

# **Pré-Projeto**

## **ZeptoProcessador-II de 16 bits**

**Marcus Paulo Kaller Vargas, 20/0041096**

**Victor Manuel Brito Santos, 20/0044184**

**Grupo 29**

<sup>1</sup>Dep. Ciência da Computação – Universidade de Brasília (UnB)  
CIC0231 - Laboratório de Circuitos Lógicos

victorvpn13@gmail.com, kmarcuspaulo@gmail.com

**Abstract.** This report contains the proposed solution for the project, activities already carried out and the difficulties faced.

**Resumo.** Este relatório contém a proposta de solução do projeto, atividades já realizadas e as dificuldades enfrentadas.

### **1. Introdução**

O projeto consiste no desenvolvimento de um processador de 16 bits, capaz de executar um programa com até 4096 instruções, e a implementação de algoritmos para o processador com objetivo de testar o mesmo e o raciocínio do aluno, com instruções em baixo nível.

#### **1.1. Objetivos**

Este projeto tem como objetivo projetar, implementar e programar um processador de 16 bits, apresentando os conceitos básicos que envolvem o projeto de processadores utilizando circuitos digitais, colocando em prática vários dos conceitos vistos nas aulas de Circuitos Lógicos [Koike and Mandelli 2021].

### **2. Metodologia Proposta**

Os blocos de Unidade Lógico-Aritmética 2.1 e do Comparador 2.2 foram os primeiros a serem projetados pois não necessitam de outros blocos para funcionar, tornando os testes mais fáceis de serem executados. Logo após projetamos os blocos que contém a Memória de Instruções 2.3, o Banco de Registradores 2.4 e por fim o Bloco de Controle 2.5. Após isso conectamos todos os blocos em um arquivo principal, que contém o sinal de Reset e de Clock, os sinais de monitoramento, o registrador *Program Counter* (PC) e o circuito que incrementa ou soma um imediato a esse registrador.

Para compilar as instruções foi criado um programa em Python que codifica as instruções e as salva em dois arquivos, um com as instruções e outro com os imediatos, com a formatação e a extensão usada pela memória do software *Deeds*.

Devido as dificuldades de se testar um programa no software *Deeds*, foi criado um outro programa em Python que simula as instruções do processador, mostrando os valores contidos nos registradores após cada instrução executada, possibilitando um melhor controle sobre o código.

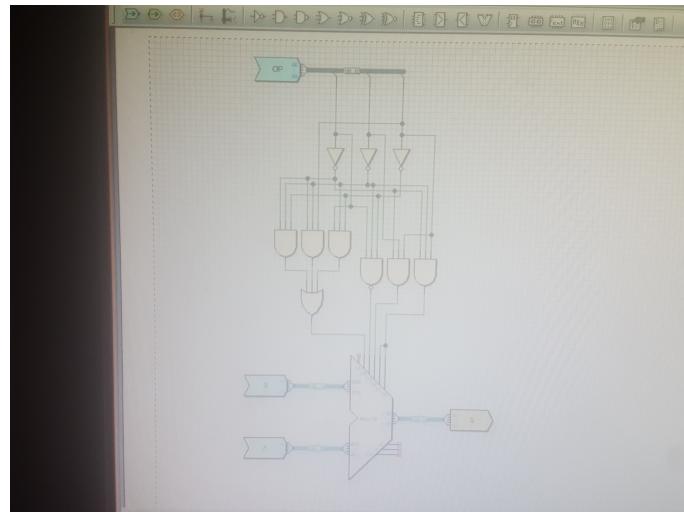
## 2.1. Unidade Lógico-Aritmética (ULA)

Este bloco recebe o *opcode* da instrução de 4 bits e dois números de 16 bits, e na saída indica o resultado da operação equivalente, também com 16 bits.

Por ser permitida a utilização da ULA que contém no software *Deeds*, foi necessário apenas um circuito combinacional para converter o *opcode* da instrução, de 4 bits, para o que é utilizado pela mesma, de 5 bits, conforme a Tabela 1.

**Tabela 1. Tabela verdade do conversor de *opcode***

Opcode				Opcode ULA				
O <sub>3</sub>	O <sub>2</sub>	O <sub>1</sub>	O <sub>0</sub>	S <sub>4</sub>	S <sub>3</sub>	S <sub>2</sub>	S <sub>1</sub>	S <sub>0</sub>
0	0	0	0	1	1	0	0	0
0	0	0	1	1	1	0	1	1
0	0	1	0	0	1	0	0	0
0	0	1	1	0	1	1	0	0
0	1	0	0	1	0	0	0	0



**Figura 1. Circuito do conversor de *opcode***

## 2.2. Comparador

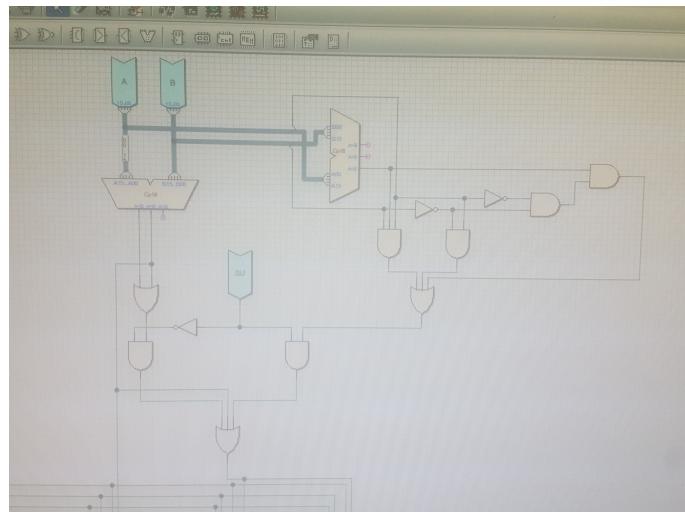
Este bloco recebe dois números de 16 bits, um sinal de controle SU, que indica se é uma comparação com sinal (SU=1) ou sem sinal (SU=0) e o *opcode* da instrução, e na saída indica se a condição é verdadeira, caso seja uma instrução de salto condicional.

O software *Deeds* contém um comparador de palavras de 16 bits, porém apenas para números sem sinal. Para fazer a comparação de números com sinal utilizando esse circuito verificamos o bit mais significativo (MSB), que é o bit de sinal. Caso o MSB de ambos for igual, podemos utilizar a comparação com sinal, se o MSB dos números forem diferentes, o que tiver o dígito mais significativo igual a 0 é o maior, conforme podemos

ver na Tabela 2, onde *Comparação Padrão* equivale ao circuito do comparador presente no software *Deeds*.

**Tabela 2. Tabela verdade da conversão de opcode**

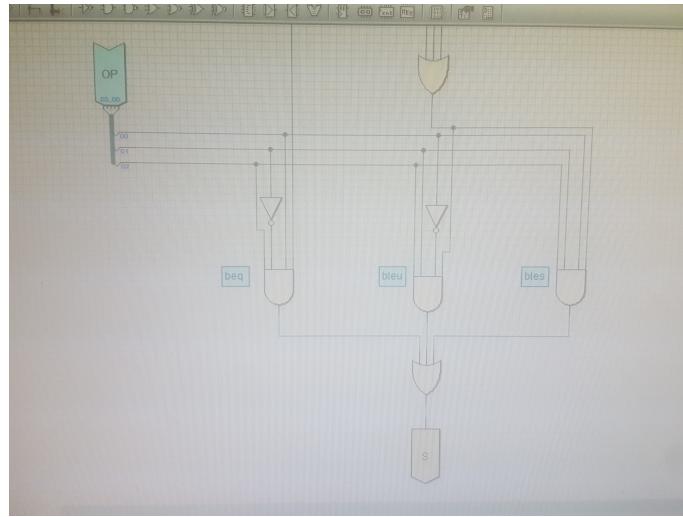
Entradas		Saída
A <sub>15</sub>	B <sub>15</sub>	A < B
0	0	Comparação Padrão
0	1	0
1	0	1
1	1	Comparação Padrão



**Figura 2. Circuito de comparação de números com e sem sinal**

Para as instruções **bleu** e **bles** foi necessário colocar uma porta OR com a saída de igualdade do comparador, que não é influenciada pelo sinal.

Por fim há um circuito combinacional que verifica se é uma instrução de salto e a associa ao resultado da verificação equivalente, conforme a Figura 3.



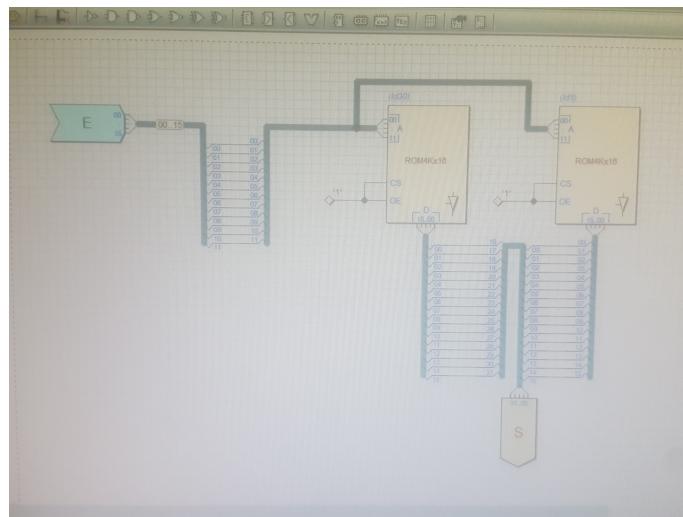
**Figura 3. Circuito de verificação de salto condicional**

### 2.3. Memória de Instruções

Este bloco consiste em uma memória do tipo ROM, que recebe um endereço de 16 bits e na saída retorna a instrução equivalente, de 32 bits.

Foram utilizados dois módulos da memória ROM de 4K x 16 bits presente no software *Deeds*. Como o endereço havia apenas 12 bits, consideramos apenas os 12 bits menos significativos do endereço recebido.

Para formar a saída, os 16 bits menos significativos são equivalentes ao conteúdo da memória com as instruções e os outros 16 bits equivalem ao conteúdo da memória que contém os imediatos.



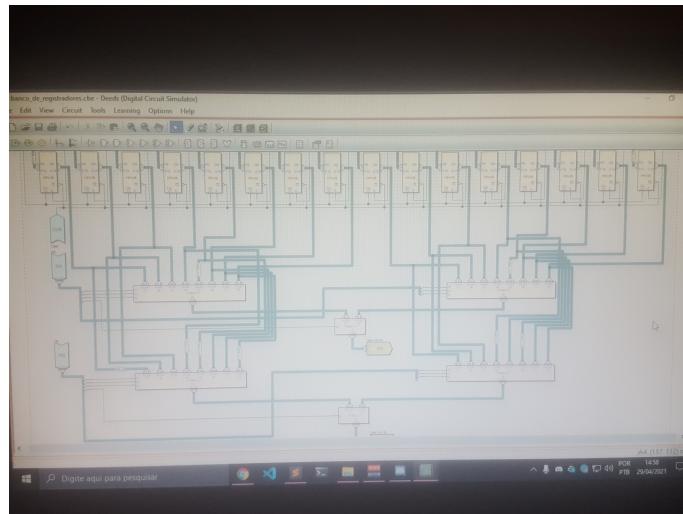
**Figura 4. Circuito do bloco que contém a memória ROM**

### 2.4. Banco de Registradores

Este componente consiste em um bloco contendo 16 registradores de 16 bits cada, onde é possível a leitura e escrita em cada um individualmente, um sinal para indicar escrita e

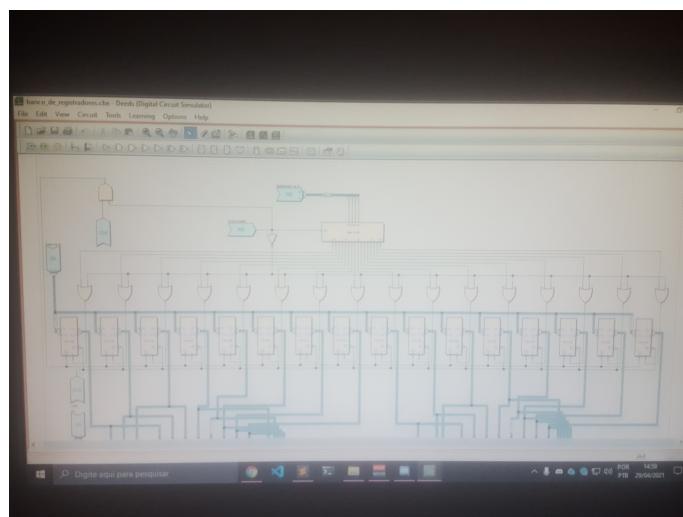
outro para limpar os valores de todos eles.

Para leitura utilizamos dois multiplexadores 8x1 e um multiplexador 2x1, para selecionar individualmente de acordo com cada um dos 16 endereços possíveis. Caso o sinal de escrita seja falso (WE=0), todos os registradores são ligados e a seleção da saída é feita por meio do endereço recebido em Ra na subida do Clock. O mesmo circuito foi duplicado para ler o valor do registrador Rb simultaneamente.



**Figura 5. Circuito de leitura do Banco de Registradores**

Para escrita utilizamos um decodificador 4x16, que recebe o endereço do registrador de escrita Rd e o dado de escrita. Caso o sinal de escrita seja verdadeiro (WE=1), somente o registrador do endereço de escrita é ativado e, durante a subida do Clock, o dado de escrita é armazenado no registrador com endereço Rd.



**Figura 6. Circuito de escrita do Banco de Registradores**

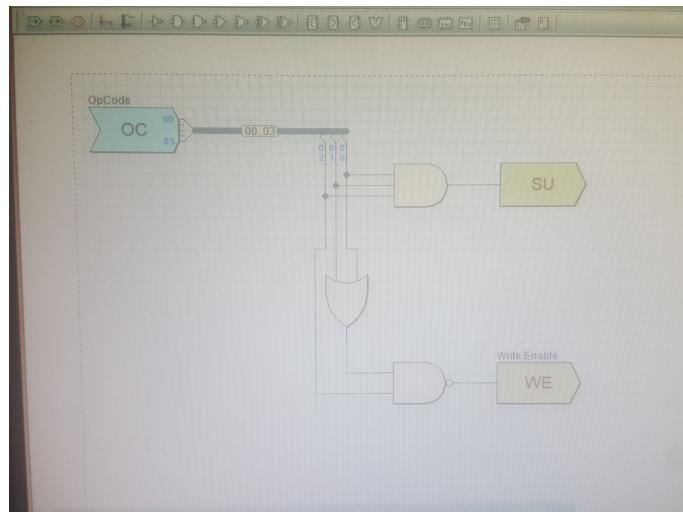
O sinal para limpar os valores dos registradores é ligado ao pino CLEAR dos mesmos, e caso ele seja verdadeiro (CLEAR=0), os valores são redefinidos.

## 2.5. Bloco de Controle (BC)

Este bloco consiste em um circuito combinacional que recebe o *opcode* da instrução e indica nas saídas os sinais de controles: *Write Enable* (WE), que indica para o **Banco de Registradores 2.4** caso seja uma instrução que requer escrita nos registradores, e a saída SU, que indica se a operação considera números com (SU=1) ou sem sinal (SU=0).

**Tabela 3. Tabela verdade dos sinais de controle**

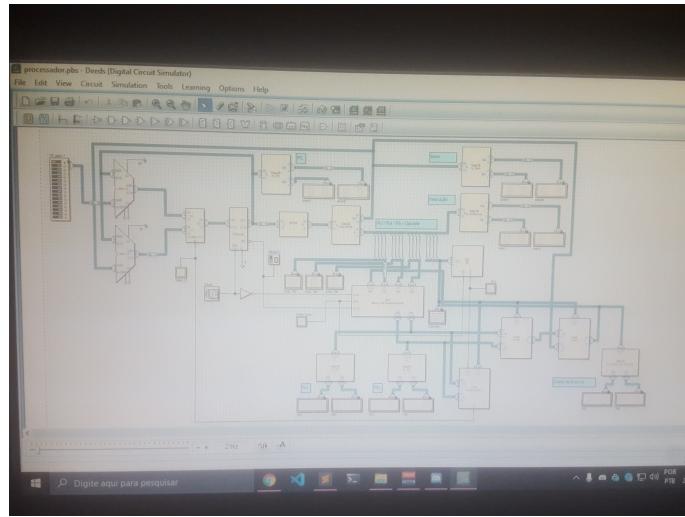
Instrução	Opcode				Saídas		
	Mnemônico	O <sub>3</sub>	O <sub>2</sub>	O <sub>1</sub>	O <sub>0</sub>	WE	SU
addi	0	0	0	0	0	1	X
subi	0	0	0	1	1	1	X
andi	0	0	1	0	1	1	X
ori	0	0	1	1	1	1	X
xori	0	1	0	0	1	1	X
beq	0	1	0	1	0	0	X
bleu	0	1	1	0	0	0	0
bles	0	1	1	1	0	0	1



**Figura 7. Circuito do Bloco de Controle**

## 2.6. Circuito Principal do Processador

Neste arquivo unimos todos os blocos de acordo com suas respectivas funções, adicionamos o registrador *Program Counter* (PC) e o circuito que incrementa ou soma um imediato a esse registrador, de acordo com o sinal recebido pelo bloco do comparador, além dos sinais de monitoramento por displays de 7 segmentos, 1 e de 4 bits, o sinal de Clock e o de Reset, para limpar todos os registradores.



**Figura 8. Circuito principal do processador**

### 3. Conclusão

No decorrer do processo de implementação surgiram algumas dificuldades, as principais foram: testar os blocos individualmente, compilar um programa e testar um programa. Para testar os blocos individualmente criamos um outro arquivo para cada elemento com um circuito contendo o próprio bloco e as entradas manuais necessárias. Para compilar as instruções foi criado um programa em Python que codifica as instruções.

Para testar um programa no software *Deeds* é necessário carregar os dois arquivos nas memórias manualmente e logo após remover e inserir o circuito da Memória novamente no arquivo principal. Logo após, na simulação interativa, apenas simulamos o Clock alternando seu estado. A simulação por forma de onda é bem lenta, mesmo sendo executada em um computador com bom desempenho, então sempre optamos pela primeira forma de simulação, que pode se tornar maçante em um programa com muitas instruções. Para solucionar esse problema foi criado um programa em Python que simula as instruções do processador partindo do programa escrito com os Mnemônicos, possibilitando o monitoramento de todos os registradores a cada subida de Clock, testando assim a lógica do programa.

Então, para a finalização do projeto, é necessário concluir a gravação da simulação dos programas propostos e o desenho do diagrama temporal da execução dos mesmos na maior frequência.

### Referências

- [Koike and Mandelli 2021] Koike, C. and Mandelli, M. (2021). *Logic Circuits*. 1st edition. [Online; accessed 23-April-2021].