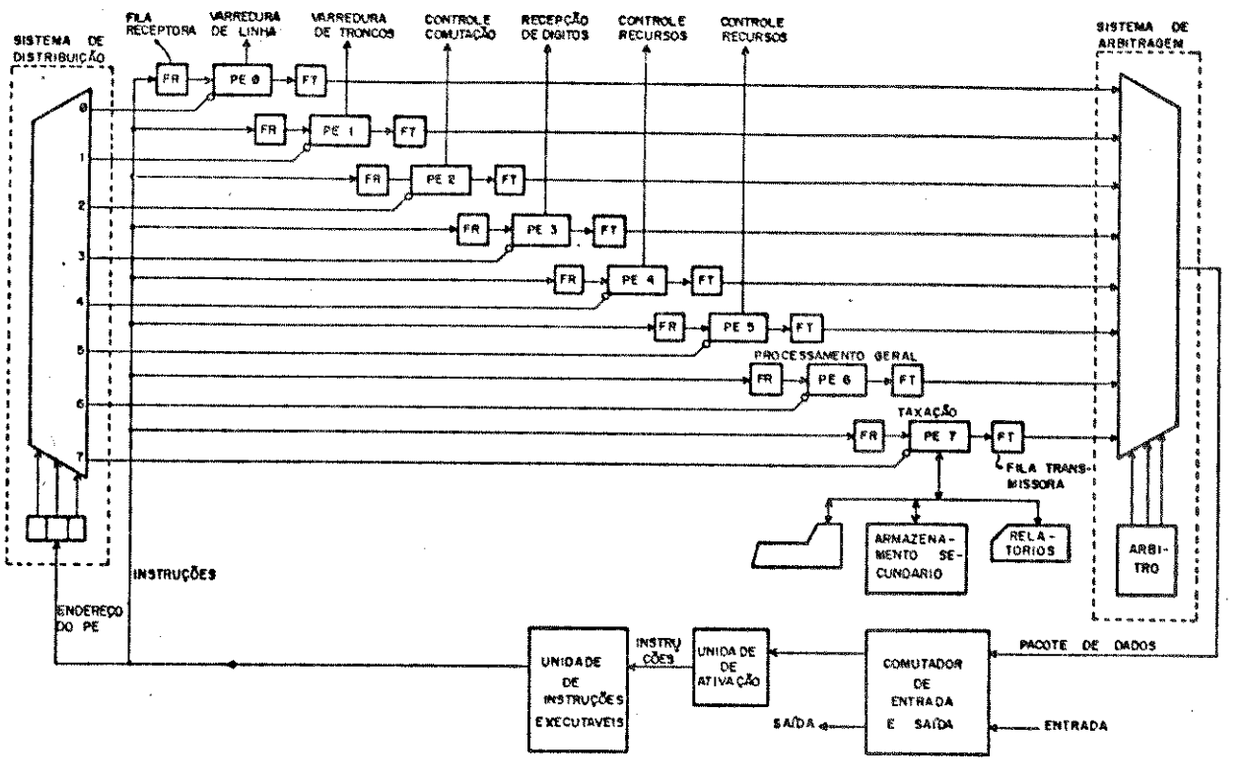


FIGURA 3.8 - ARQUITETURA DO CONTROLE DA CENTRAL DE COMUTAÇÃO

3.4.1 - Comutador de Entrada e Saída

Este bloco desempenha as funções de Entrada e Saída do sistema fluxo de dados.

Instruções provenientes da Entrada são sempre enviadas para o anel; instruções provenientes da unidade de processamento podem ter dois destinos:

- saída do sistema,
- permanecer no anel.

Este destino é decidido com a ajuda do campo denominado bloco da palavra de instrução (ver endereço do pacote de dados).

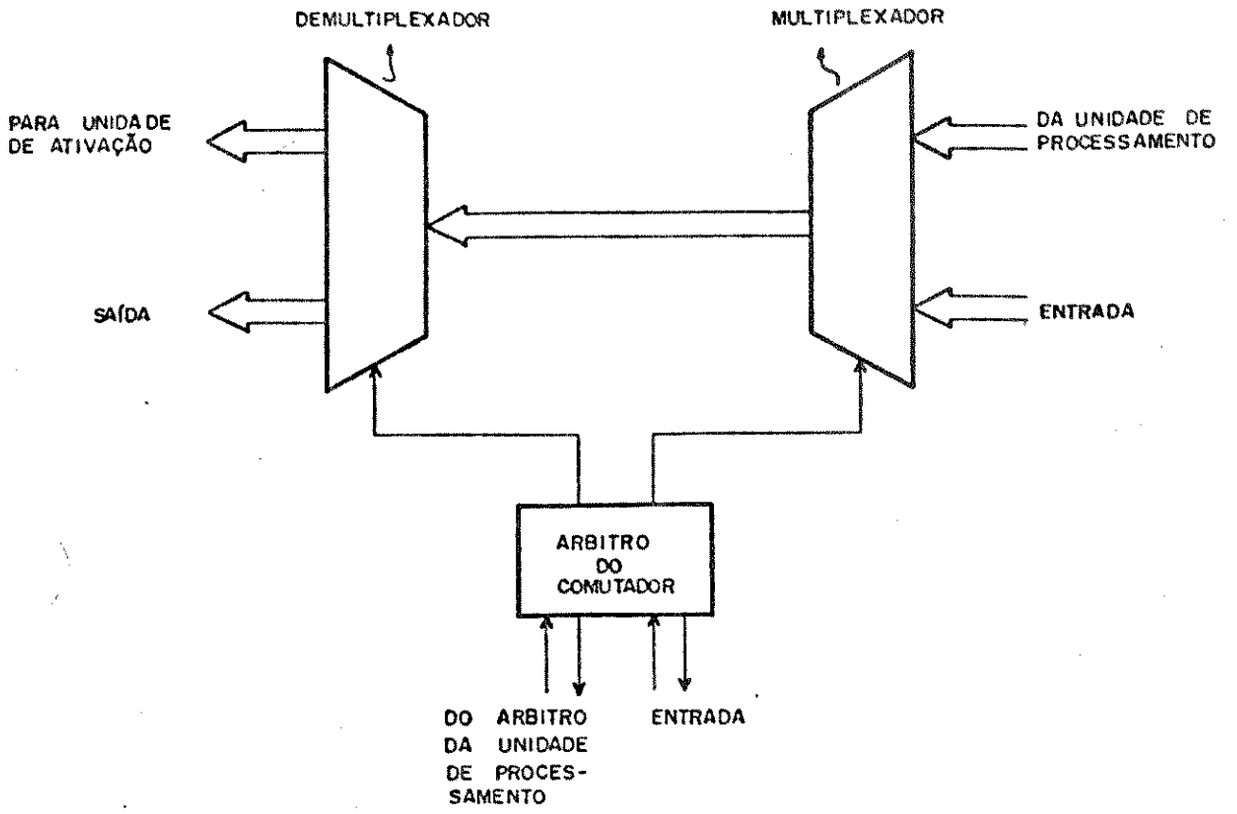


FIGURA 3.9 - COMUTADOR DE ENTRADA E SAIDA

req.UP	req.CES	rec.UP	rec.CES
0	0	0	0
0	1	0	1
1	0	1	0
1	1	0	1
		1	0

TABELA 3.1 - TABELA VERDADE DO ÁRBITRO DO COMUTADOR DE ENTRADA E SAÍDA

O comutador de entrada e saída mostrado na figura 3.9, deve arbitrar entre chegadas simultâneas de resultados em suas

entradas. Isto é feito no bloco denominado árbitro do comutador, cuja implementação pode ser realizada com o circuito que é mostrado na figura 3.10, com a ajuda da tabela 1 [PI82].

Desse modo a ocorrencia de requisições simultanêas na unidade de processamento e do comutador de entrada e saída, o árbitro irá gerar o reconhecimento (rec) apenas para um pedido.

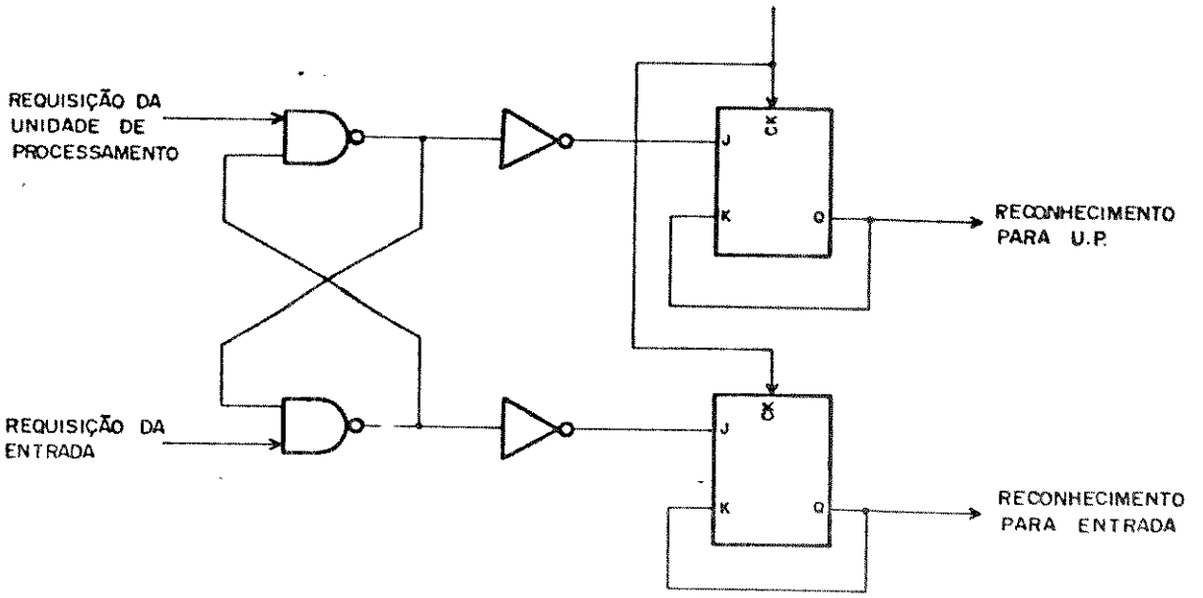


FIGURA 3.10 - ÁRBITRO DO COMUTADOR DE ENTRADA E SAÍDA

3.4.2 - Unidade de Ativação

A Unidade de ativação é o bloco mais importante e também crítico desse sistema fluxo de dados em anel. O desempenho deste bloco influenciará de maneira preponderante o desempenho do sistema de controle como um todo.

Esta unidade tem as funções de :

-armazenar os programas fluxo de dados que geram o controle da central de comutação.

-realizar o mecanismo de disparo para processamento de instruções executáveis (instruções que tenham todos os seus operandos disponíveis).

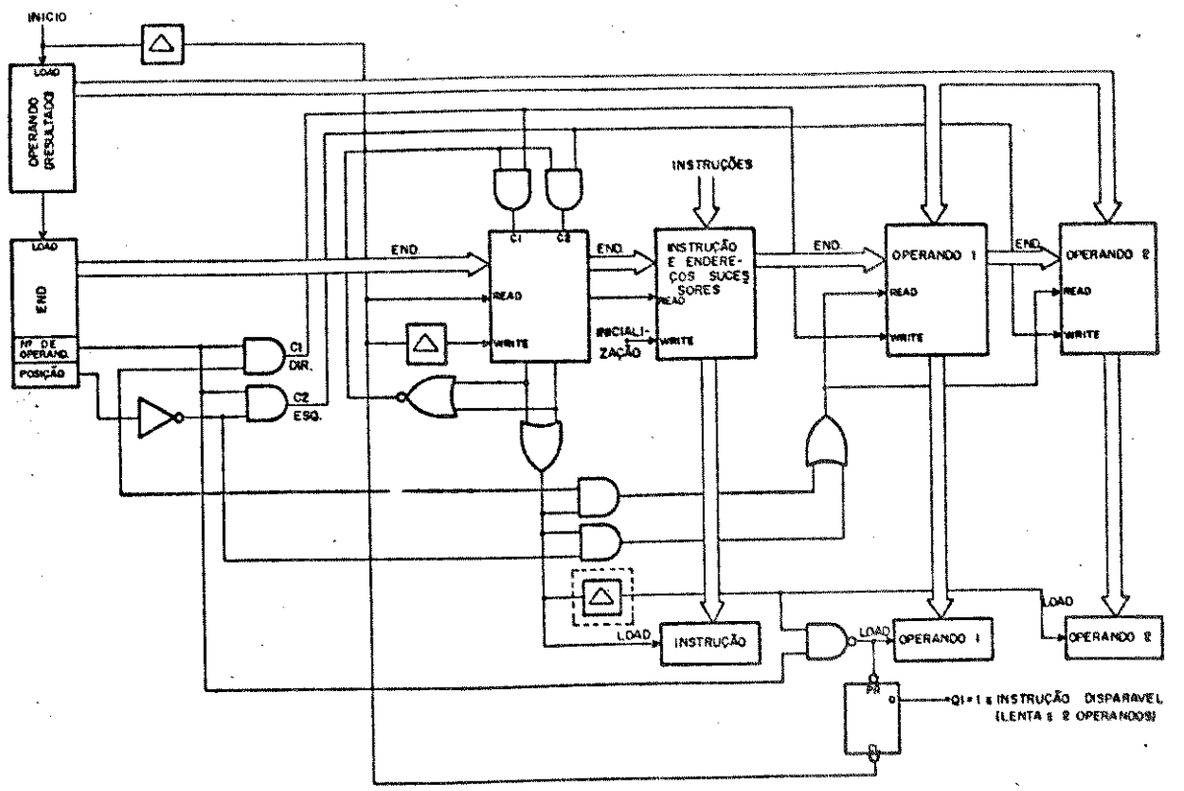


FIGURA 3.11 - UNIDADE DE ATIVAÇÃO

A figura 3.12 mostra a Unidade de Ativação em diagrama de blocos e a figura 3.11 mostra uma possível implementação desta unidade; seu desempenho será tanto melhor quanto menor o tempo de ciclo de leitura e escrita das memórias de acesso aleatório usados na implementação.

Este esquema de implementação da Unidade de Ativação usando vários bancos de memória com controles independentes,reproduzindo os campos de uma palavra de instrução da máquina a fluxo de dados facilitará sobremaneira a implementação do mecanismo de disparo de instruções executáveis, embora em algumas situações reproduza-

se desnecessáriamente os campos de operandos.

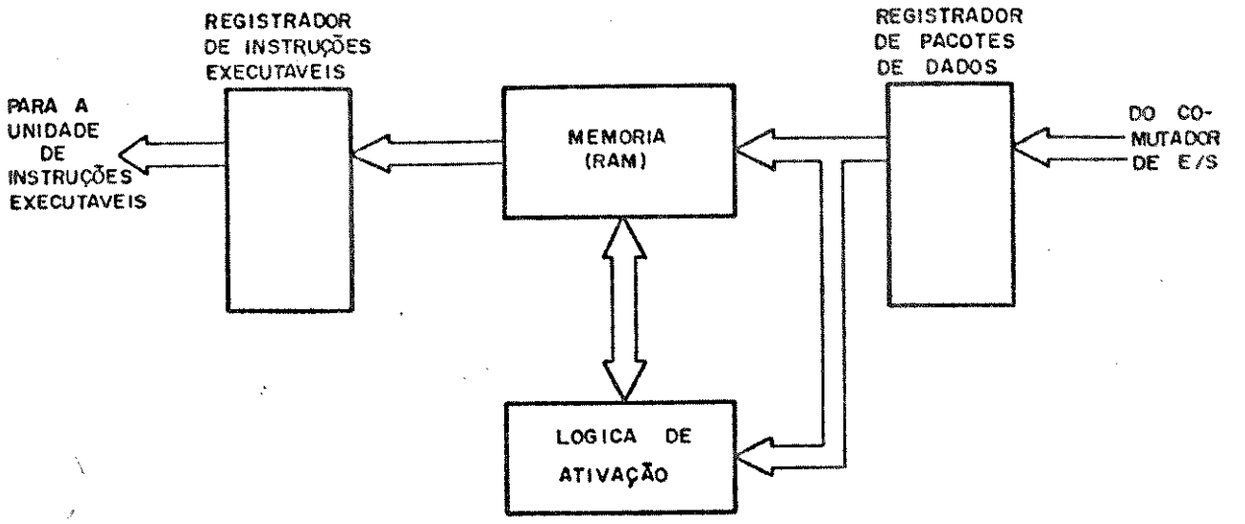


FIGURA 3.12 - DIAGRAMA DE BLOCOS DA UNIDADE DE ATIVAÇÃO

3.4.2.1 - O Mecanismo de Disparo de Instruções Executáveis

O algoritmo de disparo de instruções executado pela unidade de ativação, é o seguinte:

" INICIO

PASSO 1 : inicialização

A unidade de ativação recebe em seu registrador de pacotes de dados um resultado proveniente do comutador de entrada e saída.

PASSO 2 : verificação das condições de disparo

A lógica de ativação, através do bit 23 do campo de endereço do resultado (denominado "número de operandos"), verifica a quantidade de operandos necessários para o disparo da instrução.

- se for instrução unária

então:

Carrega o registrador de instrução executável com o conteúdo da posição apontada pelo campo de endereço do registrador de pacote de dados e ativa-se um "flag" de instrução executável

senão:

Verifica a disponibilidade de operandos na instrução, através do campo de memória denominado "C1 C2".

- se existir operando anterior

então:

Carrega registrador de instrução executável com o conteúdo de posição apontada pelo campo de endereço do registrador de pacote de dado. Ativa-se "flag" de instrução executável.

senão:

Carrega-se o campo de resultado do registrador de instrução executável no banco operando1 ou operando2 na posição apontada pelo campo de endereço do registrador de pacotes de dados (ver campo de endereço de um pacote de dados).

PASSO 3: finalização

Atualiza-se campo de memória "C1 C2".

Pede-se um novo pacote de dado para o comutador de entrada e saída.

se flag de instrução executável = 1

então:

envia-se instrução contida no registrador de instrução executável para a unidade de instruções executáveis.

FIM ".

Notamos no algoritmo que podemos ter três possíveis procedimentos na unidade de ativação que geram tempos de execução diferentes, e que são:

instrução unária - tempo de execução igual a 1 ciclo de leitura".

instrução binária não executável - tempo de execução igual a 1 ciclo de leitura + 1 ciclo de escrita.

instrução binária executável - tempo de execução igual a 1 ciclo de leitura.

Concluimos que as instruções binárias para serem disparadas gastam aproximadamente dois ciclos de leitura e um ciclo de escrita; enquanto que instruções unárias gastam apenas um ciclo de leitura.

3.4.3 - Unidade de Instruções Executáveis

Esta unidade desempenha uma atividade moderadora de tráfego entre unidade de ativação e unidade de processamento; ela armazenará instruções que foram transmitidas para um determinado processador elementar e encontraram o receptor "fifo" do PE selecionado no estado "cheio"; sendo necessário esperar a desocupação do buffer do processador elementar para que a instrução seja retransmitida; durante esta espera a instrução fica armazenada na unidade de instruções executáveis.

A figura 3.13 descreve em diagrama de blocos a Unidade de Instruções Executáveis.

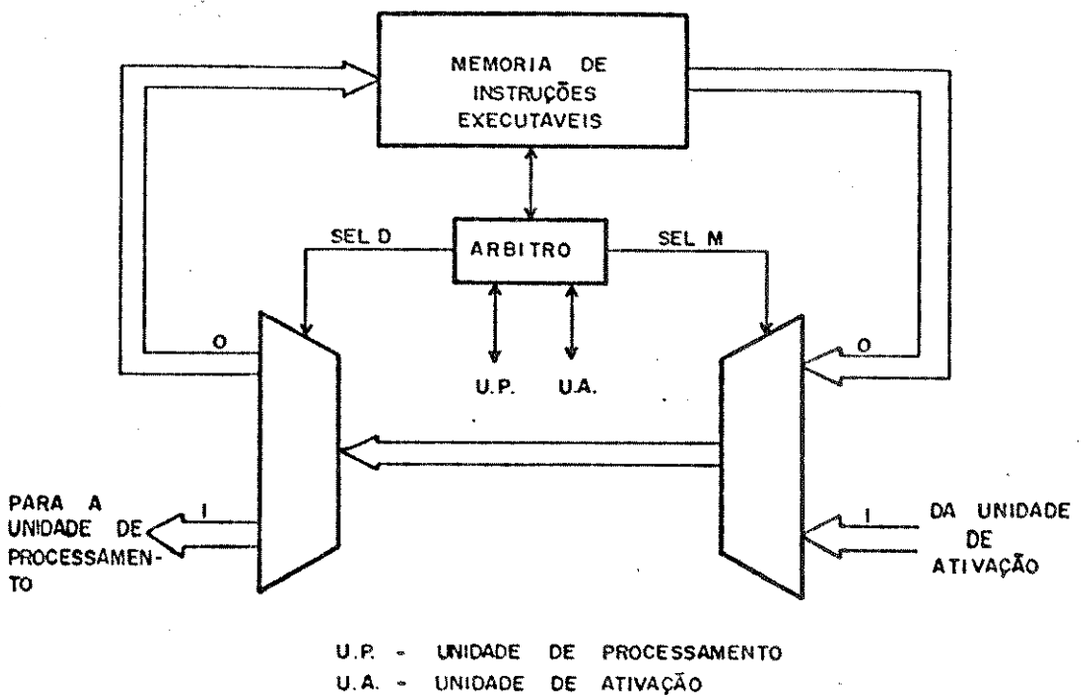


FIGURA 3.13 - UNIDADE DE INSTRUÇÕES EXECUTÁVEIS

Enquanto a Unidade de Instruções Executáveis contiver instruções, ela irá disputar com a unidade de ativação o acesso a unidade de processamento.

A figura 3.14 mostra em maiores detalhes o bloco memória de instruções executáveis.

O bloco RE na figura 3.14 é o registrador de endereço da memória que conterá as instruções executáveis, ele pode ser implementado como sendo um contador "crescente-decrescente" (conta para cima e para baixo), sendo que os controles incrementa contador (inc) e decrementa contador (dec) são efetuados respectivamente pelos comandos de leitura e escrita na memória.

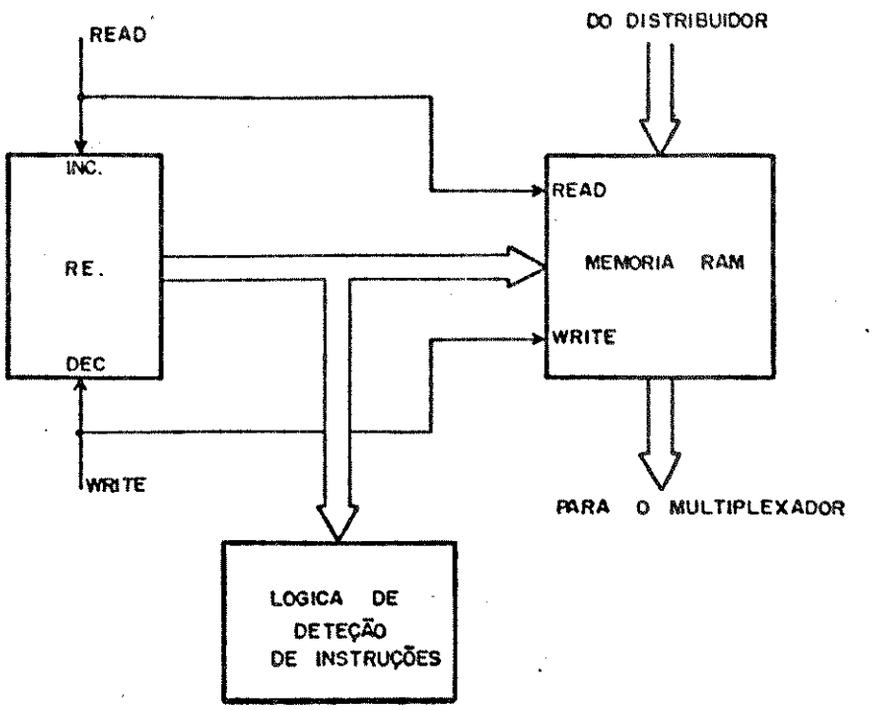


FIGURA 3.14 - MEMÓRIA DE INSTRUÇÕES EXECUTÁVEIS

A lógica de detecção de instruções detecta quando todos os bits do contador forem iguais a um (1), o que indica que a memória de instruções executáveis está cheia.

Quando isto acontecer, dependendo da causa que gerou o preenchimento de toda memória, poderemos tomar as seguintes providências cabíveis:

-Reformulação de programas armazenados na unidade de ativação quando for detectada a sobrecarga de algum processador elementar, devido a má distribuição das instruções feitas por estes programas para os processadores elementares.

-Reparo de algum processador elementar quando for detectada alguma falha de circuito (hardware) nestes.

-Desativação de alguma chamada quando acontecer

possíveis congestionamentos de tráfego na Central.

Da análise acima vemos que a Unidade de Instruções Executáveis é de grande importância na fase de implementação de um protótipo (gerando informações para melhoria de software) e depois , durante a vida útil da central (gerando informações para manutenção desta).

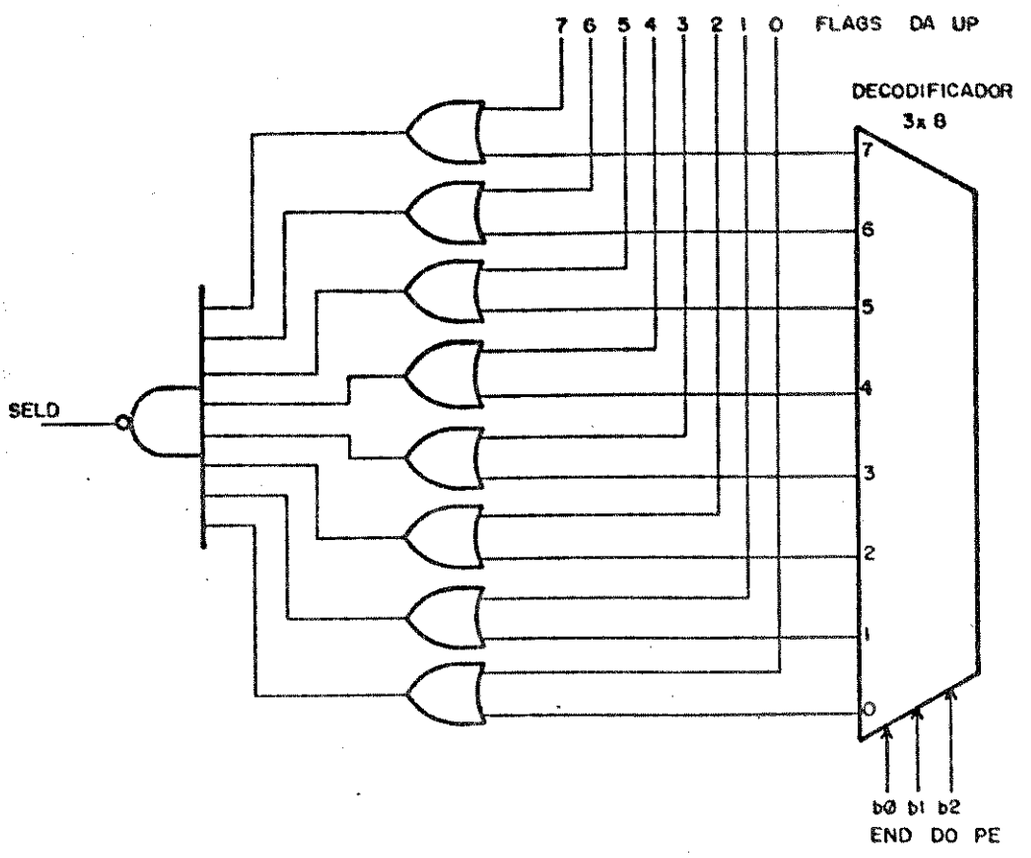


FIGURA 3.15 - ÁRBITRO DA UNIDADE DE INSTRUÇOES EXECUTÁVEIS

A figura 3.15 mostra em detalhes a lógica de seleção da Unidade de Instruções Executáveis. Ela gera a seleção do demultiplexador (seld será igual a zero se a fila receptora estiver cheia).

A seleção do multiplexador (selm) é gerada pela requisição do envio do pacote de dado da unidade de ativação. Assim a

memória de instruções executáveis será sempre selecionada quando a unidade de ativação não estiver requisitando o envio de pacotes de dados.

3.4.4 - Unidade de Processamento

É na Unidade de Processamento que serão executados os programas fluxo de dados, sendo que cada nó pertencente ao programa terá um microprograma que o realiza em um determinado Processador Elementar. Como consequência direta do campo "end.PE" e do bit "ext" da palavra de instrução (figura 3.4) temos que o número máximo de Processadores Elementares que a Unidade de Processamento poderá conter será 16 (dezesseis).

Podemos ver na figura 3.17 uma possível implementação da Unidade de Processamento contendo 8 (oito) Processadores Elementares, cada um deles ligados a uma fila receptora e uma fila transmissora que armazenará temporariamente as instruções destinadas ao seu processador elementar. A fila transmissora conterá também temporariamente pacotes de dados gerados pelo seu processador elementar, a serem transmitidas para as instruções sucessoras.

O bloco denominado Sistema de Distribuição é formado por um Registrador de Processador Elementar e um Selecionador. O Registrador de Processador Elementar é um registrador de 4 bits que conterá os campos "end.PE" e "ext" da palavra de instrução e sua função é selecionar o Processador Elementar endereçado pela instrução a ser executada. O selecionador será um demultiplexador 4x16, e terá a função de gerar a seleção do processador especificado pelo Registrador de Processador Elementar.

elementar. Chegadas de requisições simultanêas num mesmo árbitro dual são decididos por ele próprio. Chegadas de requisições de árbitros duais diferentes no primeiro nível de árbitros duais será decidida nos níveis de árbitros subsequentes (2 e 3). O bloco árbitro dual da figura 3.17 é mostrado em detalhes na figura 3.10.

Esta segunda proposta de implementação, embora exija um circuito mais complicado, diminuirá o tempo gasto por um processador elementar para enviar seus pacotes de dados para a unidade de ativação, pois será atendida a primeira requisição a chegar ao sistema de arbitragem.

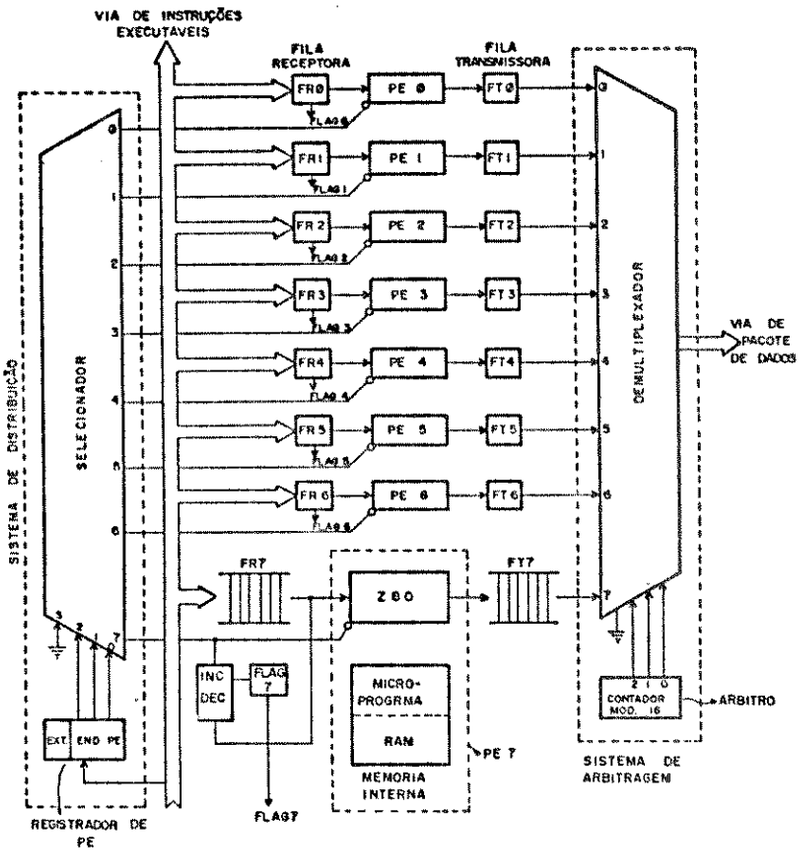


FIGURA 3.17 - UNIDADE DE PROCESSAMENTO

3.4.5 - Comunicação entre Blocos Básicos do Sistema em Anel

A comunicação síncrona ou assíncrona num sistema envolve um compromisso temporal. Sistemas síncronos são mais apropriados para situações onde as tarefas envolvidas gastam aproximadamente o mesmo tempo para serem completadas e portanto não necessitando de um sistema de arbitragem complicado para resolver conflitos entre tarefas distintas. Quando os tempos de execução são muito diferentes entre si, existe a necessidade de uma comunicação assíncrona. Portanto em nossa proposição de arquitetura a comunicação interna aos blocos básicos é síncrona (cada bloco terá um relógio interno próprio) e a comunicação entre os blocos será assíncrona e unidirecional [GKW85].

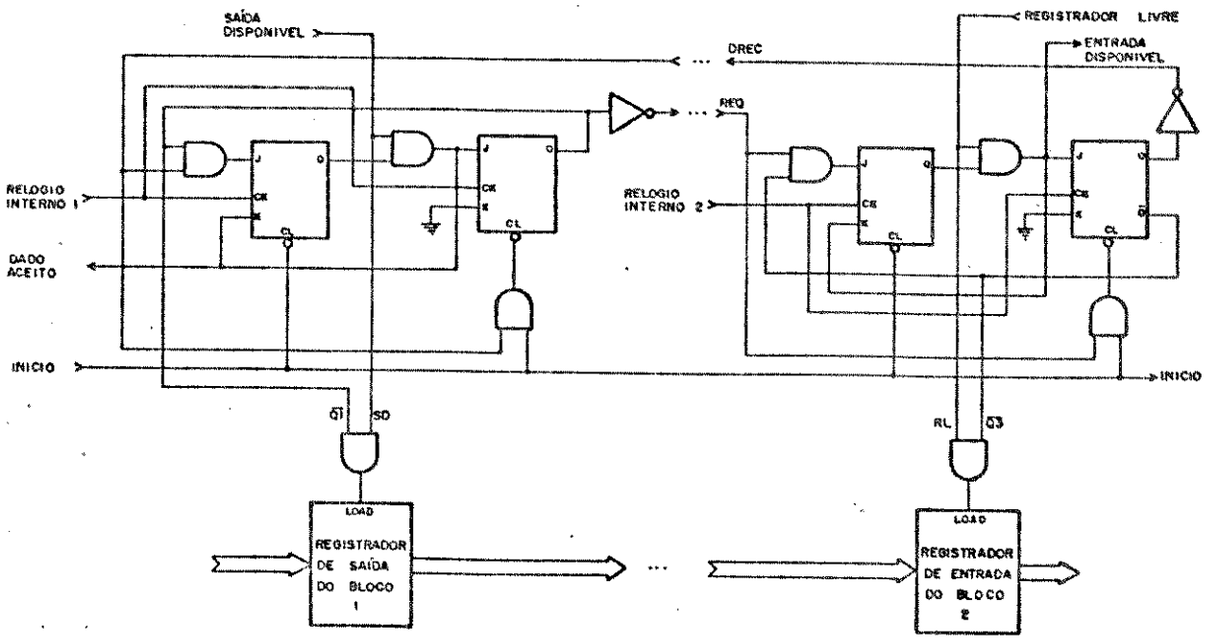


FIGURA 3.18 - COMUNICAÇÃO ENTRE DOIS BLOCOS BÁSICOS

A figura 3.18 exemplifica a comunicação entre os blocos. Ela é feita básicamente pelo envio de uma requisição para enviar dado (req) e o retorno de uma resposta de dado recebido (drec). Os

dados são transferidos do registrador de saída de um bloco básico para o registrador de entrada do bloco básico subsequente.

Como pode ser visto na figura 3.18 o circuito que realiza a transferência de um pacote de dado ou de instrução entre duas unidades subsequentes pode ser implementado usando-se 4 flip-flops JK; denominado FF0,FF1,FF2 e FF3; e uma pequena lógica de controle. A tabela 3.2 ilustra o funcionamento do circuito da figura 3.18. Nela podemos ver que o acontecimento de um "drec" gera a realização das seguintes tarefas, atuando nas entradas J0 e CL1:

-carrega informação no registrador de entrada do respectivo bloco

-inibe o "req" que gerou colocando o circuito no estado inicial (pronto para realizar nova transferência).

RL	SD	J0	J1/K0	Q0	Q1	CL1	J2	K2	J3/Q2	$\overline{Q3}$	CL3	LD1	LD2	IN
1	1	X	X	0	0	0	X	X	0	1	0	1	1	0
1	1	1	0	1	0	1	0	0	0	1	0	1	1	1
1	1	1	1	0	1	1	0	0	0	1	1	0	1	1
1	1	0	0	0	1	1	1	0	1	1	1	0	1	1
1	1	0	0	0	1	1	1	1	0	0	1	0	1	1
1	1	0	0	0	1	0	0	0	0	0	1	0	0	1
1	1	0	0	0	0	0	0	0	0	0	0	0	0	1
1	1	0	0	0	0	0	0	0	0	1	0	1	0	1
1	1	1	0	1	0	1	0	0	0	1	0	1	1	1

TABELA 3.2 -

O acontecimento de um "req" atuando nas entradas J2 e CL3 respectivamente, ativa os biestáveis FF2 e FF3 para realizarem o tratamento de uma nova requisição de envio de informação.

Com o auxílio da tabela 3.2 e da figura 3.18 obtemos as seguintes relações:

DA = K0 = J1 = SD * Q0

J0 = Q1' * Q3

ED = K2 = J3 = Q2' * RL

J2 = Q1 * Q3'

Q3 = drec'

Q1 = req

load1 = SD * Q1'

load2 = Q3' * R1

CL3' = inicio' * Q1

CL1' = inicio' * Q3'

DA ==> Dado Aceito

ED ==> Entrada Disponível

SD ==> Saída Disponível

RL ==> Registrador Livre

3.5 - Arquitetura Expandida

A arquitetura fluxo de dados proposta aqui possui uma limitação de paralelismo que ela pode explorar. Esta limitação é ditada pela quantidade de blocos contidos no anel, pela velocidade de operação da Unidade de Ativação e pela quantidade de Processadores Elementares contidos na Unidade de Processamento.

Uma possibilidade de aumentar significativamente o paralelismo dessa arquitetura é fazer um arranjo de vários anéis idênticos ao anterior trabalhando em paralelo, como mostra a figura 3.19.

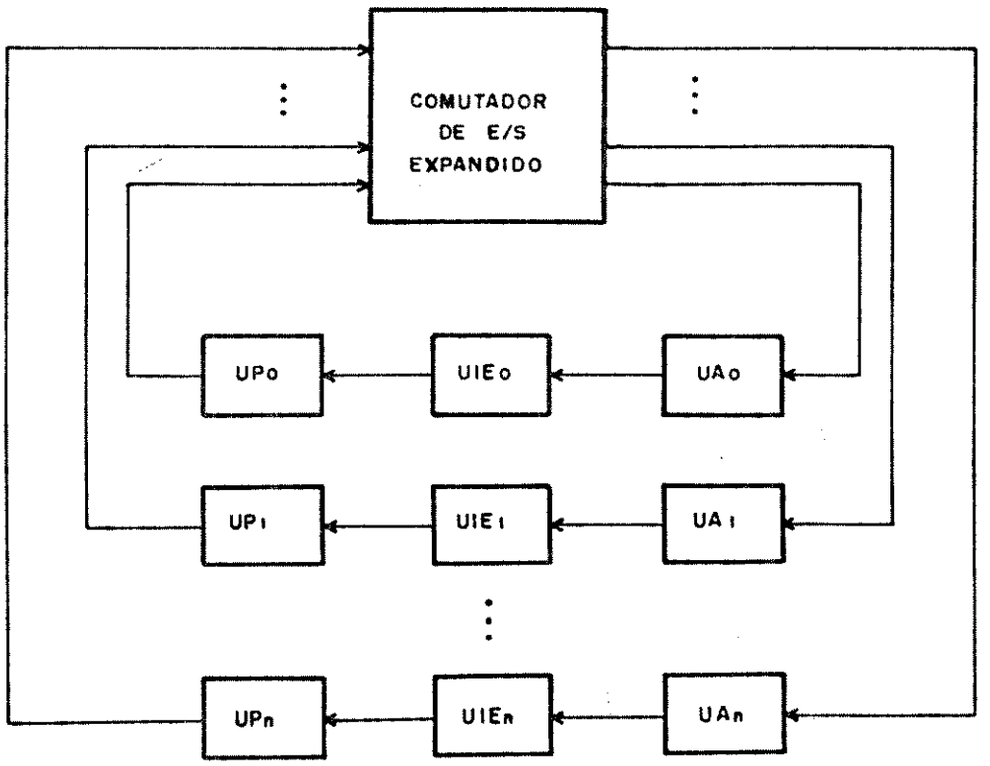


FIGURA 3.19 - EXPANSÃO DA ARQUITETURA EM ANEL

Para a realização dessa expansão torna-se necessária uma mudança na implementação do Comutador de Entrada e Saída. A natureza unidirecional do anel torna factível a implementação mostrada na figura 3.20.

A quantidade de anéis que poderão ser colocados em paralelo na arquitetura expandida está limitado em 8 (oito) pelo campo "bloco" do pacote de dado (figura 3.7).

Para realizar a comutação de 8 (oito) possíveis entradas para 8 (oito) quaisquer saídas usaremos uma combinação de multiplexadores e demultiplexadores como mostra a figura 3.20. Esta implementação torna o atraso no comutador de entrada e saída no mínimo 3 (três) vezes maior que a implementação do mesmo bloco na arquitetura anterior, pois possui três níveis de

UNICAMP
BIBLIOTECA CENTRAL

multiplexadores e demultiplexadores mais o controle desses multiplexadores.

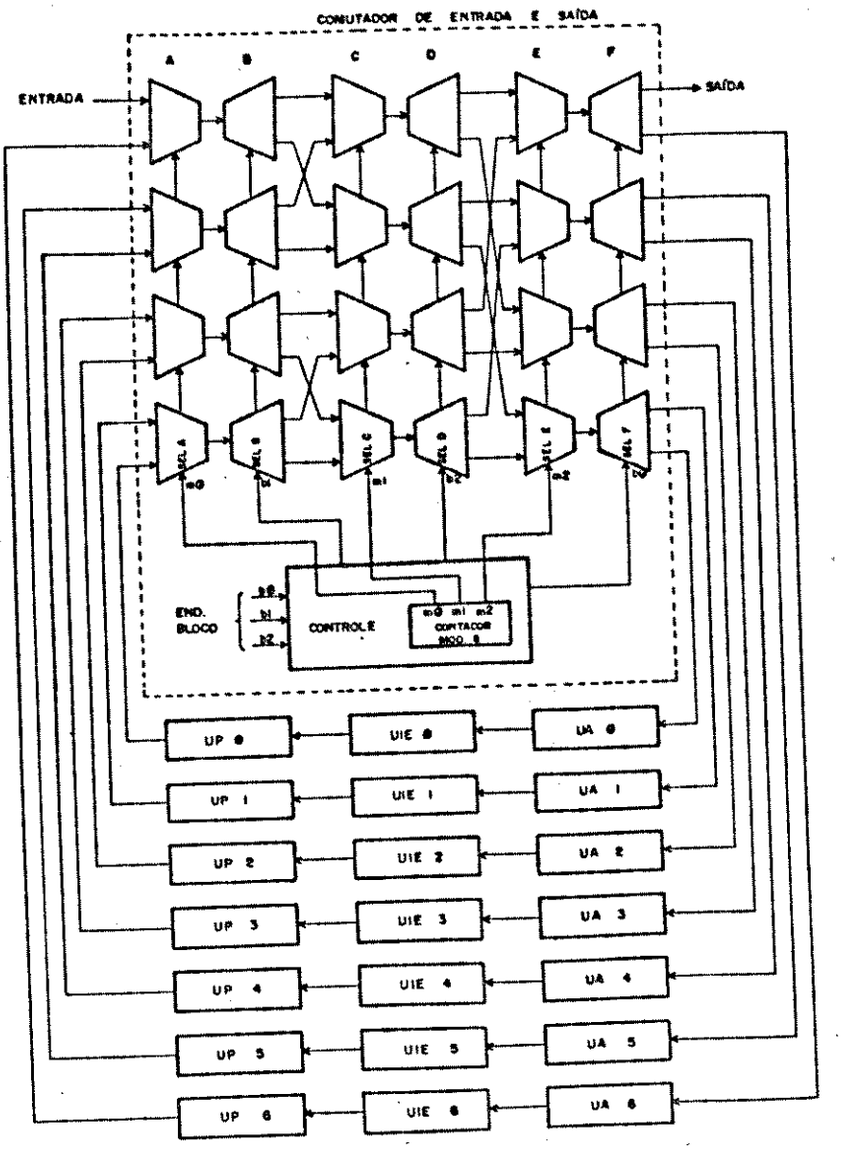


FIGURA 3.20 - ARQUITETURA EXPANDIDA

O controle do comutador de Entrada e Saída pode ser implementado como mostra a figura 3.21. Onde m0,m1 e m2 é o conteúdo de um contador módulo oito (8) que tem a função de varrer sequencialmente as entradas do comutador de entrada e

saida, e b0,b1 e b2 é o conteúdo do campo bloco dentro do campo endereço de uma palavra de instrução fluxo de dados (figura 3.7).

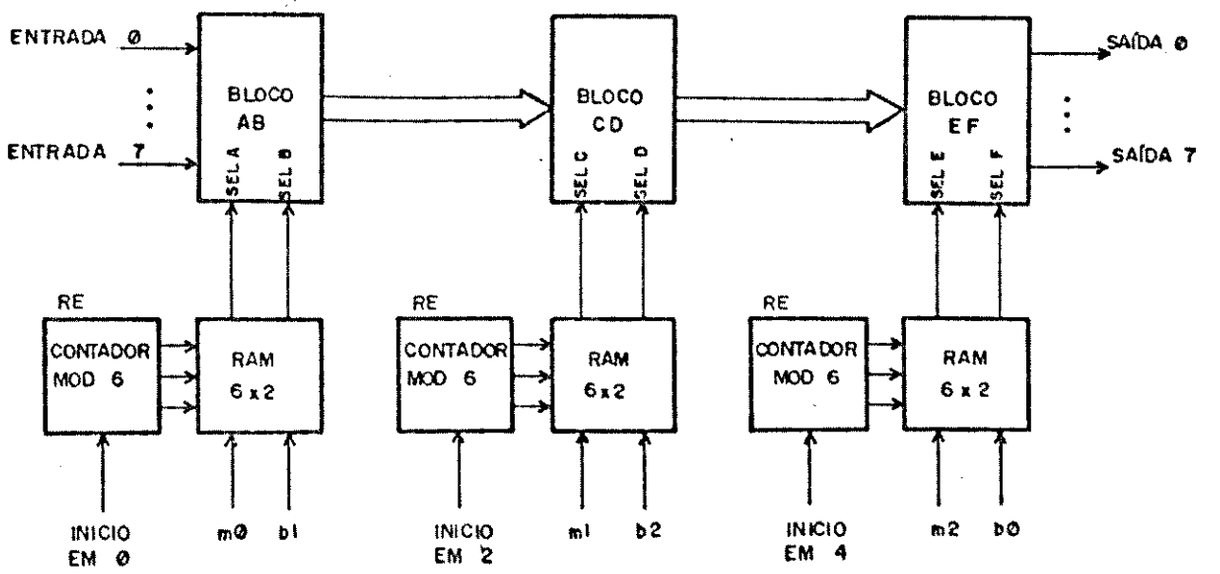


FIGURA 3.21 - CONTROLE DO COMUTADOR DE ENTRADA E SAÍDA

Mas, como pode ser visto na figura 3.20 e está sintetizado no apendice 1; a lógica de controle pode ser simplesmente o conteúdo dos bits m2,m1 e m0 e b2,b1 e b0. Esta implementação permite a comutação de um único pacote de dado por vez, pois temos apenas um pacote de dado presente no circuito comutador durante um intervalo de tempo. Com a inclusão das memórias de acesso aleatório da figura 3.21, para armazenar as informações da trajetória de cada pacote de dado presente no Circuito Comutador durante um intervalo de tempo, poderemos ter até 3 mensagens sendo comutadas simultâneamente (sobrepostas nos níveis de comutação [A,B],[C,D] e [E,F]). Entretanto isto implicaria tambem na inclusão de "buffers" entre os estágios de comutação que juntamente com os tempos de leitura e escrita nas memórias de acesso aleatório introduziria atrasos consideráveis ao circuito

comutador, além de complicar sobremaneira a implementeção do bloco de controle do comutador de entrada e saída. Diante destas características optamos pela simplicidade e rapidez do circuito de controle da figura 3.20 e cuja síntese é mostrada no apendice-1. Entretanto com um número maior de anéis presentes na arquitetura expandida o circuito de controle aqui escolhido continua sendo mais simples, mas poderá não ser o mais rápido.

Uma outra alternativa de implememtar-se o comutador de Entrada e Saída, como mostra a figura 3.22, é fazer o uso de dois blocos de comutação operando em paralelo, onde cada bloco é constituido de um mutiplexador, um demultiplexador e uma memória de acesso aleatório (RAM) que pode armazenar até 8 pacotes de dados. A comutação em paralelo realizada pelos dois blocos tem como objetivo aumentar significativamente a própria velocidade de comutação do circuito comutador. O controle do circuito é realizado por um contador módulo oito (8) que tem a função de varrer sequencialmente as entradas de pacotes de dados e escreve-los também sequencialmente na RAM de um bloco. E concomitantemente o contador realizará a leitura sequencial da RAM do outro bloco, direcionando o pacote de dado para a saída correta (especificada por b0,b1 e b2) do demultiplexador daquele bloco.

Dessa maneira os dois blocos comutadores trabalharão em paralelo, alternando as atividades de leitura e escrita de pacotes da dados nas RAMs dos blocos. Portanto, enquanto um bloco está realizando a aquisição de oito pacotes de dados dos anéis de origem o outro estará enviando também oito pacotes da dados para os anéis de destino.

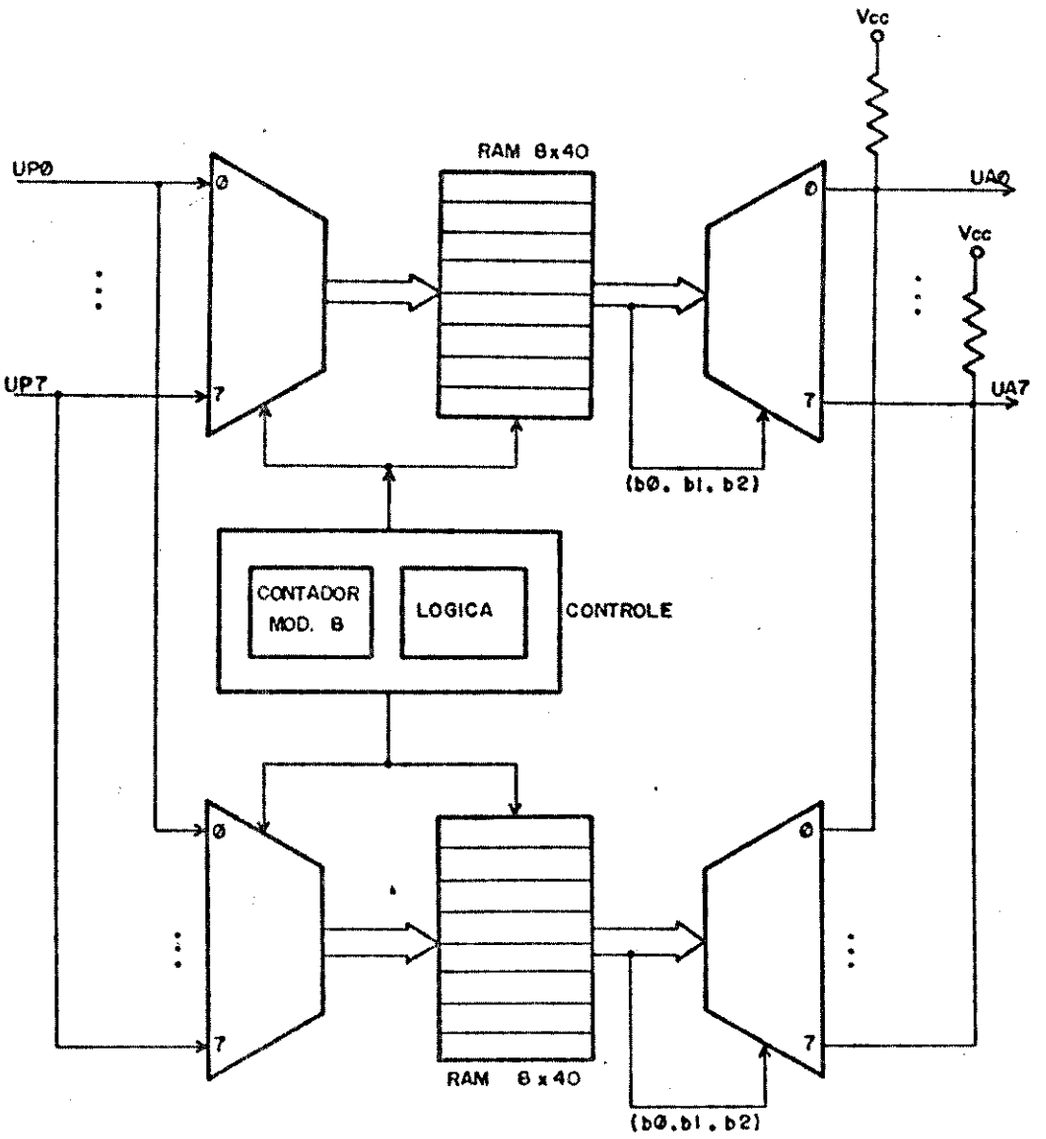


FIGURA 3.22 - COMUTADOR DE ENTRADA E SAÍDA COM DOIS BLOCOS COMUTADORES EM PARALELO

3.6 - Tolerância a Falhas

A natureza da arquitetura mostrada na figura 3.8, com unidades de trabalho sobrepostos (pipeline), torna a tolerância a falhas de muitos componentes impraticável. Pois a tolerância a falhas somente é possível em componentes idênticos que trabalham em paralelo com outro. Portanto a falha de algum processador elementar da Unidade de Processamento pode ser manuseada

extraindo-se o PE defeituoso, operando-se assim mais lentamente. Todas as outras unidades no anel são entretanto, críticas com respeito a falhas.

Já na arquitetura expandida mostrada na figura 3.20, fica claro que a central de comutação pode isolar falhas nos anéis fluxo de dados, isolando-se o anel. Entretanto falhas que ocorram no Comutador de Entrada e Saída agora são críticas. Este é um ponto fraco do sistema expandido que pode ser fortalecido usando-se técnicas de implementação tolerantes a falhas (colocando-se dois comutadores de entrada e saída em paralelo,por exemplo).

3.7 - Conclusão 2

Neste capítulo foi desenvonvida uma arquitetura à fluxo de dados constituída por quatro blocos básicos organizados em forma de anel (figura 3.1).

Esta organização permite velocidade de execução, explorando seu alto grau de paralelismo, além de fornecer facilidade de expansão e grande confiabilidade, conectando-se vários anéis em paralelo através de um bloco comutador.

Alguns blocos básicos foram projetados a níveis de portas lógicas. O bloco básico comutador de entrada e saída foi projetado visando possibilitar facilidades para expansão da arquitetura, obtendo-se paralelamente maior confiabilidade através da conexão em paralelo de vários anéis na arquitetura expandida. Ao bloco Unidade de Ativação foi dedicada uma maior atenção devido ser este bloco aquele que influencia de maneira preponderante o desempenho da arquitetura. Ele é responsável

pelo mecanismo de disparo de instruções.

O projeto desses blocos a níveis de portas lógicas, além de visar uma futura implementação também é importante para avaliar-se os tempos de atraso desses blocos. Estes tempos de atraso serão importantes na avaliação de desempenho a ser realizada no capítulo 4.

CAPÍTULO 4

ANÁLISE DE DESEMPENHO

4.1 - Introdução

Neste capítulo é feita uma avaliação da eficiência da arquitetura proposta no capítulo 3. O método de análise utilizado é o mesmo encontrado em [GWG78].

O método baseia-se num teorema que afirma que para processadores de um sistema perfeito (veja item 4.2.4), a eficiência de execução é maior que 50% [BMc86].

Será também estudada a quantidade de processadores necessários, no caso em que a arquitetura proposta seja utilizada como central telefônica com um determinado tráfego telefônico (medido em Erlangs) a ser oferecido. Para essas avaliações é necessário considerar-se as características de execução do programa, e a tecnologia usada para implementar o sistema.

A seguir serão definidos termos que serão utilizados neste capítulo.

4.2 - Definições

São descritas inicialmente algumas características necessárias aos programas fluxo de dados, para que estes utilizem eficientemente o paralelismo oferecido pelo sistema.

4.2.1 - Comprimento de um Programa (Cj)

O Comprimento de um Programa (Cj) é definido como sendo um número inteiro de passos necessários para executá-lo. Isto é, Cj representa o número de passos necessários para a execução de um programa se "j" processadores são disponíveis para realizar a computação de até "j" instruções elegíveis (instruções com todos

seus operandos disponíveis) em cada passo de execução.

4.2.2 - Sistema de Execução Perfeita

Um sistema de execução é dito de execução perfeita se:

i) todos processadores iniciam e terminam suas instruções simultaneamente.

ii) cada processador gasta uma unidade de tempo para realizar uma instrução.

iii) no final de qualquer passo, se "e" instruções são elegíveis, então todas "e" intruções serão executadas imediatamente se "e<j" e exatamente "j" instruções se "e>j".

Pela definição de Cj nós podemos concluir que os valores máximos e mínimos de Cj são respectivamente C1 e Coo(infinito).

Onde:

C1 é o número de passos requeridos para executar um programa com apenas um processador elementar; isto é equivalente a termos a execução totalmente serial do programa.

Coo é o número de passos requeridos para que sejam executadas todas as instruções elegíveis a cada passo. Isto pode requerer um número arbitrariamente grande de processadores elementares.

4.2.3 - Paralelismo Médio de um Programa Fluxo de Dados (Cm)

O paralelismo médio (Cm) é definido como sendo:

$$Cm = \frac{C1}{Coo} \qquad (4-1)$$

4.2.4 - Eficiência de um Sistema com "j" Processadores (Ej)

A eficiência de um sistema com "j" processadores (Ej) é dada por:

$$Ej = \frac{Coo}{Cj} \qquad (4\text{-}2)$$

Verifica-se que a eficiência Ej é sempre menor ou igual a 1(um), pois por definição Cj>=Coo.

Tomando como exemplo o grafo da figura 4.1, obeserva-se que:

C1 = 6

C3 = C4 = Coo = 2

Cm = 3

Entretanto, C2 pode assumir os valores 3 e 4, de acordo com a escolha das instruções na ordem:

{ (1,2); (5,3); (4,6) } ou { (1,2); (4,5); (3,_); (6,_)}.

A eficiência no caso de 2 (dois) processadores poderá ser:

$$E2 = \frac{Coo}{C2} = \frac{2}{3} \qquad \text{ou} \qquad E2 = \frac{1}{2},$$

dependendo da ordem de escolha das instruções elegíveis.

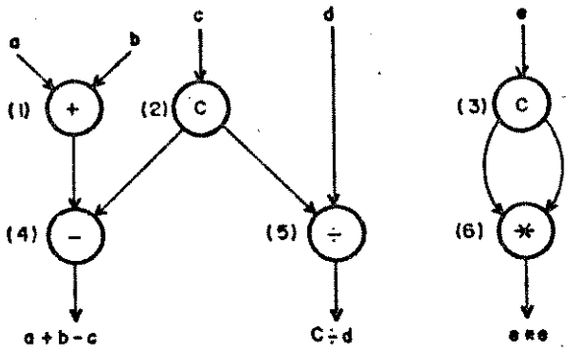


FIGURA 4.1 - GRAFO ILUSTRANDO UMA INDETERMINAÇÃO Cj

Desse modo pode-se dizer que a escolha aleatória de "j" instruções elegíveis a cada passo pode afetar o comprimento Cj, e também a eficiência (Ej) do sistema.

Define-se m' como sendo o menor inteiro maior ou igual a C1/Coo, isto significa que Cm' é o número de passos para executar um programa com m' processadores.

Então, considerando um sistema com m' processadores , a eficiência do sistema será dada por:

$$Em' = \frac{C1}{Cm * Cm'} = \frac{C1}{\frac{C1}{Coo} * Cm'} = \frac{Coo}{Cm'} \qquad (4\text{-}3)$$

A seguir será enunciado um teorema sobre a eficiência de um sistema fluxo de dados, encontrado em [GWG78] e que será utilizado neste capítulo (ver prova no apêndice 2).

Teorema 4.1-Para um programa fluxo de dados executado num sistema de execução perfeita com m' processadores temos que:

$$\frac{1}{2} < Em' <= 1 \qquad (4\text{-}4)$$

Isto significa que num sistema de execução perfeita com m' processadores tem-se, em média, mais que 50% dos processadores elementares ocupados.

4.3 - Paralelismo na Arquitetura em Anel

A arquitetura à fluxo de dados elaborada neste trabalho é

baseada num anel circular de blocos que desempenham atividades em paralelo de maneira sobreposta (pipeline). Cada bloco possui um tempo de atraso para processar os pacotes de dados ou pacotes de instruções. A avaliação de eficiência será feita através da aproximação da arquitetura proposta ao sistema perfeito mencionado no item 4.2.

O comportamento do sistema em anel pode aproximar-se do sistema de execução perfeita se tivermos o seguinte:

i) As diferentes espécies de instruções obedecerem a uma distribuição uniforme,

ii) um tempo médio de execução de instrução (Tm).

Com um tempo médio de execução bem definido pode-se determinar o tempo de passo (Tp) da arquitetura em anel; somando-se o tempo total de atraso no anel (Ta) ao tempo médio de execução (Tm) de uma instrução obtendo-se:

$$Tp = Ta + Tm \qquad (4-5)$$

Com uma distribuição uniforme das instruções pode-se determinar o tempo mínimo entre instruções no anel (Tmin) e assim estimar o grau de paralelismo no anel (K) .

Define-se o grau de paralelismo K como sendo o menor inteiro maior ou igual a Tp/Tmin.

O valor K representa o paralelismo da Unidade de Processamento e das atividades sobrepostas no anel.

Deve-se observar que quando Cm<K a ocupação mínima de 50% do anel não pode ser garantida, entretanto pode-se afirmar que para valores maiores de Cm/K utiliza-se mais eficientemente a arquitetura. Obeserva-se também que executando mais que um

programa simultaneamente aumenta-se o paralelismo médio de programa (Cm).

Aproximando-se o sistema em anel a um sistema de execução perfeita (ver apendice 3), pode-se avaliar a eficiência geral da arquitetura em anel determinando-se Tmin, Ta, Tm e K.

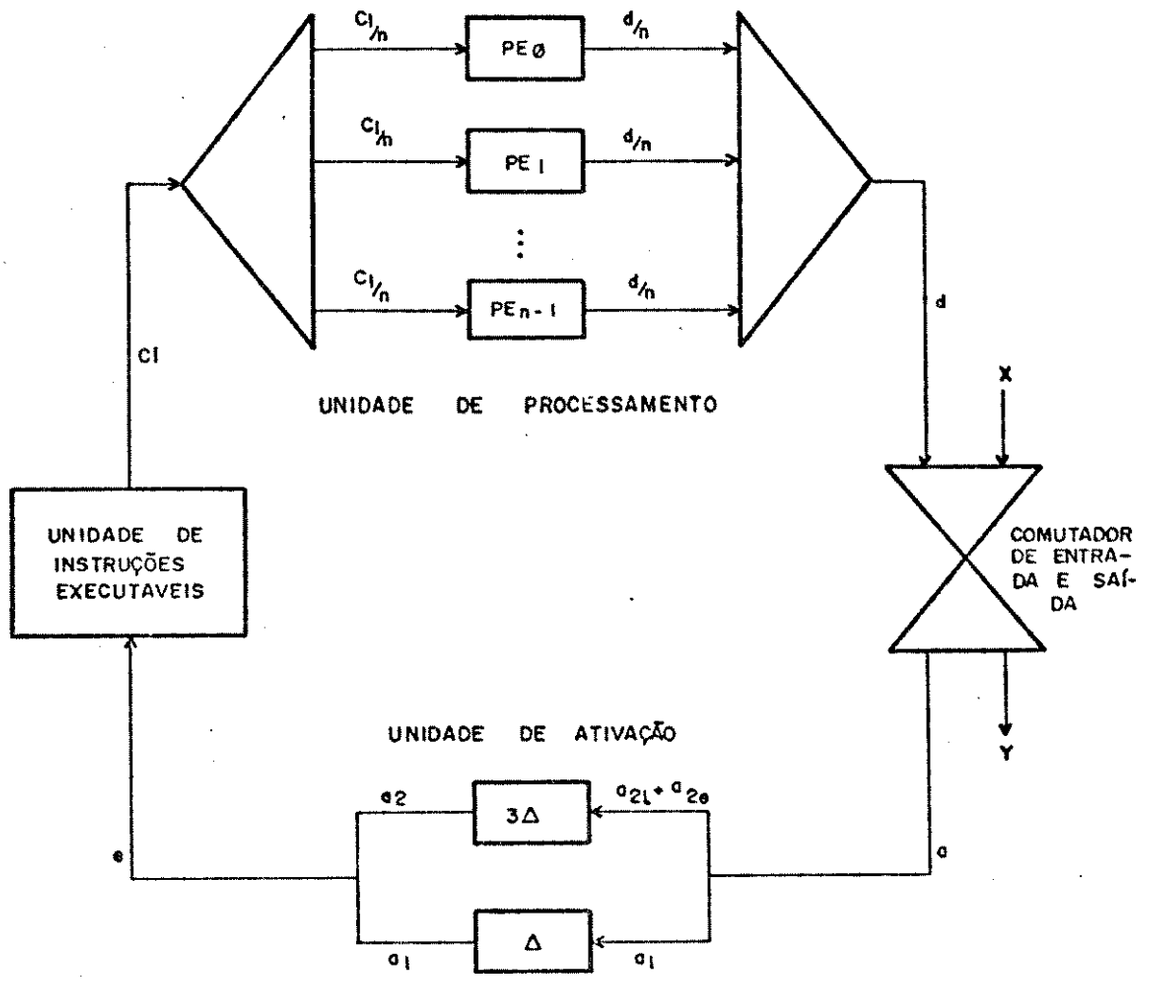


FIGURA 4.2 - NÚMERO DE PACOTES ATRAVÉS DO ANEL DURANTE A EXECUÇÃO DE UM PROGRAMA

Para determinar-se Tmin, considera-se a execução de um programa completo que consome "x" pacotes de dados de entrada, executa **Cl** instruções, e produz "y" resultados de saída. Vamos determinar as relações entre o número total de pacotes passando

através de cada parte do sistema durante o tempo de execução do programa (Te). Na figura 4.2 é mostrada a arquitetura em anel com símbolos representando os números totais de pacotes que caminham através dos enlaces de dados entre os módulos.

Onde:

a - número total de pacotes de dados necessários para executar um programa.

x - número de pacotes de dados que entram no sistema.

y - número de pacotes de dados que saem do sistema.

a1 - quantidade de instruções unárias.

a2 - quantidade de instruções binárias.

a21 - quantidade de operandos que tornam instruções binárias elegíveis.

a2e - quantidade de operandos que ficam armazenados na Unidade de Ativação (primeiro operando disponível de uma instrução binária) .

d - quantidade de resultados (pacotes de dados) gerados pela Unidade de Processamento.

C1 - quantidade de instruções contidas num programa fluxo de dados.

Pela ação dos componentes que implementam os blocos do sistema, deduz-se as seguintes relações:

$$C1 = a1 + a2, \qquad (4-6)$$

$$a2 = a21 \qquad (4-7)$$

$$a = d + x - y = a1 + a21 + a2e \qquad (4-8)$$

Para programas bem formados (well conformed programs), isto

é, programas que nos seus términos não deixam nenhum pacote de dado armazenado na Unidade de Ativação, conclue-se que:

$$a2 = a2e = a21 \qquad (4\text{-}9)$$

Portanto:

$$C1 = a1 + a2 = a1 + a2e \qquad (4\text{-}10)$$

Considerando-se que:

i) o número de pacotes de dados consumidos por cada instrução, podendo ser 1 ou 2 e,

ii) o número de pacotes de dados produzidos por cada instrução executada podendo ser 0, 1 ou 2.

Podemos definir:

P2e - a proporção de instruções requerendo dois pacotes de dados (instruções binárias).

P2s - a proporção de instruções que geram dois pacotes de dados (geram duas saídas).

P0s - a proporção de instruções que não geram pacotes de dados.

P1e - a proporção de instruções que requerem um pacote de dado (instruções unárias), sendo:

$$P1e = 1 - P2e \qquad (4\text{-}11)$$

P1s - a proporção de istruções que geram um pacote de dados de saída , sendo:

$$P1s = 1 - P2s - P0s \qquad (4\text{-}12)$$

Utilizando-se essas definições nas expressões anteriores temos:

$$a1 = C1 * (1 - P2e) = C1 * P1e \qquad (4-13)$$

$$a2 = C1 * P2e \qquad (4-14)$$

$$d = C1 * (2 * P2s + P1s) \qquad (4-15)$$

Substituindo o valor de P1s (expressão 4-12) na expressão 4-15 tem-se:

$$d = C1 * (P2s + P0s) \qquad (4-16)$$

Combinando-se as equações 4-8, 4-13 e 4-14 obtêm-se:

$$C1 = \frac{a}{1 + P2e} \qquad (4-17)$$

O tempo médio de atraso entre instruções é dado por:

$$\frac{Te}{C1} = (1 + P2e) * \frac{Te}{a} \qquad (4-18)$$

Onde a é o número total de pacotes de dados necessários para executar um programa e Te é o tempo total de execução de um programa. O valor mínimo de atraso entre instruções (Tmin) é o valor mínimo de Te/C1.

Vamos calcular os atrasos em cada bloco do anel na arquitetura proposta.

A Unidade de ativação é o bloco que necessita de maior tempo para o processamento dos pacotes de dados. A máxima taxa de atividade é igual a $3 * a2 + a1$ ciclos de acesso à memória.

Portanto, o tempo mínimo entre os pacotes de dados no anel deve ser:

$$\frac{Te}{3 * a2 + a1} \qquad (4\text{-}19)$$

Seja Tb, o tempo de atraso em um bloco do anel.

Em cada bloco, o Tb deve ser menor ou igual ao tempo mínimo assim:

$$Tb <= \frac{Te}{3 * a2 + a1} \qquad (4\text{-}20)$$

Substituindo os valores de a2 e a1 (expressões 4-13 e 4-14) na expressão 4-20 tem-se:

$$Tb <= \frac{Te}{C1 * (1 + 2 * P2e)} \Longrightarrow \frac{Te}{C1} >= Tb * (1 + 2 * P2e) \qquad (4\text{-}21)$$

O valor mínimo de Te/C1 que é Tmin será:

$$Tmin = \left[\frac{Te}{C1} \right]_{min} = Tb * (1 + P2e) \qquad (4\text{-}22)$$

4.4 - Tempo de Atraso dos Blocos

Serão agora estabelecidos os tempos de ciclos dos blocos pertencentes ao anel observando que nenhum deles pode ser menor que Tb.

4.4.1 - Tempo de Ciclo do Comutador de Entrada e Saída (tc)

Para obter-se tc, considera-se o fato de que a atividade de

entrada e saída no anel é pequena em relação a C1 (x<<C1 e y<<C1) e desta forma considera-se, também, que x≅y, assim tem-se:

$$Tb <= tc <= \frac{Te}{d} \cong \frac{Te}{a} = \frac{Te}{C1 * (1 + P2e)} \qquad (4\text{-}23)$$

como:

$$\frac{Te}{C1} = Tb * (1 + 2 * P2e) \qquad (4\text{-}24)$$

obtem-se:

$$Tb <= tc <= Tb * \frac{1 + 2 * P2e}{1 + P2e} \qquad (4\text{-}25)$$

4.4.2 - Tempo de Ciclo da Unidade de Processamento (tp)

Para estimarmos o tempo de atraso da Unidade de Processamento, devemos levar em conta o tempo de passo (Tp) do anel.

O tempo de passo do anel pode variar de programa para programa. Seja @ o número de microciclos (a unidade básica de tempo do processador) requeridos para realizar uma função. O valor de @ pode variar de programa para programa, consequentemente acarretando a variação no tempo de passo.

Desse modo, podemos escrever:

$$@ * Tp <= tp <= \frac{\frac{Te}{C1}}{n} = n * \frac{Te}{C1} = n * Tb * (1 + 2 * P2e) \qquad (4\text{-}26)$$

Onde n é o número de processadores elementares na Unidade de Processamento.

Da expressão 4-26 podemos concluir que:

$$n \geq \frac{Tp}{Tb} * \frac{\partial}{1 + 2 * p2e} \qquad (4\text{-}27)$$

Para a execução de uma instrução, o valor de Tp pode ser, no mínimo, igual a Tb, portanto:

$$n \geq \frac{\partial}{1 + 2 * P2e} \qquad (4\text{-}28)$$

4.4.3 -Tempo de Ciclo da Unidade de Instruções Executáveis (ti)

Como a Unidade de Instruções Executáveis realiza apenas uma função moderadora de tráfego,considera-se pequena a probabilidade de armazenamento de pacotes de instruções na memória desta unidade. Assim, faz-se o atraso médio nesta unidade igual ao atraso na unidade de ativação, ou seja:

$$Tb \leq ti \leq Tb \Longrightarrow ti=Tb \qquad (4\text{-}29)$$

4.4.4 - Tempo de Ciclo na Unidade de Ativação (ta)

Pela definição de Tb temos que o tempo de ciclo na Unidade de Ativação (ta) é o próprio Tb, portanto tem-se:

$$ta = Tb \qquad (4\text{-}30)$$

Considerando-se que todo atraso nos recursos da arquitetura deve ser maior que Tb; com a ajuda da figura 4.3; monta-se os atrasos em cada bloco do anel. Ressaltando-se que se objetiva estabelecer o tempo de atraso no anel (Ta) e o tempo de passo na arquitetura (Tp) , com a finalidade de obter-se o paralelismo (K) contido no anel.

Para tanto, com a ajuda da figura 4.3 faz-se:

$$\ell c = tac + tdc \geq tc \qquad (4\text{-}31)$$

onde:

tac - tempo do árbitro do comutador de E/S.

tdc - tempo do distribuidor do comutador de E/S.

$$\ell p = tdp + tap + tp = 2 * Tb + tp \qquad (4\text{-}32)$$

onde:

tdp - tempo do distribuidor da unidade de processamento.

tap - tempo do árbitro da unidade de processamento.

$$\ell i \geq ti = Tb \qquad (4\text{-}33)$$

$$\ell a \geq ta = Tb \qquad (4\text{-}34)$$

Assim tem-se os valores de Ta e Tb como sendo:

$$Ta = \ell c + \ell a + \ell i \geq tc + Tb + Tb = tc + 2 * Tb \qquad (4\text{-}35)$$

$$Tp = Ta + \ell p \geq tc + 4 * Tb + tp \qquad (4\text{-}36)$$

desta forma escreve-se o paralelismo K no anel como sendo:

$$K = \frac{Ta + \ell p}{Tmin} = \frac{Tp}{Tmin} = \frac{\ell c + \ell a + \ell i + \ell p}{Tmin} = \frac{tc + 4 * Tb + tp}{Tb * (1 + P2e)} \qquad (4\text{-}37$$

Para valores mínimos de tc e tp obtem-se:

$$Kmin \geq \frac{Tb + 4 * Tb + \alpha * Tb}{Tb(1 + P2e)} = \frac{Tb * (5 + \alpha)}{Tb * (1 + P2e)} = \frac{5 + \alpha}{1 + P2e} \qquad (4\text{-}38)$$

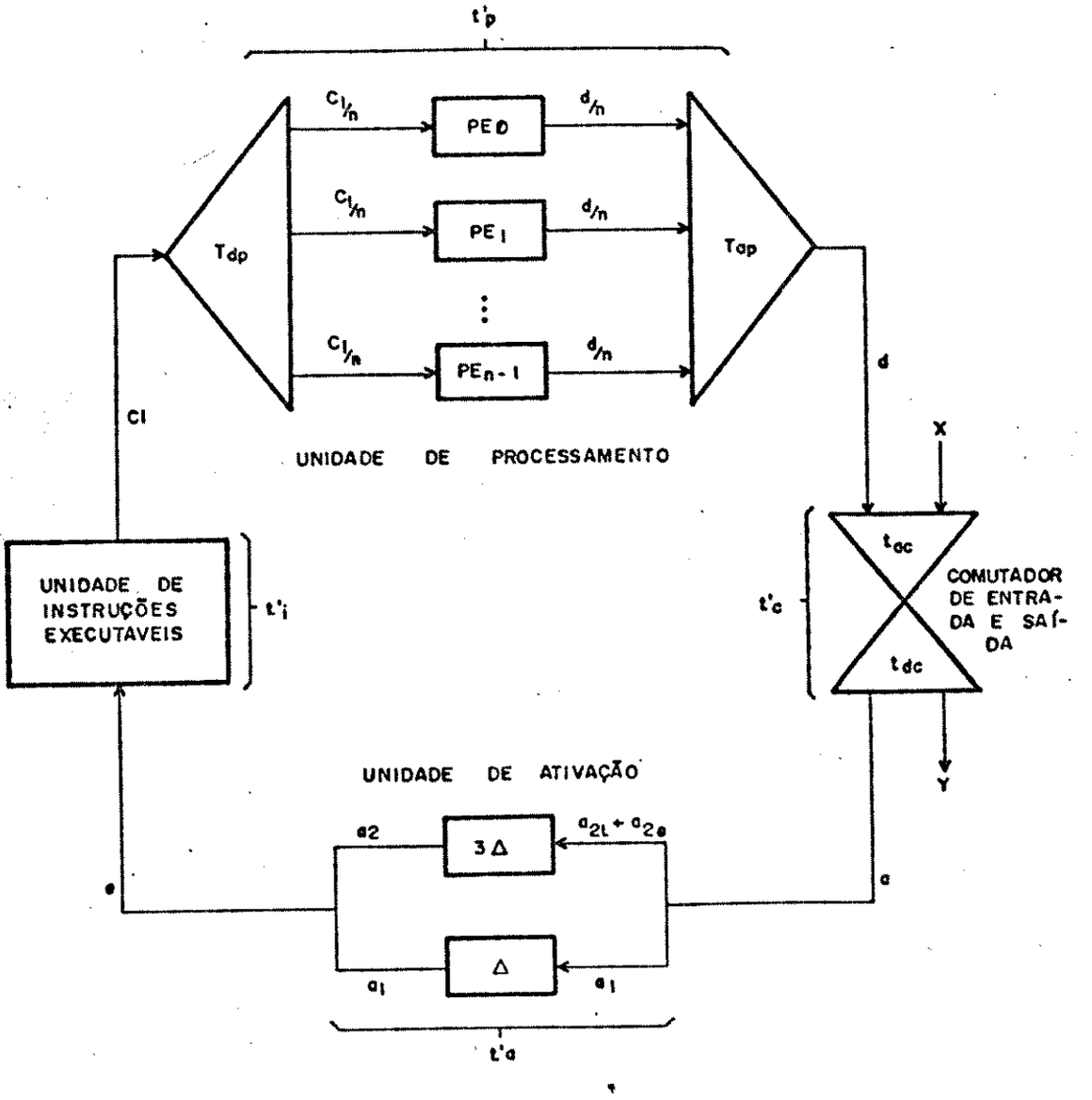


FIGURA 4.3 - ATRASOS NOS BLOCOS DO ANEL

Para valores máximos de tc e tp obtem-se:

$$Kmax \geq \frac{1}{1 + P2e} + \frac{4}{1 + 2 * P2e} + n \qquad (4\text{-}39)$$

Para obter-se o paralelismo esperado, foram tabulados os valores mínimos inteiros de n ,Kmin e Kmáx nas tabelas 4.1, 4.2, e 4.3.

P2e \ @	5	10	15	20	25	30	35	40
0	5	10	15	20	25	30	35	40
1/4	4	7	10	14	17	20	24	27
1/2	3	5	8	10	13	15	18	20
3/4	2	4	6	8	10	12	14	16
1	2	4	5	7	9	10	12	14

TABELA 4.1 - Valores mínimos de n >= $\frac{@}{1 + 2 * P2e}$

P2e \ @	5	10	15	20	25	30	35	40
0	10	15	20	25	30	35	40	45
1/4	8	12	16	20	24	28	32	36
1/2	7	10	14	17	20	24	27	30
3/4	6	9	12	15	18	20	23	26
1	5	8	10	13	15	18	20	23

TABELA 4.2 - Valores mínimos de Kmin >= $\frac{5 + @}{1 + P2e}$

P2e \ @	4	8	12	16	20	24	28
0	9	13	17	21	25	29	33
1/4	8	12	16	20	24	28	32
1/2	7	11	15	19	23	27	31
3/4	7	11	15	19	23	27	31
1	6	10	14	18	22	26	30

TABELA 4.3 - Valores de Kmax $\geq \frac{1}{1 + P2e} + \frac{4}{1 + 2 * P2e} + n$

Com uma rápida análise da tabela 4.1 obtem-se para programas com @ médio = 20 (número de microciclos para realizar uma instrução) e P2e = 3/4 (porcentagem de instruções binárias) necessita-se de no mínimo oito processadores elememtares na Unidade de Processamento. Agora na tabela 4.3 verifica-se que com oito processadores e P2e = 3/4, necessita-se de um paralelismo Kmáx = 11 no anel. Significando que com 8 processadores elementares P2e = 3/4 e programas com @ = 20, necessita-se de 3 níveis de paralelismo "pipeline", ou seja, de mais três blocos trabalhando em "pipeline" na arquitetura além da Unidade de Processamento, para manter-se a eficiência da arquitetura.

Finalmente define-se a razão média de execução de instruções (Ø) como sendo:

$$Ø = \frac{Ci}{Tj} \qquad (4\text{-}40)$$

onde Tj é o tempo de execução usando J processadores e

$$Tj = J * Cj * Tmin \qquad (4-41)$$

assim:

$$\emptyset = \frac{C1}{J * Cj * Tmin} = Ej * \frac{1}{Tmin} \qquad (4-42)$$

sendo 1/Tmin a razão de execução máxima e para:

$$Pm >= J \quad ====> \quad Ej > \frac{1}{2}$$

Como Tmin=Tb*(1+2*P2e), para Tb=150 ns (nanosegundos) e P2e=0,5 tem-se a razão de execução em torno de 3,3 MIPS (Milhões de Intruções Por Segundo) e a razão média de execução maior que 1,66 MIPS, para programas com paralelismo médio (Pm) maior ou igual ao paralelismo do anel.

Para programas com P2e mínimo (P2e = 0) tem-se a razão de execução máxima em torno de 6,6 MIPS e a razão de execução média maior que 3,3 MIPS.

Para programas com P2e máximo (P2e = 1) tem-se a razão de execução máxima em torno de 2,2 MIPS e a razão de execução média maior que 1,1 MIPS.

Conclue-se que a razão de execução da arquitetura proposta estará entre 1,1 e 6,6 MIPS, para programas com Tm >= K (paralelismo do anel).

Fez-se uma análise similar para uma arquitetura expandida contendo 8 anéis, Tb=150 ns e P2e=0,5 obtendo-se uma taxa de execução máxima de instruções perto de 27 MIPS e a taxa de

execução média será sempre maior que 3 MIPS para programas com paralelismo maior que 64.

De posse desses resultados pode-se dizer que o custo para aumentarmos a eficiência das arquiteturas fluxo de dados proposta nesta trabalho está na elaboração de programas altamente paralelos, que utilizem eficientemente os recursos disponíveis na arquitetura proposta.

4.5 - Dimensionamento da Central de Comutação

Nesta seção são feitos cálculos para o dimensionamento do número de processadores elementares necessários à arquitetura em anel para que esta ofereça um determinado tráfego telefônico (medido em Erlangs).

Para realizar-se este dimensionamento consideram-se os Processadores Elementares como sendo o conjunto de servidores de uma fila e a Unidade de Instruções Executáveis (que contem as instruções) como sendo a sala de espera, formando a fila.

Considerando o tempo de serviço na Unidade de Ativação com distribuição exponencial negativa, podemos afirmar que as chegadas de instruções na Unidade de Instruções Executáveis obedece a uma distribuição Poissoniana. E considerando também que o tempo de serviço na Unidade de Processamento obedece a uma distribuição exponencial, podemos aproximar a fila por um modelo M/M/n (chegada e atendimento poissoniano com n servidores) na notação de Kendall.

Numa fila M/M/n , a expressão que determina o comprimento médio da fila (número médio de instruções presentes na fila de

espera) é dado por:

$$E(n) = \sum_{n=0}^{\infty} n * \frac{\dfrac{(n * RO)^n}{n!} * \dfrac{1}{1 - RO}}{\sum_{K=0}^{n-1} \dfrac{n * RO}{K!} + \dfrac{(n * RO)^n}{n!} * \dfrac{1}{1 - RO}} \qquad (4\text{-}43)$$

onde:

$$RO = \frac{L}{n * \#} \qquad (4\text{-}44)$$

sendo:

L - a taxa de chegada de instruções durante o período de um segundo, determinado pelo tráfego a ser oferecido.

\# - a taxa de atendimento de instruções por segundo da Unidade de Processamento, considerou-se o tempo de execução de uma intrução no processador Z80A entre 1 e 7,5 microsegundos.

Para calcular o tamanho médio da fila de espera (E(n)), foi executado um programa cujos resultados encontram-se plotados na figura 4.4.

Para chegar-se ao número de processadores necessários à arquitetura, em função de um determinado tráfego telefônico, foram também considerados dados sobre a central de Comutação D10-NTT (NIPPON TELEPHONE AND TELEGRAPH) relacionados abaixo:

i - uma chamada esta subdividida em 6 tarefas,

ii - uma tarefa contém 15 macro-tarefas e,

iii - uma macro-tarefa contém 60 instruções.

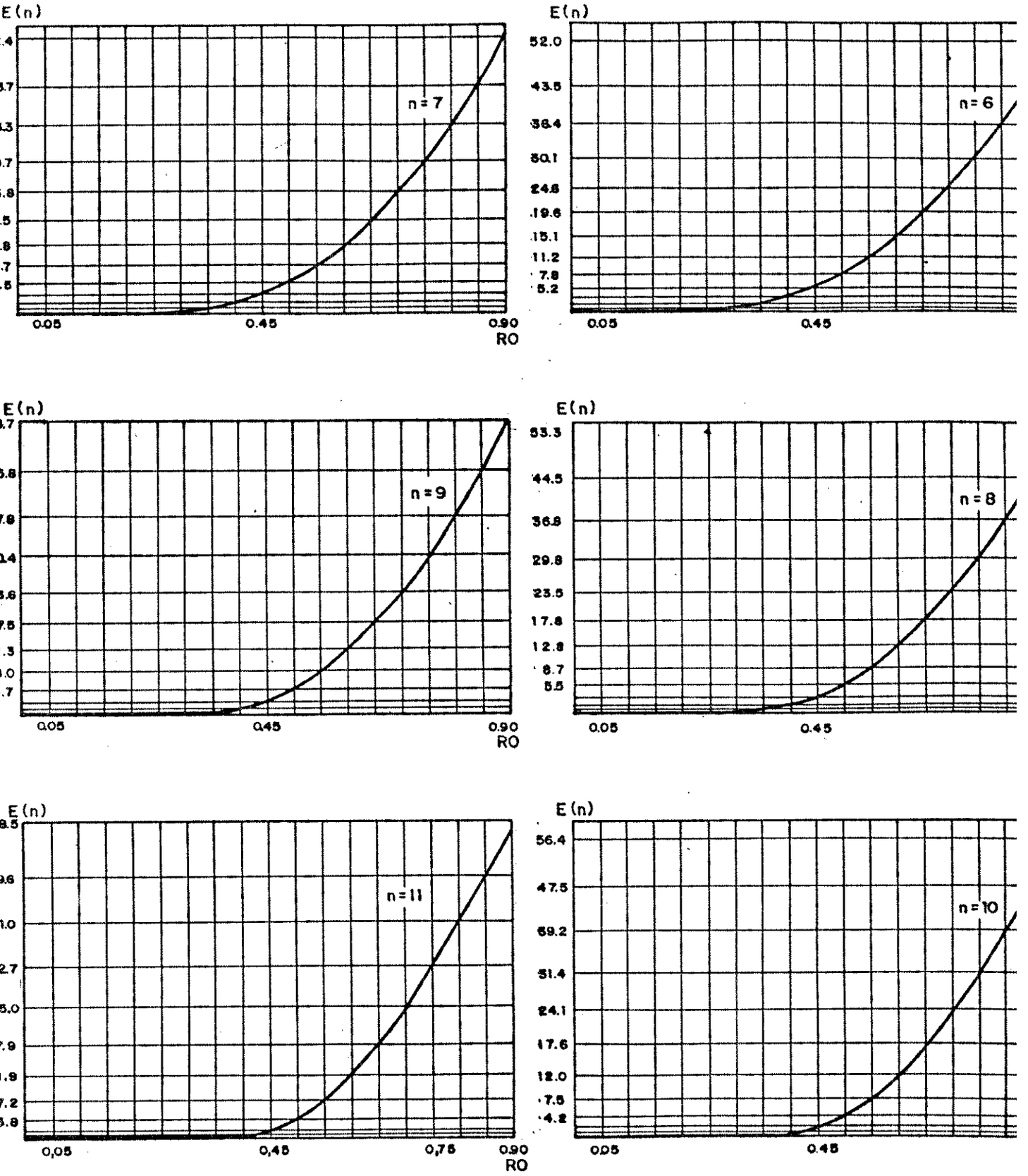


FIGURA 4.4 - E(n) em função de RO (não normalizado) e n

Assim em uma chamada são necessárias em média 5400 instruções.

4.6 - Exemplo de Dimensionamento

Será dimensionado neste exemplo o número de processadores necessários à arquitetura de controle em anel, para um tráfego de 4.000 Erlangs e um tempo de retenção de chamada de 120 segundos. Isto implica em 33,33 chamadas por segundo e consequentemente 180.000 instruções por segundo (de acordo com as características da D10-NTT).

Utilizando-se o tempo médio de execução de uma instrução do microprocessador Z80A, em torno de 3 microsegundos, serão necessários em média 24 microsegundos para processar uma palavra de 64 bits, fazendo R0 = 0,5, teremos que n será:

$$n = \frac{L}{R0 * \#} = \frac{180.000}{0,5} * 24 * 10^{-6} = \frac{0,18}{0,5} * 24 = 8,64 \qquad (4\text{-}45)$$

Assim serão necessários 9 processadores para que a central de comutação a fluxo de dados atenda a um tráfego de 4.000 Erlangs, com um tempo de retenção de 120 segundos.

Além de verificar-se a necessidade do número de processadores elementares necessários para atender a um determinado tráfego telefônico, é necessário obter-se outro importante parametro para avaliar-se a eficiência da central, que é o atraso médio de uma instrução. Neste exemplo, obteve-se que o número médio de instruções presentes na fila de espera (Unidade

de Instruções Executáveis) está em torno de 12. Consultando-se a curva para 9 processadores mostrada na figura 4,4 obtem-se um atraso médio de instruções em torno de 66 microsegundos.

Na análise de desempenho da arquitetura em anel obteve-se que a taxa de execução de instruções está entre 1,1 MIPS e 6,6 MIPS. Sendo que para efetuar esta análise foram consideradas certas condições ideais que são: um sistema de execução perfeito (ver item 4.2.2) e, programas que contenham paralelismo médio (Pm) maior ou igual ao paralelismo do anel.

No exemplo de dimensionamento efetuado, modelou-se a arquitetura através de uma fila M/M/n, considerado mais realístico, e obteve-se que para a central atender a um tráfego de 4.000 Erlangs o processador da central de commutação D10-NTT precisa executar 180.000 instruções .

Consegue-se fazer uma comparação entre a análise de eficiência do modelo M/M/n com o modelo de análise ideal. Supondo que uma instrução na D10-NTT equivale a um pacote de instrução fluxo de dados do anel e levando-se em consideração que o tamanho de um pacote de instrução do anel é em média oito "bytes" maior que o tamanho de uma instrução do Processador Elementar (Z80A), obtém-se, para um tráfego de 4.000 Erlangs, uma taxa de execução média de instrução em torno de 1,44 MIPS, dentro dos limites obtidos pela análise ideal.Entretanto no exemplo de dimensionamento nós temos em média 12 pacotes de instruções a espera de atendimento na Unidade de Instruções Executáveis, e no caso de análise ideal somente um pacote. A taxa de execução de 1,44 MIPS do exemplo, é conseguida através de atraso em cada pacote de instrução, o que aumenta a eficiência dos Processadores

Elementares. No caso de análise ideal a eficiência é conseguida através de execuções de programas cujo paralelismo seja maior ou igual ao paralelismo do anel.

4.7 - Conclusão 3

Neste capítulo foi realizada uma análise de desempenho da arquitetura proposta no capítulo 3. Conseguindo-se uma razão de execução máxima de instruções em torno de 27 MIPS na arquitetura expandida, no anel temos que a taxa de execução está entre 1,1 e 6,6 MIPS, sendo que a taxa de execução média será maior que 1,66 MIPS.

Foi também realizado um exemplo de dimensionamento da arquitetura para um tráfego telefônico de 4000 Erlangs e um tempo de retenção de 120 segundos, verificando-se que são necessários 9 Processadores Elementares para que a central de comutação controlada pela arquitetura em anel atenda às características mencionadas acima. Obtendo-se também um tempo de atraso médio de instruções em torno de 66 microsegundos.

CAPÍTULO 5

CONCLUSÃO GERAL

Este trabalho foi realizado com o intuito de solucionar o problema de engarrafamento de "software" hoje existentes nas centrais de comutação eletrônicas comerciais, que utilizam um processador com arquitetura de Von Neumann para realizar a tarefa de controle da central.

Para alcançarmos o objetivo do trabalho foi realizado um estudo sobre arquiteturas não convencionais de computadores que apresentassem características de paralelismo na execução de instruções. Feito este estudo, optou-se por uma arquitetura à fluxo de dados por esta possuir características de desempenho semelhantes ao de uma central de comutação telefônica convencional, facilitando dessa forma a confecção do "software" de controle.

No fluxo de dados o mecanismo de execução de instruções é dirigido por dados ("data-driven"). Uma tarefa de uma chamada telefônica só é executada se houver disponibilidade de operandos e recursos simultaneamente, portanto podemos dizer que o mecanismo de execução dessa tarefa também é dirigido por dados.

Decidindo-se pelo fluxo de dados, foi então feito um levantamento sobre o estado atual da arte, onde foi estudado os conceitos inerentes ao fluxo de dados e pesquisado algumas arquiteturas propostas por vários grupos de pesquisadores. Após este levantamento foi feito uma comparação entre as várias arquiteturas existentes na bibliografia para finalmente propor uma arquitetura a fluxo de dados organizada em forma de anel para efetuar o controle de uma central telefônica.

A arquitetura fluxo de dados proposta é composta por quatro blocos básicos formando um anel. De forma a obtermos vantagens do

paralelismo "pipeline" inerentes à organização em anel. Alguns dos blocos básicos foram projetados a níveis de blocos lógicos visando velocidade de processamento e facilidades para expansões, conseguindo-se assim um processador cujas principais características são eficiência e flexibilidade para expansão, possibilitando consequentemente facilidades de dimensionamento de uma central telefônica.

Tendo definido a arquitetura a níveis de portas lógicas, foi realizado uma análise de desempenho dessa arquitetura obtendo-se uma razão de execução máxima de instruções em torno de 27 MIPS (milhões de instruções por segundo), sendo que a taxa de execução média será sempre maior que 1,66 MIPS para programas que contenham um paralelismo médio maior ou igual ao paralelismo do anel (paralelismo "pipeline" mais o paralelismo na Unidade de Processamento). Assim chega-se à conclusão que para utilizarmos eficientemente a arquitetura fluxo de dados em anel é necessário realizar um grande esforço na elaboração de programas altamente paralelos.

Foi também feito um exemplo de dimensionamento da arquitetura proposta para que esta realizasse o controle de uma central de comutação telefônica que atendesse a um tráfego telefônico de 4.000 Erlangs com um tempo de retenção de 120 segundos, concluindo-se que são necessários 9 Processadores Elementares na Unidade de Processamento. Neste exemplo obteve-se também que o número médio de instruções à espera de atendimento é igual a 12, sendo que o tempo médio de atraso de uma instrução está em torno de 66 microsegundos.

De posse de todos esses dados verifica-se que este tipo de

arquitetura oferece boas perspectivas para conseguir-se centrais de comutação telefônicas com alto desempenho e confiabilidade.

Para convalidar-se de forma mais realística os resultados obtidos neste trabalho, seria necessário a realização de uma simulação da arquitetura fluxo de dados em anel aqui proposta e, a construção de ferramentas que possibilitem a confecção de programas de fluxo de dados altamente paralelos, esta ferramenta deverá ter recursos para a representação de grafos dirigidos, recursos gráficos por exemplo, pois como visto, um programa de fluxo de dados é um grafo dirigido.

A simulação a ser realizada deverá considerar a distribuição de tarefas (carga) nos processadores elementares para dimensionar-se de forma mais realista o número de processadores necessários para que a central atenda a um determinado tráfego telefônico. De posse dos resultados desta simulação será possivel avaliar-se de forma mais concreta a viabilidade da arquitetura a fluxo de dados proposta neste trabalho, realizando-se comparações com centrais de comutação comerciais com controle também distribuido.

Uma possivel ferramenta para a realização desta simulção pode ser o ferramental de Redes de Petri, devido às características de desempenho semelhantes entre um grafo fluxo de dados e uma Rede de Petri.

Então, os próximo passo para a continuidade do trabalho é a realização da simulação da arquitetura aqui proposta e, a contrução de ferramentas para confecção de programas, visando uma futura implementação.

6

REFERENCIAS

AA82 - AGERWALA,T. & ARVIND.. Data Flow Systems. IEEE Computer, Feb., 1982. p.10-4.

Ac82 - ACKERMAN, W. B.. Data Flow Languages. IEEE Computer, Feb., 1982. p.15-25.

AYM84 - AKIYAMA,M.; YAMASHITA,M.; MISU,T.. DATAFLEX-1.. An Experimental Data Flow Computer Controlled Eletronic Switching Systems. ISS'84 May., 1984. 7p. (session 23B, paper 2).

BMa86 - BUGATTI, I. G. & MOTOYAMA, S.. Computadores Baseados em Fluxo de Dados : Conceitos e Aplicacões. DEE.FEC.UNICAMP., RI04,33p., jul., 1986.

BMb86 - BUGATTI, I. G. & MOTOYAMA, S.. Uma Proposta de Arquitetura Fluxo de Dados para Controlar uma Central de Comutação. DEE.FEC.UNICAMP., RI05, 32p., jul., 1986.

BMc86 - BUGATTI, I. G. & MOTOYAMA, S.. Central de Comutaçao Controlada por um Computador á Fluxo de Dados. Quarto Simpósio Brasileiro de Telecomunicações (SBT). Mar., 1986. p.115-20.

CPqD85- CENTRO DE PESQUISA E DESENVOLVIMENTO Trópico R: Central local Tandem digital do sistema trópico. CPqD TELEBRÁS, 1985. 163p.

De74 - DENNIS, J. B.. First version of a Data Flow Procedure Language. Lecture Notes in Computer Science, (19):362-76, 1974.

De80 - DENNIS, J. B.. Data Flow Supercomputers. IEEE Computer, Nov., 1980. p.48-56.

DM75 - DENNIS, J. B. & MISUMAS D. P.. A Preliminary Architecture for a Basic Data-Flow Processor. In: ANNUAL SYMP. COMPUTER ARCHITETURES, 2. Proceedings. 1975. p.126-32.

DK82 - DAVIS,A.L. & KELLER,R.M.. Data Flow Programs Graphs. IEEE Computer, Feb., 1982. p.26-41.

GKW85 - GURD,J.R.; KIRKHAN,C.C.; WATSON,I.. The Manchester Prototype Dataflow Computer. Communication of The ACM, 28, (1):34-52, 1985.

GW80 - GURD,J. & WATSON,I.. Structuring software for Parallel Execution I. In:Data Driven Systems for High Speed Parallel Computing. Computer Design, Jun., 1980. p.91-100.

GW80 - GURD,J. & WATSON,I.. Hardware Design.II. In: Data Driven System for High Speed Parallel Computing. Computer Design, Jul., 1980. p.97-106.

GWG78 - GURD,J.; WATSON,I.; GLAUERT,J.. A Multilayered Data Flow Computer Architecture. University of Manchester, Departament of Computer Science, Jul., 1978. 56p.

In81 - INTERCONNECTION Networks. Computer, 14(12):8-65, Dec., 1981.

P172 - PLUMMER, W. W.. Asynchronous Arbiters. IEEE Transactions on Computers, c.21(1):37-42, 1972.

Ru77 - RUMBAUGH, J.. A Data Flow Multiprocessor. IEEE Transactions on Computer, Feb.,1977. p.138-46.

Sp77 - SPECIAL Issue in Telecomunications Circuit Switching. Procedings of the IEEE, 65(9) : 1235-1424, Sep., 1977.

TBH82 - TRELEVEAN,P.C.; BROWNBRIDGE,D.R.; HOPKINS, R.P.. Data - Driven and Demand - Driven Computer Architecture. ACM Computer Surveys, Mar.,1982. p.93-143.

WG82 - WATSON,I. & GURD,J.. A Practical Data Flow Computer. IEEE Computer, Feb., 1982. p.51-7.

7

APENDICES

APENDICE 1

SÍNTESE DO CIRCUITO DE CONTROLE DO COMUTADOR DE ENTRADA E SAÍDA DA ARQUITETURA EXPANDIDA

Tabelas verdade para síntese do circuito de controle do Comutador de Entrada e Saída da arquitetura expandida.

Podemos verificar que as colunas referentes aos blocos A, B, C, D, E, e F serão sempre coincidentes com o conteúdo dos bits m_0, b_1, m_1, b_2, m_2 e b_0 respectivamente, portanto o controle será o mostrado pela figura 7.1. Onde C_2, C_1 e C_0 é o conteúdo de um contador módulo oito, que tem a função de varrer sequencialmente as entradas do comutador de entrada e saida, e b_2, b_1 e b_0 é o conteúdo do campo bloco especificado no campo de endereço da palavra de instrução da arquitertura em anel proposta (figura 3.7)

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
0	0	0	0	0	0	0	0	0	0	0	0
			0	0	1	0	0	0	0	0	1
			0	1	0	0	1	0	0	0	0
			0	1	1	0	1	0	0	0	1
			1	0	0	0	0	0	1	0	0
			1	0	1	0	0	0	1	0	1
			1	1	0	0	1	0	1	0	0
			1	1	1	0	1	0	1	0	1

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
0	0	1	0	0	0	1	0	0	0	0	0
			0	0	1	1	0	0	0	0	1
			0	1	0	1	1	0	0	0	0
			0	1	1	1	1	0	0	0	1
			1	0	0	1	0	0	1	0	0
			1	0	1	1	0	0	1	0	1
			1	1	0	1	1	0	1	0	0
			1	1	1	1	1	0	1	0	1

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
0	1	0	0	0	0	0	0	1	0	0	0
			0	0	1	0	0	1	0	0	1
			0	1	0	0	1	1	0	0	0
			0	1	1	0	1	1	0	0	1
			1	0	0	0	0	1	1	0	0
			1	0	1	0	0	1	1	0	1
			1	1	0	0	1	1	1	0	0
			1	1	1	0	1	1	1	0	1

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
0	1	1	0	0	0	1	0	1	0	1	0
			0	0	1	1	0	1	0	1	1
			0	1	0	1	1	1	0	1	0
			0	1	1	1	1	1	0	1	1
			1	0	0	1	0	1	1	1	0
			1	0	1	1	0	1	1	1	1
			1	1	0	1	1	1	1	1	0
			1	1	1	1	1	1	1	1	1

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
1	0	0	0	0	0	0	0	0	0	1	0
			0	0	1	0	0	0	0	1	1
			0	1	0	0	1	0	0	1	0
			0	1	1	0	1	0	0	1	1
			1	0	0	0	0	0	1	1	0
			1	0	1	0	0	0	1	1	1
			1	1	0	0	1	0	1	1	0
			1	1	1	0	1	0	1	1	1

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
1	0	1	0	0	0	1	0	0	0	1	0
			0	0	1	1	0	0	0	1	1
			0	1	0	1	1	0	0	1	0
			0	1	1	1	1	0	0	1	1
			1	0	0	1	0	0	1	1	0
			1	0	1	1	0	0	1	1	1
			1	1	0	1	1	0	1	1	0
			1	1	1	1	1	0	1	1	1

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
1	1	0	0	0	0	0	0	1	0	1	0
			0	0	1	0	0	1	0	1	1
			0	1	0	0	1	1	0	1	0
			0	1	1	0	1	1	0	1	1
			1	0	0	0	0	1	1	1	0
			1	0	1	0	0	1	1	1	1
			1	1	0	0	1	1	1	1	0
			1	1	1	0	1	1	1	1	1

origem			destino			níveis de comutação					
m_2	m_1	m_0	b_2	b_1	b_0	A	B	C	D	E	F
1	1	1	0	0	0	1	0	1	0	1	0
			0	0	1	1	0	1	0	1	1
			0	1	0	1	1	1	0	1	0
			0	1	1	1	1	1	0	1	1
			1	0	0	1	0	1	1	1	0
			1	0	1	1	0	1	1	1	1
			1	1	0	1	1	1	1	1	0
			1	1	1	1	1	1	1	1	1

APÊNDICE 2

O teorema que garante o grau de utilização da arquitetura a fluxo de dados em anel foi adaptado de [GWG78] e possui o seguinte enunciado:

Teorema- Para um programa fluxo de dados executado num sistema de execução perfeita com m' processadores temos que:

$$\frac{1}{2} < Em' <= 1$$

Prova:

i- Para Cm' finito, pela própria definição de Cj (ver pag. 74) temos que:

$$Cm' >= Coo$$

então:

$$Em = \frac{Coo}{Cm'} <= 1$$

ii- O número de passos em Cm' pode ser determinado considerando-se os passos de Coo. Para o i-ésimo passo de Coo, o número máximo de passos em Cm é o primeiro inteiro maior que ei/m',onde ei é o número de instruções elegíveis no passo i. Assim:

$$Cm \leq \sum_{i=1}^{Coo} \frac{ei}{m'}$$

$$< \sum_{i=1}^{Coo} \frac{Coo * ei}{Ci} + \sum_{i=1}^{Coo} S \quad (1)$$

$$< \sum_{i=1}^{Coo} \frac{Coo * ei}{Ci} + Coo$$

$$< \frac{Coo}{Ci} \sum_{i=1}^{Coo} S\ ei + Coo$$

$$< \frac{Coo}{Ci} * Ci + Coo$$

$$< 2 * Coo$$

Portanto:

$$Em = \frac{Coo}{Cm'} > \frac{Coo}{2 * Coo} = \frac{1}{2}$$

Assim:

$$\frac{1}{2} < Em' \leq 1$$

Em' é uma medida de ocupação média do sistema e esta ocupação é sempre maior que 50% dos componentes do sistema.

APENDICE 3

EQUIVALÊNCIA ENTRE UM SISTEMA PERFEITO E O SISTEMA EM ANEL

A equivalência entre um sistema perfeito e o sistema em anel é facilmente entendida com os dois exemplos citados a seguir:

Exemplo 1: Considere uma computação totalmente serial. Cada instrução produz pacotes de dados que tornam elegível a próxima instrução. Somente uma das instruções torna se elegível em cada passo. No sistema em anel os resultados correspondentes aos pacotes de dados devem caminhar através do anel (da saída da unidade de processamento) até gerar um novo pacote de instruções (na entrada da unidade de processamento). Existe então um tempo medio de execução Tm, antes do início do proximo passo. Claramente o tempo de passo nesse caso é Ta+Tm e o tempo de execução de um programa completo é Ci*(Ta+Tm). O tempo médio para, executar uma instrução é (Ta+Tm). Note que a Unidade de Processamento necessita conter um único processador.

Exemplo 2: Considere L computações seriais (como a do exemplo 1) sendo executadas simultaneanente (em paralelo). Essas computações podem ser executadas de maneira sobreposta, de tal modo que a iésima computação inicie a execução da instrução

corrente assim que o resultado produzido pela instrução anterior da (i+1) mod Lésima computação completar seu caminho através do anel, o tempo de passo e o tempo total de execução continuam sendo Ta+Tm e Ci*(Ta+Tm) respectivamente. O tempo médio para executar uma instrução é agora (Ta+Tm)/L e a Unidade de Pocessamento necessitará ao menos Tm/Tmin processadores.

Em ambos os casos acima, o comportamento do anel é idêntico ao do sistema perfeito. Obtendo-se um comportamento similar para qualquer programa fluxo de dados, consegue-se determinar a eficiência geral da arquitetura em anel avaliando-se Tmin, Ta, Tm e K.

Para avaliar-se Tmin, considera-se a execução de um programa completo que consome x pacotes de dados de entrada, executa Ci instruções,e produz y resultados de saída. Determina-se as relações entre o número total de pacotes passando através de cada parte do sistema durante o tempo de execução do programa (Te). Na figura 4.2 é mostrada a arquitetura em anel com simbolos representando os números totais de pacotes que caminham através dos enlaces de dados entre os módulos.

APENDICE 4

RELAÇÃO DE TERMOS

a - quantidade de resultados gerados pela Unidade de Processamento.

a1 - quantidade de instruções unárias.

a2 - quantidade de instruções binárias.

a2e - quantidade de operandos que ficam armazenados na Unidade de Ativação (primeiro operando disponível de uma instrução binária).

a21 - quantidade de operandos que tornam instruções elegíveis.

CES - Canal de Entrada e Saída.

Cj - número de passos necessários para executar um programa com "j" processadores.

Cm - Paralelismo Médio do programa.

C1 - Quantidade de instruções num programa (tempo serial).

Coo - Número de passos para executar todas as intruções de um programa (tempo paralelo).

d - quantidade de resultados gerados pela Unidade de Processamento

ei - número de intruções elegíveis no passo "i".

Ej - Eficiencia de ocupação média do sistema com "j" processadores (medida de ocupação média do sistema).

K - Paralelismo no anel.

P1e - Proporção de instuções unárias .

P2e - Proporção de instruções binárias.

P1s - Proporção de instruções com apenas uma instrução sucessora.

P2s - Proporção de instruções com duas instruções sucessoras.

P0s - Proporção de instruções sem instrução sucessora.

ta - tempo de ciclo da Unidade de Ativação.

Ta - Tempo de atraso no anel.

Tb - Tempo de atraso num bloco do anel.

tc - tempo de ciclo do Canal de Entrada e Saída.

Te - Tempo de execução de um programa.

ti - tempo de ciclo da Unidade de Instruções Executáveis.

Tm - Tempo de execução médio de uma instrução.

Tmin - Tempo mínimo entre instruções no anel.

tp - tempo de ciclo da Unidade de Processamento.

Tp - Tempo de passo da arquitetura.

U.A. - Unidade de Ativação.

U.I.E. - Unidade de Instruções Executáveis.

U.P. - Unidade de Processamento.

x - número de pacotes de dados que entram no sistema.

y - número de pacotes de dados que saem do sistema.