

UNIVERSIDADE DE SÃO PAULO - CAMPUS SÃO CARLOS  
INSTITUTO DE CIÊNCIAS MATEMÁTICAS E DE COMPUTAÇÃO

Disciplina: Prática em Sistemas Digitais - Turma 4  
Alunos: João Guilherme Madeira Araújo - 9725165  
Luísa Souza Moura - 10692179

## Unidade Central de Processamento

### 1 - Introdução

O projeto teve como objetivo o desenvolvimento de uma Unidade Central de Processamento (CPU) no software Quartus, contendo unidade de controle, registradores, contador, memória, e outros circuitos relevantes. Seu funcionamento consiste na interpretação e execução de um arquivo de memória (do tipo .mif) fornecido previamente.

A elaboração do projeto teve como base o diagrama abaixo (Figura 1), que mostra a hierarquia dos componentes da CPU.

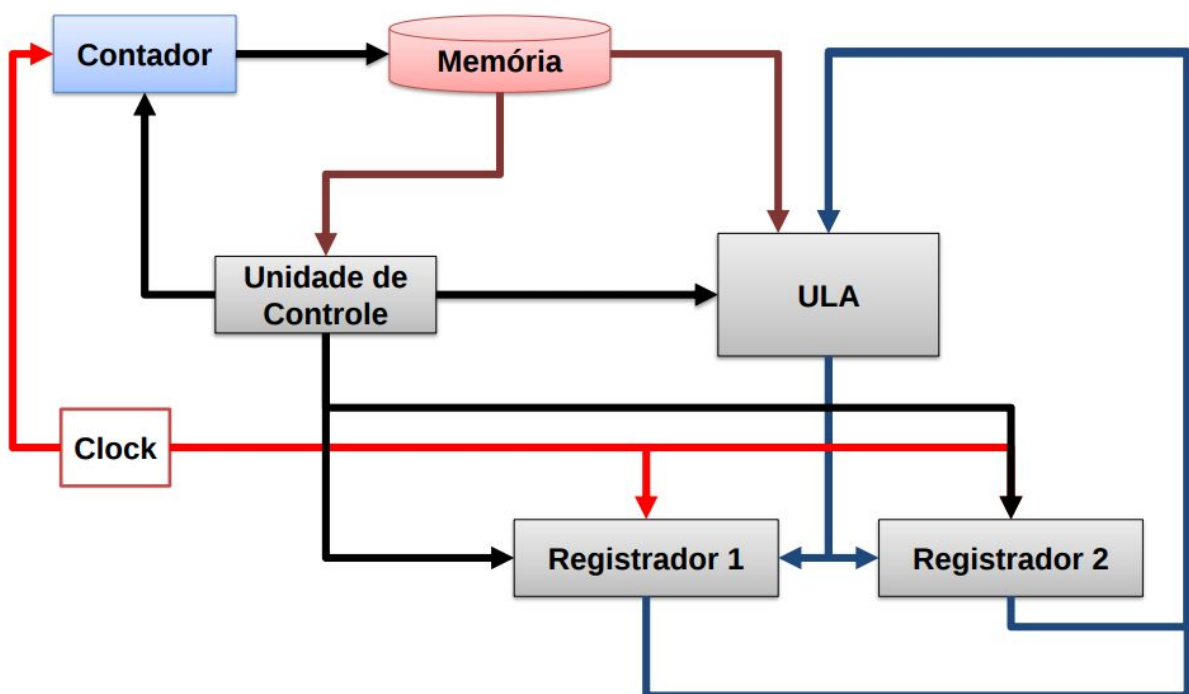


Figura 1 - Diagrama da CPU

Podemos ver na Figura 2, o formato da memória e os valores de cada operação, que serão selecionadas pela UC, executados pela ULA e armazenados nos registradores de acordo ao pulso de clock.

15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00
RgTO				RgIN		JMP		ULA		Disponível			Operando		
<ul style="list-style-type: none"><li>• <b>RgTO</b>: Registrador de destino (4 bits)</li><li>• <b>RgIN</b>: Registrador de origem (4 bits)</li><li>• <b>JMP</b>: uma das 4 opções abaixo:<ul style="list-style-type: none"><li>• 00: Operação de ULA;</li><li>• 01: Reinicia Registradores;</li><li>• 10: Reinicia o contador de memória (reset PC);</li><li>• 11: Jump para posição de memória do <b>Operando</b>;</li></ul></li><li>• <b>ULA</b>: uma das 4 operações abaixo:<ul style="list-style-type: none"><li>• 00: <math>RgTO \leftarrow RgIN + Operando</math>;</li><li>• 01: <math>RgTO \leftarrow Operando * 2</math>;</li><li>• 10: <math>RgTO \leftarrow RgIN - Operando</math>;</li><li>• 11: <math>RgTO \leftarrow Operando / 2</math>;</li></ul></li><li>• <b>Disponível</b>: Disponível para melhorar a CPU caso queiram</li><li>• <b>Operando</b>: 4 bits diretamente da memória na ULA</li></ul>															

Figura 2 - Formato da memória

## 2 - Circuitos

O circuito desenvolvido pode ser observado na Figura 3.

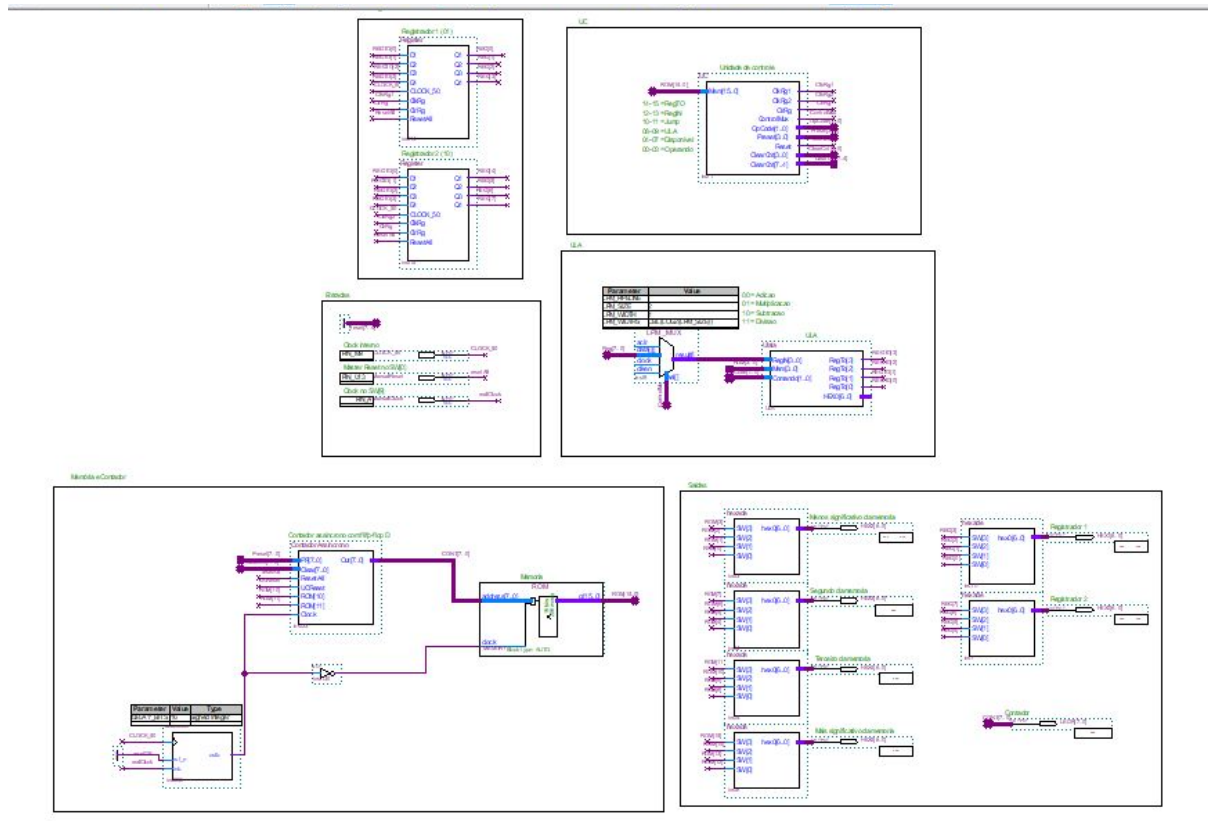


Figura 3 - Circuito completo da CPU

Os detalhes de implementação e funcionamento de cada componente serão abordados nos tópicos a seguir.

## 2.1 - Entradas

O circuito como um todo possui apenas duas entradas manuais e uma externa. O CLOCK\_50 é o clock interno da placa utilizada, sendo necessária para utilização do debouncer e do próprio clock manual.

Além disso, há o ManualReset (SW[0]), responsável por zerar todos os valores do contador e os registradores. Seu funcionamento se dá de forma assíncrona, ou seja, os valores são obrigatoriamente resetados no próximo sinal de clock.

Por fim, temos o ManualClock (SW[9]) que é o pulso de clock fornecido manualmente para que o contador e a memória funcionem.

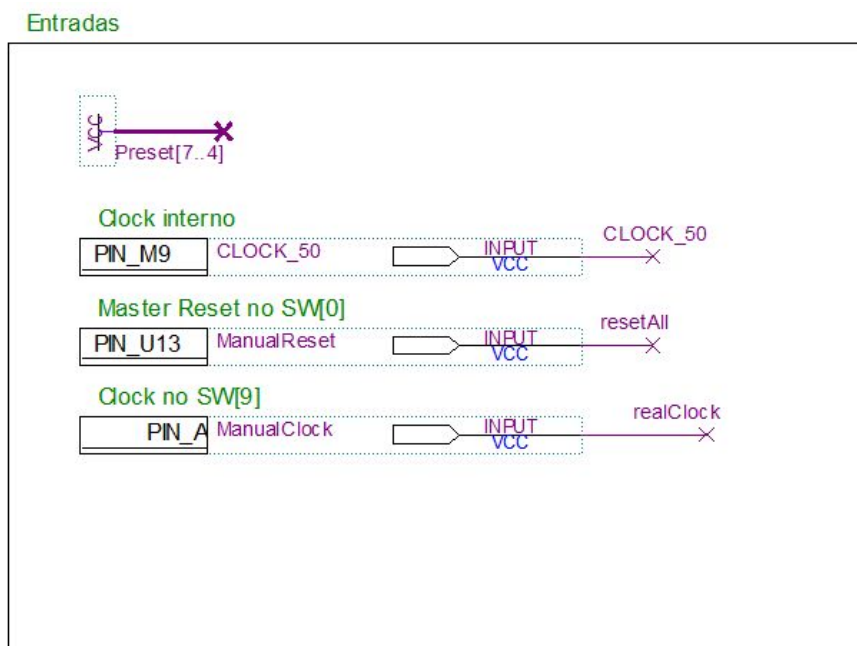


Figura 4 - Entradas da CPU

## 2.2 - Saídas

Os circuitos de saída são 6 conversores hexadecimal para displays de 7 segmentos e 8 leds.

Os displays de 7 segmentos representam, os 16 bits da memória (HEX5 a HEX2), e os outros dois (HEX1 e HEX0) são os valores dos registradores, conforme a ordem prevista e mostrada da figura 2.

Os leds representam o valor do contador, que vai de 0 a 255, sendo o LEDR[7] o bit mais significativo.

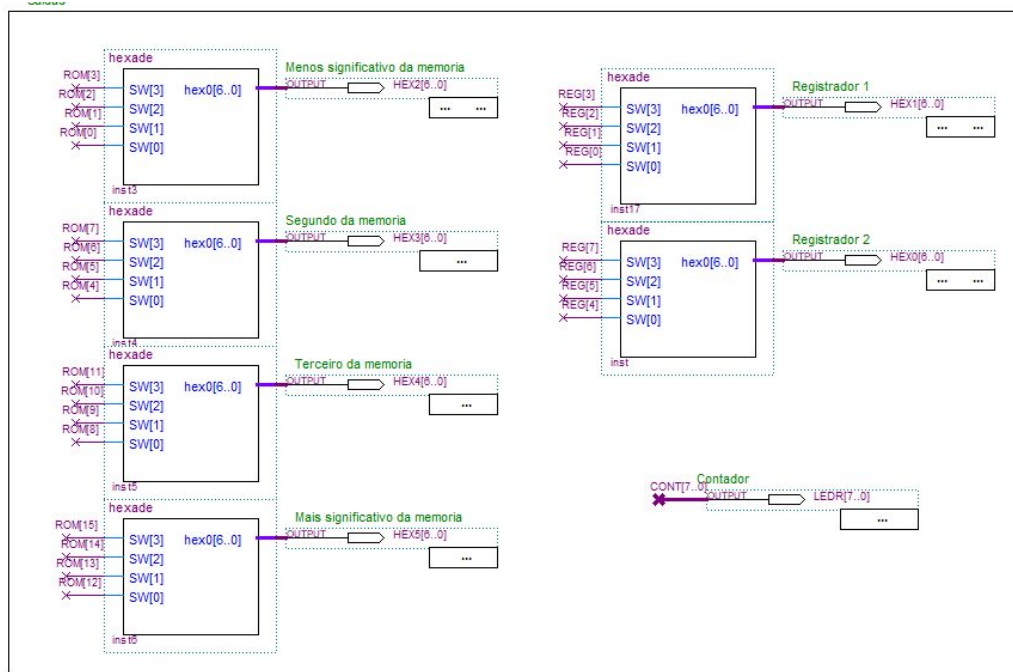


Figura 5 - Saídas do circuito

## 2.3 - ULA

A Unidade Lógica Aritmética (ULA), foi utilizada para realizar operações aritméticas dentro da CPU. De acordo à entrada, é feita uma das seguintes operações: adição, subtração, multiplicação ou divisão, a qual é selecionada por um multiplex (LPM\_MUX), circuito pronto do Quartus.

As entradas do bloco da ULA são:

- RegIn[3..0]: 4 bits que representam o valor do registrador de entrada, o qual é selecionado pela Unidade de Controle, a partir de dados da memória.
- Mem[3..0]: 4 bits que representam o operador presente na memória.
- Comando[1..0]: 2 bits que representam a instrução da memória para a ULA, ou seja, qual operação deve ser executada. Na prática, há um outro multiplex na ULA que utiliza esses dados de comando para selecionar a saída adequada.

As saídas do bloco são:

- RegTo[3..0]: 4 bits que representam o valor que deve ir para o registrador de destino, o qual será selecionado pela UC.
- HEX[6..0]: saída do valor em hexadecimal, presente no bloco, mas não utilizado no projeto

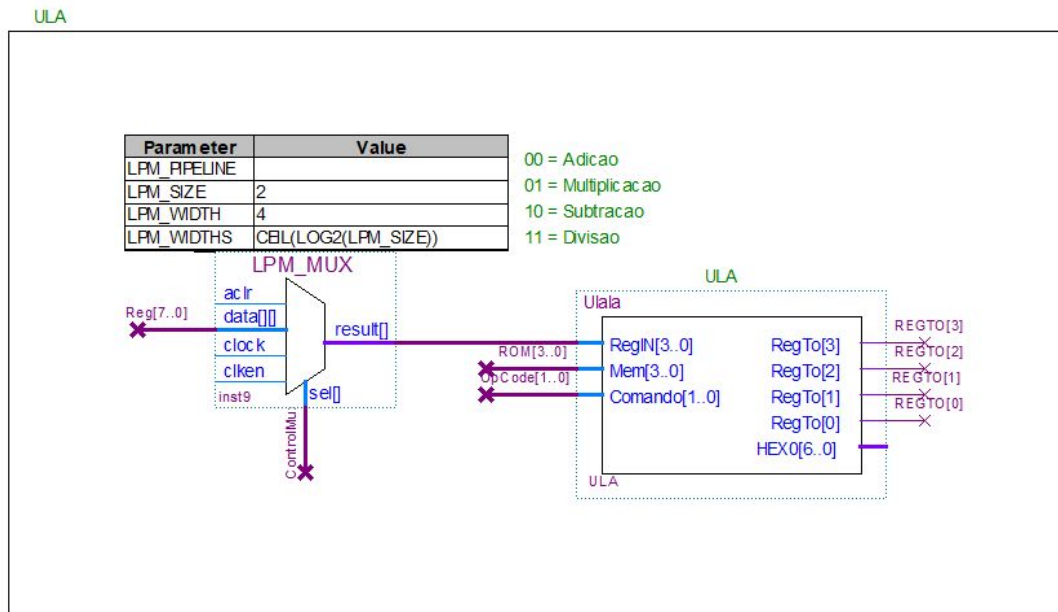


Figura 6 - Unidade Lógica Aritmética

## 2.4 - Registradores

Os registradores são usados para guardar resultados das operações da ULA. Seu clock é controlado pela UC, assim, pode ser controlado qual dos registradores vai receber o resultado da operação. Suas saídas são enviadas para um multiplexer (lpm-mux) que seleciona qual dos registradores será usado na próxima operação da ULA.

Suas entradas são:

- RegTo[3..0]: 4 bits que são a saída da ULA e representam o valor a ser armazenado no registrador.
- Outb: saída do debouncer, clock interno da placa
- ClkRg: clock de cada registrador. Esse bit é uma saída da UC e indica se o respectivo registrador deve ser utilizado. Caso tal registrador esteja indicado pela memória como sendo o registrador de destino do valor da ULA, seu clock será 1, caso contrário, será 0.
- ClrRg: bit que indica se valor armazenado no registrador deve ser zerado. Esse sinal também vem na UC.
- ResetAll: bit de entrada manual responsável por zerar todos os valores do contador e da memória.

Já as saídas são:

- Reg[3..0] e Reg[7..4]: 4 bits que indicam o valor de saída de cada registrador. O valor de saída real é escolhido por um multiplex de acordo à memória.

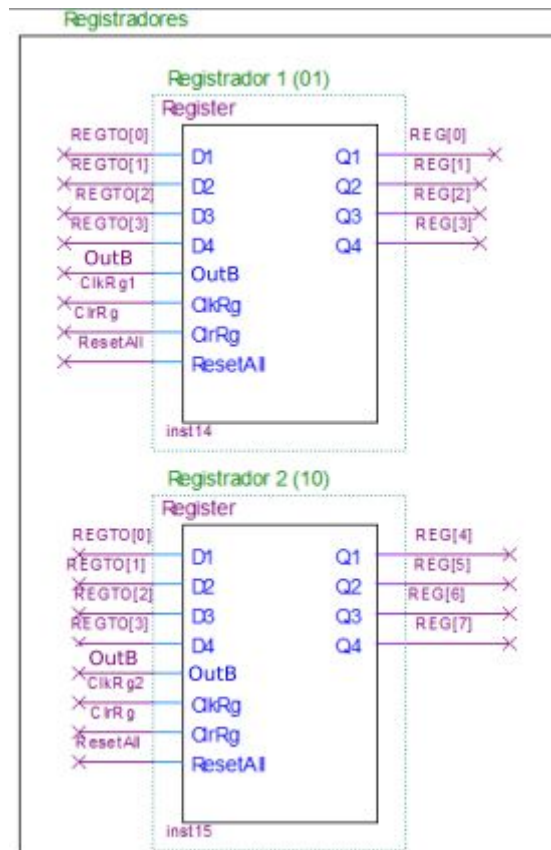


Figura 7 - Blocos dos registradores

Internamente, os registradores são compostos por flip-flops do tipo D. Seu clock é a saída de uma porta AND do clock interno com o ClkRg, uma vez que o sinal só deve ser propagado quando ambos forem altos. Já o clear pode ser feito tanto num sinal de ClrRg (sinal da UC) quanto com o sinal manual de reset (resetAll). O preset é ligado no vcc, pois funciona em sinal baixo e nunca deve ser ativo.

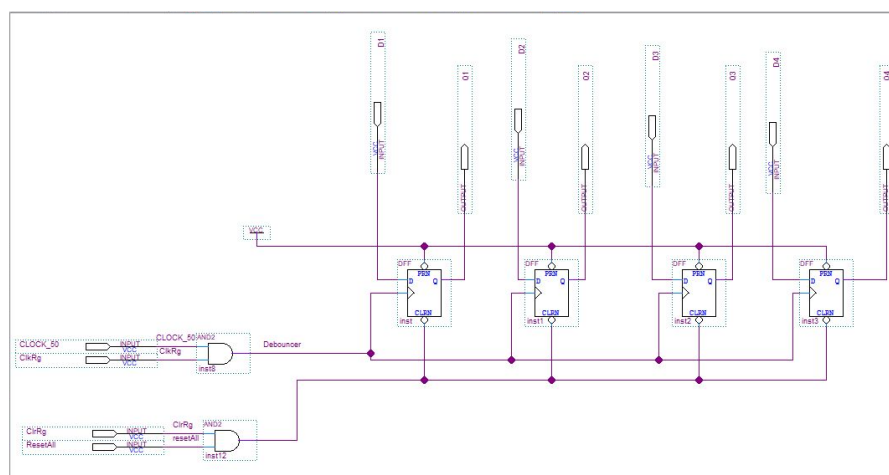


Figura 8 - Registrador



## 2.5 - Contador

Foi feito, ao decorrer do projeto, um contador síncrono e um assíncrono. No final foi utilizado o contador assíncrono, que pode ser conferido na imagem abaixo.

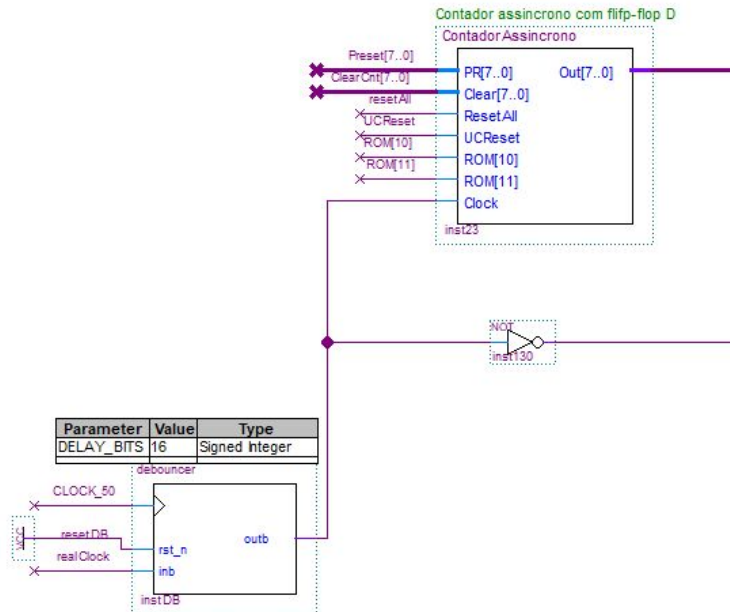


Figura 9 - Contador e debouncer

Esse contador tem, como entradas:

- Preset[7..0]: 8 bits que representam a posição dos bits que devem ser forçados para 1. Esse sinal é assíncrono, ou seja, ao receberem 0, os bits do contador assumem tal valor. Os quatro bits mais significativos sempre recebem sinal alto, como pode ser vista na figura 4, pois o jump ocorre apenas para valores de 0 a 15. Os bits menos significativos são selecionados pela UC.
- ClearCnt[7..0]: funcionam de forma similar aos bits do preset, porém o sinal é forçado para 0.
- resetAll: bit de entrada manual responsável por zerar todos os valores do contador e da memória.
- UCRReset: bit que vem da UC e é um sinal que indica se o contador deve ou não ser zerado.
- ROM[11..10]: bits da memória que representam a instrução de jump. São utilizados internamente para informar ao contador se o clock deve ou não ser considerado. Caso haja uma instrução de jump, o clock não é considerado.

A saída é apenas um valor de 8 bits, que é justamente o número atual da contagem.



O circuito interno do contador é o seguinte:

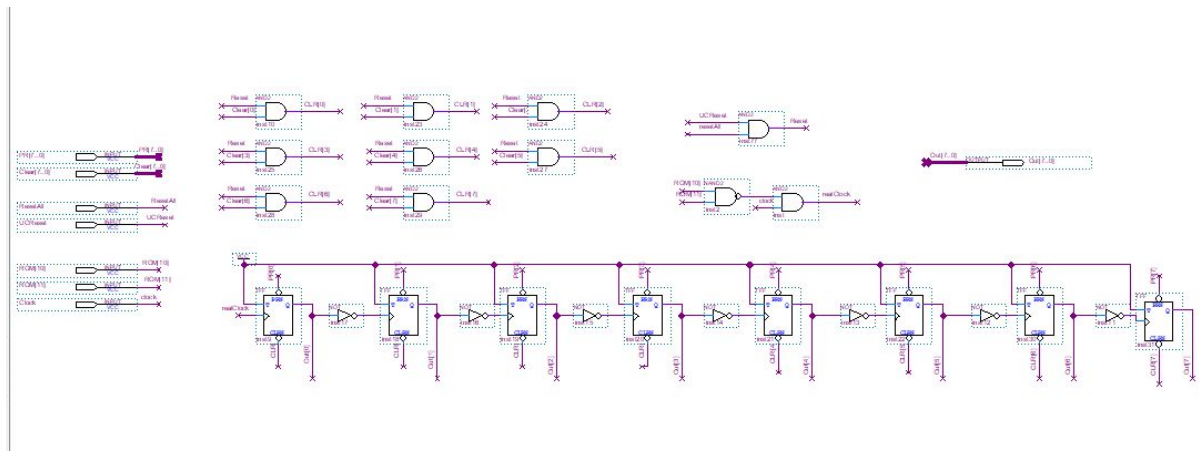


Figura 10 - Circuito interno do contador

As operações lógicas indicam se o há algum valor a ser forçado no contador. Há também um circuito para indicar o clock, conforme já foi descrito e um para o reset.

## 2.6 - Memória

A memória é um bloco pronto do Quartus. Utilizamos uma memória até 256 instruções de 16 bits. Nesse bloco, há entrada do contador e do clock que vem do debouncer. Sua saída (ROM[15..0]) vai para a UC, que é responsável por interpretá-la e executar as respectivas instruções.

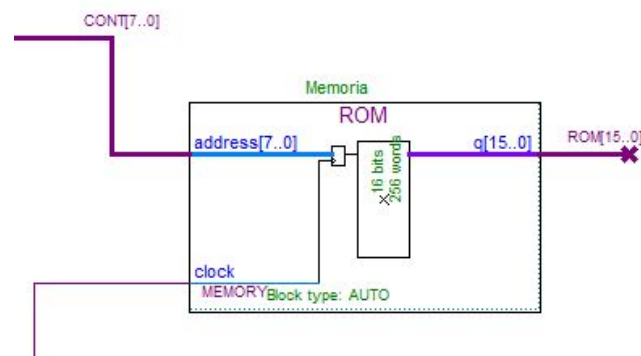


Figura 11 - Memória

## 2.5 - Unidade de Controle

A Unidade de Controle(UC), foi utilizada para interpretar a memória e coordenar os demais circuitos da CPU. A memória indexada pelo contador é mandada como entrada, junto de um dos registradores, selecionados por um multiplexer, como saída temos:

- a operação que a ULA deverá executar,
- Um sinal que indica qual registrador deve receber o clock,
- Preset e Clear para o contador, que são ativados em caso de Jump ou reset,
- Clear dos registradores, que deve fazer os registradores valerem 0;

A única entrada da UC é a saída memória e suas saídas são:

- ClkRg1: indica se o registrador 1 vai ser destino desta instrução.
- ClkRg2: indica se o registrador 2 vai ser destino desta instrução.
- ClrRg: indica se os registradores devem ser zerados.
- ControlMux: indica qual saída dos registradores será utilizada.
- OpCode: código da operação da ULA.
- Preset[3..0]: valor para o qual serão forçados os bits menos significativos do contador no momento do jump.
- ClearCnt[3..0]: valor para o qual serão forçados os bits menos significativos do contador no momento do jump.
- ClearCnt[7..4]: envia sinal baixo quando há um jump, pois os quatro bits mais significativos devem ser forçados para 0.
- UCReset: indica se o contador deve ser resetado.

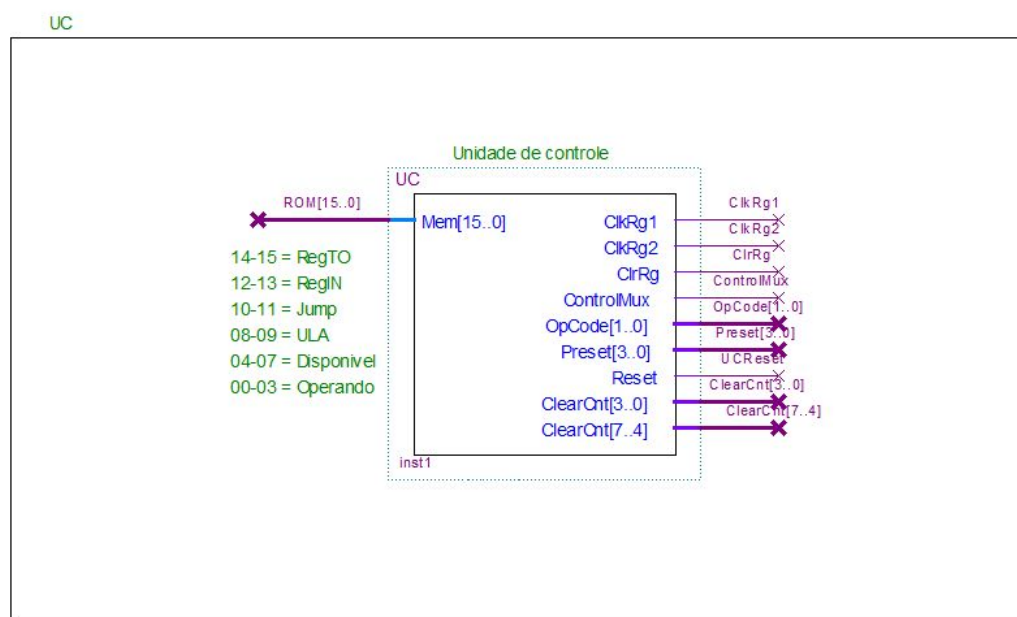


Figura 12 - Unidade de controle

Por dentro, o bloco da UC tem a seguinte configuração:

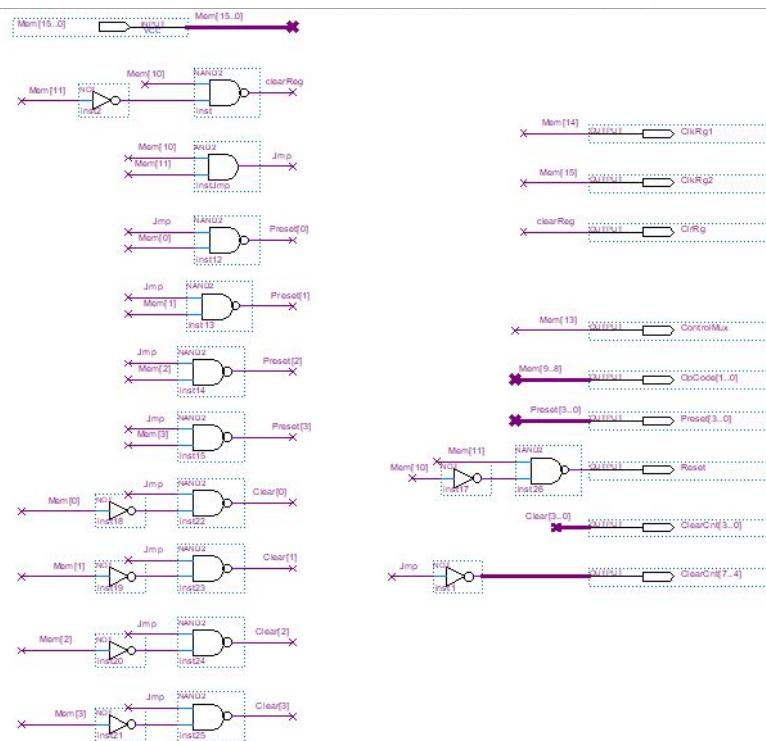


Figura 13 - UC por dentro

### 3 - Clock

O clock rege a sincronia das instruções, do contador e dos registradores. Para que haja um funcionamento mais eficiente e adequado da UC, é preciso que cada parte do circuito funcione em uma borda específica. Para utilizar o clock na FPGA, é preciso de um bloco debouncer.

O contador e os registradores funcionam em borda de subida e a memória funciona em borda de descida. Dessa forma, ao enviar um sinal de clock alto, o contador é incrementado e o valor de cada registrador é atualizado. Já no sinal baixo, a memória é atualizada e a próxima instrução é carregada.

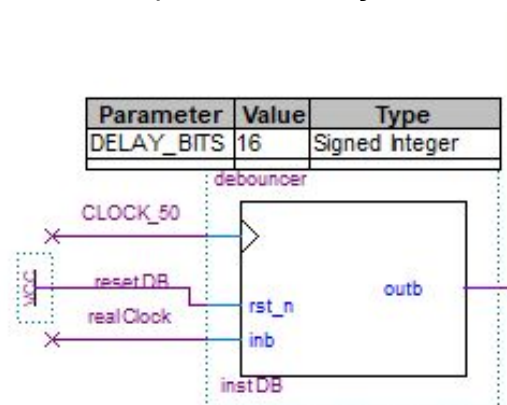
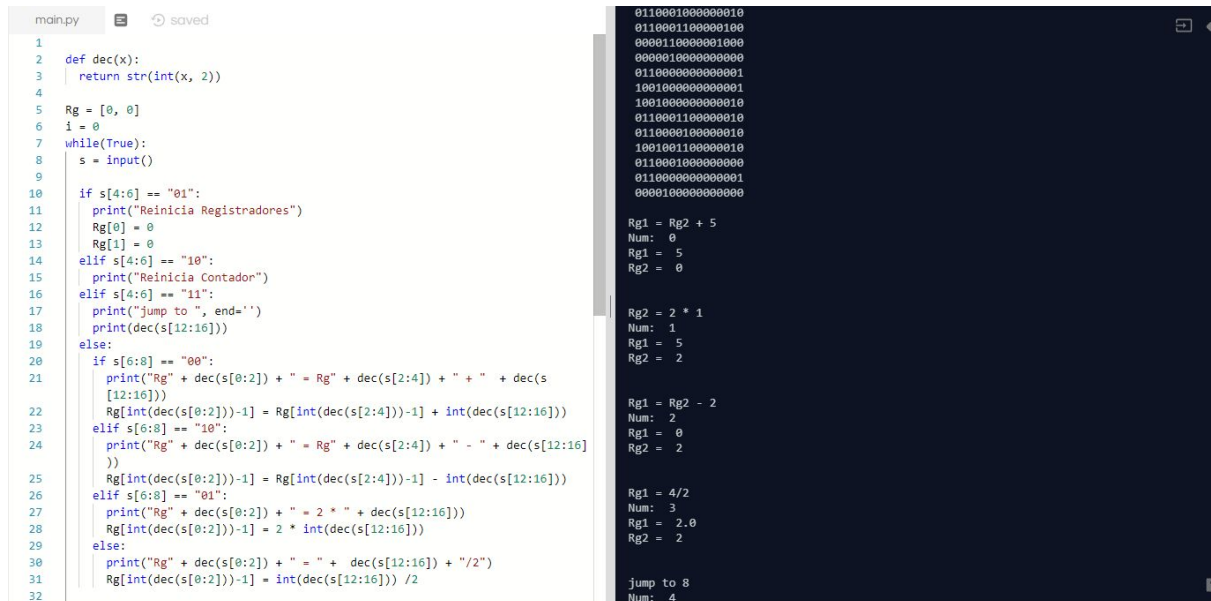


Figura 14 - Debouncer

## 4 - Extra

Além do circuito, foi desenvolvido um código em python3 para auxiliar a interpretação da memória. Ele recebe como entrada os bits da memória e imprime a função que deve ser feita e os valores esperados para cada registrador naquele momento.



```
main.py saved
1
2 def dec(x):
3     return str(int(x, 2))
4
5 Rg = [0, 0]
6 i = 0
7 while(True):
8     s = input()
9
10    if s[4:6] == "01":
11        print("Reinicia Registradores")
12        Rg[0] = 0
13        Rg[1] = 0
14    elif s[4:6] == "10":
15        print("Reinicia Contador")
16    elif s[4:6] == "11":
17        print("jump to ", end='')
18        print(dec(s[12:16]))
19    else:
20        if s[6:8] == "00":
21            print("Rg" + dec(s[0:2]) + " = Rg" + dec(s[2:4]) + " + " + dec(s[12:16]))
22            Rg[int(dec(s[0:2]))-1] = Rg[int(dec(s[2:4]))-1] + int(dec(s[12:16]))
23        elif s[6:8] == "10":
24            print("Rg" + dec(s[0:2]) + " = Rg" + dec(s[2:4]) + " - " + dec(s[12:16]))
25            Rg[int(dec(s[0:2]))-1] = Rg[int(dec(s[2:4]))-1] - int(dec(s[12:16]))
26        elif s[6:8] == "01":
27            print("Rg" + dec(s[0:2]) + " = 2 * " + dec(s[12:16]))
28            Rg[int(dec(s[0:2]))-1] = 2 * int(dec(s[12:16]))
29        else:
30            print("Rg" + dec(s[0:2]) + " = " + dec(s[12:16]) + "/2")
31            Rg[int(dec(s[0:2]))-1] = int(dec(s[12:16])) / 2
32
0110001000000010
0110001100000100
0000110000001000
0000010000000000
0110000000000001
1001000000000001
1001000000000010
0110001100000010
0110000100000010
1001001100000010
0110001000000000
0110000000000001
0000100000000000

Rg1 = Rg2 + 5
Num: 0
Rg1 = 5
Rg2 = 0

Rg2 = 2 * 1
Num: 1
Rg1 = 5
Rg2 = 2

Rg1 = Rg2 - 2
Num: 2
Rg1 = 0
Rg2 = 2

Rg1 = 4/2
Num: 3
Rg1 = 2.0
Rg2 = 2

jump to 8
Num: 4
```

Figura 15 - Código interpretador

## 5 - Dificuldades

Houve uma intrigante dificuldade na realização do projeto. Um problema de sincronia e de propagação de clock. Ao ser executado um jump, um valor X é forçado no contador, e ao receber um novo clock, o contador ainda não deixou de ser forçado, então o contador não assume o valor que ele deveria assumir. Tal questão não foi resolvida pois nem os monitores nem o professor conseguiram encontrar a origem do problema.

## 6 - Conclusão

Concluimos, assim, que apesar das dificuldades, foi possível desenvolver uma CPU operante, capaz de realizar operações indicadas pela memória.