ESTUDO E AVALIAÇÃO DE ARQUITETURAS RISC PARA USO EM SISTEMAS MULTIPROCESSADORES

Gabriel Pereira da Silva

TESE SUBMETIDA AO CORPO DOCENTE DA COORDENAÇÃO DOS PROGRAMAS DE PÓS-GRADUAÇÃO EM ENGENHARIA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Aprovada por:

Prof. Julio Salek Aude, Ph. D.
(Presidente)

Prof. Valmir C. Barbosa, Ph. D.

Prof. Edil S. T. Fernandes, Ph. D.

Prof. Ageu C. Pacheco Jr, Ph. D.

RIO DE JANEIRO, RJ - BRASIL
JUNHO DE 1991

SILVA, GABRIEL PEREIRA DA
Estudo e Avaliação de Arquiteturas Risc para Uso em Sistemas Multiprocessadores [Rio de Janeiro] 1991.
x, 65 p. 29,7 cm (COPPE/UFRJ. M. Sc., Engenharia de Sistemas e Computação, 1991)
Tese - Universidade Federal do Rio de Janeiro, COPPE
1. Arquiteturas RISC 2.Microprocessadores
I. COPPE/UFRJ II. Título (série).

Para minha mãe, Laura,
a quem devo o sucesso desta jornada.
Para Danilo e Maristela,
por toda compreensão e apoio recebidos.

AGRADECIMENTOS

Agradeço ao Prof. Julio Salek Aude o apoio e orientação recebidos na elaboração desta tese.

Aos colegas Norival Ribeiro Figueira e Mário Afonso Barbosa pelo auxílio na definição e detalhamento da arquitetura do microprocessador SPARC.

Ao DCC/UFMG pela utilização de seus laboratórios para obtenção do código dos programas utilizados nas simulações realizadas.

Ao NCE/UFRJ pelo suporte humano e material para realização deste trabalho.

Ao CNPQ e FINEP pelo apoio financeiro a esta tese.

Resumo da Tese apresentada à COPPE/UFRJ como parte dos requisitos necessários para obtenção do grau de Mestre em Ciências (M.Sc.).

ESTUDO E AVALIAÇÃO DE ARQUITETURAS RISC PARA USO EM SISTEMAS MULTIPROCESSADORES

Gabriel Pereira da Silva
Junho, 1991

Orientador: Julio Salek Aude
Programa: Engenharia de Sistemas e Computação

Esta tese está dividida em duas partes: a primeira faz a análise das características de alguns microprocessadores FUSC de 32 bits e a segunda estuda as alternativas para a implementação de uma destas arquiteturas, no caso o SPARC.

A primeira parte deste estudo foi realizada para permitir a escolha do microprocessador a ser utilizado no projeto MULTIPLUS, um sistema multiprocessador em desenvolvimento no NCE/UFRJ. Duas características foram consideradas fundamentais nas arquiteturas candidatas: permitir o desenvolvimento de uma arquitetura compatível com o processador inicialmente utilizado e possuir suporte para uso em sistemas multiprocessadores.

A segunda parte desta tese faz um estudo sobre as opções de implementação para a arquitetura escolhida: o processador SPARC. O SPARC tem uma arquitetura aberta e permite a existência de implementações diversas, embora mantendo compatibilidade binária. O simulador funcional SIMUS foi elaborado para avaliar opções para o tamanho do conjunto de registradores, para a organização do barramento e para a estrutura de uma cache interna. Os resultados desta simulação são mostrados e analisados, e uma proposta de irnplementação para a arquitetura em estudo é apresentada.

Abstract of Thesis presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.).

STUDY AND ANALYSIS OF RISC ARCHITECTURES FOR USE IN MULTIPROCESSOR SYSTEMS

Gabriel Pereira da Silva
June, 1991

Thesis Supervisor: Julio Salek Aude
Depaitment: Computes Science and Systems Engineering

This thesis consists of two parts: the first analyses the characteristics of some 32 bit RISC microprocessors and the second one studies the options for the implementation of one of these microprocessors architectures: the SPARC.

The first part of this work has been done to help the choice of a RISC microprocessor for the MULTIPLUS project, a shared memory multiprocessor system under development at NCE/UFRJ. Two fundamental characteristics of the candidate architectures have been considered: it should allow the local development of a compatible architecture that could replace the commercial processor to be initially used and it should have support for use in multiprocessor systems.

The second part of this thesis studies options for the implementation of the chosen architecture: the SPARC processor. The SPARC has an open architecture which allows different implementations, while keeping binary compatibility. The SIMUS functional simulator has been developed to evaluate options to the register file size, to the bus organization and to an internal cache structure. The results of this simulation are shown and analysed, and an implementation proposal for the studied architecture is presented.

ÍNDICE

LISTA DE FIGURAS

TABELAS

1. INTRODUÇÃO

A primeira parte desta tese apresenta um estudo das principais arquiteturas RISC existentes, fazendo-se um levantamento das principais características necessárias a um processador para trabalhar em ambiente com multiprocessamento. A partir deste estudo pôde-se definir um microprocessador para utilização no projeto MULTIPLUS.

O projeto MULTIPLUS [AUD90] é um sistema multiprocessador com memória global compartilhada em desenvolvimento no NCE/UFRJ. Sua proposta é obter um sistema paralelo de alto desempenho com o uso de rnicroprocessadores RISC acoplados a unidades aritméticas de ponto flutuante.

O MULTIPLUS é um sistema do tipo multiprocessador projetado para ter uma arquitetura modular capaz de suportar até 2048 nós de processamento. Cada nó de processamento consiste de:

. Um microprocessador RISC baseado na definição do SPARC;

. Memória de até *32* Mbytes com endereçamento global;

. Uma cache de instruções e outra de dados de 64 Kbytes cada;

. Um coprocessador de ponto flutuante;

. Uma unidade de gerência de memória.

Até 8 nós de processamento podem ser interconectados através de um barramento duplo, formando um grupo de processadores. Os diferentes grupos se interconectam através de uma rede de interconexão multiestágio do tipo N-cubo.

A rede de interconexão está sendo implementada com módulos de chaves 2 X 2 com FIFOs, e se liga aos barramentos através de interfaces inteligentes. Com isto, o MULTIPLUS obtém características de modularidade e particionamento, que permitem soluções adequadas às características de muitas aplicações.

No âmbito do projeto MULTIPLUS, além do estudo e implementação de arquiteturas paralelas, tem-se o objetivo de contribuir na criação de capacitação nacional na área de projeto de circuitos integrados, especialmente em arquiteturas RISC com tecnologia CMOS. Por causa disto, o microprocessador comercial escolhido para utilização na primeira fase do projeto será substituído posteriormente por um microprocessador funcionalmente equivalente projetado no NCE/UFRJ.

Havia duas opções básicas na escolha do microprocessador: com utilização de conjunto complexo de instruções (CISC) ou com uso de conjunto reduzido de instruções (RISC). Os primeiros são baseados no desenvolvimento de instruções complexas, cujo controle e feito por um microprograma em uma memória intema a pastilha. A segunda utiliza instruções simples, com o controle realizado através de circuitos lógicos, transferindo para o compilador a responsabilidade de emular as funções mais complexas.

Os microprocessadores com filosofia RISC [PAT80] possuem uma arquitetura interna mais simples e, conseqüentemente, podem ser implementados com tecnologia menos sofisticada e com menor gasto de área. A velocidade de execução das instruções é bem maior, o que compensa um tamanho maior dos programas objetos. São de especial utilização em ambientes do tipo UNIX, já que as "chamadas-padrão" e o uso da linguagem "C" permitem tomar transparente ao usuário qual processador está sendo utilizado.

Estas qualidades fazem o microprocessador RISC de particular interesse para ser desenvolvido no país por dois aspectos principais: o menor grau de integração exigido toma viável sua implementação em curto prazo pela indústria nacional de microeletrônica; a experiência anterior acumulada com o desenvolvimento de sistemas compatíveis com o UNIX, como o SOX e o PLURIX, toma possível a elaboração de sistemas que utilizem este microprocessador.

A segunda parte desta tese trata então do estudo das opções existentes para irnplementação de uma arquitetura compatível com o microprocessador comercial escolhido: o SPARC. A arquitetura SPARC é uma arquitetura aberta, que permite a elaboração de implementações distintas, embora mantendo a compatibilidade binária. Esta tese realiza

a avaliação de desempenho do processador em diversas configurações de sua microarquitetura. As modificações realizadas têm como objetivo aumentar o desempenho do microprocessador, sem comprometer sua fabricação em uma única pastilha, nas tecnologias CMOS correntemente disponíveis as Universidades brasileiras.

Esta tese está organizada em 6 capítulos: No Capítulo 2 é feito um estudo das características de projeto de vários microprocessadores RISC. Nos Capítulos 3 e 4 são descritas as características do SPARC e do simulador SIMUS, respectivamente. No Capítulo 5 são apresentados os programas de avaliação utilizados e feita uma análise dos resultados das simulações. Finalmente, no Capítulo 6 são apresentadas as conclusões, uma proposta para a implementação da arquitetura SPARC no NCE/UFRJ e perspectivas de desdobramento deste trabalho.

2. COMPARAÇÃO DAS ARQUITETURAS RISC

2.1 - Introdução

Os microprocessadores RISC se distinguem por uma série de características comuns, que resultam em uma solução de baixo custo e alto desempenho. Contudo, cada microprocessador coloca soluções específicas para problemas como acesso aos dados, organização de registradores, estrutura do "pipeline" e conjunto de instruções. Nas seções seguintes são apresentadas as soluções adotadas pelas arquiteturas estudadas, ou seja, o SPARC [SUN87,CYP90], o MIPS [HEN82,KAN88], o MC88100 [MOT88], o Am29000 [AMD88]. Além destes quatro processadores estudados com mais profundidade, são mencionadas soluções adotadas também no CLIPPER [FAI86], no CRISP [BER87] e no i80860.

As características comuns, encontradas na maioria dos processadores RISC são as seguintes:

- Uso de "pipeline": a tarefa para execução de uma instrução é dividida em estágios bem definidos, com alocação de recursos de uso exclusivo para cada uma delas. Assim, podem ser executadas simultaneamente tantas instruções quanto forem os estágios definidos, para se atingir a taxa de 1 ciclo/instrução na maioria das instruções.

- Uso de circuitos lógicos para controle e sequenciamento das instruções. Não é utilizada memória de microcódigo, o que permite menor gasto de área internamente ao processador para as funções de controle. O uso de circuitos lógicos só é possível pelo fato de o conjunto de instruções ser de extrema simplicidade.

- Arquitetura orientada a registrador: todas as operações aritméticas são realizadas entre registradores. Os acessos a memória existem

apenas para transferências de dados de/para os registradores. Esta filosofia é baseada no fato de que grande parte das variáveis utilizadas por um programa são temporárias e com tempo de vida muito curto. Assim, idas e vindas desnecessárias a memória são evitadas, pois estas variáveis ficam contidas nos registradores.

- Uso de formato fixo das instruções: Isto facilita o controle do "pipeline" e simplifica a decodificação das instruções. A opção norrnalmente utilizada é por instruções com 32 bits de largura.

- Uso intensivo de registradores internos e memória cache (interna ou externa), como forma de reduzir a utilização do barramento. A maior limitação destas arquiteturas é a alta taxa de instruções que é solicitada pelo processador. O uso de caches e registradores diminui a utilização do barramento, sendo fundamental para sistemas com múltiplos processadores.

- Migração de funções para o compilador: somente o essencial é implementado na pastilha. Consequentemente, existe uma extrema dependência do compilador-otimizador, específico para cada arquitetura, para executar as tarefas com bom desempenho. Poderíamos dizer que um processador RISC é metade silício e metade compilador.

- Da mesma maneira que nos processadores CISC, existe a necessidade do uso de processador de ponto flutuante para a execução de operações aritméticas mais complexas.

Com o uso destas características os microprocessadores RISC conseguem ter um desempenho superior aos dos processadores do tipo CISC, com um "hardware" bem mais simples [PAT81]. Uma característica

fundamental para um bom desempenho em aplicações científicas é o uso de unidades aritméticas de ponto flutuante. Portanto, as arquiteturas que não oferecem uma interface para este tipo de unidade não foram consideradas neste estudo.

Além daquelas já mencionadas, existe uma série de outras características com forte influência no desempenho de cada microprocessador, que são analisadas a seguir.

2.2 - Conjunto de Instruções

Na avaliação do conjunto de instruções procurou-se determinar um conjunto de instruções simples, que facilitasse a implementação de uma arquitetura compatível, mas que também não comprometesse a eficiência do processador nas condições de trabalho pretendidas. Neste estudo as instruções são divididas em *3* grupos principais: as executadas pela unidade inteira, as de interface com outras unidades, e as de ponto flutuante. As instruções dos dois primeiros grupos são normalmente executadas pelas unidades inteiras, e são as instruções que influenciam na complexidade final da máquina. As instruções do último grupo definem a capacidade de lidar com problemas científicos, importantes para a aplicação pretendida, mas são normalmente implementadas em pastilhas a parte, e não comprometem a área disponível para integração. Foi estabelecido um critério idêntico para classificação e determinação do total de instruções de cada microprocessador, embora não tenha sido uma tarefa fácil em certas situações, distinguir o que fosse uma variação de instrução, de uma nova instrução. Os resultados da análise realizada são descritos a seguir:

O SPARC herdou de seus antecessores a execução de desvios atrasados e uso de instruções para suporte a linguagem orientada a objeto. Na sua definição foram introduzidas extensões para suporte a multiprocessamento, para coprocessador de ponto flutuante, e para um coprocessador genérico. Existem 5 formatos para as instruções inteiras. O seu conjunto de instruções tem 58 instruções inteiras, outras 16 para acesso ao coprocessador de ponto flutuante e ao coprocessador genérico e 36 instruções de ponto flutuante.

O ambiente de programação prevê os modos usuário e supervisor, com instruções privilegiadas. As instruções de "load" e "store" podem ter operandos de 8, 16, 32 ou 64 bits, e permitem o acesso direto do supervisor ao espaço de endereçarnento do usuário com o uso de "load/store alternate". Não possui instrução de multiplicação, mas sim uma de "multiply step", que permite a execução de uma multiplicação de 32x32 bits em 6 ciclos na média ou em 36 ciclos no pior caso. As instnições de ponto flutuante executam em uma unidade a parte e em paralelo com a execução de instruções na unidade inteira. O processamento só é suspenso se uma instrução necessitar de um dado em um registrador do processador de ponto flutuante que ainda não está pronto.

O SPARC possui um registrador com códigos de condição para avaliação de desvios condicionais. Certas instruções aritméticas alteram o valor deste códigos de condição e a instrução de desvio condicional avalia os códigos de condição para decidir se executa o desvio ou não. Em outros microprocessadores, como veremos adiante, esta forma de avaliação pode ser diferente. As instruções de desvio permitem que a execução da instrução seguinte seja opcional. Este bit a mais no código da instrução dá maior facilidade para relocação de código pelo compilador. Os desvios máximos permitidos são de 2^24 a partir do endereço da instrução de desvio.

Todas as instruções do SPARC são tnádicas, ou seja, possuem três operandos, dois fontes e um destino. Quanto aos tipos de operandos fontes para as instruções aritméticas, eles podem ser qualquer combinação de dois registradores, ou de registrador e dado imediato. O destino, como em todo processador RISC, é um registrador. Finalmente, as instruções de LDSTUB e SWAP perrnitem a execução de instruções de ciclo indivisível na memória, importantes para operação em ambientes com múltiplos processadores.

O conjunto de instruções do MIPS possui 59 instruções de aritmética inteira, organizadas em 3 formatos básicos, mais ***32*** para acesso a 4 coprocessadores e 25 instruções de ponto flutuante. O ambiente de programação possui os modos usuário e kemel, com instruções privilegiadas. As instruções de "load" e "store" podem ter

apenas operandos de 32 bits para acesso a memória. Contudo, constantes de 8 ou 16 bits podem ser carregadas como dado imediato nos registradores.

O MIPS não possui instruções específicas para acesso direto do supervisor ao espaço de endereçamento do usuário. Por outro lado, possui instruções de multiplicação e divisão de 32 bits que são executadas por uma unidade de execução separada da ALU. Assim é possível um paralelismo entre estas instruções e as demais instnições inteiras. Em termos de instruções de ponto flutuante, existe um coprocessador externo que executa as instnições de ponto flutuante, concorrentemente com a unidade inteira e com uso também de um "pipeline" de instrução.

O MIPS não possui códigos de condição. A instrução de desvio faz a comparação direta entre o conteíido de dois registradores e desvia de acordo com o resultado desta comparação. Os desvios máximos permitidos são de 2^18 a partir do endereço da instrução seguinte, que é sempre executada. Sua instruções também são triádicas, mas não possui instruções para execução de ciclo indivisível na memória. A ausência desta instrução obriga a existência de um circuito externo para uso em sistemas com míiltiplos processadores, e compromete o desempenho e a compatibilidade entre máquinas diferentes.

As instruções do Am29000 possuem 6 formatos básicos, sendo que 74 são inteiras, 2 são pseudos instruções, 7 para acesso ao processador de ponto flutuante e 74 são instruções de ponto flutuante. O processador pode trabalhar nos modos usuário e supervisor, existindo instruções privilegiadas. As instnições de "load" e "store" podem ter operandos de 8, 16 ou 32 bits, e permitem o acesso direto do supervisor ao espaço de endereçarnento do usuário. Possui também instruções de "load" e "store" múltiplo, que são capazes de realizar até 256 transferências contínuas. Durante qualquer operação de "load" ou "store", a execução das instruções prossegue no "pipeline", até que um dado envolvido na operação seja necessário.

O Am29000 possui duas pseudos-instruções, para multiplicação e divisão, que se forem executadas provocam o desvio para uma rotina de exceção. Estão implementadas apenas as instruções de

"multiply" e "divide step", que permitem a execução de uma multiplicação ou divisão de 32x32 bits em **36** ciclos no pior caso. Existe previsão para uso de uma unidade de ponto flutuante para execução das instruções de ponto flutuante. Todos os acessos a esta unidade, inclusive para a carga de instruções, são realizados pelo Am29000.

O Am29000 não utiliza os códigos de condição da ALU para realização de desvios condicionais. Para isto existem instruções especiais de comparação que colocam o resultado da comparação ("true/false") no registrador destino especificado na instrução. A instrução de desvio condicional especifica este registrador e verifica o resultado da comparação realizada. O máximo desvio relativo ao PC que pode ser realizado é de 2^18, sendo que as instruções imediatamente após a instrução de desvio são sempre executadas. O Am29000 possui três instruções de ciclo indivisível, LOADSET, LOADL e STOREL, para utilização em ambientes com multiprocessamento.

O conjunto de instruções do MC88100 é o maior dos microprocessadores estudados, possuindo até instruções para manipulação de campo de bits. Possui um total de 124 instruções inteiras e 47 de ponto flutuante. Este tamanho implica em um aumento considerável do gasto da área disponível na pastilha para a implementação da lógica de controle, o que de certa forma contraria o conceito original de arquiteturas RISC. O MC88100 possui suporte para uso em multiprocessamento, inclusive com esquema para processamento tolerante a falhas, com possibilidade de uso de processadores redundantes para verificação do funcionamento do processador principal.

O seu conjunto de instruções foi otimizado para as intruções mais utilizadas por um compilador C. As instruções de multiplicação e divisão inteiras são executadas na unidade de ponto flutuante. Como a unidade de ponto flutuante está integrada na mesma pastilha, não há instruções especiais para transferências de dados dos seus registradores. Apesar do grande tamanho do conjunto de instruções, todas as instruções lógicas e inteiras são executada em um único ciclo. Não existe um registrador específico de código de condição, os resultados vão para um registrador de uso geral determinado na instrução de comparação, que é então testado pela instrução de desvio condicional.

A unidade de ponto flutuante possui multiplicador e somador independentes. Devido ao grande número de unidades funcionais existentes, possui um alto grau de paralelismo na execução das instruções. Para exemplificar, em apenas um único ciclo podemos ter até cinco adições de ponto flutuante, quatro multiplicações de inteiros, três acessos a memória, e duas buscas de instrução em andamento. Como consequência possui um alto desempenho, mas o tratamento dos casos de exceção é bastante complexo, necessitando de um grande número de registradores auxiliares para a recuperação de contexto.

2.3 - Organização dos Registradores

As arquiteturas estudadas apresentam três tipos básicos de registradores: os de serviço ou especiais, os de uso geral e, eventualmente, os de ponto flutuante. Os registradores de serviço são diretamente proporcionais ao número de unidades funcionais integradas na pastilha. Assim sendo, o SPARC possui 5 registradores de serviço, o MIPS possui 10 registradores, o Am29000 tem 23 especiais e o MC88100 21 de serviço, 4 internos e 11 de ponto flutuante.

Os registradores de ponto flutuante, mesmo quando em uma pastilha à parte, são vistos normalmente como uma extensão dos registradores de uso geral. O MC88100 possui um conjunto de registradores de ponto flutuante com 5 portas de acesso, que pode ser configurado com 8 x 128, 16 x 64 ou 32 x 32 bits. Já a definição do SPARC possui 32 registradores de ponto flutuante que, de acordo com o tipo de operando, podem ser concatenados para formar registradores de 64 ou 128 bits. O MIPS possui 32 registradores de ponto flutuante, podendo ser organizados em 32x32 bits ou 16x64 bits. O conjunto de instruções do Am29000 possui suporte para operandos de precisão simples e de precisão dupla, mas não define o uso de nenhum coprocessador específico.

Os registradores de uso geral podem ser organizados de três formas básicas, aqui chamadas de convencional, pilha e janela.

Da maneira convencional o conjunto de registradores possui endereços físicos determinados e são referenciados explicitamente pelas instruções, tal como ocorre nos processadores CISC convencionais. Este tipo de estrutura foi adotado nos microprocessadores MC88100 (Motorola) e no R3000 (MIPS),ambos com 32 registradores de 32 bits. A vantagem deste método é a simplicidade e rapidez para o endereçamento dos registradores. A desvantagem é que este esquema obriga a existência de um reduzido número de registradores, já que a referência a estes registradores consome um grande número de bits no código das instruções.

No modo pilha, os registradores são referenciados indiretamente por um "stack pointer" e aparecem como uma continuação da pilha existente na memória. Este esquema se mostra bastante eficiente, pois existe uma acentuada localidade de referências de dados ao topo da pilha, além de ser eliminada a necessidade de alocação de registradores por "software". Há uma redução por um fator de três as referências externas, quando comparado com alocação feita por compilador [BER87].

A pilha também permite que as chamadas de rotinas sejam agilizadas, pois na maior parte dos casos não há necessidade de salvamento/recuperação de registradores durante a chamada ou retorno das rotinas. O número de registradores na pilha pode ser modificado em futuras irnplementações sem que isto implique em alterações nos programas e/ou compiladores, pois não há referência explícita aos registradores no código das instruções. Esta solução foi adotada pelo Am29000, da AMD, que possui 128 registradores locais organizados desta forma, e pelo CRISP, um projeto da AT&T, com ***32*** registradores internos que emulam o topo da pilha.

O esquema de janelas é semelhante ao de pilha e utilizado quando se deseja a otimização do tempo gasto com chamadas de subrotinas. Estudos [PAT81] mostram que os tipos mais frequentes de operandos são variáveis escalares locais, ou seja, não são vetores e só existem no escopo da subrotina. Assim sendo, uma forma de otimizar a utilização destes operandos seria o uso de múltiplos bancos de registradores para o armazenamento destes dados. Neste caso, a cada chamada de rotina um conjunto de registradores diferente é ativado,

permitindo a cada rotina operar com suas variáveis locais. Normalmente as janelas de registradores se sobrepoem parcialmente, permitindo a passagem de um determinado número de parâmetros entre as rotinas. Levantamentos [GAR88] indicam um índice de 20% de acessos a dados na memória, contra 30 a 40% em microprocessadores sem uso de janelas de registradores. Este esquema e utilizado no SPARC e os processadores RISC de Berkeley.

2.4 - Desvio Atrasado

Ender.	Normal	Atrasado	Otimizado
100	LOAD X,A	LOAD X,A	LOAD X,A
101	ADD I,A	ADD I,A	JMP 105
102	JMP 105	JMP 106	ADD I,A
103	ADD A,B	NOP	ADD A,B
104	SUB C,B	ADD A,B	SUB C,B
105	STORE A,Z	SUB C,B	STORE A,Z
106		STORE A,Z	

Figura 2.1 - Exemplo de Desvio Atrasado

O uso de "pipeline" melhora consideravelmente o desempenho da máquina. A execução das instruções é dividida em vários estágios, e é possível a execução simultânea de tantas instruções quantos forem os estágios do "pipeline". Esta simultaneidade, contudo, introduz algumas dificuldades, já que uma nova instrução é buscada em paralelo com a decodificação da instrução anterior. Quando for executada uma instrução de desvio, a instrução seguinte já terá sido buscada e estará pronta para entrar no próximo estágio do "pipeline". A solução normalmente adotada é também executar a instrução que já foi buscada, caracterizando o que é chamado de "desvio atrasado". O exemplo da figura 2.1 mostra como o aproveitamento desta instrução pode ser feito pelo compilador.

Em alguns microprocessadores a execução da instrução que se encontra no "delay slot" é opcional, em outros é obrigatória. No caso do MIPS e Am29000 esta execução é obrigatória, e no SPARC e MC88100 ela é opcional. Normalmente o compilador consegue introduzir código útil após estas instruções de uma maneira transparente ao programador, mas o fato desta execução ser opcional traz uma maior flexibilidade no aproveitamento destas instruções pelo compilador.

2.5 - Dependências no Pipeline

Outro efeito colateral que surge com o uso do "pipeline", é a ocorrência de dependências entre os dados das instruções que estão no "pipeline". Isto acontece quando uma instrução deseja utilizar um registrador que foi modificado por uma instrução anterior, mas que devido ao processo inerente ao "pipeline" não foi atualizado. Existem dois procedimentos básicos para lidar com este caso: congelar o "pipeline" ("interlock") até o dado estar atualizado ou deixar para o compilador a tarefa de evitar que estas dependências ocorram.

O MIPS transfere para o compilador esta tarefa, por este motivo foi batizado de "Microprocessor without Interlocked Pipeline Stages (MIPS)".A mudança de complexidade do "hardware" para o "software" tem as seguintes vantagens:

- A complexidade só influencia uma vez durante a compilação. Quando um usuário executa seu programa em urna arquitetura complexa, ele paga o preço deste custo cada vez que executa o programa.

- Isto permite a concentração de energias no "software", ao invés de se construir uma máquina com "hardware" complexo, que é difícil de projetar, depurar e utilizar eficientemente. O "software" não é necessariamente mais fácil de construir, mas o ambiente de VLSI faz a simplicidade do "hardware" ser importante.

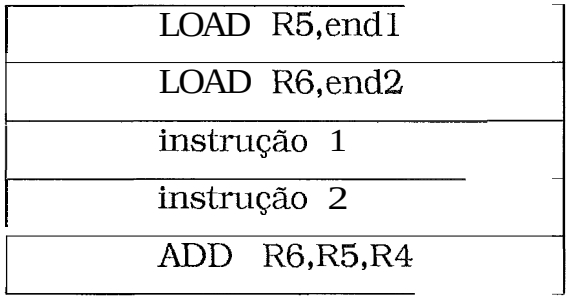


Figura 2.2 - Exemplo de Interlock

O SPARC e o Am29000 utilizam uma forma mais simplificada de "interlock" por "hardware", verificando a dependência apenas entre as instruções que estão no "pipeline". Uma técnica mais complexa introduzida pelo CLIPPER da Fairchild, e utilizada no MC88100, é o uso de "register scoreboarding", que permite o controle quando existe mais de uma unidade funcional na mesma pastilha. Esta técnica perrnite a execução simultânea de instruções de LOAD com outras instruções. Quando a instrução de LOAD é iniciada, um bit é setado em um registrador especial. Quando a operação de memória realmente termina, o bit é resetado. Neste intervalo, qualquer referência a um registrador marcado faz o processamento parar até que a operação de carga tenha terminado. Um caso típico de "interlock" é mostrado na Figura 2.2. As instruções 1 e 2 são executadas em paralelo com as instruções de LOAD, sem nenhum gasto adicional de tempo. A instrução de ADD só será executada quando os operandos estiverem disponíveis nos registradores.

Em quaisquer dos casos, a ocorrência destas dependências de dado atrasa o processamento, e deve ser evitada através de um seqüenciamento adequado das instruções em tempo de compilação.

2.6 - Organização dos Barramentos

Existem três tipos principais de organização para a interface destes microprocessadores: utilizar um barramento único para dados e instruções; barramentos totalmente separados de dados e instruções, com endereçarnento independente; e barramento separado de dados e

instruções, mas com endereçamento compartilhado. O primeiro método é utilizado no MIPS e na implementação CYPRESS do SPARC. O segundo método é adotado no MC88100 e no i80860, e o terceiro no Am29000.

O MIPS e SPARC da CYPRESS possuem um barramento de 32 bits para dados e instruções, e outro, também de 32 bits, para endereçamento. O SPARC possui bits adicionais para informar o tipo de acesso que está sendo feito, dado ou instrução, usuário ou supervisor. Já o MIPS, como possui uma tabela de translação de endereços interna a pastilha, não precisa utilizar este recurso.

O MC88100 é um processador com barramentos separados para dados e instruções, cada um deles com 64 bits de largura. O MC88200 é uma pastilha que integra um controlador de cache e uma gerência de memória, sendo necessário o uso de uma pastilha MC88200 para cada barramento. A implementação deste tipo de estrutura envolve grande gasto de área na pastilha.

O barramento Am29000 possui uma estrutura singular: existe um barramento de endereço compartilhado por dois barramentos distintos para dados e instrução, com capacidade de leitura de dados em modo rajada, com taxa de transferência de até 100 Mbytes/s. A busca de instruções é feita do seguinte modo: é colocado um endereço inicial no barramento de endereços, e a busca prossegue sequencialmente no barramento de instruções, até que haja uma instrução de desvio, seja ultrapassada uma fronteira de página, ou ocorra uma exceção de memória. Assim, o barramento de endereços pode ficar livre para endereçar as operações de leitura/escrita dos dados.

Os microprocessadores estudados fazem o uso de registradores e de cache interna, como forma de diminuir a ocupação destes barramentos. Quando é utilizado um grande número de registradores internos, estes realizam a função de uma cache interna de dados. Os microprocessadores mais recentes, como o i80860, apresentam caches internas de dados e instruções. Uma forma intermediária utilizada pelo Am29000 e um grande conjunto de registradores e uma cache de endereços de desvio, que permite manter o "pipeline" sempre cheio.

Mesmo assim, ainda existe uma alta demanda de dados, e normalmente são utilizadas caches externas, como no MC88100, SPARC e MIPS, separadas em dados e instruções.

2.7 - Arquitetura Escolhida

O objetivo último deste estudo foi a escolha de uma destas arquiteturas para utilização no projeto MULTIPLUS. Esta escolha foi realizada a partir de requisitos básicos descritos a seguir:

- Simplicidade de código: Um conjunto de instruções mais extenso implica em um controle e lógica mais complexos, que poderia inviabilizar a implementação de um protótipo pelo tamanho final que o circuito teria com a tecnologia de implementação disponível.

- Possibilidade de implementação de um subconjunto de registradores: Caso não seja possível implementar toda a arquitetura, a organização dos registradores deve permitir a utilização de um subconjunto dos registradores para tomar a implementação viável. Neste caso, a compatibilidade é mantida emulando-se os registradores em memória.

- Suporte para multiprocessamento: Atualmente, o melhor caminho para obtenção de um sistema de alto desempenho é através da utilização de processamento paralelo, ou seja, dividir o programa em várias tarefas, cada uma delas executada por um microprocessador distinto, conseguindo-se um tempo total de execução muito menor. Para que isto seja possível, em um ambiente com memória compartilhada como o MULTIPLUS, cada processador deve possuir instruções de sincronização para permitir a utilização concorrente dos recursos do sistema.

- A existência de uma forma simples de "interlock" por "hardware" e o uso do desvio atrasado opcional tomariam mais fácil a futura implementação de um compilador. Neste ponto não havia ainda uma boa estimativa para o gasto adicional de lógica de controle requerido para estas facilidades. Estimava-se que a implementação do desvio opcional fosse de baixa complexidade, mas o mesmo não se podia afirmar sobre o uso de "interlock".

Não foram estabelecidos critérios rigidos para o tipo de barramento que deveria ser utilizado, nem para a forma de organização ou mesmo utilização de uma cache interna. Neste caso, a melhor linha para orientação é a simplicidade. Com base nestes critérios, a equipe de projeto optou pela utilização do SPARC, como justificado a seguir:

- Apresenta um dos menores conjuntos de instruções, com apenas 58 instruções inteiras. Os demais microprocessadores apresentam conjuntos maiores de instruções, sendo que apenas o comjunto do MIPS possuía um tamanho equivalente. Foi um fator de peso na decisão final, pois o gasto de área do controle é proporcional ao número de instruções.

- Possui suporte para multiprocessamento através de instruções de ciclo indivísivel (SWAP, LDSTUB). Em alguns microprocessadores, como o MC88100 e Am29000, existe o mesmo tipo de suporte, mas no MIPS ele é inexistente. Este foi um fator decisivo para a não utilização do MIPS.

- Possui um "annul bit" que faz com que a execução de uma instrução após um desvio seja opcional. Isto dá mais flexibilidade para o compilador.

- Implementa uma forma simples de "interlock" por "hardware". Como não havia estimativas seguras sobre os custos envolvidos, evitou-se a utilização de esquemas mais complexos, como "register scoreboarding".

  O tamanho do conjunto de registradores não é fixo, podendo variar entre 40 e 520. Possui urn esquema de janelas, que permite a emulação de registradores virtuais em memória. Nos demais microprocessadores estudados o tamanho do conjunto de registradores é fixo, e apenas o Am29000 apresenta uma organização do tipo pilha para seus registradores, mas que não é muito flexível. Esta flexibilidade é importante para permitir a implementação de um protótipo com número reduzido de registradores.

- É uma arquitetura aberta, sendo produzida por diversos fabricantes, permitindo inclusive o uso de tecnologias diferentes, como CMOS ou ECL. Detalhes como a largura do barramento e a existência ou não de cache intema são deixados a cargo de cada implementação. Isto dá bastante liberdade na implementação de urna arquitetura compatível de acordo com nossas necessidades e possibilidades. Este foi um fator decisivo na escolha deste processador.

- Não possui uma gerência de memória interna de translação de endereços, mas uma pastilha externa que já realiza a busca automática do descritor na tabela de páginas. O Motorola MC88100 utiliza o mesmo esquema, mas o MIPS R3000 e o AMD Am29000 possuem urna cache de endereços intema a pastilha, o que consome muito espaço, e obriga o processador a executar uma rotina para busca e carga dos descritores nesta cache.

- Há implementações com desempenho de ***33*** Mips a 40 Mhz [CYP90], e uma taxa de execução em torno de 1.5 ciclos para cada instrução. Em outros processadores com arquitetura mais sofisticada esta taxa não é menor que 1.3.

- Possui uma interface para a unidade de ponto flutuante que permite acoplar o processador que for escolhido pelo usuário, oferecendo maior flexiblidade para uso de um processador de ponto flutuante de maior desempenho ou de uma implementação própria.

3. DESCRIÇÃO DO SPARC

3.1 - Histórico

O SPARC, acrossemia para Scalable Processor ARChitecture, originou-se nos projetos pioneiros de Berkeley, desenvolvidos a partir de 1981 por alunos de graduação e pós-graduação: o RISC I [PATB1,SEQ82], RISC II [KAT83] e SOAR [UNG84].

O RISC I foi o primeiro destes projetos, tendo introduzido diversos conceitos novos. Houve várias dificuldades para sua fabricação, pois apenas se iniciava a utilização de "foundries" para este fim. Por exemplo, de quarenta unidades produzidas apenas quatro funcionaram, a uma frequência de "clock" bem abaixo daquela esperada. O RISC II foi desenvolvido em dois anos, sendo um projeto mais refinado, com um nível a mais de "pipeline", mais registradores, e uma área final menor que o RISC I. Estes dois projetos foram desenvolvidos com tecnologia NMOS.

O RISC I e o RISC II não possuíam previsão para uso em ambiente multiprocessador, nem para utilização de coprocessador de ponto flutuante, sendo o percentual de área gasto na lógica de controle de 5% e 10%, respectivamente. O RISC I apresentava barrarnentos distintos para dados e instruções, enquanto que o RISC II possuía apenas um barramento compartilhado.

O SOAR tinha o mesmo conjunto básico de instruções, com a diferença de possuir instruções específicas para programação orientada a objeto, em particular para o SMALLTALK. Estas instruções servem para facilitar o gerenciarnento de dados e operações sobre variáveis, já que o tipo da variável só é conhecido em tempo de execução no SMALLTALK. Além disto, ao contrário do RISC I e II, possuia instruções para carga e armazenamento de múltiplos registradores, e não possuia endereçamento a byte, pois este tipo de dado não existe em SMALLTALK.

A tabela 3.1 apresenta de forma sucinta os principais parâmetros de comparação dos inicroprocessadores RISC I, RISC II e SOAR.

	SOAR	RISC I	RISC II
Tamanho da Instrução	32 bits	32 bits	32 bits
Nº de Registradores	72	78	138
Total de Instruções	20	31	31
Nº de Estágios Pipeline	3	2	3
Resolução do Processo (h)	1.5 µm	2 µm	1.5 µm

Tabela 3.1 - Microprocessadoi-es de Berkeley

3.2 - Descrição Geral

Entre 1984 e 1987 a SUN Microsystems definiu o SPARC [SUN87]. O SPARC [GAR88] é uma arquitetura aberta com várias implementações em silício, com compatibilidade binária assegurada através da obediência as definições propostas. Esta liberdade de arquitetura permite obter implementações em tecnologias tão diversas como: matriz de portas CMOS [NAM88a], CMOS totalmente personalizado [NAM88b], ECL bipolar [AGR88] e GaAs. Com esta filosofia conseguiu-se também que cada fabricante realizasse melhoramentos na microarquitetura adequados as suas implementações, sem perda de compatibilidade binária. A sua arquitetura simples permite ainda obter excelente desempenho, baixo custo de projeto, facilidade de fabricação e maior confiabilidade.

O SPARC é urna arquitetura RISC de 32 bits com "pipeline". Está dividido em duas partes: uma UNIDADE INTEIRA e uma UNIDADE DE PONTO FLUTUANTE. Cada uma destas unidades tem seu próprio conjunto de registradores, todos de ***32*** bits.

A Unidade Inteira pode conter de 40 a 520 registradores, em um total de 2 a 32 janelas de registradores e 8 registradores globais. As janelas de registradores se sobrepõem, permitindo passagem de parâmetros internamente a pastilha. A Unidade de Ponto Flutuante tem 32 registradores de 32 bits. Estes registradores podem ser acessados através de instruções de LOAD e STORE de até 2 palavras (64 bits). As

operações de Ponto Flutuante podem ser executadas concorrentemente com as instruções da Unidade Inteira, sendo que o sincronismo é determinado por sinais especiais entre as duas Unidades.

No SPARC estão definidas 58 Instruções Inteiras, 16 de Interface com coprocessador e 36 de Ponto Flutuante. Todas as instruções são de 32 bits em 3 formatos básicos. É uma arquitetura orientada a registrador, ou seja, as únicas instruções que fazem referência a memória são as instruções de LOAD e STORE. Seu conjunto de instruções é bastante simples e todas as operações complexas, tais como multiplicação, divisão, troca de contexto e indexação de vetores, devem ser feitas por "software".

O SPARC herdou do RISC I e II um esquema de janelas de registradores e a execução de desvios atrasados, e do SOAR instruções para suporte a linguagem orientada a objeto. Foram introduzidas extensões para suporte a multiprocessamento, para coprocessador de ponto flutuante, e para um coprocessador genérico.

3.3 - Conjunto de Registradores

O SPARC utiliza esquema de janelas de registradores para otimização do tempo gasto com chamadas de subrotinas. O diagrama da figura 3.1 ilustra o funcionamento deste sistema de janelas.

Neste exemplo, podemos ver que os conjuntos de registradores utilizados por duas rotinas adjacentes são sobrepostos parcialmente, ou seja, a rotina que faz a chamada e a que é chamada trocam parâmetros nos registradores que lhe são comuns. No esquema acima, os registradores físicos R98 a R105 são os de saída de parâmetros para a rotina B (R8-R15) e os de entrada para a rotina C (R24-R31).

A cada momento, existe um conjunto de 32 registradores ativos chamado de janela, que é dividido da seguinte maneira: 8 registradores são globais [RO-R7); oito são de entrada de parâmetros passados pela rotina anterior (R24-R31); oito são locais (R16-R23); e oito são de saída de parâmetros, passados para a rotina seguinte (R8-R15). Quando e feita uma chamada de rotina, os registradores lógicos de 8 a 31 são deslocados

de 16 posições no conjunto físico de registradores. Desta maneira, os registradores de saída da rotina anterior passam a ser os registradores de entrada da rotina cliamada, onde são lidos os parâmetros passados. No retorno da rotina, realiza-se o procedimento inverso.

Estas janelas são circulares, e quando não há mais registradores físicos para serem alocados, ocorre uma exceção e o processador desvia para urna rotina de gerenciamento que salva a primeira janela ocupada para a memória, deixando o espaço livre necessário para a janela da nova rotina.

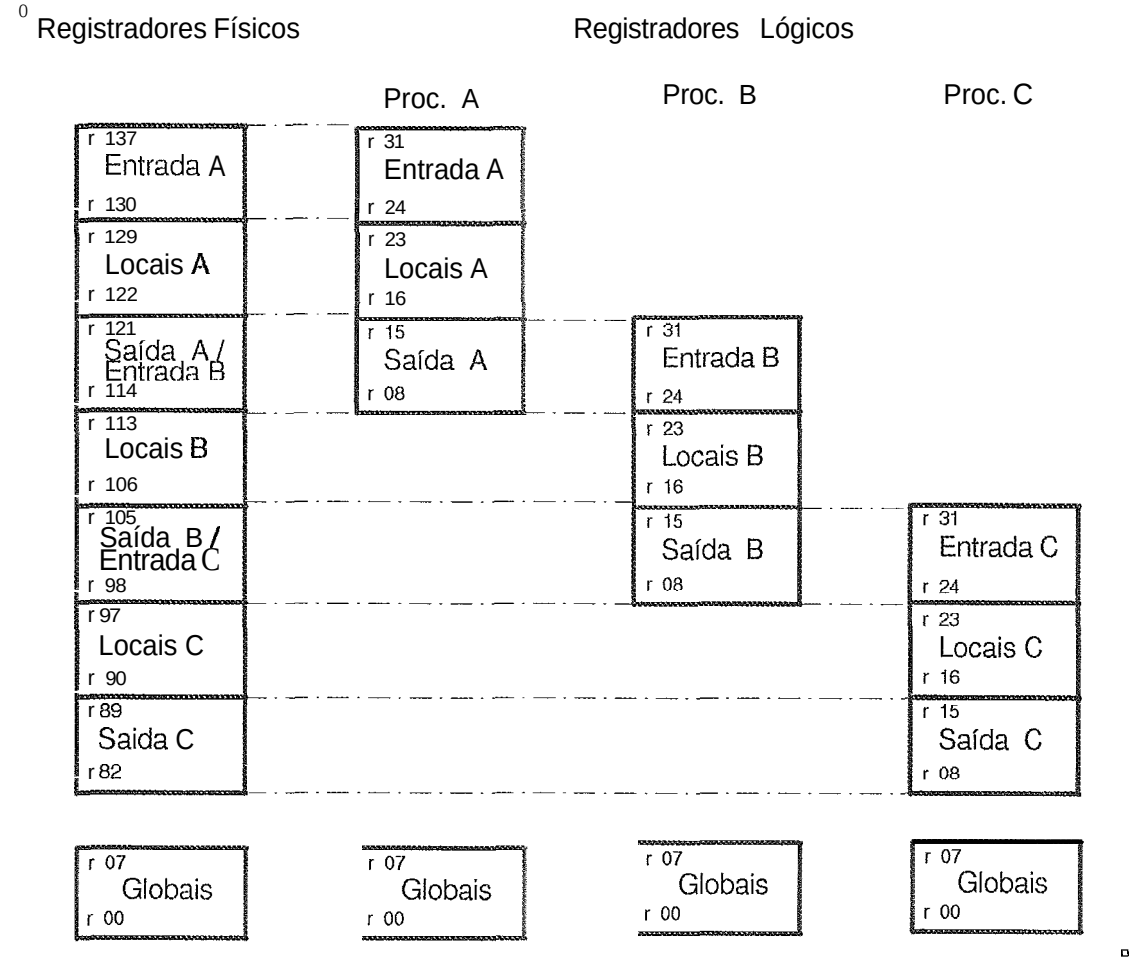


Figura 3.1 - Organização do Registradores do SPARC

3.4 - Pipeline de Instruções

O SPARC é definido como sendo uma arquitetura com "pipeline". Por isto, a maioria das instruções pode ser executada em um ciclo de máquina, com exceção das instruções de carga (LOAD), armazenamento (STORE), desvio absoluto (JMPL), retomo de exceção (RETT), de ciclos indivisível (LDSTUB, SWAP) e as de Ponto Flutuante.

A maioria das instruções de transferências de controle são atrasadas, ou seja, a instrução seguinte a instrução de desvio também pode ser executada, caracterizando o que é chamado de "desvio atrasado". A execução desta instrução é opcional, dependendo de um bit no código da instrução e do desvio ter sido tomado ou não. Normalmente os compiladores possuem uma fase de otimização que rearranja as instruções, colocando código útil após as instruções de desvio.

As dependências de dados que acontecem no "pipeline" são resolvidas pela lógica de controle. Normalmente isto é conseguido sem que haja atraso na execução das instruções, mas quando isto não for possível, o "pipeline" deve ser congelado até que os dados estejam disponíveis para a instrução seguinte. Esta operação é chamada de "interlock".

3.5 - Outras Características

No momento, o SPARC está sendo licenciado para diversos fabricantes, tendo versões em CMOS e ECL. Existem implementações com tecnologia CMOS com forma de projeto padronizada (matrizes de portas) e personalizada. As velocidades de "clock" obtidas são cerca de duas vezes maiores que as obtidas em microprocessadores CISC de tecnologia equivalente.

Não estão definidas na arquitetura características como: tipo e largura do barramento, gerência de memória, uso ou não de cache interna, número de estágios do "pipeline", tipo de lógica de controle e tecnologia de implementação. Para realizar um estudo sobre o desempenho da arquitetura nas diversas opções existentes, foi elaborado um simulador configurável, cuja descrição é feita no capítulo seguinte. A figura 3.2 mostra um diagama em blocos de uma possível organização para a arquitetura SPARC.

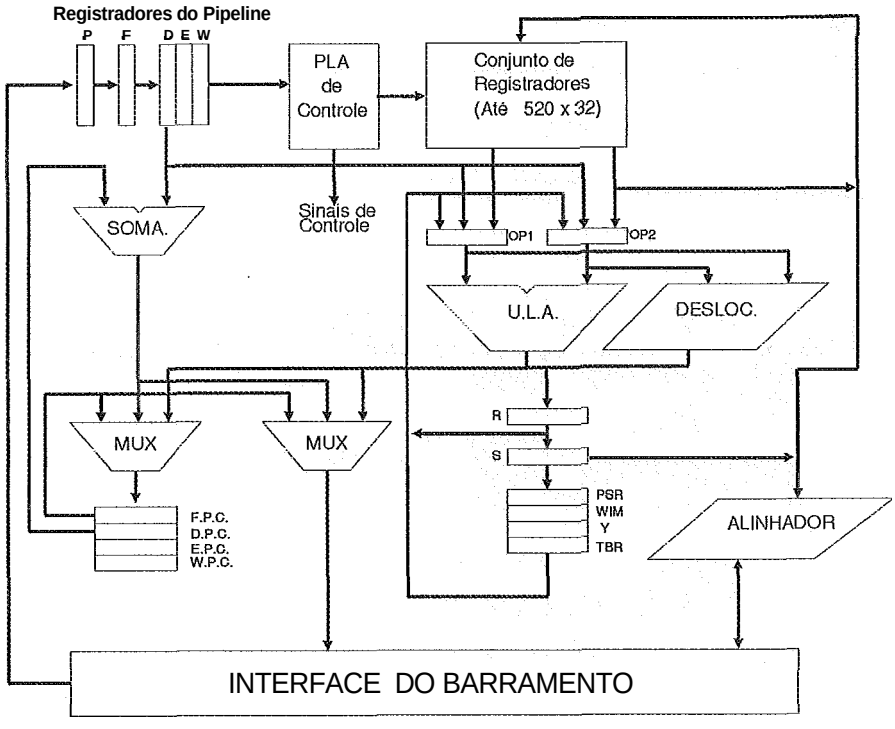


Figura 3.2 - Arquitetura do SPARC

Nesta figura podemos observar os registradores do "pipeline" de instruções (P,F,D,E,W). Normalmente os dois primeiros registradores não são utilizados, e a instrução buscada é carregada direto no registrador D. Contudo, se houver uma instrução que demore mais de um ciclo para ser executada, a instrução buscada é carregada no registrador F, ou no P, caso o primeiro também esteja ocupado. Em todo caso é garantido que estes dois registradores estarão vazios após a execução da instrução que demandou mais de um ciclo.

A arquitetura mostrada possui vários PCs, que são responsáveis pela recuperação do processamento normal caso haja alguma exceção. É necessário um PC para cada estágio do "pipeline", já que nada se pode garantir sobre a sequencialidade dos endereços das instruções que estão no "pipeline". O somador mostrado junto ao "pipeline" é responsável pelo cálculo do novo PC quando houver uma instrução de desvio sendo executada. Na Figura 3.2 estes PCs estão descritos como FPC, DPC, EPC e WPC, e não são visíveis diretamente pelo programador.

O conjunto de registradores da arquitetura acima possui *3* portas, o que permite 1 escrita e 2 leituras simultâneas. Esta simultaneidade é necessária porque este registrador é utilizado ao mesmo tempo pelo estágio de decodificação (busca de operandos) e pelo estágio de escrita (armazenamento do resultado). Após a busca dos operandos, os valores lidos são carregados em dois registadores (OP1 e OP2) para ser efetuada a operação aritmética adequada. O resultado da ULA é guardado no registardor de resultado R. No ciclo seguinte é copiado para o registrador S e então escrito no conjunto de registradores. Caso haja alguma dependência de dados entre o estágio de escrita e o estágio de decodificação, o valor do registrador R é imediatamente copiado para OP1 ou OP2, através de um caminho especial, para que não haja nenhum atraso no processamento.

Por causa da diversidade dos tipos de operandos escritos/lidos da memória, existe a necessidade de um alinhador entre a interface de barramento e o conjunto de registradores. Este alinhador também é responsável pela extensão do bit de sinal do dado lido, quando for necessário.

Na figura 3.2 ainda podem ser vistos 4 registradores especiais, que fazem parte do modelo de programação da máquina. O PSR contém uma cópia dos códigos de condição, um ponteiro para a janela de registradores atual, o nível da mascára de interrupção e bits de usuário/supervisor e habilitação de exceções. O registrador WIM contém a mascara que diz quais janelas estão válidas no conjunto de registradores. O registrador Y possui o resultado da instrução de Passo de Multiplicação, e pode ser lido e escrito pelo usuário. O TBR possui o endereço da base da tabela de vetores de exceção da máquina, permitindo que seja colocada em qualquer posição de memória.

4. O SIMULADOR SIMUS

4.1 - Descrição

A partir da escolha da arquitetura SPARC, foi desenvolvido um estudo minucioso de sua arquitetura e do funcionamento de suas instruções. Este estudo detalhou as unidades funcionais internas, registradores de trabalho, caminhos de dados e instruções. Com este detalhamento pronto, foi elaborado um mapa do funcionamento de cada instrução em cada estágio do "pipeline". Este mapeamento por sua vez exigiu mudanças na proposta inicial de arquitetura, em um processo iterativo que prosseguiu até que uma situação estável fosse atingida. Terminado este estudo, pôde-se então elaborar um simulador funcional para a arquitetura, denominado SIMUS.

O SIMUS é um simulador configurável dinamicamente, que simula o comportamento do SPARC a nível de "pipeline". Este simulador foi elaborado para orientar o projeto final da arquitetura compatível. 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 obter uma indicação para a melhor utilização da área disponível em silício.

Além destas facilidades, o SIMUS possui uma interface de usuário bastante sofisticada, que permite a alteração de qualquer registrador ou posição de memória. Podem ser examinados e alterados os conteúdos dos contadores de programa (PC's), dos códigos de condição, dos registradores do "pipeline" de instrução e dos registradores internos, mesmos os não disponíveis ao programador. Possui modos de execução passo a passo e direto, com possibilidade de inclusão de pontos de parada. São disponíveis ainda opções para carga e salvamento de programas objetos e armazenamento dos resultados da simulação.

A Figura 4.1 mostra uma esquematização da interface com o usuário do simulador. A seleção das opções para a configuração é feita dinamicamente, sem necessidade de arquivos de configuração, através de menus apresentados na tela para o usuário.

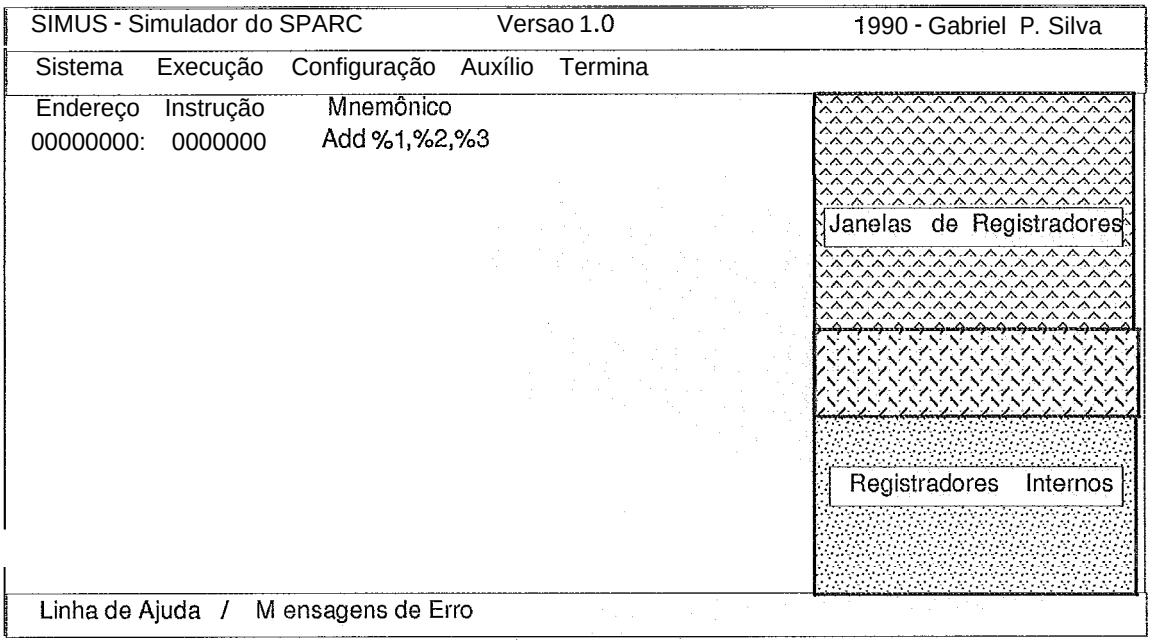


Figura 4.1 - Interface de Usuário

É objetivo do projeto que o programa seja portátil e possa estar disponível em outras instalações. Para tanto, foi implementado em turbo Pascal, versão 5.5, e possui módulos muito bem definidos, que permitem separar a interface com o usuário do restante do programa, facilitando o transporte do PC para outras máquinas hospedeiras.

O programa possui 6 unidades principais, sendo *3* para interface com o usuário e 3 para o simulador. A interface de usuário consumiu cerca de 2000 linhas de código, e o módulo de simulação em torno de 3600 Linhas de Pascal. Apesar disto, o programa compilado possui apenas 89 Kbytes de código objeto, com área de dados menor que 64 Kbytes, incluindo-se a memória do simulador, com 32 Kbytes. O trabalho de programação, depuração, compilação dos programas de teste, simulação e coleta de dados levou cerca de 8 meses para ser concluído.

Nas seções seguintes são vistas em detalhe as opções de arquitetura que podem ser avaliadas com este simulador.

4.2 - Parâmetros Simulados

4.2.1 - Número de Janelas

O dimensionamento do conjunto de registradores foi avaliado com vistas a se obter um tamanho mínimo, liberando área na pastilha para implementar outras facilidades. Rotinas específicas para gerenciar os casos de "overflow" e "underflow" foram codificadas, criando um ambiente mais realístico, já que seu custo também é considerado na execução dos programas.

Existem várias estratégias possíveis para o gerenciamento das janelas de registradores. Podemos salvar/restaurar duas ou mais janelas a cada ocorrência de uma exceção, realizando uma busca antecipada das janelas necessárias, ou mesmo elaborar algoritmos mais sofisticados de previsão. Entretanto, estudos anteriores realizados [TAM83] indicam que a melhor estratégia consiste em se mover apenas uma janela por vez, a cada ocorrência de uma exceção.

O número de janelas disponíveis na arquitetura pode variar entre 2 e 32. Como vimos anteriormente, o uso de um conjunto de registradores interno a pastilha diminui consideravelmente o trafégo no barramento, com conseqüente aumento no desempenho e diminuição da interferência com outros processadores, no caso de sistemas com múltiplos 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]. Se esta área for muito grande, pode comprometer a integração do projeto em uma única pastilha em determinadas tecnologias.

Outro fator limitante para o tamanho do conjunto de registradores, é que o SPARC é um microprocessador de uso geral, voltado para sistemas multiusuário e multitarefa. O tempo perdido no salvamento do contexto pode ser de grande impacto no desempenho, se a troca de tarefas for freqüente e houver um grande número de janelas ativas para serem salvas. Através da simulação pretendemos determinar um compromisso ótimo, entre desempenho e gasto de área.

4.2.2 - Largura e Tipo do Barramento

Uma das características importantes no desempenho de urna 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 existirem barramentos independentes, estas operações podem ser realizadas em paralelo, com consequente aumento de desempenho. A contrapartida e o gasto de área da periferia, que é de peso significativo na área total da pastilha. Em algumas Lmplementações RISC [KAT83], este gasto chega a ser de 40%. Entretanto, é uma opção que tem sido utilizada com maior freqüência nos processadores mais modernos, como o MC88100 e i80860, devido a alta tecnologia de integração disponível, e à necessidade de uma banda passante maior.

Com o uso de um barramento de 64 bits devem-se obter ganhos mais modestos, já que apenas uma instrução de 32 bits pode ser alimentada no "pipeline" a cada ciclo, e não há uso imediato para a banda passante disponível. Um barramento com esta largura é importante para instruções como "load" e "store" duplos, que lidam com operandos de 64 bits. Outras instruções que se beneficiam deste barramento são as instruções para acesso aos registradores da unidade de ponto flutuante, desde que a mesma possua um barramento com largura idêntica. Entretanto, é uma opção mais barata em termos de área que a utilização de barramentos separados para dados e instruções.

4.2.3 - Cache de Desvio

O uso de cache interna é um dos meios mais eficientes para aumentar o desempenho do processador [HIL84], mas o gasto de área necessário para implementá-la dentro da pastilha é bastante grande. Várias Lmplementações do SPARC optaram pela utilização de uma cache externa de dados e/ou instruções, com esquema de entrelaçamento, em

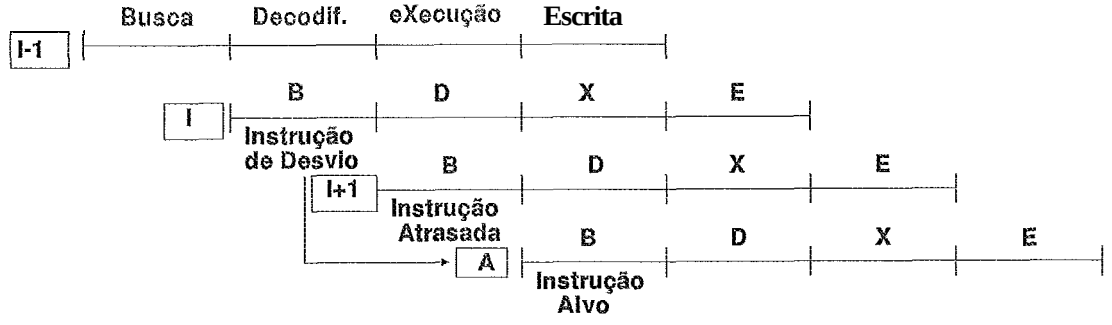
que o endereço é fornecido em um ciclo e os dados são lidos no seguinte. Isto dá tempo ao circuito de controle para verificar se houve acerto ou falha no acesso ao cache. Consegue-se assim trabalhar sem estados de espera, apesar das altas frequências de "clock" envolvidas, da ordem de 40 Mhz para as pastilhas CMOS.

O esquema que iremos 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, cada uma composta por uma instrução alvo, endereço de desvio e código de condição. O tamanho de cada linha pode variar 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.

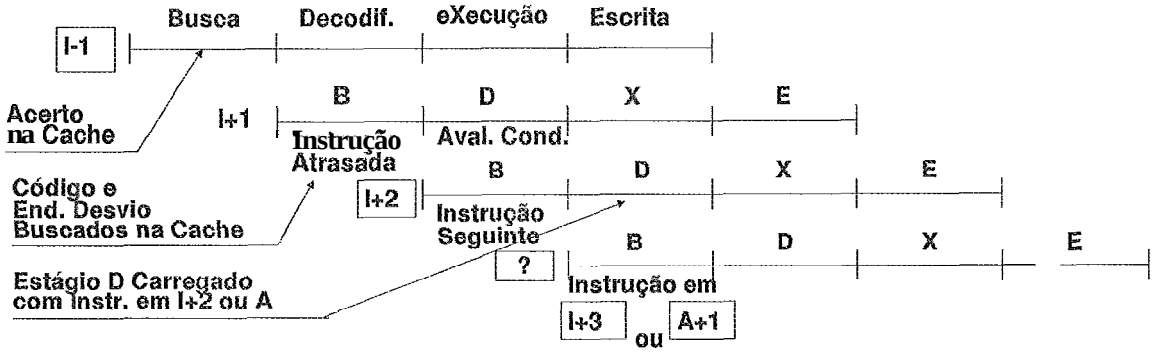
A execução de qualquer instnição é feita em tantos ciclos quanto forem os estágios do "pipeline", mas uma nova instnição pode ser buscada a cada ciclo. Desta forma, o custo de execução de cada instrução é de um ciclo. Contudo, se uma instrução necessitar de mais ciclos para ser executada, diz-se que o custo dela é igual ao número de ciclos em que uma nova instrução não pode ser carregada no "pipeline". Isto acontece em instruções como "load" e "store".

Considerando-se o uso de um "pipeline" de 4 estágios sem cache de desvio, a execução de um desvio condicional se processa da seguinte maneira: durante o estágio de busca, a instrução é trazida da memória, no endereço I. No ciclo seguinte, é feita a decodificação da instrução, quando então descobre-se que é um desvio condicional e, dependendo da implementação, já pode ser feita a avaliação dos códigos de condição. Ainda neste ciclo é feita a busca da instrução seguinte, chamada de instrução atrasada, no endereço I+1. No ciclo seguinte, de posse do resultado da avaliação, a busca da instrução vai ser feita no endereço I+2, se for falso, ou A, se for verdadeiro. Nesta situação não se gasta nenhum ciclo adicional para avaliação dos códigos de condição, e o custo de execução do desvio condicional é de um ciclo. Em algumas implementações este custo é maior, pois não se consegue fazer a avaliação dos códigos de condição no estágio de decodificação, mas apenas no de

execução. Neste caso o custo de um desvio condicional é de dois ciclos. Entretanto, a melhor situação ocorre quando utilizamos uma cache de desvio.



Execução de um Desvio Atrasado em um Pipeline de 4 Estágios



Execução da um Desvio Atrasado com Cache de Desvio

Figura 4.2 - Operação da Cache de Desvio

Na cache de desvio estão armazenados o "tag", que é o endereço da instrução anterior ao desvio (I-1), os códigos de condição da instrução de desvio (em I), o endereço da iiistrução alvo (A) e a instrução alvo (em A), para onde pode ser transferido o controle. Se houver um acerto na cache, a busca da instrução de desvio (em I) é substituída pela busca da instrução "atrasada" (em I+1). No ciclo seguinte, os códigos de condição da instrução que está na cache são avaliados em paralelo com a execução da instrução atrasada e a busca da instrução seguinte (em I+2). Se o código de condição for avaliado falso, o processamento prossegue normalmente, com a busca da instrução em I+3 no ciclo seguinte. Se entretanto for avaliado verdadeiro, no ciclo seguinte o estágio de

decodificação será carregado com o conteúdo da instrução alvo que está na cache, a busca das instruções prossegue no endereço **A+1**, e a cache é carregada com informação do desvio tomado. Em ambos os casos, o desvio deixa de ter custo um e passa a ter custo zero, pois sua busca e avaliação é feita em paralelo com a busca e execução da instrução atrasada. A Figura 4.2 ilustra o efieto da cache de desvio na execução de um desvio condicional.

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.

4.2.4 - Conjunto de Instruções

Uma das opções de configuração existentes no SIMUS é a carga de um arquivo de definição, que indique algumas instruções como não implementadas. Apesar de definidas internamente, o simulador considera estas instruções como não implementadas, e dá início a um procedimento de exceção idêntico a uma instrução não definida. Assim, pode-se associar um custo a emulação destas instruções por "software".

Para a implementação a ser realizada estamos considerando não incluir as intruções do tipo TAGGED e as instruções MULScc. As primeiras, emboras originárias do SOAR [UNG84], não foram utilizadas na implementação do SMALLTALK correntemente em uso nas estações da SUN [MUC88]. A segunda é utilizada somente para multiplicação entre variáveis, envolvendo até 32 passos de programação. Como forma de otimização, estamos considerando o uso da Unidade de Ponto Flutuante para a execução destas instruções. Conforme os resultados da simulação isto pode se confirmar ou não. Todas as demais instruções deverão ser implementadas, incluindo as de exclusão mútua.

4.3 - Outras Considerações

Para a implementação de algumas destas opções, modificações profundas tiveram que ser feitas na arquitetura inicial. Enquanto que uma arquitetura com barramento compartilhado tem apenas quatro

estágios no "pipeline", uma com barrarnento Harvard exige um estágio a mais no "pipeline", para que possa haver melhoria no desempenho. O uso de uma cache de desvio e de um barramento de 64 bits também exige modificações na unidade de busca de instruções. Estas modificações foram feitas mantendo-se a compatibilidade binária.

A implementação do NCE/UFRJ tem uma característica particular para lidar com as dependências de dados no "pipeline". Na arquitetura SPARC estas dependências só ocorrem entre o estágio de decodificação e os demais estágios da máquina, já que este é o único estágio a ler os registradores. Se a instrução que estiver no estágio de decodificação precisar de um operando que esteja sendo escrito no conjunto de registradores, não há problema, pois a implementação do NCE/UFRJ prevê leitura e escrita simultânea nos registradores. Se o dado necessário estiver no registrador de resultado da ALU, o controle fornece o dado através de um caminho especial de "bypass", direto da saída da ALU para os registradores do estágio de decodificação. Mesmo na definição da máquina com 5 estágios de "pipeline", este "bypass" sempre é possível. Desta maneira, os únicos casos de dependência do "pipeline" ocorrem quando o estágio de decodificação aguarda um dado que vai ser carregado em um registrador pela instrução de "load". Nestes casos, o "pipeline" é congelado até que o dado esteja sendo escrito no conjunto de registradores, quando então pode ser lido pelo estágio de decodificação.

Pretendemos observar nas simulações os seguintes parâmetros:

- Número total de ciclos executados.
- Número total de instruções executadas.
- Ciclos/Instrução: número médio de ciclos para execução de uma instrução.
- Número médio de janelas ativadas.
- Maior valor de janela ativada.
- Total de acessos a dados e instruções, de usuário e supervisor.

- Total de vezes que cada uma das instruções foi executada.

- Total das ocorrências de "interlock".

Existem duas situações em que o comportamento do simulador não é idêntico ao do microprocessador: quando da ocorrência de uma chamada ao sistema e quando da existência da 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 [KLEI88]. Em nossa simulação não estão previstos os custos de troca de contexto, mas este gasto pode ser estimado a partir do níimero médio de janelas ativas e do tempo médio de execução de uma tarefa, este último especifico para cada sistema.

5. EXPERIÊNCIAS E ANÁLISE DOS RESULTADOS

5.1 - Os Programas de Avaliação

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

Para melhor caracterização dos programas utilizados levantou-se um histograma das instruções executadas por cada programa. No levantamento destes histogramas foi considerada uma arquitetura hipotética, em que não existe a necessidade de salvamento de janelas na memória, ou seja, há um número infinito de janelas. Entretanto, para o restante das simulações realizadas, considerou-se uma arquitetura padrão, com barramento compartilhado de dados e instruções, oito janelas de registradores e sem cache de desvio.

O programa Quicksort ordena um vetor de 4 Kbytes preenchido pseudo-aleatoriamente. Este programa possui chamadas recursivas de melhor caso O(NlogN), sendo que a profundidade de chamada de rotinas alcançada foi de 19 com a massa de dados utilizada.

A Figura 5.1 mostra o histograma das instruções executadas pelo Quicksort. Por este histograma verificamos que há cerca de 27% de desvios, 2% de chamadas/retornos de rotinas, 18% de acessos a memória e o restante de operações aritméticas. Este valores vão justificar o comportamento do programa em cada uma das configurações avaliadas. Por exemplo, o grande número de instruções de desvio indica que o uso de uma cache de desvio deve resultar em um aumento de desempenho. Mas já o tamanho do conjunto de registradores deve ser de pouca influência no desempenho final, dado o pequeno percentual de chamadas de rotinas no total de instruções.

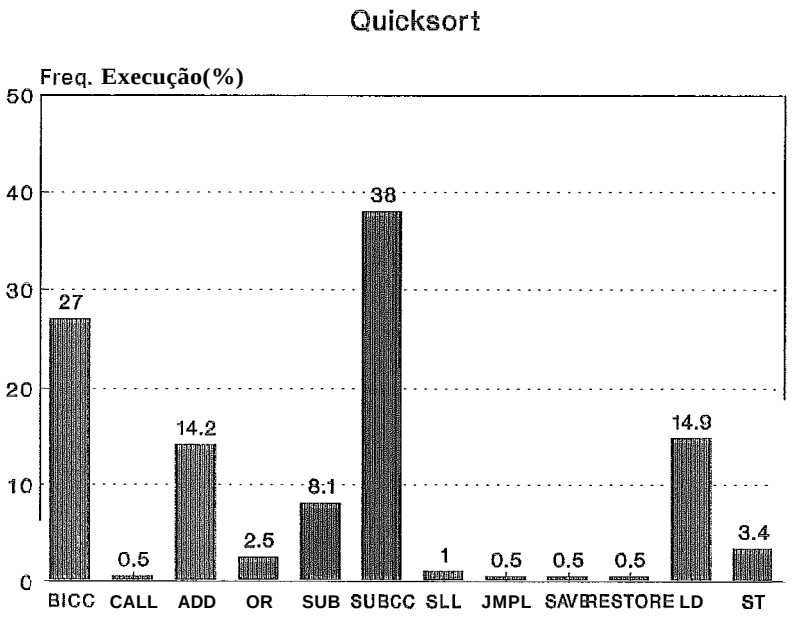


Fig 5.1 - Histograma de Instruções do Quicksort

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 5.1 - Simulação do Quicksort

A Tabela 5.1 mostra os resultados para a simulação do mesmo programa com uso da arquitetura padrão. Nos dados apresentados, as instruções de supervisor representam o total de acessos para busca de Instruções das rotinas de gerenciamento das janelas. As instruções de usuário constituem as instruções buscadas para a execução do programa. O total de acessos de leitura ou escrita mostram quantas operações foram efetivamente realizadas no barramento, já que algumas delas envolvem mais de um ciclo. Ou seja, a soma destes dois valores não expressa o total de ciclos. Da mesma forma o total de instruções buscadas é maior que o total de instruções executadas. A diferença é o total de instruções anuladas por desvios, exceções e "interlocks".

Na parte de estatísticas da CPU observamos que o maior valor de janelas ativadas é igual a 19, que corresponde a altura da árvore de chamadas de rotinas, e o valor médio de janelas ativadas é 7.3. A grande diferença entre estes dois valores indica a necessidade de um número maior de registradores implementados para evitar exceções. A média de ciclos por instrução é o total de ciclos gastos para execução do programa dividido pelo número de instruções realmente executadas. O valor 1.44 é típico para estimativas anteriores da arquitetura [Nam88a]. Neste programa observamos 15923 "interlocks", ou cerca de 9% das instruções buscadas. É um ponto fraco da arquitetura, pois o compilador não reescalona as instruções para evitar este tipo de ocorrência.

O programa Sieve gera os primeiros 1899 números primos manipulando um vetor de 16 Kbytes. Primeiramente o vetor é inicializado com l'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. Neste, como em outros programas, o número de iterações do "loop" principal foi reduzido, sem invalidar o resultado das simulações, de modo a permitir tempos de simulação menores. A Figura 5.2 mostra um histograma de instruções e a Tabela 5.2 os dados obtidos na simulação da arquitetura padrão.

Este programa tem apenas 7 instruções no seu histograma, mostrando que podem ser necessários poucos tipos de instruções para execução de um programa de usuário. Há cerca de 25% de desvios, 18% de acessos a dados na memória e o restante de operações aritméticas. Não há chamada de subrotinas.

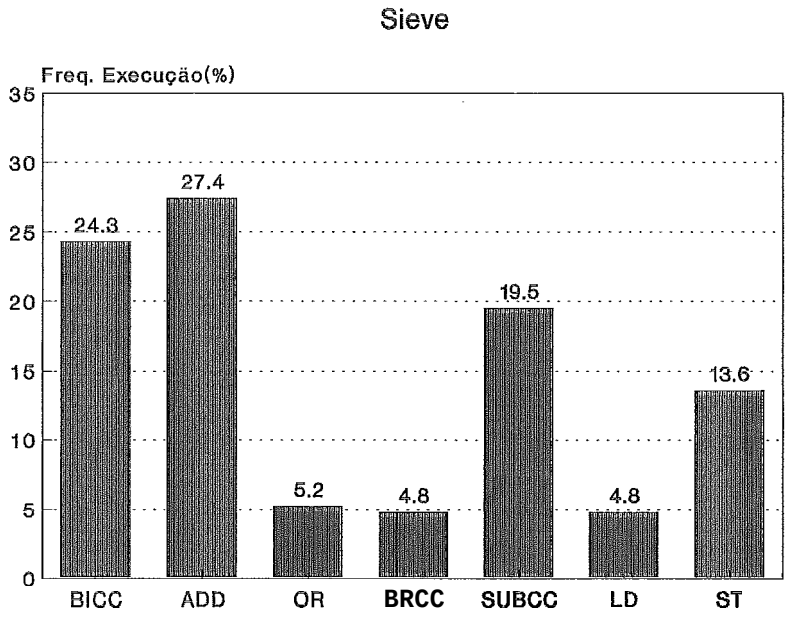


Fig 5.2 - Histograma de Instruções do Crivo de Eratóstenes

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	=	1O
Instr. Usuário	=	182234	Maior Valor Jan.	=	0
Instr. Supervisor	=	0	"Interlocks"	=	0

Tabela 5.2 - Simulação do Crivo de Esatóstenes

A Tabela 5.2 mostra uma média de 1.39 ciclos por instrução e nenhuma instrução de supervisor, pois não há chamadas/retornos de subrotinas, e a média de janelas ativadas é igual a 1. Mesmo com o grande número de acessos a dados, não houve ocorrência de "interlocks" no código gerado pelo compilador, talvez pelo fato de a maior parte dos acessos a dado serem de escrita.

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, fazendo apenas a soma dos resultados devolvidos pelas duas recursões seguintes, além de urna comparação para verificar se a recursão já chegou ao primeiro termo da série.

Pelo histograma da Figura 5.3 podemos verificar que não são executadas instruções de "load" ou "store". Este programa é bastante sujeito a influência do número de janelas presentes na arquitetura, pois executa um grande número de chamadas de rotinas, gerando a necessidade de se efetuar várias vezes a busca e salvamento de janelas. Como as rotinas elaboradas possuem um peso computacional muitas vezes maior do que o próprio programa, por utilizarem um grande número de instruções de "load" e "store", o tempo de execução do programa ficará bastante comprometido se o número de janelas for pequeno.

Pode ser visto pela Tabela 5.3 que o maior valor de janelas ativadas é de 19. Este valor é relativamente alto, como ressalvado em estudos anteriores [TAM83] e expressa o alto percentual de "calls" e "jmpls" encontrados no seu histograma. Não há "interlocks" pois não há acessos de usuário a dados na memória, e somente nos acessos de carga de registradores é que pode haver "interlock" na arquitetura. Tudo o que o programa precisa é uma semente inicial e todos as operações são efetuadas em registradores. Nesta simulação, mesmo com 8 janelas implementadas, pode-se observar o alto número de instruções de supervisor das rotinas de gerenciamento de janelas. Nesta situação, os únicos acessos de dado a memória são os de supervisor.

O programa Dlirystone [REI84] é um programa de avaliação sintético que possui um conjunto de instruções típico de uma aplicação inteira. O número de iterações é definido no programa original como

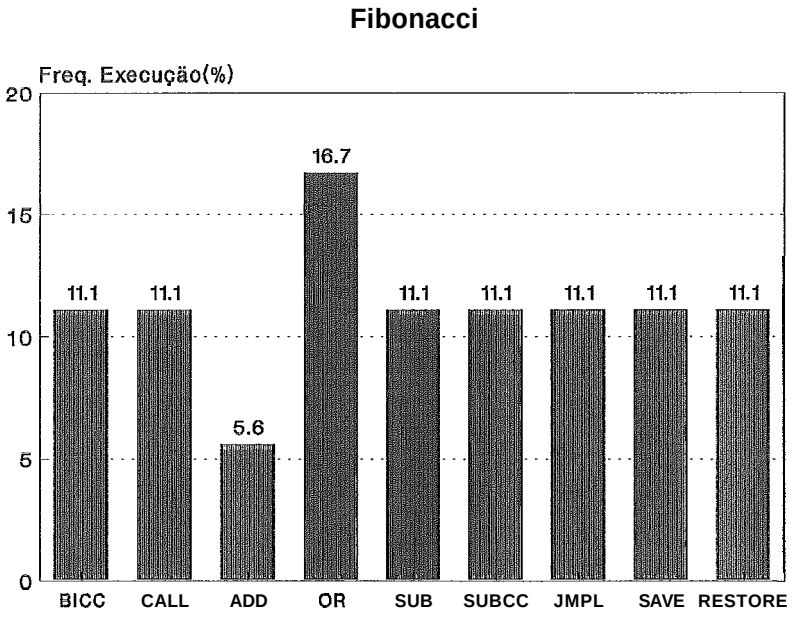


Fig 5.3 - Histograma de Instruções do Fibonacci

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/Instiuções	=	1.34
Dados Supervisor	=	8352	Média Jan. Ativas	=	13.4
Instr. Usuário	=	55191	Maior Valor Jan.	=	19
Instr. Supervisor	=	16129	"Interlocks"	=	0

Tabela 5.3 - Simulação do Fibonacci

50000. Na versão implementada foram utilizadas apenas 256 iterações. 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%

Cerca de 100 sentenças diferentes são dinamicamente executadas. O programa é balanceado com relação a três aspectos:

- tipo de comando
- tipo de operando (para dados simples)
- acesso ao operando: operando global, local, parâmetro ou constante.

A combinação destes três aspectos é balanceada apenas aproximadamente. O programa não calcula nada significativo, mas é sintática e semanticamente correto. A Figura 5.4 apresenta um histograma das instruções executadas para uma arquitetura com número infinito de janelas implementadas e a Tabela 5.4 mostra os dados da simulação para a arquitetura "padrão".

Pelo histograma podemos observar um percentual de cerca de **13%** para os desvios, 7% de chamadas/retornos de rotinas, 35% para acessos de dados a memória e o restante para operações aritméticas. 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".

Pela Tabela 5.4 podemos observar que a média de janelas ativadas e o maior valor de janelas ativadas são bem próximos indicando a necessidade de um menor número de janelas implementadas. Como resultado, o programa executa com apenas 8 janelas sem necessidade de

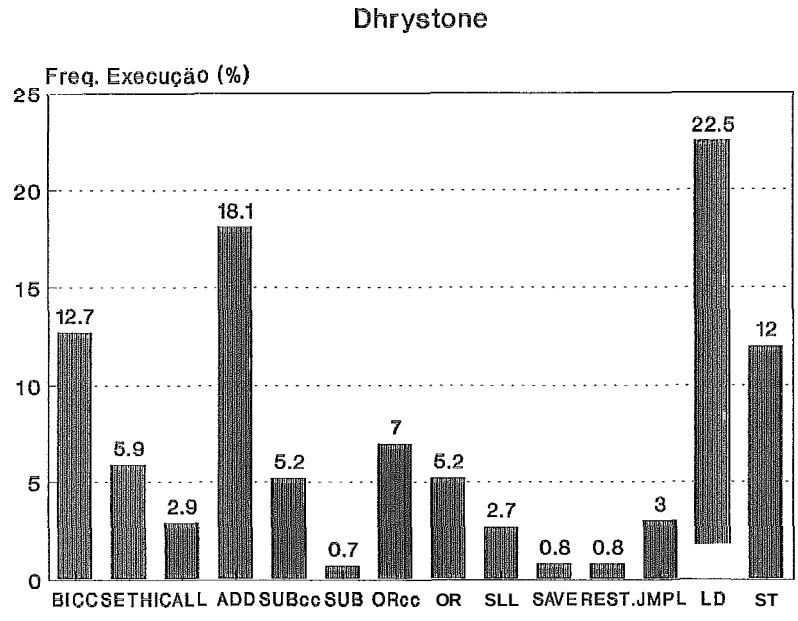


Fig 5.4 - Histograma de Instruções do Dhystone

RESULTADOS DA SIMULAÇÃO
DWRYSTONE

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 5.4 - Simulação do Dhrystone

salvamento/recuperação de janelas de/para a memória. Pode-se observar também que cerca de *3%* das instruções buscadas foram anuladas por causa de "interloclcs" no pipeline.

O programa Towers irnplementa 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 "loads" ou "stores" e a profundidade de chamadas de rotinas alcançada é 20 e possui uma árvore de chamadas diferente da do Quicksort e do Fibonacci. Durante as primeiras simulações, foi observado que os programas Fibonacci e Quicksort, apesar de serem bastante diferentes, apresentavam a mesma curva de "taxa de acerto de janelas". Por este motivo foi utilizado este programa, e apenas neste caso, para assegurar a validade dos resultados obtidos. Ao contrário dos demais programas, não foi codificado em "C", sendo escrito em "assembler". O histograma de instruções executadas pode ser visto na Figura 5.5 e outros dados da simulação da arquitetura padrão são encontrados na Tabela 5.5.

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 níimero de sequências de "calls" e "returns". 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.

5.2 - Os Resultados *das* Simulações

Os resultados das simulações com variação de parâmetros da arquitetura são mostrados nas Figuras 5.6 até 5.15. São avaliadas a variação do tamanho do conjunto de registradores (níimero de janelas implementadas),o tamanho da cache de desvio (número de entradas) e o tipo de organização do barramento.

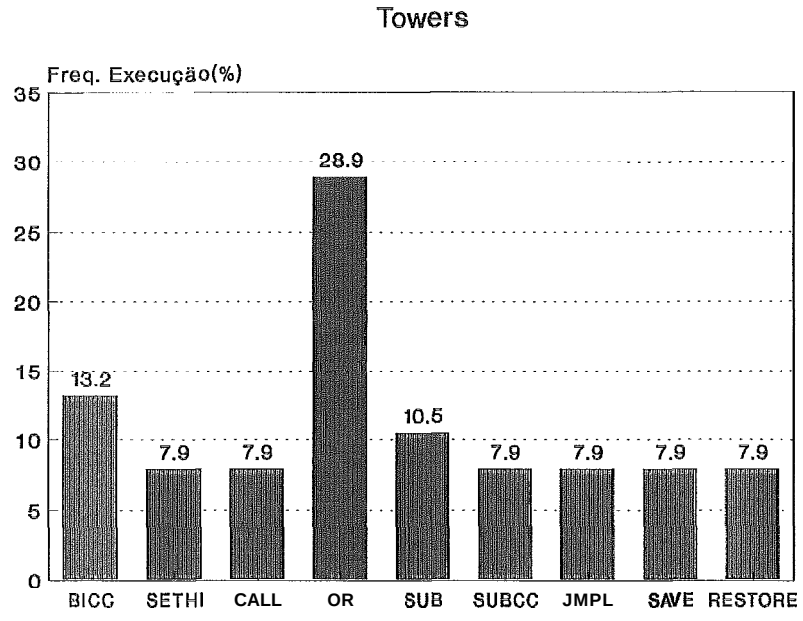


Fig 5.5 - Histograma de Instruções do Torres de Hanoi

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 5.5 - Simulação do Torres de Hanoi

As Figuras 5.6 e 5.7 mostram os efeitos da variação do tamanho do conjunto de registradores nos programas Fibonacci, Quicksort e Towers. Estes programas foram escolhidos por fazerem uso siginificativo de chamadas de rotinas. A "taxa de falhas" da Figura 5.6 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 janelas e diminuição no desempenho final do programa. Esta perda vai depender do peso das instruções de "save" e "restore" no total de instruções executadas. Assim, podemos ver na Figura 5.7 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 5.8 e 5.9 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). As Figuras 5.10 e 5.11 apresentam os resultados para os programas Quicksort, Dhrystone, Fibonacci e Sieve com a cache de desvio organizada com mapeamento direto.

As Figuras 5.8 e 5.10 apresentam 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 nas Figuras 5.9 e 5.10. A taxa de acerto na cache vai determinar um maior ou menor número de ciclos para o programa ser executado, dependendo do percentual de desvios no total de instruções executadas. Por sua vez, para um dado tamanho de cache, a taxa de acerto será maior ou menor em função do total, estático, de instruções de desvio existentes no programa. Avaliando o código de cada programa foi verificado que o programa Fibonacci possuía o menor número de instruções de desvio (2), seguido do Sieve(8), do Quicksort(13) e pelo Dhrystone, que possui o maior número, com 38 desvios condicionais. O histograma de instruções mostra que o valor dinâmico é de 11% de desvios para o Fibonacci, 13% para o Dhrystone, 24% para Sieve e 27% para o Quicksort.

Por causa do pequeno número de instruções de desvio, somente o programa Dhrystone foi simulado com diversas opções de associatividade, já que os outros programas se mostraram insensíveis a esta variação de parâmetro.

A Figura 5.12 compara uma configuração da arquitetura utilizando um barramento único para dados e instruções com outra com barramentos independentes (Harvard). Ambas as configurações não possuíam cache de desvio e possuiam 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 relação ganho/percentual de "loads" e "stores" é vista na Tabela 5.6. Em realidade, o programa Fibonacci só apresentou ganho devido a existência de "loads" e "stores" na rotina de salvamento de janelas.

Programa	Ganho	Load+Store(%)
Fibonacci	1.06	0.0
Quicksort	1.15	18.3
Sieve	1.15	18.4
Dhrystone	1.29	44.5

Tabela 5.6 - Relação Ganho Harvard/Percentual Load+Store

A Figura 5.13 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 foram obtidos 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 chamadas às rotinas de gerenciamento de janelas, que que utilizam "loads" e "stores" de 64 bits para recuperação e salvamento dos registradores, justificando assim o ganho obtido.

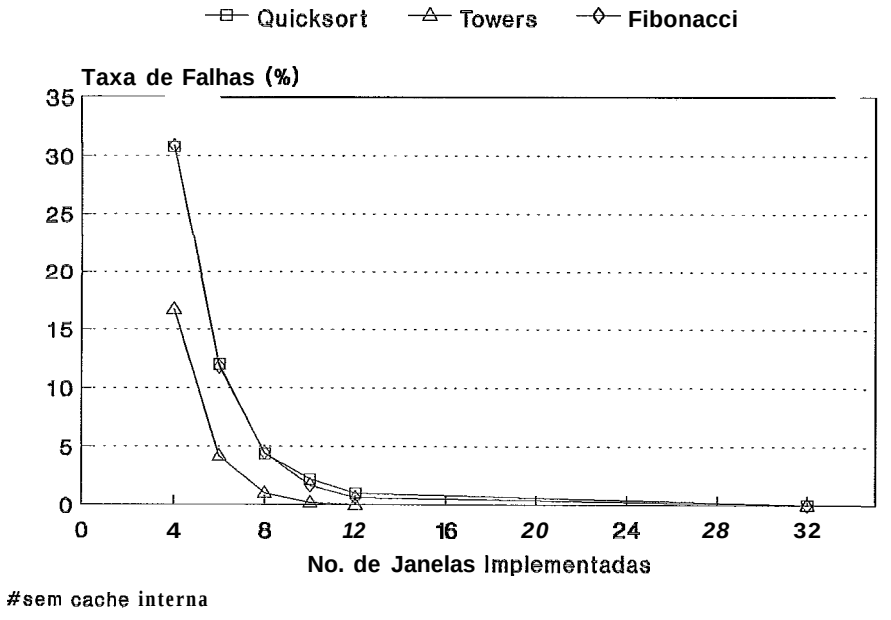


Fig 5.6 - Taxa de Falhas versus Número de Janelas (FQT)

	Taxa de Falhas		
Total de Janelas	**Quicksort**	**Tower**	**Fibonacci**
4	30.8	16.7	30.9
6	12	4.2	11.8
8	4.4	1	4.5
10	2.2	0.2	1.7
12	1	0	0.6
	0	0	0

Tabela 5.7 - Taxa de Falhas vs Número de Janelas (FQT)

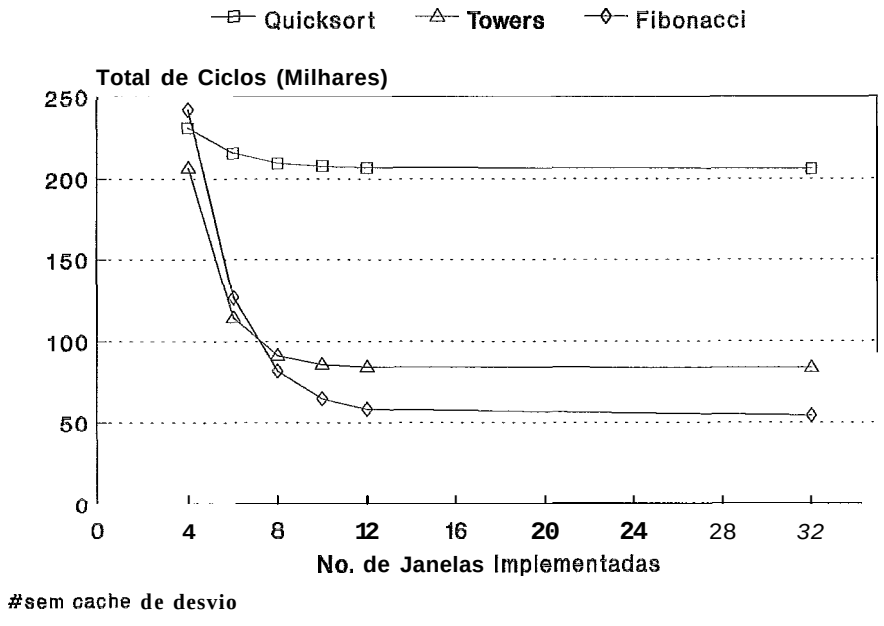


Fig 5.7 - Total de Ciclos versus Número de Janelas (FQT)

	Total de Ciclos		
Total de Janelas	**Quicksort**	**Tower**	**Fibonacci**
4	231217	206509	242223
6	215779	114493	126939
8	209546	91453	81992
10	207752	85741	64745
12	206800	84313	58200
32	205964	83953	54270

Tabela 5.8 - Total de Ciclos vs Número de Janelas (FQT)

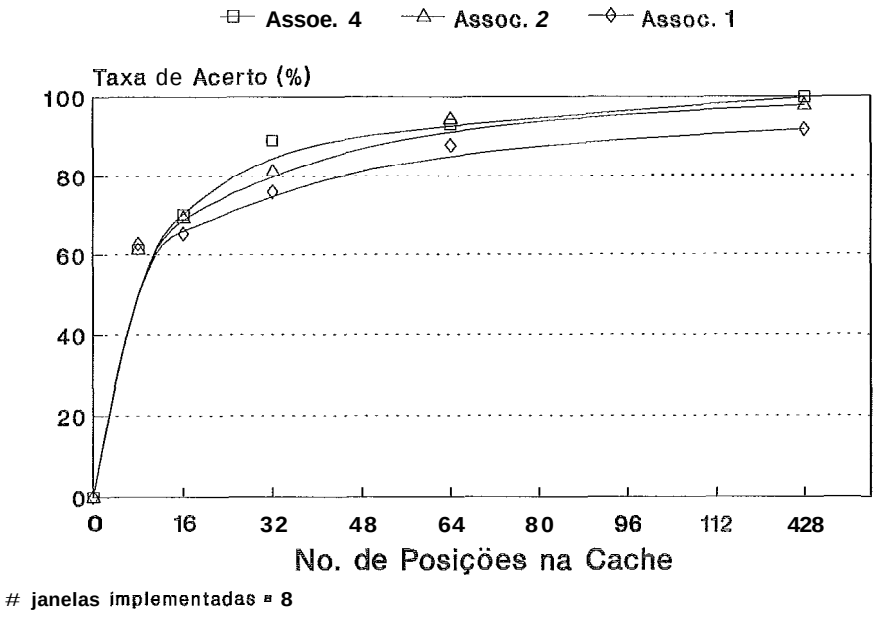


Fig 5.8 - Taxa de Acertos vs Tamanho da Cache - Dhrystone

Dhrystone Taxa de Acertos			
Tam. da Cache	**Assoc. 4**	**Assoc. 2**	**Assoc. 1**
0	**0**	0	**0**
8	61.5	61.4	62.8
16	70.2	69.4	65.4
32	89.2	81.3	76
64	93.2	94.5	87.9
128	99.8	97.9	91.9

Tabela 5.9 - Taxa de Acertos vs Tamanho da Cache (D)

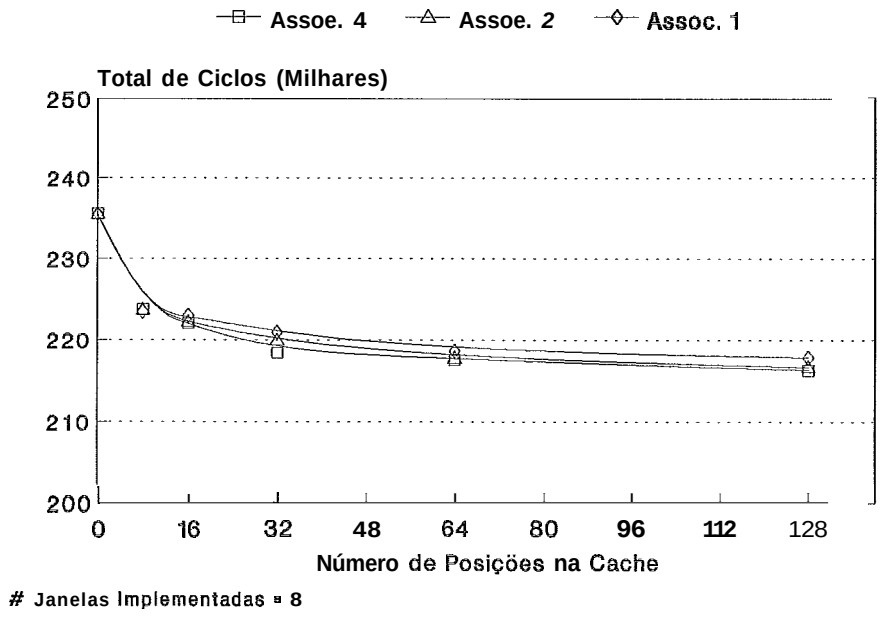


Fig 5.9 - Total de Ciclos vs Tamanho da Cache - Dhrystone

	Dhrystone Total de Ciclos		
Tam. da Cache	**Assoc. 4**	**Assoc. 2**	**Assoc. 1**
0	235537	235537	235537
6	223698	223699	223448
16	222023	221175	222936
32	218375	219875	220895
64	217579	217724	218600
128	216304	216686	217835

Tabela 5.10 - Total de Ciclos vs Tamanho da Cache (D)

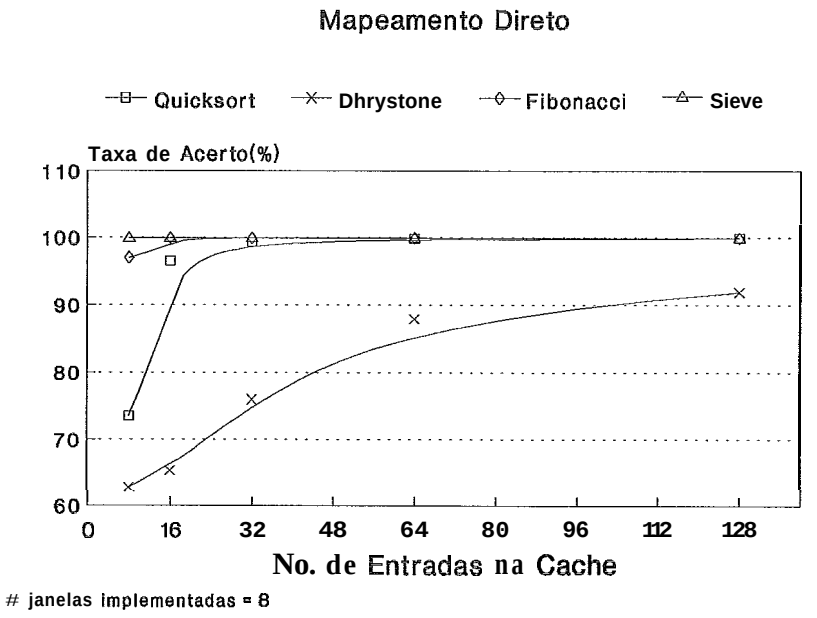


Fig 5.10 - Taxa de Acerto por Tamanho da Cache (FDQS)

Todos - Mapeamento Direto
Taxa de Acertos

Tam. da Cache	Quicksort	Dhrystone	Fibonacci	Sieve
0	0	0	0	0
8	73.6	62.8	97	99.9
16	96.5	65.4	99.9	99.9
32	99.3	76	99.9	99.9
64	99.8	87.9	99.9	99.9
128	99.9	91.9	99.9	99.9

Tabela 5.11 - Taxa de Acertos vs Tamanho da Cache (FDQS)

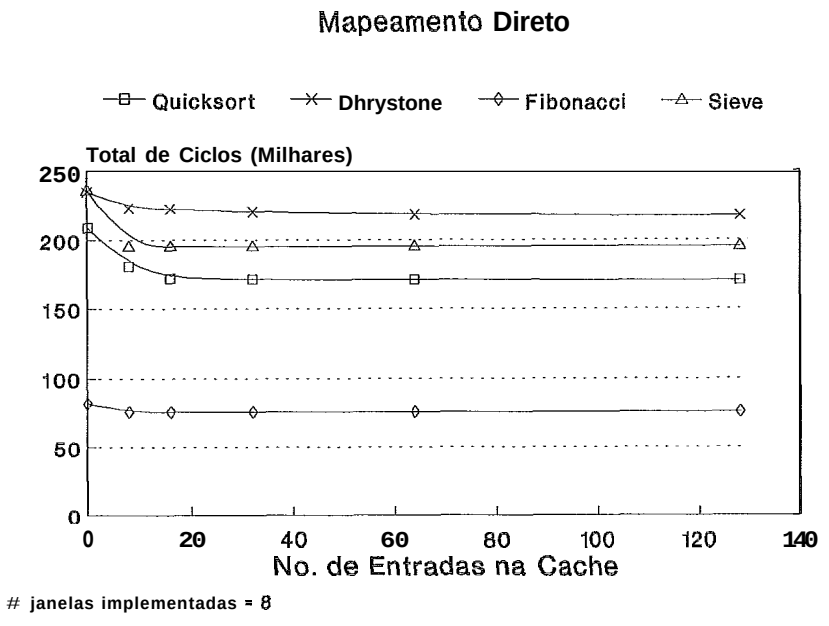
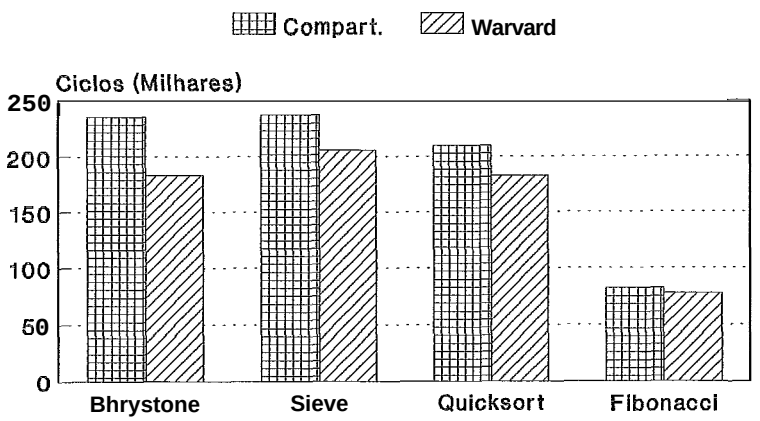


Fig 5.11- Total de Ciclos por Tamanho da Cache (FDQS)

Todos - Mapeamento Direto
Total de Ciclos

Tam. da Cache	Quicksort	Dhrystone	Fibonacci	Sieve
0	209546	235537	81992	236806
6	180980	223448	76078	195349
16	172115	222936	75902	195349
32	171296	220895	75902	195349
64	170802	218600	75902	195349
128	170753	217835	75902	195349

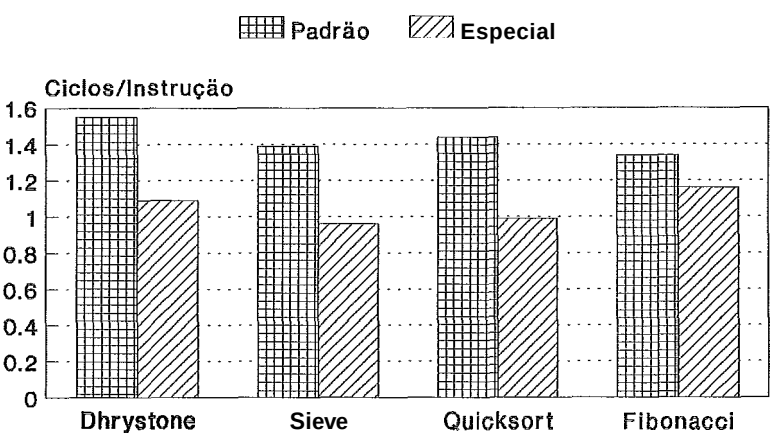
Tabela 5.12 - Total de Ciclos vs Tamanho da Cache (FDQS)



	Bhrystone	Sieve	Quicksort	Fibonacci
Compart.	235.537	236.806	209.546	81.992
Harvard	183.108	205.425	182.683	77.352
Ganho	1.29	1.15	1.15	1.06

janelas implementadas = 8
Cache de desvie com 0 entradas

Fig 5.12 - Barramento Harvard vs Barramento Compartilhado



	Dhrystone	Sieve	Quicksort	Fibonacci
Padrao	1.55	1.39	1.44	1.34
Especial	1.09	0.96	0.99	416
Qanho	1.42	1.45	1.46	1.16

Arq. Esp.: Cache 64 pos/ Harvard
Ambas: # janelas implem. = 8

Fig 5.13 - Ganho da Arquitetura Especial

5.3 - Considerações de Custo

Na seção anterior foram avaliados os ganhos que podem ser obtidos com variações na irnplementação da arquitetura do SPARC. Nesta seção serão feitas estimativas dos custos envolvidos na implementação de cada urna destas alternativas. Não dispomos de dados exatos sobre o total de área gasto por cada uma das alternativas de implementação analisadas na seção anterior. Entretanto, baseando-se no relato de irnplementações anteriores de microprocessadores RISC, podemos fazer algumas estimativas.

Implementações anteriores em CMOS apresentam um gasto de área entre 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 adicional 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 a arquitetura "padrão", entre 50% e 60%. Se fizermos a média dos ganhos obtidos na execução dos programas de teste, obtemos o valor de 38% em relação a arquitetura padrão, o que é um valor bastante significativo.

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 por exemplo, 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 "stores".

- A associação de uma cache de desvio a uma arquitetura com barramento Harvard 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 Capítulo 4.

Foi também observada a ocorrência de muitos "interlocks" durante a execução dos programas Dhrystone **(3.3%)** e Quicksort (9.2%), indicando principalmente a necessidade de algum nível de otimização a nível de compilador, com reescalonamento das instruções a fim de minimizar a ocorrências destas dependências de dado no pipeline.

Finalmente, deve-se notar a pequena quantidade de instruções necessárias para execução dos programas. Mesmo em um programa mais complexo como o Dhrystone, apenas 14 instruções são necessárias para sua execução. Isto vem reforçar a já comprovada eficiência dos processadores RISC em relação aos processadores convencionais.

6. CONCLUSÕES

Os microprocessadores apresentaram uma evolução muito grande na década passada, tirando o computador dos grandes centros de processamento e colocando-o disponível em equipamentos de mesa. Como continuação deste processo de evolução, as arquiteturas RISC se firmaram como solução de menor custo e maior desempenho.

Nesta década espera-se uma grande utilização do processamento paralelo, que se apresenta como solução para aumentos significativos de desempenho, sem o dispêndio equivalente em termos de complexidade de "hardware". Desta maneira, a filosofia utilizada nos rnicroprocessadores RISC se adapta perfeitamente aos conceitos de processamento paralelo. O processamento paralelo demanda um grande número de unidades funcionais trabalhando cooperativamente. Na medida que estas unidades possam ser mais baratas e também possam ocupar menor área de integração, mais facilmente e melhores serão os resultados obtidos.

Em termos nacionais, este tipo de arquitetura, aliado a novos tipos de tecnologia de integração, como GaAs, permitiria ao país a obtenção de autonomia em uma área bastante crítica. Tal fato cresce em importância na medida em que restrições cada vez maiores são aplicadas à exportação de produtos com tecnologia de ponta pelos países desenvolvidos. A indíistna nacional de microeletrônica precisa de um carro chefe, tal como as memórias foram para a indústria japonesa na década passada, capaz de lançar os produtos nacionais no mercado externo e sustentar a manutenção de um setor onde a competitividade é cada vez maior.

Neste sentido, acredito que o trabalho desenvolvido atualmente no NCE/UFRJ seja de extrema valia para a consecução destes objetivos. Esta tese estudou os rnicroprocessadores RISC, avaliando as diversas arquiteturas existentes, selecionando uma delas para a implementação de uma arquitetura compatível e determinando quais dispositivos devem ser integrados, para obtenção de uma solução final de custo ótimo.

A iinplementação do microprocessador do NCE/UFRJ [BAR90] devera 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 e o tipo do barramento, o uso de cache de desvio e o tamanho do conjunto de registradores. A não implementação de algumas instruções está sendo considerada. 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. Conforme a área disponível para integração poderemos tentar a integração de uma pequena cache de desvio de 16 posições. Não serão implernentadas as instruções do tipo TAGGED e as instruções MULScc. Esta última função deverá ser realizada pelo coprocessador de ponto flutuante, já que a aplicação principal para qual se destina o microprocessador necessita da utilização destas unidades. 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 com o usuário que foi utilizada 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 se 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 as 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.

7. BIBLIOGRAFIA

[AGR88] - Agrawal, A. et alii "Design Considerations for a Bipolas Implementation of SPARC" Proceedings of the IEEE COMPCON 88, New Yosk, NY, IEEE, pp 6-9,1988.

[AMD88] - AMD "AM29000 Streamlined Instruction Processos". Johnson M, ed., Advanced Micro Devices, Sunnyvale, 1988,440 pp.

[AUD90] - Aude, J. S. et alii "MULTIPLUS: Um Multipsocessadorde Alto Desempenho" Anais do X Congresso da SBC, Vitória, pp 93-105, Julho de 1990.

[BAR90] - Barbosa, M.A.S. et alii "Implementação de Micsoprocessador RISC com Arquitetura SPARC" Anais do V Simpósio Brasileiro de Concepção de Circuitos Integrados , Ouro Preto, pp 121-131, Outubso 1990.

[BER87] - Besenbaum, A.D. et alii "CRISP: A pipelined 32-bit RISC micsoprocessor with 13 K-bit of Cache Memory" E E E Journal of Solid State Circuits, New Yosk, NY, SC-22(5):776-782, Oct 1987.

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

[CYP90] - Cypress Semicondutor "SPARC RISC USER'S GUIDE", Cypress, 2nd Edition, Feb 1990,416 pp.

[FAI86] - Fairchild Semiconductoss "CLIPPER MODULE 32-bit Micropsocessos", Fairchild, 1986, 94 pp.

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

[GIM87] - Gimarc, C.E. & Milutinovic, U. "A Survey of RISC Processors and Computers of the MID 1980s." IEEE Computes, 20(9):59-69, Sep 1987.

[HEN82] - Henessy, J. et alii "MIPS: A Micsopsocessos Architecture" ACM SIGMICRO NEWSLETTER, New Yosk, NY, 13(4):17-2, Dec 1982.

[HIL84] - Hill, M.D. & Smith, A.J. "Experimental Evaluation of On-Chip Cache Microprocessors" Proceedings of the 11th International Symposium on Computer Architecture, pp 158-166, 1984.

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

[KAN88] - Kane, G. "MIPS Risc Architecture" Prentice-Hall, Englewood Cliffs, NJ, 1988,334 pp.

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

[KLEI88] - 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.

[MOT88] - Motorola "MC88100 Reduced Instruction Set Computer Users Manual." Motorola, Austin, Texas, 1988,235 pp.

[MUC88] - Muchnick, S.S. et alii " 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 alii "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 alii "CMOS Custom Irnplementation of the SPARC Architecture" Proceedings of the IEEE COMPCON 88, New York, NY, IEEE, pp 18-20, 1988.

[PAT80] - Patterson, D.A. & Ditzel, D.R. "The Case for a Reduced Instruction Set Computer" ACM Computer Architecture News, New York, NY, 8(6):25-32, Oct 1980.

[PAT81] - Patterson, D.A. & Sequin, C.H. "RISC I: A Reduced Instruction Set VLSI Computer" ACM SIGARCH Newsletter, New York, NY, 9(3):443-57, May 1981.

[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 I." Beskeley, CA, University of California, Report, UCB/CSD 821106, 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, pp 132-142, Outubro 1990.

[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 Computess, Vol C-32 (11): 977-989, Nov 1983.

[UNG84] - Ungas, D. et alii "Architecture of SOAR: Smalltalk on a RISC" Proceedings of the 11th International Symposium on Computes Architecture, pp 158-166, 1984.

APENDICE A

Conjunto de Instruções do SPARC

Instruções de Load/Store

Nome		Descrição
LDSB	(LDSBA)	Carrega Reg. c/ 8 bits c/ sinal (do espaço altern.)
LDSH	(LDSHA)	Carrega Reg. c/ 16 bits c/ sinal (do espaço altern.)
LDUB	(LDUBA)	Carrega Reg. c/ 8 bits sem sinal (do espaço altern.)
LDUH	(LDUHA)	Carrega Reg. c/ 16 bits sem sinal (do espaço altesn.)
LD	(LDA)	Carrega Reg. com 32 bits (do espaço alternativo)
LDD	(LDDA)	Carrega Reg. com 64 bits (do espaço alternativo)
STB	(STBA)	Armazena 8 bits na Memória (no espaço alternativo)
STH	(STHA)	Armazena 16 bits na Memória (no espaço alternativo)
ST	(STA)	Armazena 32 bits na Memória (no espaço alternativo)
STD	(STDA)	Armazena 64 bits na Memória (no espaço alternativo)
LDTSUB	(LSTUBA)	Carga-Armazenamento Atômico de 8 bits (esp.alt.)
SWAP	(SWAPA)	Troca Registrador de 32 bits com Memória (esp.alt.)

Instruções de Interface com Coprocessador

Nome	Descrição
LDF	Carrega 32 bits no Reg. Ponto Flutuante
LDDF	Carrega 64 bits no Reg. Ponto Flutuante
LDFSR	Carrega 32 bits no Reg. de Status Ponto Flutuante
STF	Armazena 32 bits do Reg. Ponto Flutuante na Memória
STDF	Armazena 64 bits do Reg. Ponto Flutuante na Memória
STFSR	Armazena 32 bits do Reg. de Status Ponto Flutuante
STDFQ	Retira 64 bits da Fila Instr. do Ponto Flutuante
LDC	Carrega 64 bits no Reg. Coprocessador
LDDC	Carrega 32 bits no Reg. Coprocessador

LDCSR	Carrega 64 bits no Reg. de Status Coprocessador
STC	Armazena 32 bits do Reg. Coprocessador na Memória
STDC	Armazena 64 bits do Reg. Coprocessador na Memória
STCSR	Armazena 32 bits do Reg. de Status Coprocessados
STDCQ	Retira 64 bits da Fila Instr. do Coprocessados

Instruções Aritméticas e Lógicas

Nome		**Descrição**	
ADD	(ADDcc)	Soma	(modifica Código de Condição)
ADDX	(ADDXcc)	Soma com Cany	(modifica Código de Condição)
TADDcc	(TADDccTV)	Soma c/ Tag e modifica cc	(gera Trap no Overflow)
SUB	(SUBcc)	Subtração	(modifica Código de Condição)
ADDX	(ADDXcc)	Subtração com Carry	(modifica Código de Condição)
TSUBcc	(TSUBccTV)	Sub. c/ Tag e modifica cc	(gera Trap no Overflow)
MULScc		Passo de Multiplicação e modifica Código Condição	
AND	(ANDcc)	E Lógico	(modifica Código de Condição)
ANDN	(ANDNcc)	E Lógico Negado	(modifica Código de Condição)
OR	(ORcc)	OU Lógico	(modifica Código de Condição)
ORN	(ORNcc)	OU Lógico Negado	(modifica Código de Condição)
XOR	(XORcc)	OU Exclusivo	(modifica Código de Condição)
XORN	(XORNcc)	OU Exclusivo Negado	(modifica Código de Condição)
SLL		Deslocamento Lógico à Esquerda	
SRL		Deslocamento Lógico à Direita	
SRA		Deslocamento Aritmético à Direita	
SETHI		Carrega Registrador com Constante Imediata	

Instruções de Transferência de Controle

Nome	Descrição
SAVE	Avança de uma Janela de Registradores
RESTORE	Recua de urna Janela de Registradores
Bicc	Desvia testando Código de Condição Inteiro
FBfcc	Desvia testando Código de Condição Flutuante
CBccc	Desvia testando Código de Condição Coprocessados
CALL	Chamada de Subrotina
JMPL	Desvia e Salva PC
RETT	Retomo de Exceção
Ticc	Desvia para Rotina de Exceção testando cc Inteiro

Instruções Diversas

Nome	Descrição
RDY	Leitura do Registrador Y
RDPSR	Leitura do Registsador PSR
RDWIM	Leitusa do Registrador WIM
RDTBR	Leitusa do Registrador TBR
WRY	Escrita no Registrador Y
WRPSR	Escrita no Registrador PSR
WRWIM	Escrita no Registrador WIM
WRTBR	Escrita no Registrador TBR
UNIMP	Instrução Não Implementada
IFLUSH	Flush da Cache Interna de Instruções