International Journal of Computer Architecture Education (IJCAE)

AD3W: Um simulador educacional para análise de dependências de dados em nível de instrução

Alexandre S. Roque; Guilherme Schievelbein; Maikel Losekann; Denilson Rodrigues da Silva

Departamento de Engenharias e Ciência da Computação - DECC
Laboratório de Circuitos e Sistemas Digitais - LABCD
Universidade Regional Integrado do Alto Uruguai e das Missões – URI – Santo Ângelo, RS, Brasil

{guilhermeschievelbein, maikellosekan}@hotmail.com, {deniro, roque}@santoangelo.uri.br

***Resumo* — Este artigo apresenta um simulador para a análise de dependências em nível de instrução, com o objetivo de ilustrar de forma educacional as possibilidades de paralelismo em um programa. O algoritmo de controle utiliza como técnica a detecção de desvios condicionais e escritas em registradores (WAR, WAW e RAW), para analisar dependências a nível de instrução. Como resultados obtidos destacam-se a redução no tempo de realização de exercícios e na sua correção em sala de aula, além da percepção prática de que o simulador ajuda os alunos na compreensão dos conceitos relacionados ao paralelismo em nível de instrução.**

***Palavras Chave* — Análise de dependências; paralelismo em nível de instrução; linguagem assembly.**

I. INTRODUÇÃO

No estudo de arquitetura de computadores, diversas ferramentas e técnicas são usadas para auxílio no processo de ensino aprendizagem [1] [3] [12]. No contexto de ferramentas de auxílio os simuladores se destacam por serem ótimas opções para ilustrar conceitos de maneira simples e com um bom nível de interação, promovendo uma prática que muitas vezes fica somente em leituras avançadas de determinado conteúdo [5][9][10]. Por conseguinte, em arquiteturas paralelas também é importante ilustrar de forma educativa as possibilidades de paralelismo em nível de instrução, como um escalonador pode distribuir instruções em arquiteturas e ambientes multiprocessados, com o objetivo de prover um ganho de *speedup* (*tempo total / n processadores*) e economia de recursos [11].

Desta forma, o uso de ferramentas e softwares de apoio ao ensino encoraja e estimula o estudo nas áreas de processamento paralelo e computação de alto desempenho (PAD/HPC). Assim, é possível efetuar um estudo detalhado das propriedades de um sistema e observar seu funcionamento interno [3].

Dentro deste contexto, o objetivo deste artigo é apresentar uma ferramenta que auxilie de forma simples o processo de ensino-aprendizagem com relação à verificação de dependências de dados (*data hazards*) em instruções *assembly* para detectar possibilidades de paralelismo em determinados conjuntos de instruções.

Este artigo esta organizado da seguinte maneira: A Seção II, faz uma breve contextualização dos conceitos envolvidos e descreve alguns aspectos metodológicos; a Seção VI, descreve e apresenta a o desenvolvimento do simulador; a Seção V, apresenta alguns resultados dos testes realizados com alguns exercícios dados em aula; por fim, a Seção VI apresenta as conclusões relativas ao desenvolvimento do trabalho.

II. CONTEXTUALIZAÇÃO E ASPECTOS METODOLÓGICOS

Para obtenção do paralelismo em nível de instrução a utilização de pipelines paralelos nas arquiteturas é fundamental. Três políticas de iniciação de instruções são tratadas para o aprendizado dos pipelines [7]:

- Iniciação em Ordem com terminação em Ordem;

- Iniciação em Ordem com terminação fora de Ordem;

- Iniciação fora de Ordem com terminação fora de Ordem.

O simulador proposto trata destas políticas de forma a ilustrar esta maneira de execução, por meio de exemplos e exercícios que podem ser previamente discutidos antes da execução. As informações de dependência de dados são essenciais para detectar iterações de laços que podem ser executadas em paralelo por diferentes arquiteturas de processadores [13]. Dependência de dados ocorre quando uma instrução depende do resultado de outra instrução que ainda não foi executada (ainda está no pipeline). Este tipo de dependência é originado na natureza sequencial do código em execução, cuja ordem normal é alterada dentro do pipeline.

De acordo com [7] os tipos de dependências (*hazard classification*) são organizados em: *data hazards, structural hazard e control hazard/branching hazards.*

A análise de dependências nestes níveis é importante para tratamento de diversos problemas comuns que são abordados na literatura de ensino de arquitetura de computadores e possibilitam diversos algoritmos e técnicas para tratamentos específicos, como por exemplo, um algoritmo para renomeação de registradores, o algoritmo de *Tomasulo* (que é

mais abrangente), e algoritmos de predição de desvios (*branch prediction*) [7]. No simulador desenvolvido são tratadas as dependências de dados (*data hazards)* e os desvios de controle (*branching hazards),* com o intuito de abordar os referidos algoritmos.

- Dependência de dados / *Data hazards*

- WAR (*Write After Read*) – ocorre quando a instrução i+1 tenta escrever um dado antes que a instrução i possa lê-lo;
- RAW (*Read After Write*) – ocorre quando a instrução i+1 tenta ler um dado que a instrução i ainda não atualizou;
- WAW (*Write After Write*) – ocorre quando a instrução i+1 tenta escrever um dado antes que a instrução i atualize o mesmo;

- Dependencia de desvios / *Branching hazards*

- DESVIO – instruções a baixo de uma instrução de desvio.

Lembrando que dependências do tipo RAR (Read After Read) não é considerada uma dependência. A Fig. 1 ilustra alguns exemplos de dependências.

RAW	*WAR*	*WAW*
A = B + C	A = B + C	A = B + C
X = A + Y	C = X + Y	A= X + Y

Fig 1: Exemplos de dependências. (Fonte: Autor)

A análise das dependências de dados é fundamental para possibilitar otimizações e detecção de paralelismo implícito em programas sequenciais. Esta análise oferece as informações necessárias para realizar transformações coerentes capazes de proporcionar melhorias de desempenho em determinadas aplicações [13].

O simulador AD3W foi desenvolvido na linguagem de programação JAVA, linguagem que oferece suporte a orientação a objetos, com o objetivo de reduzir as dificuldades de aprendizagem sobre a detecção de dependência de dados, sendo amplamente utilizada nas disciplinas de arquitetura de computadores. Uma das principais causas que motivaram o desenvolvimento nesta plataforma foi a facilidade oferecida para o desenvolvimento de interfaces gráficas com o usuário. Diferentemente das linguagens convencionais, que são compiladas para código nativo, a linguagem JAVA é compilada para um *bytecode* [8] (resultado de um processo semelhante ao dos compiladores de código-fonte que não é imediatamente executável, mas sim interpretado) que é executado por uma máquina virtual, oferecendo, assim, uma maior portabilidade. O desenvolvimento de simuladores apresenta diversos exemplos de *softwares* desenvolvidos na plataforma JAVA, principalmente os *Applets*. Entre estes aplicativos observamos o *Easy Java Simulations* que provê um mecanismo de criação de simuladores de Física em JAVA [2].

III. TRABALHOS RELACIONADOS

Para o estudo de um conjunto de instruções, simuladores são comumente utilizados no ensino de características inerentes a arquitetura de computadores, e muitos deles baseiam-se no processador MIPS [5] [6].

Para o desenvolvimento deste trabalho foram analisados alguns trabalhos relacionados, como por exemplo o R10k, um simulador de arquitetura superescalar [3] e o PS – CAS MIPS: um simulador de pipeline do processador MIPS [4]. Com base nestas referências, procurou-se diferenciar o simulador proposto focando nas técnicas para análise de dependências de dados em nível de instrução e na sua prática educacional, não utilizando também um ISA específico como o MIPS, mas sim um conjunto padrão/genérico que represente instruções comuns utilizadas na maioria dos processadores.

IV. PROPOSTA DO SIMULADOR

Para o desenvolvimento do simulador foram considerados aspectos relativos à programação orientada a objetos, estabelecendo relação entre uma classe e seus atributos com a composição de uma instrução *assembly*.

Como cada instrução tem vários aspectos (atributos) que divergem entre si e estes são comparados uns com os outros para detectar as dependências de dados, esta comparação acaba se tornando menos complexa quando cada aspecto destes é definido como um atributo de uma classe, no caso como a definição de uma classe "instruções", realizada para o desenvolvimento do simulador AD3W, descrito na Fig. 2.

```
Instrucao
- private id : int
- private nroCiclos : int
- private cicloExe : int
- private cicloSaida : int
- private receptor : String
- private op1 : String
- private op2 : String
- private nop : boolean = false
- private rot : boolean = false
- private desvio : boolean = false
- private executado : boolean = false
- private naJanela : boolean = false
- private foiPrim : boolean = false
- private escrita : boolean = false
- private WARcom [ ] : Integer = new ArrayList <>(): Integer
- private RAWcom [ ] : Integer = new ArrayList <>(): Integer
- private WAWcom [ ] : Integer = new ArrayList <>(): Integer
- private DESVcom [ ] : Integer = new ArrayList <>(): Integer
```

Fig 2: Classe que compõe os aspectos de uma instrução. (Fonte: Autor)

Para fazer a análise são atribuídos a cada instrução alguns atributos de controle, que são:

- Id: identificação de cada instrução, varia de 1 a "n" em que n é o número total de instruções, atribuído de forma crescente a cada instrução.
- Número de ciclos: a quantidade de ciclos que cada instrução levará para executar.
- Número de ciclos executados: número incrementado a cada ciclo de execução e representa quantos ciclos da instrução foi executado.

- Ciclo de saída: qual o ciclo de execução que a instrução irá sair da janela de execução.
- Operadores: representam os registradores de cada instrução, o receptor (registrador que recebe o resultado da operação) e os operandos.
- Variáveis booleanas: são variáveis que irão representar se a instrução é uma instrução de desvio, de rótulo ou do tipo NOP (*no operation* ou finalização), bem como se a instrução foi executada, foi escrita, se já foi para a execução ou ainda está na janela de instruções.

Além destes valores cada instrução possui uma lista contendo quais instruções possuem dependências da mesma. As instruções são carregadas um uma lista, onde cada uma ocupa uma posição. Neste momento que são atribuídos os valores citados acima, sendo que alguns podem ter seus valores alterados durante a execução do simulador. Para ter acesso a uma instrução na lista, basta referenciar sua "id" que resultará na posição ocupada por ela na lista. A seguir a Fig. 3 mostra como o algoritmo para a análise das instruções foi desenvolvido, detalhando as decisões e aspectos mais relevantes a considerar.

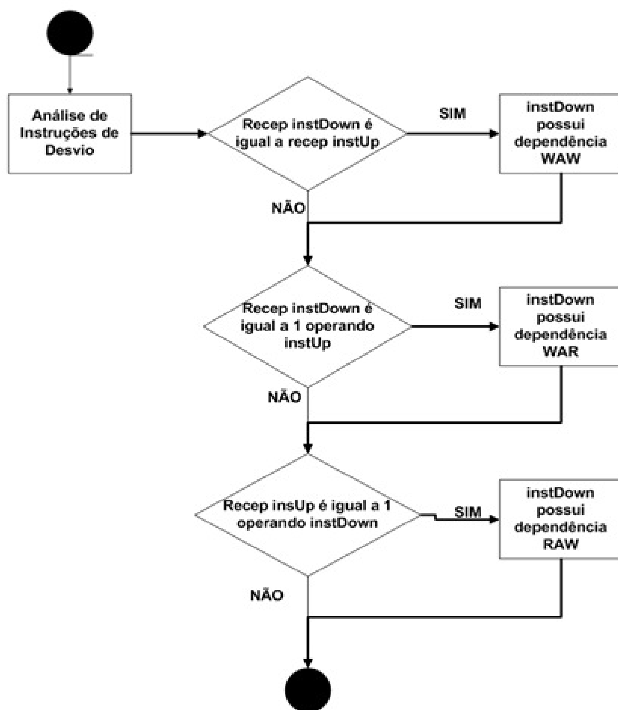

Fig. 3: Fluxograma do algoritmo (Fonte: Autor)

Para a análise de dependências é realizada uma estrutura de repetição que percorre o array de instruções a partir da segunda instrução até a última, em cada iteração desta repetição cada instrução é atribuída a uma instância auxiliar de instruções, definida em ***instDown***. Em outra repetição interna cada instrução i-1, em que i representa a posição de instDown, é atribuída a instUp, onde serão realizadas as comparações, como ilustra a Fig. 3.

Para verificar a dependência do tipo WAR compara-se o registrar destino da instrução i com os operandos da instrução i-1, caso um deles seja igual, a instrução i tem dependência do tipo WAR com a instrução i-1. Para verificar se há dependências do tipo RAW compara-se o registrador destino da instrução i-1 com os operandos da instrução i. E para verificar instruções do tipo WAW compara-se o registrador destino de i com o registrador destino de i-1. Prioritariamente é verificada a ocorrência de instruções de desvio, para tal, é percorrida a lista até chegar ma instrução de desvio "i".

Como definido anteriormente, as dependências analisadas até o momento pelo simulador AD3W são do tipo: WAR, RAW, WAW e de Desvio.

Para interação com um aluno realizando exercícios em sala de aula, o simulador solicita ao usuário que informe as instruções em uma linguagem *assembly* básica (o ISA utilizado no momento é genérico/básico, ou seja, representa as principais operações executadas em qualquer processador), contendo dois registradores de operandos, juntamente com o número de ciclos que a instrução levará para executar, característica adicionada para fins de simular instruções mais demoradas, como demonstrado no exemplo da Fig 4.

```
Forma de execução das instrucoes:

R1 = R1 + R3  ----------------->  com 1 ciclos de execucao
R2 = R2 + R1  ----------------->  com 1 ciclos de execucao
R1 = R1 / R9  ----------------->  com 1 ciclos de execucao
R4 = R4 - R9  ----------------->  com 3 ciclos de execucao
R5 = R5 / R6  ----------------->  com 1 ciclos de execucao
R7 = R7 * R6  ----------------->  com 1 ciclos de execucao
R9 = R9 + 10H  ----------------->  com 1 ciclos de execucao
SE R9 = 10H DESVIA PARA ROT  ----------------->  com 1 ciclos de execucao
R7 = R7 + R9  ----------------->  com 1 ciclos de execucao
R7 = R7 + 1H  ----------------->  com 1 ciclos de execucao
R10 = R10 + R6  ----------------->  com 1 ciclos de execucao
```

Fig. 4: Forma de execução das instruções. (Fonte: Autor)

```
1   ADD R1,R3 1
2   ADD R2,R1 1
3   DIV R1,R9 1
4   SUB R4,R9 3
5   DIV R5,R6 1
6   MUL R7,R6 1
7   ADD R9,10h 1
8   BNQ R9,10h, ROT 1
9   ADD R7,R9 1
10  ADD R7,1h 1
11  ADD R10,R6 1
12  ROT 1
```

Neste exemplo, o número a direita de cada instrução apenas indica quantos ciclos de *clock* a mesma gastará, de acordo com a entrada do usuário.

Fig. 5: Entrada de dados para a simulação. (Fonte: Autor)

A análise da entrada do usuário resulta em três informações de saída: a forma de execução das instruções, que pode ser visualizada através da Fig. 4, as colunas de decodificação, janela (que é implementada de forma centralizada), a execução e escrita e qual o tipo de dependência que cada instrução possui com as demais instruções, desta forma, é possível verificar o conflito que gera uma dependência de dados.

Na seção IV, a Fig. 6 ilustra em mais detalhes a área do simulador que faz a análise da execução do conjunto de instruções, para fins de melhor compreensão.

V. RESULTADOS

Na Fig. 6 são apresentados os resultados obtidos pelo simulador.

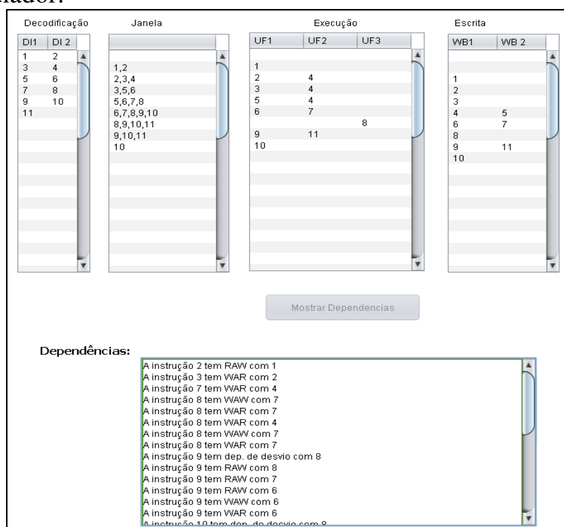

Fig. 6: Tela para apresentação dos resultados ao usuário. (Fonte: Autor)

Primeiramente são mostradas as instruções que foram decodificadas. Na tabela “Janela” são mostradas as instruções que estão na janela de instruções, ou seja, as instruções que estão na espera para executar. Na tabela “execução” são mostradas as instruções e o número de ciclos que cada uma leva para executar, e cada linha representa um ciclo de *clock*. As colunas UF1, UF2 e UF3 representam as unidades funcionais disponibilizadas para a simulação, de acordo com tipos que podem ser configurados no simulador. Por fim é possível visualizar as dependências que as instruções possuem.

Para fins de verificar a efetividade da utilização do simulador, foram realizados testes em sala de aula, onde foi disponibilizado para um grupo de alunos, 3 (três) atividades a serem resolvidas manualmente e com o auxilio do Simulador AD3W. Para a realização das atividades 1, 2 e 3 foram disponibilizados 10, 20 e 30 minutos, com diversos exercícios contendo 10 instruções *assembly* cada. Estes exercícios foram dimensionados com diferentes níveis de dificuldade, alguns com mais dependências que outros. O gráfico da Fig. 7 elenca os resultados de tal teste mostrando a quantidade de exercícios resolvidos manualmente e com o auxilio do simulador.

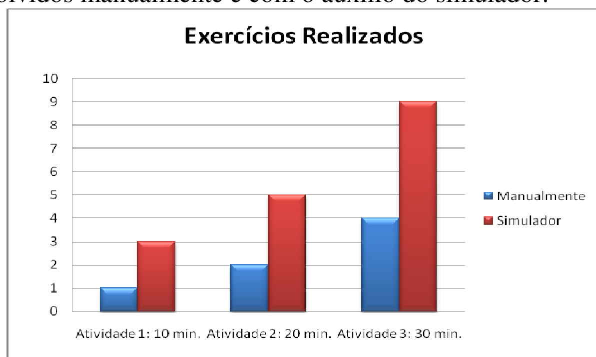

Fig. 7: Gráfico de testes das atividades. (Fonte: Autor)

VI. CONCLUSÕES

Com a apresentação do simulador AD3W é proposta uma forma alternativa, de maneira simples e rápida, para ensino dos conceitos de paralelismo em nível de instrução com identificação de diferentes tipos de dependências. Pode-se destacar a forma de interação com o aluno, pois, o simulador possibilita a configuração de acordo com as características dos exercícios a realizar (quantidade de unidades funcionais, quantidade de ciclos de *clock* por classe de instrução).

De acordo com a literatura estudada e os testes realizados, foi possível verificar como os simuladores podem ajudar no processo de ensino de diversos conceitos de arquitetura de computadores. Com base nos testes realizados pode-se verificar um aproveitamento de mais de cem por cento no tempo para a realização efetiva dos exercícios. Outros benefícios identificados pelo uso do simulador foi à redução do tempo de correção e também um melhor entendimento do conteúdo por parte dos alunos.

REFERÊNCIAS

[1] A. I. T. Ribeiro; A. Rimsa. Técnica Motivacional para o Ensino de Arquitetura de Computadores com Ênfase nos Grandes Desafios da Computação. Workshop sobre Educação em Arquitetura de Computadores - WEAC 2008.

[2] A. L. Almeida; M. F. O. Araujo. Utilização de Ferramentas Multimidia Para a Construção de Simuladores de Fenomenos Físicos. Universidade do Estado da Bahia. BA. Brasil. 2008.

[3] A. N. Gonçalves; R. C. L Silva; R. A. L. Gonçalves; J. A. Martini; R10k: Um Simulador de Arquitetura Superescalar. Workshop Sobre Educação em Arquitetura de Computadores – WEAC, 2007.

[4] W. N. M. Davidson; M. V. Marcelo; F. P. Ramon; PS – CAS MIPS: Um Simulador De Pipeline Do Processador MIPS 32 Bits Para Estudo de Arquitetura de Computadores. Workshop sobre Educação em Arquitetura de Computadores - WEAC 2009

[5] M. Brosson; MipsIt: a simulation and development environment using animation for computer architecture education. In Proceedings of 2002 Workshop on Computer Architecture Education: Held in Conjunction with the 29th international Symposium on Computer Architecture (Anchorage, ACM, New York, NY, 12. p. WCAE‘02.Alaska),p.1 -8.

[6] B. F. Souza; M.P.A. Moreira; R. S. Nogueira; C.A.P.S. Martins; WebSimple MIPS. Workshop de Sistemas Computacionais de Alto Desempenho – WSCAD, 2008, p. 1 – 4.

[7] D. A. Patterson, J. L. Hennessy. Arquitetura de Computadores: uma abordagem quantitativa. 4. ed. Rio de Janeiro: Elsevier, 2008.

[8] H. M. Deitel; Java, como programar. Porto Alegre: Bookman. 2003.

[9] H. Grunbacher; Teaching computer architecture/organisation using simulators, 28th Annual Frontiers in Education Conference, p.1107-1112 vol. 3, 1998.

[10] V. Heckler; F. M. Saraiva. Uso de Simuladores, Imagens e Animações como Ferramentas Auxiliares no Ensino/Aprendizagem de Óptica. Revista Brasileira de Ensino de Física, v. 29, n. 2, p. 267-273, 2007.

[11] J. Dongarra,, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White. Sourcebook of Parallel Computing. Morgan Kaufmann Publishers Inc., San Francisco, USA, 2003.

[12] L. M. Coutinho; J. L. Mendes; C. A. Martins; Web-MHE: Ambiente web de auxílio ao aprendizado de hierarquia de memória. Workshop sobre Educação em Arquitetura de Computadores - WEAC 2006.

[13] C. B. Martins. Detecção de Paralelismo a partir de Semântica Denotacional e de Grafos de Dependências. Dissertação de Mestrado. Pontifícia Universidade Catolica. Rio de Janeiro. Brasil. 2000.