FABIANY LAMBOIA

ANÁLISE COMPARATIVA DE USO DOS CONJUNTOS DE INSTRUÇÕES DOS MICROPROCESSADORES DE 32 BITS MIPS, POWERPC E SPARC

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Roberto A Hexsel.

CURITIBA

2008

FABIANY LAMBOIA

ANÁLISE COMPARATIVA DE USO DOS CONJUNTOS DE INSTRUÇÕES DOS MICROPROCESSADORES DE 32 BITS MIPS, POWERPC E SPARC

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Roberto A Hexsel.

CURITIBA

2008

FABIANY LAMBOIA

ANÁLISE COMPARATIVA DE USO DOS CONJUNTOS DE INSTRUÇÕES DOS MICROPROCESSADORES DE 32 BITS MIPS, POWERPC E SPARC

Dissertação aprovada como requisito parcial à obtenção do grau de Mestre no Programa de Pós-Graduação em Informática da Universidade Federal do Paraná, pela Comisso formada pelos professores:

Orientador: Prof. Dr. Roberto A Hexsel.
Departamento de Informática, UFPR

Prof. Dr. Rodolfo Jardim de Azevedo
Instituto de Computação, UNICAMP

Prof. Dr. Eduardo Todt
Departamento de Informática, UFPR

Curitiba, 22 de agosto de 2008

AGRADECIMENTOS

Agradeço a Deus por sempre me iluminar e me dar forças em todas os momentos difíceis e nos momentos de fraqueza, foi com ajuda Dele que consegui vencer mais uma batalha e realizar um sonho.

Agradeço e dedico essa dissertação a minha mãe Nelsi e ao meu pai Francisco que sempre me apoiaram em todas as minhas decisões e pelo incentivo que me deram para fazer o mestrado mesmo sabendo que a distância traria muitas saudades.

Ao meu orientador Prof. Roberto de todo coração, por sempre me ajudar quando precisei, pela paciência de sempre me responder o que fosse necessário, pelos ensinamentos, os conselhos e as conversas. Essa conquista é em grande parte também dedicada a você, Prof. Roberto.

Ao meu companheiro e namorado Daniel, por sempre estar ao meu lado, me apoiar, me consolar nos momentos difíceis e para qualquer problema sempre esteve disposto a me ajudar. Obrigado do fundo do coração por tudo que você fez e faz por mim.

Ao meu amigo Joylan, pelas horas de estudos para as disciplinas, pelas conversas, pela paciência comigo e por toda ajuda que sempre me deu a qualquer momento que eu precisasse.

Ao meu amigo Ricardo, pelas consultorias online, sempre me ajudando e dando dicas importantes para resolver vários problemas.

As amigas moradoras, Alyne, Cristine e Janaína por estarem sempre por perto e me aguentarem desde o começo tanto nos momentos alegres como nos momentos mais estressantes.

A todos que me apoiaram e incentivaram para que eu realizasse este trabalho, e que sempre estiveram por perto e contribuíram de alguma forma.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

RESUMO

Os sistemas embarcados estão em constante evolução e há uma grande pressão de mercado para disponibilizar novos produtos com uma quantidade maior de funcionalidades em um curto espaço de tempo. Desse modo, é necessário que os projetistas utilizem ferramentas que auxiliem no desenvolvimento de uma nova arquitetura, em otimizações de interfaces entre os componentes ou na escolha de um processador mais adequado para uma determinada aplicação embarcada. Dentre as ferramentas disponíveis, salientam-se os simuladores que executam as aplicações e produzem medidas relevantes de desempenho.

Este trabalho apresenta uma comparação, no nível do conjunto de instruções, dos microprocessadores de 32 bits MIPS, PowerPC e SPARC. Sete programas da suíte CommBench [40], sete da suíte MediaBench [6] e treze da suíte MiBench [21], foram compilados com a versão 3.3.1 do GCC e simulados com modelos funcionais escritos em ArchC. Considerando os totais para os 27 programas, os números de instruções executadas são $89,23$, $103,92$, e $99,75 \cdot 10^9$ para MIPS, PowerPC e SPARC, respectivamente. Os resultados mostram que o PowerPC executa o maior número de instruções, o SPARC obteve um melhor desempenho em relação aos acessos à memória e o MIPS obteve um melhor resultado no total de instruções executadas.

Também foram avaliados os efeitos dos três níveis de otimização no código gerado e a distribuição no tempo de referências à memória. Cerca de 80% dos LOADs e 60% dos STOREs encontram-se separados por menos de 5 instruções. Para MIPS e PowerPC, cerca de 80% dos desvios condicionais estão afastados de até 6 instruções, enquanto que para SPARC 90% dos desvios são separados de até 3 instruções.

Os resultados destas simulações são úteis para a otimização do gerador de código no compilador, para o projeto da interface entre processador e memória, da hierarquia de caches, bem como para subsidiar a escolha do processador para aplicações embarcadas.

CAPÍTULO 1

INTRODUÇÃO

Sistemas embarcados são sistemas que incluem microprocessadores capazes de realizar uma ou mais tarefas específicas, com recursos computacionais como memória e capacidade de processamento projetados sob restrições e para um propósito especial. Exemplos de tais sistemas incluem os telefones celulares, o sistema de controle de freios ABS, os computadores portáteis palm-top e eletrodomésticos sofisticados.

Estes produtos são concebidos sob restrições de projeto tais como portabilidade e limite de consumo de potência sem perda de desempenho, a utilização de memória, tamanho do código da aplicação, a necessidade de segurança e confiabilidade, o curto tempo de projeto e, principalmente, custo. O custo envolve tanto os componentes de hardware como o desenvolvimento do software aplicativo.

Baseado nessas restrições, projetistas de sistemas embarcados enfrentam dificuldades para a finalização de projetos em tempo hábil. A complexidade de tais sistemas e a grande variedade de configurações possíveis podem tornar a escolha do sistema ideal demorada, prolongando o tempo de projeto e, consequentemente, seu ingresso no mercado.

O processo de desenvolvimento de um novo projeto envolve várias escolhas tais como, qual será a arquitetura empregada, qual o conjunto de instruções e a hierarquia de memória, quais serão as interfaces de comunicação entre os componentes do hardware, qual o sistema operacional a ser empregado e quais serão os aplicativos.

As aplicações embarcadas valorizam o custo e a energia, assim o tamanho do código é importante, porque menos memória significa um menor consumo de energia. É, portanto, essencial a análise do conjunto de instruções da arquitetura em questão, pois algumas classes de instruções, como por exemplo as de ponto flutuante, podem ser eliminadas para reduzir os custos de circuitos integrados. O conjunto de instruções tem influência direta no tempo de execução da aplicação.

Essas escolhas precisam ser analisadas com cuidado, buscando encontrar as características relevantes de cada componente que atenderão melhor os requisitos do projeto. Desse modo, os projetistas necessitam de ferramentas para auxiliá-los na escolha e na análise dos componentes que estão sendo considerados para o projeto. Ferramentas importantes que auxiliam os projetistas são simuladores que executem as aplicações em arquiteturas diferentes, e que permitam avaliar o desempenho e comparar o comportamento da arquitetura para diversas aplicações.

Este trabalho apresenta uma comparação no nível de conjunto de instruções, dos microprocessadores de 32 bits MIPS, PowerPC e SPARC, que são populares em aplicações embarcadas, tipicamente como parte de um sistema. Sete programas da suíte CommBench [40, 8], sete da suíte MediaBench [6, 23] e treze da suíte MiBench [21, 24], foram compilados com a mesma versão do GCC (3.3.1) e simulados em modelos funcionais escritos em ArchC [34].

As comparações entre os processadores são efetuadas em condições de igualdade, com o mesmo compilador e o mesmo conjunto de programas. Os resultados destas simulações são comparados quanto ao número de instruções executadas, número e distribuição dos acessos à memória, e número e distribuição de instruções de controle de fluxo.

Estes resultados são úteis para (i) a otimização do gerador de código no compilador, (ii) para o projeto da interface entre processador e memória, (iii) hierarquia de caches, (iv) melhorar a implementação do previsor de desvios, e (v) para subsidiar a escolha do processador para aplicações embarcadas.

Como o tamanho do código é um fator importante nos sistemas embarcados, otimizar o gerador de código é bastante útil, pois a otimização pelo compilador pode reduzir o tamanho do código significamente. Uma das estratégias de otimização é a *strenght reduction*, que é a troca de operações complexas por simples, como na substituição da multiplicação por uma constante, por adições e deslocamentos. Com o conhecimento detalhado do conjunto de instruções de uma arquitetura, podem ser melhoradas as otimizações executadas pelo compilador dependentes do processador.

A utilização de memória também é um fator crítico para aplicações embarcadas e,

portanto, é essencial que o projeto de memória tenha um bom desempenho, reduza o tempo de acesso à memória e economize energia. A memória cache além de reduzir o tempo de acesso à memória, também pode poupar energia. Conhecer a distribuição no tempo de referências à memória pode ser muito útil no projeto de hierarquias de memória para sistemas embarcados.

O número de desvios condicionais em uma aplicação é em grande parte fixado pela aplicação, com exceção de algumas anomalias da arquitetura e possíveis otimizações pelo compilador. A frequência em que os desvios são executados é importante porque desvios estão entre as instruções mais difíceis de implementar para execução rápida e também são difíceis de otimizar pelo compilador. Conhecer o comportamento dos desvios auxilia na implementação de um melhor previsor de desvios.

Os objetivos deste trabalho são três: (1) disponibilizar uma comparação que permita ao projetista de sistemas embarcados analisar características relevantes de cada arquitetura, assim como o seu comportamento em relação a diversas aplicações utilizadas em sistemas embarcados; (2) prover auxílio na depuração de desempenho, na otimização do gerador de código, e no projeto de hierarquia de memória; e (3) proporcionar subsídios para a escolha de um processador para um propósito específico, relacionados aos recursos de execução (uso dos registradores, classe de instruções, referências à memória, controle de fluxo, e lógica suportada) para se alcançar um bom desempenho.

Esta trabalho está organizado da seguinte maneira: o Capítulo 2 contém definições e métodos para o desenvolvimento de sistemas embarcados e também apresenta uma revisão dos trabalhos relacionados na literatura. A seção 2.3 contém uma revisão geral dos conceitos de arquitetura de computadores, e na seção 2.4 são descritos as principais características dos microprocessadores utilizados para as simulações. O Capítulo 3 apresenta o ambiente de simulação, descreve o simulador utilizado, os conjuntos de programas de teste e a metodologia de simulação e caracterização. O Capítulo 4 apresenta os resultados obtidos e compara os três processadores do ponto de vista das contagens de instruções dinâmicas. O Capítulo 5 traz as conclusões e as sugestões de trabalhos futuros.

CAPÍTULO 2

REVISÃO BIBLIOGRÁFICA

2.1 Sistemas Embarcados

A tarefa desempenhada pelo projetista de computadores embarcados é complexa e inclui determinar quais atributos são importantes para uma nova máquina, projetar uma máquina para maximizar o desempenho e, ao mesmo tempo, permanecer dentro das restrições de custo e energia [15].

O projeto de sistemas embarcados é bastante complexo e essa complexidade é crescente como conseqüência dos novos requisitos e porque o espaço de projeto arquitetural a ser explorado é muito amplo. Tipicamente, um sistema embarcado pode conter um ou mais processadores, memória, interfaces para periféricos e blocos dedicados [2]. Os componentes são interligados por uma estrutura de comunicação que pode variar de um barramento a uma rede complexa.

Os processadores podem ser de arquiteturas diversas: RISC, VLIW, DSP, separados ou como *Processadores Integrados para Aplicações Específicas* (*ASIPs*). No caso de sistemas contendo componentes programáveis, o software de aplicação pode ser composto por múltiplos processos, distribuídos entre diversos processadores e comunicando-se através de diferentes mecanismos [18]. Em muitos casos os processos são gerenciados através de um sistema operacional de tempo real (*RTOS*), que fornece, pelo menos, serviços de comunicação e escalonamento de processos.

Em muitas aplicações é adequada a integração do sistema em uma única pastilha (*System-on-a-Chip* - SoC). Neste caso, quando as restrições de recursos tais como potência, memória e tempo, tornam-se críticas, pode ser indispensável a construção de um *Circuito Integrado para Aplicação Específica* (*ASIC*) ou um *Field Programmable Gate Array* (*FPGA*) [3].

Os *FPGAs* mais sofisticados apresentam uma grande densidade interna de blocos

lógicos configuráveis que podem ser interconectados através de reconfiguração dinâmica de unidades funcionais. Os *FPGAs* têm a característica de agregar a flexibilidade dos processadores de uso geral com a "especialização" de hardware que é oferecida nos *ASICs*. Adicionalmente, tem menor custo que os *ASICs*, especialmente a cada inovação dos componentes.

Tanto *ASICs* quanto *ASIPs* se baseiam em um conjunto de blocos de hardware construído especialmente para uma determinada aplicação. Os *ASIPs* são processadores que possuem a arquitetura e o conjunto de instruções personalizados para um dado sistema. Um *ASIP* deve ser implementado sobre uma plataforma de hardware também específica, que pode ser um *ASIC* ou um *FPGA*.

Para atender as necessidades do mercado, um projeto de sistema embarcado geralmente deve ser desenvolvido em um curto espaço de tempo. Assim, há uma tendência no projeto de sistemas embarcados em realizar a modelagem em software do sistema completo a ser desenvolvido. Esta modelagem visa aumentar a qualidade dos componentes, identificar erros em fases iniciais e aumentar a produtividade do projetista.

2.2 Trabalhos relacionados

A simulação é uma ferramenta de grande importância para o desenvolvimento de sistemas embarcados. Os simuladores são necessários para fornecer uma plataforma conveniente para testes, eliminação de erros e otimização de aplicações.

Através de simulações, diversas análises podem ser feitas enfocando as características necessárias ao projeto. Os resultados obtidos com as simulações podem auxiliar o projetista na escolha do sistema adequado ao seus objetivos, permitindo fazer comparações entre os sistemas simulados.

Em [3] é apresentado um ambiente de simulação que consiste de simuladores que permitem estimar métricas de desempenho como tempo de execução e memória utilizada em código, pilha e conjunto de dados. O ambiente suporta os processadores Motorola DSP56F827 [27], Rabbit R2000 [32] e Atmel Atmega8515 [5] e pode ser facilmente expandido com outros processadores. Os simuladores foram desenvolvidos utilizando a Lingua-

gem de Descrição de Arquiteturas *ArchC* [34].

Conforme descrito pelo autor, a construção dos simuladores foi trabalhosa, principalmente em relação ao microprocessador DSP56F827 por causa da sua arquitetura complexa. A versão, desenvolvida no trabalho, do simulador do Motorola DSP56F827 possui precisão melhor que 85% nas medidas de tempo de execução, e 89% a 97% nas medidas de utilização de memória quando comparado com uma implementação em hardware. O autor recomenda que no momento da escolha do microprocessador para um projeto não basta apenas o levantamento das características básicas do processador, como quantidade de memória e frequência do relógio, sendo necessária uma avaliação mais cuidadosa da arquitetura, como verificação de segmentação e dos detalhes do conjunto de instruções.

Uma comparação com objetivo similar ao deste trabalho é apresentada em [31]. A análise é feita a partir das contagens dinâmicas das instruções do conjunto de instruções (CdI) do MIPS e SPARC, com os programas do SPEC-1991, usando os compiladores nativos dos dois processadores e simuladores distintos. Os resultados obtidos mostram que o MIPS executa 18% a mais de instruções que o SPARC, e o SPARC executa 4% a mais de instruções em rotinas de biblioteca que o MIPS. A diferença mais significativa das arquiteturas constatada no trabalho, é que as instruções de ponto flutuante de dupla-precisão LOAD/STORE do SPARC oferecem vantagens nas aplicações de ponto flutuante do SPEC, enquanto as instruções *compara-e-desvia* (*compare-and-branch*) do MIPS oferecem vantagem nas aplicações de inteiros do SPEC.

Em um dos apêndices de [15], há um comparação funcional entre os conjuntos de instruções do Alpha, MIPS, PA-RISC, PowerPC, e SPARC. São discutidos os modos de endereçamento e os formatos de instruções dessas arquiteturas RISC, e também as instruções e características exclusivas de cada uma das cinco arquiteturas.

Uma das fontes de inspiração para este trabalho foi [36]. O texto traz um comparação funcional entre PowerPC 601 e o Alpha 21064, e analisa aspectos das duas arquiteturas e detalhes da implementações dos pipelines. Estas arquiteturas, o PowerPC e o Alpha 21064, seguem duas filosofias bastantes diferentes para alcançar implementações de alto desempenho. A arquitetura PowerPC define um conjunto de instruções mais complexo,

que consegue realizar mais trabalho por instrução. A simplicidade da arquitetura Alpha, por outro lado, proporciona implementações com taxas mais altas de relógio. Assim, o PowerPC 601 ganha desempenho pela engenhosidade de projeto, e o Alpha 21064 ganha desempenho pela simplicidade de projeto. Este *trade-off* é um clássico, e o fato de ambas filosofias levarem a processadores viáveis é provavelmente uma indicação de que qualquer escolha é satisfatória desde que a implementação seja bem feita.

Uma comparação entre três arquiteturas RISC, Intel i860, Motorola 88000 e SPARC é mostrada em [29]. São comparados os pontos fortes e fracos das três arquiteturas em relação a áreas chaves, tais como conjunto de instruções, desvios, modos de endereçamento, registradores, tipos de dados e gerência de memória. Com base nesta comparação são avaliadas as vantagens e desvantagens de cada arquitetura, a fim de determinar se uma arquitetura é claramente superior ou inferior, devido às vantagens ou desvantagens que dispõe sobre as outras. A análise dos diversos componentes das arquiteturas revelou que cada uma tem algumas áreas que oferece melhor suporte do que as outras, e áreas em que oferecem suporte claramente pior. Considerando todos os fatores analisados no trabalho, concluiu-se que nenhuma das três arquiteturas é claramente inferior ou superior às outras, pois uma implementação particularmente ruim ou particularmente boa de qualquer umas das três arquiteturas pode contrabalençar qualquer diferença que tenha sido identificada.

2.3 Arquitetura de Conjunto de Instruções

A arquitetura de conjunto de instruções é a parte do computador visível para o programador ou para o projetista de compiladores. Existe uma grande variedade de alternativas de projeto disponíveis para o arquiteto do conjunto de instruções.

Podemos classificar a arquitetura de conjunto de instrução em relação ao armazenamento interno do processador e as principais formas de armazenamento são uma pilha, um acumulador ou um conjunto de registradores. Os operandos podem ser nomeados de forma implícita ou explícita. Em uma arquitetura de pilha os operandos estão (implicitamente) no topo da pilha e em uma arquitetura de acumulador, um operando é (implicita-

mente) o acumulador. As arquiteturas de registradores de uso geral tem apenas operandos explícitos, sejam eles registradores ou posições de memória. Os operandos explícitos podem ser acessados de forma direta a partir da memória ou devem ser carregados primeiramente em um armazenamento temporário, dependendo da classe de arquitetura e da escolha de instruções específicas. A Figura 2.1 mostra a relação entre a unidade funcional de lógica e aritmética e o armazenamento em memória e em registradores [15].

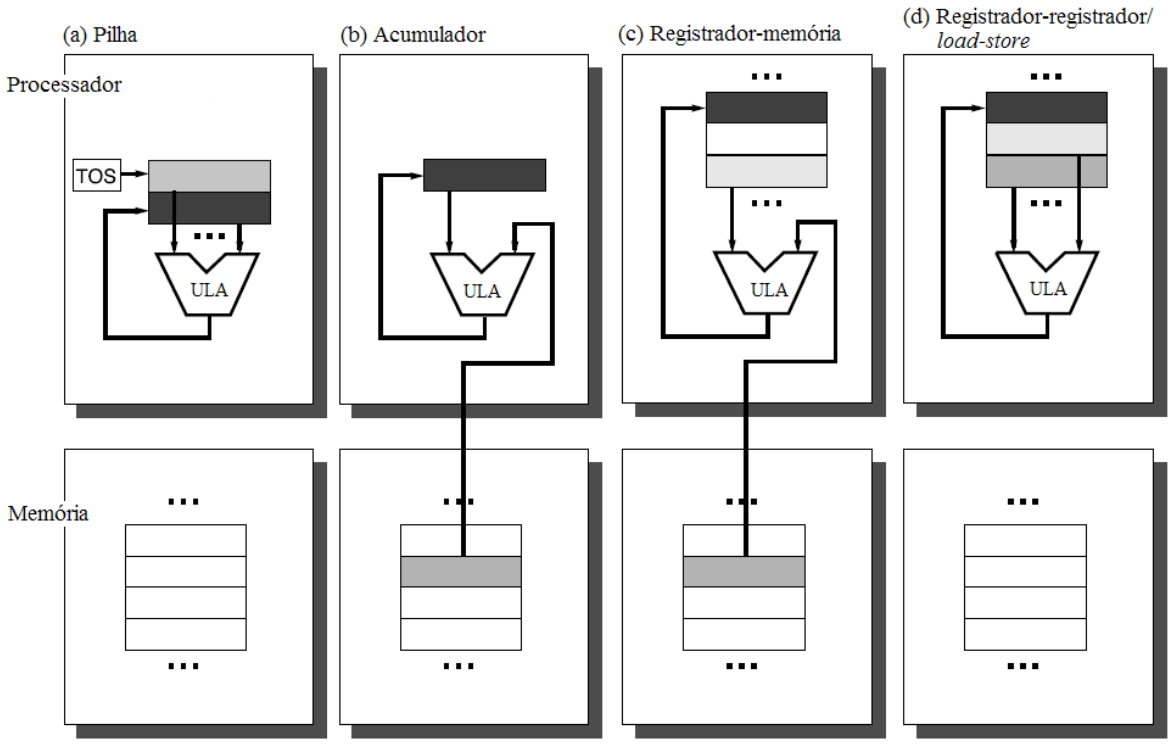

Figura 2.1: Posições de operandos para quatro classes de arquiteturas de conjunto de instruções.

Como pode ser visto na Figura 2.1 existem duas classes de arquiteturas de registrador. A arquitetura registrador-memória pode acessar diretamente a memória com uma instrução e a arquitetura registrador-registrador ou LOAD/STORE pode acessar a memória somente com instruções de LOAD e STORE.

Os primeiros computadores construídos usavam arquiteturas de pilha ou de acumulador porque estas necessitam de circuitos simples. A partir de 1980 as arquiteturas novas passam a utilizar a arquitetura de registrador LOAD-STORE. As principais razões para o surgimento de computadores de registradores de uso geral ou *General-Purpose Regis-*

ter (*GPR*) são que os registradores são mais rápidos que a memória e são mais eficientes para uso por um compilador do que outras formas de armazenamento interno [13].

Os conjuntos de instruções possuem duas características importantes que dividem as arquiteturas GPR, e estas características estão relacionadas à natureza dos operandos de uma instrução aritmética ou lógica (instruções que usam a Unidade de Lógica e Aritmética - ULA). A primeira característica corresponde à quantidade de operandos de uma instrução, que pode ser dois ou três operandos. No formato de três operandos, a instrução contém um operando de resultado e dois operandos propriamente ditos. No formato de dois operandos, um dos operandos é ao mesmo tempo uma origem e um resultado para a operação. A segunda característica corresponde a quantos operandos podem ser endereços de memória em instruções de ULA. O número de operandos de memória suportados por uma instrução de ULA típica pode variar de nenhum até três. Na Tabela 2.1 são mostradas as possíveis combinações e exemplos de computadores que empregam as várias combinações [15].

Número de endereços de memória	**Número de operandos permitidos**	**Arquitetura**	**Exemplos**
0	3	Registrador-registrador	Alpha, ARM, MIPS, PowerPC, SPARC, Trimedia
1	2	Registrador-memória	IBM 360/370, Intel 80x86, Motorola 68000
2	2	Memória-memória	VAX
3	3	Memória-memória	VAX

Tabela 2.1: Combinações típicas de operandos de memória e operandos totais por instrução de ULA.

2.3.1 Modos de Endereçamento

O modo de endereçamento é a maneira como as arquiteturas especificam o endereço de um objeto a ser acessado. Os modos de endereçamento especificam constantes e registradores, além de posições de memória. Quando uma posição de memória é usada, o endereço de memória real especificado pelo modo de endereçamento é chamado “endereço efetivo”.

Na Tabela 2.2 são mostrados todos os modos de endereçamento de dados utilizados [13]. Os modos de endereçamento permitem uma efetiva redução da quantidade de instruções, mas aumentam a complexidade da implementação de um computador e podem aumentar o valor médio de *Ciclos por Instrução* (CPI).

Modo	**Exemplo de instrução**	**Endereço Efetivo (EE)**	**Quando é usado**
Registrador	ADD r_4,r_3	—	Valor está em um registrador.
Imediato	ADD r_4,#3	—	Para constantes.
Deslocamento	ADD r_4,100(r_1)	EE = m[[r_1] + 100]	No acesso a variáveis locais (simula modos de endereçamento indireto de registrador e direto).
Indireto de registrador	ADD r_4,(r_1)	EE = m[[r_1]]	No acesso com o uso de um ponteiro ou um endereço calculado.
Indexado	ADD r_3,(r_1+r_2)	EE = m[[r_1] + [r_2]]	Endereçamento de vetores: r_1 = base do vetor; r_2 = valor do índice.
Direto ou absoluto	ADD r_1,(1001)	EE = m[1001]	Acesso a dados estáticos; a constante de endereço talvez deva ser grande.
Indireto de memória	ADD r_1,@(r_3)	EE = m[m[[r_3]]]	Se r_3 é o endereço de um ponteiro p, então o modo produz $*p$.
Auto-incremento	ADD r_1,(r_2)+	EE = m[[r_2]];r_2++	Percorrer vetores em um laço. r_2 aponta para o inicio do vetor; cada referência incrementa r_2 pelo tamanho de um elemento.
Auto-decremento	ADD r_1,-(r_2)	r_2--;EE = m[[r_2]]	Mesmo uso do auto-incremento. Auto-decremento/auto-incremento podem ser usados como operações push/pop para implementar uma pilha.
Escalonado	ADD r_1,n(r_2)[r_3]	EE = m[(r_2) + [r_3]*n]	Usado para indexar vetores. Pode ser aplicado a qualquer modo de endereçamento indexado em alguns computadores.

Tabela 2.2: Principais Modos de endereçamento.

2.3.2 Operandos: tipos e tamanhos

Em geral, o tipo de um operando efetivamente é seu tamanho. Os tipos de operando comuns são caracteres (*character* - 8 bits), meia palavra (*halfword* - 16 bits), palavra (*word* - 32 bits), palavra-dupla (*double-word* - 64 bits), ponto flutuante de precisão simples (*single-precision floating point* - 32 bits) e ponto flutuante de precisão dupla (*double-precision floating point* - 64 bits).

Os números inteiros são representados como números binários em complemento de dois. Os caracteres normalmente são representados em ASCII. Para representação de ponto flutuante, a maioria dos computadores segue o padrão 754 do IEEE.

2.3.3 Tipo de Operações

A maioria das arquiteturas suportam um determinado conjunto de operações. Este conjunto pode ser divido em classes:

- **Aritmética e lógica** - operações de aritmética de inteiros e operações lógicas: adição, subtração, conjunção, disjunção, negação, multiplicação e divisão;
- **Transferência de dados** - operações de *load-store*, instruções de *move* com endereçamento de memória;
- **Controle** - desvios condicionais e incondicionais, chamada e retorno de procedimento;
- **Sistema** - chamadas do sistema operacional, instruções de gerenciamento de memória virtual, (*traps*, *syscall*);
- **Ponto flutuante** - operações de ponto flutuante: adição, multiplicação, divisão, comparação;
- **Decimal** - adição decimal, multiplicação decimal, conversões de decimais para caracteres;
- **Strings** - movimentação de *strings*, comparação de *strings* e pesquisa de *strings*;

- **Gráficos** - operações de pixels e vértices, operações de compactação e/ou descompactação.

A maioria das arquiteturas fornece um conjunto completo de operações para as três primeiras classes. Para as outras classes o suporte no conjunto de instruções pode variar de nenhum até um conjunto extenso de instruções especiais [15].

Instruções de Controle de Fluxo

As instruções de controle de fluxo possuem modos de endereçamento e comportamento diferentes da outras instruções e podem ser divididos em quatro tipos principais:

- Desvios condicionais (*branches*);
- Desvios incondicionais (*jumps*);
- Chamadas de procedimentos;
- Retorno de procedimentos.

Existem ainda outras situações em que o fluxo de controle é modificado, como no tratamento de interrupções e de exceções (*traps*).

Modos de endereçamento para controle de fluxo. O endereço de destino de uma instrução de controle de fluxo deve ser especificado, e este é geralmente explícito, com exceção do retorno de procedimentos, pois o destino não é conhecido em tempo de compilação.

O modo mais comum de especificar o endereço de destino é fornecer um deslocamento que será adicionado ao contador do programa (*Program Counter - PC*). O uso de desvios relativos ao PC oferece diversas vantagens. Primeiramente porque o destino frequentemente está perto da instrução atual, e a especificação da posição em relação ao PC exige um número menor de bits. O endereçamento relativo ao PC também permite a execução do código independentemente da posição em que é carregado.

Para o caso de retorno de procedimentos e saltos indiretos, nos quais o destino não é conhecido em tempo de compilação, é necessário usar outro método de endereçamento.

Este método deve especificar o destino dinâmicamente, de forma que ele possa mudar em tempo de execução. Esse endereço dinâmico pode ser obtido nomeando um registrador que contém o endereço de destino, ou a instrução de salto pode permitir que qualquer modo de endereçamento seja usado para fornecer o endereço de destino [13].

Especificações de Desvio Condicional. Geralmente a maioria das instruções de controle de fluxo são desvios, assim se torna importante a decisão de como especificar a condição de desvio. Uma das propriedades de desvios é que um grande número das comparações são testes simples, e uma grande parcela são comparações com zero. Assim, algumas arquiteturas tratam estas comparações como casos especiais. Na Tabela 2.3 são mostradas as três principais técnicas usada nas arquiteturas atuais [15].

Nome	Exemplos	Como a condição é testada	Vantagens	Desvantagens
Código Condicional (CC)	80x86, ARM, PowerPC, SPARC, SuperH	Testa bits especiais definidos por operações de ULA, possivelmente sob controle do programa.	As vezes, a condição é definida livremente.	CC é um registrador de estado extra. Os códigos condicionais limitam a ordenação de instruções, pois eles transferem informação de uma instrução para um desvio.
Registrador de condição	Alpha, MIPS	Testa um registrador com o resultado de uma comparação.	Simples.	Consome um registrador.
Comparação e desvio	PA-RISC, VAX	A comparação faz parte do desvio. Com frequência, a comparação se limita a um subconjunto de operandos.	Uma instrução em vez de duas para um desvio.	Pode significar muito trabalho por instrução no caso de execução em pipeline.

Tabela 2.3: Métodos mais comuns para avaliar condições de desvio.

2.3.4 Codificação de um Conjunto de Instruções

A forma de representação binária para execução do processador afeta o tamanho do programa compilado e também a implementação do processador que tem que decodificar essa representação para encontrar rapidamente a operação e seus operandos. Geralmente uma operação é especificada em campo chamado *opcode* (*operation code*).

Na codificação de instruções, o número de registradores e o número de modos de endereçamento tem um impacto significativo sobre o tamanho das instruções, pois o campo de registrador e campo de modo de endereçamento podem aparecer muitas vezes em uma única instrução.

São três os tipos mais comuns para codificação das instruções:

- Variável - permite que quase todos os modos de endereçamento sejam combinados com todas as operações. Este tipo de codificação pode dar suporte a qualquer número de operandos e permite uma menor representação de código, pois campos que não são utilizados não precisam ser incluídos.

- Fixo - possui apenas um tamanho para todas as instruções e pode ser usado para própositos diferentes por instruções diferentes.

- Híbrida - instruções com codificação variável e fixa, tenta reduzir o tamanho do programa mantendo a facilidade de decodificação.

2.4 Descrição dos Microprocessadores

Dois dos três microprocessadores comparados aqui, MIPS e SPARC, foram concebidos no início da década de 80 e foram projetados segundo a filosofia *RISC* ou *Reduced Instruction Set Computers*. Naquela época, desejava-se projetar processadores que pudessem ser implementados num único circuito integrado e, portanto, seus conjuntos de instruções deveriam ser simples (*reduced*). A arquitetura do terceiro processador, o PowerPC, é derivada de um processador que também foi concebido na década de 80. Não é coincidência que os três conjuntos de instruções sejam similares porque todos foram

projetados para atender aos mesmos requisitos, que são satisfeitos por: (i) operandos de instruções aritméticas e lógicas são obtidos de registradores; (ii) modos de endereçamento simples, instruções LOAD carregam registradores da memória e instruções STORE salvam seu conteúdo na memória; (iii) todas as instruções tem o mesmo tamanho; e (iv) muitos (32) registradores de uso geral, os quais possuem o mesmo número de bits.

As descrições dos processadores são apresentadas do ponto de vista de seus conjuntos de instruções, nas versões de 32 bits. Sob este ponto de vista, pode-se dizer que o conjunto de instruções é a "camada de aplicação" (API) do *hardware* e portanto esconde do programador os detalhes da implementação. Para simplificar o texto, o conjunto de instruções (CdI) do MIPS é apresentado primeiro e os outros dois CdIs são definidos com a mesma nomenclatura e sintaxe.

2.4.1 MIPS

No início da década de 80 na Universidade de Standford uma equipe conduzida por John L. Hennessy projetou a arquitetura MIPS (*Microprocessor without Interlocked Pipe Stages*) [14, 33]. Em 1984, o microprocessador MIPS começou a ser fabricado pela *Silicon Graphics, Inc.*. Em meados da década de 90 estimava-se que um em cada três microprocessadores RISC comercializados eram MIPS. Atualmente o MIPS é desenvolvido pela *MIPS Technologies* [25].

A primeira implementação do *Microprocessor without Interlocked Pipeline Stages*, e que determinou o modelo de execução e o conjunto de instruções, foi um processador segmentado em cinco estágios. Neste modelo, cada instrução somente efetua poucas operações simples em cada segmento. As primeiras implementações não comportavam a lógica de intertravamento entre os segmentos do processador, forçando os projetistas a expor a implementação em cinco segmentos ao programador e/ou compilador. Este devia garantir que existe uma instrução útil (ou NOP) após uma instrução de desvio para preencher o *branch delay slot*. O programador também devia garantir que o resultado de um LOAD não fosse usado pela instrução seguinte, devido ao *load delay slot*. Versões correntes da arquitetura (MIPS32) mantém o *branch delay slot*; no caso dos LOADs, implementações

modernas paralisam a execução da instrução dependente até que o valor do LOAD seja entregue pela memória.

A arquitetura MIPS é baseada em um conjunto de instruções de tamanho fixo e segue o modelo LOAD/STORE. As operações de aritmética e de lógica usam um formato de três operandos, permitindo que os compiladores otimizem a formulação de expressões complexas.

A disponibilidade de 32 registradores de uso geral permite que o compilador favoreça a geração de código otimizado para um melhor desempenho, mantendo os dados frequentemente acessados em registradores. Entre os diversos produtos que utilizam a arquitetura MIPS podemos citar dispositivos de segurança (POD), dispositivos digitais (camêras, DVDs, Video Games), automação de escritório (impressoras, copiadoras, *scanners*) e controladores industriais.

Desde que surgiu, foram definidas muitas versões da arquitetura: MIPS I, MIPS II, MIPS III, MIPS IV, MIPS32 e MIPS64. Todas as versões usam o mesmo conjunto básico de instruções; o conjunto de instruções de 32 bits é definido em [26]. As características mais relevantes do CdI, especialmente as visíveis ao programador, são listadas abaixo.

Registradores O MIPS possui 32 registradores de uso geral (*General-Purpose Register - GPR*): $0, $1, $2, ..., $31. A Tabela 2.4 mostra os nomes e a descrição do uso convencional de cada registrador.

Registradores	**Nome**	**Descrição**
0	zero	Sempre contém o valor zero.
1	at	*Assembler Temporary*: usado pelo montador.
2-3	v0-v1	Valor do retorno de uma chamada de função.
4-7	a0-a3	Quatro primeiros parâmetros para uma chamada de função.
8-15	t0-t7	Variáveis temporárias; não precisam ser preservadas por funções.
16-23	s0-s7	Variáveis salvadas; devem ser preservadas.
24-25	t8-t9	Variáveis temporárias.
26-27	k0-k1	Registradores para o uso do sistema operacional.
28	gp	*global pointer*.
29	sp	*stack pointer*.
30	fp	Apontador para registro de ativações ou variáveis de subrotinas.
31	ra	Endereço de retorno da última chamada de subrotina.

Tabela 2.4: Registradores do MIPS.

O MIPS contém três registradores especiais:

- Contador de Programa (Program Counter - PC): registrador que aponta o endereço de memória da próxima instrução a ser executada;
- *Multiply and Divide register higher result* HI e *Multiply and Divide register lower result* LO: esses registradores armazenam o produto ou o resultado da multiplicação, e depois de uma operação de divisão, o quociente é gravado em LO e o resto em HI;

A arquitetura MIPS32 possui 32 registradores de ponto flutuante (*Floating Point Registers - FPRs*), e cinco registradores de controle de ponto flutuante:

- *Implementation and Revision register -FIR*;
- *Condition Codes register - FCCR*;
- *Exceptions register -FEXR*;
- *Enables register - FENR*; e
- *Control/Status register - FCSR*.

Tipo de dados O MIPS suporta quatro tipos básicos de dados, que são *bytes*, *halfword*, *word* e *doubleword*, com 8, 16, 32 e 64 bits respectivamente.

Os tipos de dados inteiros disponíveis nas instruções LOAD e STORE são bytes, meia palavra e palavra. Nas instruções aritméticas e lógicas são meia palavra (estendida para 32 bits) e palavra, e para dados em ponto flutuante os tipos de dados são precisão simples de 32 bits e precisão dupla de 64 bits.

Modo de endereçamento Como o MIPS é uma arquitetura do tipo LOAD/STORE, apenas instruções de carregamento e armazenamento fazem acessos à memória. As instruções de ALU operam apenas sobre os valores dos registradores.

São cinco os modos de endereçamento: (i) *a registrador*– operandos estão em registradores; (ii) *imediato*– um dos operandos é uma constante na própria instrução; (iii) *base-deslocamento*– o endereço efetivo é a soma do conteúdo de um registrador (base) e de

um imediato de 16 bits (deslocamento); (iv) *relativo ao PC*– o endereço de destino é a soma de um imediato de 16 bits (multiplicado por 4) com o PC; e (v) *pseudo-absoluto*– o endereço efetivo é formado pela concatenação de um imediato de 26 bits (multiplicado por 4) com os 4 bits mais significativos do PC. A multiplicação por quatro é necessária para alinhar a referência com o tamanho de uma instrução.

Os acessos à memória devem ser alinhados como *half-word*s (bit 0 em zero) ou como *word*s (bits 0 e 1 em zero): os dados de 32 bits devem estar em endereços múltiplos de 4 e os dados de 16 bits devem estar em endereços múltiplos de 2. O MIPS, possui algumas instruções para a manipulação de dados não alinhados como LWL, LWR, SWL e a SWR.

O único modo de endereçamento utilizado no acesso à memória é o modo *base-deslocamento*, como na instrução LW: `r`$_d$`, desl16(r`$_b$`);` # `r`$_d \leftarrow$ `M[r`$_b$`+desl16]`. A constante `desl16` é codificada na instrução. O modelo de memória é um vetor de 4Gbytes, referenciado byte a byte.

O modo *registrador* é usado nas instruções aritméticas, lógicas e para uma instrução de desvio incondicional. O modo *relativo ao PC* é usado nas instruções de desvio condicional: o endereço é a soma do PC com deslocamento contido na instrução, o deslocamento é dado em palavras.

No modo *imediato* o dado tem tamanho de 16 bits e é contido na própria instrução, sendo estendido para 32 bits. A extensão é feita com sinal nas instruções aritméticas e sem sinal nas instruções lógicas.

O modo *pseudo-absoluto* é usado para instruções de desvio incondicional. Esta instrução tem campo com endereço de palavra com 26 bits, o endereço da palavra é obtido com os dois bits menos significativos iguais a 0 e os 4 bits mais significativos são obtidos do PC.

Formato da Instruções O MIPS possui três formatos de instruções, do tipo I, R e J. Na Figura 2.2 é mostrado o formato das instruções. Os modos de endereçamento são codificados no *opcode* e todas as instruções são de 32 bits com um opcode primário de 6 bits. Isto facilita a decodificação pelo processador [15].

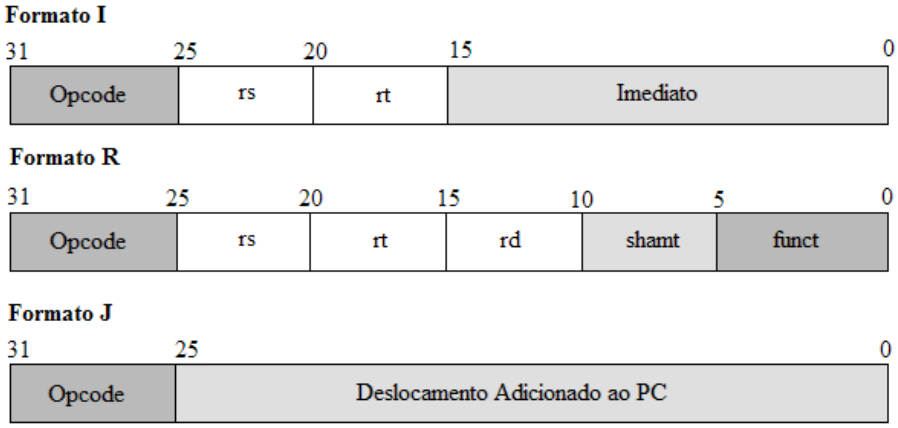

Figura 2.2: Formato das instruções do MIPS.

O *formato I* contém um campo de 16 bits para o imediato, e campos para registradores fonte (r_s) e destino (r_t). Sempre que faz sentido (desvios, memória e aritmética) o campo imediato é codificado em complemento de dois.

O *formato R* contém três campos para identificar dois registradores fonte (r_s e r_t) e um destino (r_d), um campo de 5 bits usado nas instruções de deslocamento (`shamt`), e 6 bits que estendem o `opcode` para definir a função (`funct`).

O *formato J* contém um campo de 26 bits para o endereço pseudo-absoluto.

Algumas instruções de controle do processador ou da unidade de gerenciamento de memória usam formatos distintos destes três formatos básicos.

Tipos de Instruções O MIPS possui quatro classes gerais de instruções, de memória, operações de ULA, desvios e saltos e operações de ponto flutuante.

Instruções de LOAD e STORE transferem dados entre o sistema de memória e o conjunto de registradores da CPU. Há instruções separadas para diferentes propósitos: transferência de diversos tamanhos, tratamento de dados carregados como inteiros com sinal ou sem sinal, acesso a campos desalinhados e proporcionar uma atualização atômica da memória (*read-modify-write*) [7].

Para as instruções LOAD/STORE, qualquer registrador de propósito geral ou de ponto flutuante pode ser carregado ou armazenado na memória. Existem versões das instruções LOAD e STORE nas versões *word*, *half-word* e *byte*: LW, LH, LB, SW, SH, SB.

As instruções executadas na ULA são instruções registrador-registrador. As formas imediatas de instruções de ULA usam um imediato de 16 bits estendido com sinal, ou com zero, que é parte da instrução. As instruções de deslocamento podem deslocar de um valor constante, ou de um valor armazenado num registrador.

Existem instruções para multiplicação e divisão com operandos de 32 bits e resultados em 64 (MUL) ou 32+32 (DIV) para o resto e quociente de uma divisão. Estas instruções usam os registradores especiais hi e lo, e as instruções MFHI e MTHI copiam valores dos registradores de uso geral de/para hi e lo. Existem versões das instruções que verificam a ocorrência de *overflow*, que é sinalizado com uma exceção. As instruções com sufixo U (*unsigned*) não verificam o *overflow* no resultado das somas. Há instruções que estendem constantes para 32 bits preenchendo os bits mais significativos com o sinal (*signed*) ou com zero (*unsigned*).

A instrução LUI (*load upper immediate*) carrega 16 bits imediatos na parte superior do registrador e parte inferior do registrador é preenchida com zeros. A instrução LUI permite que uma constante de 32 bits seja construída em duas instruções: LUI; ORI.

As instruções de comparação são usadas para comparar o valor de dois registradores e armazenar o resultado em um terceiro registrador. O primeiro registrador é comparado com o segundo, caso o valor do primeiro registrador seja menor que o valor do segundo registrador, o registrador de destino terá valor igual a um, ou zero do contrário.

Os desvios incondicionais (saltos) são ditos *pseudo-absolutos* porque o destino é especificado em 26 bits, que apontam para uma instrução numa faixa de 256MB, ou 64M instruções. Os 26 bits do destino são deslocados 2 bits para a esquerda e então concatenados com os 4 bits mais significativos do PC, produzindo um endereço absoluto de 32 bits.

O salto para uma função se dá com a instrução JAL que salta para o destino como numa instrução de salto e armazena no registrador r_{31} o endereço de retorno (PC+4). A instrução JR salta para o endereço contido num registrador e é usada para o retorno de funções (JR r_{31}).

Os desvios condicionais são *relativos ao PC*, com alcance de ±32K instruções. Não

há registrador de status e as decisões são tomadas pela comparação de igualdade entre dois registradores (BEQ, BNE, BEQZ, BNEZ). Desvios por comparação de magnitude necessitam duas instruções: a instrução SLT ($\mathsf{r}_r \leftarrow (\mathsf{r}_i < \mathsf{r}_j)$) determina o resultado da comparação e o salva em um registrador (r_r) que então é comparado com r_0. As instruções SLT $\mathsf{r}_r,\mathsf{r}_i,\mathsf{r}_j$; BNE $\mathsf{r}_r,\mathsf{r}_0,$`dest` equivalem a instrução BLE (*branch-less-than*).

2.4.2 SPARC

SPARC (*Scalable Processor ARChitecture*) é uma arquitetura de processador desenvolvida pela *Sun Microsystems* no período de 1984 a 1987 e define uma arquitetura RISC de propósito geral que visa bom desempenho a baixo custo [37]. Os propósitos desta arquitetura são a otimização de compiladores e a facilidade de implementação da segmentação.

A arquitetura SPARC deriva do RISC 1 [9] e é muito similar à do MIPS. Atualmente a arquitetura está sob a responsabilidade da organização *Sparc Internacional* e o conjunto de instruções de 32 bits é definido em [1]. A arquitetura SPARC utiliza o padrão IEEE Std 754-1985 para 32 e 64 bits.

A principal característica da arquitetura é a *Janela de Registradores*, que permite uma compilação de alto desempenho, assim como a redução de instruções do tipo LOAD/STORE em comparação com outras arquiteturas RISC. Entre os diversos produtos da arquitetura SPARC podemos citar computação militar, fotografia digital, computação aeroespacial, processamento de imagens médicas e telecomunicações.

Ao contrário das outras duas arquiteturas que usam *opcodes* de 6 bits em posições fixas, este CdI usa 2 bits como *opcode* primário e mais um ou dois campos adicionais para definir a instrução.

Registradores O processador SPARC inclui dois tipos de registradores: registradores de propósito geral e registradores de controle/status. As UIs (Unidades de Inteiros) usam registradores de propósito geral que são chamados de registradores r, e as Unidades de Ponto Flutuante (FPUs) usam registradores f.

Uma UI pode conter de 40 a 520 registradores r de 32 bits. Os registradores são divi-

didos em 8 registradores globais, e uma ou mais janelas que contém 24 registradores cada. Uma janela de registradores ainda é particionada em 8 registradores de entrada (*ins*), 8 registradores locais e 8 registradores de saída (*outs*).

Em um determinado momento, uma instrução pode acessar os 8 registradores globais e uma janela de 24 registradores. Os registradores visíveis num dado instante são apontados pelo *Current Window Pointer* (CWP). Cada janela compartilha suas entradas e saídas com as duas janelas adjacentes. As saídas da janela CWP+1 são endereçáveis como as entradas da janela atual, e as saídas da janela atual são as entradas da janela CWP-1. Os registradores locais de cada janela não são compartilhados com as janelas vizinhas, como mostra a Figura 2.3 [1].

O número de janelas *NWINDOWS* varia de 2 a 32, dependendo da implementação. O número total de registradores r em um determinada implementação é 8 (para registradores globals), mais o número de conjuntos × 16 registradores/conjunto. Assim, o número mínimo de registradores r é 40 (2 conjuntos), e no máximo é 520 (32 conjuntos).

A janela atual é definida pelo ponteiro da janela atual (CWP), um contador de 5 bits num campo no Registrador de Estado (*Processor State Register (PSR)*). O CWP é incrementado por uma instrução RESTORE (ou RETT) e decrementado por uma instrução SAVE ou um *trap*. *Overflow* e *underflow* de janelas são detectados através do registrador *Window Invalid Mask (WIM)*, que é controlado por software de nível privilegiado.

Um registrador r com endereço o, onde $8 \leq o \leq 15$, é referenciado como $(o+16)$ depois de CWP ser incrementado por 1. Um registrador com endereço i, onde $24 \leq i \leq 31$, é referenciado como endereço $(i - 16)$ depois que CWP é decrementado por 1.

Os registradores de controle/status de 32 bits incluem o *Processor State Register (PSR)*, *Window Invalid Mask register (WIM)*, *Trap Base Register (TBR)*, o registrador *multiply/divide (Y)*, o contadores de programa (PC e nPC), e o opcionalmente, o *Ancillary State Registers (ASRs)* e a fila UI *deferred-trap*.

Tipo de dados A arquitetura SPARC também define quatro tipos de dados fundamentais: inteiro com sinal e sem sinal de 8, 16, 32, e 64 bits e mais dois tipos diferentes

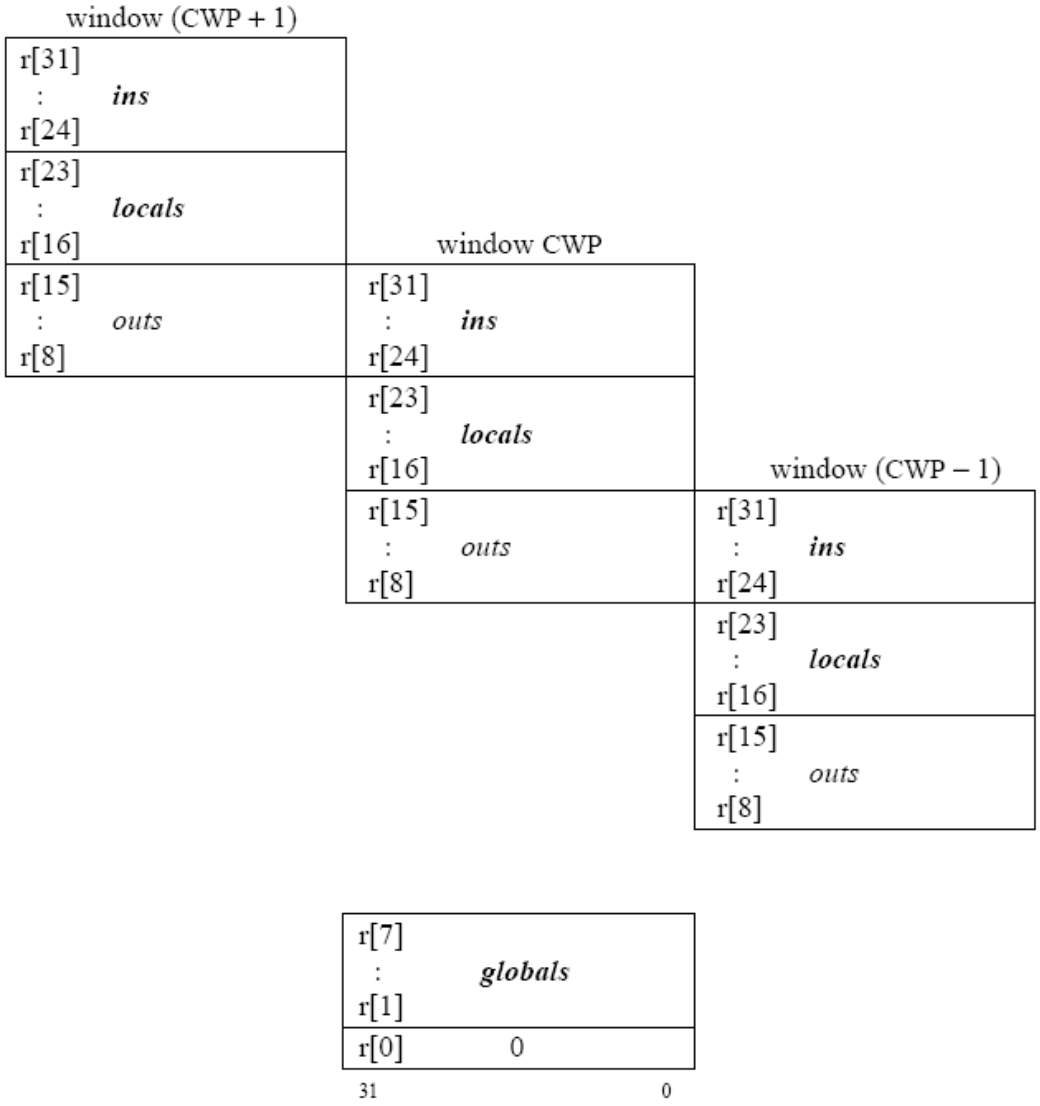

Figura 2.3: Janela de Registradores e os 8 Registradores Globais.

que não existem nas outras duas arquiteturas: *Tagged Word* - 32 bits (valor de 30 bits mais 2 tag bits) e *Quadword* - 128 bits;

Modo de endereçamento O acesso à memória pode ser com endereçamento *indexado* e *base-deslocamento*. O par de instruções LDD e STD permite acessar operandos de 64 bits em memória. Imediatos são especificados com 13 bits. O destino de um salto é especificado em 30 bits, com um *opcode* de 2 bits.

Formato das Instruções As instruções estão codificadas em três grandes formatos, como pode ser visto na Figura 2.4 [1]. O primeiro formato é utilizado para instrução

CALL, o campo `op` é um *opcode* de 2 bits e `disp30` uma constante de 30 bits.

O segundo formato é usado para instruções SETHI e desvios. O campo `op2` com 3 bits também é utilizado para codificar as instruções, o r_d é o registrador de destino com 5 bits. O campo `a` com 1 bit indica se a próxima instrução será executada ou anulada (*branch delay slot*) dependendo da condição do salto, o `cond` com 4 bits seleciona os códigos de condição para testar uma instrução de desvio. O campo `imm22` é uma constante de 22 bits usada pela instrução SETHI e `disp22` também é uma constante de 22 bits usada para as outras instruções.

O terceiro formato é utilizado por instruções de acesso à memória e instruções de ULA. O campo `op3` com 6 bits, codifica as instruções. O r_{s1} possui 5 bits e é o registrador do primeiro operando e r_{s2} é o registrador do segundo operando se o campo `i = 0`. O campo `asi` (*address space identifier*) é usado por instruções LOAD/STORE e possui tamanho de 8 bits. O `simm13` é um campo usado para valores imediatos com tamanho de 13 bits e pode ser estendido para 32 bits. O campo `opf` é utilizado para codificar instruções de ponto flutuante, com 9 bits.

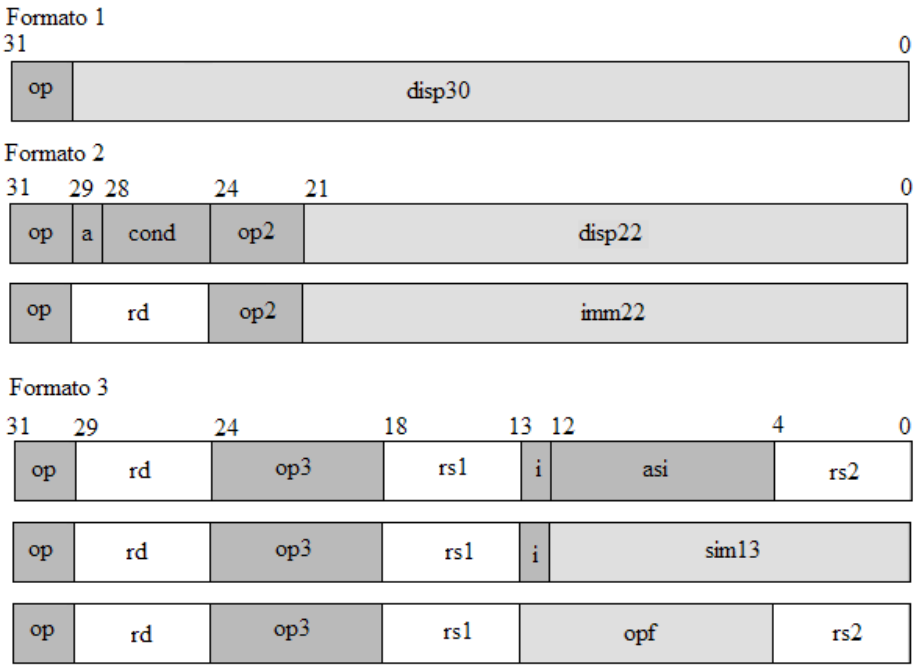

Figura 2.4: Formato da Instruções do SPARC.

Tipos de Instruções As instruções do SPARC podem ser agrupadas em cinco categorias: LOAD/STORE, aritmética de inteiros, transferência de controle (CTI), operações de ponto flutuante e operações de coprocessador.

Somente instruções LOAD/STORE acessam a memória. Estas instruções usam dois registradores ou um registrador `r` e uma constante para calcular endereço de memória de 32 bits alinhado. O campo de destino de uma instrução LOAD/STORE especifica ou um registrador `r`, `f`, (ou um registrador do coprocessador) que fornece os dados para um STORE, ou recebe os dados de um LOAD. Instruções LOAD e STORE suportam as larguras *byte* (8-bit), *halfword* (16-bit), *word* (32-bit), e *doubleword* (64-bit).

As instruções de aritmética de inteiros são geralmente instruções que utilizam três registradores, as quais calculam o resultado que é uma função de dois operandos fonte, e escrevem o resultado em um registrador destino r_d ou o resultado é descartado. Um dos operandos fonte sempre é r_{s1}. O outro operando fonte depende do bit i da instrução: se `i = 0`, o operando é r_{s2}, mas se `i = 1`, o operando e uma constante `simm13` estendida para 32 bits.

As instruções de lógica e aritmética podem, opcionalmente, alterar bits de status no registrador de condição $\mathsf{r_{cc}}$ (*condition code register*). Comparações de magnitude são obtidas por subtração de r_0.

As instruções de transferência de controle mudam o valor do *next program counter* (nPC). Há 5 tipos básicos de instruções de transferência de controle.

O desvio condicional transfere o controle se a condição especificada é verdadeira. Se o bit `annul` é 0, a instrução no *delay slot* é sempre executada. Se o bit `annul` é 1, a instrução no *delay slot* não é executada a menos que o desvio condicional seja tomado.

Um desvio incondicional transfere o controle incondicionalmente se a condição especificada é "sempre", e nunca transfere o controle se a condição especificada é "nunca". Se o bit `annul` é 0, a instrução no *delay slot* é sempre executada. Se o bit `annul` é 1, a instrução no *delay slot* nunca é executada.

Sempre que possível, os registros de ativação são montados pelo compilador diretamente nas janelas de registradores, aumentando a eficiência de chamada e retorno de

funções. Quando todas as janelas estão ocupadas, uma nova requisição provoca uma excessão que salva registradores em memória (*spilling*), liberando o espaço necessário para a nova janela.

A instrução CALL escreve o conteúdo do PC no r_{15} (registrador de saida 7) e então causa uma transferência de controle atrasada para o endereço efetivo relativo ao PC. O valor escrito em r_{15} é visível para instrução no *delay slot*.

A instrução JMPL escreve o conteúdo do PC no r_d e então causa uma transferência de controle atrasada para o endereço efetivo relativo ao PC. O valor escrito em r_d é visível para instrução no *delay slot*.

A instrução RETURN é usada para o retorno do tratador de excessão em modo não-privilegiado. RETURN combina as características de transferência de controle da instrução JMPL com o r_0 especificado como o registrador destino e efetua a troca de janela de registradores como a instrução RESTORE.

A instrução TRAP inicia uma ação em resposta a presença de uma exceção ou uma interrupção. O código de condição especificado pelo campo `cond` é avaliado, se o resultado for verdadeiro, causa um *trap*, senão executa como um NOP.

2.4.3 PowerPC

A arquitetura do PowerPC (*Power Performance Computing*) é um descendente do projeto pioneiro do processador 801 na IBM [11], sendo PowerPC 601 o primeiro membro da nova família [22]. O PowerPC (PPC) foi lançado em 1991, como um projeto comum da IBM, Motorola e Apple. Os processadores PowerPC tornaram-se famosos por equipar máquinas Macintosh da Apple.

A arquitetura PowerPC foi projetada para permitir mudanças na sua implementação com facilidade, logo a flexibilidade é uma das características mais importantes desta arquitetura. O aspecto mais notável da flexibilidade da arquitetura é o fato de estarem incluídos definições de 32 e 64 bits, garantindo que aplicativos escritos para processadores de 32 bits executem em processadores de 64 bits sem problemas de compatibilidade.

Um outro aspecto interessante da arquitetura PowerPC é que ela é dividida em três

níveis que correspondem ao três ambientes de programação, que são:

- A arquitetura do conjunto de instruções do usuário (*User Instruction Set Architecture - UISA*): inclui as instruções de nível de aplicação do usuário e os registradores acessados por essas instruções. A UISA também define parâmetros relacionados à instrução, tais como modos de endereçamento e formatos de instruções;

- A arquitetura de ambiente virtual (*Virtual Enviroment Architecture - VEA*): descreve o modelo de memória por meio do qual dispositivos podem acessar a memória, e define o modelo de cache e a base de tempo. A VEA define os recursos usados primeiramente pelos compiladores;

- A arquitetura de ambiente operacional (*Operating Enviroment Architecture - OEA*): define o modelo de gerenciamento de memória, o modelo de exceção e, principalmente, os recursos utilizados pelos sistemas operacionais.

Entre os diversos produtos que utilizam a arquitetura PowerPC podemos citar video games, telefones celulares, PDAs e servidores de arquivos e computação.

Diferentemente do MIPS, os projetistas deste CdI optaram por definir instruções mais complexas do que seria de se esperar num processador RISC. Isso decorre da implementação super-escalar com três unidades funcionais que operam em paralelo: uma unidade de inteiros, uma de busca e desvios, e uma de ponto flutuante, de tal forma que operações mais complexas podem ser efetuadas em cada segmento do processador. Decorreu quase uma década entre as definições dos CdIs MIPS e PowerPC e isso transparece na complexidade das implementações iniciais do PowerPC e portanto no projeto de seu CdI.

A definição da arquitetura é controlada pela *Power.org* e a última versão do CdI encontra-se em [30]. As diferenças com relação ao MIPS são discutidas abaixo.

Registradores A maioria das instruções do PowerPC define operações registrador-registrador. A arquitetura possui 32 GRPs (*General-Purpose Registers*), 32 FPRs (*Floating-Point Registers*), SPRs (*Registradores de Uso Específico*) e vários outros registradores.

O PowerPC possui dois níveis de privilégio, nível de usuário e nível supervisor, e todas as instruções e registradores definidos pelo UISA são acessados pelo software no nível de usuário. A VEA proporciona acesso de leitura no nível de usuário aos registradores de base de tempo. Todos os demais registradores, definidos pela OEA, são acessados somente pelo nível supervisor, normalmente usado por sistemas operacionais.

O conjunto de registradores UISA é composto por:

- Registradores de Uso Geral (*General-Purpose Registers - GPRs*): dados inteiros são manipulados nos 32 GRPs de 32 bits. Os GPRs são acessados como registradores de fonte ou destino;

- Registrador de Exceção de Inteiros (*Fixed-point Exception Register - XER*): o registrador XER possui o tamanho de 32 bits e é usado para registrar e controlar exceções de operações com inteiros;

- Registradores de Ponto Flutuante (*Floating-Point Registers - FPRs*): o PowerPC possui 32 FPRs de 64 bits. Estes registradores são acessados como registradores de fonte ou destino por instruções de ponto flutuante. Cada FPR suporta o formato de ponto flutuante de precisão dupla;

- Registrador de Status e Controle de Ponto Flutuante (*Floating Point Status e Control Register - FPSCR*): este registrador possui tamanho de 32 bits e é usado para registrar exceções geradas por operações de ponto flutuante, registrar o tipo de resultado produzido por operações de ponto flutuante e controlar o modo de arredondamento usado por operações de ponto flutuante;

- Registrador de Condição (*Condition Register - CR*): este registrador é de 32 bits, indica o resultado de determinadas operações de lógica e aritmética e fornece um mecanismo de teste e desvio. Os bits no registrador CR são agrupados em oito campos de 4 bits, CR0 a CR7 e são interpretados por instruções de condição de desvio (*branch condition*);

- Registrador de Ligação (*Link Register - LR*): o LR é um registrador de 32 bits, utilizado para dar suporte a execução de subrotinas, mais precisamente para instruções BL (*branch and link*) e BLR (*branch to link register*) e é usado para armazenar o endereço de retorno de uma subrotina; e

- Registrador de Contagem (*Count Register - CTR*): o CTR é um registrador de 32 bits, utilizado em algumas instruções de desvio. Ele funciona como um contador que é decrementado e usado no teste de desvio de um laço.

O VEA define registradores além daqueles definidos pelo UISA. O conjunto de registradores VEA pode ser acessados por todas as aplicações com privilégio de usuário ou administrador.

A arquitetura PowerPC VEA oferece acesso a uma base de tempo (*time base - TB*), que é uma estrutura de 64 bits que consiste de dois registradores de 32 bits: o *Time base upper* (TBU) que armazena a parte significativa e o *Time base lower* (TBL) que armazena a parte menos significativa. Esses dois registradores são do tipo "tempo", e são apenas de leitura.

Os registradores que são acessados somente por software supervisor são todos definidos pela OEA. São usados na configuração do sistema, tratamento de exceção e etc.

Tipo de dados A arquitetura PowerPC define um conjunto básico de tipos de dados que são conhecidos pelo processador e manipulados diretamente com instruções individuais. Os tipos de dados incluem: *unsigned byte*, *unsigned halfword*, *signed halfword*, *unsigned word*, *signed word* (somente em implementações de 64 bits), *unsigned doubleword*, *byte string* e *single-precision floating point* e *double-precision floating point* [16].

Modo de endereçamento O PowerPC especifica duas maneiras de endereçamento de memória, o modo *indireto* e *indireto-indexado*, e ambos possuem a opção de atualizar o registrador de base.

No modo de *endereçamento indireto* (base-deslocamento), a instrução inclui um deslocamento de 16 bits, a ser adicionado a um registrador base, que pode ser qualquer um

dos registradores de propósito geral. Adicionalmente, a instrução pode especificar que o novo endereço efetivo seja gravado no registrador base. A opção de atualização é útil para indexação progressiva de vetores em laços.

No modo *indireto-indexado* (indexado), a instrução referencia um registrador base e um registrador índice, que podem ser quaisquer registradores de uso-geral. O endereçamento efetivo é a soma do conteúdo destes dois registradores. A opção de atualização tem como efeito a atualização do registrador base com o novo endereço efetivo.

Há instruções para carregar e armazenar cadeias de caracteres de tamanho e alinhamento arbitrários (LSW e STSW).

Os modos de endereçamento adicionais são: (i) *indexado*– o endereço efetivo é a soma do conteúdo de dois registradores (base e índice); (ii) *base-deslocamento com incremento* e *indexado com incremento*– o registrador base é atualizado com o endereço efetivo recém computado: LWU: $\mathtt{r}_d \leftarrow \mathtt{M[}\langle x = \mathtt{r}_b + \mathtt{r}_i \rangle\mathtt{]}$; $\mathtt{r}_b \leftarrow x$. O destino de um salto é especificado em 24 bits, com um *opcode* de 6+2 bits.

Formato da Instruções Todas as instruções do PowerPC têm 32 bits e seguem um formato regular. Os primeiros 6 bits da instrução especificam a operação a efetuar.

As instruções devem ser alinhadas como palavras, assim quando o endereço da instrução é apresentada ao processador (como em instruções de desvios) os dois bits menos significativos do endereço são ignorados.

Na Figura 2.5 são mostrados alguns dos principais formatos de instruções do PowerPC. O campo `Op` é o opcode principal e possui tamanho de 6 bits, `Opx` é uma extensão do opcode e pode ter 2 ou 6 ou 11 bits, $\mathtt{r}_d$ é o registrador de destino, $\mathtt{r}_{s1}$ é o registrador de origem 1 e $\mathtt{r}_{s2}$ é o registrador de origem 2. O campo `Const` é uma constante (usada como um imediato ou como um endereço) [15].

Tipos de Instruções As instruções do PowerPC estão classificada nas seguintes categorias: instruções de inteiros; instruções de ponto flutuante; instruções de LOAD/STORE e instruções de controle de fluxo.

As instruções de inteiros consistem de instruções de aritmética, comparação, lógica,

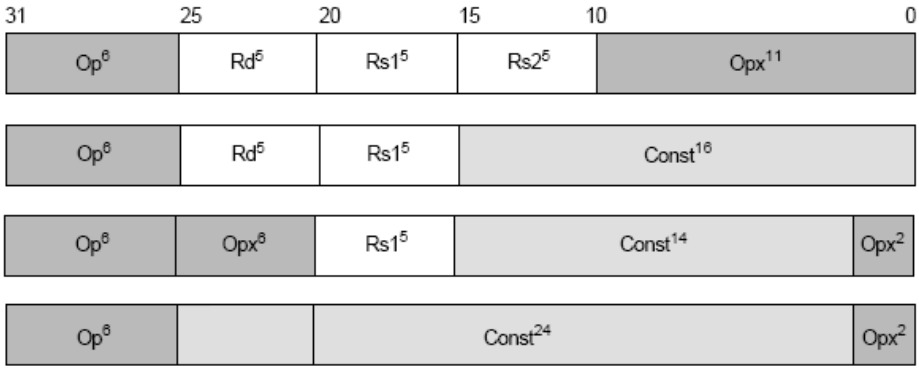

Figura 2.5: Formato da Instruções do PowerPC.

rotação e deslocamento. Estas instruções tratam os operandos fonte como inteiros com sinal, a menos que a instrução especifique uma operação sem sinal, como por exemplo as instruções MULHWU (*Multiply High-Word Unsigned*) e DIVWU (*Divide Word Unsigned*).

As instruções de lógica e aritmética com dois operandos podem, opcionalmente, alterar bits de status no registrador de condição r_{cr}. O registrador r_0 é um registrador de uso geral.

Instruções de LOAD e STORE são emitidas e traduzidas na ordem do programa, mas os acessos à memória podem ocorrer fora de ordem. Há instruções de sincronização para garantir a ordem estrita das referências à memória.

Nos desvios condicionais o registrador de condição r_{cr} tem 4 bits de status que podem ser opcionalmente definidos por uma instrução de lógica ou aritmética, e a unidade de desvios contém 8 cópias de r_{cr} para manter o status de até 8 instruções. A unidade de desvios fica próxima do segmento de busca permitindo que os desvios tenham efeito imediato, sem *delays slots*. Um registrador r_{ctr} mantém a contagem de iterações de laços, de forma que uma única instrução de desvio decrementa r_{ctr}, testa seu valor e desvia se aquele for diferente de zero.

O registrador de ligação (*link register* r_{lk}) é dedicado para manter o endereço de retorno na chamada de funções.

CAPÍTULO 3

AMBIENTE DE SIMULAÇÃO

Este capítulo apresenta o ambiente de simulação e está dividido da seguinte forma: na seção 3.1 é discutida a *linguagem de descrição de arquiteturas ArchC*, que é a base dos simuladores empregados. Na seção 3.2 são descritos os conjuntos de *benchmarks* utilizados e na seção 3.3 é apresentada a metodologia de simulação e caracterização dos programas de teste.

3.1 ArchC

Os simuladores empregados para gerar as contagens de instruções são baseados em *ArchC* e foram obtidos de `http://www.archc.org`.

A *linguagem de descrição de arquiteturas ArchC* [34] é baseada na linguagem SystemC [38] e permite construir automaticamente um simulador para uma arquitetura alvo. Entre os simuladores implementados com ArchC [4], estão disponíveis os processadores MIPS, SPARC-V8, PowerPC, Intel8051 e PIC 16F84.

Com ArchC é possível descrever processadores na forma comportamental, apenas indicando o que cada instrução faz ou com precisão de ciclos indicando o que cada instrução faz em cada estágio de execução. Com base nessa descrição, as ferramentas da linguagem são capazes de gerar simuladores para os processadores.

Para gerar um simulador com ArchC é necessário descrever as características básicas da organização do processador e descrever a sintaxe e a semântica do conjunto de instruções. Os arquivos com a descrição da organização (arquitetura), e do conjunto de instruções são submetidos ao pré-processador do ArchC, que gera automaticamente o código do simulador. Um esqueleto do arquivo com a semântica do conjunto de instruções é gerado pelo pré-processador e deve ser preenchido com a descrição do comportamento das instruções [3].

O programador é responsável pelo desenvolvimento dos arquivos com a descrição da arquitetura, com o formato ou sintaxe do conjunto de instruções, e com a semântica ou o comportamentos da instruções. A Figura 3.1 exemplifica um arquivo com a descrição da arquitetura MIPS de 32 bits, sem considerar seu *pipeline*. A Figura 3.2 exemplifica um trecho do arquivo com a descrição da sintaxe das instruções do modelo MIPS **??**.

```
AC_ARCH(mips){

  //Tamanho da palavra em bits
   ac_wordsize 32;

  //Tamanho da memória
   ac_mem MEM:256k;

  //Banco de registradores
   ac_regbank RB:32;

   ARCH_CTOR(mips) {

      ac_isa("mips_isa.ac");
      set_endian( "big" );
   };
};
```

Figura 3.1: Descrição da arquitetura MIPS no ArchC.

Uma característica importante do ArchC é a capacidade dos simuladores gerados emularem chamadas ao sistema operacional [19]. Com isso, aplicações que efetuam operações de entrada/saída, como leitura e escrita em arquivos, também podem ser simuladas.

Os simuladores implementados com ArchC produzem algumas métricas de desempenho tais como, o número que ciclos executados, uma aproximação do tempo de execução, quantidade de instruções executadas, número de chamadas de E/S e quantidade de acessos a cada instrução e à memória [35].

3.2 Conjunto de Programas de Teste

O conjunto de programas de teste escolhido para este trabalho contém aplicações geralmente utilizadas em sistemas embarcados. Os programas da suíte CommBench [40]

```
AC_ISA(mips){

   //Define o tipo das instruções e os tamanhos de cada campo
   ac_format Type_R = "%op:6 %rs:5 %rt:5 %rd:5 0x00:5 %func:6";
   ac_format Type_I = "%op:6 %rs:5 %rt:5 %imm:16:s";
   ac_format Type_J = "%op:6 %addr:26";

   //Define quais instruções pertence a qual tipo
   ac_instr<Type_R> add, addu, subu, multu, divu, sltu;
   ac_instr<Type_I> lw, sw, beq, bne;
   ac_instr<Type_I> addi, andi, ori, lui, slti;
   ac_instr<Type_J> j, jal;

   ISA_CTOR(mips){

      //Define a sintaxe da instrução lw
      lw.set_asm("lw %rt, %imm(%rs)");
      lw.set_decoder(op=0x23);

      //Define a sintaxe da instrução sw
      sw.set_asm("sw %rt, %imm(%rs)");
      sw.set_decoder(op=0x2B);

     //Define a sintaxe da instrução add
      add.set_asm("add %rd, %rs, %rt");
      add.set_decoder(op=0x00, func=0x20);

      //Define a sintaxe da instrução addi
      addi.set_asm("addi %rt, %rs, %imm");
      addi.set_decoder(op=0x08);
      ...
   };
};
```

Figura 3.2: Descrição parcial do conjunto de instruções do MIPS no ArchC.

foram usados para a coleta da maioria dos dados e nos experimentos mais detalhados. Os programas das suítes *MediaBench* [6] e *MiBench* [21] são utilizados principalmente para comparações com os resultados obtidos com o CommBench.

3.2.1 CommBench

Os programas da suíte CommBench são relativamente pequenos e são núcleos de aplicações típicas de processadores de rede. O conjunto contém programas de *processa-*

mento de cabeçalho, que examinam e processam o cabeçalho de mensagens, e de *processamento de conteúdo*, que acessam e/ou modificam os conteúdos de um fluxo de pacotes.

Os programas de processamento de cabeçalho empregados nas medições são RTR (*Radix-Tree Routing*), FRAG (fragmentação de pacotes IP), e DRR (escalonamento *Deficit Round Robin*). Os programas de processamento de conteúdo empregados são CAST (cifração *CAST-128*), ZIP (compressão Lempel-Ziv), REED (codificação Reed-Solomon) e JPEG (algoritmo de compressão de imagens). Os programas estão descritos abaixo:

- **RTR** - (*Radix-Tree Routing*) programa de consultas em tabelas de roteamento. Consultas em tabelas de roteamento são operações importantes realizadas em todos os pacotes em redes baseadas em datagrama e em cada conexão em redes baseadas em conexão. *Kernel*: operações de consultas em estrutura de dados representado como uma árvore.

- **FRAG** - aplicação de fragmentação de pacotes IP. Pacotes IP são divididos em vários fragmentos, nos quais alguns campos do cabeçalho têm de ser ajustados e um *checksum* do cabeçalho computado. *Kernel*: Computação do *checksum* e modificações no cabeçalho dos pacotes.

- **DRR** - algoritmo de escalonamento *Deficit Round Robin* para a divisão de recursos e é comumente utilizado para o escalonamento de largura de banda de enlaces de rede. A aplicação lê descrições de pacotes, coloca-os em uma representação interna de fila e faz um escalonamento para a transmissão. Quando um pacote é transmitido, a sua descrição interna é excluída. O algoritmo é implementado em vários comutadores comercialmente disponíveis. *Kernel*: fila de manutenção e de escalonamento de pacotes para uma utilização justa dos recursos.

- **CAST** - programa baseado no algoritmo de cifra de bloco CAST-128 que usa uma chave de 128 bits para criptografar dados para transmissão segura, que codifica e decodifica dados usando uma chave simétrica. A computação principal consiste de

criptografia aritmética. CAST-128 funciona de forma semelhante a outros algoritmos de cifra de bloco utilizados em redes atuais, tais como IDEA e RC5, mas ao contrário destes, CAST está em domínio público. *Kernel*: criptografia aritmética.

- **ZIP** - implementação GNU (gzip-1.2.4) do algoritmo de compressão Lempel-Ziv (LZ77). A implementação pode utilizar níveis diferentes de compressão de dados através da variação da complexidade computacional do algoritmo e exemplifica aplicações que permitem *tradeoffs* entre poder computacional e largura de banda. *Kernel*: compressão de dados.

- **REED** - implementação do esquema *Reed-Solomon Forward Error Correction* que adiciona redundância aos dados para permitir a recuperação de erros na transmissão. É comumente usado em *link* de dados não confiáveis, que podem ser encontrados em redes sem fio. *Kernel*: codificação de redundância.

- **JPEG** - algoritmo de compressão para dados de imagem, representativo da classe das aplicações de transcodificação de mídia, versão *jpeg-6b*. *Kernel*: Transformada Discreta de Cosseno (DCT) e codificação Huffmann.

3.2.2 MediaBench

O MediaBench é uma suíte com aplicações multimídia e de sistemas de comunicações, utiliza apenas linguagem de alto nível a fim de enfatizar a tecnologia de compilação. Neste trabalho são utilizados 7 programas da suíte MediaBench que são descritos a seguir:

- **ADPCM** - o ADPCM (*Adaptive Differential Pulse Code Modulation*) é uma dos mais simples e mais antigas formas de codificação de áudio.

- **EPIC** - aplicação experimental de compressão de imagem. Os filtros desta aplicação foram projetados para permitir uma decodificação extremamente rápida.

- **G721** - compressão de voz G.721: implementações de referência CCITT (*International Telegraph and Telephone Consultative Committee*) G.711, G.721 e G.723 para compressão de voz.

- **GSM** - Padrão Provisório Europeu GSM 06.10 para transcodificação *fullrate* de fala, prI-ETS 300 036.

- **JPEG** - método de compressão padronizado para imagens coloridas e escalas de cinza, versão *jpeg-6a*.

- **MPEG2** - padrão atual de alta qualidade para transmissão de vídeo digital. A computação principal é uma Transformada Discreta de Cosseno para codificação, e a Transformada inversa para decodificação.

- **PEGWIT** - programa para a codificação e autenticação de chave pública. Usa uma curva elíptica sobre GF (2255), SHA1 para *hash*, e uma cifra de bloco quadrada simétrica.

3.2.3 MiBench

A suíte MiBench é constituída por seis categorias: Automotiva e Controle Industrial, Rede, Segurança, Eletrônica de Consumo, Automação de Escritório e Telecomunicações. Estas categorias oferecem programas com diferentes características que permitem aos pesquisadores examinar os seus projetos de uma forma mais eficaz para um determinado segmento de mercado [21].

Neste trabalho são utilizados 13 programas da suíte MiBench de categorias diferentes. A seguir são descritos as categorias e os programas.

Automotiva e Controle Industrial Esta categoria é representativa da utilização de processadores em sistemas de controle embarcado. Estes processadores necessitam desempenho em habilidades matemáticas básicas, manipulação de bits, dados de entrada/saída e uma organização simples de dados. Aplicações típicas são controladores de *air bag*, monitores de desempenho de motor e sistemas de sensores. As aplicações utilizadas desta categoria são:

- **Basicmath** - executa cálculos matemáticos simples que muitas vezes não têm suporte dedicado no hardware de processadores embarcados, como por exemplo,

funções de solução cúbica, raiz quadrada inteira e conversões de ângulos em graus para radianos.

- **Bitcount** - algoritmo testa habilidades de manipulação de bits de um processador, com a contagem do número de bits em um vetor de inteiros.
- **Qsort** - ordena um vetor grande de strings em ordem crescente usando o algoritmo Quick Sort.
- **Susan** - pacote de reconhecimento de imagem que foi desenvolvido para o reconhecimento de cantos e bordas em imagens de ressonância magnética do cérebro.

Rede Esta categoria é representativa de aplicações de dispositivos de rede como *switches* e roteadores. O trabalho realizado por estes aplicativos envolve cálculos de menor caminho, tabelas de pesquisas e dados de entrada/saída. O algoritmo utilizado é:

- **Dijkstra** - constrói um grafo em uma representação de matriz adjacente e, em seguida, calcula o menor caminho entre cada par de nodos usando repetidas aplicações do algoritmo de Dijkstra. O algoritmo de Dijkstra é uma solução bem conhecida para o problema do menor caminho e executa em tempo de $O(n^2)$.

Segurança Segurança de dados é de grande importância na Internet, principalmente com a popularidade das atividades de comércio eletrônico. A categoria Segurança inclui algoritmos comuns para a criptografia dos dados, decodificação e *hashing*. As aplicações utilizadas desta categoria são:

- **SHA** - algoritmo de embaralhamento seguro que produz mensagens de 160 bits para uma dada entrada. Frequentemente utilizado na troca segura de chaves criptografadas e para gerar assinaturas digitais. Também é utilizado nas conhecidas funções *hashing* MD4 e MD5.
- **Rijndael** - Rijndael foi selecionado como o *Advanced Encryption Standard* (AES) pelo *National Institute of Standards and Technologies*. Trata-se de uma cifra de bloco com opção, de chaves e de blocos, de 128, 192, e 256 bits.

Eletrônica de Consumo Os programas desta categoria são usados em dispositivos de grande popularidade como *scanners*, câmeras digitais e assistentes pessoais digitais (PDAs). A categoria se concentra principalmente em aplicações multimídia, e a aplicação representante desta categoria é:

- **JPEG** - descrito anteriormente, no *MiBench* a versão também é *jpeg-6a*.

Automação de Escritório As aplicações de escritório contém essencialmente algoritmos de manipulação de texto encontradas em impressoras, fax e processadores de texto. A aplicação utilizada desta categoria é:

- **Stringsearch** - procura por palavras dadas em frases usando um algoritmo de comparação *case insensitive*.

Telecomunicações A importância da categoria de Telecomunicações se deve aos dispositivos portáteis que são usados na comunicação sem fio. Os *benchmarks* desta categoria consistem de algoritmos de codificação e decodificação de voz, análises de frequência e um algoritmo de computação de *checksum*.

- **FFT/IFFT** - computa *Fast Fourier Transform* e o inverso da transformada em um vetor de dados. As transformadas de Fourier são utilizadas em processamento de sinais digitais para encontrar as freqüências contidas em um determinado sinal de entrada.

- **GSM** - *Global Standard for Mobile* é o padrão para codifição/decodificação de voz na Europa e em outros países. Usa uma combinação TDMA/FDMA (*Time and Frequency Division, Multiple Access*) para codificar/decodificar fluxos de dados.

- **ADPCM** - o *Adaptive Diferencial Pulse Code Modulation* (ADPCM) é uma variação do padrão *Pulse Code Modulation* (PCM).

- **CRC32** - computa o *Cyclic Redundancy Check* (CRC) de 32 bits de um arquivo. O CRC é frequentemente utilizado para detectar erros na transmissão dos dados.

3.3 Metodologia de Simulação e Caracterização

Como mencionado na seção 3.1, os simuladores foram implementados com a linguagem de descrição de arquitetura *ArchC* e foram obtidos de [4]. Os modelos simulados dos três processadores foram escritos com base em [17] (MIPS), [28] (SPARC), e [41] (PowerPC). Estes simuladores são chamados de *funcionais* porque não computam informação de tempo, sendo a função de cada instrução computada atomicamente *em um ciclo de simulação*, que não corresponde a um ciclo de relógio.

A versão do ArchC utilizado é 2.0 e a versão do SystemC é 2.0.1. O modelo MIPS corresponde ao MIPSI, o SPARC ao SPARC-V8 e o PowerPC ao PowerPC 405.

Todos os programas foram compilados com a versão 3.3.1 do GCC que é disponibilizado com os simuladores, com otimização `-O3` e ligados estaticamente com bibliotecas que emulam as chamadas de sistema. O modelo do processador SPARC usa 256 registradores e suporta 16 janelas de registradores, o dobro de implementações típicas. Com este modelo de processador, o código gerado para suportar funções é otimista.

Os comandos utilizados nas compilações são mostrados a seguir de uma forma genérica:

- `mips-elf-gcc -specs=archc -msoft-float -O3 executável código.c`
- `powerpc-elf-gcc -specs=archc -msoft-float -O3 executável código.c`
- `sparc-elf-gcc -specs=archc -msoft-float -O3 executável código.c`

As entradas utilizadas para os programas são disponibilizadas juntos com os *benchmarks* e as saídas dos programas foram comparadas com as saídas padrões também disponibilizadas junto com os *benchmarks*. Além disso, os resultados obtidos foram comparados com a saída padrão gerada pelos simuladores.

A caracterização de cargas consiste em uma descrição da carga de trabalho por meio de parâmetros e técnicas de caracterização, com o objetivo de mostrar, capturar e reproduzir o comportamento da carga de trabalho e de suas características mais importantes.

A caracterização reduz o custo de exploração do espaço do projeto, focaliza a arquitetura em análise para a carga de trabalho pretendida e diminui a quantidade de simulação

e reformulação do projeto requerido [39]. Existem diversas técnicas usadas para a caracterização de carga de trabalho, tais como técnicas exploratórias de análise de dados, técnicas de análise numérica, técnicas estatísticas e clusterização [20].

Para caracterizar os programas e os conjuntos de instruções nos experimentos foram coletadas as contagens das instruções dinâmicas, a distribuição dos acessos à memória e a distribuição das instruções de desvios. Para coletar o número de instruções executadas foram adicionados ao código do simulador contadores para cada instrução.

As distribuições dos acessos à memória são obtidas das distâncias entre instruções LOAD/STORE nas sequências de execução. A distribuição dos desvios é medida da mesma forma. A **distância** é o número de instruções entre os dois elementos do par, sendo que **distância zero** significa duas instruções consecutivas. Um função simples foi adicionada ao código do simulador para obter as distâncias, abaixo um exemplo de trecho do código:

```
//se a instrução é um store
if (instrucao == "sw"){
   //se a distância entre dois stores for menor ou igual a 15
    if (vdiststores <= 15) {
       //incrementa o contador da distância entre stores
      dstores[vdiststores] = dstores[vdiststores] + 1;
    } else {
         if (vdiststores > 15)
            dstores[16] = dstores[16] + 1;
    }
   //zera contador para começar a contagem entre as distâncias novamente
    vdiststores = 0;
} else {
   //vai incrementando o contador de distância enquanto não encontra um store
    vdiststores++;
}
```

CAPÍTULO 4

RESULTADOS E ANÁLISE

Este capítulo apresenta os resultados obtidos com uso dos simuladores. Primeiramente fazemos uma comparação entre os três conjuntos de programas de teste para os três processadores, em seguida fazemos uma análise do perfil de execução dos processadores considerando a contagem total de instruções para cada classe de instrução.

Como mencionado anteriormente, a suíte CommBench é o principal conjunto de programas usado para a apresentação dos resultados. Por isso, as aplicaçãos do CommBench são usadas para avaliar o efeito de três níveis de otimização no código gerado pelo compilador e analisar a distribuição no tempo de referências à memória.

4.1 Comparação

A comparação entre as contagens de instruções executadas **não é um bom indicador de desempenho** porque o número de instruções executadas é somente um dos três fatores da equação que determina o tempo de execução de um certo programa num processador, que é

$$tempo = \mathcal{N} \times CPI \times \mathcal{T},$$

onde $\mathcal{N}$ é o número de instruções executadas, CPI é o número médio de ciclos por instrução e $\mathcal{T}$ é o período do relógio do processador. O conjunto de instruções interfere no número de instruções executadas e sua implementação determina o CPI e o período do relógio. Além disso, o código gerado pelo compilador tem grande influência em $\mathcal{N}$ e no CPI ao gerar código com instruções 'rápidas' ou 'lentas'. Mantidas em igualdade as demais condições, diferentes técnicas de implementação do conjunto de instruções tem grande influência no desempenho, como as duas implementações do IA-32 no 80386 e no 80786 (Pentium IV).

A Tabela 4.1 mostra as contagens e freqüências das classes de instruções de todos os programas e também os totais para o MiBench, MediaBench e o CommBench. As

contagens individualizadas das aplicações da suíte CommBench são mostradas no Apêndice. As classes de instruções mostradas nas tabelas, estão descritas a seguir:

ULA logic operações lógicas: AND,OR,XOR;

ULA arit oper. aritméticas: ADD,SUB,MUL,DIV;

ULA desl deslocamentos e rotações;

ULA comp comparações de magnitude;

const MS carga da parte mais significativa de registrador;

move cópia entre registradores; MIPS e SPARC: de e para registradores de multiplicação;

load *load*s de todos os tamanhos;

store *store*s de todos os tamanhos;

jump saltos incondicionais, endereços absolutos;

branch desvios condicionais;

call chamadas de função;

ret retorno de função e saltos a registrador;

save SPARC: muda janela de registradores;

restore SPARC: repõe janela de registradores;

nop MIPS e SPARC: preenche *delay slot*;

total total geral;

ALU total todas de lógica, aritmética e comparações;

MEM total todos acessos à memória;

CTR total todos saltos e desvios;

FUN total todas as chamadas e retornos de função;

OTR total todas as demais instruções.

As instruções estão divididas um tanto arbitrariamente em classes de operações. As escolhas não são triviais: (i) instruções listadas como *ALUcomp* poderiam ser contadas no grupo de desvios (*branch*) porque comparações de magnitude são usadas para a tomada de decisões; (ii) instruções listadas como *const MS* carregam uma constante na parte mais significativa de um registrador e geralmente são usadas na carga de endereços absolutos

em registradores base e poderiam ser contadas junto às de endereçamento; (iii) nem todas as instruções contadas como *ret* são de fato "retorno de função" porque esta operação é efetuada com instruções de salto cujo operando está num registrador, mas estas também são usadas em tabelas de saltos para implementar o comando `switch`.

4.2 Perfil de execução das instruções

Antes de analisar os resultados, era de se esperar que a contagem do PowerPC fosse menor que a dos outros dois por conta quantidade de operações executadas por instrução [36], e que o SPARC obtivesse o segundo melhor resultado devido a janela de registradores. Ao contrário das expectativas, o MIPS é o que executa o menor número de instruções dentre os três. Considerando os totais, o PowerPC executa 14% mais instruções que o MIPS, e 4% mais que o SPARC e as referências à memória parecem ser a principal causa disso, como pode ser visto na Tabela 4.1. O SPARC executa 12% mais instruções que o MIPS, com 26% a mais de instruções de ULA.

As aplicações do MediaBench e do MiBench são maiores e processam conjuntos maiores de dados que as do CommBench. Cada suíte é discutida separadamente e, em seguida, é apresentada uma análise comparativa entre os três. As contagens totais de instruções para o PowerPC são usados como referência para comparação com os outros dois processadores.

Os histogramas na Figura 4.1 comparam os totais de instruções para os três processadores, bem como as contagens para as quatro classes mais importantes: *ULA* operações de lógica, aritmética e comparações; *MEM* acessos à memória; *CTR* controle de fluxo; *FUN* chamadas e retorno de funções; e o restante das instruções (*OTR*).

MediaBench O primeiro gráfico na Figura 4.1 mostra os resultados para os 7 programas do MediaBench. O MIPS executa 8.6% a menos que o PowerPC, 2.2% mais instruções de ULA, 44% menos instruções de referências à memória, e aproximadamente 10% menos de instruções de controle de fluxo e chamada/retorno de procedimento. O MIPS executa 18% mais instruções agrupadas em outros (OTR), que incluem 1.3×10^9 NOPs, enquanto que o PowerPC executa três vezes mais MOVEs e instruções que carregam a parte mais

	MIPS		PowerPC		SPARC	
classe	$\times 10^6$	%	$\times 10^6$	%	$\times 10^6$	%
ULA logic	12.072	13,53	14.725	14,17	21.606	21,65
ULA arit	20.079	22,50	11.768	11,32	22.685	22,74
ULA desl	11.227	12,58	14.452	13,91	8.133	8,15
ULA comp	5.366	6,01	11.344	10,92	8.827	8,85
const MS	1.633	1,83	3.894	3,75	3.143	3,15
move	739	0,83	3.246	3,12	1.272	1,27
load	11.165	12,51	14.349	13,81	9.715	9,74
store	8.322	9,33	13.779	13,26	6.108	6,12
jump	1.014	1,14	1.250	1,20	1.271	1,27
branch	9.624	10,79	12.411	11,94	11.016	11,04
call	1.048	1,18	1.454	1,40	1.680	1,68
ret	1.057	1,19	1.255	1,21	1.708	1,71
save	–	–	–	–	1.147	1,15
restore	–	–	–	–	1.147	1,15
nop	5.882	6,59	–	–	324	0,32
total	89.229	100,00	103.927	100,00	99.778	100,00
ULA total	48.744	54,63	52.289	50,31	61.250	61,39
MEM total	19.488	21,84	28.128	27,07	15.823	15,86
CTR total	10.638	11,92	13.661	13,14	12.287	12,31
FUN total	2.106	2,36	2.709	2,61	5.681	5,69
OTR total	8.253	9,25	7.140	6,87	4.738	4,75
			MediaBench			
total	18.219	100,00	19.787	100,00	17.488	100,00
ULA total	9.724	53,37	10.212	51,61	10.207	58,37
MEM total	4.151	22,78	5.984	30,24	3.263	18,66
CTR total	2.131	11,70	2.357	11,91	2.231	12,76
FUN total	497	2,73	543	2,75	996	5,70
OTH total	1.716	9,42	690	3,49	791	4,52
			MiBench			
total	64.355	100,00	76.872	100,00	73.819	100,00
ALU total	35.692	55,46	39.412	51,27	46.353	62,79
MEM total	13.794	21,43	20.054	26,09	10.908	14,78
CTR total	7.462	11,59	10.002	13,01	8.292	11,23
FUN total	1.472	2,29	2.017	2,62	4.464	6,05
OTH total	5.936	9,22	5.388	7,01	3.803	5,15
			CommBench			
total	6.655	100,00	7.263	100,00	8.467	100,00
ALU total	3.328	50,01	3.381	46,55	4.687	55,36
MEM total	1.544	23,19	2.089	28,77	1.651	19,50
CTR total	1.045	15,70	1.301	17,91	1.763	20,82
FUN total	137	2,06	150	2,06	220	2,60
OTH total	601	9,03	342	4,71	145	1,71

Tabela 4.1: Perfil de uso de instruções, todos os programas

significativa de um registrador (*const MS*). Aparentemente, o modo de endereçamento mais flexível do PowerPC nao está sendo bem utilizado nesses programas.

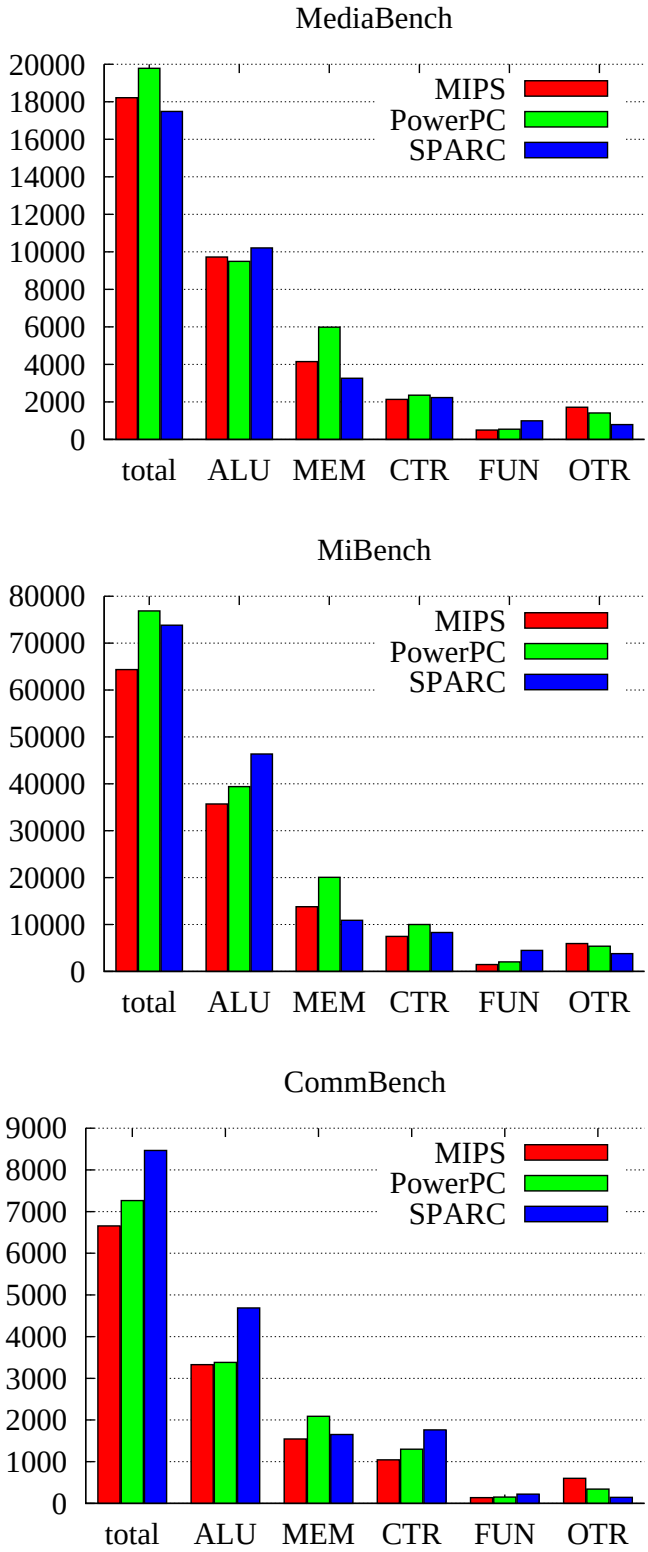

Figura 4.1: Número de instruções, total e por classe (instr.$\times 10^6$)

A Tabela 4.2 mostra que o SPARC executa 12% menos de instruções que o PowerPC, e as instruções de referência à memória são as que causam a maior diferença. O SPARC

executa aproximadamente metade de referências à memória (55%) que o PowerPC e essa vantagem advém das janelas de registradores. Para os três processadores, o número de chamadas e retornos de procedimento são próximos, em torno de 2.7% a 5.7% do total das instruções. O GCC parece ser capaz de preencher 18 vezes mais os *delay slots* do SPARC com instruções úteis do que para o MIPS (0.07 / 1.26), como pode ser visto na contagem total do NOPs na Tabela 4.2. Isso se deve, em grande medida ao intertravamento no uso de LOADs e a execução condicional das instruções no *branch delay slot* no SPARC.

Para estes programas de teste, o SPARC foi o que apresentou menor contagem no total de instruções executadas e o menor número de instruções de referências à memória. O MIPS foi que apresentou o menor número de instruções de controle e o PowerPC tem os piores resultados em relação ao total de instruções executadas e instruções de referência à memória.

	MIPS		**PowerPC**		**SPARC**	
classe	$\times 10^6$	%	$\times 10^6$	%	$\times 10^6$	%
ULA logic	2.482	13,62	2.586	13,07	4.376	25,02
ULA arit	4.040	22,18	2.164	10,94	2.851	16,30
ULA desl	2.145	11,78	2.706	13,67	1.412	8,07
ULA comp	1.057	5,80	2.038	10,30	1.569	8,97
const MS	351	1,93	719	3,63	659	3,77
move	108	0,59	691	3,49	62	0,35
load	2.435	13,37	3.068	15,50	2.000	11,44
store	1.715	9,42	2.916	14,74	1.263	7,22
jump	223	1,22	208	1,05	315	1,80
branch	1.909	10,48	2.148	10,86	1.916	10,96
call	248	1,36	294	1,48	260	1,49
ret	248	1,36	250	1,26	260	1,49
save	–	–	–	–	238	1,36
restore	–	–	–	–	238	1,36
nop	1.257	6,90	–	–	70	0,40
total	18.219	100,00	19.787	100,00	17.488	100,00
ULA total	9.724	53,37	10.212	51,61	10.207	58,37
MEM total	4.151	22,78	5.984	30,24	3.263	18,66
CTR total	2.131	11,70	2.357	11,91	2.231	12,76
FUN total	497	2,73	543	2,75	996	5,70
OTH total	1.716	9,42	690	3,49	791	4,52

Tabela 4.2: Perfil de uso de instruções, agregado de todos os programas do MediaBench

MiBench O diagrama no centro da Figura 4.1 mostra os resultados para o MiBench e estes estão detalhados na Tabela 4.3. O PowerPC executa 4% mais instruções que o SPARC, 15% menos instruções de ULA, 45% menos instruções de chamada/retorno de procedimento e 54% a mais de instruções de referência à memória. Novamente a vantagem é devida a janela de registradores do SPARC. O MIPS executa 16% menos instruções que o PowerPC, 31% menos de referências à memória e 25% menos instruções de controle de fluxo, 10% mais de operações de ULA e 10% a mais de outras instruções.

	MIPS		**PowerPC**		**SPARC**	
classe	$\times 10^6$	%	$\times 10^6$	%	$\times 10^6$	%
ALU logic	9.268	14,40	11.443	14,89	16.214	21,97
ALU arit	13.773	21,40	8.150	10,60	17.802	24,12
ALU desl	8.687	13,50	11.292	14,69	6.298	8,53
ALU comp	3.964	6,16	8.527	11,09	6.038	8,18
const MS	1.253	1,95	2.959	3,85	2.385	3,23
move	592	0,92	2.429	3,16	1.201	1,63
load	7.719	11,99	9.849	12,81	6.441	8,73
store	6.074	9,44	10.205	13,28	4.467	6,05
jump	724	1,13	886	1,15	821	1,11
branch	6.738	10,47	9.116	11,86	7.471	10,12
call	733	1,14	1.081	1,41	1.345	1,82
ret	739	1,15	936	1,22	1.368	1,85
save	–	–	–	–	875	1,19
restore	–	–	–	–	875	1,19
nop	4.092	6,36	–	–	217	0,29
total	64.355	100,00	76.872	100,00	73.819	100,00
ALU total	35.692	55,46	39.412	51,27	46.353	62,79
MEM total	13.794	21,43	20.054	26,09	10.908	14,78
CTR total	7.462	11,59	10.002	13,01	8.292	11,23
FUN total	1.472	2,29	2.017	2,62	4.464	6,05
OTH total	5.936	9,22	5.388	7,01	3.803	5,15

Tabela 4.3: Perfil de uso de instruções, agregado de todos os programas do MiBench

O SPARC executa 15% mais instruções que o MIPS, 30% mais instruções de ULA, 33% mais instruções de chamada/retorno de procedimentos e 21% menos instruções de referência à memória. Aqui, o GCC parece ser capaz de preencher melhor os *delay slots* do SPARC com instruções úteis do que para o MIPS, que executa aproximadamente 19% mais NOPs.

Neste conjunto de programas, o MIPS foi o que apresenta os melhores resultados, em relação ao total de instruções executadas e instruções de chamada/retorno de pro-

cedimento. O SPARC tem resultados similares aos obtidos com o MediaBench, com o menor número de instruções de referência à memória e o maior número de instruções de ULA. O PowerPC novamente executa mais instruções que os outros dois processadores, mas apresenta o menor número de instruções de ULA, tanto no MediaBench como no MiBench.

4.2.1 CommBench

A Tabela 4.4 mostra o número de linhas de código fonte dos programas depois de removidos os comentários e linhas em branco. Nomes em minúsculas referem-se a módulos específicos do programa.

programa	linhas	programa	linhas
RTR	1950	ZIP	11558
FRAG	793	zip	736
DRR	797	unzip	591
CAST	1665	REED	1300
CASTenc	835	REEDenc	359
CASTdec	831	REEDdec	359
JPEGenc	1642	JPEGdec	1708

Tabela 4.4: Linhas de código fonte

O diagrama de baixo na Figura 4.1 sintetiza os resultados com o conjunto CommBench, que são detalhados na Tabela 4.5. O PowerPC executa 9% mais instruções que o MIPS e 14% menos que o SPARC. O SPARC tem contagens piores, exceto nos acessos à memória.

As Tabelas 4.6 e 4.7 mostram as contagens dos *delay slots* para os programas CommBench de MIPS e SPARC respectivamente. A quantidade de NOPs executadas pelo SPARC é muito menor porque somente os *branch delay slots* devem ser preenchidos com NOPs. Além disso, uma porção dos *delay slots* (2.2% - total de NOPs em *branch delay slots*/total de desvios) é preenchida com instruções que podem ser anuladas caso o desvio não seja tomado.

Se todas as operações de acesso à memória forem consideradas (LOADs, STOREs, SAVEs, RESTOREs, NOPs em *load delay slots*), o SPARC executa 19% menos instruções de memória que o PowerPC e 8% menos que o MIPS, considerando-se que 60% dos NOPs executados

	MIPS		PowerPC		SPARC	
classe	$\times 10^6$	%	$\times 10^6$	%	$\times 10^6$	%
ALU logic	323	4,85	695	9,56	1.015	11,99
ALU arit	2.266	34,04	1.453	20,01	2.031	23,99
ALU desl	395	5,93	454	6,26	423	4,99
ALU comp	345	5,18	778	10,72	1.219	14,40
const MS	29	0,43	216	2,98	99	1,17
move	39	0,59	126	1,74	10	0,11
load	1.011	15,19	1.432	19,72	1.273	15,04
store	532	8,00	657	9,05	378	4,46
jump	68	1,02	155	2,14	135	1,60
branch	977	14,68	1.146	15,77	1.628	19,23
call	67	1,00	80	1,10	75	0,89
ret	71	1,06	70	0,96	79	0,93
save	–	–	–	–	33	0,39
restore	–	–	–	–	33	0,39
nop	533	8,01	–	–	36	0,43
total	6.655	100,00	7.263	100,00	8.467	100,00
ALU total	3.328	50,01	3.381	46,55	4.687	55,36
MEM total	1.544	23,19	2.089	28,77	1.651	19,50
CTR total	1.045	15,70	1.301	17,91	1.763	20,82
FUN total	137	2,06	150	2,06	220	2,60
OTH total	601	9,03	342	4,71	145	1,71

Tabela 4.5: Perfil de uso de instruções, agregado de todos os programas do CommBench

pelo MIPS estão em *load delay slots*, como mostra a Tabela 4.6.

Para as instruções de controle de fluxo, se 53% dos NOPs executados pelo SPARC, e 32% dos NOPs executados pelo MIPS, estão em *branch delay slot*, então o SPARC executa 37% mais instruções de controle que o PowerPC e 46% mais que o MIPS.

Surpreendemente, os dois processadores com modos de endereçamento mais sofisticados (PowerPC e SPARC) executam mais instruções de referência à memória que o MIPS, o qual tem um modo simples de endereçamento. Se 60% dos NOPs são computados como preenchendo o *load delay slots* (Tabela 4.6), o PowerPC executa 12% mais operações de memória que o MIPS, contra 35% se desconsiderados os NOPs.

Supondo que as instruções que carregam a parte alta do registrador (*const MS*) são todas usadas para criar endereços (*const MS*, LOAD, STORE, NOPs em *load delay slots*), então o PowerPC executa 21% a mais de instruções que geram endereços que o MIPS e 27% a mais que o SPARC. O SPARC executa apenas 5% a menos destas instruções

	NOP	**Load Delay**		**Branch Delay**	
programa	×1000	×1000	%	×1000	%
CASTenc	2.509	1.048	41,79	1.320	52,61
CASTdec	2.722	1.110	40,78	1.434	52,67
DRR	262.861	145.262	55,26	85.806	32,64
FRAG	29.240	15.807	54,06	10.624	36,33
JPEGenc	10.911	8.046	73,74	2.855	26,17
JPEGdec	17.957	14.592	81,26	1.940	10,81
REEDenc	66.016	65.561	99,31	388	0,59
REEDdec	87.283	43.669	50,03	43.539	49,88
RTR	12.613	8.074	64,02	2.936	23,28
ZIPenc	40.316	17.475	43,35	21.422	53,14
ZIPdec	884	723	81,84	124	14,11
total	533.317	321.373	60,26	172.393	32,32

Tabela 4.6: Contagens de *Delay Slots* no MIPS de todos os programas

	NOP	**Branch Delay**	
programa	×1000	×1000	%
CASTenc	791	397	50,29
CASTdec	851	433	50,94
DRR	22.060	12.034	54,55
FRAG	4.328	2.218	51,25
JPEGenc	269	12	4,79
JPEGdec	38	25	64,80
REEDenc	126	75	59,27
REEDdec	124	73	59,24
RTR	5.816	3.159	54,32
ZIPenc	1.715	877	51,18
ZIPdec	134	66	49,29
total	36.258	19.375	53,44

Tabela 4.7: Contagens de *Delay Slots* no SPARC de todos os programas

que o MIPS. O código gerado para o MIPS parece ser eficiente na geração dos endereços efetivos e também na manipulação de dados estáticos globais, pois o compilador os agrupa na *global area*, de onde são referenciados com base no registrador r_{gp}, que é o *global pointer*.

Supondo também que todas as comparações são usadas em desvios (*ULA comp*, *branch*, NOPs em *branch delay slots*), então o PowerPC executa 29% a mais de instruções que o MIPS, 33% a menos que o SPARC e o MIPS executa 52% menos destas instruções que o SPARC. Apesar de o MIPS ter mais NOPs em *branch delay slot*, o SPARC executa 66% a mais de desvios e 71% mais instruções de comparação quando comparado com o MIPS.

O compilador é capaz de preencher de uma forma mais eficiente a grande maioria dos *delay slots* do SPARC com instruções úteis. A relação entre os NOPs é 14.7× MIPS/SPARC, ou para o MIPS 8% de todas as instruções são NOPs enquanto que para o SPARC, são apenas 0.44% do total.

A quantidade de NOPs para todos os 27 programas é 18.2× MIPS/SPARC, 6.6% do total de instrução são NOPs no MIPS, e 0.32% do total no SPARC. É importante lembrar que o SPARC define somente *branch delay slot*, enquanto que o MIPS tem *branch* e *load delay slots*.

Na Figura 4.1 e na Tabela 4.1 é possível visualizar pequenas diferenças de comportamento dos processadores para cada conjunto de programas. O SPARC tem o menor número de instruções executadas no MediaBench e o MIPS tem os melhores números no MiBench e no CommBench. Em relação as instruções de ULA, o SPARC é o que mais executa instruções desse tipo. O PowerPC executa mais instruções de referências à memória e o MIPS mais instruções da classe nomeada OTR, que inclui os NOPS.

O SPARC é o que menos executa instruções de referência à memória; PowerPC o que executa menos instruções de ULA, e o MIPS que executa menos instruções de controle de fluxo.

4.2.2 Análise Estática

Como dito anteriomente a utilização da memória é um fator crítico para o desenvolvimento de sistemas embarcados. A partir disso, uma análise estática é necessária para ver quanto de memória é exigido por cada programa. A Tabela 4.8 mostra o tamanho em bytes que cada programa, do conjunto de testes CommBench, utiliza de memória. Para coletar os tamanhos dos binários foi utilizado a ferramenta *objdump* do compilador GCC.

O SPARC apresentou na soma de todos os programas, a menor utilização de memória (652KB) e o MIPS obteve a maior utilização, um pouco mais de 702KB. O PowerPC obteve o segundo melhor resultado, no total 700KB. No geral, é possível constatar que as três arquiteturas apresentaram tamanhos de códigos similares, em torno de 7% entre o maior e o menor.

	MIPS	**PowerPC**	**SPARC**
programa	bytes	bytes	bytes
CASTenc	22.332	19.636	19.600
CASTdec	22.352	19.660	19.616
DRR	40.844	41.140	37.712
FRAG	40.944	41.308	38.064
JPEGenc	138.444	137.192	127.104
JPEGdec	150.680	145.088	134.832
REEDenc	15.064	15.028	14.096
REEDdec	15.068	15.032	14.112
RTR	63.064	64.616	60.352
ZIPenc	105.348	107.960	101.248
ZIPdec	105.348	107.960	101.248
total	719.488	714.620	667.984

Tabela 4.8: Contagem Estática

4.3 Efeitos de Otimização

Como o compilador tem um papel importante no desempenho, nesta seção são avaliadas as diferenças no número de instruções executadas com código sem otimizar (`-O0`) e com três níveis de otimização do GCC: `-O1`, `-O2` e `-O3`. A Tabela 4.9 mostra a relação entre o número de instruções executadas no código com `-O3` ($= 1,0$) e os outros níveis. Um valor abaixo de $1,00$ significa que o número com `-O3` é *menor* que aquele gerado com o nível indicado na coluna. Esta medida foi efetuada somente com as duas versões do JPEG. A Tabela A.7, no Apêndice, contém as medidas de todas as classes de instruções.

O código otimizado executa menos que a metade das instruções do código não-otimizado, sendo o número de instruções dinâmicas do código sem otimizar de $2,2$ a $2,7$ vezes maior que aquele com `-O3`. As instruções de acesso à memória são aquelas em que ocorre o maior ganho, com eliminação de cerca de 3/4 das referências, especialmente de LOADs. Isso é significativo porque referências à memória são custosas e qualquer ganho tem reflexo importante no desempenho, não só pela redução no número de instruções executadas mas também pelo melhor aproveitamento dos dados que são trazidos para a cache.

As diferenças entre as versões otimizadas com `-O1`, `-O2`, `-O3` são pequenas, da ordem de 5%. Algumas excessões significativas merecem menção — veja a Tabela A.7 para

	JPEGenc			JPEGdec		
	`-O0`	`-O1`	`-O2`	`-O0`	`-O1`	`-O2`
	MIPS					
total	0,37	0,95	1,01	0,40	0,96	1,00
ULA total	0,75	1,02	1,00	0,92	1,07	1,00
MEM total	0,27	1,04	1,01	0,29	1,06	1,00
CTR total	0,71	1,00	1,00	0,80	1,19	1,00
FUN total	1,00	1,00	1,00	1,00	1,00	1,00
	PowerPC					
total	0,42	1,04	1,00	0,44	1,02	1,00
ULA total	0,56	1,02	1,00	0,70	1,08	1,01
MEM total	0,27	1,04	1,01	0,28	1,00	1,00
CTR total	0,71	1,00	1,00	0,69	0,91	1,00
FUN total	1,00	1,00	1,00	1,00	1,00	1,00
	SPARC					
total	0,45	1,03	1,00	0,48	1,05	1,00
ULA total	0,73	1,06	1,00	0,78	1,07	1,01
MEM total	0,24	1,00	1,00	0,29	1,04	1,00
CTR total	0,74	1,02	1,00	0,70	1,04	1,00
FUN total	1,00	1,00	1,00	0,99	0,99	1,00

Tabela 4.9: Otimização com relação a `-O3`.

detalhes. No código do MIPS a eliminação de NOPs de `-O2` para `-O3` remove de 60 a 70% deles. Para o SPARC, o número de JUMPs *aumenta* nas versões mais otimizadas, de $1,27$ e 2.5 vezes para JPEGenc e JPEDdec, respectivamente. Para o MIPS, a versão com `-O3` do JPEGdec executa $7,2$ vezes mais JUMPs do que a versão com `-O2`. Note que estes são cerca de 2% das instruções executadas e portanto seu efeito não é catastrófico. No PowerPC o número de MOVEs é maior na versão otimizada, bem como aumentam o número de operações de lógica e de carga de constantes na parte mais significativa de registradores, possivelmente por conta da computação mais trabalhosa de endereços e da troca de operações complexas por simples (*strenght reduction*).

Não se pode esquecer que estes valores referem-se ao *número* de instruções e não à sua 'qualidade' e nem a sua ordem de execução. Por 'qualidade' entenda-se, por exemplo, o número de ciclos para executar uma instrução custosa (multiplicações e divisões) quando um deslocamento é o suficiente. Outro caso de melhor 'qualidade' são as referências a dados globais estáticos no MIPS através do *global pointer*. Quanto a ordem de execução, o compilador pode tentar preencher um *delay slot* com uma instrução útil, ou afastar um

LOAD do uso de seu valor. Em processadores super-escalares, a ordem de emissão das instruções pode aumentar as chances de processamento em paralelo e, portanto, melhorar o desempenho. Estes ganhos só podem ser avaliados com simulação detalhada ou pela execução direta em hardware.

4.4 Distribuição do tempo

Conhecer a distribuição no tempo de referências à memória pode ser muito útil no projeto de hierarquias de memória para sistemas embarcados. Os histogramas na Figura 4.2 mostram a distribuição de distâncias entre pares de instruções LOADs, pares de STOREs, e pares de desvios. A 'distância' é o número de instruções entre os dois elementos do par, sendo que "distância zero" significa duas instruções consecutivas. Distâncias maiores que 15 instruções são todas agrupadas à direita, na barra marcada '+'.

Note que neste contexto *tempo* é uma noção imprecisa porque as simulações não medem tempo. Contudo, estas medidas dão uma indicação bastante boa da taxa de referências à memória e da freqüência com que o previsor de desvios deverá gerar uma previsão. Nos gráficos na Figura 4.2, para cada uma das classes, a contribuição de cada programa é ponderada pelo seu número de instruções com relação ao total de instruções daquela classe.

Para o MIPS, 80% de todos os LOADs ocorrem em distâncias menores que 5 instruções, enquanto que a grande maioria dos STOREs são adjacentes (62%) mas com 20% deles afastados de mais de 14 instruções. A distribuição de LOADs e STOREs do PowerPC é muito similar à do MIPS, com os picos um pouco menores. No SPARC os LOADs tem o mesmo comportamento que nos outros, sendo 85% deles com separação menor do que 5 instruções. Quanto aos STOREs, menos de 30% são agrupados, com 62% afastados de mais de 14 instruções, evidenciando os efeitos positivos da janela de registradores.

O comportamento do MIPS e do PowerPC é problemático para o projeto da interface de memória porque as referências são concentradas em curtos intervalos, o que implica em interfaces projetadas para atender à demanda de pico. Com tamanha concentração de STOREs, uma fila de escrita torna-se praticamente obrigatória para acomodar a diferença

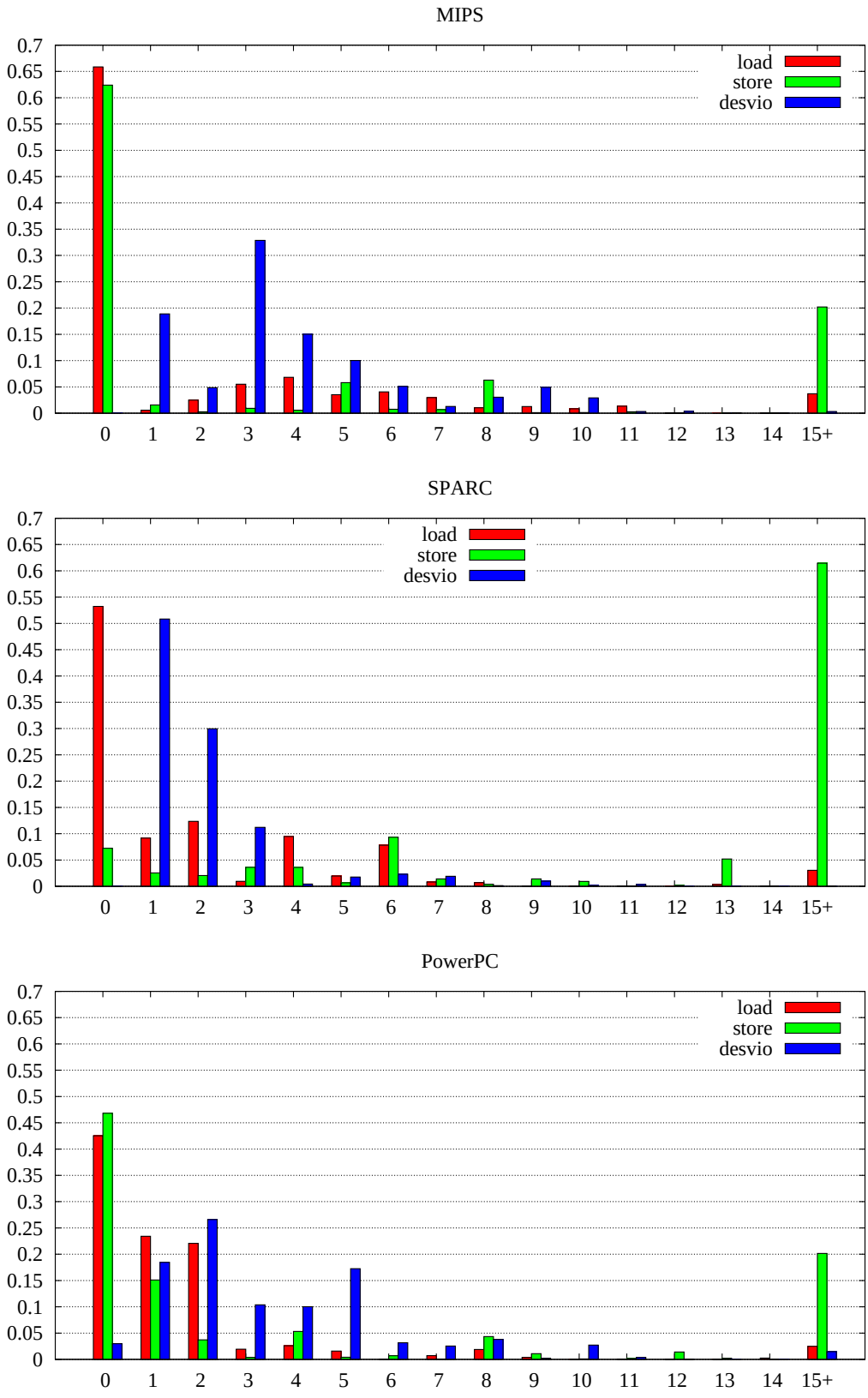

Figura 4.2: Distribuição de distâncias, todos os programas do CommBench.

de velocidade entre o processador e a memória, por melhor que seja o projeto da cache de dados.

Quanto aos desvios condicionais, MIPS e PowerPC tem uma distribuição similar, com a maioria dos desvios ($\approx 80\%$) separados de até 6 instruções. O PowerPC tem $2,5\%$ de seus desvios adjacentes, graças a sua unidade de desvios que é capaz de resolver uma condição no mesmo ciclo em que o desvio é despachado para execução. O código do SPARC tem 90% dos desvios separados por até 3 instruções. A proporção de desvios no total das instruções executadas é de $15,7\%$, $18,7\%$, e $20,8\%$ para MIPS, PowerPC e SPARC, respectivamente.

As distribuições das distâncias para o programa JPEGdec são mostradas na Figura 4.3, e neste programa os comportamentos dos três processadores são similares. Nos três casos os LOADs estão agrupados em seqüências de até 5 instruções, e cerca de metade dos STOREs estão separados por até 8 instruções. A janela de registradores do SPARC não causa diferença significativa com este programa. Os laços mais importantes são traduzidos para seqüencias de instruções parecidas para os três processadores porque as distribuições dos desvios são muito similares.

Um projeto de cache de dados para uma aplicação na qual este programa seja executado freqüentemente deve ser capaz de tolerar vazão elevada entre processador e cache, e entre cache e memória, porque codificação e decodificação JPEG apresentam taxas de faltas relativamente elevadas no primeiro nível da hierarquia de memória [12]. Por outro lado, desempenho adequado pode ser obtido de um previsor de desvios relativamente simples, porque os desvios são relativamente distantes entre si, o que facilita a recuperação de desvios previstos erradamente.

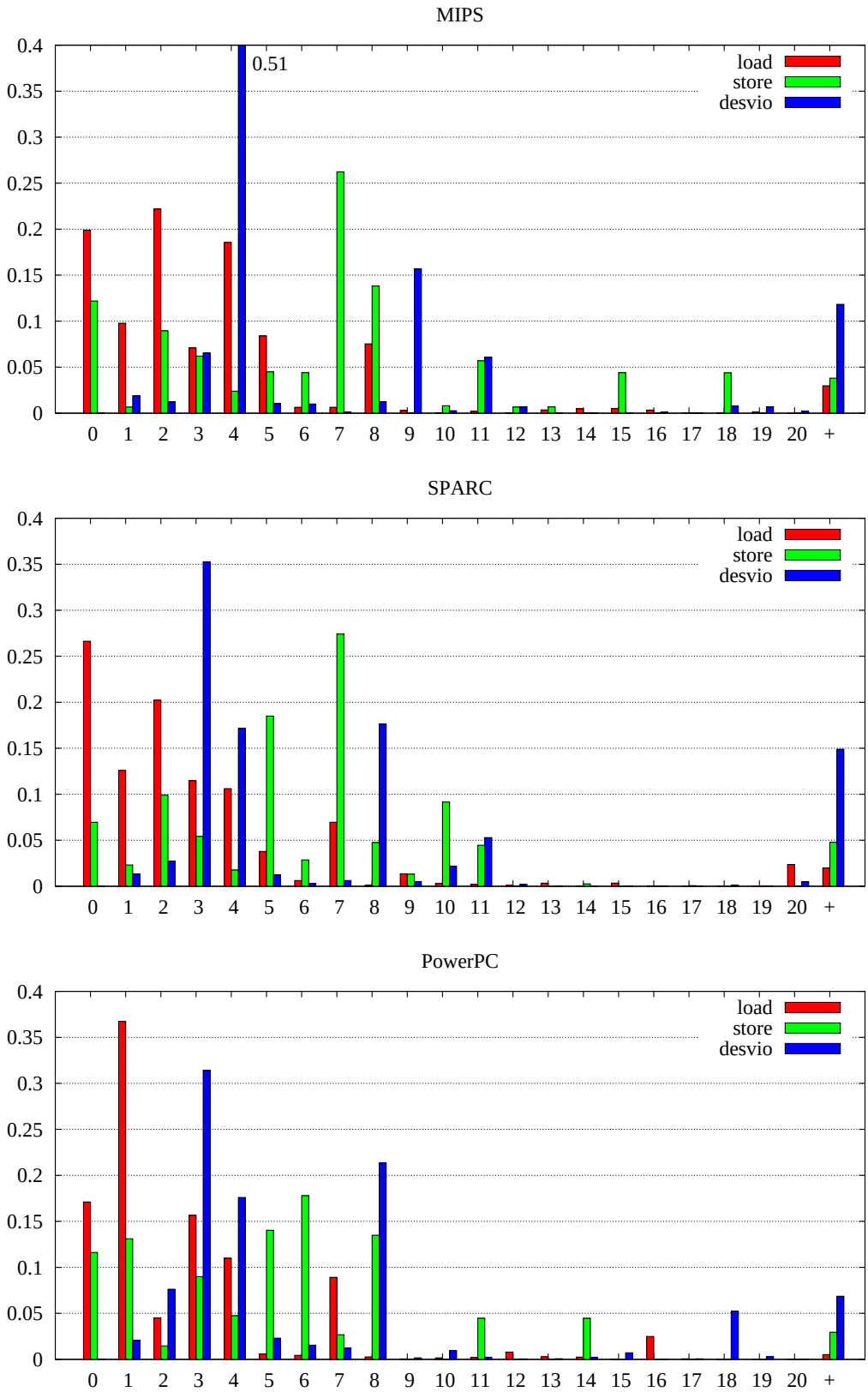

Figura 4.3: Distribuição de distâncias entre loads, stores e desvios, JPEGdec

CAPÍTULO 5

CONCLUSÃO

Este trabalho contém uma análise detalhada que compara, no nível do conjunto de instruções, três microprocessadores de 32 bits (MIPS, PowerPC, SPARC) que são populares em aplicações embarcadas. Os três conjuntos de instruções foram comparados pela simulação funcional da execução de 7 programas da suíte MediaBench, 13 programas da suíte MiBench e 7 programas da suíte CommBench. Os programas foram compilados com a versão 3.3.1 do GCC e simulados com modelos funcionais escritos em ArchC.

Considerando os totais das contagens para os 27 programas, o número de instruções executadas é de $89,23$, $103,92$, e $99,75 \cdot 10^9$ para MIPS, PowerPC e SPARC, respectivamente. O SPARC é o que executa menos instruções de referências à memória,o PowerPC que executa menos instruções de ULA e o MIPS que executa menos instruções de controle de fluxo. O SPARC é o que mais executa instruções de ULA, o PowerPC executa mais instruções de referências à memória e o MIPS mais instruções da classe nomeada OTR, que inclui os NOPS, que são uma fração significativa no total das instruções executadas.

Também foi apresentada uma análise de tamanhos dos executáveis dos programas do conjunto de testes CommBench e foi possível constatar que os três processadores tem tamanhos de código similares, a diferença é de 7%. Os programas para o SPARC necessitam no total 652KB de memória, enquanto o Power e MIPS precisam de 700KB e 702KB, respectivamente.

Como o compilador tem um papel importante no desempenho, foram avaliados os efeitos dos três níveis de otimização no código gerado pelo GCC para as duas versões do JPEG e as variações *nas contagens* são pequenas. As instruções de acesso à memória são aquelas em que ocorre o maior ganho com a otimização do compilador, com eliminação de cerca de 3/4 das referências, especialmente de LOADs. Isso é significativo porque referências à memória são custosas e qualquer ganho tem reflexo importante no desempe-

nho, não só pela redução no número de instruções executadas mas também pelo melhor aproveitamento dos dados que são trazidos para a cache.

Também foi avaliada a aglomeração de referências à memória e para os 7 programas da suíte CommBench constatamos que cerca de 80% dos LOADs e 60% dos STOREs encontram-se separados por menos de 6 instruções. Quanto a instruções de controle de fluxo, as medições indicam que para MIPS e PowerPC cerca de 80% dos desvios condicionais estão afastados de até 6 instruções, enquanto que para SPARC 90% dos desvios são separados de até 3 instruções. Estes números demonstram a necessidade de projetos sofisticados para a interface entre processador e memória e, no caso do SPARC, de um previsor de desvios sofisticado.

Os resultados apresentados neste trabalho são dependentes do compilador utilizado, provalvemente uma mudança de compilador para uma outra versão ou para compiladores nativos das arquiteturas os resultados apresentariam diferenças significativas. Assim, repetir os testes realizados com outros compiladores torna-se bem interessante e uma perspectiva para um trabalho futuro.

Dentre outras perspectivas de trabalhos futuros estão uma análise mais detalhada dos outros dois *benchmarks*, MediaBench e MiBench, pois neste trabalho o principal interesse foi a metodologia de comparação, por isso a escolha de um conjunto menor de programas. Também comparar a eficência do código de funções medindo o número de registradores acessados da pilha a cada chamada de função.

Outra trabalho futuro interessante seria estender a comparação apresentada aqui incluindo o processador ARM [10], que também é uma arquitetura de 32 bits e foi desenvolvido para obter o melhor desempenho possível com a limitação de ser simples, ocupar pouca área e ter baixo consumo de energia. É uma arquitetura muito usada na indústria de sistemas embarcados.

APÊNDICE A

MIPS	RTR		FRAG		DRR		CASTenc		CASTdec	
classe	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%
ALU logic	5.126	4,54	12.369	3,43	94.762	3,31	18.572	13,81	18.640	13,60
ALU arit	33.326	29,49	118.489	32,89	908.003	31,70	43.568	32,41	44.547	32,50
ALU desl	2.811	2,49	6.659	1,85	73.593	2,57	18.162	13,51	18.170	13,26
ALU comp	4.755	4,21	16.627	4,61	132.747	4,63	1.818	1,35	1.906	1,39
mv HI,LO	699	0,62	100	0,03	34.163	1,19	256	0,19	256	0,19
const MS	2.668	2,36	1.594	0,44	9.876	0,34	768	0,57	772	0,56
load	19.764	17,49	59.027	16,38	466.726	16,29	25.349	18,85	25.780	18,81
store	11.128	9,85	34.764	9,65	276.081	9,64	11.845	8,81	12.065	8,80
jump	1.772	1,57	6.919	1,92	42.064	1,47	654	0,49	701	0,51
branch	14.643	12,96	60.263	16,73	453.821	15,84	8.839	6,57	9.247	6,75
call	1.415	1,25	6.723	1,87	53.945	1,88	1.051	0,78	1.127	0,82
ret	2.275	2,01	7.523	2,09	55.946	1,95	1.051	0,78	1.127	0,82
nop	12.613	11,16	29.240	8,12	262.861	9,18	2.510	1,87	2.723	1,99
total	112.996		360.296		2.864.587		134.445		137.061	
ALU total	46.019	40,73	154.144	42,78	1.209.104	42,21	82.121	61,08	83.262	60,75
MEM total	30.892	27,34	93.791	26,03	742.807	25,93	37.194	27,67	37.845	27,61
CTR total	16.414	14,53	67.182	18,65	495.885	17,31	9.493	7,06	9.949	7,26
FUN total	3.690	3,27	14.245	3,95	109.891	3,84	2.103	1,56	2.254	1,64

Tabela A.1: Perfil de uso de instruções do MIPS

MIPS	ZIPenc		ZIPdec		REEDenc		REEDdec		JPEGenc		JPEGdec		TOTAL	
classe	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%
ALU logic	16.181	5,96	5.114	12,77	49.307	6,29	98.437	7,38	1.946	0,58	2.615	0,92	323.069	4,85
ALU arit	77.669	28,62	11.442	28,57	316.174	40,35	466.762	34,98	136.374	40,99	109.264	38,46	2.265.618	34,04
ALU desl	15.595	5,75	4.206	10,50	48.503	6,19	134.601	10,09	45.263	13,60	27.112	9,54	394.676	5,93
ALU comp	16.945	6,24	2.607	6,51	47.751	6,09	102.442	7,68	10.791	3,24	6.680	2,35	345.070	5,18
mv HI,LO	15	0,01	0	0,00	9	0,00	1.656	0,12	756	0,23	1.056	0,37	38.965	0,59
const MS	4.535	1,67	1.738	4,34	2.060	0,26	4.278	0,32	31	0,01	426	0,15	28.747	0,43
load	41.677	15,36	7.098	17,73	102.697	13,11	148.590	11,14	56.877	17,09	57.599	20,27	1.011.186	15,19
store	15.192	5,60	1.386	3,46	34.773	4,44	69.560	5,21	35.627	10,71	29.945	10,54	532.367	8,00
jump	1.973	0,73	182	0,45	99	0,01	4.696	0,35	2.971	0,89	5.810	2,04	67.841	1,02
branch	38.324	14,12	5.245	13,10	115.914	14,79	215.637	16,16	29.899	8,99	25.353	8,92	977.186	14,68
call	1.460	0,54	71	0,18	144	0,02	141	0,01	640	0,19	156	0,05	66.873	1,00
ret	1.460	0,54	71	0,18	144	0,02	141	0,01	640	0,19	156	0,05	70.535	1,06
nop	40.317	14,86	884	2,21	66.017	8,42	87.283	6,54	10.911	3,28	17.957	6,32	533.317	8,01
total	271.344		40.044		783.593		1.334.226		332.728		284.130		6.655.450	
ALU total	126.391	46,58	233.69	58,36	461.735	58,93	802.242	60,13	194.375	58,42	145.671	51,27	3.328.433	50,01
MEM total	56.870	20,96	8.484	21,19	137.470	17,54	218.150	16,35	92.505	27,80	87.544	30,81	1.543.552	23,19
CTR total	40.298	14,85	5.426	13,55	116.013	14,81	220.334	16,51	32.870	9,88	31.163	10,97	1.045.027	15,70
FUN total	2.919	1,08	141	0,35	289	0,04	283	0,02	1.281	0,38	313	0,11	137.408	2,06

Tabela A.2: Perfil de uso de instruções do MIPS (cont)

PowerPC	RTR		FRAG		DRR		CASTenc		CASTdec	
classe	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%
ALU logic	105.016	9,67	45.786	11,81	378.772	12,66	14.856	12,31	14.854	12,31
ALU arit	71.518	6,59	75.041	19,35	542.814	18,14	22.129	18,33	22.127	18,33
ALU desl	5.854	0,54	11.752	3,03	103.890	3,47	15.106	12,51	15.106	12,52
ALU comp	107.607	9,91	41.349	10,66	327.512	10,95	5.887	4,88	5.886	4,88
const MS	9.794	0,90	19.929	5,14	140.742	4,70	3.251	2,69	3.250	2,69
move	7.584	0,70	13.647	3,52	90.595	3,03	2.259	1,87	2.258	1,87
load	429.257	39,53	55.941	14,42	472.523	15,79	28.702	23,78	28.700	23,78
store	48.958	4,51	40.421	10,42	341.101	11,40	16.275	13,48	16.273	13,48
jump	97.550	8,98	8.331	2,15	37.112	1,24	782	0,65	782	0,65
branch	190.517	17,54	61.565	15,88	443.380	14,82	9.362	7,76	9.360	7,75
call	6.754	0,62	7.426	1,91	60.968	2,04	1.062	0,88	1.062	0,88
ret	5.521	0,51	6.621	1,71	52.936	1,77	1.047	0,87	1.047	0,87
total	1.085.931		387.811		2.992.345		120.718		120.704	
ALU total	299.790	27,61	193.859	49,99	1.493.731	49,92	61.229	50,72	61.223	50,72
MEM total	478.215	44,04	96.362	24,85	813.624	27,19	44.976	37,26	44.973	37,26
CTR total	288.067	26,53	69.896	18,02	480.492	16,06	10.144	8,40	10.141	8,40
FUN total	12.275	1,13	14.047	3,62	113.904	3,81	2.109	1,75	2.109	1,75

Tabela A.3: Perfil de uso de instruções do PowerPC

PowerPC	ZIPenc		ZIPdec		REEDenc		REEDdec		JPEGenc		JPEGdec		TOTAL	
classe	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%
ALU logic	9.469	4,48	3.668	10,49	34.328	5,12	64.213	5,58	13.702	5,01	9.845	4,60	694.510	9,56
ALU arit	32.581	15,41	7.251	20,73	198.644	29,66	342.066	29,71	79.457	29,03	59.659	27,89	1.453.288	20,01
ALU desl	16.025	7,58	4.160	11,90	94.854	14,16	150.167	13,04	25.044	9,15	12.337	5,77	454.298	6,26
ALU comp	27.016	12,78	3.823	10,93	81.767	12,21	142.366	12,36	19.040	6,96	16.157	7,55	778.409	10,72
const MS	14.111	6,67	2.249	6,43	3.406	0,51	10.504	0,91	4.184	1,53	4.685	2,19	216.106	2,98
move	1.705	0,81	142	0,41	829	0,12	1.044	0,09	5.629	2,06	536	0,25	126.228	1,74
load	49.004	23,17	6.233	17,82	103.163	15,40	149.044	12,94	55.654	20,34	53.900	25,20	1.432.122	19,72
store	19.087	9,03	1.688	4,83	35.926	5,36	70.680	6,14	37.147	13,57	29.459	13,77	657.014	9,05
jump	1.217	0,58	151	0,43	100	0,01	4.697	0,41	2.862	1,05	1.887	0,88	155.471	2,14
branch	38.327	18,12	5.463	15,62	116.514	17,39	216.319	18,79	29.659	10,84	25.104	11,74	1.145.568	15,77
call	1.460	0,69	72	0,21	182	0,03	178	0,02	649	0,24	181	0,08	79.996	1,10
ret	1.460	0,69	70	0,20	126	0,02	123	0,01	636	0,23	148	0,07	69.733	0,96
total	211.461		34.971		669.839		1.151.402		273.663		213.899		7.262.743	
ALU total	99.203	46,91	21.151	60,48	412.999	61,66	709.316	61,60	141.427	51,68	102.683	48,01	3.596.611	49,52
MEM total	68.091	32,20	7.921	22,65	139.089	20,76	219.724	19,08	92.801	33,91	83.360	38,97	2.089.136	28,77
CTR total	39.543	18,70	5.614	16,05	116.614	17,41	221.016	19,20	32.521	11,88	26.991	12,62	1.301.039	17,91
FUN total	2.919	1,38	143	0,41	308	0,05	301	0,03	1.285	0,47	329	0,15	149.729	2,06

Tabela A.4: Perfil de uso de instruções do PowerPC (cont)

SPARC	RTR		FRAG		DRR		CASTenc		CASTdec	
classe	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%
ALU logic	137.127	10,89	62.202	17,15	457.137	12,42	27.275	20,99	27.760	20,99
ALU arit	99.786	7,92	71.424	19,69	946.207	25,72	28.854	22,20	29.101	22,00
ALU desl	90.242	7,16	6.855	1,89	25.132	0,68	18.672	14,37	18.679	14,12
ALU comp	183.417	14,56	55.968	15,43	575.408	15,64	8.310	6,39	8.679	6,56
const MS	5.733	0,46	7.998	2,21	56.067	1,52	783	0,60	826	0,62
move	1.938	0,15	200	0,06	2	0,00	511	0,39	511	0,39
load	410.595	32,60	45.312	12,49	361.473	9,82	21.852	16,81	22.153	16,75
store	10.606	0,84	21.151	5,83	151.188	4,11	8.218	6,32	8.305	6,28
jump	73.434	5,83	6.921	1,91	41.009	1,11	911	0,70	958	0,72
branch	216.348	17,18	60.963	16,81	892.914	24,27	9.608	7,39	10.030	7,58
call	6.706	0,53	6.721	1,85	53.037	1,44	1.302	1,00	1.364	1,03
ret	8.552	0,68	7.425	2,05	54.053	1,47	1.308	1,01	1.384	1,05
save	4.679	0,37	2.612	0,72	21.903	0,60	786	0,60	837	0,63
restore	4.679	0,37	2.612	0,72	21.903	0,60	786	0,60	837	0,63
nop	5.817	0,46	4.329	1,19	22.060	0,60	791	0,61	852	0,64
total	1.259.658		362.694		3.679.494		129.968		132.276	
ALU total	510.571	40,53	196.449	54,16	2.003.885	54,46	83.111	63,95	84.218	63,67
MEM total	421.201	33,44	66.463	18,32	512.661	13,93	30.070	23,14	30.458	23,03
CTR total	289.783	23,00	67.884	18,72	933.923	25,38	10.519	8,09	10.988	8,31
FUN total	24.615	1,95	19.370	5,34	150.897	4,10	4.182	3,22	4.422	3,34

Tabela A.5: Perfil de uso de instruções do SPARC

SPARC	ZIPenc		ZIPdec		REEDenc		REEDdec		JPEGenc		JPEGdec		TOTAL	
classe	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%	×1000	%
ALU logic	39.870	16,34	5.866	14,90	84.994	12,06	126.415	9,85	25.122	7,24	20.975	7,39	1.014.744	11,99
ALU arit	51.372	21,05	7.674	19,49	166.705	23,66	401.192	31,25	128.706	37,09	99.886	35,21	2.030.907	23,99
ALU desl	7.291	2,99	3.920	9,96	80.980	11,49	105.186	8,19	41.516	11,96	24.258	8,55	422.732	4,99
ALU comp	29.080	11,92	4.402	11,18	115.011	16,32	194.053	15,12	26.158	7,54	18.435	6,50	1.218.921	14,40
const MS	13.548	5,55	2.204	5,60	2.122	0,30	8.461	0,66	815	0,23	388	0,14	98.944	1,17
move	30	0,01	0	0,00	18	0,00	3.312	0,26	1.017	0,29	2.111	0,74	9.652	0,11
load	43.615	17,87	7.793	19,79	102.792	14,59	148.808	11,59	51.627	14,88	57.403	20,23	1.273.423	15,04
store	12.114	4,96	1.628	4,13	34.856	4,95	69.818	5,44	30.385	8,76	29.354	10,35	377.624	4,46
jump	1.188	0,49	141	0,36	106	0,02	4.699	0,37	3.821	1,10	2.012	0,71	135.201	1,60
branch	38.335	15,71	5.467	13,89	116.505	16,53	217.957	16,98	33.540	9,66	26.152	9,22	1.627.819	19,23
call	1.475	0,60	70	0,18	135	0,02	1.779	0,14	1.327	0,38	1.160	0,41	75.076	0,89
ret	1.474	0,60	69	0,18	155	0,02	1.799	0,14	1.464	0,42	1.251	0,44	78.934	0,93
save	1.453	0,60	1	0,00	97	0,01	95	0,01	634	0,18	137	0,05	33.236	0,39
restore	1.453	0,60	1	0,00	97	0,01	95	0,01	634	0,18	137	0,05	33.236	0,39
nop	1.715	0,70	135	0,34	127	0,02	124	0,01	270	0,08	39	0,01	36.258	0,43
total	244.012		39.373		704.701		1.283.795		347.037		283.697		8.466.704	
ALU total	127.612	52,30	21.863	55,53	447.691	63,53	826.847	64,41	221.502	63,83	163.554	57,65	4.687.303	55,36
MEM total	55.729	22,84	9.421	23,93	137.648	19,53	218.626	17,03	82.012	23,63	86.757	30,58	1.651.046	19,50
CTR total	39.523	16,20	5.609	14,25	116.611	16,55	222.656	17,34	37.361	10,77	28.163	9,93	1.763.020	20,82
FUN total	5.855	2,40	142	0,36	484	0,07	3.769	0,29	4.059	1,17	2.684	0,95	220.481	2,60

Tabela A.6: Perfil de uso de instruções do SPARC (cont)

	MIPS						PowerPC						SPARC					
	JPEGenc			JPEGdec			JPEGenc			JPEGdec			JPEGenc			JPEGdec		
classe	-O0	-O1	-O2	-O0	-O1	-O2	-O0	-O1	-O2	-O0	-O1	-O2	-O0	-O1	-O2	-O0	-O1	-O2
ALU logic	1,00	1,00	1,00	1,00	1,00	1,00	0,46	2,37	0,99	1,15	2,25	1,06	0,40	1,98	1,00	0,53	1,77	1,04
ALU arit	0,71	1,05	1,00	0,92	1,10	1,00	0,60	0,98	1,00	0,70	0,99	1,00	0,83	1,00	1,00	0,88	1,00	1,00
ALU desl	0,80	0,97	1,02	0,89	1,02	0,99	0,52	1,00	1,00	0,44	0,99	1,00	0,74	1,00	1,00	0,80	1,04	1,00
ALU comp	1,13	0,97	1,00	0,95	0,96	1,00	0,64	0,82	1,00	0,65	0,91	1,00	0,85	0,99	1,00	0,73	1,00	1,00
const MS	1,00	1,00	1,00	1,00	1,00	1,00	0,44	1,01	1,00	3,99	5,61	1,00	0,08	0,38	1,00	0,16	0,94	1,00
move	0,02	0,97	1,00	0,20	1,77	1,00	3,98	3,98	1,00	1,57	1,56	1,00	1,00	1,00	1,00	1,00	1,00	1,00
load	0,23	1,03	1,01	0,26	1,07	1,00	0,22	1,01	1,01	0,24	1,00	1,00	0,21	1,00	1,00	0,26	1,05	1,00
store	0,36	1,06	1,01	0,40	1,04	1,00	0,37	1,08	1,01	0,39	1,01	1,00	0,31	1,00	1,00	0,39	1,01	1,00
jump	0,18	1,08	1,00	0,42	7,23	1,00	0,17	1,00	1,00	0,14	0,40	1,00	0,23	1,27	1,00	0,15	2,47	1,00
branch	0,99	1,00	1,00	1,00	1,00	1,00	0,99	1,00	1,00	1,00	1,00	1,00	0,99	1,00	1,00	1,00	1,00	1,00
call	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
ret	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00
save	—	—	—	—	—	—	—	—	—	—	—	—	1,00	1,00	1,00	0,96	0,96	1,00
restore	—	—	—	—	—	—	—	—	—	—	—	—	1,00	1,00	1,00	0,96	0,96	1,00
nop	0,05	0,30	1,01	0,09	0,37	1,01	—	—	—	—	—	—	0,01	1,01	1,00	0,00	0,89	1,00
total	0,37	0,95	1,01	0,40	0,96	1,00	0,42	1,04	1,00	0,44	1,02	1,00	0,45	1,03	1,00	0,48	1,05	1,00
ALU total	0,75	1,02	1,00	0,92	1,07	1,00	0,56	1,02	1,00	0,70	1,08	1,01	0,73	1,06	1,00	0,78	1,07	1,01
MEM total	0,27	1,04	1,01	0,29	1,06	1,00	0,27	1,04	1,01	0,28	1,00	1,00	0,24	1,00	1,00	0,29	1,04	1,00
CTR total	0,71	1,00	1,00	0,80	1,19	1,00	0,71	1,00	1,00	0,69	0,91	1,00	0,74	1,02	1,00	0,70	1,04	1,00
FUN total	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	1,00	0,99	0,99	1,00

Tabela A.7: Níveis de otimização com relação a -O3

APÊNDICE B

Neste Apêndice é descrito os passos necessários para obter os resultados apresentados no trabalho. O primeiro passo é compilar os programas dos conjuntos de testes que pretende-se analisar. A maioria dos programas dos *benchmarks* possuem um makefile onde tem algumas regras específicas de compilação assim como ligações com algumas bibliotecas. Mas em geral, para compilar os programas utilizamos as seguintes linhas de comando:

- `mips-elf-gcc -specs=archc -msoft-float -O3 executável código.c`
- `powerpc-elf-gcc -specs=archc -msoft-float -O3 executável código.c`
- `sparc-elf-gcc -specs=archc -msoft-float -O3 executável código.c`

Após os programas serem compilados, é necessário executá-los nos simuladores do processadores. Para executar um programa no simulador do MIPS por exemplo, utiliza-se a seguinte linha de comando:

```
./mips1.x --load=executável < entrada_dados > saida_programa 2> saida_archc.txt
```

A entrada_dados é caso o programa que esteja sendo executado precise de alguma entrada de dados, a saida_programa é caso o programa imprima uma saída e saida_archc.txt é para guardar em um arquivo as informações e estatísticas que o simulador gera.

O arquivo saida_archc.txt possui todas as informações necessárias para as análises realizadas neste trabalho. Para extrair estas informações foi utilizado um script em awk, o nome do script é NTClasses.awk. Cada processador tem um NTClasses.awk devido ao conjunto de instruções ser diferente. O comando para utilizar o script é:

```
awk -f NTClasses.awk saida_archc.txt > resultados.txt
```

A saída desse script possui o formato das tabelas apresentadas na seção 4.2. A saída tem a quantidade de instruções executadas por classe. O script NTClasses.awk para o conjunto de instruções do MIPS é mostrado a seguir:

```
BEGIN { cli=0; clog=0; csft=0; car=0; ccmp=0; cmv=0; cld=0; cst=0;
  cj=0; cb=0; cnop=0; csys=0; ccall=0; cret=0;  }

/^(and|or|xor|nor|andi|ori|xori) /{ clog += $3; }
/^(sll|srl|sllv|srlv|sra|srav) /{ csft += $3; }
/^(slt|sltu|slti|sltiu) /{ ccmp += $3 }
/^(add|addu|sub|subu|multu|mult|div|divu|addi|addiu) /{ car += $3; }
/^(lui) /{ cli += $3;}
/^(mfhi|mthi|mflo|mtlo) /{ cmv += $3; }
/^(lb|lh|lw|lbu|lhu|lwl|lwr) /{ cld += $3; }
/^(sb|sh|sw|swl|swr) /{ cst += $3; }
/^j /{ cj += $3; }
/^(jal|jalr|bltzal|bgezal) /{ ccall += $3; }
/^jr /{ cret += $3; }
/^(beq|bne|blez|bgtz|bltz|bgez) /{ cb += $3; }
/^(nop) /{ cnop += $3 }
/^(SYSCALLS|sys_call|instr_break) /{ csys += $3 }

END{
   total_alu=clog+csft+ccmp+car;
   total=total_alu+cli+cmv+cld+cst+cj+cb+cnop+csys+ccall+cret;

   printf "&&               \n" ;
   printf "&& %7.0f & %5.2f \n" , clog/1000000, clog/total*100 ;
   printf "&& %7.0f & %5.2f \n",   car/1000000,  car/total*100 ;
   printf "&& %7.0f & %5.2f \n",  csft/1000000, csft/total*100 ;
   printf "&& %7.0f & %5.2f \n",  ccmp/1000000, ccmp/total*100 ;
   printf "&& %7.0f & %5.2f \n",   cli/1000000,  cli/total*100 ;
   printf "&& %7.0f & %5.2f \n",   cmv/1000000,  cmv/total*100 ;
   printf "&& %7.0f & %5.2f \n",   cld/1000000,  cld/total*100 ;
   printf "&& %7.0f & %5.2f \n",   cst/1000000,  cst/total*100 ;
   printf "&& %7.0f & %5.2f \n",    cj/1000000,   cj/total*100 ;
   printf "&& %7.0f & %5.2f \n",    cb/1000000,   cb/total*100 ;
   printf "&& %7.0f & %5.2f \n", ccall/1000000, ccall/total*100 ;
   printf "&& %7.0f & %5.2f \n",  cret/1000000,  cret/total*100 ;
   printf "&& %7.0f & %5.2f \n",  cnop/1000000, cnop/total*100 ;
   printf "&& %7.0f &       \n", total/1000000 ;
   printf "&& %7.0f & %5.2f \n",total_alu/1000000, total_alu/total*100 ;
   printf "&& %7.0f & %5.2f \n", (cld+cst)/1000000, (cld+cst)/total*100 ;
   printf "&& %7.0f & %5.2f \n", (cj+cb)/1000000, (cj+cb)/total*100 ;
   printf "&& %7.0f & %5.2f \n", (ccall+cret)/1000000, (ccall+cret)/total*100 ;
   resto = total - (total_alu + cld+cst + cj+cb + ccall+cret);
   printf "&& %7.0f & %5.2f \n", resto/1000000, resto/total*100 ;
}
```

Para coletar as distâncias entre pares de instruções LOADs, STOREs e desvios do arquivo saida_archc.txt foi utilizados três scripts awk: HistDistDesvios.awk, HistDistLoad.awk e HistDistStore.awk. O comando para esses scripts é:

```
awk -f HistDistLoad.awk saida_archc.txt > resultados.txt
```

A saída desse script é a quantidade para cada distância entre pares de LOADs, de 0 até 21. A seguir, como exemplo o script HistDistLoad.awk:

```
BEGIN {
     cdist=0; }

/^(Loads) /{ cont[$3]=$5; tot+=$5 ; }

END {
   for (t in cont)
    { printf "%d %d\n",t, cont[t] ; }
}
```

REFERÊNCIAS BIBLIOGRÁFICAS

[1] The SPARC Architecture Manual, Version 8. SPARC International, Inc.

[2] A. I. Gervini, E. de F. Corrêa, L. Carro e F. R. Wagner. Avaliação de Desempenho, Área e Potência de Mecanismos de Comunicação em Sistemas Embarcados. *XXX Seminário Integrado de Software e Hardware, Campinas, Brasil*, 2003.

[3] Andréia Aparecida Barbiero. Ambiente de Suporte ao Projeto de Sistemas Embarcados. Dissertação de mestrado, Departamento de Informática, UFPR, julho de 2006. http://www.inf.ufpr.br/roberto/dissAndreia.pdf.

[4] The ArchC Arquitecture Description Language. `http://www.archc.org`, acessado em 25/02/2007.

[5] ATMEL AVR Microcontrollers. `http://www.atmel.com/`, acessado em 16/02/2007.

[6] C. Lee, M. Potkonjak and W. H. Mangione-Smith. MediaBench: a tool for evaluating and synthesizing multimedia and communications systems. *30th Annual IEEE/ACM Int Symp on Microarchitecture (MICRO'97)*, páginas 330–335, agosto de 1997.

[7] Charles Price. MIPS IV Instruction Set, Revision 3.2. Relatório técnico, MIPS Technologies, Inc., setembro de 1995.

[8] CommBench. `http://www.ecs.umass.edu/ece/wolf/nsl/software/cb/`, acessado em 03/03/2007.

[9] D. A. Patterson and C. H. Sequin. RISC I: A reduced instruction set VLSI computer. *Proceedings of the 8th International Symposium om Computer Architecture (ISCA'81)*, páginas 443–457, 1981.

[10] Dave Jaggar. ARM Architecture and Systems. *IEEE Micro*, 17(4):9–11, julho de 1997.

[11] G. Randin. The 801 Minicomputer. *Proc SIGARCH/SIGPLAN Symp on Architectural Support for Programming Languages and Operating Systems*, páginas 39–47, março de 1982.

[12] G. C. Heck e R. A. Hexsel. The performance of pollution control victim cache for embedded systems. *21st Symp on Integrated Circuits and Systems Design (SBCCI'08)*, páginas 6, setembro de 2008.

[13] J. L. Hennessy e D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann, 1st edition, 1990.

[14] J. Hennessy, N. Jouppi, S. Przybylski, C. Rowen, T. Gross, F. Baskett and J. Gill. MIPS: A Microprocessor Architecture. *International Symposium on Microarchitecture - Proceedings of the 15th annual workshop on Microprogramming - Palo Alto, California, United States*, páginas 17–22. IEEE Press, 1982.

[15] J. L. Hennessy and D. A. Patterson. *Arquitetura de Computadores: Uma abordagem Quantitativa; tradução de V. D. de Souza*, volume 1. Campus, 3rd edition, 2003.

[16] K. Diefendorff and E. Silha. The PowerPC User Instruction Set Architecture. *IEEE Micro*, 14(5):30–41, outubro de 1994.

[17] G. Kane e J. Heinrich. *MIPS RISC Architecture*. Prentice Hall, 2nd edition, 1991.

[18] L. Carro e F. R. Wagner. Sistemas Computacionais Embarcados. *JAI'03 - XXII Jornadas de Atualização em Informática*, agosto de 2003.

[19] M. Bartholomeu. *Simulação Compilada para Arquiteturas Descritas em ArchC*. Tese de doutorado, Instituto de Computação, Univ Estadual de Campinas, Setembro de 2005.

[20] M. Calzarossa, L. Massari and D. Tessera. Workload Characterization Issues and Methodologies. *Performance Evaluation - Origins and Directions*, 1769:459–481, 2000.

[21] T. Austin T. Mudge R. Brown M. Guthaus, J. Ringerberg. MiBench: A free, commercially representative embedded benchmark suite. *Workload Characterization, 2001. WWC-4.*, páginas 3–14, dezembro de 2001.

[22] M. K. Becker, M. S. Allen, C. R. Moore, J. S. Muhich and D. P. Tuttle. The Power PC 601 Microprocessor. *IEEE Micro*, 13(5):54–68, setembro de 1993.

[23] MediaBench. `http://euler.slu.edu/~fritts/mediabench/mb1/`, acessado em 03/03/2007.

[24] MiBench. `http://www.eecs.umich.edu/mibench/index.html`, acessado em 03/03/2007.

[25] MIPS Technologies. `http://www.mips.com`, acessado em 14/02/2007.

[26] MIPS. *MIPS32 Architecture For Programmers – The MIPS32 Instruction Set*, volume 2. MIPS Technologies, julho de 2005.

[27] Motorola DSP Family. `http://www.freescale.com/webapp/sps/site/homepage.jsp?nodeId=012795`, acessado em 14/02/2007.

[28] Richard P. Paul. *SPARC Architecture, Assembly Language Programming, and C.* Prentice Hall, 2nd edition, 2000.

[29] R. S. Piepho e W. S. Wu. A comparison of RISC architectures. *IEEE Micro*, 9(4):51–62, agosto de 1989.

[30] Power.org. *Power ISA User Instruction Set Architecture.* IBM Corp., outubro de 2007. Book 1 vers 2.05.

[31] R. F. Cmelik, S. I. Kong, D. R. Ditzel and E. J. Kelly. An Analysis of MIPS and SPARC instruction set utilization on the SPEC benchmarks. *4th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS'91)*, páginas 290–302. ACM Press, 1991.

[32] RABBIT Semiconductor. `http://www.rabbitsemiconductor.com/products/dc/index.shtml`, acessado em 16/02/2007.

[33] S. Mirapuri, M. Woodacre and N. Vasseghi. The MIPS R4000 Processor. *IEEE Micro*, 12(2):10–22, 1992.

[34] S. Rigo, G. Araujo, M. Bartholomeu and R. Azevedo. ArchC: A SystemC-based architecture description language. *16th Symposium on Computer Architecture and High Performance Computing (SBAC'04)*, outubro de 2004.

[35] S. Rigo, R. Azevedo and G. Araujo. The ArchC Architecture Description Language. Relatório técnico, IC - Unicamp, junho de 2003.

[36] J. Smith e S. Weiss. PowerPC 601 and Alpha 21064: A tale of two RISCs. *IEEE Computer*, 27(6):46–58, junho de 1993.

[37] SPARC International, Inc. `http://www.sparc.com`, acessado em 14/02/2007.

[38] T. Groetker, S. Liao, G. Martin, S. Swan. *System Design with SystemC*, volume 1. Webman, 2002.

[39] T. M. Conte and W. W. Hwu. Benchmark Characterization. *IEEE Computer Society Press*, 24(1):48–56, 1991.

[40] T. Wolf e M. A. Franklin. CommBench – a telecommunications benchmark for network processors. *Proc IEEE Intl Symp on Performance Analysis of Systems and Software (ISPASS)*, páginas 154–162, abril de 2000.

[41] Xilinx. *PowerPC Processor Reference Guide - Embedded Development Kit*. Xilinx Inc., setembro de 2003.

FABIANY LAMBOIA

ANÁLISE COMPARATIVA DE USO DOS CONJUNTOS DE INSTRUÇÕES DOS MICROPROCESSADORES DE 32 BITS MIPS, POWERPC E SPARC

Dissertação apresentada como requisito parcial à obtenção do grau de Mestre. Programa de Pós-Graduação em Informática, Setor de Ciências Exatas, Universidade Federal do Paraná.
Orientador: Prof. Dr. Roberto A Hexsel.

CURITIBA

2008