UNIVERSIDADE ESTADUAL DE FEIRA DE SANTANA

DEPARTAMENTO DE TECNOLOGIA

TEC499 SISTEMAS DIGITAIS

RELATÓRIO TÉCNICO

Thiago Sampaio Lima, Matheus Moura Batista, Joacy Mesquita

Tutor: João Carlos Nunes Bittencourt

1 INTRODUÇÃO

Atualmente existem diversos modelos de arquitetura de processadores disponíveis ao redor do mundo, cada um com suas especificidades e caracterísicas próprias, o que permite a cada arquitetura se adequar melhor a determinadas situações e/ou necessidades. Esta característica pode ser observada, por exemplo, na arquitetura IA-32 da intel (utilizada principalmente em processadores de propósito geral) e nas arquiteturas ARM (utilizadas principalmente em sistemas embarcados).

Devido às especficidades apresentadas pelo grupo Xing-Ling (financiador do projeto físico), foi preciso criar uma arquitetura específica que desse suporte às características requisitadas. Dentre estas, estão presentes características que dizem respeito ao suporte de instruções específicas (ISA), ao desempenho e pipeline e ao tratamento de conflitos (hazards).

O projeto conta com três estágios funcionais (IFD, EXM, WB) com detecção e tratamento de conflitos de dados. Tais conflitos são tratados com a técnica de forwarding e em nenhuma situação é necessário a inserção de bolhas no pipeline. O projeto foi desenvolvido de forma que não exista conflitos de controle. O processador foi desenvolvido na linguagem de descrição de hardware Verilog bem como os testes unitários e de integração. O modelo de memória utilizado para simulação foi fornecido por terceiros e foi feito na linguagem de descrição vhdl.

As próximas sessões apresentam uma visão geral do projeto em questão e são organizadas da seguinte forma: primeiro será apresentada uma visão geral dos passos e das técnicas utilizadas para a descrição funcional do processador, posteriormente é mostrado os principais módulos funcionais do processador, apresentando suas descrições e intercoexões e, por fim, é apresentado uma visão geral das simulações realizadas juntamente com a definição da cobertura dos testes e os resultados obtidos com os mesmos.

2 FUNDAMENTAÇÃO TEÓRICA

2.0.1 Microarquitetura de Processadores

A microarquitetura de um processador é definida como a forma de implementar o Conjunto de Instruções da Arquitetura (nível ISA), ou seja, a definição do caminho de dados e do controle

Intel	
Microarquitetura	**Processador**
P5	Pentium; Pentium MMX
P6	Pentium Pro; Pentium II; Pentium II Xeon; Pentium III; Pentium III Xeon
NetBurst	Pentium 4; Xeon; Pentium D
Core	Core 2 Duo; Core 2 Quad; Core 2 Extreme
Nehalem	Core i7; Core i7 Extreme
AMD	
Microarquitetura	**Processador**
K5	AMD K5
K6	AMD K6; K6-2; K6-III
K7	Athlon; Athlon XP; Duron
K8	Athlon 64; Athlon FX; Athlon X2; Sempron
K10	Phenom

Tabela 1. Exemplo de microarquiteturas Intel e AMD.

para o funcionamento correto do circuito. Ela pode ser apresentada por meio de um diagrama de blocos, composto pelos módulos funcionais do processador, e a forma como se interligam entre si e com a unidade de controle do mesmo. Pode ser afirmado que uma arquitetura pode gerar diversas microarquiteturas, assim como diferentes implementações do circuito do processador podem existir para a mesma microarquitetura (HENNESSY; PATTERSON, 2005).

Podemos entender a microaquitetura como o modo com que se usa a arquitetura para melhorar a velocidade e desempenho de processamento, dentro de uma família de processadores, a microarquitetura pode ser atualizada frequentemente, a fim de melhorar o processamento. A tabela abaixo mostra como a mesma microarquitetura pode ser implementada de modos diferentes, a tabela se refere a processadores *Intel* (INTEL, 2016) e *AMD* (AMD, 2016).

Quando se está definindo a microarquitetura esfecifica-se o modo de operação do processador (monociclo ou multiciclo), além da forma pela qual a unidade de controle será implementada. De acordo com (HENNESSY; PATTERSON, 2005), existem formas distintas de projetar a unidade de controle do processador, para o caminho de dados de ciclo único podem ser utilizadas um conjunto de tabelas verdade que especificam a definição dos sinais de controle de acordo com o tipo de instrução, já para o caminho de dados multiciclo o controle precisa especificar os sinais a

serem definidos em qualquer etapa e também a próxima etapa na sequência. Podem ser destacadas também as técnicas baseada no uso de máquinas de estado finito, e a microprogramação.

2.0.2 Pipeline

A idéia de um Pipeline é a de dividir o ciclo da execução da instrução em estágios consecutivos, como uma linha de montagem, onde uma nova entrada é aceita em uma extremidade, antes que entradas aceitas anteriormente apareçam como saídas na outra extremidade (STALLINGS, 2002). Desta forma, consegue-se acelerar a execução das instruções, resultando em uma redução do tempo de execução e por consequência o aumento do desempenho. Em teoria, quanto maior a quantidade de estágios do pipeline, maior o desempenho do processador, na prática, é observado que o ganho com o pipeline é dimimuído por conta do aumento do custo (FERLIN, 2004).

Tomando como base o processador MIPS, identificamos o ciclo de execução de uma instrução em cinco etapas:

1. Buscar a instrução na memória.

2. Decodificar da instrução e ler registradores.

3. Executar a operação ou calcular um endereço.

4. Acessar um operando na memória de dados.

5. Escrever o resultado em um registrador.

Itentificamos, portanto, cinco estágios no pipeline desse processador (HENNESSY; PATTERSON, 2005). Em um processador monociclo, a execução de três instruções durariam 15 ciclos de clock, a imagem abaixo mostra como a aplicação do pipeline faz com que as mesmas intruções sejam executadas em 7 ciclos de clock, causando redução significativa no tempo gasto para tal.

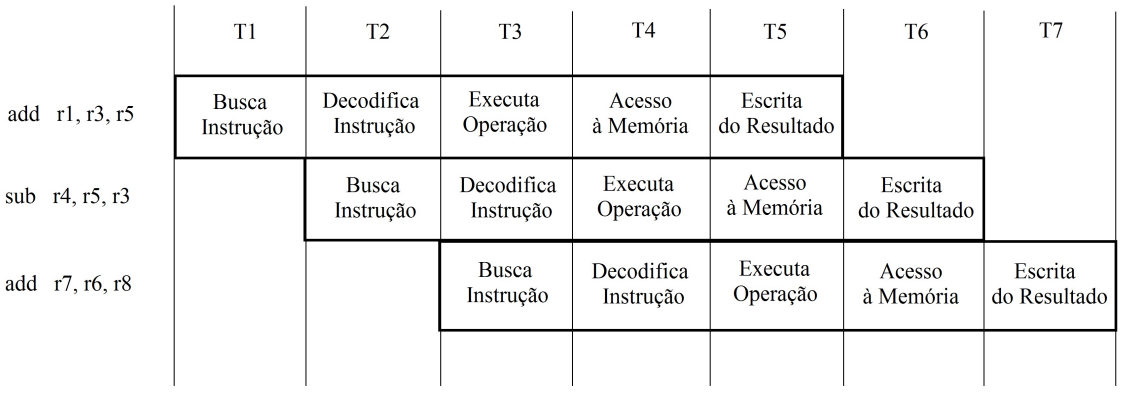

Figura 1. Exemplo de execução de instruções com pipeline.

2.0.2.1 Conflitos no Pipeline

Apesar da melhora de desmpenho com o pipeline, em certas situações não é possível que a execução da próxima instrução aconteça no ciclo de clock seguinte, nessas situações há a ocorrência dos *hazards* ou conflitos. Existem três tipos *hazards*: os estruturais, os de dados e os de controle.

Os conflitos estruturais caracterizam uma falta de recursos de *hardware* para suportar a combinação de instruções a serem executadas no mesmo ciclo, pode ser citado como exemplo um processador que possui uma memória única que armazene instruções e dados, a solução para esse conflito é a utilização de duas memórias, uma para instruções e outra para dados.

Os conflitos de dados ocorrem quando a instrução a ser executada necessita de algum dado ainda não disponível, por exemplo, quando o registrador fonte de uma instrução aritmética é igual ao registrador destino da instrução anterior. No trecho de código abaixo temos uma instrução sub seguida por uma instrução add que usa a subtração (**$s0**):

```
sub $s0, $t1, $t2

add $t3, $s0, $t0
```

No caso do MIPS o resultado da subtração será escrito no quinto estágio, como a adição precisa do resultado no seu segundo estágio, três bolhas seriam inseridas ao pipeline. Para solucionar esse problema, parte-se do princípio de não ser necessário esperar o fim da execução da instrução para adquirir o dado, percebe-se que o resultado da subtração existe na saída da Unidade Lógico-Aritmética (ALU), a saída principal para os *hazards* de dados é fornecer o resultado da operação, saída da ALU, como entrada para a adição. Para que isso ocorra é preciso inserir um *hardware* extra para coletar o resultado antes do previsto, essa inserção é chamada de *forwarding* ou *bypassing*.

Os conflitos de controle ocorrem quando há a necessidade de tomar uma decisão para o valor do PC baseada nos resultados de uma instrução que ainda não foi concluída, ou seja, está ligado a instruções de desvio condicional. O problema se dá quando pipeline inicia a busca da instrução subsequente ao desvio no próximo ciclo de clock, no entanto não há como o pipeline saber qual é a instrução correta a ser buscada (não se sabe se o desvio deverá ou não ser tomado), uma vez que acabou de receber o próprio desvio da memória.

Para o tratamento dos *hazards* de controle, há certas possibilidades de abordagem, uma delas é esperar o pipeline determinar o resultado do desvio e que o endereço da próxima instrução esteja determinado, ou seja, causar *stall* (parada) no pipeline logo após buscarmos um desvio, para isso há a necessidade de *hardware* para o teste dos registradores, o cálculo do endereço de desvio e a atualização do PC no segundo estágio do pipeline.

Outra abordagem é a predição de desvios, que pode ser de dois tipos: estática e dinâmica. Na predição estática existem duas possibilidades de tratamento, a primeira corresponde a assumir que os desvios nunca serão tomados, a segunda prevê que se o endereço do desvio for anterior ao PC atual (o que acontece em laços de repetição) ele sempre será tomado, do contrário, se o endereço do desvio for posterior ao PC atual ele nunca será tomado.

Na predição dinâmica, o comportamento do desvio é analisado durante a execução do programa, um histórico do comportamento (tomado ou não tomado) de cada desvio é mantido e atualizado. Quando a predição é errada, a unidade de controle terá que garantir que as instruções após o desvio errado não sejam executadas, quando isso ocorre, o pipeline é reiniciado a partir do endereço de desvio apropriado. Quanto mais longo for o pipeline, maior o custo por uma previsão incorreta (HENNESSY; PATTERSON, 2005).

3 METODOLOGIA

Para dar início ao projeto foi feita a definição da arquitetura do sistema. Para tal, foi criado um documento de Arquitetura contendo todas as características pertinentes a esta etapa do processo. Estão inclusos neste documento a arquitetura do conjunto de instruções do processador (ISA), bem como uma visão geral dos blocos funcionais que foram criados posteriormente. Juntamente com o documento de arquitetura, foram desenvolvidos dois programas em linguagem C (Assembler e simulador) com o objetivo de dar suporte aos testes iniciais do sistema. No documento também é mostrado como compilar e executar tais programas para a realização de testes e simulações.

Logo após, deu-se início ao desenvolvimento da microarquitetura e da descrição funcional em Verilog. A microarquitetura foi definida na forma de um documento onde são descritos os principais blocos funcionais do projeto físico do sistema, suas respectivas entradas e saídas e suas interconexões com os demais componentes. Além disso, a fim de dar uma visão geral dos componentes desenvolvidos, O documento também apresenta um diagrama de blocos do caminho de dados (datapath) onde são mostrados todos os módulos e suas interligações. Já a descrição em Verilog foi feita, inicialmente, para os blocos mais simples do sistema como multiplexadores e ULA e, posteriormente, para os blocos mais complexos como a unidade de controle, banco de registradores e unidade de tratamento de conflitos. Testes e correções de erros foram feitos durante todo o processo de descrição funcional.

Tanto a microarquitetura quanto a descrição foram feitas considerando duas etapas de desenvolvimento, ou seja, foram desenvolvidos um documento e um conjunto de código-fonte para a primeira etapa e depois estes foram incrementados e corrigidos para atender às necessidades segunda etapa.

Para a primeira etapa, o processador foi desenvolvido considerando a execução de uma instrução por ciclo de clock e sem a utilização de pipeline. Esta etapa teve foco na descrição funcional dos módulos básicos do processador, na integração entre eles e nas simulações de códigos assembly. As questões que envolvem desempenho e tratamento de conflitos (que surgem com o pipeline) não são tratadas nesta etapa do desenvolvimento.

Já na segunda etapa, o projeto foi feito utilizando a técnica de pipeline juntamente com o tratamento de conflitos. O pipeline foi feito com três estágios funcionais e, devido a isso, foi necessário adicionar dois registradores de estágio ao projeto do sistema. Tais registradores são úteis no sentido de manter as informações necessárias para os estágios seguintes do pipeline. No que diz respeito ao tratamento de conflitos, o processador conta com uma unidade de detecção

e tratamento que é capaz de verificar a ocorrência de conflitos de dados e corrigir a execução do pipeline utilizando-se da técnica de *forwarding* (adiantamento). Nesta versão do projeto, o processador foi desenvolvido com algumas características que diferem de alguns modelos clássicos como a arquitetura MIPS e ARM. Tais características se traduzem em dois principais benefícios, um do ponto de vista da complexidade de projeto e outro do ponto de vista do desempenho na execução em pipeline.

No que diz respeito à complexidade, devido à organização utilizada para o pipeline, o processador é capaz de identificar e tratar uma instrução de salto no mesmo ciclo de clock, permitindo que o endereço de salto seja calculado e enviado ao contador de programa no mesmo estágio do pipeline. Tal característica diminui a complexidade do projeto no sentido de não ser necessário a criação de módulos adicionais para tratar dos conflitos de controle que surgiriam caso esta operação fosse realizada em diversos estágios.

Em relação ao desempenho do pipeline, a escolha da quantidade de estágios juntamente com sua organização (IFD, EXM, WB), proporcionam uma execução em pipeline sem interrupções (bolhas). Isto é possível devido ao fato de que os dados nunca ficam presos em determinada operação como, por exemplo, conflitos por depêndencia de memória (*load-use*). O trecho de código a seguir exemplifica esta situação.

```
load r1, r0
add r2, r1, r0
```

Figura 2. Exemplo de código em que poderia ocorrer bolhas no pipeline.

O exemplo acima mostra um trecho de código que lê da memória um determinado valor e faz o endereço em *r0* ser somado a este valor. Em uma arquitetura clássica como a do processador MIPS, este trecho causaria bolha no pipeline devido ao fato do valor que será gravado em r1 pela primeira instrução ainda não ter sido lido da memória no momento que ele é requerido pela segunda instrução. Na arquitetura desenvolvida, a etapa de acesso à memória está distante apenas um estágio da busca de operandos e, com isso, somente a utilização da técnica de forwarding é suficiente para resolver todos os conflitos que surgirem.

A partir desta organização utilizada para o pipeline surge uma restrição de projeto no que diz respeito à velocidade do ciclo de clock. Esta restrição surge devido ao tamanho do caminho de dados no primeiro estágio do pipeline, pois, ao agrupar componentes para se evitar os conflitos de controle, o caminho combinacional que os sinais elétricos devem percorrer se torna maior. Com isso, a frequência de clock do sistema é diminuída a fim de que este estágio possa realizar suas operações sem problemas de sincronização.

Outra característica importante diz respeito à forma de construção da unidade de controle. Para este projeto, a unidade de controle foi descrita com o métodologia *hardwire*, que tem como principal vantagem a maior velocidade na decodificação de instruções. Sua principal desvantagem está na complexidade gerada no módulo. Porém, devido ao pequeno número de instruções utilizadas, a complexidade apresentada se torna satisfatória.

Para a realização de simulações é utilizado um modelo de memória com entrada e saída via arquivo de texto. Este modelo foi utilizado tanto para a memória de dados quanto para a memória de instruções e foi desenvolvido na linguagem de descrição de hardware VHDL. O módulo funcional foi fornecido por terceiros e pode ser utilizado somente para simulações pois não é sintetizável.

4 MÓDULOS FUNCIONAIS

A descrição funcional do processador, como mencionado na seção anterior, foi desenvolvida utilizando-se a técnica de pipeline e conta com três estágios para à execução das instruções. Cada um destes estágios é responsável por realizar uma parte do processo de execução de uma instrução e possui características e componentes específicos atrelados ao seu funcionamento. A descrição do funcionamento de cada estágio, juntamente com suas particularidades é feita nas subseções seguintes.

4.0.1 Busca e decodificação de instruções (IFD)

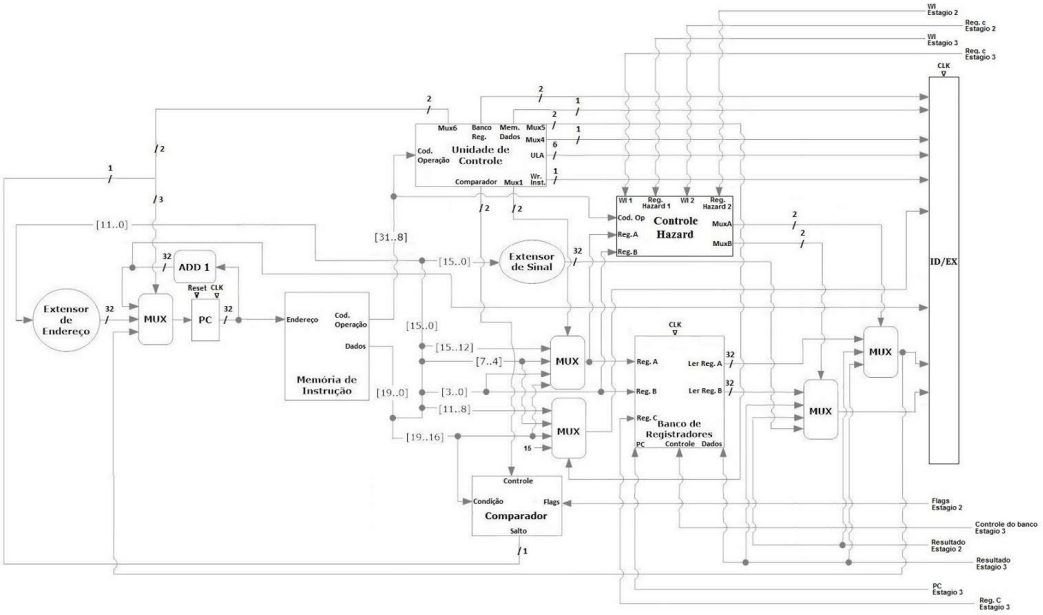

Figura 3. Caminho de dados do primeiro estágio do pipeline.

Este estágio é o responsável por ler e decodificar as instruções vindas da memória e, para tal, são utilizados os módulos de memória de instruções, contador de programa (PC), unidade de controle e banco de registradores.

Durante esta etapa, o contador de programa (PC) fornece à memória de instruções o endereço da instrução a ser buscada e este módulo torna-se responsável por buscar a instrução no endereço fornecido e libera-lá em sua saída. Após isso, a instrução segue por dois caminhos distintos. Uma parte dos bits da instrução são levados à unidade de controle para serem

decodificados e outra parte dos bits seguem para o banco de registradores a fim de realizar a busca dos operandos.

A unidade de controle, após a realização da decodificação, libera em suas saídas os sinais de controle de todos os módulos do processador que necessitam de algum tipo de gerenciamento. Já o banco de registradores utiliza a informação recebida para determinar qual registrador será lido e/ou escrito. Os demais sinais de controle seguem para o registrador de estágio para continuarem a ser propagados no pipeline até o momento em que serão necessários.

Neste estágio também estão localizados a unidade de conflitos e a unidade de comparação. A primeira é a responsável por detectar conflitos e realizar o *forwarding* dos dados. Já a segunda unidade é utilizada para determinar se um salto será ou não tomado e, para tanto, são necessários sinais da unidade de controle e das flags vindas da ULA.

4.0.2 Execução de instrução e acesso à memória (EXM)

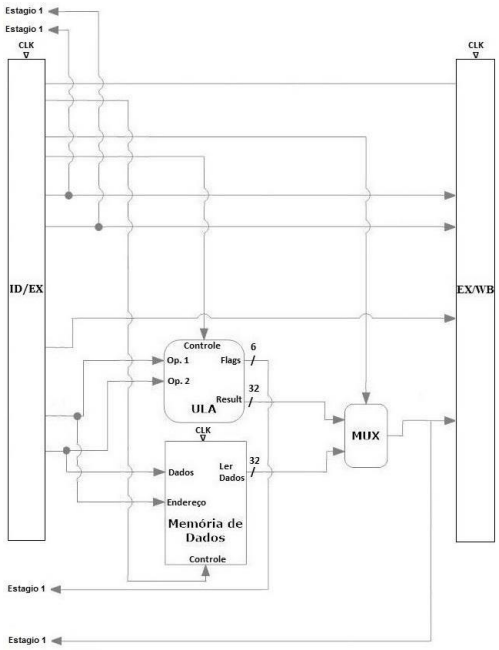

Figura 4. Caminho de dados do segundo estágio do pipeline.

O segundo estágio do pipeline tem como principal atividade executar a instrução lida no estágio anterior. Essa execução pode ser na forma de uma operação lógica-aritmética (que utilizará a unidade lógica e aritmética) ou na forma de acesso à memória (que usará a memória de dados do sistema).

Nesta fase, as insformações de dados e controle são recebidas através do registrador de estágio e seguem, simultâneamente, para a unidade lógica e aritmética (ULA) e para a memória de dados. O sinal de controle é o responsável por definir qual destes dois módulos será utilizado e como o será.

No caso de uma instrução de operação lógica ou aritmética, o sinal de controle irá fornecer

à ULA a operação que ela deve realizar com os operandos que estão em sua entrada. Durante a realização da operação também são definidos os valores das flags alteradas pela instrução. Após a realização da operação, o resultado e as flags são colocados em suas respectivas saídas do módulo.

Já no caso de uma instrução de acesso à memória, o sinal de controle irá indicar se a memória será utilizada no modo de leitura ou escrita de dados. O modo de leitura irá utilizar um endereço fornecido pela instrução a partir de onde será realizada a leitura de dados e o de escrita irá utilizar, além de um endereço de escrita, a informação que será gravada em tal endereço. As operações de escrita não apresentam nenhum tipo de saída enquanto que as de leitura apresentam a informação buscada na memória através da saída de dados da mesma.

Os sinais de controle que não foram utilizados até esta fase seguem para outro registrador de estágio onde continuarão a serem propagados no pipeline até que sejam necessários.

4.0.3 Escrita nos registradores (WB)

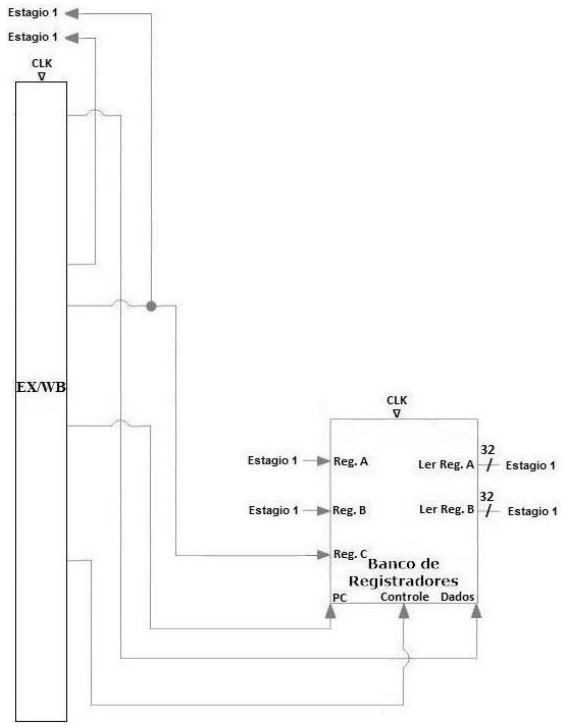

Figura 5. Caminho de dados do terceiro estágio do pipeline.

O terceiro e último estágio é o responsável por armazenar, em um determinado registrador, os dados provenientes de operações. Para tal, somente o banco de registradores é utilizado.

Neste estágio, assim como no segundo, as informações pertinentes são recebidas através de um registrador de estágio. Essas informações carregam o dado que será gravado no banco de registradores, o endereço do registrador de gravação e o sinal de controle indicando uma operação de escrita. A execução desta etapa é feita simplesmente colocando tais valores nas respectivas entradas do banco de registradores para que ele realize a operação. A habilitação da leitura de tais informações é feita somente no próximo ciclo de clock.

5 SIMULAÇÕES E RESULTADOS

Os testes realizados correspondem aos testes de unidade de cada módulo do processador, e aos testes de integração, onde o módulo correspondente à junção dos módulos individuais do processador foi testado.

A metodologia utilizada para a implementação dos teste de unidade foi a de testar os casos mais extremos. Dos testes de unidade tem destaque o da Unidade Lógica e Aritmética e da Unidade de Controle.

Para o teste da ULA, um módulo simula as entradas correspondentes a operandos e operações os resultados referentes aos testes. Em, um laço de repetição, foram atribuídos valores randômicos aos dois operandos e à operação a ser realizada, em cada iteração do teste foi verificada a validade do código das operações. No caso de ser uma operação válida, são impressos em um arquivo de texto os valores de entrada recebidos, e das saídas correspontes a eles.

Para o teste da Unidade de Controle, também foi feito um módulo que simula a entrada corresponte aos 27 bits mais significativos da instrução (o código do registrador destino não é considerado) e os sinais de controle gerados a partir dele. Em um laço de repetição, foram atribuídos valores aleatórios ao código de entrada, em cada iteração do teste foi verificada o tipo da instrução, lógico-aritmética, carregamento de constantes, acesso à memória e de desvio, em um arquivo de texto são impressos o código de entrada recebido, e as saídas correspontes aos sinais de controle gerado por cada tipo de instrução.

Dos testes gerados randômicamente para a ULA e a unidade de controle, 100% dos casos que geraram dados válidos apresentaram resultados corretos.

Os testes de integração realizados dizem respeito aos testes em linguagem *Assembly* utilizados nos problemas anteriores, foram eles o Selection Sort, o Binary Search, o Bubble Sort, o teste de Flags, o teste gerador de Números Primos, e o teste gerador da Sequência de Fibonacci. Em simulação no *ModelSim*, o processador se comportou de maneira adequada, e apresentou os resultados corretos para os seis testes.

Também foi feito um arquivo de teste *Assembly* adicional, com o intuito de testar a detecção e o tratamento de possíveis conflitos de dados no pipeline. Nesse teste são calculados os n triplos sucessivos do número 2, a primeira iteração triplica o 2, nas iterações posteriores, o número atual é triplicado, o resultado gerado corresponde a equação 5.1.

$$2 * 3^n \tag{5.1}$$

O número máximo de iterações é n=19, pois gera o maior valor capaz de ser representado com 32 bits, equação 5.2.

$$2 * 3^{19} \;=\; 2324522934_{10} \;=\; 8A8D67B6_{16} \tag{5.2}$$

Também nesse teste, o processador se comportou de maneira correta, apresentando os resultados adequados para ele.

REFERÊNCIAS

AMD. *AMD Desktop Processors*. 2016. <http://www.amd.com/en-us/products/processors/desktop>. [Online; accessado em 06 de Março de 2016]. Citado na página 2.

FERLIN, E. P. O avanço tecnológico dos processadores e sua utilização pelo software. *da Vinci, Curitiba*, v. 1, n. 1, p. 43–60, 2004. Citado na página 3.

HENNESSY, J. L.; PATTERSON, D. A. *Organização e Projeto de Computadores: A interface Hardware/Software*. Rio de Janeiro, RJ, Brasil: Elsevier, 2005. Citado 3 vezes nas páginas 2, 3 e 5.

INTEL. *Processadores para desktop*. 2016. <http://ark.intel.com/pt-br/?_ga=1.2139760.1309669859.1460221700>. [Online; accessado em 06 de Março de 2016]. Citado na página 2.

STALLINGS, W. *Arquitetura e organização de computadores*. São Paulo, SP, Brasil: Prentice Hall, 2002. Citado na página 3.