International Journal of Computer Architecture Education (IJCAE)

Ensinando Arquiteturas Vetoriais Utilizando um Simulador de Instruções MIPS

Fredy A. M. Alves[1], Danilo Almeida[1], Lucas Bragança[1],
André B. M. Gomes[1,2], Ricardo S. Ferreira[1,2], José Augusto M. Nacif[1,2]
[1]Instituto de Ciências Exatas e Tecnológicas, Campus UFV-Florestal, Brasil
[2]Departamento de Informática, UFV, Brasil
{fredy.alves, danilo.damiao, lucas.braganca, andre.maciel, ricardo, jnacif}@ufv.br

***Abstract*—Os processadores vetoriais juntamente com o processador MIPS constituem dois marcos importantes no desenvolvimento e no ensino de arquitetura de computadores. Nas últimas décadas, várias extensões do conjunto de instruções MIPS foram especificadas, incluindo uma extensão SIMD em 2014. Este artigo apresenta o primeiro simulador vetorial baseado no MIPS para ser utilizado em aulas de arquitetura de computadores de cursos de Ciência da Computação e Engenharias. A implementação proposta é baseada no MARS, um simulador MIPS32 de código aberto. O simulador proposto é denominado Vector MARS ou VMARS e apresenta uma interface robusta e intuitiva que auxilia no processo de simulação das instruções vetoriais MIPS oferecendo também suporte ao conjunto padrão de instruções do MIPS32. Juntamente com o simulador VMARS são apresentados exemplos para ilustrar o uso das instruções vetoriais presentes na extensão do MIPS.**

Keywords*—*Processadores Vetoriais; Simulador MIPS; Ensino de Arquitetura de Computadores

I. Introdução

Na década de 60, o projetista de computadores S. Cray inovou ao apresentar os computadores CDC 6600 e 7600 com múltiplas unidades funcionais, unidade de *load/store* desacoplada, *pipeline*, refrigeração e os princípios dos computadores RISC. Posteriormente, na década de 70, com a fundação da sua própria empresa, o primeiro computador vetorial foi projetado e comercializado, criando um marco na história da computação. Outro marco importante foi o projeto do processador MIPS nos anos 80, que herdou vários princípios dos computadores da linha CDC dando origem ao termo RISC (*Reduced Instruction Set Computer*). Recentemente, para o ensino destes conceitos, a literatura da área [1], [2] propôs o MIPS vetorial (VMIPS) e destacou também as extensões SIMD multimídia da Intel. Entretanto não existem simuladores para auxiliar no ensino da extensão do modelo vetorial do MIPS.

Ensinar organização e arquitetura de computadores com aulas de laboratório ou atividades práticas é uma técnica amplamente utilizada para ajudar alunos a alcançar um entendimento melhor de conceitos abstratos [3], [4]. A utilização de simuladores é largamente adotada em cursos de organização e arquitetura de computadores de graduação e pós-graduação. Este recurso pedagógico não exige uma infraestrutura especializada e contribui para o entendimento e assimilação de conceitos abstratos, como processamento vetorial e linguagem *assembly*.

Vários simuladores MIPS tradicionais [5], [6], [7], [8], [9] estão disponíveis para alunos desenvolverem atividades práticas. Porém, os simuladores atuais não oferecem suporte para instruções vetoriais. Com o surgimento das GPUs, que também oferecem recursos para ensino [10] além do alto desempenho, o processamento vetorial voltou a ser um tópico importante para melhorar o desempenho de programas na resolução de problemas complexos como, por exemplo, mineração de dados, extração de características em vídeos e processamento de imagem e vídeo [11].

O módulo da arquitetura MIPS SIMD (MIPS SIMD *Architecture* - MSA) é uma extensão para uma das arquiteturas de processadores mais utilizadas no âmbito industrial e acadêmico, que é o processador MIPS [1], [2]. Com o MSA é possível realizar o processamento paralelo para a execução de operações vetoriais de uma forma mais rápida e eficiente utilizando a abordagem SIMD (*Single Instruction, Multiple Data*). O MSA é semelhante também aos modelos MMX e SSE da Intel que foram propostos como extensões multimídia com execução SIMD. Este trabalho apresenta o primeiro simulador para a extensão vetorial do MIPS32 com suporte completo ao conjunto de instruções da MSA. Este simulador denominado VMARS (*Vector MIPS Assembler and Runtime Simulador*), por ter sido baseado no MARS [9].

Este artigo está organizado da seguinte forma. Na Seção II, explicamos os conceitos básicos da arquitetura MIPS SIMD. Na Seção III são apresentados os trabalhos correlatos. A Seção IV introduz o simulador VMARS e a Seção V apresenta alguns trechos de código para serem usados em sala de aula. Finalmente, na Seção VI, apresentamos as conclusões e trabalhos futuros.

II. Conceitos básicos

Inicialmente, esta Seção descreve os conceitos gerais dos processadores vetoriais. Posteriormente, a arquitetura vetorial MSA do simulador VMARS será detalhada.

A. Arquiteturas vetoriais: Instrução única, múltiplos dados

Processadores vetoriais são classificados na classe de arquiteturas SIMD (*Single Instruction Multiple Data*), na qual a mesma operação é executada sobre múltiplos dados, em constrate com as arquiteturas escalares ou mono-processadores que são classificadas como processadores SISD (*Single Instruction, Single Data*). Além disso, outra característica importante é a presença de registradores vetoriais. A vantagem

principal é que o programador ainda pode pensar de forma sequencial enquanto opera sobre dados paralelos através dos uso de vetores. As operações vetoriais são realizadas através dos seguintes passos:

1) Os dados são carregados da memória para os registradores vetoriais utilizando a unidade de *Load/Store* vetorial;
2) Uma ou mais instruções são executadas sobre os registradores vetoriais e/ou escalares;
3) Os resultados são gravados dos registradores vetoriais para a memória.

A Figura 1 apresenta o diagrama básico de uma arquitetura vetorial cujo modelo original foi proposto por S. Cray na década de 70. Em particular, a arquitetura MSA do VMARS trata os registradores vetoriais como palavras (*words*), de forma similar aos modelos MMX e SSE da Intel. Os principais módulos do processador vetorial MIPS são:

- **Memória de dados**: A memória principal do MIPS vetorial é compartilhada com o conjunto de instruções escalares MIPS.
- **Unidade vetorial de Load/Store**: Esta unidade é responsável pela comunicação entre os registradores vetoriais e a memória. Este módulo é normalmente implementado utilizando *pipeline* aumentando a largura de banda e a vazão com paralelismo temporal mascarando a latência de memória.
- **Registradores vetoriais**: 32 registradores vetoriais de 128 bits cada. Eles podem ser manipulados como um vetor de *bytes*, *halfwords*, *word* ou *double words*.
- **Unidades funcionais**: Diversos tipos de unidades funcionais estão disponíveis no MIPS vetorial. Elas são responsáveis por executar instruções de ponto flutuante, de inteiros e/ou operações lógicas.
- **Registradores escalares**: Os 32 registradores da arquitetura MIPS original também estão disponíveis na versão vetorial.

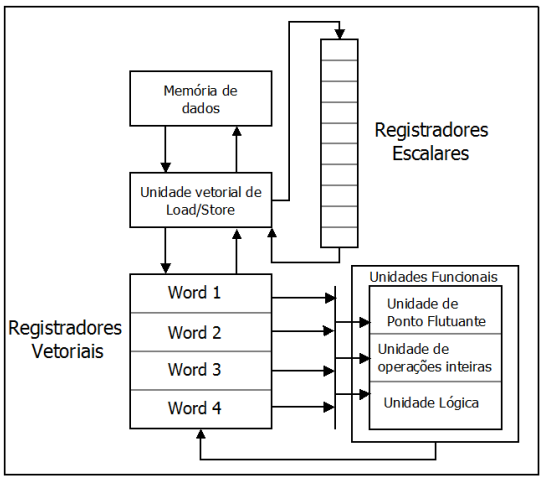

Fig. 1. Arquitetura de um processador vetorial.

B. Arquitetura MIPS SIMD

A MSA (MIPS SIMD Architecture) é uma extensão SIMD para a arquitetura padrão MIPS Release 5. Esta extensão foi recentemente especificada pela MIPS Technologies, Inc. [11] e utiliza os princpios RISC (*Reduced Instruction Set Computers*). A MSA adiciona mais de 150 novas instruções na arquitetura MIPS original, incluindo também mais 32 registradores de 128 bits. Estes registradores podem ser indexados e particionados de forma vetorial com quatro tipos de dados: *byte* (8 bits), *halfword* (16 bits), *word* (32 bits) e *doubleword* (64 bits) [11]. De acordo com o formato de dado que é especificado pela instrução, o número de elementos é indexado de 0 a n onde $n = \frac{128}{formato}$.

O bit menos significativo (LSB) é o bit mais à direita e o mais significativo (MSB) é o mais à esquerda. A Figura 2 apresenta um exemplo de registrador vetorial indexado com o tipo de dados *word*, onde temos 4 elementos $n = \frac{128}{32} = 4$.

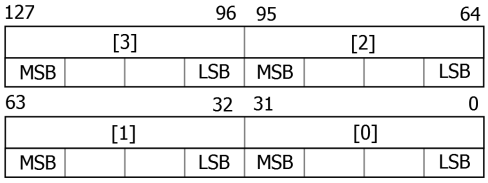

Fig. 2. Elementos do tipo *word* de um registrador vetorial do VMIPS.

Cada instrução pode atuar em mais de um formato de dado. O exemplo a seguir ilustra a instrução *addv* em dois formatos: byte e word.

- *addv.***b** *wd,ws,wt* - Soma os *bytes* dos registradores vetoriais *ws* e *wt* e guarda o resultado em *wd*.
- *addv.***w** *wd,ws,wt* - Soma as *words* dos registradores vetoriais *ws* e *wt* e guarda o resultado em *wd*.

III. Trabalhos Relacionados

Existem diversos simuladores para a arquitetura MIPS original, mas nenhum deles implementa o conjunto de instruções vetoriais. O SPIM [5] é um dos simuladores MIPS mais utilizados. Além do simulador SPIM existem muitos outros simuladores MIPS, onde podemos destacar o MIPSASM [6], o TinyMIPS [7], o EduMIPS64 [8] e o MARS [9]. O EduMIPS64 foi o primeiro simulador MIPS a suportar a extensão 64 bits. O VMARS foi desenvolvido à partir do simulador MARS devido à sua facilidade de uso, popularidade e flexibilidade para inclusão de novas funcionalidades.

Uma comparação entre o SPIM e o MARS foi realizada em [9]. A IDE MARS é mais intuitiva que a SPIM. Por exemplo, no MARS é mais fácil e mais rápido modificar valores dos registradores e utilizar *breakpoints*. O SPIM não fornece uma IDE completa. A principal diferença entre o MARS e o VMARS é que o VMARS adiciona um conjunto com mais de 150 instruções vetoriais e recursos de visualização para os formatos de dados vetoriais.

IV. Simulador VMARS

A interface implementada para o VMARS fornece recursos como preenchimento automático e destacamento de sintaxe

para a linguagem *assembly* do MIPS e da extensão vetorial MSA. Além de incluir registradores específicos que estão disponíveis somente na extensão vetorial.

Para ensino do processamento vetorial, um componente importante é a visualização dos conteúdo dos registradores. Existem 5 abas diferentes no painel de registradores do VMARS. Foram incluídas duas novas abas: *Vector Registers* e *Float Vector Registers* como apresentado na Figura 3. Os valores dos 32 registradores vetoriais são apresentados como números inteiros com sinal ou valores hexadecimais. Os registradores vetoriais inteiros são representados por $\$wn$ onde n é o número do registrador correspondente. A aba relacionada aos registradores vetoriais de ponto flutuante apresenta os valores de todos os 32 registradores que compõem a unidade de ponto flutuante. O VMARS considera a unidade de ponto flutuante como um coprocessador. Os registradores vetoriais de ponto flutuante são representados por $\$cn$ onde n é o número do registrador correspondente. É importante notar que existem dois conjuntos de registradores: $\$C_n$ para ponto flutuante e $\$W_n$ para inteiros. Eles são dois grupos independentes de 32 registradores vetoriais que totalizam 64 registradores.

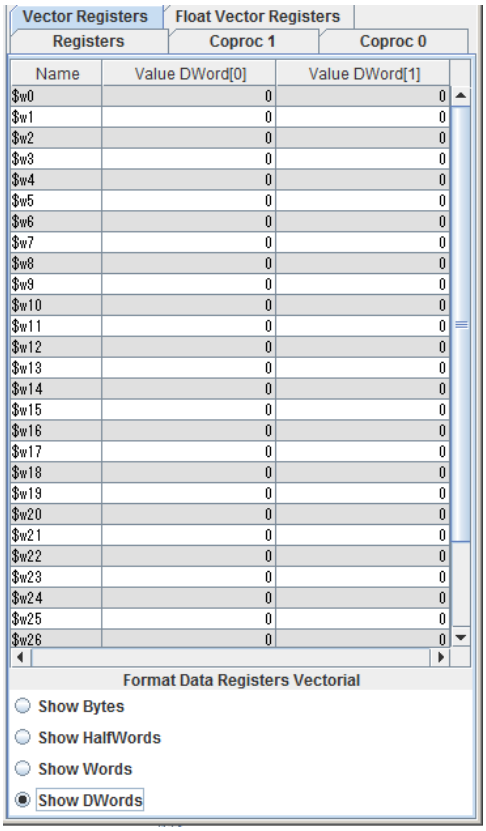

Fig. 3. Abas de registradores do VMARS.

Ambas as abas *Vector Registers* e *Float Vector Registers* podem ser configuradas para apresentar dados em diferentes formatos. A Tabela I apresenta as opções de visualização disponíveis. As instruções que utilizam *Registradores de ponto flutuante vetoriais* são executadas somente em formato *word* e *doubleword*, que são, respectivamente, precisão simples e dupla no formato IEEE 754 [12].

O VMARS implementa as instruções vetoriais MSA, que

TABLE I. FORMATOS DISPONÍVEIS PARA APRESENTAR OS DADOS DE REGISTRADORES VETORIAIS.

Opção	Formato	Vector Reg.	Float Vector Reg.
Show bytes	8 bits	Disponível	Não Disponível
Show Halfwords	16 bits	Disponível	Não Disponível
Show Words	32 bits	Disponível	Disponível
Show Double Words	64 bits	Disponível	Disponível

utilizam o sufixo para determinar variações da instrução vetorial. Por exemplo, *add.b* significa que a instrução irá somar os bytes de dois registradores vetoriais. Por outro lado, *add.v* realiza a operação de adição entre os registradores vetoriais inteiros.

O código *assembly* e de máquina são apresentados na interface do VMARS, que também possibilita a utilização de *breakpoints*.

V. EXEMPLOS

O VMARS pode ser utilizado em aulas práticas e como ferramenta de visualização para facilitar o entendimento de conceitos relacionados a arquiteturas vetoriais. Esta seção apresenta dois exemplos retirados do livro “Arquitetura de Computadores - Uma Abordagem Quantitativa” [2], onde um conjunto de instruções VMIPS foi especificado com 32 instruções sem detalhamento de formato. Este conjunto tem fins didáticos para ilustrar os principais recursos e operações de processadores vetoriais. Enquanto que o VMARS segue a especificação formal do MSA com mais de 150 instruções vetoriais e formato detalhado em binário. Os dois exemplos serão ilustrados em três formatos: algoritmo alto nível, VMIPS e VMARS MSA.

O primeiro exemplo é o DAXPY que é amplamente usado na literatura. DAXPY é a sigla para *Double-precision AX Plus Y*, onde A é um escalar, X, Y são dois vetores. O pseudocódigo para o DAXPY apresentado na Figura 4(a).

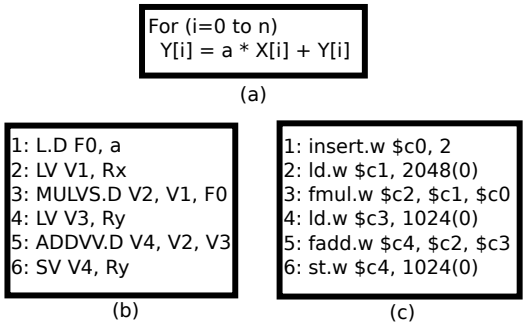

Fig. 4. DAXPY: (a) pseudo código; (b) VMIPS; (c) VMARS.

Na implementação VMIPS, cada registrador vetorial tem 64 palavras de 64 bits. Supondo que n seja menor que 64, a implementação VMIPS pode ser realizada sem laço como ilustra a Figura 4(b), onde o escalar A é armazenado no registrador F_0 e os vetores X e Y nos registradores vetoriais V_1 e V_3, respectivamente. O registrador vetorial V_2 é usado temporariamente e o resultado final foi gerado no registrador V_4. O VMIPS segue a sintaxe do MIPS adicionando o sufixo **V** para as instruções vetoriais.

Já a sintaxe do VMARS segue a especificação formal definida em [11]. A implementação VMARS para o DAXPY

está ilustrada na Figura 4(c). Os registradores c_i são vetoriais para ponto flutuante. Semelhamente ao código do VMIPS temos para este exemplo o escalar A armazenado no registrador c_0 e os vetores X e Y nos registradores vetoriais c_1 e c_3, respectivamente. O registrador vetorial c_2 é usado temporariamente e o resultado final foi gerado no registrador c_4.

O segundo exemplo ilustra a vetorização de um código com condicionais dentro de laços. O pseudo código para a operação é mostrado na Figura 5(a). O laço é executado com 2 vetores. Se a posição i no vetor X for diferente de 0, Y_i é subtraído de X_i.

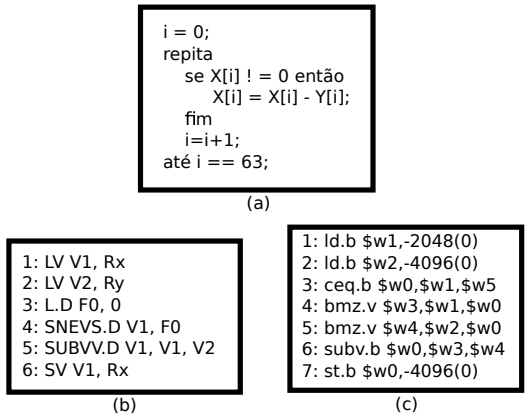

Fig. 5. Laço condicional: (a) pseudo código; (b) VMIPS; (c) VMARS.

Uma máscara vetorial é utilizada para resolver este tipo de problema, fornecendo execução condicional para cada operação entre elementos de vetores em uma instrução vetorial. A máscara vetorial é um vetor booleano. O algoritmo ilustrado na Figura 5(b) apresenta o código com instruções pseudo VMIPS para a operação. Inicialmente, Os vetores X e Y são carregados em V_1 e V_2. O escalar zero é carregado em F_0. A instrução SNEVS.D percorre V_1 para verificar se $V_1[i]! = F_0$ e gerar a máscara V_0. A instrução SUBVV.D subtrai $V_2[i]$ de $V_1[i]$ se $V_0[i]$ que é a máscara vetorial for 1. O resultado é guardado em V_1. e a instrução SV grava o resultado na memória.

A implementação VMARS é ilustrada na Figura 5(c). Primeiro, os vetores X e Y são carregados nos registradores w_1 e w_2 no formato de bytes utilizando a instrução ld.b. A instrução ceq.b percorre w_1 comparando com w_5 para gerar a máscara para X que está em w_1. As instruções bmz.v são usadas para carregar X e Y segundo a máscara. Por fim, w_0 executa a subtração e o resultado é salvo na memória.

VI. Conclusão e Trabalhos Futuros

Este trabalho apresenta o primeiro simulador de um processador MIPS vetorial, denominado VMARS. Este simulador foi desenvolvido para auxiliar estudantes no entendimento de conceitos abstratos de processadores vetoriais. Neste trabalho foram descritas as operações básicas do simulador VMARS juntamente com exemplos de códigos vetoriais para ilustrar sua utilização em sala de aula.

Podemos destacar que o VMARS executa um conjunto completo de mais de 150 instruções vetoriais seguindo a especificação formal do MSA [11]. As facilidades na edição dos códigos com texto de ajuda para a sintaxe das instruções agilizam o aprendizado. A visualização dos registradores vetoriais auxilia na compreensão dos conceitos de máscaras dentre outros.

Como trabalhos futuros, o simulador será avaliado nas disciplinas de Arquitetura de Computadores da Universidade Federal de Viçosa. Com o retorno dos estudantes pretendemos implementar melhorias no simulador VMARS. Após esta etapa, o código fonte, o simulador, roteiros com exemplos e sugestões de aulas práticas serão disponibilizados a toda comunidade acadêmica.

Agradecimentos

Nós gostaríamos de agradecer a CAPES, CNPQ, FAPEMIG, FUNARBE, GAPSO e a UFV pelo apoio financeiro e ao projeto VMARS por fornecer a base para o desenvolvimento deste projeto. Agradecemos também ao aluno Fernando Augusto por suas contribuições no desenvolvimento do simulador.

References

[1] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 3rd ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007.

[2] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*, 5th ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2011.

[3] R. S. Ferreira, A. C. S. Beck, L. Carro, A. Toledo, and A. Silva, "A java framework to teach computer architecture," in *New Trends and Technologies in Computer-Aided Learning for Computer-Aided Design*. Springer, 2005, pp. 25–35.

[4] R. Ferreira, J. Nacif, S. Magalhaes, T. Almeida, and R. Pacifico, "Be a simulator developer and go beyond in computing engineering," in *Frontiers in Education Conference (FIE), 2015 IEEE*, Oct 2015, pp. 1–8.

[5] J. R. Larus, "Spim: A mips32 simulator," 1990–2010, http://sourceforge.net/projects/spimsimulator/ (Visited on April 15, 2015).

[6] K. Meinz, "Webmipsasm v2.1," 2015, http://www.kurtm.net/mipsasm/index.cgi (Visited on April 15, 2015).

[7] T.Pittman, "Tinymips (machine instruction processing simulator)," 2003, http://www.ittybittycomputers.com/Courses/Prior/MachOrg/GateSim/TinyMIPS.htm (Visited on April 15, 2015).

[8] D. Patti, A. Spadaccini, M. Palesi, F. Fazzino, and V. Catania, "Supporting undergraduate computer architecture students using a visual mips64 cpu simulator," *Education, IEEE Transactions on*, vol. 55, no. 3, pp. 406–411, Aug 2012.

[9] K. Vollmar and P. Sanderson, "Mars: An education-oriented mips assembly language simulator," *SIGCSE Bull.*, vol. 38, no. 1, pp. 239–243, Mar. 2006.

[10] G. Fontes and R. Ferreira, "Ensino de organizacoes de memoria em arquiteturas paralelas usando placas graficas aceleradores," *International Journal of Computer Architecture Education (IJCAE)*, vol. 2, no. 1, 2013.

[11] I. Technologies., *MIPS Architecture for Programmers Volume IV-j: The MIPS32 SIMD Architecture Module*, 1st ed. 955 East Arques Avenue, Sunnyvale, CA 94085-4521: MIPS Technologies Inc., 2014.

[12] "IEEE Standard for Floating-Point Arithmetic," Microprocessor Standards Committee of the IEEE Computer Society, 3 Park Avenue, New York, NY 10016-5997, USA, Tech. Rep., Aug. 2008.