Processador de Rede com Suporte a Multi-protocolo e Topologias Dinâmicas

Henrique Cota de Freitas[1], Carlos Augusto P. S. Martins[2]
[1] Datapuc – Processamento de Dados
Pontifícia Universidade Católica de Minas Gerais
Av. Francisco Sales 540, Floresta, Belo Horizonte
{cota@pucminas.br}
[2] Laboratório de Sistemas Digitais e Computacionais (LSDC)
Instituto de Informática
Pontifícia Universidade Católica de Minas Gerais
Av. Dom José Gaspar, 500, Coração Eucarístico, Belo Horizonte
{capsm@pucminas.br}

Resumo—

Este artigo descreve um processador de rede com microarquitetura dedicada e conjunto de instruções otimizadas para aplicação em redes de computadores (SAN, LAN, WAN). O grande uso de roteadores e outros *gateways*, que são responsáveis pelo tráfego e gargalos nas redes, têm contribuído para a melhoria dos processadores. Esta pesquisa mostra um novo conceito de processador que suporta multi-protocolo e topologias dinâmicas usando microarquitetura de aplicação específica e conjunto de instruções otimizadas. Algumas características e resultados são comparados com processadores de propósito geral e processadores de rede comerciais.

Palavras-chave— **Processador de Rede; Multi-protocolo; Topologias Dinâmicas, Arquitetura Reconfigurável.**

Abstract—

This paper presents a network processor with dedicated micro-architecture and instruction set optimized to be used in computer networks (SAN, LAN, WAN). The large use of routers and other gateways, that are responsible for traffic and bottleneck in networks, has contributed to better processors. This research shows a new concept of processor that supports multi-protocol and dynamic topology using optimized and application specific micro-architecture and instruction set. Some characteristics and results are compared with commercial general-purpose and network processors.

Keywords— **Network Processor; Multi-protocol; Dynamic Topology, Reconfigurable Architecture.**

I. INTRODUÇÃO

A internet tem se mostrado cada vez mais importante, no cotidiano das pessoas, instituições de ensino e empresas. Sua grande utilização acarreta um aumento muito grande no congestionamento dos links de transmissão. Os principais responsáveis são os *gateways,* mais especificamente os roteadores. Os roteadores confinam o tráfego entre redes, filtram os pacotes e por isso devem ter um desempenho rápido e eficaz para evitar gargalos. A grande questão é aumentar a capacidade de processar os pacotes, para que o desempenho não comprometa o tráfego, ocasionando congestionamentos e excessiva utilização da largura de banda das linhas de comunicação.

A Lucent Technologies publicou alguns artigos [LUC 99] relacionados com os processadores de rede, e um dos artigos trata da evolução desses processadores. Até meados da década de noventa, os roteadores eram baseados em processadores de propósito geral e sua arquitetura era parecida com a dos computadores pessoais. Sua principal vantagem era a possibilidade de solucionar os problemas através dos softwares, se adequando às necessidades de roteamento. Com a evolução dos protocolos [CAM 00] [COM 95] e aumento das tarefas que um roteador tem que realizar, ficou impraticável utilizar softwares para solucionar os problemas. Os códigos se tornaram maiores, comprometendo o desempenho dos roteadores. Desenvolveu-se então, os processadores de rede [INT 00] [LUC 99] [MOT 99]. Dedicados para trabalhar na camada de redes, estes processadores são mais rápidos, utilizam menos código e sua arquitetura está preparada para se adaptar às topologias das redes de comunicação.

Nosso principal objetivo é desenvolver um processador de rede [FRE 00], capaz de se comportar de forma eficiente em um ambiente de rede heterogêneo. Suas características principais são: i) suporte a multi-protocolo, ii) suporte a topologias dinâmicas, iii) Portas de entradas reconfiguráveis, iv) matriz de comutação interna (*crossbar*) e v) conjunto de instruções específicas.

O processador de rede proposto, apesar de ter sido inicialmente motivado pela internet, pode ser utilizado em qualquer tipo de rede, entre as quais podemos citar: i) SAN (*System Area Network*), ii) LAN (*Local Area Network*), iii) MAN (*Metropolitan Area Network*), iv) WAN (*Wide Area Network*) e v) Redes sem fio. Ou seja, este processador pode ser usado em uma rede de *cluster* [BAR 00] [IEE 01] passando pela internet [COM 95] e usando qualquer protocolo que se deseje implementar, só depende do programa (*software*) que estiver sendo executado.

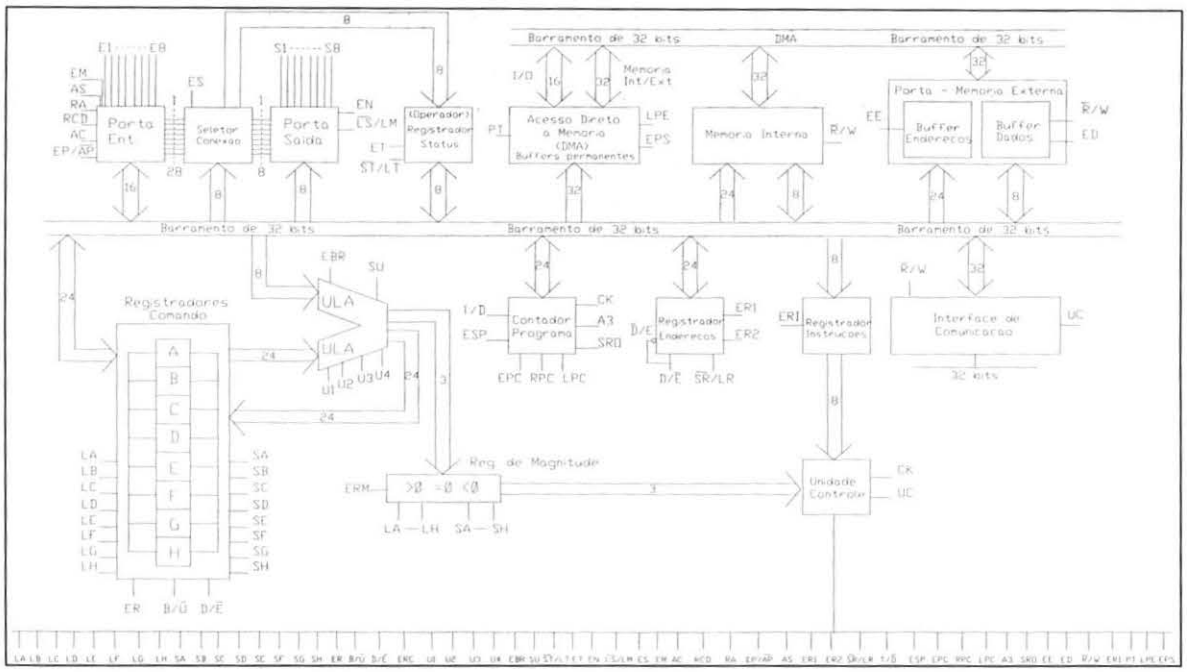

Fig. 1 – Microarquitetura do processador

II. ARQUITETURA DO PROCESSADOR PROPOSTO

O processador [FRE 00] foi desenvolvido com o objetivo inicial de validar alguns conceitos da microarquitetura e do conjunto de instruções. Sua tecnologia CISC (*Complex Instruction Set Computing*), possui um barramento de 24 bits de endereçamento e 8 bits de dados. A memória de 16 Mbytes pode ser acessada por um computador externo, para alteração de dados, e pelas portas de entrada e saída, através de DMA (Acesso Direto à Memória). A Figura 1 ilustra a arquitetura interna do processador, e os principais blocos construtivos serão descritos com mais detalhes a seguir.

A **Porta de Entrada** (8 entradas) tem a função de receber os pacotes que serão redirecionados para o barramento principal, barramento DMA ou diretamente para a Porta de Saída, através do Seletor de Conexão. A Porta de Entrada possui um conjunto de *buffers* reconfiguráveis, capazes de armazenar os pacotes recebidos de acordo com o seu tamanho. Estes *buffers* aumentam a otimização da arquitetura, já que a escolha da alocação de pacotes vai acontecer de acordo com seu tamanho, liberando mais espaço de armazenamento temporário, para pacotes de tamanhos maiores e que necessitam de *buffers* de maior capacidade. Outra característica importante é a possibilidade de utilização de circuitos externos às Portas de I/O para aumentar o número de nós atendidos pelo processador durante sua utilização.

O **Seletor de Conexão** tem função importante nesta arquitetura. Ele irá redirecionar os pacotes recebidos pela Porta de Entrada, para a Porta de Saída, sem que os pacotes sejam direcionados para qualquer um dos barramentos. Este seletor funciona como uma matriz ou um comutador *crossbar* "configurável". Dessa forma, é possível obter topologias diferentes a cada nova conexão realizada. Todas estas conexões são dependentes do programa que está sendo executado no processador.

A **Porta de Saída** (8 saídas) realiza o envio dos pacotes recebidos. Esta porta está conectada ao Seletor de Conexão e aos dois barramentos. A comunicação com os dois barramentos é necessária, para que os pacotes guardados em memória possam ser enviados para a saída desejada (ou pelo processador, ou pelo DMA). Através do **Registrador de Status,** é possível saber qual saída está sendo usada, para que não haja conflitos entre pacotes.

O bloco **DMA** (Acesso Direto à Memória) possibilita o acesso das portas de I/O à memória, sem que seja necessário o uso do processador e do barramento principal. Este bloco utiliza um barramento próprio para acessar a memória.

A **Memória** utilizada por este processador é segmentada em interna e externa. Parte da memória de 16Mbytes é interna ao processador, aumentado a velocidade de acesso e diminuindo a latência. A maior parte é externa, podendo ser acessada através de uma porta dedicada.

A **Interface de Comunicação** é um bloco lógico encarregado de ampliar a comunicação entre processadores, do tipo proposto, e também com outros dispositivos de rede. Ainda não definimos o protocolo a ser usado, devendo ser implementado juntamente com a prototipação em VHDL (*VHSIC Hardware Description Language*) e FPGA (*Field Programmable Gate Array*) [WAG 98] [XIL 98].

É importante ressaltar que, a arquitetura interna proposta, se diferencia pelo fato de poder, através das Portas de I/O, do Seletor de Conexão e do DMA, suportar qualquer topologia. O Seletor de Conexão (comutador *crossbar*) pode fechar conexão entre qualquer porta de entrada e saída, inclusive fechando conexões entre uma entrada e todas as saídas ao mesmo tempo, possibilitando o *broadcast*. Existe a possibilidade de se criar uma topologia em anel unidirecional, já que uma entrada pode enviar dados apenas para seu vizinho da direita e receber apenas do vizinho da esquerda. Estas variações de topologias são de responsabilidade do programa que estiver rodando. Portanto, as portas de I/O e Seletor de Conexão são comandadas pelas instruções executadas pelo processador.

III. CONJUNTO DE INSTRUÇÕES DO PROCESSADOR

Ao todo são 54 instruções, entre operações lógicas e aritméticas, acesso às portas, acesso à memória, saltos condicionais e incondicionais, acesso à pilha de dados, chamada de rotina e movimentação de valores.

O conjunto de instruções foi projetado com o intuito de otimizar o processamento de instruções específicas de roteamento, que estejam ligadas diretamente às Portas de Entrada / Saída, ao Registrador de *Status* e Seletor de Conexão. Através destas instruções, é possível aumentar a velocidade de processamento e, portanto, do recebimento e envio dos pacotes utilizados na comunicação. Instruções de saltos condicionais como o *jump*, também tem características otimizadas. Esta análise se refere ao tipo do modo de endereçamento, se é direto, indireto, indexado ou imediato.

Uma das formas de se entender este aumento de velocidade, é por meio de instruções que se encontram em *loops* e por instruções que acessam as portas de comunicação (Tabela 1). Exemplos:

1) Uma vez que se precisa testar um valor, através da instrução condicional do tipo, *JNZ End* (se valor de Reg. Magnitude não for zero, pula para End), o endereço (End) deve ser buscado em memória. Dentro de um *loop* este valor seria buscado quantas vezes fosse o valor de voltas à instrução de teste, e por conseqüência, o acesso à memória seria feito várias vezes, aumentando o tempo de processamento. Caso a instrução fosse, *JNZ F*, o acesso à memória seria feito apenas uma vez, já que o endereço estaria, neste caso, carregado e guardado nos registradores FGH, já que cada registrador é de 1 byte.

2) No caso de acesso às portas de comunicação, é importante perceber que as instruções irão controlar o funcionamento dos *buffers* reconfiguráveis, aumentando a otimização do armazenamento dos pacotes. Irão controlar também, o acesso ou não, ao barramento e ao Seletor de Conexão, aumentando, portanto, a flexibilidade de conexões entre portas, e por conseqüência, criando topologias novas sempre que necessário.

As instruções *LRS* e *SRS* referem-se ao acesso feito no Registrador de *Status*. Através do registrador de comando B, é possível carregar (*LRS*) e ler (*SRS*) dados de *status*.

ENT BC e *ENT, SET C,DE* e *SETDSC* se referem à Porta de Entrada. *ENT BC* retira dos *buffers*, o dado, de acordo com a posição descrita nos registradores B e C concatenados. *ENT* retira os dados da primeira posição dos *buffers* (descarregando-os) e *SET C,DE* carrega em registros de cada *buffer* temporário, a região do pacote que contém a informação de tamanho. Reg. C define qual o *buffer* e o segmento de início em reg. D e final em reg. E, define a região do pacote. A utilização desta instrução e o significado de *buffers* temporários e permanentes serão descritos no tópico VI, que trata da reconfiguração. *SETDSC* é uma instrução que seta em registradores dos *buffers* temporários (reg. D define o *buffer*) qual o valor (definido em reg. C) que, contido no pacote, desfaz a conexão feita por *FCX* (que será descrito a seguir).

TABELA 1
Algumas instruções específicas de roteamento

LRS	Reg. Status ← Reg. B
SRS	Reg. B ← Reg. Status
SEC	Buffer (Seletor Conexão) ← Reg. B
SEL	Buffer (seleção PS) ← Reg. B
SAI	Buffer (mensagem PS) ← Reg. A
BRC	PS (Todas) ← Buffer de Entrada
FCX	PS ← Buffer temporário (Fecha conexão)
ENT BC	Reg. A ← Buffer (mensagem PE – posição BC)
ENT	Reg. A ← Buffer (mensagem PE – posição inicial)
PUT	Buffer (seleção PE) ← Algoritmo de escolha
PUT B	Buffer (seleção PE) ← Reg. B
SETDSC	Reg. Desconexão ← Reg. D (buffer) Reg. C (dado)
SET C,DE	Buffer C (tam. Pacote) ← Reg. D até E
TAM	Reg. C e D ← Buscar tamanho pacote (Buffer de trabalho, escolhido por PUT)
TAM A	Reg. C e D ← Buscar tamanho pacote (Buffer escolhido por Reg. A)

As instruções *TAM, TAM A* , *PUT* e *PUT B* também se referem a Porta de Entrada. A instrução *PUT indica* à Porta de Entrada, que o próximo *buffer* a ser escolhido para análise do pacote, será feita através do próprio *hardware*, que contém algoritmo de escolha *FIFO – First in first out*. *PUT B* indica, através do reg. B, em qual *buffer* será feito à análise do pacote. *TAM A* busca o tamanho do pacote, indicado anteriormente por *SET C,DE* (*A,* indica qual *buffer*) e carrega nos registradores C e D, o tamanho do

pacote. A instrução *TAM* faz o mesmo procedimento, porém o *buffer* em análise é escolhido por *PUT* ou *PUT B*.

A instruções *SEL* e *SAI* se referem à porta de saída. *SEL* indica, para pacotes guardados em memória, qual saída (indicado por reg. B) será usada. *SAI* envia dados do reg. A, para a saída escolhida anteriormente por *SEL*.

As instruções *SEC e FCX* se referem ao Seletor de Conexão. O *SEC*, através do reg. B, indica qual conexão entre Porta de Entrada (*PUT* escolhe o *buffer*) – Seletor de Conexão – Porta de Saída (*SEC* escolhe a saída) deve ser feita. *FCX* é uma instrução muito simples, ela fecha conexão permanente entre uma Porta de Entrada e outra de saída. Ó Registrador de *Status*, neste caso, fica setado até que se desfaça a conexão. A entrada e a saída são escolhidas após uma primeira utilização de *PUT* e *SEC*. A desconexão é feita quando um *byte* do pacote é igual ao setado por *SETDSC*.

IV. TOPOLOGIAS DINÂMICAS

As figuras seguintes (Seletor de Conexão / *crossbar*) ilustram que, a possibilidade de se ter conexões diversas em momentos distintos, pode vir a ser muito interessante. Alguns exemplos serão citados para explicar melhor a importância da topologia dinâmica.

1) Um conjunto de computadores (Figura 2a e 2b) em uma rede local, forma um *cluster* [IEE 01], onde rodam aplicações diversas. A formação deste *cluster* varia de acordo com cada aplicação, em alguns casos o computador 1 precisa se conectar com o computador 2 e o computador 3 precisa se conectar com o computador 5. Estes pares de computadores, neste dado momento, rodam aplicações paralelas distintas. O computador 4 está conectado com outra rede de computadores, rodando determinada aplicação que, mais tarde irá compartilhar dados com o computador 1 em aplicações paralelas. A rede 6 e a rede 7 utilizam o barramento (pacote em memória) para trocar informações.

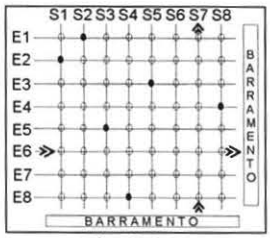

Fig. 2 (a)

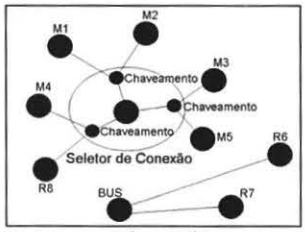

Fig. 2 (b)

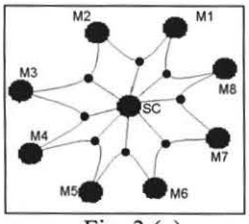

Fig. 3 (a)

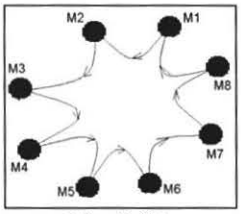

Fig. 3 (b)

2) Inicialmente a interface funciona como a topologia estrela (Figura 3a), todos conectados a ela. De acordo com as necessidades de comunicação, topologias irregulares ou regulares poderão ser criadas. A lógica de conexões, onde o micro da esquerda envia dados para o micro da direita, forma uma comunicação anti-horária e por conseqüência em anel unidirecional (Figura 3b).

Estas características se devem ao fato de que a arquitetura do processador possui Portas de I/O e Seletor de Conexão, dedicados para trabalhar e suportar qualquer tipo de topologia necessária. O conjunto de instruções também é muito importante, já que o controle sobre as portas e o seletor são feitos através delas.

V. SUPORTE A MULTI-PROTOCOLO

Outra característica importante é a capacidade de funcionar com qualquer tipo de protocolo. Esta facilidade se deve ao conjunto de instruções e a microarquitetura. Através deste processador é possível implementar qualquer tipo de programa, que responda ao tipo de protocolo ou padrão de comunicação. Se este processador for utilizado para rotear pacotes pela internet, o protocolo implementado poderá ser o IP (*Internet Protocol*) [COM 95].

Nos exemplos anteriores (IV. 1 / IV. 2), sobre as topologias dinâmicas, o protocolo que poderia ser usado, é o chamado protocolo leve. Este protocolo mais enxuto e simples do que o IP, por exemplo, seria responsável pela comunicação entre os computadores, que formam o *cluster* [BAR 00] [IEE 01]. No caso da comunicação com as redes locais, também conectadas ao processador, a comunicação se daria, por exemplo, pelo protocolo IP.

O importante é que, da mesma forma que várias topologias podem ser criadas, diversos protocolos poderão ser utilizados para garantir a comunicação entre os diversos nós que constituírem a rede e que estiverem conectados através deste processador. O que possibilita então, o suporte a vários protocolos é o conjunto de instruções dedicados à arquitetura de redes de comunicações.

VI. PORTAS DE ENTRADA RECONFIGURÁVEIS

Como mencionado no item II, uma das principais características deste processador é a possibilidade de reconfiguração dos *buffers* de entrada. O funcionamento acontece da seguinte forma: Existem fisicamente 8 entradas e, portanto 8 *buffers* temporários. Quando um pacote chega por uma destas entradas, o processador analisa o cabeçalho e tenta redirecionar para a saída desejada, caso esta, esteja livre (*cut and through*). Caso a saída esteja sendo utilizada, o processador guarda o pacote em memória ou libera o pacote para ser armazenamento em *buffers* permanentes da Porta de Entrada. Uma vez o pacote liberado, é feita a verificação do seu tamanho (reconfiguração – análise feita pelo hardware) para alocação nos *buffers* permanentes. O

posicionamento da informação de tamanho, varia de protocolo para protocolo e por isto deve ser usada anteriormente a instrução *SET C,DE*, que irá configurar o *hardware* (ou registradores de cada *buffer* temporário) com o local exato onde encontrar a informação do tamanho do pacote que será recebido. Cada *buffer* permanente possui um pequeno registrador que guarda a informação da última Porta de Entrada, de onde veio o pacote. Esta informação é muito útil quando não se encontra o endereço de destino na tabela de roteamento e é necessário reenviar o pacote para sua origem.

Quando a instrução *SEC* é utilizada, é feita uma conexão entre um *buffer* de entrada e o Seletor de Conexão. No caso da instrução *FCX*, que é o fechamento permanente de conexão, utilizamos os *buffers* temporários e o Seletor de Conexão, liberando assim, *buffers* permanentes para receber pacotes de qualquer outra entrada.

A técnica *stored and forward* é usada quando há necessidade de guardar o pacote na memória ou nos *buffers* permanentes. A técnica *cut and through* é usada pelo processador, que tenta enviar o pacote sem precisar guardá-lo (pacotes em *buffers* temporários).

Existe a possibilidade dos *buffers* usarem DMA para acessar a memória, liberando o processador para trabalhar em paralelo com as portas de entrada.

VII. COMPARAÇÕES COM OUTROS PROCESSADORES

Existem duas classes de arquiteturas de processadores que merecem atenção especial. Os processadores de propósito geral (GPPs – *General-Purpose Processor*) e os processadores de aplicação específica (ASPs – *Application Specific Processor*).

A.1 Processadores de Propósito Geral (GPPs)

Os processadores de propósito geral foram desenvolvidos com o intuito de poder resolver qualquer tipo de problema proposto. Nestes tipos de processadores é possível executar um grande conjunto de programas de aplicações distintas. O problema deste tipo de processador é que sua complexidade é alta. Dessa forma, um problema computacional simples pode requerer várias instruções para ser solucionado, o que poderia acarretar baixo desempenho [LUC 99] de execução.

Um problema claro deste tipo de processador, em comparação com o que estamos propondo, é a falta de portas de entrada e saída específicas para redes de comunicação. Esta deficiência pode ser vista da seguinte forma: Todo o pacote que chega em um GPP deve ser imediatamente guardado em memória, não existem *buffers* capazes de armazenar os pacotes. Estes *buffers* economizam tempo, já que o processo de guardar em memória pode ser eliminado. Com utilização dos *buffers* o pacote poderá ser redirecionado, através do Seletor de Conexão, para a saída, com o mínimo de intervenção do processador.

A.2 Processadores de Rede Comerciais (ASPs)

Com o avanço das redes de comunicações, o roteador passou a ser o gargalo de desempenho. As altas taxas de transmissão e o grande número de pacotes transmitidos fizeram com que novas tecnologias fossem projetadas e o roteador evoluiu com o desenvolvimento dos processadores de rede.

Os processadores de rede da Intel e da Motorola são: Intel *IXP 1200 Network Processor* [INT 00] e Motorola *MPC860* [MOT 99]. A Tabela 2 faz uma breve comparação entre estes processadores e o proposto neste artigo.

TABELA 2
Comparação entre processadores

	Processador proposto	**IXP 1200 Intel**	**Motorola MPC860**
Processador	CISC	RISC	RISC
Barramento de dados	8 bits	32 bits / 64 bits	8, 16 e 32 bits
Barramento de endereço	24 bits	32 bits / 64 bits	32 bits
Portas seriais de comunicação	8 Entradas / 8 Saídas	1 Entrada / 1 Saída	6 Entradas / 6 Saídas
Interface de comunicação	Sim	Sim	Sim
Possui cache de instrução	Não	Sim	Sim
Possui cache de dados	Não	Sim	Sim
Memória interna	Sim	Sim	Sim
Possui buffers reconfiguráveis	Sim	Não	Não
Suporta várias topologias	Sim	Não	Não
Suporta vários protocolos	Sim	Sim	Sim

Esta tabela mostra, que o número de portas de comunicação, *buffers* reconfiguráveis e Seletor de Conexão, que dão suporte à topologia dinâmica, são características únicas do processador proposto. Apesar de todos serem multi-protocolo, as empresas costumam deixar os processadores pré-programados, sem possibilidade de implementar novos protocolos pelo usuário, limitando a capacidade do processador de rede. A tecnologia RISC (*Reduced Instruction Set Computing*) possibilita maior velocidade de processamento das instruções e com um tamanho de palavra de dados maior, a velocidade também aumenta, já que quanto maior a palavra, maior a capacidade de guardar informação. Com base nestes dados, um novo processador está sendo desenvolvido por nós, com tecnologia RISC e palavra de dados maior, com o objetivo de aumentar o desempenho. Este processador RISC, que terá todas as características, que hoje estão validadas para o processador CISC (*Complex Instruction Set Computing*)

apresentado, será prototipado através de VHDL (*VHSIC Hardware Description Language*) e FPGA (*Field Programmable Gate Array*) [WAG 98] [XIL 98].

VIII. RESULTADOS

A validação da arquitetura interna foi realizada usando simulação funcional do processador de rede.

Os resultados apresentados são referentes a este tipo de simulação, testando principalmente a microarquitetura do processador, o conjunto de instruções do processador, e o suporte a topologia dinâmica e multi-protocolo.

A. Simulador da arquitetura do processador

Como forma de validar a arquitetura e conjunto de instruções proposto foi implementado um simulador capaz de mostrar através de sua interface gráfica o comportamento dos principais blocos da arquitetura e das instruções. Uma figura irá ilustrar a interface deste simulador: Figura 4 – Montador (*Assembler*).

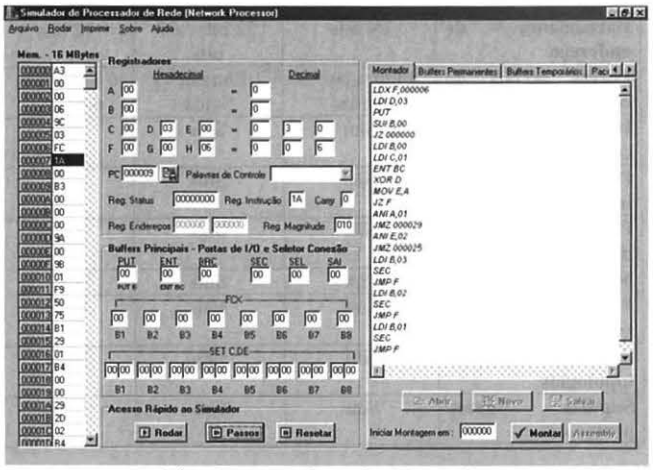

Fig. 4 – Interface do Montador

O simulador possui seis janelas gráficas para auxílio dos testes e validações do comportamento do processador de rede proposto. As outras cinco interfaces são: Confecção do pacote a ser enviado para o processador, Porta de Entrada (*buffers* permanentes), Porta de Entrada (*buffers* temporários), interface do Seletor de Conexão e interface de DMA. Estas cinco outras interfaces gráficas não serão mostradas.

Através deste simulador, foi possível escrever programas, implementar pacotes, trabalhar com os diversos *buffers* e registradores do processador e por fim, estudar o comportamento do mesmo, através dos pacotes recebidos, processados e roteados. A versão de teste pode ser obtida no endereço www.inf.pucminas.br/projetos/pad-r.

B. Validação da arquitetura e do conjunto de instruções

Durante alguns meses foi estudado qual seria a forma mais interessante de se otimizar o processamento de pacotes da camada de rede. A primeira idéia foi o desenvolvimento de um processador dedicado [FRE 00], capaz de se comportar como um roteador. Algumas características deste "roteador", que o tornariam multi-protocolo e com capacidade de suportar várias topologias, surgiram da idéia de incorporar às portas de comunicação, um seletor ou chave *crossbar* configurável, possibilitando um grande número de combinações de conexões entre entrada e saída, tal qual a matriz interna de um *switch*. A linha de pesquisa, Computação Reconfigurável, foi responsável pela idéia de criação dos *buffers* reconfiguráveis da porta de entrada, que alocam os pacotes de acordo com o tamanho. Dessa forma, foi possível projetar uma microarquitetura, dedicada e otimizada para comunicações de dados. Dois conjuntos de instrução foram propostos. Simulações foram realizadas e um terceiro conjunto foi projetado para atender às necessidades da arquitetura e das operações freqüentes dos protocolos. Os detalhes de microarquitetura, tal como sinais de comando, e lógica de *hardware* serão validados através da simulação usando VHDL (linguagem de descrição de *hardware*) [WAG 98] [XIL 98].

A arquitetura e o conjunto de instruções do processador dedicado, apresentados nos itens II e III, são os resultados finais da etapa de projeto do processador, que possui como etapa de validação a simulação funcional. Desse modo, alguns resultados finais da validação já foram apresentados nos itens anteriores.

Nos itens II e III foram descritos os funcionamentos da arquitetura interna e do conjunto de instruções. A Figura 1 e a Tabela 1 exemplificam estes resultados que são importantes para o futuro do projeto.

C. Exemplos de Simulação

Os códigos, que serão descritos a seguir, se basearam em duas topologias (Figuras 5 e 6), onde cada nó representa um processador (processador proposto). Foram implementados dois algoritmos de roteamento e não foi levada em consideração a possibilidade de conflito para saída de pacotes. Por isso o reg. de *status* não foi verificado em nenhum dos algoritmos.

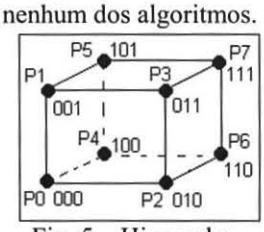

Fig. 5 – Hipercubo

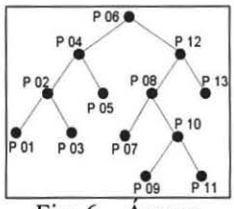

Fig. 6 – Árvore

C.1 Topologia baseada em Hipercubo

Neste caso temos um *cluster* [IEE 01] com topologia no formato de um hipercubo. Cada endereço do hipercubo tem apenas 1 bit de diferença para os endereços dos nós adjacentes. Este algoritmo foi feito para o processador P3.

Para esta topologia foi desenvolvido o seguinte código:

```
LDX F,000006    //Endereço de PUT
LDI D,03        //Processador = P3
PUT             //Busca buffer permanente
SUI B,00        //B recebe valor do buffer.
JZ 000000       //Se for zero, não existe pacote
LDI B,00        //Posição que se encontra endereço.
LDI C,05        //Posição que se encontra endereço.
ENT BC          //Busca o endereço no buffer
XOR D           //Operação XOR (A ⊕ D)
MOV E,A         //Move dado de reg. A para reg. E
JZ F            //Se chegou ao destino, volta a PUT
ANI A,01        //Senão verifica se é saída 01
JMZ 000029      //Se for maior é verdadeiro e jump S1
ANI E,02        //Verifica se é saída 02
JMZ 000025      //Se for maior é verdadeiro e jump S2
LDI B,03        //Saída 3, carregada em B
SEC             //Redireciona, buffer entrada → PS 03
JMP F           //Voltar a PUT
LDI B,02        //Saída 2, carregada em B
SEC             //Redireciona, buffer entrada → PS 02
JMP F           //Voltar a PUT
LDI B,01        //Saída 1, carregada em B
SEC             //Redireciona, buffer entrada → PS 01
JMP F           //Voltar a PUT
```

C.2 Topologia baseada em Árvore

Um outro *cluster* [IEE 01] de processadores usa uma topologia de árvore. O endereço de cada nó está no desenho. À esquerda de cada nó, sempre existe um endereço de menor valor. À direita, sempre um endereço de maior valor. Este algoritmo foi feito para o processador P 06.

Para esta topologia foi desenvolvido o seguinte código:

```
LDI D,06        //Processador 06 – P 06
PUT             //Busca aleatória do buffer de entrada
SUI B,00        //B recebe valor do buffer.
JZ 000025       //Se for zero, não existe pacote.
LDI B,00        //Posição que se encontra endereço.
LDI C,05        //Posição que se encontra endereço.
ENT BC          //Busca o endereço no buffer
SUB D           //Verifica se é menor ou maior do que 06.
JMI 00001^A     //Jump se maior ou igual
LDI B,04        //Senão, a saída é 04
SEC             //Redireciona, buffer entrada → PS 04
JMP 000002      //Volta para PUT
JZ 000025       //Se for 0, já chegou ao destino
LDI B,05        //Se for maior, a saída é 05
SEC             //Redireciona, buffer entrada → PS 05
JMP 000002      //Volta para PUT
HLT             //Fim do programa
```

Obs.: P 06 = local, P 04 = PS 04, P 12 = PS 05

C.3 Valores quantitativos

Os gráficos das Figuras 7 e 8 ilustram a quantidade de acesso à memória e a quantidade de instruções de entrada e saída usadas pelo processador proposto e por um processador CISC de propósito geral de 8 bits, para execução dos dois algoritmos descritos. Os valores dos gráficos são referentes a um pacote de apenas 10 bytes. Independentemente do tamanho do pacote, a simulação ocorreu tendo o endereço de destino na posição 5.

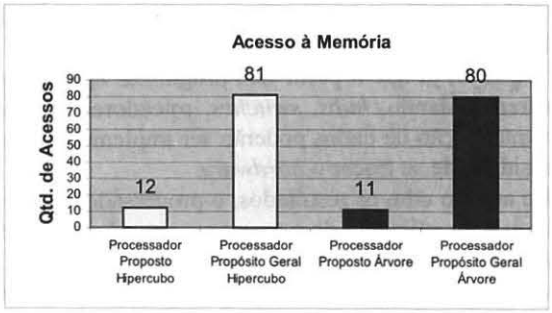

Fig. 7 – Acesso à Memória

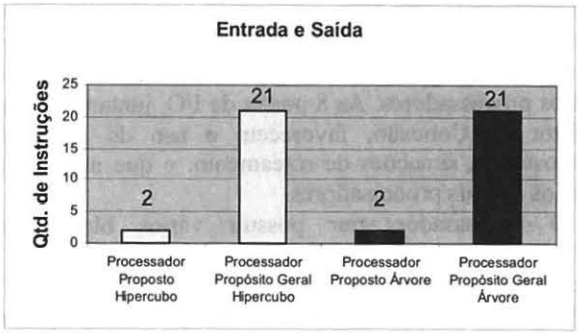

Fig. 8 – Entrada e Saída

Se o pacote fosse de 1500 *bytes*, os valores do processador proposto, permaneceriam em 12 e 11 para acesso à memória e 2 e 2 para quantidade de instruções de I/O. Estes valores são independentes do tamanho do pacote. Para o processador de propósito geral, os valores subiriam para 1061 e 1060, para acesso à memória, e 511 e 511 para quantidade de instruções de I/O. Estes valores variam de acordo com o tamanho do pacote.

É importante ressaltar que as instruções de entrada e saída do processador proposto acessam *buffers* internos, ao contrário do processador de propósito geral, que acessa *buffers* externos. O processador proposto utiliza apenas uma instrução para ler o endereço de destino do *buffer* interno, enquanto o GPP precisa ler *byte* a *byte* do *buffer* externo e guardar em memória, até que a posição do pacote, que guarda o destino, seja lida. Apenas uma instrução é utilizada para redirecionar todo conteúdo do *buffer* interno para a saída desejada, enquanto que o GPP precisa ler *byte* a *byte* do *buffer* externo e escrever *byte* a *byte* na saída

desejada. O processo usado pelo GPP é lento, já que usa *loops* com instruções de acesso à memória e de leitura e escrita nas portas de I/O. O processador proposto é mais rápido, lê-se diretamente da posição desejada do *buffer* e redireciona todo o conteúdo para a saída sem precisar ler ou escrever nas portas de I/O e/ou na memória.

IX. CONCLUSÃO

A arquitetura do processador, descrita neste artigo, tem por objetivo, demonstrar um conceito mais eficiente e eficaz de se processar dados em uma rede de comunicação.

Este processador amplia as possibilidades de utilização de *gateways*, já que a partir dos programas aplicativos que estiverem rodando, *hubs*, *switches*, roteadores e interfaces de comunicação de dados poderão ser implementadas sem a necessidade de se trocar o *hardware*.

De acordo com os resultados, o processador proposto se mostrou mais eficiente do que um processador de propósito geral, com menos instruções para acesso a memória e as portas de I/O. Sua microarquitetura e conjunto de instruções dedicadas possibilitam maior velocidade e melhor desempenho de processamento. Com relação aos processadores de rede comerciais, a microarquitetura proposta contribui com novidades, como o Seletor de Conexão e as portas reconfiguráveis não encontrados em outros processadores. As 8 portas de I/O, juntamente com o Seletor de Conexão, favorecem o uso do processador proposto em situações de roteamento, o que não acontece com os demais processadores.

O processador, por possuir vários blocos como memória, portas e interface de comunicação, pode ser classificado também como um *System on Chip (SOC)* [TRI 01]. Este tipo de estrutura da microarquitetura proporciona condições para o aumento de velocidade e de processamento, já que a proximidade dos blocos lógicos se resume ao interior da pastilha do processador.

Este trabalho possui contribuições importantes, já que os processadores de rede fazem parte de pesquisas recentes na área de arquitetura de computadores. Seus resultados irão contribuir para o aumento de conhecimento técnico específico e novas metodologias de projeto, trazendo impactos diretos para a instituição e empresas que desejam desenvolver tais processadores.

Trabalhos futuros: i) projeto da versão RISC, ii) prototipação utilizando VHDL e FPGA's, iii) simulação em redes de comunicações de dados [WAG 98] [XIL 98].

X. AGRADECIMENTOS

Agradecemos ao Datapuc – Processamento de Dados, ao Instituto de Informática e aos colegas do LSDC, pela colaboração e incentivo, que possibilitou o progresso da pesquisa e nossa participação no WSCAD'01.

XI. REFERÊNCIAS

[BAR 00] BARBOZA, Alexandre Ignácio; KOFUJI, Sérgio Takeo "Projeto Mercúrio – Interface de Comunicação de Alta Velocidade", I Workshop de Sistemas Computacionais de Alto Desempenho, WSCAD'2000, São Pedro – SP, pp.9-13

[CAM 00] CÂMARA, Daniel; LOUREIRO, Antônio A. F. "Um Novo Protocolo de Roteamento para Redes Móveis Ad hoc", 18º Simpósio de Redes de Computadores, Belo Horizonte, 2000, pp.259-274

[COM 95] COMER, Douglas E., "Internetworking with TCP/IP Principles, Protocols and Architecture Volume 1", 3ª Edition, Ed. Pearson, 1995

[FRE 00] FREITAS, Henrique C. de; MARTINS, Carlos Augusto P. S., "Processador Dedicado para Roteamento em Sistemas de Comunicação de Dados", I Congresso de Lógica Aplicada à Tecnologia, LAPTEC'2000, São Paulo, pp.717-721, (Iniciação Científica)

[FRE 00] FREITAS, Henrique C. de; MARTINS, Carlos Augusto P. S., "Projeto de Processador com Microarquitetura Dedicada para Roteamento em Sistemas de Comunicação de Dados", I Workshop de Sistemas Computacionais de Alto Desempenho, WSCAD'2000, São Pedro - SP, pp.63, (Iniciação Científica)

[IEE 01] IEEE CS Task Force on Cluster Computing (TFCC) http://www.ece.arizona.edu/~hpdc/tfcc-network/

[INT 00] INTEL, "IXP 1200 - Network Processor", Datasheet, May 2000

[LUC 99] LUCENT Technologies, Building for Next Generation Network Processors, September 1999, http://www.lucent.com/micro/netcom/platform/npu.html#product

[LUC 99] LUCENT Technologies, The Challenge for Next Generation Network Processors, September 10, 1999, http://www.lucent.com/micro/netcom/platform/npu.html#product

[MOT 99] MOTOROLA's MPC860 PowerQUICC Processor, Hardware Specifications, 1999

[TRI 01] TRISCEND Corporation, http://www.triscend.com

[WAG 98] WAGNER, F.R., PORTO I.J., WEBER R.F., WEBER T.S., "Métodos de Validação de Sistemas Digitais", VI Escola de Computação, SBC, Campinas, 1998

[XIL 98] XILINX Development System, "Synthesis and Simulation Design Guide - Designing FPGAs with HDL", 1998