UNIVERSIDADE FEDERAL DE MINAS GERAIS

DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

Organização de Computadores II
Trabalho Prático IV – Inclusão de uma Unidade de Multiplicação

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
06 de novembro de 2017

Sumário

1 Introdução

Tendo em vista que nos trabalhos anteriores foram construídas uma Unidade Lógico-Aritmética (ALU) e um Banco de Registradores, este trabalho tem como objetivo principal a adaptação do caminho de dados do processador desenvolvido para a inclusão de uma Unidade de Multiplicação. Dessa forma, com esta adaptação do *hardware* o processador resultante agora é capaz de executar instruções de multiplicação de entradas de 16 *bits*, emitindo um resultado de 32 *bits* como saída. Para tanto, foi utilizada a linguagem de descrição de *hardware* Verilog e o pacote de desenvolvimento da Altera – *Quartus* e *ModelSim*. 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 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.

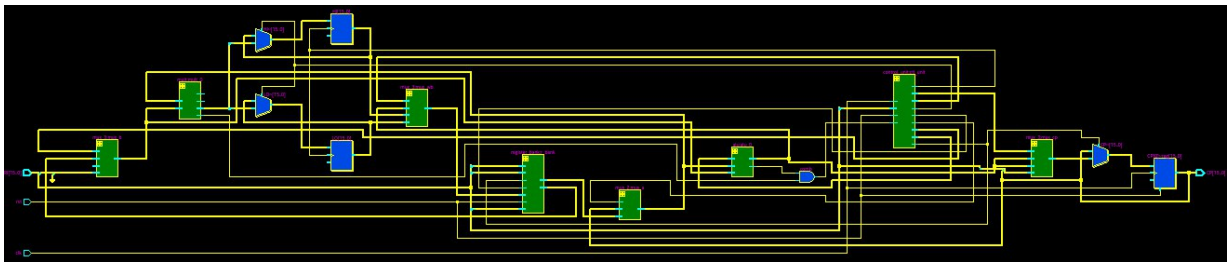

Figura 1 – Diagrama esquemático do Circuito.

2.1 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. 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.

2.1.1 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	*reg*	Resultado da operação.
reg_b	16	*wire*	*Flag* indicativa de resultados negativos.

2.2 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*. Assim, de forma a proporcionar uma melhor compreensão acerca do funcionamento deste módulo, a Figura 2 ilustra o formato definido para as instruções e, em seguida, a Tabela 3 apresenta o conjunto de instruções suportadas.

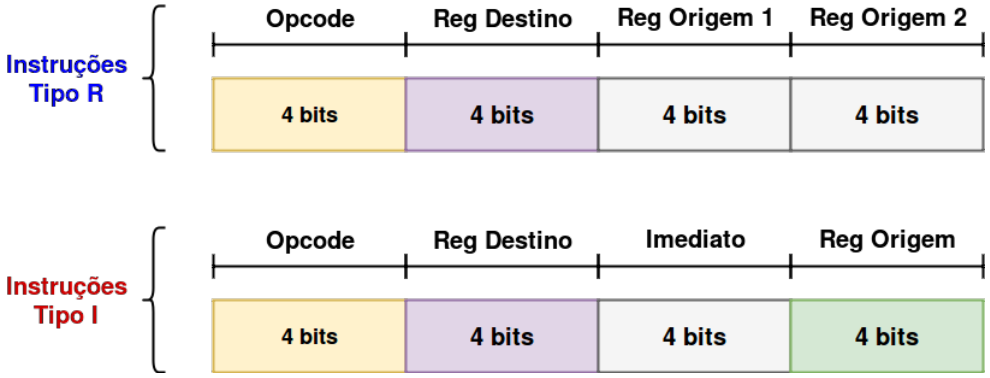

Figura 2 – Formato das instruções.

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

2.2.1 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.

2.3 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.

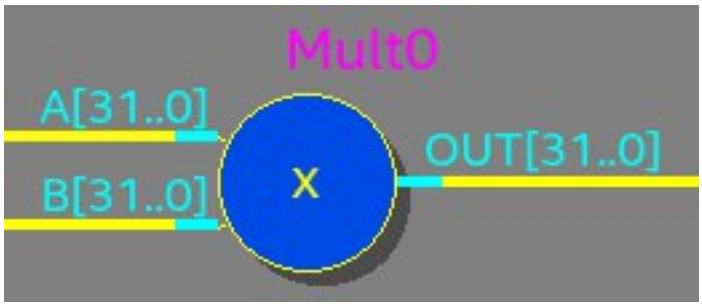

Figura 3 – Diagrama esquemático do Módulo de Multiplicação.

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 $s4, $s3, $s2	$s4 = $s3 * $s2

2.3.1 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	*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.

2.4 Unidade de Controle

A última unidade funcional presente neste trabalho é a 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*. Para tanto, o caminho de dados do processador foi implementado através de uma máquina de estados finitos (FSM), como ilustra a Figura 4.

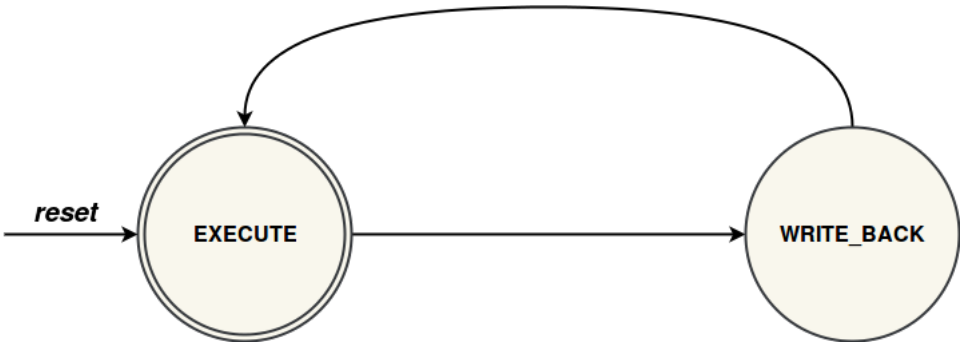

Figura 4 – FSM pertinente à Unidade de Controle.

A partir da Figura 4, percebe-se a existência de apenas dois estados. O primeiro deles é intitulado EXECUTE, o qual é responsável por efetuar a configuração dos sinais referentes à seleção dos operandos da ALU ou da Unidade de Multiplicação, dependendo da instrução a ser executada, levando em consideração também o tratamento de imediatos. Vale observar, ainda, que não existe um estado para *Fetch* da instrução. Isso se deve ao fato de que a memória de instruções foi retirada nesta implementação, sendo a instrução transmitida diretamente como sinal de entrada para o módulo de controle.

Por fim, o estágio WRITE_BACK é acionado. Este estado, por sua vez, é responsável por efetuar a escrita no Banco de Registradores, tendo como base os sinais previamente configurados. Além disso, o estado em questão também é responsável por atualizar o Contador de Programa (CP), de acordo com o tipo da instrução. Neste caso, é importante observar que essa atualização é realizada de acordo com a situação do *branch* caso exista, isto é, se foi tomado ou não. Após essas operações, o primeiro estado é novamente ativado, possibilitando assim a execução das demais instruções presentes na Memória de Instruções. É importante ressaltar, entretanto, que, mesmo com a ausência da Memória de Instruções, as instruções de *jumps* e *branches* foram mantidas, resultando em uma saída do módulo com o valor do Contador de Programa, de forma que o módulo de memória possa ser incluído no projeto facilmente no futuro.

2.4.1 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.
rst	1	*wire*	Sinal de *reset*.
zero	1	*wire*	Flag da ALU, identifica se o *branch* foi tomado ou não. Necessário para a atualização do CP.
op_code	4	*wire*	Código da operação a ser executada pela ALU.

Tabela 10 – Sinais de saída utilizados pela Unidade de Controle

Sinal	**Tamanho (bits)**	**Tipo**	**Descrição**
exe	1	*reg*	Identifica o estágio *Execute* da FSM.
wb	1	*reg*	Identifica o estágio *Write Back* da FSM.
ula_op	4	*reg*	Identifica o código da operação que será realizada na ALU.

alu_a	1	*reg*	Determina se o primeiro operando da ALU é CP ou RegA.
alu_b	2	*reg*	Determina se o segundo operando da ALU é 1, imediato estendido ou RegB.
fonte_cp	2	*reg*	Determina qual o valor a ser escrito em CP: $(CP + 1)[0]$, ALU_S[1] ou *Endereço de Salto*[2].
fonte_wb	2	*reg*	Sinal para a identificação do dado a ser escrito no registrador de destino, isto é, da ALU ou HI/LO.
mul	1	*wire*	Sinal que identifica se a operação foi executada pela ALU ou pela Unidade de Multiplicação.
r_w	1	*reg*	Determina se será realizada uma operação de leitura ou escrita no Banco de Registradores.

3 Validação

Como a essência deste trabalho é a construção de uma Unidade de Multiplicação, para sua validação foi criado um conjunto de testes de multiplicações de registradores. Os testes foram criados de forma a configurar todas as possibilidades de multiplicação, isto é, *positivo* com *positivo*, *positivo* com *negativo* e *negativo* com *negativo*. Através da observação dos resultados obtidos via simulação, constatou-se o funcionamento correto do circuito. Vale ressaltar, ainda, que os resultados foram conferidos através do ambiente de simulação proporcionado pelo *ModelSim*, através da utilização de um *script* de simulação (**simu_def.do**) criado para este propósito.

4 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 FSM composta por dois estados, responsáveis pela execução da instrução e da escrita no banco de registradores seguida pela atualização do contador de programa (CP), respectivamente. Em seguida, para a validação do projeto foi criado um conjunto de testes de forma a configurar todas as possibilidades de multiplicação entre dois registradores. 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 implementado através de uma máquina de estados. Além disso, um ponto importante para o aprendizado dos integrantes do grupo está relacionado à utilização de um *script* **.do** para a automação dos testes via simulação, que permitiu aos alunos envolvidos a familiarização e execução de testes em ambiente de simulação para uma futura prototipação em *hardware* reconfigurável.

5 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.