*** RELATÓRIO TÉNICO ***

AVALIAÇÃO DE UMA ALTERNATIVA DE IMPLEMENTAÇÃO PARA UM MICROPROCESSADOR SPARC

Gabriel Pereira da Silva
Júlio Salek Aude

NCE 19/91
Novembro/91

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

Este artigo foi apresentado no VI Congresso da Sociedade Brasileira de Microeletrônica (VI SBMICRO), realizado em Belo Horizonte/MG, de 15 a 19 de julho de 1991, pp. 312-324.

UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
NÚCLEO DE COMPUTAÇÃO ELETRÔNICA

AVALIAÇÃO DE ALTERNATIVAS DE IMPLEMENTAÇÃO PAR A UM MICROPROCESSADOR SPARC.

Resumo

Este trabalho estuda algumas opções para a implementação de uma arquitetura compatível com a definição do SPARC. Por ser uma arquitetura aberta, o SPARC permite a elaboração de implementações distintas, sem perda de compatibilidade binária. Este artigo avalia o desempenho do processador em várias configurações, os resultados destas simulações são então mostrados e avaliados, e uma proposta de implementação para a arquitetura é apresentada

EVALUATION OF IMPLEMENTATION ALTERNATIVES FOR A SPARC MICROPROCESSOR.

Abstract

This work evaluates some options for the implementation of a SPARC compatible architecture. Because SPARC is an open architecture, it is possible to have distinct implementations, while keeping binary compatibility. This paper evaluates processors throughput in some configurations, simulation results are then shown and analyzed, and an implementation for the architecture is proposed.

AVALIAÇÃO DE ALTERNATIVAS DE IMPLEMENTAÇÃO PARA UM MICROPROCESSADOR SPARC[1]

Autores: Gabriel Pereira da Silva[2]

Julio Salek Aude[3]

Afiliação: NCE/UFRJ, Cx. Postal 2324, CEP 20001,

Rio de Janeiro, RJ. Tel: 55-21-5983212 Fax: 55-21-2708554

RESUMO:

O NCE/UFRJ desenvolve atualmente um sistema multiprocessador com memória global compartilhada, o projeto MULTIPLUS. Sua proposta é obter um sistema paralelo de alto desempenho com o uso de microprocessadores RISC acoplados a unidades aritméticas de ponto flutuante. Um dos objetivos deste projeto é o desenvolvimento de um microprocessador RISC compatível com o microprocessador utilizado: o SPARC.

Este trabalho estuda algumas opções para implementação desta arquitetura compatível. Por ser uma arquitetura aberta, o SPARC permite a elaboração de implementações distintas, sem perder a compatibilidade binária. Este artigo avalia o desempenho do processador em várias configurações, os resultados destas simulações são apresentados e avaliados, e uma proposta de implementação para a arquitetura é apresentada.

ABSTRACT:

The NCE/UFRJ is currently developing a high performance shared memory multiprocessor system called MULTIPLUS. Its proposal is to obtain a parallel high performance system with the use of RISC microprocessors coupled to floating point units. One of the goals of this project is to develop a RISC microprocessor compatible with the chosen one: the SPARC.

This works evaluates some options for the implementation of this compatible architecture. Because SPARC is an open architecture, it is possible to have different implementations, while keeping binary compatibility. This article evaluates processors throughput in many configurations, simulation results are shown and analyzed, and an implementation for the architecture is proposed.

---

1 Este trabalho conta com o auxílio financeiro da FINEP e do CNPQ.

2 Eng. Eletrônico, UFRJ, 1983; M.Sc. Eng. de Sist. e Computação, COPPE/UFRJ, 1991.

3 Eng. Eletrônico, UFRJ, 1974; M.Sc. COPPE/UFRJ, 1978; Ph.D. Ciência da Computação, Manchester University, UK 1986.

1 - INTRODUÇÃO

Atualmente o NCE/UFRJ desenvolve um sistema multiprocessador de alto desempenho com memória global compartilhada, o projeto MULTIPLUS [AUD90]. Sua proposta é obter um sistema paralelo de alto desempenho com o uso de microprocessadores RISC acoplados a unidades aritméticas de ponto flutuante. Um dos objetivos deste projeto é o desenvolvimento de um microprocessador RISC compatível com o microprocessador comercial a ser utilizado na primeira fase do projeto.

Este trabalho estuda diversas opções para implementação de um microprocessador de 32 bits com filosofia RISC, compatível com o SPARC. A arquitetura SPARC [GAR88, SUN87] é uma arquitetura aberta, que permite a elaboração de implementações distintas, sem perda de compatibilidade binária. Este artigo faz a avaliação de desempenho do processador para determinar a relação custo benefício de cada uma delas.

O artigo está organizado nas seguintes seções: Na Seção 2 é apresentado o simulador e os parâmetros que vão ser avaliados na arquitetura. Na Seção seguinte são apresentados os programas de avaliação utilizados e feita uma análise dos resultados da simulação. Finalmente, na Seção 4 são apresentadas as conclusões, juntamente com uma proposta para a implementação da arquitetura SPARC no NCE/UFRJ e perspectivas de desdobramento deste trabalho.

2 - O SIMULADOR E OS PARÂMETROS SIMULADOS

O SIMUS [SIL90] é um simulador configurável dinamicamente, que simula o comportamento do SPARC a nível de "pipeline". Com o simulador, a arquitetura pode ser configurada com um tamanho variável de conjunto de registradores, com um barramento único ou separado para dados e instruções, e com um tamanho de cache interna também variável. Com isto, pretende-se determinar a melhor utilização da área disponível em silício. A seguir são apresentados os parâmetros de arquitetura que podem ser simulados:

2.1- Número de Janelas

O microprocessador SPARC possui os seus registradores organizados em forma de janelas. Para o programador existe um número ilimitado de janelas. A nível físico, a definição da arquitetura permite a existência desde 2 até 32 janelas. Cada vez que o programa faz uso de um número de janelas maior que o implementado, existe a necessidade de desvio para uma rotina de gerenciamento.

Na simulação realizada, estas rotinas de gerenciamento de janelas foram implementadas em "assembler" e o custo do salvamento/recuperação de janelas também é considerado na execução dos programas. O resultado final é um ambiente mais real, já que o custo de gerenciamento das janelas pode ser considerável em muitos casos. A estratégia de gerenciamento utilizada foi a de melhor custo/desempenho [TAM83], e consiste em mover apenas uma janela por vez, a cada ocorrência de uma exceção.

O uso de janelas de registradores internamente à pastilha diminui consideravelmente o trafégo no barramento [GAR88], com conseqente aumento no desempenho e diminuição da interferência com outros processadores. Entretanto, o conjunto de registradores é responsável por um gasto de área entre 25 e 40% da pastilha, dependendo

da tecnologia utilizada e da forma de projeto [QUA88,SEQ82,KAT83]. Um gasto de área excessivo pode comprometer a integração do projeto em determinadas tecnologias.

2.2- Largura e Tipo do Barramento

Uma das características importantes no desempenho de uma arquitetura é o tipo de barramento empregado. Para determinar o grau de influência nas aplicações, o SIMUS pode ser configurado para uso com barramento único ou distintos (Harvard) para dados e instruções. Além disto, permite que a largura destes barramentos seja de 32 ou 64 bits.

O uso de um barramento Harvard é importante para diminuir a interferência da busca de dados no fluxo de instruções. Normalmente as instruções de LOAD e STORE contribuem com mais de 20% do total de instruções executadas [NAM88a] em um programa típico escrito em "C". Se existem barramentos independentes, estas operações podem ser realizadas em paralelo, com consequente aumento de desempenho. A contrapartida é o gasto de área da periferia, que é de peso significativo na área total da pastilha. Em algumas implementações RISC [KAT83], este gasto chega a ser de 40%. Entretanto, é uma opção que tem sido utilizada com maior freqência nos processadores mais modernos, como o MC88100 e o i80860, devido à alta tecnologia de integração disponível e à necessidade de uma banda passante maior.

O uso de um barramento de 64 bits produz ganhos mais modestos, pois só é importante para busca de dados com 64 bits, já que apenas uma instrução é alimentada no pipeline a cada ciclo, e não há uso imediato para a banda passante disponível. Entretanto, é uma opção mais barata em termos de área que a utilização de barramentos separados para dados e instruções.

2.3- Cache de Desvio

O uso de cache interna é um dos meios mais eficientes para aumentar o desempenho do processador. Contudo, o gasto de área necessário para implementá-la dentro da pastilha é bastante grande. Várias implementações do SPARC optaram pela utilização de uma cache externa de dados e/ou instruções, com esquema de entrelaçamento, em que o endereço é fornecido em um ciclo e os dados lidos no seguinte. Isto dá tempo ao circuito de controle para verificar se houve acerto ou falha no acesso ao cache. Com isto consegue-se trabalhar sem estados de espera, apesar das altas frequências de "clock" envolvidas, na ordem de 40 Mhz para as pastilhas CMOS.

Porque então a simulação de uma cache interna à pastilha? O esquema que pretendemos simular é de uma cache de desvio [COR88], que é direcionado para arquiteturas com "pipeline" e permite a avaliação do desvio condicional em paralelo com a execução da instrução "atrasada". O simulador permite tamanhos de cache entre 8 e 128 entradas, com tamanho de cada linha variando entre 1 e 4 conjuntos. O modelo simulado considera o uso de uma memória externa com tempo de resposta que permita ao processador buscar um dado a cada ciclo. Esta aproximação é verdadeira se houver uma cache externa com alta taxa de acerto.

Na cache de desvio estão armazenados o endereço da instrução anterior ao desvio, a instrução de desvio e a instrução alvo, para onde pode ser transferido o controle. Se houver acerto na cache, a busca da instrução de desvio é substituída pela busca da

instrução "atrasada". No ciclo seguinte, os códigos de condição são avaliados em paralelo com a execução da instrução "atrasada". Assim, a execução do desvio deixa de ter custo um e passa a ter custo zero.

Na Figura 1 é apresentado um esquema de funcionamento da cache de desvio. Nas arquiteturas SPARC os desvios representam cerca de 20% das instruções executadas [NAM88a]. Com o uso da cache de desvio, espera-se obter um ganho entre 10 e 20% no desempenho das aplicações, podendo-se inclusive superar a barreira de um ciclo por instrução.

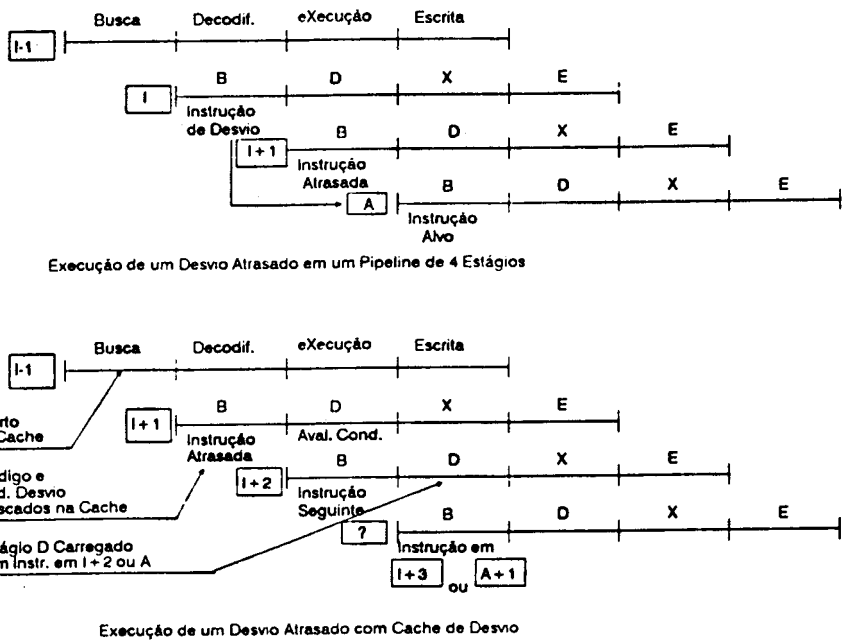

Figura 1 - Operação da Cache de Desvio

2.4 - Outras Considerações

Existem duas situações especiais que podem ocorrer durante a simulação: quando é feita uma chamada ao sistema e quando existe a necessidade de carga/salvamento de uma janela de registradores de/para a memória. No primeiro caso, não existe suporte para as chamadas de sistema, consequentemente não pode haver operações de E/S nos programas simulados. No segundo caso, a operação é feita por um programa carregado junto com o programa simulado.

Este programa exerce a função de um pequeno núcleo, e o tempo gasto neste núcleo é somado aos tempos totais da simulação do programa. O procedimento adotado neste núcleo é bem similar ao que uma rotina real executaria [KLE88]. Em nossa simulação não estão previstos os custos de troca de contexto, mas este gasto pode ser estimado a partir do número médio de janelas ativas e do tempo médio de execução de uma tarefa, este último específico para cada sistema.

3 - EXPERIÊNCIAS E ANÁLISE DOS RESULTADOS

3.1- Os Programas de Avaliação

Foram utilizados programas convencionais para avaliação das configurações da arquitetura. Estes programas foram escritos em linguagem "C" [HIN84,REI84] e alguns deles já foram utilizados anteriormente em outras avaliações [TAM81, TAM83]. Estes programas foram compilados em uma SPARCstation e os códigos objetos gerados foram transportados para um PC e adaptados para o simulador.

RESULTADOS DA SIMULAÇÃO
QUICKSORT

ESTATÍSTICAS DE BARRAMENTO		ESTATÍSTICAS DA CPU	
Acessos Leitura	= 198922	Tot. de Ciclos	= 209546
Acessos Escrita	= 5432	Tot. de Instruções	= 145299
Dados Usuário	= 26263	Ciclos/Instruções	= 1.44
Dados Supervisor	= 1080	Média Jan. Ativas	= 7.3
Instr. Usuário	= 174922	Maior Valor Jan.	= 19
Instr. Supervisor	= 2089	"Interlocks"	= 15923

Tabela 1 - Resultados da Simulação Quicksort

O programa Quicksort ordena um vetor preenchido pseudo-aleatoriamente, com chamadas recursivas de melhor caso O(NlogN). A profundidade de chamada de rotinas alcançada foi de 19 com a massa de dados utilizada, ou seja, um vetor de 4 Kbytes idêntico para todas as simulações. A Figura 2 mostra o histograma de instruções executadas para o caso em que há um número de registradores suficiente para que não haja necessidade do uso das rotinas de gerenciamento. A Tabela 1 mostra os resultados da simulação de uma arquitetura "padrão", com oito janelas, sem cache de desvio e com barramento compartilhado de dados instruções.

RESULTADOS DA SIMULAÇÃO
SIEVE

ESTATÍSTICAS DE BARRAMENTO		ESTATÍSTICAS DA CPU	
Acessos Leitura	= 190425	Tot. de Ciclos	= 236806
Acessos Escrita	= 23190	Tot. de Instruções	= 170695
Dados Usuário	= 31381	Ciclos/Instruções	= 1.39
Dados Supervisor	= 0	Média Jan. Ativas	= 1.0
Instr. Usuário	= 182234	Maior Valor Jan.	= 0
Instr. Supervisor	= 0	"Interlocks"	= 0

Tabela 2 - Resultados da Simulação Crivo de Eratostenes

O programa Sieve gera os primeiros 1899 números primos manipulando um vetor de 16 Kbytes. Primeiramente o vetor é inicializado com 1's, em seguida cada número primo encontrado tem seus múltiplos marcados com 0. Ao final do programa o vetor contém apenas os números primos marcados com 1. A Figura 3 mostra o histograma de instruções; a Tabela 2 os dados obtidos na simulação de uma arquitetura padrão idêntica à descrita anteriormente.

RESULTADOS DA SIMULAÇÃO
FIBONACCI

ESTATÍSTICAS DE BARRAMENTO		ESTATÍSTICAS DA CPU	
Acessos Leitura	= 75496	Tot. de Ciclos	= 81992
Acessos Escrita	= 4176	Tot. de Instruções	= 61248
Dados Usuário	= 0	Ciclos/Instruções	= 1.34
Dados Supervisor	= 8352	Média Jan. Ativas	= 13.4
Instr. Usuário	= 55191	Maior Valor Jan.	= 19
Instr. Supervisor	= 16129	"Interlocks"	= 0

Tabela 3 - Resultados da Simulação do Fibonacci

O programa Fibonacci calcula o 18º termo da série de Fibonacci através de uma série de chamadas recursivas. Cada rotina chamada tem um peso computacional muito leve, e faz apenas a soma dos resultados devolvidos pelas duas recursões seguintes, além de uma comparação para verificar se a recursão já chegou ao primeiro termo da série.

Pelo histograma da Figura 4, podemos verificar que não são executadas instruções de LOAD ou STORE. Assim sendo, este programa é bastante sujeito às influências do número de janelas presentes na arquitetura, porque existe a necessidade de se efetuar várias vezes a busca e o salvamento de janelas. Tais rotinas possuem um peso computacional maior, já que utilizam um grande número de instruções de LOAD e STORE.

O programa Dhrystone [REI84] é um programa de avaliação sintético que possui um conjunto de instruções típico de uma aplicação inteira. O programa contém a seguinte distribuição de sentenças considerada representativa de um programa de alto nível em linguagem C :

atribuições	53%
comandos de controle de fluxo	32%
rotinas, chamadas de funções	15%

RESULTADOS DA SIMULAÇÃO
DHRYSTONE

ESTATÍSTICAS DE BARRAMENTO		ESTATÍSTICAS DA CPU	
Acessos Leitura	= 199041	Tot. de Ciclos	= 235537
Acessos Escrita	= 18248	Tot. de Instruções	= 151792
DadosUsuário	= 52429	Ciclos/Instruções	= 1.55
Dados Supervisor	= 0	Média Jan. Ativas	= 3.5
Instr. Usuário	= 164853	Maior Valor Jan.	= 5
Instr. Supervisor	= 7	"Interlocks"	= 4867

Tabela 4 - Resultados da Simulação Dhrystone

O programa não calcula nada significativo, mas é sintática e semanticamente correto. A Figura 5 apresenta um histograma das instruções executadas para uma arquitetura com número infinito de janelas implementadas e a Tabela 4 mostra os dados da simulação

de uma arquitetura "padrão" com 8 janelas. A distribuição das instruções em baixo nível fica alterada, pois as sentenças de atribuição em "C" resultam em um número maior de instruções em assembler.

RESULTADOS DA SIMULAÇÃO
TORRES DE HANOI

ESTATÍSTICAS DE BARRAMENTO		ESTATÍSTICAS DA CPU	
Acessos Leitura	= 89689	Tot. de Ciclos	= 91453
Acessos Escrita	= 1134	Tot. de Instruções	= 81781
Dados Usuário	= 0	Ciclos/Instruções	= 1.12
Dados Supervisor	= 2268	Média Jan. Ativas	= 12.6
Instr. Usuário	= 84198	Maior Valor Jan.	= 14
Instr. Supervisor	= 4357	"Interlocks"	= 0

Tabela 5 - Resultados da Simulação do Torres de Hanoi

O programa Towers implementa o algoritmo de torres de Hanoi com 12 discos e foi utilizado nas simulações de dimensionamento do tamanho do conjunto de registradores. É um programa também recursivo e com um peso computacional de cada rotina um pouco maior que o Fibonacci. Não utiliza instruções de LOAD ou STORE e a profundidade de chamadas de rotinas alcançada é 20. O programa possui uma árvore de chamadas diferente da do Quicksort e do Fibonacci, sendo por isto utilizado. Ao contrário dos demais programas, **não** foi codificado em "C", sendo escrito em "assembler". O histograma de instruções executadas pode ser visto na Figura 6 e outros dados da simulação da arquitetura padrão são encontrados na Tabela 5.

Todos os programas escritos em "C" foram compilados com nível 2 de otimização [MUC88]. Em alguns casos foi observada a eliminação de chamadas recursivas ao final da rotina, como no Quicksort, resultando em um menor número de sequências de instruções de SAVE e RESTORE. Este tipo de otimização foi utilizado porque nas arquiteturas RISC o compilador é uma extensão da arquitetura e um recurso que não pode ser ignorado. O número total de ciclos apresentados nas tabelas considera os ciclos perdidos nas rotinas de salvamento/restauração de janelas de/para a memória.

3.2- Os Resultados das Simulações

Os resultados da simulação com variação de parâmetros da arquitetura são mostrados nas Figuras 7 a 12. São avaliadas a variação do tamanho do conjunto de registradores (número de janelas implementadas), o tamanho da cache de desvio (número de entradas) e o tipo de organização do barramento.

As Figuras 7 e 8 mostram os efeitos da variação do tamanho do conjunto de registradores nos programas Fibonacci, Quicksort e Towers. Estes programas foram escolhidos por fazerem uso significativo de chamadas de rotinas. A "taxa de falhas" da Figura 7 expressa o percentual de instruções de SAVE ou RESTORE que não encontraram um "frame" de janela disponível no conjunto de registradores. Como consequência há necessidade de desvio para a rotina de gerenciamento de janela e diminuição no desempenho final do programa. Esta perda vai depender do peso das instruções de SAVE e RESTORE no total de instruções executadas. A Figura 8 mostra o total de ciclos gastos para executar cada

programa considerando-se a perda nas rotinas de gerenciamento de janelas. Assim, podemos ver que o programa Quicksort é menos sensível a esta variação de parâmetro do que o programa Fibonacci, apesar de apresentarem a mesma curva de "taxa de falhas".

As Figuras 9 e 10 ilustram a influência do tamanho da cache de desvio na execução do programa Dhrystone, com curvas para a cache configurada com associatividade quatro , dois e um (mapeamento direto). A Figura 9 apresenta a taxa de acertos da cache de desvio. Esta taxa expressa o percentual de instruções de desvio encontradas na cache de desvio. A influência desta taxa de acerto no desempenho dos programas é mostrado na Figura 10. O desempenho final em cada caso é ligado principalmente ao total de instruções de desvio condicional existentes no programa, estáticamente, e ao seu percentual em relação ao total de instruções executadas, dinâmicamente.

A Figura 11 faz uma comparação entre a utilização de um barramento único para dados e instruções e o uso de barramentos independentes (Harvard). Ambas as configurações não possuíam cache de desvio e tinham um conjunto de registradores dimensionado para 8 janelas. Foram simulados quatro programas, Fibonacci, Quicksort, Sieve e Dhrystone. Os programas que apresentaram maior ganho foram aqueles com maior percentual de "loads" e "stores" no histograma de instruções.

A Figura 12 faz uma comparação entre a arquitetura "padrão" descrita anteriormente e uma arquitetura "especial", que agrega facilidades como barramento Harvard, cache de desvio de 64 posições e 8 janelas de registradores implementadas. Nesta situação o ganho é significativo em todos os programas, e a taxa de execução chega a ser menor que um ciclo por instrução.

O simulador permite também a configuração da arquitetura com um barramento de 64 bits. Contudo, na simulação dos programas descritos não houve ganhos significativos. O único programa que teve algum ganho (11%) foi o Fibonacci, mas com a arquitetura configurada para apenas 4 janelas. Este ganho só ocorreu por causa do alto número de operações de salvamento/recuperação de janelas que existe nesta situação. As rotinas de gerenciamento de janela utilizam instruções de LOAD e STORE com operandos de 64 bits, justificando assim o ganho obtido.

3.3 Considerações de Custo

Implementações anteriores em CMOS apresentam um custo de 25% a 30% para o conjunto de registradores. A interface de entrada e saída ocupa tipicamente de 30% a 40% da área de pastilha. Uma cache de desvio com 16 posições, equivale aproximadamente a 2 janelas de registradores. A partir daí as seguintes figuras de custo podem ser obtidas: uma janela equivale a cerca de 3% a 4% de aumento na área; cada 16 posições na cache equivalem a 6% ou 8% de aumento de área; o uso de um barramento Harvard significa um aumento de cerca de 25% a 30% na área total. O uso de um barramento de 64 bits implica em um gasto adicinal entre 12% e 15% no total da área.

Confirmando-se estas relações, podemos determinar a validade da implementação, por exemplo, da arquitetura especial sugerida, com barramento Harvard e cache de desvio de 64 posições. Nesta situação temos um aumento na área, em relação à

arquitetura "padrão", entre 50% e 60%. Se fizermos a média dos ganhos obtidos na execução dos programas de teste, obtemos o valor de 38%, o que é um valor bastante significativo.

4 - CONCLUSÕES

A partir dos resultados das simulações realizadas, podemos determinar as seguintes recomendações:

- O uso de um conjunto de registradores com tamanho entre 6 e 10 é o ideal. Abaixo de 6 janelas a taxa de falhas é muito grande, comprometendo o desempenho dos programas. A partir de 10 janelas a curva de resposta é plana, não havendo nenhum ganho adicional. O restante da área que for disponível pode ser utilizada para outras facilidades, como p. ex., o uso de uma cache de desvio interna.

- Os resultados indicam que caches de tamanhos pequenos, com apenas 8 a 64 posições, são bastante efetivas. O aumento médio de área fica entre 4% e 28%, com aumento no desempenho entre 5% e 18%, respectivamente. O número de conjuntos (grau de associatividade) da cache é de pouca influência na taxa de acerto, como também mostrado por Lee & Smith [LEE84]. A utilização de uma cache com menor associatividade permite a elaboração de circuitos de controle mais simples.

- Os resultados sugerem que o uso de um barramento Harvard resulta em grande melhoria, com aumentos de desempenho entre 6% e 29%, mas os custos envolvidos são os mais altos apresentados (25% a 30%). Os programas que oferecem maior ganho são aqueles com maior percentual de acesso a dados. Os ganhos obtidos se encontram dentro da expectativa inicial, que considera o ganho proporcional ao número de instruções de LOAD ou STORE.

- A associação de uma cache de desvio a uma arquitetura com barramento Havard pode levar a taxas de execução tão baixas quanto um ciclo por instrução. Esta opção de configuração oferece uma boa relação de custo/desempenho, com aumento de área em torno de 55%, mas com taxas de aumento de desempenho na arquitetura entre 16 e 46%. Estes valores são maiores que os esperados inicialmente, como observado no Seção 2.

Como política de implementação, sugerimos que a área da pastilha seja primeiramente dedicada à implementação de registradores, até um máximo de 10 janelas. A partir daí sugerimos que a área disponível seja utilizada para implementação de uma cache de desvio até o valor máximo de 64 posições com mapeamento direto e, finalmente, que seja implementado um barramento do tipo Harvard.

Foi também observada a ocorrência de muitos interlocks durante a execução dos programas Dhrystone (3.3%) e Quicksort (9.2%). A realização de algum tipo de otimização, a nível de compilador (p. ex., despacho de instruções), ou de arquitetura (p. ex, bypass interno de dados), deve ser avaliada, já que podem ser relevantes no desempenho final da arquitetura.

A implementação do microprocessador do NCE/UFRJ deverá ser feita em tecnologia CMOS de 1.2 micra, parte com tecnologia de projeto em células padronizadas, parte totalmente personalizada. Alguns parâmetros que estavam em aberto na definição da

arquitetura eram a largura do barramento, do tipo Harvard ou não, o uso de cache de desvio e o tamanho do conjunto de registradores. Estava-se ainda considerando a não implementação de algumas instruções. Neste caso, a compatibilidade seria mantida com o uso de filtros para o código objeto, ou através de emulação em tempo de execução.

A partir do estudo aqui apresentado, definiu-se uma implementação inicial com o uso de um "pipeline" de 4 estágios, 6 janelas de registradores, um barramento único de 32 bits. Em função da existência ou não de área disponível para integração poderemos tentar incorporar a arquitetura uma pequena cache de desvio de 16 posições.

Não serão implementadas as instruções do tipo TAGGED e as instruções MULScc. Os resultados da simulação confirmam esta opção, como mostrado nos histogramas de instruções, que indicam a baixa utilização destas instruções em aplicações inteiras. Todas as demais instruções serão implementadas, incluindo as de exclusão mútua e de interface com o processador de ponto flutuante.

As chamadas ao sistema e as operações de E/S que forem do padrão SunOS deverão ser emuladas pelo simulador em implementação ainda a ser concluída. Os testes foram realizados com programas sem uso de E/S ou chamadas ao sistema. A interface de usuário definida se mostrou bastante útil, facilitando a depuração do próprio simulador, sem necessidade do uso de outras ferramentas.

Como perspectiva de evolução deste trabalho, existe a intenção de expandir o simulador para instruções de ponto flutuante, verificando o comportamento da interface processador-coprocessador nas diversas configurações de barramento e utilizando-se programas destinados à aplicações científicas. Outra linha de pesquisa que deverá ser explorada é a de arquiteturas superescalares, utilizando-se o simulador para verificar dependências de dados no fluxo de instruções e o desempenho de diversas políticas de escalonamento.

Finalmente, gostaria de agradecer à orientação e apoio recebidos do Prof. Julio Salek Aude, e aos colegas de trabalho Mário Afonso Barbosa e Norival Ribeiro Figueira no auxílio para a definição e detalhamento das instruções da implementação SPARC do NCE/UFRJ.

5- BIBLIOGRAFIA

[AUD90] - Aude, J. S. et alli "MULTIPLUS: Um Multiprocessador de Alto Desempenho" Anais do X Congresso da SBC, Vitoria, Julho de 1990

[COR88] - Cortadella, J. & Jové, T. "Designing a Branch Target Buffer for Executing Branches with Zero Time Cost in a RISC Processor", Microprocessing and Microprogramming, North-Holland, 24: 573-580, 1988.

[GAR88] - Garner, B. et alli "The Scalable Processor Architecture" Proceedings of the IEEE COMPCON 88, New York, NY, IEEE, pp 278-283, 1988.

[HIN84] - Hinnant, D.F. "Benchmarking UNIX Systems" Byte, Peterborough, N.H., McGraw Hill, 9(8):132-5,400-9, Aug 1984.

[KAT83] - Katevenis M.G.H. et alli "The RISC II Micro-architecture." IFIP 1983 (VLSI 1983), Anceu F & Acs E J (Eds) pp. 349-359, Elsevier Science Publishers (North-Holland), 1983.

[KLE88] - Kleiman S.R. & Willians D. "SunOS on SPARC" Proceedings of the IEEE COMPCON 88, IEEE, New York, NY, pp 289-293, 1988.

[LEE84] - Lee, J.K.F. & Smith, A.J. "Branch Prediction Strategies and Branch Target Buffer Design" Computer, New York, IEEE, 17(1):6-22, Jan 1984.

[MUC88] - Muchnick, S.S. et alli " Optimizing Compilers for the SPARC Architecture: An Overview" Proceedings of the IEEE COMPCON 88, IEEE, New York, NY, pp 284-288, 1988.

[NAM88a] - Namjoo, M. et alli "CMOS Gate Array Implementation of the SPARC Architecture" Proceedings of the IEEE COMPCON 88, New York, IEEE, NY, pp 10-13 , 1988.

[NAM88b] - Namjoo, M. et alli "CMOS Custom Implementation of the SPARC Architecture" Proceedings of the IEEE COMPCON 88, New York, NY, IEEE, pp 18-20, 1988.

[QUA88] - Quach L. & Chueh R. "CMOS Gate Array Implementation of SPARC" Proceedings of the IEEE COMPCON 88, New York, NY, IEEE, pp 14-17, 1988.

[REI84] - Reinhold, P.W. "Dhrystone: A synthetic Systems Programming Benchmark", Communications of the ACM, New York, 27(10):1013-1030, Oct 1984

[SEQ82] - Sequin C.H. & Patterson D.A. "Design and Implementation of RISC ." Berkeley, CA, University of California, Report, UCB/CSD 82/106, Oct 1982, 23 pp.

[SIL90] - Silva, G.P. "Um Simulador Configurável para uma Arquitetura RISC" Anais do V Simpósio Brasileiro de Concepção de Circuitos Integrados , Ouro Preto, Outubro 1990, pp 133-142.

[SUN87] - Sun Microsystems Inc "The SPARC Architecture Manual", Mountain View CA, 1987, 199 pp.

[TAM81] - Tamir, Y. "Simulation and Performance Evaluation of the RISC Architecture" Berkeley , CA, University of California, Memorandum, UCB/ERLM 81/17, 1981, 29 pp.

[TAM83] -Tamir, Y. & Sequin, C.H. "Strategies for Managing the Register File in RISC" IEEE Transactions on Computers, Vol C-32 (11): 977-989, Nov 1983

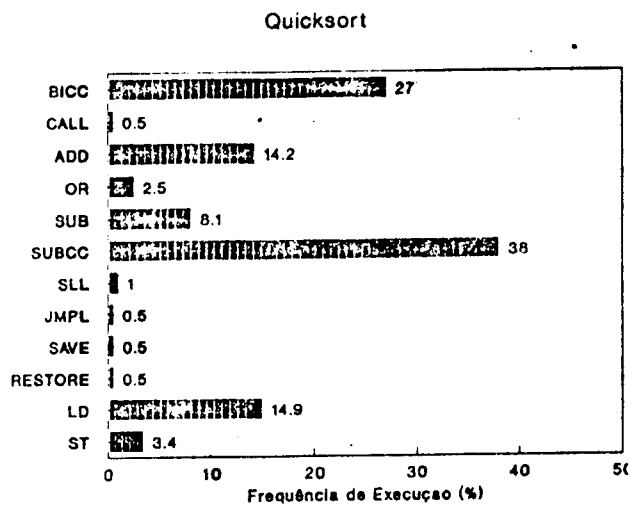

Figura 2- Histograma do Quicksort.

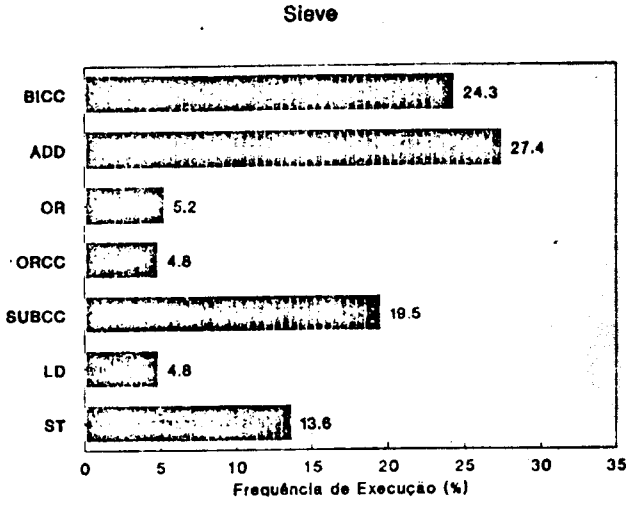

Figura 3- Histograma do Crivo de Eratóstenes

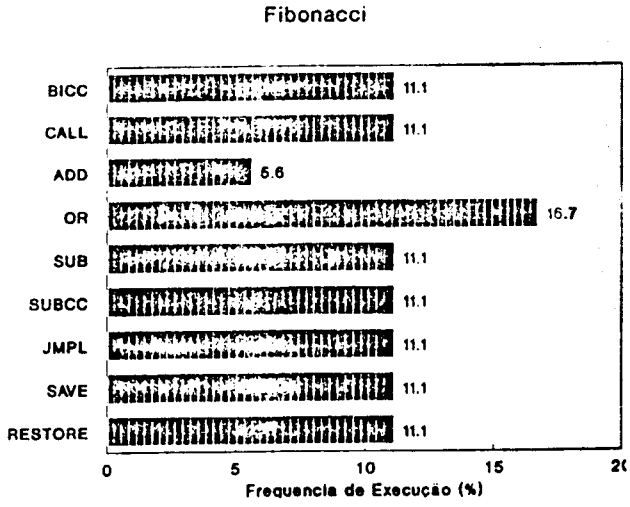

Figura 4- Histograma do Fibonacci.

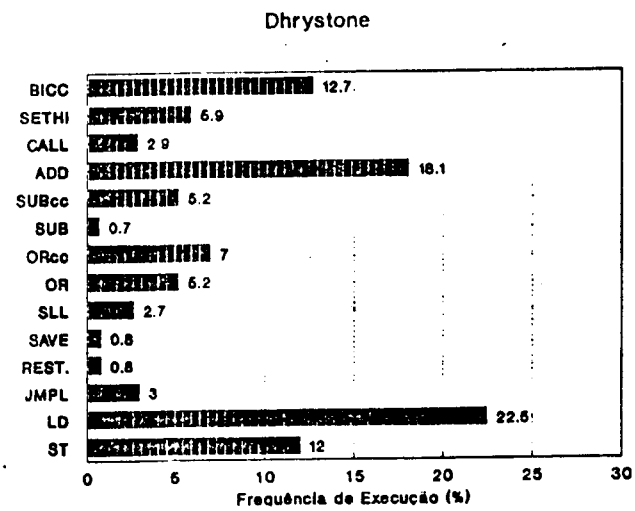

Figura 5- Histograma do Dhrystone.

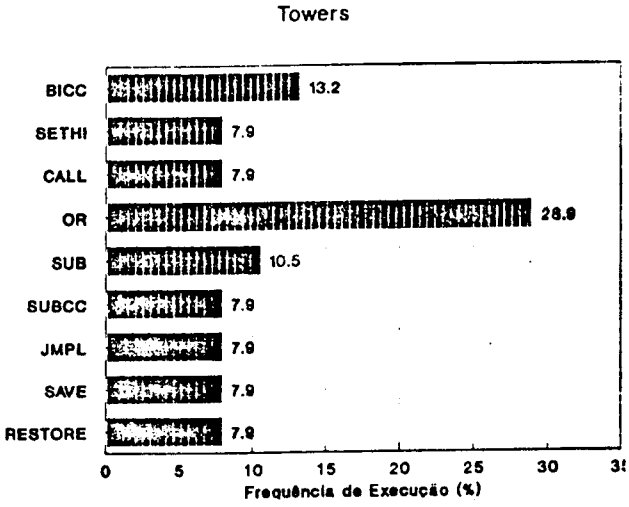

Figura 6- Histograma do Torres de Hanoi.

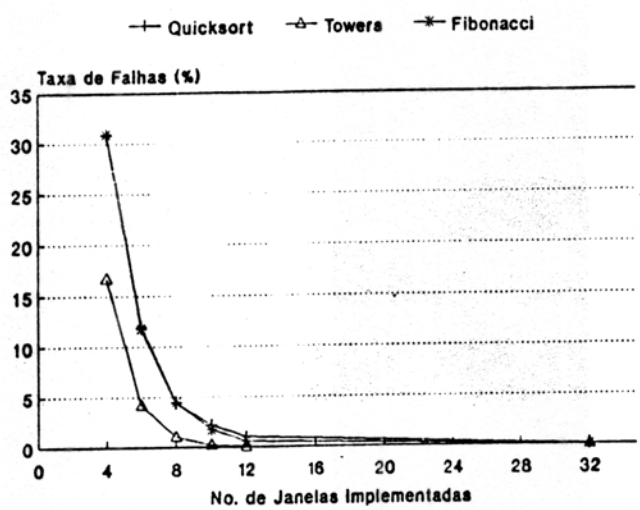

Figura 7 - Taxa de Falhas vs. Número de Janelas(FQT).

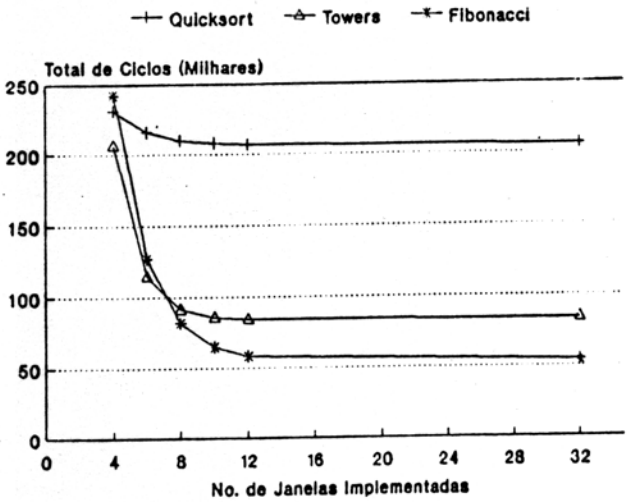

Figura 8- Total de Ciclos vs. Número de Janelas(FQT).

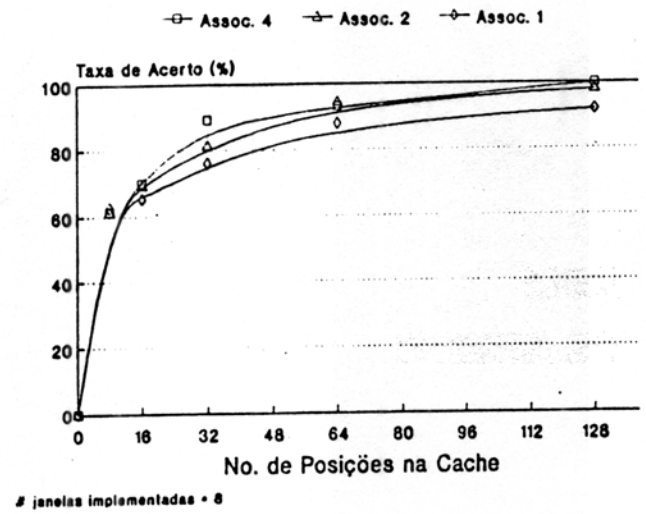

Figura 9- Taxa de Acerto vs. Tamanho da Cache de Desvio(Dhrystone).

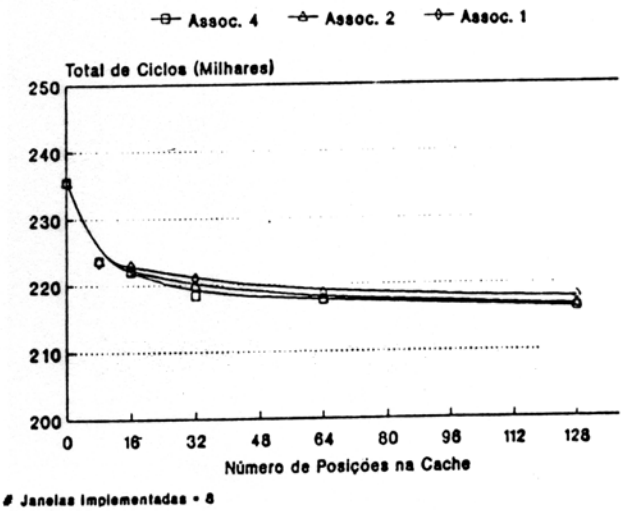

Figura 10- Total de Ciclos vs. Tamanho da Cache de Desvio(Dhrystone).

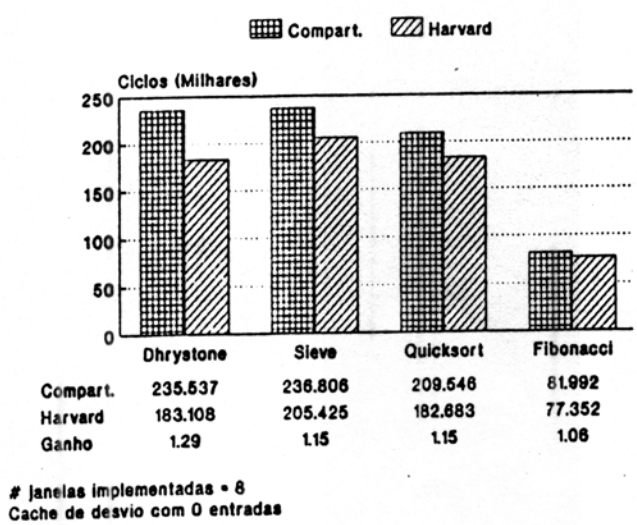

Figura 11- Organização do Barramento.

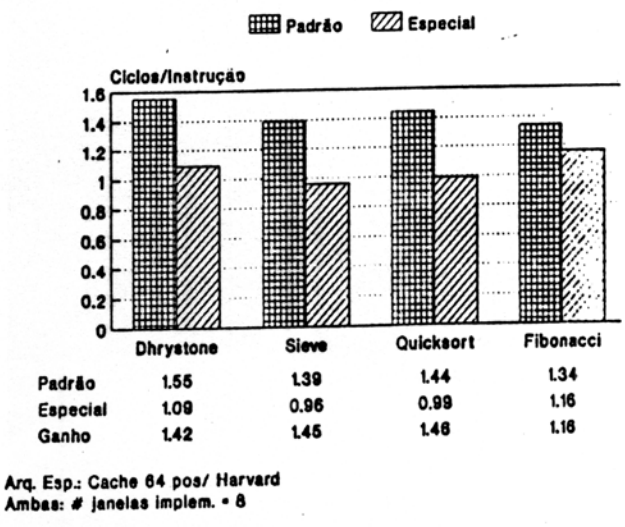

Figura 12- Comparação de Arquiteturas