T&D-Bench2 - Um Ambiente de Modelagem de Processadores

Sandro Neves Soares[1]
Universidade de Caxias do Sul
CARVI
snsoares@ucs.tche.br

Flávio Rech Wagner
Universidade Federal do Rio Grande do Sul
Instituto de Informática
flavio@inf.ufrgs.br

Resumo

Este artigo apresenta o T&D-Bench2, um ambiente para o desenvolvimento de modelos de simulação de processadores do estado da arte a serem usados, preferencialmente, no ensino. O T&D-Bench2 concilia a flexibilidade existente em ambientes de projeto de processadores, usando HDLs ou ADLs, no que diz respeito às inúmeras alternativas arquiteturais que podem existir nestes processadores, com os recursos especializados de interação, tanto na concepção do modelo como na sua execução, existentes em ambientes de ensino.

1. Introdução

O ensino dos conteúdos de Arquitetura de Computadores está longe de ser uma tarefa trivial, e os resultados acabam não sendo satisfatórios nem para educadores, nem para estudantes. Isso é causado por vários fatores, como a complexidade dos temas abordados, a grande e sempre crescente quantidade de conteúdos a ser desenvolvida e a maior dificuldade de aplicação direta dos conhecimentos adquiridos em comparação com outras áreas da Computação. Porém, o maior problema parece ser mesmo como ajudar os estudantes a fazer a ligação entre o seu conhecimento teórico e a experiência prática. A Arquitetura trata, fundamentalmente, do hardware do computador e este, pelo fato de não se prestar à experimentação direta, dificulta a prática da teoria desenvolvida.

Tomando-se, como base, uma unidade de ensino fundamental da área de Arquitetura de Computadores, a que trata da organização interna de processadores, a impossibilidade de experimentação direta é compensada pelo uso de ambientes de software que podem ser de duas classes distintas: ambientes de projeto e ambientes de ensino. Ambientes de projeto são mais complexos em sua utilização, uma vez que devem oferecer recursos, normalmente linguagens especializadas, para o projeto de uma grande variedade de processadores. Eles geram, normalmente, ferramentas de software para a experimentação do processador projetado (simuladores, compiladores, montadores, depuradores). Ambientes de ensino, por outro lado, devem oferecer uma interação mais simples com o usuário e tratam, geralmente, de um processador específico, mas possuem interfaces especializadas para a configuração de aspectos do funcionamento deste e para a visualização de resultados.

Este artigo apresenta o ambiente T&D-Bench2, que alia a flexibilidade e o poder existentes em ambientes de projeto de processadores com os recursos especializados de interação existentes em ambientes de ensino. Tal ambiente é uma evolução do T&D-Bench, apresentado em [1].

Este artigo está organizado da seguinte maneira. A Seção 2 discute trabalhos relacionados em ambientes de ensino e de projeto. A metodologia de modelagem de processadores do T&D-Bench2, bem como a correspondente ferramenta, são apresentadas em detalhe na Seção 3. As Seções 4 e 5 introduzem brevemente outras ferramentas existentes no ambiente de suporte à metodologia e o ambiente de execução dos modelos, respectivamente. Conclusões e trabalhos futuros são apresentados na Seção 6.

2. Trabalhos relacionados

O projeto de hardware usando HDLs (Hardware Description Languages), como VHDL ou Verilog, com ou sem prototipação rápida através de FPGAs e placas especializadas, é usado na parte prática de disciplinas da área de Arquitetura de Computadores. Porém, devido à complexidade envolvida nestes ambientes, seu uso restringe-se a disciplinas mais avançadas, normalmente vinculadas ao nível de pós-graduação, ou então a experiências com processadores didáticos bastante simples.

[1] Doutorando no PPGC da UFRGS

As ADLs (Architectural Description Languages), como LISA [2] e EXPRESSION [3], permitem o projeto num nível de abstração mais alto e, geralmente, possuem recursos mais simples para a definição da estrutura (organização) e do comportamento (arquitetura) dos processadores do que as HDLs. O objetivo das ADLs é a rápida exploração do espaço de projeto. A partir da descrição do processador, são gerados, de forma automatizada, ferramentas como simulador, depurador, compilador e montador para testes com a arquitetura alvo. Não há, no entanto, relatos do uso de ADLs no ensino de Arquitetura de Computadores, apesar das suas características até mais interessantes que a das HDLs, possivelmente pela ausência de recursos didáticos interativos.

Ambientes de ensino são simuladores especializados para este fim. Alguns, como SimpleScalar [4] e RSIM [5], são softwares complexos que objetivam, inicialmente, a descrição detalhada do funcionamento de um dado processador para uso em pesquisa. Posteriormente, passaram a ser usados no ensino. A grande maioria dos simuladores, porém, possui um escopo menor, já tendo sido desenvolvidos com recursos especializados para o ensino de características selecionadas e mais relevantes dos processadores, visando a favorecer o aprendizado do aluno. Normalmente, estes ambientes descrevem um dado processador, muitas vezes com mais de um modo de execução (com e sem pipeline, por exemplo), e, eventualmente, o seu sistema de memória e de entrada e saída. Contam com interface gráfica, um número razoável de possibilidades de configuração e uma quantidade restrita de dados de saída da simulação. Aqui incluem-se os simuladores PCSpim [6], SATSim [7], WinDLX [8], DLXview [9], ESCAPE [10] e SPIECS [11]. O simulador HASE [12] encaixa-se nesta segunda categoria, porém ele é mais genérico, permitindo a criação de novos modelos de simulação de sistemas computacionais. Porém, o desenvolvimento de novos modelos de processadores, em HASE, prevê apenas a estrutura do processador e não o seu comportamento.

T&D-Bench2 é um ambiente para o desenvolvimento de modelos de simulação de processadores do estado da arte, que incluam pipelines e superescalaridade, a serem usados, preferencialmente, no ensino. Sua metodologia de modelagem, usando conceitos de orientação a objetos, permite o desenvolvimento rápido de modelos de novos processadores, descrevendo tanto sua estrutura como seu comportamento, a exemplo dos ambientes de projeto com ADLs. Possui, adicionalmente, um conjunto amplo de recursos visuais de interação a serem usados não apenas na experimentação do modelo, para a sua configuração e a visualização de resultados, a exemplo dos ambientes de ensino, mas, também, no desenvolvimento de novos modelos de processadores.

3. Metodologia de modelagem

A metodologia de modelagem de T&D-Bench2 é implementada por um ambiente de criação de modelos, composto de biblioteca e ferramenta de modelagem. A ferramenta de modelagem permite a criação de novos modelos de simulação de processadores do estado da arte usando as classes disponíveis na biblioteca. Os modelos desenvolvidos são, então, executados no ambiente de execução do T&D-Bench2.

3.1 Biblioteca

A biblioteca é um repositório de classes a serem usadas na criação de modelos de simulação de processadores do estado da arte. Tais classes são divididas em: funcionais, de controle e de visualização.

3.1.1 Classes funcionais. As classes funcionais correspondem aos diversos elementos encontrados no caminho de dados, ou bloco funcional, de um processador, como registradores, unidades lógicas e aritméticas, bancos de registradores, memórias, multiplexadores, entre outros. Estas classes são usadas, posteriormente, na ferramenta de modelagem para a composição do caminho de dados do processador. Há tantas classes quantos forem os elementos previstos para esta composição.

A criação de uma classe funcional constitui-se na definição de seus atributos, entradas, saídas e comportamento. Atributos, entradas e saídas são representados por dados da classe e o comportamento por um ou mais métodos. Atributos são variáveis que armazenam valores que descrevem um objeto daquela classe. Entradas podem representar dados de entrada propriamente ditos ou dados de controle para o elemento. Os métodos do comportamento correspondem às diversas funções que podem ser executadas pelo elemento.

A classe unidade lógica e aritmética é descrita da seguinte forma:

- Atributos: string nome;
- Entradas: inteiros E1, E2, OP;
- Saídas: inteiro S;
- Comportamento: métodos set_alu_E1, set_alu_E2, get_alu_S, set_alu_OP e operate_alu, com as funções de colocar valores nas entradas E1 e E2, obter valor da saída S, controlar qual operação a executar, e executar a operação sobre as entradas gerando o resultado, respectivamente. Os métodos *sets* e *gets* são auxiliares, não estando relacionados ao comportamento do elemento (eles são usados na classe *barramento* e, também, para a entrada interativa de dados).

A classe *barramento* é uma classe funcional distinta. Ao contrário das demais, que serão instanciadas uma ou poucas vezes na ferramenta de modelagem, esta será instanciada sempre que houver uma conexão de dados entre um elemento e outro.

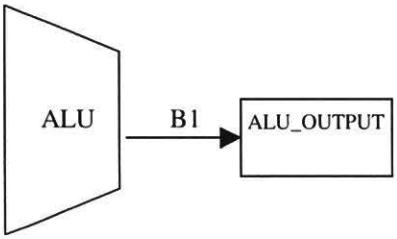


Figura 1. Barramento entre uma unidade lógica e aritmética e um registrador

Uma instância B1 da classe *barramento*, entre a saída da unidade lógica e aritmética ALU e a entrada do registrador ALU_OUTPUT (Figura 1), seria descrita pela identificação dos elementos que conecta e a especificação de quais saídas e entradas destes elementos serão usadas, no caso dos mesmos terem mais de uma entrada ou saída (no exemplo, há apenas uma saída, da ALU, e uma entrada, do ALU_OUTPUT). E esta instância utilizaria o método da classe que executa o envio do dado da saída de um elemento para a entrada do outro, invocando os respectivos métodos auxiliares *sets* e *gets* das classes destes elementos, sempre após a ativação do elemento anterior no caminho (a ALU no exemplo).

A relação entre classes de elementos do caminho de dados e a classe *barramento* é de 1 para n, com as primeiras possuindo um ponteiro para que suas instâncias identifiquem as instâncias de barramentos que estão ligadas às suas saídas.

Por fim, existe a classe funcional *datapath*, cuja única instância em tempo de execução serve para agregar as instâncias de elementos e de barramentos (Figura 2), bem como para fazer a comunicação com o bloco de controle. O seu uso, porém, é transparente para o usuário e será melhor detalhado adiante.

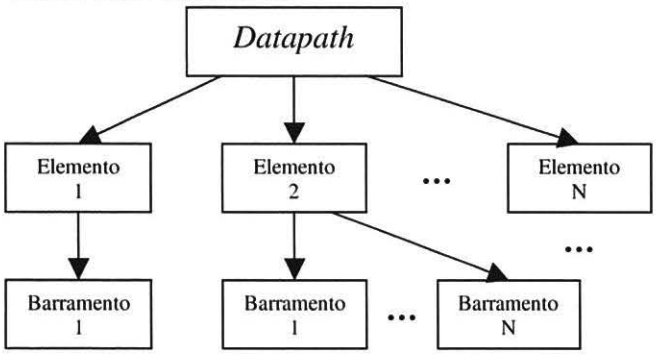


Figura 2. Diagrama de instâncias mostrando a relação entre as classes funcionais

A biblioteca deve ser bem completa com relação às classes funcionais típicas de um processador. Mas serão oferecidos recursos para a criação de novas classes, a partir da especialização ou modificação de outras classes já existentes, usando as facilidades da orientação a objetos. Ao contrário das classes de controle, descritas na próxima seção, classes funcionais não são alteradas na ferramenta de modelagem.

3.1.2 Classes de controle. As classes de controle são usadas para a definição do bloco de controle do processador, sendo de três tipos: *controle principal*, *estágio de pipeline* e *instrução* (Figura 3).

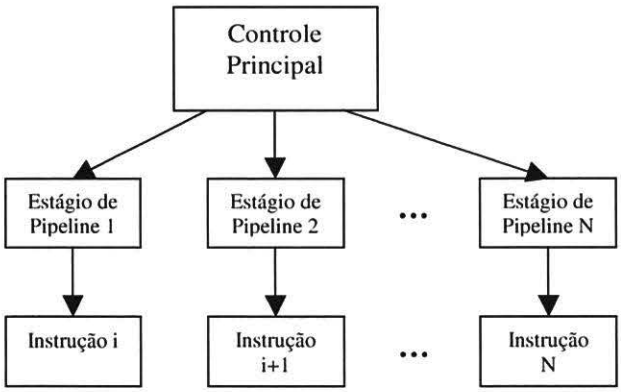


Figura 3. Diagrama de instâncias mostrando a relação entre as classes de controle

A classe *controle principal* mantém informações sobre o processador que vai ser modelado, como nome e velocidade (atributos do processador), bem como sobre a estrutura do pipeline e a instância da classe *datapath* que agrega os elementos e barramentos do caminho de dados. Além destes dados, esta classe possui, no mínimo, um método *behavior* a ser preenchido na ferramenta de modelagem e que será o responsável pela simulação do processador.

A classe *estágio de pipeline* mantém variáveis que descrevem um objeto desta classe, como o nome (atributo) e a identificação da instância de instrução que está sendo executada naquele estágio num dado instante de tempo. Esta classe conta, também, com, no mínimo, um método *behavior* usado para disparar a execução da instrução. Tal método pode ser customizado na ferramenta de modelagem.

A classe *instrução* possui atributos, como o nome, e um conjunto de variáveis para representar os diferentes campos de uma instrução. Estas variáveis poderão ser usadas, ou excluídas posteriormente, na ferramenta de modelagem, conforme o número de campos existente na instrução modelada. Na descrição da classe (ver abaixo), há três variáveis para identificar os registradores fonte da instrução. Se ela usa apenas dois, uma será excluída, pois não é necessária. A classe

instrução inclui um método *behavior*, no mínimo, a ser preenchido durante a modelagem do processador. Este método indica os elementos do caminho de dados a serem ativados para a execução da função da instrução.

```
class instruction {
    string nome;
    int opcode;     // opcode
                    // registradores fonte
    int rs1, rs2, rs3;
                     // registradores destino
    int rd1, rd2, rd3;
    int imm1, imm2, imm3; // imediatos
    int aux1, aux2, aux3; // auxiliares

    int behavior ( ) {
    ...
    }
}
```

Na ferramenta de modelagem, é descrita uma classe para cada tipo de instrução do processador. Instruções são de mesmo tipo quando possuem os mesmos campos e têm o seu comportamento diferenciado apenas pelos valores nestes campos. Por exemplo, o que diferencia uma instrução ADD de uma SUB pode ser apenas o código de operação a ser fornecido para a unidade lógica e aritmética. Os elementos do caminho de dados a serem ativados são os mesmos. Este código de operação é, então, fornecido por uma variável da classe.

As classes de controle terão o seu código definitivo apenas na ferramenta de modelagem. Por exemplo, a definição do método *behavior* de uma classe *instrução* vai depender da descrição do caminho de dados do processador a ser feita na etapa de modelagem, e, também, das informações sobre os estágios de pipeline deste, o que é, igualmente, importante para o método *behavior* da classe *controle principal*.

3.1.3 Classes de visualização. As classes de visualização são usadas na criação do modelo de simulação, que é feita de forma visual e interativa, bem como na interação com este para a entrada de dados e apresentação de resultados. Estas classes são de três tipos: de representação, de interface, e de apresentação de resultados.

A classe de representação permite a sua instanciação para armazenar os dados de primitivas 2D, que irão representar, graficamente, cada classe funcional (elementos e barramento), numa relação 1:1, através de símbolos gráficos classicamente adotados em livros-texto da área. Estes símbolos serão usados na ferramenta de modelagem para a criação do caminho de dados do novo processador de forma visual e interativa. Esta mesma representação visual do diagrama de blocos do caminho de dados será utilizada durante a execução do modelo.

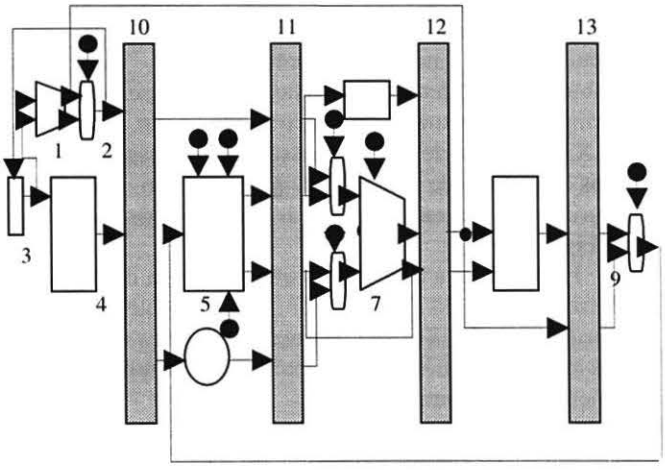
As classes de interface mantém relação de 1:1 com as classes funcionais (elementos e barramento) e com as classes de controle, servindo para a configuração destas e para a entrada interativa de dados. Para exemplificar, elas podem ser usadas para alterar o valor de um registrador, ou para mudar os registradores usados por uma instrução.

Finalmente, as classes de apresentação de resultados têm uma relação de n:1 com as classes funcionais. Elas objetivam apresentar os dados relativos aos diversos componentes do processador das mais diferentes formas. Podem apresentar as entradas e as saídas de uma unidade lógica e aritmética, os campos de uma instrução, ou mesmo a evolução das instruções nos estágios de pipeline. Classes de interface e de apresentação são formulários com componentes como botões, caixas de texto, listas, imagens, etc.

3.2 Ferramenta de modelagem

Através da ferramenta de modelagem do processador, o projetista define o caminho de dados do mesmo e, posteriormente, a funcionalidade do seu bloco de controle. Para isso, ele faz uso das classes disponíveis na biblioteca.

3.2.1 Definição do caminho de dados. O caminho de dados do novo processador é criado através de instâncias de classes de representação, conforme descrição acima. Escolhendo, posicionando e conectando tais objetos, o projetista cria o caminho de dados do processador. A Figura 4 mostra o caminho de dados do processador hipotético DLX, descrito em [13]. Este mesmo exemplo será utilizado no restante deste artigo para ilustrar os recursos do ambiente.




```
Legenda:
1. ADD        2. MUXPC     3.PC         4.IMEMORY
5.REGISTERS   6. MUXA      7. MUXB      8. ALU
9. MUXWB      10. IF/ID    11. ID/EX    12. EX/MEM
13. MEM/WB
```

Figura 4. Caminho de dados do DLX

3.2.2 Definição do controle. A definição do bloco de controle do processador é o próximo passo. Inicialmente, o projetista trabalha com a classe *controle principal*, adicionando novos dados e atribuindo valores aos atributos da classe. Ele pode, por exemplo, inserir uma variável para contabilizar o número de *taken branches* ou *untaken branches* executados (depois, ele deverá inserir, no método *behavior* da classe, o código para esta contabilização), e atribuir o nome "DLX" e a velocidade 500MHz aos atributos existentes para a criação da instância DLX da classe *controle principal*.

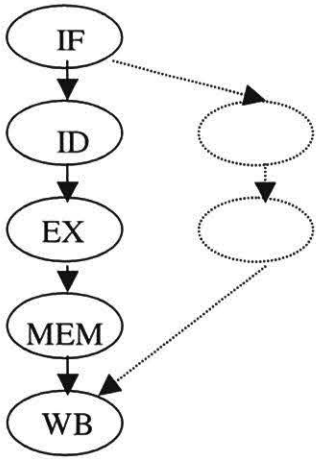


Figura 5. Grafo descrevendo os estágios do pipeline

A seguir, com recursos disponíveis na ferramenta de modelagem, ele desenha um grafo acíclico direcionado (Figura 5) que irá representar os estágios de pipeline existentes neste processador. Adicionalmente, ele pode adicionar novos dados à classe *estágio de pipeline*, assim como deve, necessariamente, atribuir valores aos atributos das diferentes instâncias da classe, como, por exemplo, os nomes dos cinco estágios do processador DLX. No caso de um processador superescalar, um ou mais estágios de diferentes pipelines do processador estariam no mesmo nível da árvore (desenho pontilhado na Figura 5).

Finalmente, a tarefa mais trabalhosa é a definição das classes que irão representar os diferentes tipos de instruções do DLX. Uma instrução aritmética registrador a registrador do DLX (tipo R), por exemplo, tem os campos indicados na Figura 6.

opcode	rs1	rs2	rd	...	func

Figura 6. Formato de instrução aritmética do DLX

Sendo assim, o código da classe *instrução*, para este tipo de instrução, seria customizado da seguinte maneira:

```
class instruction {
    string nome;
    int opcode;     // opcode
                    // registradores fonte
    int rs1, rs2;
                    // registrador destino
    int rd1;
    int aux1        // func

    int behavior ( ) {
    ...
    }
}
```

Resta especificar o método *behavior* da classe, que define a função desempenhada por aquele tipo de instrução. Isto é feito interativamente a partir do desenho do caminho de dados do processador. O projetista tem que seguir a seqüência dos elementos a serem ativados para esta classe *instrução*, respeitando as conexões entre eles, e escolher o método a ser executado pelo elemento, assim como a ativação de entradas de controle. Tome-se como exemplo uma leitura no banco de registradores REGISTERS. Os métodos a serem ativados seriam:

```
set_REGISTERS_E1 (rs1);
set_REGISTERS_E2 (rs2);
read_REGISTERS ( );
```

Uma informação adicional que se faz necessária é: em qual estágio de pipeline dar-se-á a ativação daquele elemento do caminho de dados, quando da execução da instrução. No final, uma instrução aritmética registrador a registrador ficaria com o método *behavior* especificado da seguinte forma:

```
int behavior ( ) {
switch ( estagio) {
  case IF:
    read_PC ( );
    read_IMEMORY ( );
    write_IF/ID.IR ( );
    add_ADD ( );
    write_IF/ID.NPC ( );
  case ID:
    read_IF/ID.IR ( );
    /* Argumentos das funções set são
provenientes da instância de instrução */
    set_REGISTERS_E1 (rs1);
    set_REGISTERS_E2 (rs2);
    read_REGISTERS ( );
    write_ID/EX.S1ULA ( );
    write_ID/EX.S2ULA ( );
  case EXEC:
    read_ID/EX.S1ULA ( );
    read_ID/EX.S2ULA ( );
    /* SOURCE_ULA_REGISTERS seleciona, como
entradas da ULA aquelas provenientes do banco
de registradores */
    set_MUXA_SEL (SOURCE_ULA_REGISTERS);
    activate_MUXA ( );
```

```
        set_MUXB_SEL (SOURCE_ULA_REGISTERS);
        activate_MUXB ( );
        set_ALU_OP (aux1)
        operate_ALU ( );
        write_EX/MEM.ALUOUTPUT ( );
      case MEM:
        read_EX/MEM.ALUOUTPUT ( );
        write_MEM/WB.ALUOUTPUT ( );
        /* NOBRANCH informa que o novo PC é o
anterior + 4 */
        set_MUXPC_SEL (NOBRANCH);
        activate_MUXPC ( );
      case WB:
        read_MEM/WB.ALUOUTPUT ( );
        /* SOURCE_ULA seleciona, como entrada do
banco de registradores, aquela proveniente da ULA
*/
        set_MUXWB_SEL (SOURCE_ULA);
         activate_MUXWB ( );
         set_REGISTERS_E3 (rd1);
         write_REGISTERS ( );
        }
    }
```

Os métodos que ativam elementos (cujos nomes contêm os nomes dos elementos que aparecem na Figura 4) não possuem argumentos, pois suas entradas são informadas a partir da ativação das instâncias de classes *barramento* provenientes do elemento anterior. As entradas da ULA, no exemplo, são informadas pelos barramentos que saem do registrador ID/EX.

Os elementos de memória, como registradores, bancos de registradores, memória de dados e/ou de instruções, apenas ativam seus barramentos de saída quando executam métodos *read*.

Os métodos de ativação dos elementos correspondem à ativação do *clock* e, em conjunto com a divisão em estágios de pipeline, definem a temporização do processador.

Os métodos *sets* que aparecem no método *behavior* configuram as entradas de controle dos elementos (registrador a ser escrito no banco de registradores, por exemplo) e têm que aparecer no método por exigência do ambiente, que faz a consistência para saber se, quando um elemento for ativado, todos os seus controles foram informados. Outra consistência feita pelo ambiente é obrigar a ativação de todos os elementos que formam o caminho pelo qual passa a execução daquele tipo de instrução.

Após o detalhamento do controle principal, dos estágios de pipeline e das classes *instrução*, o ambiente gera, automaticamente, o seguinte método *behavior* para a classe *controle principal*:

```
int behavior ( ) {
  instruction i [5] = {NULL, NULL, NULL,
  NULL, NULL};

  for ( ; ;) {
        i [0] = IF.default (NULL);
        ID.default (i [1]);
        EX.default (i [2]);
        MEM.default (i [3]);
        WB.default (i [4]);
        Advance_time ( );
        i [1] = i [0];
        i [2] = i [1];
        i [3] = i [2];
        i [4] = i [3];
  }
}
```

Os métodos *behavior* das instâncias da classe *estágio de pipeline*, que também são gerados automaticamente pelo ambiente, limitam-se a executar o método *behavior* da instrução informada no argumento. O método *behavior* da instância IF, porém, terá um código adicional, fornecido pelo projetista, adaptado à forma como as instruções estão armazenadas no elemento *memória de instruções*. As possibilidades são muitas: as instruções estão em código binário, em código assembly ou como um array de structs; e poderão ser provenientes de programas-exemplo disponibilizados juntamente com o modelo ou informados, interativamente, pelo usuário. Este código deve, então, entender este formato e traduzi-lo para a criação de instâncias da classe *instrução* do tipo adequado (aritméticas registrador a registrador, *loads*, etc).

3.3 Alternativas de modelagem

A metodologia de modelagem de T&D-Bench2 é bastante poderosa e flexível. Como exemplo, uma versão monociclo do DLX poderia ser implementada com a definição de apenas um estágio de pipeline, enquanto uma versão multiciclo poderia ser implementada com o avanço do tempo (método advance_time) após a ativação do método *behavior* de cada estágio.

A detecção de hazards pode ser feita dentro do método *behavior* da classe *controle principal*, ou em métodos especialmente criados para isso, através de comparações entre os campos das instruções. Por exemplo, a expressão condicional *i [0].rs1 == i [1].rd || i [0].rs2 == i [1].rd* verifica se os registradores fonte da instrução no estágio IF são o mesmo registrador destino da instrução em ID. Se a expressão retornar verdadeiro, um tratamento deve ser executado. Tal tratamento pode ser feito através da alimentação dos estágios de pipeline com instruções NOP, o que pressupõe uma classe de instruções deste tipo, e o armazenamento temporário da instrução dependente até que a outra instrução escreva no registrador destino. O código que segue ilustra este fato:

```
guarda = i [ 0]
/* Supondo 3 ciclos para a instrução
      escrever o resultado no reg. */
for ( j = 0; j <4; j ++) {
```

```
            i [0] = NOP;
            ID.default (i [1]);
            EX.default (i [2]);
            MEM.default (i [3]);
            WB.default (i [4]);
            Advance_time ( );
            i [1] = i [0];
            i [2] = i [1];
            i [3] = i [2];
            i [4] = i [3];
        }
        i [ 0] = guarda;
        /* Agora, volta a execução normal */
```

Uma deteção similar, com tratamento adequado, também pode ser feita para dependências causadas por instruções de desvio e para predições de desvio.

4. Ambiente de suporte à modelagem

Um ambiente computacional implementa recursos que dão suporte ao usuário para a criação de modelos de processadores segundo a metodologia apresentada na seção anterior. Além da ferramenta de modelagem, já descrita em detalhes, estarão disponíveis os seguintes recursos:

- editores de texto para o desenvolvimento das classes funcionais e classes de controle, assim como para a customização destas;
- ferramenta de consulta e acesso às classes existentes e sua hierarquia;
- acesso a compilador e link-editor, bem como a *templates* para o teste das classes desenvolvidas;
- editor de primitivas 2D, com facilidades para o armazenamento, consulta e associação destas primitivas a classes funcionais. Ele é usado, também, para o desenho do caminho de dados do processador e do seu pipeline. Em conjunto com a ferramenta de consulta às classes existentes, ele será ainda usado para a definição do método *behavior* de classes *instrução* mostrando os métodos disponíveis de cada elemento do caminho selecionado. O projetista seleciona o elemento no desenho e surgem, para escolha, os métodos disponíveis daquela classe. O método selecionado é inserido, então, automaticamente, no método *behavior* da instrução;
- editor para a criação e customização de formulários de interface, com facilidades para o armazenamento, consulta e associação destes formulários a classes funcionais ou de controle;
- gerador automático de código para as classes de controle.

Bastante importante na metodologia de modelagem é a consistência entre os projetos do bloco de controle e do caminho de dados do bloco funcional. Esta consistência é garantida automaticamente em tempo de modelagem. A criação do modelo deve seguir necessariamente a seguinte seqüência de passos:

- criação do caminho de dados;
- definição e customização do controle principal;
- definição e customização dos estágios de pipeline;
- definição e customização dos tipos de instrução.

Qualquer alteração nos dados de um destes passos, exige alterações, também, nos passos subseqüentes. Por exemplo, a inclusão de um novo elemento no caminho de dados pressupõe a sua conexão a elementos já existentes, tanto no que diz respeito às suas entradas quanto às suas saídas. Se estes elementos são ativados por uma classe *instrução*, o novo elemento, provavelmente, também o será (através da invocação de um método da sua classe). Cabe ao projetista efetuar esta inclusão sob notificação do ambiente, que identificará a linha da inclusão no método *behavior* da classe *instrução* e permitirá a seleção de um método daquela classe de elemento adicionado. No caso da exclusão de um elemento, alguns barramentos ficarão sem destino ou sem fonte. Cabe ao projetista rearranjá-los. Nos métodos *behavior* de classes *instrução*, o método de ativação deste componente será removido automaticamente pelo ambiente. Em resumo, não é possível realizar transferências no bloco de controle que não correspondam a conexões no *datapath*, em função do uso dos barramentos na transferência dos valores entre os elementos conectados.

5. Ambiente de execução

O ambiente de execução é responsável pela simulação do modelo criado no ambiente de modelagem. A partir das informações geradas pela ferramenta de modelagem, o ambiente de execução roda o modelo percorrendo as seguintes etapas:

1. Criação das instâncias de elementos do caminho de dados;
2. Criação das instâncias de barramentos;
3. Adição da informação das instâncias de barramentos conectadas às saídas das instâncias de elementos nestas últimas;
4. Criação da instância *datapath* e adição dos elementos criados nos passos 1 e 2 a esta instância;
5. Criação da instância do controle principal, o que envolve, internamente, a criação, também, das instâncias de estágios de pipeline;
6. Execução do método *behavior* da classe controle principal iniciando a simulação;
7. Instâncias de instruções serão criadas e destruídas conforme a execução do modelo avança.

A ligação entre bloco de controle e bloco funcional dá-se nas chamadas de métodos de objetos instrução. Estas chamadas, na verdade, não invocam diretamente métodos das classes funcionais, mas montam e enviam mensagens para o objeto *datapath*. Este, por sua vez, recebe a mensagem, decodifica-a e chama o objeto

elemento do caminho de dados respectivo para a execução da função. Como dito anteriormente, após a ativação de um elemento, ele propaga os seus valores de saída usando as instâncias da classe *barramento* adicionado a este objeto no passo 3 acima.

Finalmente, este modelo sendo executado insere-se numa plataforma de execução cliente-servidor, no lado servidor. Nos clientes, existirão objetos instanciados das classes de visualização que trocarão dados com o modelo a partir do comando do usuário. A arquitetura de software usada é a mesma descrita em [1].

7. Conclusões e trabalhos futuros

Este artigo descreveu um ambiente para o desenvolvimento de modelos de simulação de processadores do estado da arte a serem usados, preferencialmente, no ensino, que concilia a flexibilidade existente em ambientes de projeto de processadores, usando HDLs ou ADLs, com os recursos especializados de interação com o modelo, tanto na sua criação como na sua execução, existente em ambientes de ensino.

Incorporando os pontos fortes destas duas classes de ambientes, T&D-Bench2 será uma plataforma valiosa para o ensino de conceitos de processadores do estado da arte. E, devido à sua metodologia de modelagem baseada em conceitos de orientação a objetos e implementada usando recursos visuais interativos, poderá ser usado, também, na avaliação de alternativas de projeto de processadores.

Já tendo sido bem definida a metodologia de modelagem, o projeto T&D-Bench2 encontra-se agora na fase de implementação do ambiente de suporte à mesma e do ambiente de execução.

Referências

[1] D.Becker, J.C.Otero, F.R.Wagner. "**T&D-Bench: An Environment for Modeling and Simulating Complex Processor Architectures**". Submetido ao SBAC-PAD'2002.

[2] V. Zivojnovic, S. Pees, H. Meyr. "**LISA – machine description language and generic model for HW/SW co-design**". In Proceedings of the IEEE Workshop on VLSI Signal Processing, San Francisco, Oct. 1996.

[3] P. Grun, A. Halambi, A. Khare, V. Ganesh, N. Dutt, A. Nicolau. "**EXPRESSION: An ADL for System Level Design Exploration**". Technical Report #98-29, Setembro de 1998. Disponível por www em http://www.ics.uci.edu/~aces/

[4] AUSTIN, Todd M.. **A User's and Hacher's Guide to the SimpleScalar Architectural Research Tool Set (for tool set release 2.0)**. Disponível por WWW em ftp://ftp.cs.wisc.edu/sohi/Code/simplescalar2.0/hack_guide.pdf (28 dez. 1999)

[5] PAI, Vijay S.; PARTHASARATHY, Ranganathan; ADVE, Sarita V.. **RSIM: An Execution-Driven Simulator for ILP-Bases Shared-Memory Multiprocessors and Uniprocessors**. IEEE Computer Society Technical Comittee on Computer Architecture Newsletter, p. 32-38, Set. 1997.

[6] LARUS, James R..**SPIM S20: A MIPS R2000 Simulator**. Disponível por WWW em ftp://ftp.cs.wisc.edu/pub/spim/spim_documentation.ps (26 dez. 1999)

[7] WOLFF, Mark; WILLS, Linda. **SATSim: A Superscalar Architecture Trace Simulator Using Interactive Animation**. IEEE Computer Society Technical Comittee on Computer Architecture Newsletter, p. 23-26, Set. 2000.

[8] GRÜNBACHER, Herbert. **WinDLX Tutorial – A first example**. Disponível por E-mail em maziar@vlsivie.tuwien.ac.at

[9] ADAMS, George. **DLXview – (Preliminary) User's Manual**. Disponível por WWW em http://yara.ecn.purdue.edu/~teamaaa/dlxview (28 dez. 1999)

[10] VERPLAETSE, Peter. **ESCAPE v1.1 Manual**. Disponível por WWW em http://www.elis.rug.ac.be/escape (27 dez. 1999)

[11] DJORDJEVIC, Jovan; MILENKOVIC, Aleksandar; GRBANOVIC, Nenad. **An Integrated Environment for Teaching Computer Architecture**. IEEE Micro, p. 66-74, May-June 2000.

[12] IBBETT, Roland N.. **Hase DLX Simulation Model**. IEEE Micro, p. 57-65, May-June 2000.

[13] PATTERSON, David A; HENNESSY, John L.. **Computer Architecture A Quantitative Approach**. Morgan Kaufmann Publishers, Inc, 1996. 760p.