UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Organização de Computadores II
Trabalho Prático V – Implementação de um Processador Superescalar com *Pipeline* sobre o Caminho de Dados

Artur Henrique Marzano Gonzaga
Caio Felipe Zanatelli
Matheus Henrique de Souza
Tiago de Rezende Alves

Professor: Omar Paranaiba Vilela Neto
Monitor: Laysson Oliveira Luz

Belo Horizonte
05 de dezembro de 2017

Sumário

1 Introdução

Tendo em vista que nos trabalhos anteriores foram construídas uma Unidade Lógico-Aritmética (ALU), um Banco de Registradores e uma Unidade de Multiplicação, formando assim um processador superescalar com multiplicação paralela, este trabalho tem como objetivo principal a implementação de um *pipeline* sobre o caminho de dados desenvolvido. Para tanto, foi utilizada a linguagem de descrição de *hardware* Verilog e o pacote de desenvolvimento da Altera, composto pelo ambiente de simulação proporcionado pelos *softwares Quartus* e *ModelSim*, além da placa DE2-115, utilizada para a prototipação do *hardware* criado. Por fim, as especificações de cada unidade funcional, bem como as decisões de projeto relativas a cada uma delas, serão abordadas nas seções seguintes.

2 Implementação do *Pipeline*

Conforme solicitado na especificação deste trabalho, foi implementado um processador superescalar, com multiplicação paralela, que utiliza um *pipeline* sobre o caminho de dados, o qual possui três estágios em sua execução. O primeiro deles é referente à decodificação de instruções, o segundo estágio refere-se à execução da instrução e o último à escrita, no banco de registradores, do resultado obtido. Para tal, foi necessário refazer todo o módulo de controle obtido no trabalho prático anterior, no qual estava implementado uma máquina de estados finitos (FSM), uma vez que tal estratégia não dialogava com a estrutura do novo processador, alterando a estrutura multiciclo para um modelo em *pipeline*. Outro módulo refeito foi o módulo principal do processador – **hal_4.v**, que teve seus sinais e unidades devidamente divididos de acordo com os três estágios do *pipeline*.

2.1 Estratégias de Encaminhamento

De forma a evitar *stalls* no caminho de dados e com isso otimizar a *performance* do processador, foram adotadas algumas estratégias de encaminhamento (*forwarding*), que cobrem todas as possibilidades de encaminhamentos em todos os estágios do *pipeline* em questão. Para facilitar a esquematização do *pipeline*, considere a sigla ID para o estágio de decodificação de instruções (*Instruction Decode*), EX para o estágio de execução (*Execute*) e WB para o estágio de escrita do resultado no banco de registradores (*Write-Back*). A partir disso, a seguir são apresentadas todas as formas de encaminhamentos suportadas.

- **EX para ID**: encaminha o resultado da última instrução executada para a instrução seguinte que vai adentrar o estágio de decodificação (ID), de forma a disponibilizar o resultado antes de sua escrita no banco de registradores (WB), evitando assim *hazards* de dados.
- **EX para EX**: encaminha o resultado da última instrução executada para a instrução seguinte que vai adentrar o estágio de execução (EX), de forma a disponibilizar o resultado antes de sua escrita no banco de registradores (WB), evitando assim *hazards* de dados.
- **ALU/MULT para ID**: encaminha o resultado da última operação realizada pela ALU ou pela Unidade de Multiplicação ($LO/$HI) para o estágio de decodificação (ID), o que é realizado antes do estágio de execução (EX), dado que a ALU e a Unidade de Multiplicação são módulos assíncronos. Esse encaminhamento é um caso especial para evitar *stalls* em instruções de desvio condicional (*branches*), que será abordado mais detalhadamente na seção seguinte.

2.2 Evitando *Stalls* Relacionados a Instruções de Desvio

As instruções de desvio do fluxo de execução, a saber, **beq** e **jump**, demandaram modificações na estrutura do processador. A primeira delas é em relação ao valor do próximo PC, que antes, no processador multiciclo, era feito no estágio de escrita no banco de registradores e agora é realizado junto à decodificação das instruções. Aliado a tal alteração, as duas instruções de desvio são executadas na própria etapa de decodificação, sendo necessário adicionar um circuito de comparação para o **beq** e retirar tal operação da ALU. Para o **jump** não foi necessária nenhuma modificação adicional, pois bastou redirecionar os 12 *bits* menos significativos referentes ao imediato no multiplexador que seleciona o novo valor do PC.

Outra modificação que permitiu a completa ausência de *stalls* no processador foram os dois encaminhamentos, do estágio EX para o estágio ID e da ALU/MULT para o ID, explicados na subseção anterior. Tais encaminhamentos operam sobre os registradores a serem utilizados pela instrução *beq* e que, respectivamente, serão escritos por instruções que começaram dois ciclos antes e instruções imediatamente anteriores à instrução de desvio condicional.

3 Unidades Funcionais

Esta seção tem como objetivo apresentar cada unidade funcional desenvolvida, abordando suas especificações e decisões tomadas na fase de projeto. Neste contexto, de forma a possibilitar uma melhor compreensão acerca do circuito final obtido, a Figura 1 ilustra o diagrama esquemático resultante.

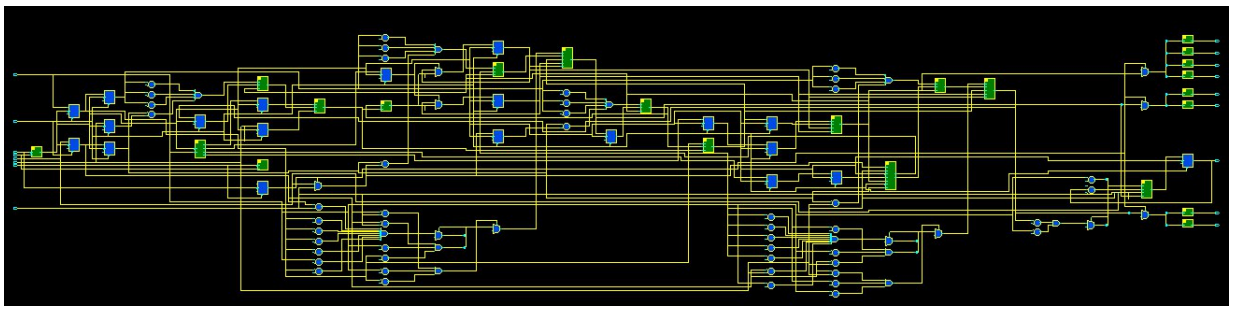

Figura 1 – Diagrama esquemático do Circuito.

Ainda, com o intuito de fornecer uma base acerca do funcionamento das instruções suportadas pelo processador desenvolvido, a Figura 2 ilustra o formato definido para tais instruções, o que inclui operações lógico/aritméticas, instruções de desvio (*jumps* e *branches*) e instruções de multiplicação, as quais são executadas por um *hardware* dedicado, como será abordado posteriormente.

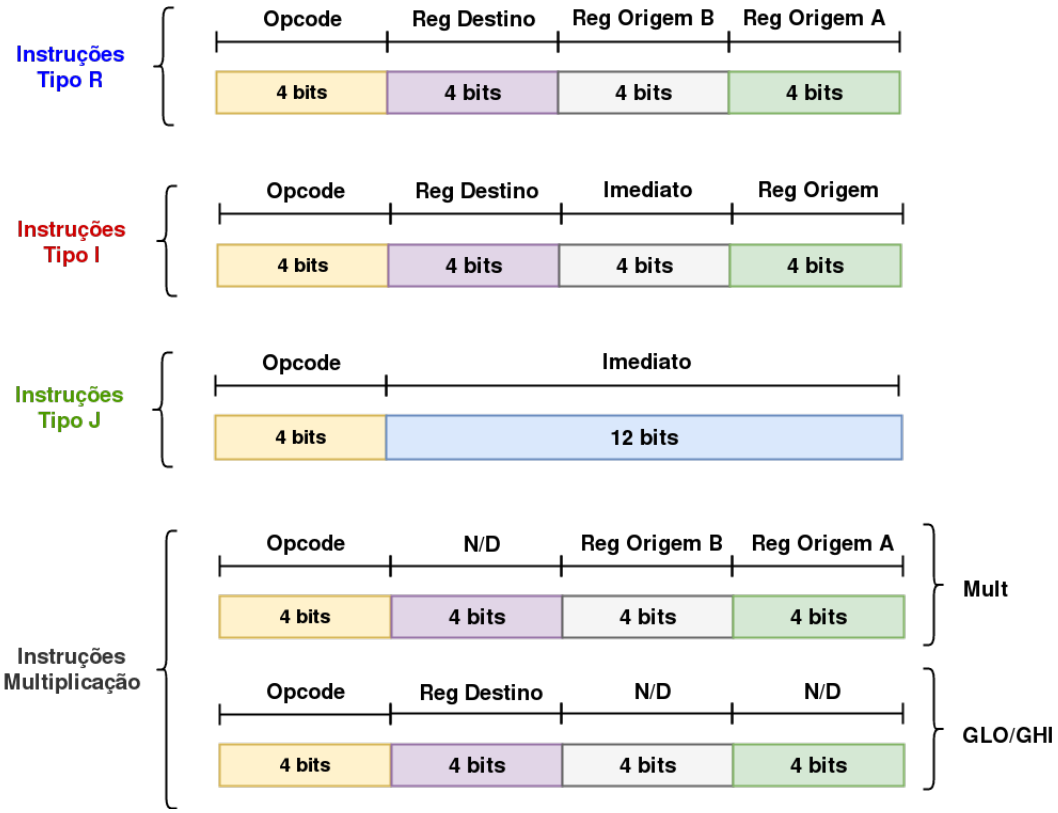

Figura 2 – Formato das instruções suportadas.

3.1 Banco de Registradores

O banco de registradores é a primeira unidade funcional que compõe este projeto. Sua finalidade é proporcionar o acesso rápido, por parte do processador, a valores endereçados pelo banco, uma vez que o acesso à memória é bem mais lento. Vale ressaltar, neste ponto, que, no contexto deste trabalho, o banco de registradores é composto por 16 registradores, com cada um contendo 16 *bits*. Com isso, tem-se que o endereçamento é realizado através de 4 *bits*, como abordado na sequência. Outro detalhe é que a escrita no banco de registradores se dá na borda de descida do sinal de clock. Por fim, é importante observar, ainda, que o banco em questão também suporta números negativos, uma vez que os valores são armazenados como complemento de dois.

3.1.1 Sinais de Entrada e Saída

De modo a garantir o funcionamento correto do módulo, é necessária a utilização de alguns sinais de entrada e saída. Os sinais em questão são ilustrados pela Tabela 1 e pela Tabela 2, respectivamente.

Tabela 1 – Sinais de entrada utilizados pela Banco de Registradores

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
clk	1	*wire*	*Clock* da unidade.
addr_a	4	*wire*	Endereço do registrador a ser lido.
addr_b	4	*wire*	Endereço do registrador a ser lido.
reg_data	16	*wire*	Dado a ser escrito em um registrador.
write_reg	4	*wire*	Endereço do registrador que será escrito.
r_w	1	*wire*	*Flag* de leitura e escrita. 0 indica leitura, 1 escrita.

Tabela 2 – Sinais de saída utilizados pela Banco de Registradores

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
reg_a	16	*wire*	Resultado da operação.
reg_b	16	*wire*	*Flag* indicativa de resultados negativos.

3.2 Unidade Lógico Aritmética (ALU)

A segunda unidade funcional que compõe este projeto é a unidade lógico-aritmética (ALU), a qual opera sobre palavras de 16 *bits*. As instruções suportadas por este módulo são as lógico/aritméticas, como pode ser observado através da Figura 2. A Tabela 3 apresenta o conjunto de instruções suportadas, com exceção da instrução **Branch**, que não é mais suportada pela ALU, conforme exposto da subseção referente aos *stalls* e instruções de desvio, sendo mantido na tabela apenas para manter o organização na descrição das instruções.

Tabela 3 – Conjunto de instruções suportadas pela ALU

OpCode	**Instrução**	**Mnemônico**	**Operação**
0	Add	Add $s4, $s3, $s2	$s4 = $s3 + $s2
1	Sub	Sub $s4, $s3, $s2	$s4 = $s3 – $s2
2	Slti	Slti $s4, Imm, $s2	$s2 > Imm ? $s4 = 1 : $s4 = 0
3	And	And $s4, $s3, $s2	$s4 = $s3 $s2
4	Or	Or $s4, $s3, $s2	$s4 = $s3 | $s2
5	Xor	Xor $s4, $s3, $s2	$s4 = $s3 ^ $s2
6	Andi	Andi $s4, Imm, $s2	$s4 = $s2 & Imm
7	Ori	Ori $s4, Imm, $s2	$s4 = $s2 | Imm
8	Xori	Xori $s4, Imm, $s2	$s4 = $s2 ^ Imm
9	Addi	Addi $s4, Imm, $s2	$s4 = $s2 + Imm
10	Subi	Subi $s4, Imm, $s2	$s4 = $s2 – Imm
11	Jump	j Imm (12 bits)	$CP = Imm
12	Branch	bez -, $s3, $s2	*if* $s3 = 0 , $CP = $s2

3.2.1 Sinais de Entrada e Saída

Para que as operações definidas pela Tabela 3 possam ser realizadas, faz-se necessária a utilização de alguns sinais de entrada e saída pelo módulo. Esses sinais são apresentados pela Tabela 4 e pela Tabela 5, respectivamente.

Tabela 4 – Sinais de entrada utilizados pela ALU

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
codop	4	*wire*	Código de operação.
data_a	16	*wire*	Operando 1.
data_b	16	*wire*	Operando 2.

Tabela 5 – Sinais de saída utilizados pela ALU

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
out	16	*reg*	Resultado da operação.
neg	1	*wire*	*Flag* indicativa de resultados negativos.
zero	1	*wire*	*Flag* indicativa de resultados nulos.
overflow	1	*wire*	*Flag* indicativa de overflow.

3.3 Unidade de Multiplicação

A terceira unidade funcional implementada neste trabalho trata-se da Unidade de Multiplicação. Este módulo é responsável por receber dois sinais de 16 *bits* como entrada e retornar um valor de 32 *bits* como saída, contendo a multiplicação das entradas. A Figura 3 ilustra o diagrama esquemático do módulo em questão.

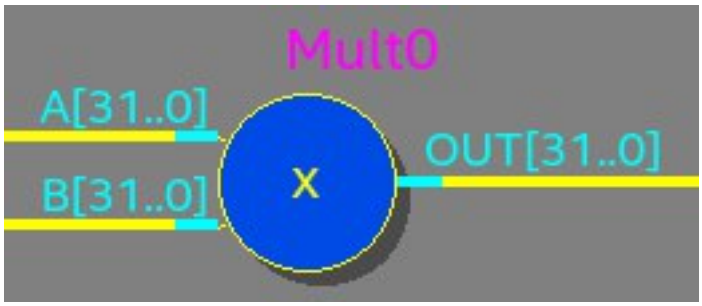

Figura 3 – Diagrama esquemático do Módulo de Multiplicação.

O conjunto de instruções suportadas pela Unidade de Multiplicação é apresentado pela Tabela 6.

Tabela 6 – Conjunto de instruções suportadas pela Unidade de Multiplicação

OpCode	**Instrução**	**Mnemônico**	**Operação**
13	GHI	ghi $s4, -, -	$s4 = $HI
14	GLO	glo $s4, -, -	$s4 = $LO
15	Mult	Mult $s3, $s2	{$HI, $LO} = $s3 * $s2

3.3.1 Sinais de Entrada e Saída

Para que as operações definidas pela Tabela 3 possam ser realizadas, faz-se necessária a utilização de alguns sinais de entrada e saída pelo módulo. Esses sinais são apresentados pela Tabela 7 e pela Tabela 8, respectivamente.

Tabela 7 – Sinais de entrada utilizados pela Unidade de Multiplicação

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
data_b	16	*wire*	Operando 1.
data_b	16	*wire*	Operando 2.

Tabela 8 – Sinais de saída utilizados pela Unidade de Multiplicação

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
out	32	*wire*	Resultado da operação.
neg	1	*wire*	*Flag* indicativa de resultados negativos.
zero	1	*wire*	*Flag* indicativa de resultados nulos.

Note que o sinal indicativo de *overflow* não está presente entre os sinais de saída da Tabela 8. Isso decorre do fato de operações de multiplicação, no contexto deste trabalho, nunca gerarem *overflow*, uma vez que os operandos possuem 16 *bits*, resultando, no máximo, no valor $2^{16} \times 2^{16} = 2^{32}$. Como o resultado da multiplicação é armazenado em dois registradores de 16 *bits* – $L0 contendo os 16 *bits* menos significativos e $HI contendo os 16 *bits* mais significativos, temos que é possível armazenar, no máximo, o valor 2^{32}, o que impossibilita a existência de *overflow* neste módulo.

3.4 Unidade de Controle

Outra unidade funcional presente neste trabalho se trata da Unidade de Controle. O módulo em questão é responsável por efetuar a devida configuração dos sinais de controle de forma a garantir o funcionamento correto de cada unidade de *hardware*. Esses sinais são configurados de acordo com a instrução que será executada, sendo este procedimento realizado durante a decodificação de tal instrução. Para tanto, a instrução é verificada quanto ao seu tipo, isto é, *Tipo R*, *I* ou *J* (*branches* e *jumps*), além daquelas instruções relativas à multiplicação, que são realizadas por *hardware* dedicado, são elas: GHI, GLO e *mult*.

3.4.1 Sinais de Entrada e Saída

Para a implementação deste módulo, foram utilizados alguns sinais de entrada e saída. Esses sinais são apresentados pela Tabela 9 e pela Tabela 10, respectivamente.

Tabela 9 – Sinais de entrada utilizados pela Unidade de Controle

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
clk	1	*wire*	*Clock* da unidade.
op_code	4	*wire*	Código da instrução.

Tabela 10 – Sinais de saída utilizados pela Unidade de Controle

Sinal	Tamanho (bits)	Tipo	Descrição
alu_b	1	*reg*	Determina se o segundo operando da ALU é um registrador, identificado por 0, ou um imediato, identificado por 1.
mul	1	*reg*	Sinal que identifica se a operação foi executada pela ALU, sinalizado pelo valor 0, ou pela Unidade de Multiplicação, sinalizado pelo valor 1.
source_wb	2	*reg*	Identifica a origem do valor a ser escrito no banco de registradores. O valor 0 indica a ALU, 1 indica o registrador $HI e 2 indica o registrador $LO.
r_w	1	*reg*	Determina se será realizada uma operação de escrita no Banco de Registradores.

3.5 Multiplexador

De forma a permitir a seleção de valores diferentes dependendo da instrução executada, como por exemplo os operandos da ALU ou ainda o registrador a ser escrito no estágio de *Write Back*, foi construída uma unidade que implementa um multiplexador. Este módulo, em específico, implementa três multiplexadores, com duas, três e quatro entradas, respectivamente. Sua utilização é descrita a seguir.

- **2 Entradas**: utilizado para o encaminhamento para os estágios de decodificação da instrução (ID) e de execução (EX). Também é utilizado para selecionar o operando B da ALU, podendo ser um imediato ou um registrador.
- **3 Entradas**: utilizado para selecionar o próximo valor do contador de programa (PC), isto é, se este será incrementado em uma unidade ou se será modificado através de um *branch* ou um *jump*.
- **4 Entradas**: utilizado exclusivamente para o encaminhamento para instruções de *branch*, encaminhando os valores imediatamente da saída da ALU, $LO ou $HI para o estágio de decodificação de instruções (ID).

3.5.1 Sinais de Entrada e Saída

Tabela 11 – Sinais de entrada utilizados pelo módulo *mux*

Sinal	Tamanho (bits)	Tipo	Descrição
clt	1	*wire*	Chave seletora.
sig_0	16	*wire*	Primeiro valor para seleção. Utilizado pelos multiplexadores de 2, 3 e 4 entradas.
sig_1	16	*wire*	Segundo valor para seleção. Utilizado pelos multiplexadores de 2, 3 e 4 entradas.
sig_2	16	*wire*	Terceiro valor para seleção. Utilizado apenas pelos multiplexadores de 3 e 4 entradas.
sig_3	16	*wire*	Quarto valor para seleção. Utilizado pelo multiplexador de 4 entradas.

Tabela 12 – Sinais de saída utilizados pelo módulo *mux*

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
out	16	*wire*	Valor de saída selecionado com base na chave seletora.

3.6 *Debouncer*

Um dos problemas encontrados na realização deste trabalho está relacionado ao *debouncing* (Figura 4) produzido pelos *push-buttons*, mesmo que o manual indique que esse problema é tratado por meio de um circuito *Schmitt trigger*. Devido a este fenômeno, o comportamento do sinal não se mantinha estável nas transições, produzindo falsas bordas. Com isso, ao pressionar o *key0* ou *key3*, usados no trabalho, o circuito se comportava de maneira inadequada, acionando outros módulos, como a ALU e o banco de registradores, mais de uma vez ou em momentos inapropriados, produzindo assim resultados errôneos. Com o intuito de corrigir esse comportamento inerente a toda chave mecânica, foi adicionado mais um módulo ao projeto – *debouncer.v*.

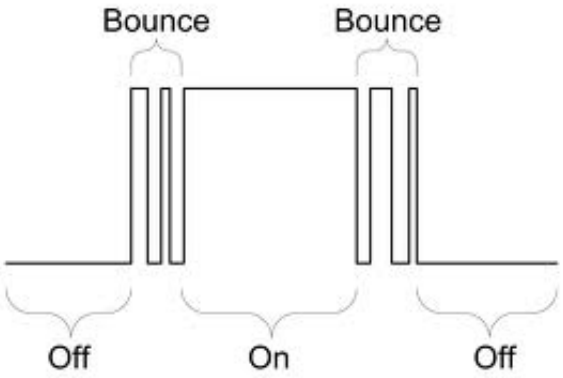

Figura 4 – Exemplo de *debouncing*.

O módulo em questão funciona monitorando o sinal em busca de mudanças no nível lógico. Ao detectar uma borda, o módulo ignora outras mudanças e após um periodo de tempo, tipicamente poucos milissegundos, ele confirma se o sinal de fato mudou de estado. Para isso, ele primeiro sincroniza o sinal com o *clock* proveniente da FPGA e, após sincronizado, utiliza um contador para criar a base de tempo para a reavaliação do sinal.

3.6.1 Sinais de Entrada e Saída

Para que o módulo implementasse suas funções corretamente, são necessários alguns sinais de entrada e saída, os quais são introduzidos pela Tabela 13 e Tabela 14, respectivamente.

Tabela 13 – Sinais de entrada utilizados pelo módulo *debouncer*

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
clk	1	*wire*	*Clock* da unidade.
PB	1	*reg*	Sinal com *debouncing*.

Tabela 14 – Sinais de saída utilizados pelo módulo *debouncer*

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
PB_state	1	*wire*	Sinal de saída, sem *debouncing*.

3.7 *Display* BDC 7 Segmentos

Por fim, o último módulo presente neste projeto tem como finalidade efetuar a decodificação de um *display* de 7 segmentos, onde são mostrados, de acordo com a especificação, os valores solicitados no formato hexadecimal. Para tanto, é recebido um valor de 4 *bits* codificado como BCD. A saída consiste de um registrador de 7 *bits*, onde a cada *bit* é atribuído o valor 0 ou 1 de acordo com o segmento que deverá ser ligado, possibilitando assim a formação de todos os valores compreendidos entre 0x0 e 0xF.

3.7.1 Sinais de Entrada e Saída

A Tabela 15 e a Tabela 16 sintetizam os sinais utilizados por esse módulo.

Tabela 15 – Sinais de entrada utilizados pelo módulo de controle do *display* BCD de 7 segmentos

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
hex_value	4	wire	Valor hexadecimal a ser escrito no *display*.

Tabela 16 – Sinais de saída utilizados pelo módulo de controle do *display* BCD de 7 segmentos

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
disp_value	7	reg	Saída que contém a configuração para cada segmento do *display*.

4 Validação

Este trabalho teve como principal objetivo a implementação de um *pipeline*, aproveitando da estrutura que advém dos trabalhos anteriores. Desse modo, sua validação foi feita por meio de vários testes, sintetizados em três *scripts* de simulação. O primeiro deles (**simu_fwd.do**) tem por função testar os encaminhamentos no geral, ao fazer algumas poucas operações que demandam todas as combinações de encaminhamentos possíveis. O segundo e o terceiro (**simu_branch_tom.do** e **simu_branch_n_tom.do**) são testes específicos da instrução *beq*. Isso se deve ao fato desta instrução ter demandado tratativas específicas que deveriam ser testadas com o intuito de assegurar seu perfeito funcionamento.

5 Conclusão

Este trabalho apresentou o desenvolvimento de um sistema de *hardware* constituído por um Banco de Registradores, uma Unidade Lógico-Aritmética (ALU), uma Unidade de Multiplicação

e uma Unidade de Controle. De forma a garantir o funcionamento correto de cada unidade desenvolvida, foi implementada uma unidade de controle, responsável por administrar os sinais de controle que serão passados na decodificação e trafegarão através dos três estágios do pipeline. Em seguida, para a validação do projeto foi feito alguns testes para as instruções no geral e dois para os branchs, que necessitaram de uma atenção especial devido a sua implementação não trivial. Para os testes, foi utilizado o ambiente de simulação *ModelSim*, possibilitando assim a verificação da corretude do trabalho desenvolvido.

Neste contexto, este trabalho se mostrou importante para a consolidação dos conceitos teóricos vistos em sala de aula, pois proporcionou a construção de um processador que utiliza *pipeline*, que é, sem dúvidas, uma das estratégias de otimização mais difundidas, presente na grande maioria dos processadores modernos. Além disso, um ponto importante para o aprendizado dos integrantes do grupo está relacionado à utilização de encaminhamento (*forwarding*), aliada à busca por se evitar *stalls* no processador, devido ao prejuízo em termos de tempo de execução e *performance* do *pipeline* que bolhas nele inseridas podem ocasionar.

6 Referências Bibliográficas

Patterson, David; Hennessy, John. Computer Organization and Design: the Hardware/Software Interface, 3th ed. Elsevier, 2005.

Tanenbaum, Andrew S.; Austin, Todd. Structured Computer Organization, 6th ed. Prentice Hall, 2012.