UNIVERSIDADE FEDERAL DO PARANÁ

ANDRIELI LUCI GONÇALVES

RELATÓRIO SOBRE A IMPLEMENTAÇÃO DE UMA UNIDADE LÓGICA E ARITMÉTICA A PARTIR DE CIRCUITOS COMBINACIONAIS

CURITIBA

2024

ANDRIELI LUCI GONÇALVES

RELATÓRIO SOBRE A IMPLEMENTAÇÃO DE UMA UNIDADE LÓGICA E ARITMÉTICA A PARTIR DE CIRCUITOS COMBINACIONAIS

Relatório apresentado ao curso de Graduação em Ciência da Computação, Setor de Ciências Exatas, Universidade Federal do Paraná, como parte do Trabalho I para a disciplina de Circuitos Digitais.

Professor: Armando Luiz Nicolini Delgado.

CURITIBA

2024

SUMÁRIO

1 INTRODUÇÃO

O presente relatório, direcionado à disciplina de Circuitos Digitais (CI1068), tem como objetivo explorar o processo de desenvolvimento e testagem de uma Unidade Lógica e Aritmética (ULA), construída somente a partir de portas lógicas.

Integrada à Unidade Central de Processamento (CPU), a ULA representa um circuito combinatório responsável pela realização de operações aritméticas (como soma, subtração, incremento e decremento), lógicas (como XOR e complemento), bem como deslocamentos à esquerda e à direita, em um sistema digital.

Para o projeto, foi utilizado o simulador *Digital*, ferramenta também empregada em sala de aula para realização de testes e compreensão do funcionamento de circuitos combinacionais.

2 DESCRIÇÃO DO PROJETO

A Unidade Lógica e Aritmética desenvolvida é composta pelas entradas A e B, ambas de 16 bits e na representação de complemento de dois, uma entrada F de três bits e duas saídas R e S, de 16 e três bits, respectivamente. Para uma compreensão clara da ULA, explicam-se abaixo suas principais entradas e saídas:

a) A e B: operandos;
b) F: seleciona a operação que será efetuada sobre A e B;
c) R: resultado da operação;
d) S_0: será 1 quando R for zero — e 0 caso contrário;
e) S_1: será 1 quando houver *overflow* nas operações aritméticas — e 0 caso contrário;
f) S_2: será 1 quando R for negativo — e 0 caso contrário.

O projeto foi organizado em 11 arquivos, sendo nove dedicados aos componentes intermediários ou circuitos que executam as operações específicas. Para além desses, há, também, um arquivo que abrange todos os blocos personalizados das oito operações, incluindo configurações das saídas e testes, enquanto outro arquivo encapsula o bloco que representa a ULA propriamente dita.

2.1 COMPONENTES INTERMEDIÁRIOS

2.1.1 Multiplexador 8:1

Para a ação futura de selecionar as operações da ULA, foi criado um multiplexador 8:1 (arquivo **mux8-1_1bit.dig**). Este componente é capaz de escolher entre oito entradas distintas $D_{0:7}$, com base em três seletores $S_{0:2}$, para produzir uma única saída, representada por Y. Foram utilizados três inversores (um para cada bit do seletor), oito portas lógicas AND e três portas lógicas OR.

2.1.2 Multiplexador 2:1

Seguindo a mesma premissa do multiplexador 8:1, foi criado um multiplexador 2:1 (arquivo **mux2-1.dig**). Ele é composto por duas entradas $A_{0:1}$, um seletor B e uma saída Y. Utilizou-se um inversor, duas portas lógicas AND e uma porta lógica OR.

2.1.3 Somador completo

Para a criação das operações aritméticas envolvendo 16 bits, foi importante, inicialmente, criar um somador completo (arquivo **somador_1bit.dig**) que trabalha apenas com um bit de dado. O somador é composto pelas entradas A (operando), B (operando) e Ci (*carry* de entrada), tal qual por duas saídas, S (gerada a partir de um XOR entre as três entradas) e Co (*carry* de saída, que é gerado a partir da combinação das portas lógicas AND e OR sobre A, B e Ci).

2.2 OPERAÇÕES

2.2.1 Deslocamento à esquerda

Selecionada a partir do código 000, a operação de deslocamento à esquerda (arquivo **deslocador_esquerda.dig**) é representada por $A<<B_{3:0}$, o que significa dizer que sua funcionalidade é "mover" os bits do operando A até 15 casas para a esquerda.

A partir das entradas A e B, foram utilizados dois distribuidores, um para segmentar A — tornando uma entrada de 16 bits em 16 entradas de 1 bit cada — e outro para obter apenas os quatro bits menos significativos de B. Ademais, foram adicionados 16 multiplexadores 16:1 (derivados da combinação de dois multiplexadores 8:1 e um 2:1), cujos seletores eram os bits obtidos do operando B.

Para a saída Y_{15}, todas as entradas do multiplexador continham os bits de A, enquanto na saída Y_{14} a última entrada foi substituída pelo número 0. Essa lógica foi replicada até Y_0, em que apenas a primeira entrada continha dados de A, e as demais eram 0.

2.2.2 Deslocamento à direita

Selecionada a partir do código 001, a operação de deslocamento à direita (arquivo **deslocador_direita.dig**) é representada por $A>>B_{3:0}$ e segue a mesma proposta do deslocamento à esquerda. A diferença está presente no sentido do deslocamento, já que as casas são “movidas” para a direita. Para tal ação, alterou-se o preenchimento das entradas do multiplexador 16:1: na saída Y_{15}, apenas a primeira entrada (D_0) contém uma parte do operando A (o bit A_{15}); na saída Y_{14}, as entradas

D_0 e D_1 são ocupadas pelos bits A_{14} e A_{15}, respectivamente, enquanto as demais são 0. A lógica é estendida até a saída Y_0, onde todas as entradas são preenchidas.

2.2.3 XOR

Selecionada a partir do código 010, a operação XOR (arquivo **xor.dig**) é representada por A $\oplus$ B e tem como finalidade comparar bits correspondentes e retornar 1 se são diferentes ou 0 se são iguais. Tal operação é constituída de duas entradas, A e B, e uma saída Y. Para o circuito, trabalhou-se com a porta lógica XOR da própria ferramenta, considerando 16 bits de dados.

2.2.4 Complemento

Selecionada a partir do código 011, a operação complemento (arquivo **inversor.dig**), representada por !A, tem como objetivo inverter o valor armazenado na entrada A por meio da porta lógica conhecida como NOT (ou inversor). A operação possui apenas uma saída Y.

2.2.5 Soma

Selecionada a partir do código 100, a operação de soma (arquivo **somador_16bits.dig**), representada por A + B, pretende somar os operandos A e B. O somador é constituído, a princípio, pelas entradas A, B e Ci. No entanto, foi aplicado um distribuidor em A e B, de modo que cada uma dessas duas entradas de 16 bits se transformassem em 16 entradas de um bit cada. É importante destacar que, para este contexto, o valor inicial de Ci é 0.

Diante desse cenário, foram adicionados ao circuito 16 somadores completos (arquivo **somador_1bit.dig**). O primeiro recebe as entradas A_0, B_0 e Ci, tendo como saída S_0. Sua outra saída, Co, é conectada ao próximo somador completo — processo que se repete até o bloco final, que se diferencia por resgatar os últimos valores de Co e Ci e aplicar um XOR.

A saída do XOR chama-se Ov e tem como finalidade indicar se houve *overflow* na operação. Além dessa saída, há também S — formada pelos bits $S_{0:15}$ após o uso de um distribuidor.

2.2.6 Subtração

Selecionada a partir do código 101, a operação de subtração, representada por A - B, é derivada da soma. A diferença encontra-se no arquivo onde há a reunião de todos os componentes da ULA: é aplicado um inversor sobre o operando B e o valor inicial de Ci é 1. Estas ações são necessárias para que o subtraendo, durante a operação, permaneça em sua forma negativa em complemento de dois.

2.2.7 Incremento

Selecionada a partir do código 110, a operação de incremento, representada por A + 1, também é derivada da soma. Neste caso, a segunda parcela da soma é apenas o número 1; o valor inicial de Ci permanece 0.

2.2.8 Decremento

Selecionada a partir do código 111, a operação de decremento, representada por A - 1, resgata características da subtração. Neste contexto, o valor do número a ser decrementado (1) é invertido e a entrada Ci recebe 1 como seu valor inicial.

2.3 ULA COM A REUNIÃO DE TODOS OS COMPONENTES

Este circuito (arquivo **ula.dig**) foi separado em 5 grandes partes: túneis, testes, operações, multiplexador 8:1 e saídas. Os túneis (para as entradas F, 1 e 0) foram criados para facilitar a organização do circuito combinacional, já que tais itens seriam usados mais de uma vez entre os componentes. A seção de testes, por sua vez, foi adicionada a este arquivo para ajudar na análise acerca das operações.

O bloco de operações é composto por três principais entradas: A, B e 1 — todas de 16 bits. Além disso, B e 1 também possuem suas versões invertidas. Para cada operação lógico-aritmética da ULA, foi adicionado seu respectivo bloco, o qual foi conectado às entradas A, B e 1 anteriormente citadas.

Ao centro, há um multiplexador 8:1 de 16 bits, cujas entradas $D_{0:7}$ são conectadas às saídas principais das operações. Seu seletor é representado por F. Para criar tal multiplexador, foi importante desenvolver um novo circuito (arquivo

mux8-1_16bits.dig): ele é composto por 16 multiplexadores 8:1 (arquivo **mux8-1_1bit.dig**), uma entrada S de 3 bits, 128 entradas de 1 bit cada e uma saída Y. Cada entrada D_n (referente às oito operações) é divida em 16 entradas menores $Dn_{0:15}$. Tais entradas são conectadas aos 16 multiplexadores 8:1, que lidam com dados de apenas 1 bit por vez. A reunião de todos esses componentes dá origem a um único multiplexador que trabalha com 16 bits de dados. As saídas $Y_{0:15}$ dos multiplexadores foram transformadas em Y através de um distribuidor.

Retornando ao arquivo da ULA, tem-se a seção das saídas. R (resultado da operação) é a saída do multiplexador 8:1 de 16 bits de dados. Ademais, têm-se as *flags* S_0, S_1 e S_2:

a) A primeira foi moldada a partir da inversão do resultado de R, separação dos seus 16 bits e reunião de cinco portas lógicas AND, permitindo identificar se R é zero ou não;
b) A segunda utiliza de um multiplexador 8:1 (com 1 bit de dado), o mesmo seletor F (que define a operação a ser feita) e tem as suas quatro últimas entradas $D_{4:7}$ preenchidas pelas saídas Ov de cada uma das operações aritméticas (soma, subtração, incremento e decremento). Enquanto isso, as quatro primeiras entradas $D_{0:3}$ são iguais a 0. A partir da operação escolhida e do seu respectivo resultado, a *flag* resgata o valor da saída Ov e indica se houve (ou não) *overflow*;
c) A terceira obtém o valor do bit mais significativo da saída R, usado como bit de sinal (negativo ou positivo).

2.4 ULA ENCAPSULADA

Por fim, a ULA em formato encapsulado (arquivo **ula_principal.dig**) foi criada com o objetivo de tornar toda a Unidade Lógica e Aritmética criada anteriormente em um único bloco personalizado. Ainda são utilizadas as entradas A, B e F e as saídas R, S_0, S_1 e S_2. Neste arquivo, considera-se o sistema binário para a representação dos números.

3 TESTES REALIZADOS

Para garantir que todas as operações e configurações aplicadas funcionassem corretamente, foi importante realizar testes com valores estratégicos para A e B. Desse modo, qualquer resultado inesperado poderia ser rapidamente identificado e corrigido.

Para facilitar na criação, análise e legibilidade dos testes, foi optado por trabalhar com valores em hexadecimal (sistema de numeração composto por 16 algarismos). Entretanto, o resultado continua sendo o mesmo ao considerar a representação binária em complemento de dois.

Nos dois primeiros testes a seguir, referentes às operações de deslocamento, optou-se por utilizar os valores 0, 0xFFFF e 0xAAAA para o operando A, de maneira a observar como eles ficariam após deslocamentos de 0, 1, 2, 6, 0x000F e 0xFFFF casas.

Tabela 1 — Deslocamento à esquerda

Entradas			Saídas			
A	B	F	R	S_0	S_1	S_2
0	0	0	0	1	0	0
0	1	0	0	1	0	0
0xFFFF	1	0	0xFFFE	0	0	1
0xFFFF	6	0	0xFFC0	0	0	1
0xFFFF	0x000F	0	0x8000	0	0	1
0xFFFF	0xFFFF	0	0x8000	0	0	1
0xAAAA	2	0	0xAAA8	0	0	1

Tabela 2 — Deslocamento à direita

Entradas			Saídas			
A	B	F	R	S_0	S_1	S_2
0	0	1	0	1	0	0
0	1	1	0	1	0	0
0xFFFF	1	1	0x7FFF	0	0	0
0xFFFF	6	1	0x3FF	0	0	0
0xFFFF	0x000F	1	1	0	0	0
0xFFFF	0xFFFF	1	1	0	0	0
0xAAAA	2	1	0x2AAA	0	0	0

Como o valor máximo permitido para B é F (15 em decimal e 1111 em binário), valores acima deste serão ignorados, como é possível ver no teste em que A = 0xFFFF e B = 0xFFFF.

Para o XOR, foram majoritariamente escolhidos valores iguais e contrários para A e B — ou que tinham, em algum momento, conjuntos de bits opostos —, de modo a facilitar a visualização dessa operação.

Tabela 3 — XOR

Entradas			Saídas			
A	B	F	R	S_0	S_1	S_2
0	0	2	0	1	0	0
0xFFFF	0	2	0xFFFF	0	0	1
0xFFFF	0xFFF0	2	0x000F	0	0	0
0xFFFF	0xFFFF	2	0	1	0	0
0x5555	0xAAAA	2	0xFFFF	0	0	1
0xAAAA	0x5555	2	0xFFFF	0	0	1

No que tange ao complemento, também se optou por valores fáceis de visualizar/compreender o resultado final.

Tabela 4 — Complemento

Entradas		Saídas			
A	F	R	S_0	S_1	S_2
0	3	0xFFFF	0	0	1
0xFFFF	3	0	1	0	0
0x5555	3	0xAAAA	0	0	1
0xAAAA	3	0x5555	0	0	0

Em 0x5555 (representado em binário por 0101010101010101), por exemplo, o seu complemento é AAAA (representado em binário por 1010101010101010). Nesse caso, é nítida a inversão de bits.

Para as operações aritméticas, foram escolhidos os valores de A e B de modo a abranger múltiplos cenários possíveis: soma e subtração de valores positivos/negativos, tal qual soma e subtração de valores com sinais opostos. Dessa maneira, tornou-se possível analisar as situações em que o resultado é zero, positivo, negativo ou houver *overflow*.

Tabela 5 — Soma

Entradas			Saídas			
A	B	F	R	S_0	S_1	S_2
0	0	4	0	1	0	0
0xFFFF	2	4	1	0	0	0
0x5555	0x5555	4	0xAAAA	0	1	1
0xAAAA	0xAAAA	4	0x5554	0	1	0

Tabela 6 — Subtração

Entradas			Saídas			
A	B	F	R	S_0	S_1	S_2
0	0	5	0	1	0	0
0xFFFF	2	5	0xFFFD	0	0	1
0x5555	0x5555	5	0	1	0	0
0x5555	0xAAAB	5	0xAAAA	0	1	1
0xAAAA	0xAAAA	5	0	1	0	0

Tabela 7 — Incremento

Entradas		Saídas			
A	F	R	S_0	S_1	S_2
0	6	1	0	0	0
0xFFFF	6	0	1	0	0
0x5555	6	0x5556	0	0	0
0xAAAA	6	0xAAAB	0	0	1

Tabela 8 — Decremento

Entradas		Saídas			
A	F	R	S_0	S_1	S_2
0	7	0xFFFF	0	0	1
0xFFFF	7	0xFFFE	0	0	1
0x5555	7	0x5554	0	0	0
0xAAAA	7	0xAAA9	0	0	1

4 CONCLUSÃO

Após a realização de todos os testes comparando os resultados esperados com os resultados obtidos, nota-se que ambos são equivalentes, indicando que o circuito de fato funciona.

Embora tenham surgido desafios durante o desenvolvimento do projeto, estes foram superados ao longo do processo, seja por meio de testes e pesquisas, seja por meio de discussões com colegas e o professor. Vale destacar a análise de *overflow* como um desses desafios: por se tratar de uma situação mais "abstrata", foi complexo identificá-la nas operações e reconhecer em quais momentos ocorreria. De todo modo, ao término do projeto nenhum problema significativo foi observado.