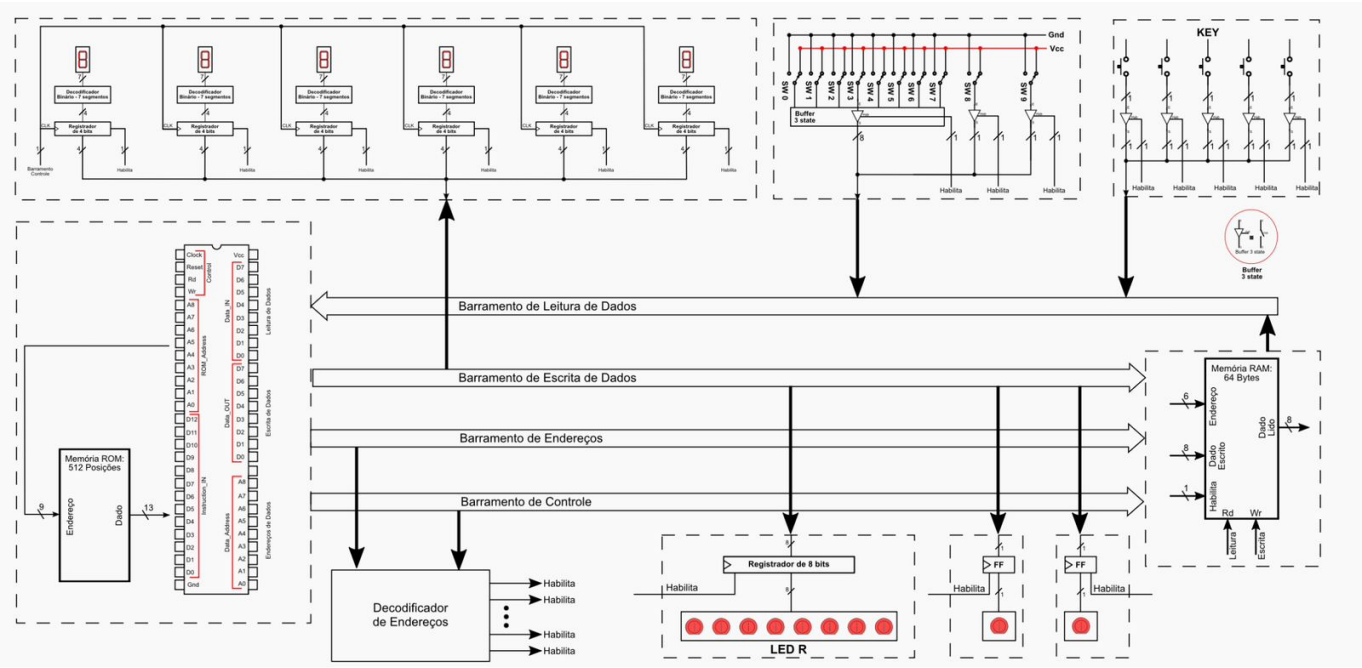
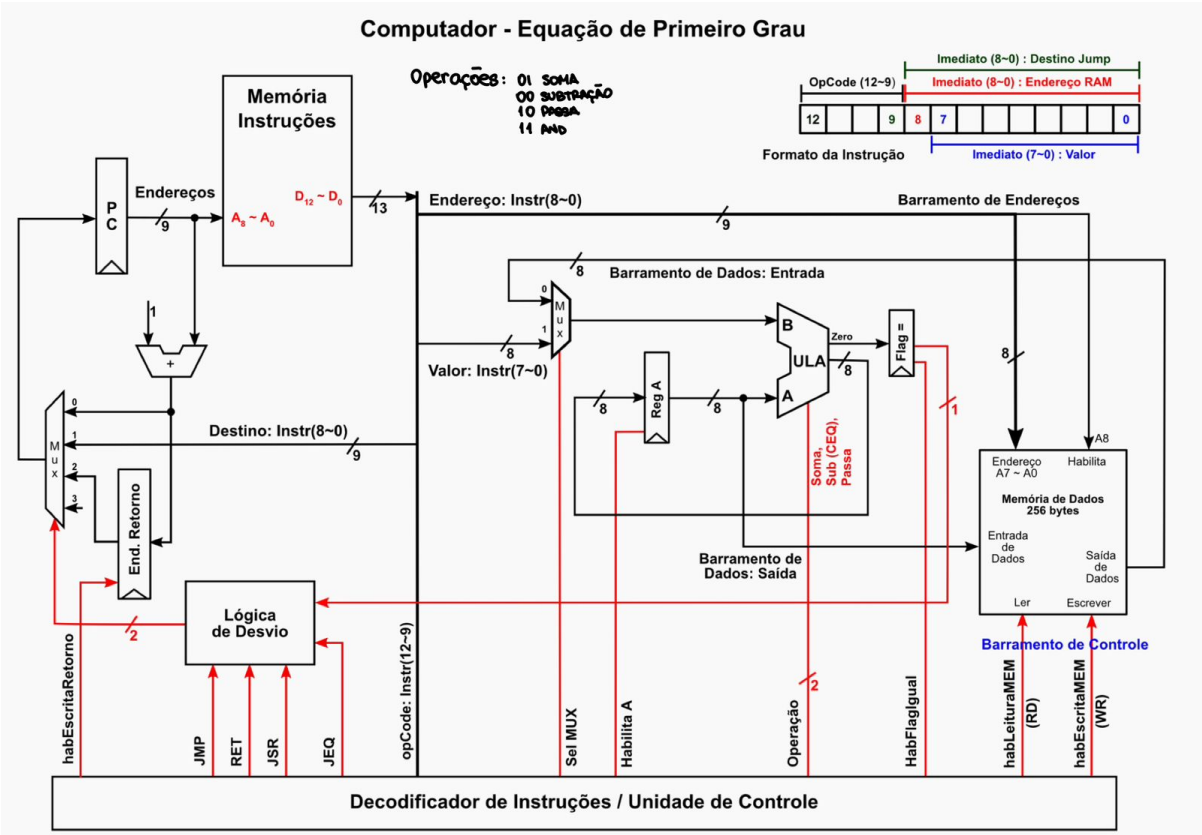
# Projeto de Contador Programado em VHDL

Este projeto consiste em um contador programado em VHDL que implementa uma CPU simples e sua integração em uma placa FPGA. O objetivo é demonstrar o funcionamento de instruções personalizadas em assembly e a interação com os componentes da placa.

## Diagrama do circuito




CPU mais detalhada



## Declaração sobre a divisão do trabalho

O projeto foi desenvolvido em parceria por **Breno Schneider** e **Thiago Victoriano**. Ambos trabalharam separadamente na montagem do hardware, optando por utilizar como base o projeto do Thiago, uma vez que ele iniciou as implementações primeiro. O código principal em assembly foi elaborado em conjunto, com divisões básicas dentro de cada função de cada parte do código. O assembler foi desenvolvido pelo Thiago, enquanto as novas funcionalidades do hardware específicas para o projeto foram implementadas pelo Breno.

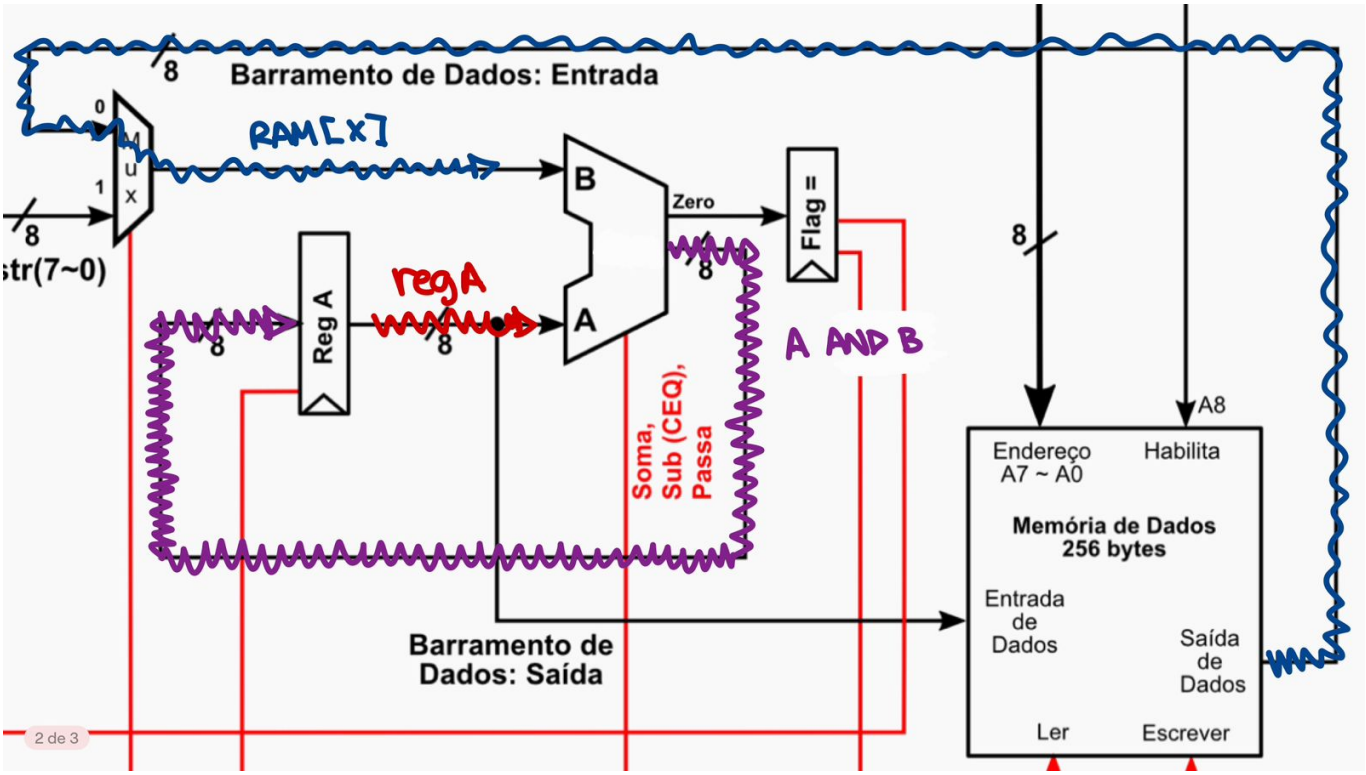
## Funcionamento dos botões e chaves

- **SW3 a SW0**: Determinam o limite da contagem. Ao pressionar o botão **KEY1**, o valor das chaves é definido, incrementando de unidade, dezena, centena até centena de milhar.
- **KEY0**: Incrementa a contagem.
- **KEY2**: Decrementa a contagem.
- **FPGA_RESET**: Reinicia a contagem.
- **SW9 a SW4**: Não possuem funcionalidade no projeto atual.
- **KEY3**: Não possui funcionalidade no projeto atual.

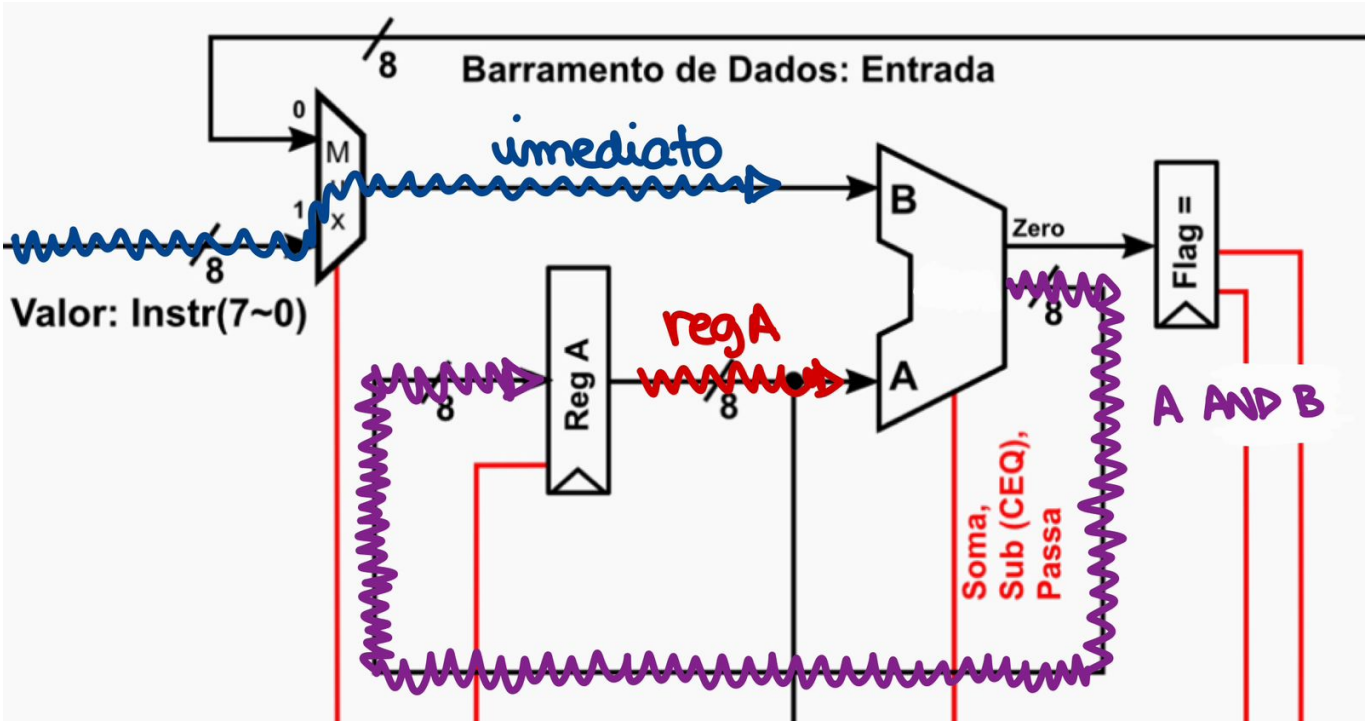
## Novas funções implementadas

Foram implementadas as seguintes novas instruções no assembly:

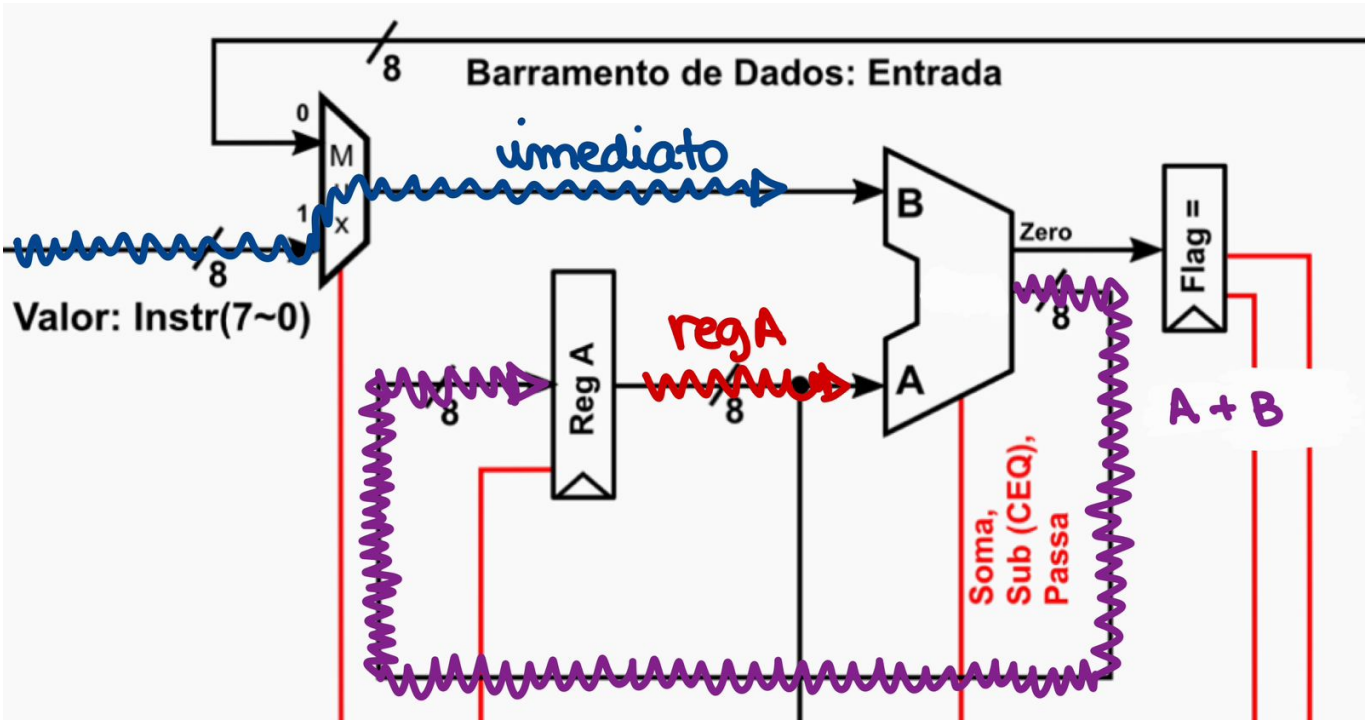
- **AND**: Realiza a operação lógica AND entre o registrador A e o conteúdo de um endereço de memória especificado.
  - **Sintaxe**: `AND @<endereço>`
  - **Exemplo**: `AND @4` (Realiza o AND entre o registrador e o conteúdo do endereço 4 da memória)



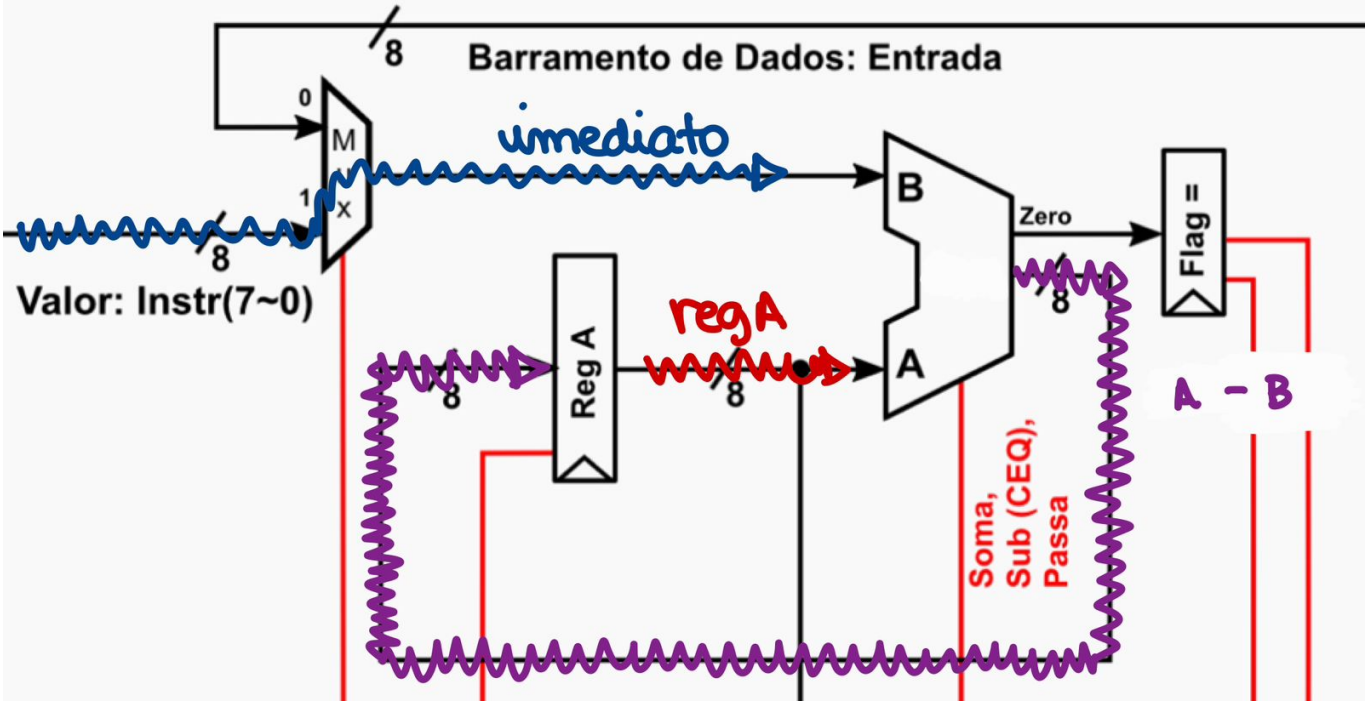

- **ANDi**: Realiza a operação lógica AND entre o registrador A e um valor imediato.
  - **Sintaxe**: `ANDi $<valor_imediato>`
  - **Exemplo**: `ANDi $3` (Realiza o AND entre o registrador e o valor 3)



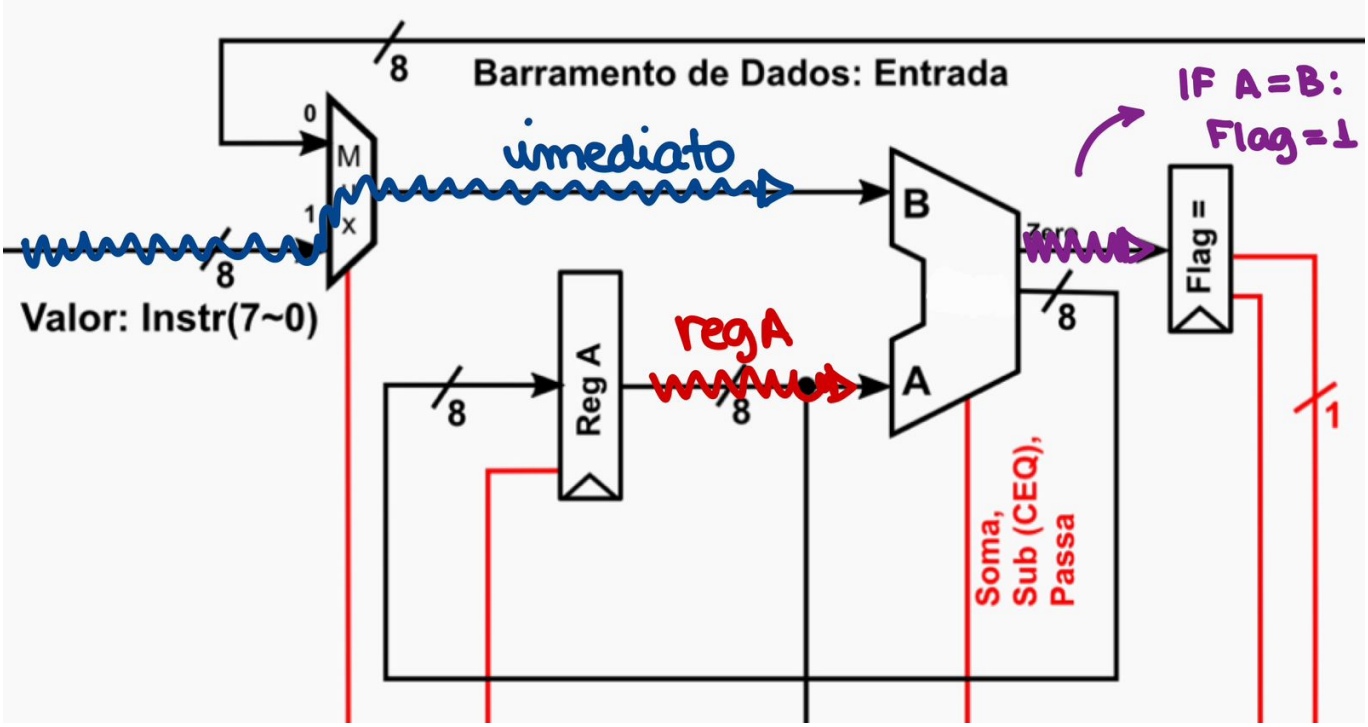

- **ADDi**: Realiza a soma entre o registrador A e um valor imediato.
  - **Sintaxe**: `ADDi $<valor_imediato>`




- **SUBi**: Realiza a subtração entre o registrador A e um valor imediato.
  - **Sintaxe**: `SUBi $<valor_imediato>`




- **CEQi**: Compara o valor do registrador A com um valor imediato, ativando a flag de igualdade se ambos forem iguais.
  - **Sintaxe**: `CEQi $<valor_imediato>`




Foi implementada a **contagem regressiva**:

- Ao apertar o **KEY2** a contagem regride em 1 unidade. Caso o usuário regrida para números negativos, a placa irá reiniciar.

## Novos periféricos

Não foram utilizados periféricos extras neste projeto.

# Código-fonte em Assembly

```
LDI $0      #Carrega o acumulador com o valor 0
STA @288    #Armazena o valor do acumulador em HEX0
STA @289    #Armazena o valor do acumulador em HEX1
STA @290    #Armazena o valor do acumulador em HEX2
STA @291    #Armazena o valor do acumulador em HEX3
STA @292    #Armazena o valor do acumulador em HEX4
STA @293    #Armazena o valor do acumulador em HEX5
NOP
LDI $0      #Carrega o acumulador com o valor 0
STA @256    #Armazena o valor do bit0 do acumulador no LDR0 ~ LEDR7
STA @257    #Armazena o valor do bit0 do acumulador no LDR8
STA @258    #Armazena o valor do bit0 do acumulador no LDR9
NOP
LDI $0      #Carrega o acumulador com o valor 0
STA @0      #Armazena o valor do acumulador em MEM[0] (unidades)
STA @1      #Armazena o valor do acumulador em MEM[1] (dezenas)
STA @2      #Armazena o valor do acumulador em MEM[2] (centenas)
STA @6      #Armazena o valor do acumulador em MEM[6] (milhares)
STA @7      #Armazena o valor do acumulador em MEM[7] (dezenas de milhares)
STA @8      #Armazena o valor do acumulador em MEM[8] (centenas de milhares)
STA @9      #Armazena o valor do acumulador em MEM[9] (flag inibir contagem)
LDI $9      #Carrega o acumulador com o valor 9
STA @10     #Armazena o valor do acumulador em MEM[10] (inibir unidade)
STA @11     #Armazena o valor do acumulador em MEM[11] (inibir dezena)
STA @12     #Armazena o valor do acumulador em MEM[12] (inibir centena)
STA @13     #Armazena o valor do acumulador em MEM[13] (inibir milhar)
STA @14     #Armazena o valor do acumulador em MEM[14] (inibir dezena de milhar)
STA @15     #Armazena o valor do acumulador em MEM[15] (inibir centena de milhar)
NOP
STA @511    #Limpa a leitura do botão zero
STA @510    #Limpa a leitura do botão um
INICIO:
NOP
LDA @352    #Carrega o acumulador com a leitura do botão KEY0
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $0     #Compara com constante 0
JEQ .NAO_CLICOU0    #Desvia se igual a 0 (botão não foi pressionado)
JSR .INCREMENTO #O botão foi pressionado, chama a sub-rotina de incremento
NOP         #Retorno da sub-rotina de incremento
NAO_CLICOU0:
JSR .SALVA_DISP #Escreve o valor das váriaveis de contagem nos displays
NOP     #Retorno da sub-rotina de salvar nos displays
LDA @353    #Carrega o acumulador com a leitura do botão KEY1
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $0     #Compara com constante 0
JEQ .NAO_CLICOU1    #Desvia se igual a 0 (botão não foi pressionado)
JSR .DEFINE_LIM #O botão foi pressionado, chama a sub-rotina de incremento
NOP         #Retorno da sub-rotina de definir limite
NAO_CLICOU1:
```

```
JSR .VERIFICA_LIM
NOP     #Retorno da sub-rotina de verificar limite
LDA @354    #Carrega o acumulador com a leitura do botão KEY2
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $0     #Compara com constante 0
JEQ .NAO_CLICOU2    #Desvia se igual a 0 (botão não foi pressionado)
JSR .DECREMENTO #O botão foi pressionado, chama a sub-rotina de incremento
NOP         #Retorno da sub-rotina de incremento
NAO_CLICOU2:
LDA @356    #Carrega o acumulador com a leitura do botão FPGA_RESET
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $1     #Compara com constante 1
JEQ .REINICIO   #Desvia se igual a 1 (botão não foi pressionado)
JSR .RESET  #O botão foi pressionado, chama a sub-rotina de reset
REINICIO:
NOP     #Retorno da sub-rotina de reset
JMP .INICIO #Fecha o laço principal, faz uma nova leitura de KEY0
NOP
INCREMENTO:
STA @511    #Limpa a leitura do botão
LDA @9      #Carrega o valor de MEM[9] (flag inibir contagem)
CEQi $0     #Compara o valor com constante 0
JEQ .INCREMENTAR
RET
INCREMENTAR:
LDA @0      #Carrega o valor de MEM[0] (contador)
ADDi $1     #ADDi com a constante 1
CEQi $10        #Compara o valor com constante 10
JEQ .VAIUM_D    #Realiza o carry out caso valor igual a 10
STA @0      #Salva o incremento em MEM[0] (contador)
RET     #Retorna da sub-rotina
VAIUM_D:
LDI $0      #Carrega valor 0 no acumulador (constante 0)
STA @0      #Armazena o valor do acumulador em MEM[0] (unidades)
LDA @1      #Carrega valor de MEM[1] no acumulador (dezenas)
ADDi $1     #ADDi com a constante 1
CEQi $10        #Compara o valor com constante 10
JEQ .VAIUM_C    #Realiza o carry out caso valor igual a 10
STA @1      #Salva o incremento em MEM[1] (dezenas)
RET
VAIUM_C:
LDI $0      #Carrega valor 0 no acumulador (constante 0)
STA @1      #Armazena o valor do acumulador em MEM[1] (dezenas)
LDA @2      #Carrega valor de MEM[2] no acumulador (centenas)
ADDi $1     #ADDi com a constante 1
CEQi $10        #Compara o valor com constante 10
JEQ .VAIUM_M    #Realiza o carry out caso valor igual a 10
STA @2      #Salva o incremento em MEM[2] (centenas)
RET
VAIUM_M:
LDI $0      #Carrega valor 0 no acumulador (constante 0)
STA @2      #Armazena o valor do acumulador em MEM[2] (centenas)
LDA @6      #Carrega valor de MEM[6] no acumulador (milhares)
ADDi $1     #ADDi com a constante 1
```

```
CEQi $10        #Compara o valor com constante 10
JEQ .VAIUM_DM   #Realiza o carry out caso valor igual a 10
STA @6      #Salva o incremento em MEM[6] (milhares)
RET
VAIUM_DM:
LDI $0      #Carrega valor 0 no acumulador (constante 0)
STA @6      #Armazena o valor do acumulador em MEM[6] (milhares)
LDA @7      #Carrega valor de MEM[7] no acumulador (dezenas de milhares)
ADDi $1     #ADDi com a constante 1
CEQi $10        #Compara o valor com constante 10
JEQ .VAIUM_CM   #Realiza o carry out caso valor igual a 10
STA @7      #Salva o incremento em MEM[7] (dezenas de milhares)
RET
VAIUM_CM:
LDI $0      #Carrega valor 0 no acumulador (constante 0)
STA @7      #Armazena o valor do acumulador em MEM[6] (milhares)
LDA @8      #Carrega valor de MEM[7] no acumulador (dezenas de milhares)
ADDi $1     #ADDi com a constante 1
CEQi $10        #Compara o valor com constante 10
JEQ .RESET  #Zera se chegar ao final
STA @7      #Salva o incremento em MEM[7] (dezenas de milhares)
RET
SALVA_DISP:
LDA @0      #Carrega o valor de MEM[0] (unidades)
STA @288    #Armazena valor do acumulador de unidades no HEX0
LDA @1      #Carrega o valor de MEM[1] (dezenas)
STA @289    #Armazena valor do acumulador de dezenas no HEX1
LDA @2      #Carrega o valor de MEM[2] (centenas)
STA @290    #Armazena valor do acumulador de centenas no HEX2
LDA @6      #Carrega o valor de MEM[6] (milhares)
STA @291    #Armazena valor do acumulador de unidades no HEX3
LDA @7      #Carrega o valor de MEM[7] (dezenas de milhares)
STA @292    #Armazena valor do acumulador de dezenas no HEX4
LDA @8      #Carrega o valor de MEM[8] (centenas de milhares)
STA @293    #Armazena valor do acumulador de centenas no HEX5
RET
RESET:
LDI $0      #Carrega o acumulador com o valor 0
STA @0      #Armazena o valor do acumulador na MEM[0] (unidades)
STA @1      #Armazena o valor do acumulador na MEM[1] (dezenas)
STA @2      #Armazena o valor do acumulador na MEM[2] (centenas)
STA @6      #Armazena o valor do acumulador na MEM[6] (milhar)
STA @7      #Armazena o valor do acumulador na MEM[7] (dezena de milhar)
STA @8      #Armazena o valor do acumulador na MEM[8] (centena de milhar)
STA @9      #Armazena o valor do acumulador na MEM[9] (flag inibir contagem)
STA @257    #Armazena o valor do bit0 do acumulador no LDR8
LDI $9      #Carrega o acumulador com o valor 9
STA @10     #Armazena o valor do acumulador em MEM[10] (inibir unidade)
STA @11     #Armazena o valor do acumulador em MEM[11] (inibir dezena)
STA @12     #Armazena o valor do acumulador em MEM[12] (inibir centena)
STA @13     #Armazena o valor do acumulador em MEM[13] (inibir milhar)
STA @14     #Armazena o valor do acumulador em MEM[14] (inibir dezena de milhar)
STA @15     #Armazena o valor do acumulador em MEM[15] (inibir centena de milhar)
RET
```

```
VERIFICA_LIM:
LDA @0      #Carrega o valor de MEM[0] (unidades)
CEQ @10     #Compara o valor de MEM[10] (inibir unidade)
JEQ .NEXT_LIM1
RET
NEXT_LIM1:
LDA @1      #Carrega o valor de MEM[1] (dezenas)
CEQ @11     #Compara o valor de MEM[11] (inibir dezenas)
JEQ .NEXT_LIM2
RET
NEXT_LIM2:
LDA @2      #Carrega o valor de MEM[2] (centenas)
CEQ @12     #Compara o valor de MEM[12] (inibir centenas)
JEQ .NEXT_LIM3
RET
NEXT_LIM3:
LDA @6      #Carrega o valor de MEM[6] (milhar)
CEQ @13     #Compara o valor de MEM[13] (inibir milhar)
JEQ .NEXT_LIM4
RET
NEXT_LIM4:
LDA @7      #Carrega o valor de MEM[7] (dezena de milhar)
CEQ @14     #Compara o valor de MEM[10] (inibir dezena de milhar)
JEQ .NEXT_LIM5
RET
NEXT_LIM5:
LDA @8      #Carrega o valor de MEM[8] (centena de milhar)
CEQ @15     #Compara o valor de MEM[10] (inibir centena de milhar)
JEQ .TODOS_IGUAL
RET
TODOS_IGUAL:
LDI $1      #Carrega o acumulador com o valor 1
STA @9      #Armazena o valor do acumulador em MEM[9] (flag inibir contagem)
STA @257    #Armazena o valor do bit0 do acumulador no LDR8
RET
DEFINE_LIM:
STA @510    #Limpa a leitura do botão um
LDA @320    #Carrega o acumulador com a leitura do SW7TO0
STA @10     #Armazena o valor do acumulador em MEM[10] (inibir unidade)
LDI $4      #Carrega o acumulador com o valor 4
STA @256    #Armazena o valor do bit0 do acumulador no LDR0 ~ LEDR7
AGUARDA_D:
LDA @353    #Carrega o acumulador com a leitura do botão KEY1
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $0     #Compara com constante 0
JEQ .AGUARDA_D  #Desvia se igual a 0 (botão não foi pressionado)
STA @510    #Limpa a leitura do botão um
LDA @320    #Carrega o acumulador com a leitura do SW7TO0
STA @11     #Armazena o valor do acumulador em MEM[11] (inibir dezena)
LDI $16     #Carrega o acumulador com o valor 16
STA @256    #Armazena o valor do bit0 do acumulador no LDR0 ~ LEDR7
AGUARDA_C:
LDA @353    #Carrega o acumulador com a leitura do botão KEY1
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
```

```
CEQi $0     #Compara com constante 0
JEQ .AGUARDA_C  #Desvia se igual a 0 (botão não foi pressionado)
STA @510    #Limpa a leitura do botão um
LDA @320    #Carrega o acumulador com a leitura do SW7TO0
STA @12     #Armazena o valor do acumulador em MEM[12] (inibir centena)
LDI $32     #Carrega o acumulador com o valor 32
STA @256    #Armazena o valor do bit0 do acumulador no LDR0 ~ LEDR7
AGUARDA_M:
LDA @353    #Carrega o acumulador com a leitura do botão KEY1
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $0     #Compara com constante 0
JEQ .AGUARDA_M  #Desvia se igual a 0 (botão não foi pressionado)
STA @510    #Limpa a leitura do botão um
LDA @320    #Carrega o acumulador com a leitura do SW7TO0
STA @13     #Armazena o valor do acumulador em MEM[13] (inibir milhar)
LDI $128    #Carrega o acumulador com o valor 128
STA @256    #Armazena o valor do bit0 do acumulador no LDR0 ~ LEDR7
AGUARDA_DM:
LDA @353    #Carrega o acumulador com a leitura do botão KEY1
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $0     #Compara com constante 0
JEQ .AGUARDA_DM #Desvia se igual a 0 (botão não foi pressionado)
STA @510    #Limpa a leitura do botão um
LDA @320    #Carrega o acumulador com a leitura do SW7TO0
STA @14     #Armazena o valor do acumulador em MEM[13] (inibir dezena de milhar)
LDI $0      #Carrega o acumulador com o valor 0
STA @256    #Armazena o valor do bit0 do acumulador no LDR0 ~ LEDR7
LDI $1      #Carrega o acumulador com o valor 1
STA @258    #Armazena o valor do bit0 do acumulador no LDR9
AGUARDA_CM:
LDA @353    #Carrega o acumulador com a leitura do botão KEY1
ANDi $1     #Utiliza a máscara b0000_0001 para limpar todos os bits menos o bit 0
CEQi $0     #Compara com constante 0
JEQ .AGUARDA_CM #Desvia se igual a 0 (botão não foi pressionado)
STA @510    #Limpa a leitura do botão um
LDA @320    #Carrega o acumulador com a leitura do SW7TO0
STA @15     #Armazena o valor do acumulador em MEM[15] (inibir centena de milhar)
LDI $0      #Carrega o acumulador com o valor 0
STA @258    #Armazena o valor do bit0 do acumulador no LDR9
RET
DECREMENTO:
LDI $0      #Carrega 0 para o acumulador
STA @257    #Armazena o valor do bit0 do acumulador no LDR8
STA @9      #Armazena o valor do acumulador na MEM[9] (flag inibir contagem)
STA @509    #Limpa a leitura do botão KEY2
LDA @0          # Carrega MEM[0] (unidades) no acumulador
CEQi $0         # Verifica se MEM[0] == 0
JEQ .VEMUM_D    # Se MEM[0] == 0, realiza o "empréstimo"
SUBi $1         # Subtrai 1 de MEM[0]
STA @0          # Armazena o novo valor de MEM[0]
RET             # Retorna da sub-rotina
VEMUM_D:
LDI $9          # Carrega 9 no acumulador
STA @0          # Define MEM[0] para 9
```

```
LDA @1          # Carrega MEM[1] (dezenas) no acumulador
CEQi $0         # Verifica se MEM[1] == 0
JEQ .VEMUM_C    # Se MEM[1] == 0, realiza o próximo "empréstimo"
SUBi $1         # Subtrai 1 de MEM[1]
STA @1          # Armazena o novo valor de MEM[1]
RET             # Retorna da sub-rotina
VEMUM_C:
LDI $9          # Carrega 9 no acumulador
STA @1          # Define MEM[1] para 9
LDA @2          # Carrega MEM[2] (centenas) no acumulador
CEQi $0         # Verifica se MEM[2] == 0
JEQ .VEMUM_M    # Se MEM[2] == 0, realiza o próximo "empréstimo"
SUBi $1         # Subtrai 1 de MEM[2]
STA @2          # Armazena o novo valor de MEM[2]
RET             # Retorna da sub-rotina
VEMUM_M:
LDI $9          # Carrega 9 no acumulador
STA @2          # Define MEM[2] para 9
LDA @6          # Carrega MEM[3] (milhares) no acumulador
CEQi $0         # Verifica se MEM[3] == 0
JEQ .VEMUM_DM   # Se MEM[3] == 0, realiza o próximo "empréstimo"
SUBi $1         # Subtrai 1 de MEM[3]
STA @6          # Armazena o novo valor de MEM[3]
RET             # Retorna da sub-rotina
VEMUM_DM:
LDI $9          # Carrega 9 no acumulador
STA @6          # Define MEM[3] para 9
LDA @7          # Carrega MEM[4] (dezenas de milhares) no acumulador
CEQi $0         # Verifica se MEM[4] == 0
JEQ .VEMUM_CM   # Se MEM[4] == 0, realiza o próximo "empréstimo"
SUBi $1         # Subtrai 1 de MEM[4]
STA @7          # Armazena o novo valor de MEM[4]
RET             # Retorna da sub-rotina
VEMUM_CM:
LDI $9          # Carrega 9 no acumulador
STA @7          # Define MEM[4] para 9
LDA @8          # Carrega MEM[5] (centenas de milhares) no acumulador
CEQi $0         # Verifica se MEM[5] == 0
JEQ .RESET  # Zera se for menos que 0
SUBi $1         # Subtrai 1 de MEM[5]
STA @8          # Armazena o novo valor de MEM[5]
RET             # Retorna da sub-rotina
```

## Manual de Uso

- aperte KEY0 para incrementar a contagem
- aperte KEY2 para decrementar a contagem
- aperte FPGA_RESET para resetar a contagem e os limites
- caso queira configurar um limite para a contagem:
    - aperte FPGA_RESET para resetar a contagem e os limites
    - configure as chaves de 0 a 3 com um número binario menor ou igual a 9

- aperte KEY1 para configurar o limite das unidades
- configure as chaves de 0 a 3 com um número binario menor ou igual a 9
- aperte KEY1 para configurar o limite das dezenas
- continue utilizando a mesma lógica até as centenas de milhares (led "fugir" da placa)
    - a placa não ira voltar para a contagem até que você termine de determinar o limite, uma vez iniciado

## GitHub do Projeto

Para mais detalhes e acesso ao código completo, visite o repositório no GitHub:

GitHub - Projeto Contador VHDL

---