Design de um Processador RISC de 8-bits Multi-ciclo utilizandoVerilog HDL em FPGA

Jair Rocha Souza

***Resumo*—Este artigo apresenta o design, implementação e validação de um processador RISC de 8 bits multi-ciclo voltado para prototipagem em FPGA. O objetivo principal é aplicar no projeto conceitos fundamentais de arquitetura de computadores e otimização lógica. O processador emprega uma arquitetura Harvard Modificada, apresentando memórias separadas para programa (ROM) e dados (RAM), permitindo ao mesmo tempo a busca imediata de operandos a partir do caminho de instrução. A microarquitetura é controlada por uma Máquina de Estados Finitos (FSM) multi-ciclo, suportando um Conjunto de Instruções (ISA) de 35 instruções, incluindo operações aritméticas, fluxo de controle (saltos e loops) e manipulação de pilha (PUSH/POP/CALL/RET) suportada por registradores dedicados. O projeto foi descrito utilizando Verilog HDL, sintetizado no Quartus visando a FPGA Cyclone V 5CEBA4F23C7, e validado através de simulações no ModelSim. Resultados experimentais demonstrando rotinas de manipulação de vetores confirmam a funcionalidade do processador, validando a arquitetura proposta.**

I. INTRODUÇÃO

A indústria de processadores se tornou um dos maiores pilares de desenvolvimento da sociedade contemporânea. A produção em larga escala dos processadores ocasiona implicações econômicas e geopolíticas. As diferentes aplicações desses processadores demandam variados tipos de designs [1]. Esses aspectos tornam o design lógico em uma etapa determinante no desempenho e custo dos processadores. Por meio de circuitos integrados reconfiguráveis é possível desenvolver esses designs mesclando a flexibilidade de software com a alta performance de hardware como, por exemplo, nas placas FPGAs [2]. Já a RISC é uma técnica de design utilizada para minimizar a complexidade das instruções, a quantidade de ciclos por instrução e custo durante a etapa de implementação [3]. Em processadores RISC as operações de carregamento e armazenamento são utilizadas apenas para acesso de memória, enquanto as outras operações se baseiam nos registradores permitindo a execução de instruções em múltiplos ciclos. Desta forma, o seguinte trabalho tem como objetivo central projetar um processador RISC multi-ciclo de 8 bits, utilizando a tecnologia FPGA como plataforma de prototipagem. A partir disso será possível aplicar e compreender, em hardware reconfigurável, os conceitos fundamentais de otimização lógica e arquitetura de computadores. Para atingir esse objetivo as seguintes ferramentas foram utilizadas: a linguagem de descrição de máquina Verilog HDL, o Quartus na síntese do código e o ModelSim para validação em simulação.

II. ARQUITETURA DO PROCESSADOR

A. Visão Geral

O processador projetado tem suporte para operações com dados de até 8 bits e 256 bytes disponíveis para endereçamento na memória. A organização da memória segue a arquitetura Havard modificada, que se caracteriza pela separação entre memória de dados na RAM e memória de programa na ROM, permitindo a otimização dos barramentos para cada tipo de acesso [4]. Todavia, na Havard modificada a separação não é total. No projeto desenvolvido, caminhos de dados foram implementados para operandos imediatos entre a ROM e diferentes modulos na CPU (Unidade Central de Processamento). Por exemplo, na instrução LDC que carrega uma constante no registrador A o banco de registradores vai receber o valor a partir de um registrador conectado a saída da ROM.

B. Organização dos registradores

O processador possui 2 registradores de uso geral A e B e 2 para uso especifico ACC e COUNT. O ACC vai ser utilizado para operações em que o valor de A em diferentes estágios precise ser acumulado. COUNT vai ser dedicado para estruturas de repetição. O registrador IR vai receber o valor de instrução, o SP é o registrador de pilha, *Z_flag_reg* e *C_flag_reg* são os registradores que vão receber as flags da última operação da Unidade Lógica Aritmética (ULA) e *alu_out_reg* vai armazenar o resultado da operação. A Tabela 1 mostra o endereço de 3 bits dos registradores de uso geral e específicos que estão no banco de registradores. Os outros registradores serão acessados diretamente no Caminho de dados (Datapath).

TABELA 1. ENDEREÇO REGISTRADORES

Registrador	Endereço
A	001
B	010
ACC	011
COUNT	100

C. Conjunto de Instruções (ISA)

O Conjunto de instruções é dívida em 4 tipos de instruções, movimentação de dados, aritmética ou lógica, controle de fluxo e controle de máquina. As instruções possuem 2 partes em que os 5 primeiros bits são os *opcodes* e os 3 ultimos se referem ao endereço do registrador que será utilizado. Caso a instrução não necessite de registradores os 3 últimos bits são 000. No total foram 35 instruções e 24 *opcodes*. A Tabela 2 mostra a classificação, a representações binária das instruções e as operações correspondentes em que # representa o operando.

TABELA 2. INSTRUÇÕES DO PROCESSADOR

Instrução	Mnemônico	Operação
Logica e aritmética		
00001001	ADD	A = A + B

00010001	SUB	A = A - B
00111001	AND	A = A & B
01000001	OR	A = A | B
01010001	ADC	A = A + B + Cin
01011001	XOR	A = A ^ B
01100001	NOTA	A = ~A
01100010	NOTB	B = ~B
01101001	SHL	A = A << 1
01110001	SHR	A = A >> 1
10000010	INC_B	B = B + 1
10001011	ADD_A	ACC = A + ACC
Movimentação de dados		
00011001	LDA	A = RAM[#]
00100001	STA	RAM[#] = A
00011010	LDB	B = RAM[#]
00100010	STB	RAM[#] = B
00101001	LDC	A = #
00101010	LDC_B	B = #
01111001	LDA_IN	A = RAM[B]
00101100	LDC_COUNT	COUNT = #
00101011	LDC_ACC	ACC = #
10101001	PUSH_A	RAM[SP] = A, SP--
10101010	PUSH_B	RAM[SP] = B, SP--
10101011	PUSH_ACC	RAM[SP] = ACC, SP--
10110001	POP_A	SP++, A =RAM[SP]
10110010	POP_B	SP++, B = RAM[SP]
10110011	POP_ACC	SP++, ACC = RAM[SP]
00100011	ST_ACC	RAM[#] = ACC
11000001	ST_IN	RAM[B] = A
Controle de fluxo		
00110000	JMP	PC = #, se Z_flag == 1
01001000	JC	PC = #, se C_flag == 1
10010100	DJNZ	PC = #, se COUNT-- != 0
10011110	CALL	RAM[SP] = PC, SP--, PC = #
10100000	RET	SP++, PC = RAM[SP]
Controle de maquina		
HALT	10111000	Termina as operações

III. DESCRIÇÃO DOS MODULOS FUNCIONAIS

O processador vai possuir os seguintes modulos, ULA, Unidade de Controle, Banco de Registradores, RAM, ROM, PC e Caminho de dados. A Figura 1 é a representação básica da conexão desses componentes.

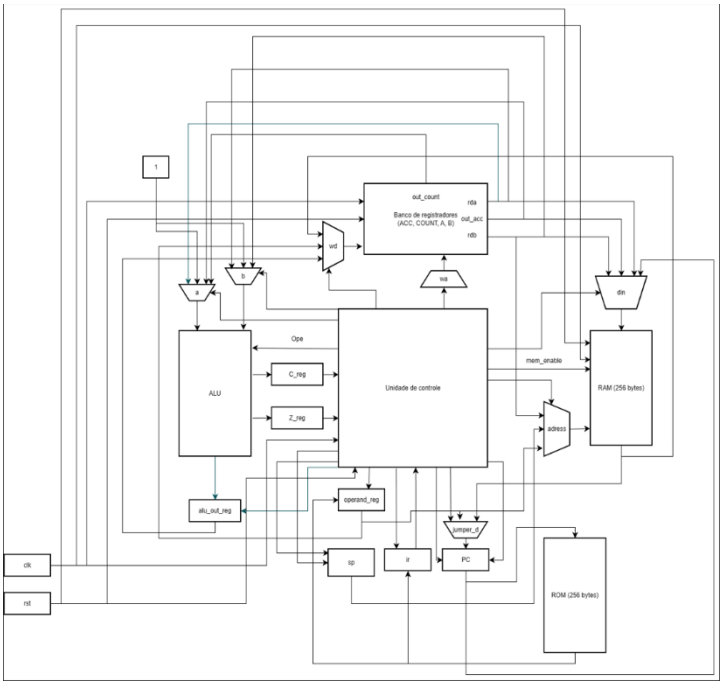


Figura 1. Arquitetura básica do processador

A. *ULA*

A ULA possui duas entradas 8-bits *a* e *b* e é responsável pela execução das operações de aritmética e logica com essas duas entradas. As suas saídas são o resultado da operação e duas flags, a *Z_flag* que indica se o resultado está zerado e a *C_flag* que representa o *carry* de saída para operações de adição ou *borrow* para subtração. *a* e *b* estão conectados a multiplexadores 4x1 que determinam a entrada a partir do sinal do seletor de 2 bits que vem da Unidade de controle. A operação é determinada com base no sinal de controle *ope*.

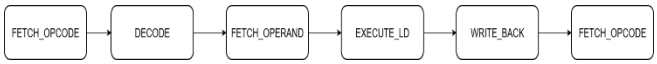
B. *Unidade de controle (FSM)*

A Unidade de Controle é responsável por gera os sinais de controle para os multiplexadores (MUXs) e modulos do Datapath. No caso de processadores multi-ciclo, os sinais são definidos com base em estados de uma Máquina de Estados Finitos (FSM). A partir de uma sequência determinada de estados ao longo de vários ciclos de *clock* será possivel executar uma instrução. Portanto, o período do *clock* deve ser ajustado para acomodar o estágio com o caminho combinacional mais longo, garantindo que a operação mais

lenta possa ser finalizada dentro de um único ciclo. A Figura 2 mostra um diagrama de estados para a instrução DJNZ, enquanto Figura 3 é o diagrama da instrução LDA.

Figura 2. Diagrama de estado DJNZ

Figura 3. Diagrama de estado LDA



C. *Banco de Registradores*

O Banco de Registradores possui 4 registradores de 8 bits que podem ser utilizados para armazenar e carregar dados para as diferentes operações. O sinal de controle *we* determina se o modulo deve executar a operação de escrita. A entrada *wd* é o dado para a operação de escrita e *wa* o endereço de 3 bits do registrador, o qual determina em qual registrador a escrita vai ocorrer. As saídas representam os valores de leitura de cada registrador.

D. *RAM*

A RAM é a memória volátil do processador com dados de 8 bits e 256 endereços disponíveis. Ela possui uma entrada de dados, uma entrada de endereço, o sinal de controle para operação de escrita e tem uma saída para leitura.

E. *ROM*

A ROM é a memória não volátil com 256 endereços disponíveis que vai armazenar as instruções e operandos de 8 bits.

F. *PC*

O PC é o modulo responsável por armazenar a próxima instrução a ser executada. Ele tem dois sinais de controle que indicam se a proxima instrução é a soma da atual mais 1 ou se deve ser igual ao dado de entrada *jumper_d* que pode ser um operando ou o valor no registrador de pilha. A sua saida é o valor de leitura da proxima instrução.

G. *Caminho de dados (Datapath)*

O caminho de dados vai ser o modulo que representa as conexões entre os diferentes componentes. Multiplexadores e registradores serão definidos nesse módulo. O registrador SP vai ser utilizado para armazenar o endereço de dados ou de instruções entre funções de *CALL* e *RETURN*.

IV. Resultados da Simulação

A simulação foi realizada na ferramenta Quartus, tendo como alvo a placa FPGA Cyclone V 5CEBA4F23C7 e os resultados foram analisados no ModelSim. A Figura 4 mostra os resultados em formato de onda do programa descrito na Tabela 3. A Tabela 4 mostra alguns resultados do *Compilation Report* gerado pelo Quartus. Para facilitar a leitura as instruções foram escritas em hexadecimal. O objetivo do programa é multiplicar todos os valores de um vetor por 2, atualizar o resultado no vetor e salvar o ultimo valor na memoria RAM. O vetor está localizado nos endereços em hexadecimal de 80 até 85 e inicialmente inclui os valores 1, 2, 3, 4, 5 em decimal. Os valores de *rda*, *rdb*, e *out_count* representam os resultados de leitura em *a*, *b* e *out_count* respectivamente e estão sempre ativos. *a* é carregado com o valor do vetor e depois recebe o resultado da operação de deslocamento para a esquerda de 1 bit. *b* é o endereço em hexadecimal do valor no vetor que será carregado em *a*. Enquanto o valor de *count* não for zerado a estrutura de repetição continua com o pulo no endereço de instrução em 04 que nesse caso seria *SHL*.

TABELA 3. PROGRAMA DA SIMULAÇÃO

Instrução em hexadecimal	**Operação**	**Operando**
2C	LDC_COUNT	05
2A	LDC_B	80
79	LDA_IN	
69	SHL	
C1	ST_IN	
82	INC_B	
94	DJNZ	04
21	STA	30
A8	HALT	

TABELA 4. COMPILATION REPORT

Utilização lógica	**Usado**	**Disponível**	**Utilização**
ALMs	1320	18,480	7%
Registradores	2145	18,480	12%
Pinos	36	224	16%
Bits de memoria	0	3,153,920	0%

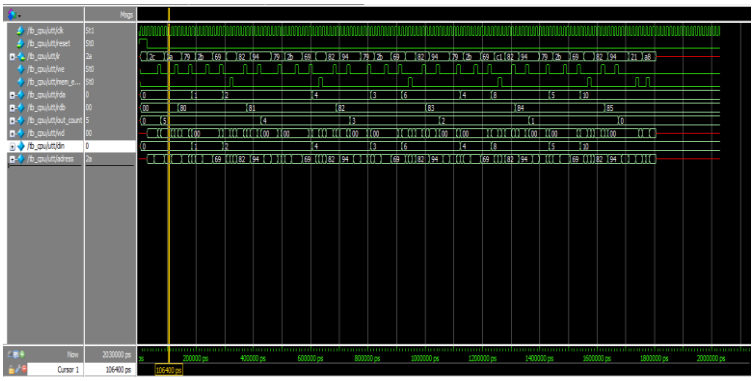


Figura 4. Resultado da simulação

V. Conclusão

O processador de 8-bits RISC utilizando Verilog HDL foi projetado e implementado na placa FPGA Cyclone V 5CEBA4F23C7. Todas as instruções foram testadas individualmente e coletivamente. O processador é capaz de executar operações básicas de lógica e aritmética, estruturas de repetição e funções com as operações de pilha. Para trabalhos futuros seria possivel fazer o mapeamento para

dispositivos de entrada e saída e aumentar o desempenho com o uso de *Pipelines*.

REFERÊNCIAS

[1] David A. Patterson, John L. Hennessy, “Computer Architecture A Quantitative Approach”, University of California, Berkley and Stanford University.

[2] J. Jeemon, "Pipelined 8-bit RISC processor design using Verilog HDL on FPGA," 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), Bangalore, India, 2016, pp. 2023-2027, doi: 10.1109/RTEICT.2016.7808194.

[3] R. Jasinski, "Sivarama P. Dandamudi Guide to RISC Processors for Programmers and Engineers. Springer Verlag (2005). ISBN 0-387-21017-2. 387 pp. Hardbound," in The Computer Journal, vol. 49, no. 3, pp. 375-375, May 2006, doi: 10.1093/comjnl/bxk001.

[4] Francillon, Aurélien, and Claude Castelluccia. "Code injection attacks on harvard-architecture devices." Proceedings of the 15th ACM conference on Computer and communications security. 2008.