

**Projeto 1 – Design de Computadores**

**Parte Intermediária**

Por João Zsigmond e Matheus Kwon

**Outubro 2022**

Sumário

[Introdução 3](#_Toc117460926)

[Processador 3](#_Toc117460927)

[Pontos de Controle 4](#_Toc117460928)

[Instruções 5](#_Toc117460929)

[Fluxo de Dados 10](#_Toc117460930)

[Conexão periféricos 12](#_Toc117460931)

[Memória 13](#_Toc117460932)

[Assembly 14](#_Toc117460933)

# Introdução

O projeto documentado é para a matéria de Design de Computadores do Insper do 6º semestre do curso de Engenharia de Computação, realizado no mês de outubro de 2022. O projeto tem como objetivo aplicar conhecimentos de programação de hardware em FPGA, programando um contador. O contador deve incrementar em 1 quando se pressiona um botão, deve reiniciar sua contagem quando se pressiona outro botão e deve ter a funcionalidade de se definir um limite de contagem.

# Instruções de Uso

O Contador funciona a partir de dois botões, o Key 0 e o Key 3. O Key 0 é utilizado para incrementar o contador quando está no modo de contagem, e incrementar o limite quando está no modo de alterar limite. Quando o programa roda na placa, o valor do limite está no valor máximo de 999.999 e o valor do counter começa em 0. Conforme o Key 0 é pressionado, o valor do counter vai aumentando de um em um. Se quiser ajustar o limite, basta apertar o Key 3. Ao apertar o Key 3, você entra no modo de setar o limite. Nesse modo, você consegue setar o valor do limite uma casa decimal por vêz (indicada pelos leds). O Key 0 é usado para incerementar o valor dessa casa decimal, e quando o valor está correto, o Key 3 é usado para passar para próxima casa decimal. Quando você chega na última casa decimal, o key 3 é apertado para salvar o novo limite. O contador para ao atingir o limite e se tentar aumentar mais que o limite, dá um overflow (sinalizado por todos os leds). Se você tenta setar um limite menor do que o valor atual do contador, dá overflow na hora.

# Processador

O processador programado para o projeto possui arquitetura Registrador-Memória, a qual pode ser interessante para se trabalhar com diversos valores ao mesmo tempo, além de ser possível passar argumentos para sub-rotinas (ou “funções”). É utilizado um banco de 8 registradores para usos diversos.

Para a execução de operações nele, deve haver a entrada de instruções binárias de 16 bits distribuídos para os seguintes propósitos:

* OpCode: Código da operação a ser realizada, utiliza 4 bits;
* Registrador: Especifica qual registrador a ser utilizado do banco, utiliza 3 bits;
* Endereço/Imediato: Endereço contendo valor a ser trabalhado ou o próprio valor (imediato), utiliza 9 bits para endereço e 8 bits para imediato.

Todo o endereçamento feito, tanto do contador do programa quanto do endereço a se ler ou escrever, tem uma largura de 9 bits. Assim, o programa pode possuir, no máximo 512 linhas de instrução, e é apenas possível acessar 512 endereços periféricos.

Além disso, são possíveis 16 operações diferentes com os bits de OpCode. Para o projeto foram utilizadas 13 operações. O OpCode é uma forma de facilitar a referência de operações, mas cada uma é traduzida em pontos de controle do processador. Portanto, ao invés de indicar o valor de cada ponto de controle, que neste processador necessitaria de 13 bits, utiliza-se 4 bits. Existe um decodificador de OpCode dentro da CPU para os pontos de controle.

No processador, existe uma Unidade Lógica e Aritmética (ULA) de duas entradas, uma do banco de registradores (entrada A) e outra de um MUX (entrada B) que seleciona ou de um endereço ou um valor imediato. A ULA realiza 4 operações:

1. Subtração: [entrada B] - [entrada A]
2. Adição: [entrada B] + [entrada A]
3. Passa: [entrada B]
4. “E” lógico: [entrada B] e [entrada A] (“e” lógico bit a bit)

A saída da ULA é conectada ao banco de registradores. Em sua saída, também há um flip-flop que armazena a igualdade entre as entradas – flag de igualdade.

Finalmente, há um contador de programa, que indica qual instrução do programa será executado. Naturalmente, deve incrementar em 1 a cada instrução, mas também há casos de desvio do programa. Para isso, há um MUX que seleciona incrementar em 1, saltar para um endereço indicado na própria instrução ou saltar como retorno de uma sub-rotina (endereço armazenado em um registrador de retorno).

# Pontos de Controle

O processador tem os seguintes pontos de controle: WR, RD, HabFlag, OP, HabRegs, SelMUXULA, JEQC, JSRC, RETC, JMPC, HabWrRet e JNEC. Cada um tem a sua respectiva função:

* *WR*: Habilita escrita em dado endereço (0 ou 1);
* *RD*: Habilita leitura em dado endereço (0 ou 1);
* *HabFlag*: Habilita escrita no flip-flop de igualdade na comparação dos valores de entrada da ULA (0 ou 1);
* *OP*: Operação realizada pela ULA (00, 01, 10 ou 11);
* *HabRegs*: Habilita escrita no banco de registradores (0 ou 1);
* *SelMUXULA*: Seleção da entrada B da ULA, dentre um endereço e um valor imediato (0 ou 1);
* *JEQC*: Habilita desvio quando há igualdade entre os valores da ULA (0 ou 1);
* *JSRC*: Habilita desvio para uma sub-rotina (0 ou 1);
* *RETC*: Habilita desvio para o endereço armazenado no registrador de retorno (0 ou 1);
* *JMPC*: Habilita desvio incondicional (0 ou 1);
* *HabWrRet*: Habilita escrita no registrador que indica igualdade na comparação dos valores de entrada da ULA;
* *JNEC*: Habilita desvio quando há desigualdade entre os valores da ULA (0 ou 1);

# Instruções

Operações que podem ser traduzidas em pontos de controle da CPU:

* ***NOP*:** Nenhuma operação, não modifica o estado da CPU.

OpCode

constant NOP : *std\_logic\_vector*(3 downto 0) := "0000";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: XX [Usado 00] [A operação realizada pela ULA não será levada em consideração]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: X [Usado 0] [A saída do MUX da ULA não será levada em consideração]

**JEQC**: 0 [Não deve haver um salto no programa]

**JSRC**: 0 [Não deve haver um salto no programa]

**RETC**: 0 [Não deve haver um salto no programa]

**JMPC**: 0 [Não deve haver um salto no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto no programa]

* ***LDA*:** Lê um valor presente em um endereço e escreve em um registrador.

OpCode

constant LDA : *std\_logic\_vector*(3 downto 0) := "0001";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 1 [Haverá leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: 10 [A operação realizada pela ULA deve ser a operação “PASSA”]

**HabRegs**: 1 [Deve habilitar a escrita nos registradores]

**SelMUXULA**: 0 [A saída do MUX da ULA deve ser o valor que veio do endereço]

**JEQC**: 0 [Não deve haver um salto no programa]

**JSRC**: 0 [Não deve haver um salto no programa]

**RETC**: 0 [Não deve haver um salto no programa]

**JMPC**: 0 [Não deve haver um salto no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto no programa]

* ***SOMA*:** Efetua a operação de soma entre o valor armazenado em um registrador e um valor armazenado em um endereço e armazena o resultado no registrador.

OpCode

constant SOMA : *std\_logic\_vector*(3 downto 0) := "0010";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 1 [Haverá leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: 01 [A operação realizada pela ULA deve ser a operação “SOMA”]

**HabRegs**: 1 [Deve habilitar a escrita nos registradores]

**SelMUXULA**: 0 [A saída do MUX da ULA deve ser o valor que veio do endereço]

**JEQC**: 0 [Não deve haver um salto no programa]

**JSRC**: 0 [Não deve haver um salto no programa]

**RETC**: 0 [Não deve haver um salto no programa]

**JMPC**: 0 [Não deve haver um salto no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto no programa]

* ***SUB*:** Efetua a operação de subtração entre o valor armazenado em um registrador e um valor armazenado em um endereço e armazena o resultado no registrador.

OpCode

constant SUB : *std\_logic\_vector*(3 downto 0) := "0011";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 1 [Haverá leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: 00 [A operação realizada pela ULA deve ser a operação “SUB”]

**HabRegs**: 1 [Deve habilitar a escrita nos registradores]

**SelMUXULA**: 0 [A saída do MUX da ULA deve ser o valor que veio do endereço]

**JEQC**: 0 [Não deve haver um salto no programa]

**JSRC**: 0 [Não deve haver um salto no programa]

**RETC**: 0 [Não deve haver um salto no programa]

**JMPC**: 0 [Não deve haver um salto no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto no programa]

* ***AND*:** Efetua a operação de AND (bit por bit) entre o valor armazenado em um registrador e um valor armazenado em um endereço e armazena o resultado no registrador. A instrução “AND” pode ser usada para aplicar uma máscara sobre um valor.

OpCode

constant ANDD : *std\_logic\_vector*(3 downto 0) := "0100";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 1 [Haverá leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: 11 [A operação realizada pela ULA deve ser a operação “AND”]

**HabRegs**: 1 [Deve habilitar a escrita nos registradores]

**SelMUXULA**: 0 [A saída do MUX da ULA deve ser o valor que veio do endereço]

**JEQC**: 0 [Não deve haver um salto no programa]

**JSRC**: 0 [Não deve haver um salto no programa]

**RETC**: 0 [Não deve haver um salto no programa]

**JMPC**: 0 [Não deve haver um salto no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto no programa]

* ***LDI*:** Carrega em um registrador o valor lido do imediato.

OpCode

constant LDI : *std\_logic\_vector*(3 downto 0) := "0101";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: 10 [A operação realizada pela ULA deve ser a operação “PASSA”]

**HabRegs**: 1 [Deve habilitar a escrita nos registradores]

**SelMUXULA**: 1 [A saída do MUX da ULA deve ser o valor que veio do imediato]

**JEQC**: 0 [Não deve haver um salto no programa]

**JSRC**: 0 [Não deve haver um salto no programa]

**RETC**: 0 [Não deve haver um salto no programa]

**JMPC**: 0 [Não deve haver um salto no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto no programa]

* ***STA*:** Armazena o valor de um registrador em um endereço.

OpCode

constant STA : *std\_logic\_vector*(3 downto 0) := "0110";

Pontos de Controle

**WR**: 1 [Haverá escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: XX [Usado 00] [A operação realizada pela ULA não será levada em consideração]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: X [Usado 0] [A saída do MUX da ULA não será levada em consideração]

**JEQC**: 0 [Não deve haver um salto no programa]

**JSRC**: 0 [Não deve haver um salto no programa]

**RETC**: 0 [Não deve haver um salto no programa]

**JMPC**: 0 [Não deve haver um salto no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto no programa]

* ***JMP*:** Realiza um desvio de execução no programa

OpCode

constant JMP : *std\_logic\_vector*(3 downto 0) := "0111";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: XX [Usado 00] [A operação realizada pela ULA não será levada em consideração]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: X [Usado 0] [A saída do MUX da ULA não será levada em consideração]

**JEQC**: 0 [Não deve haver um salto condicional no programa]

**JSRC**: 0 [Não deve haver um salto para uma sub-rotina no programa]

**RETC**: 0 [Não deve haver um salto de retorno de sub-rotina no programa]

**JMPC**: 1 [Realiza um desvio de execução no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto condicional no programa]

* ***JEQ*:** Realiza um desvio de execução condicional no programa quando a flag de comparação está habilitada.

OpCode

constant JEQ : *std\_logic\_vector*(3 downto 0) := "1000";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: XX [Usado 00] [A operação realizada pela ULA não será levada em consideração]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: X [Usado 0] [A saída do MUX da ULA não será levada em consideração]

**JEQC**: 1 [Realiza um desvio de execução condicional no programa caso o flip flop de comparação armazenar o valor “1”]

**JSRC**: 0 [Não deve haver um salto para uma sub-rotina no programa]

**RETC**: 0 [Não deve haver um salto de retorno de sub-rotina no programa]

**JMPC**: 0 [Não deve haver um salto **incondicional** no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto condicional no programa do tipo JNE]

* ***CEQ*:** Realiza uma comparação entre um valor armazenado em um registrador e um valor armazenado em um endereço. Se os valores forem iguais, seta a flag de comparação para “1”.

OpCode

constant CEQ : *std\_logic\_vector*(3 downto 0) := "1001";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 1 [deve haver leitura do endereço]

**HabFlag**: 1 [queremos habilitar o flag de comparação]

**OP**: 00 [A operação realizada pela ULA deve ser a operação “SUB”]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: 0 [A saída do MUX da ULA deve ser o valor que veio do endereço]

**JEQC**: 0 [Não deve haver um salto condicional no programa]

**JSRC**: 0 [Não deve haver um salto para uma sub-rotina no programa]

**RETC**: 0 [Não deve haver um salto de retorno de sub-rotina no programa]

**JMPC**: 0 [Não deve haver um salto **incondicional** no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto condicional no programa do tipo JNE]

* ***JSR*:** Realiza um desvio de execução no programa para uma sub-rotina.

OpCode

constant JSR : *std\_logic\_vector*(3 downto 0) := "1010";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: XX [Usado 00] [A operação realizada pela ULA não será levada em consideração]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: X [Usado 0] [A saída do MUX da ULA não será levada em consideração]

**JEQC**: 0 [Não deve haver um salto condicional no programa]

**JSRC**: 1 [Haverá um salto para uma sub-rotina no programa]

**RETC**: 0 [Não deve haver um salto de retorno de sub-rotina no programa]

**JMPC**: 0 [Não deve haver um salto **incondicional** no programa]

**HabWrRET**: 1 [Haverá escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto condicional no programa do tipo JNE]

* ***RET*:** Realiza a instrução de retorno. Ocorre um desvio de execução para o endereço armazenado no registrador de retorno.

OpCode

constant RET : *std\_logic\_vector*(3 downto 0) := "1011";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: XX [Usado 00] [A operação realizada pela ULA não será levada em consideração]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: X [Usado 0] [A saída do MUX da ULA não será levada em consideração]

**JEQC**: 0 [Não deve haver um salto condicional no programa]

**JSRC**: 0 [Não deve haver um salto para uma sub-rotina no programa]

**RETC**: 1 [Deve haver um salto de retorno de sub-rotina no programa]

**JMPC**: 0 [Não deve haver um salto **incondicional** no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Não deve haver um salto condicional no programa do tipo JNE]

* ***JNE*:** Realiza um desvio de execução condicional no programa quando a flag de comparação está desabilitada.

OpCode

constant JNE : *std\_logic\_vector*(3 downto 0) := "1100";

Pontos de Controle

**WR**: 0 [Não deve haver escrita ao endereço]

**RD**: 0 [Não deve haver leitura do endereço]

**HabFlag**: 0 [Não queremos habilitar o flag de comparação]

**OP**: XX [Usado 00] [A operação realizada pela ULA não será levada em consideração]

**HabRegs**: 0 [Não deve habilitar a escrita nos registradores]

**SelMUXULA**: X [Usado 0] [A saída do MUX da ULA não será levada em consideração]

**JEQC**: 0 [Não deve haver desvio de execução condicional do tipo JEQ]

**JSRC**: 0 [Não deve haver um salto para uma sub-rotina no programa]

**RETC**: 0 [Não deve haver um salto de retorno de sub-rotina no programa]

**JMPC**: 0 [Não deve haver um salto **incondicional** no programa]

**HabWrRET**: 0 [Não deve haver escrita no registrador de endereço de retorno]

**JNEC**: 0 [Realiza um desvio de execução condicional no programa caso o flip flop de comparação armazenar o valor “0”]

# Fluxo de Dados

O Barramento de Leitura de Dados é um barramento de 8 bits que nos permite acessar os valores armazenados nos botões a nas chaves da nossa placa. Os botões e as chaves estão mapeados em um endereço e ao acessar esse endereço através da instrução LDA, conseguimos acessar o valor armazenado. Também usamos esse barramento para acessar valores armazenados na nossa memória RAM.

Os botões estão conectados a um detector de bordas e a saída desse detector de bordas está conectada a um flip flop. Dessa forma, quando o botão é pressionado, o detector de bordas emite um pulso. Esse pulso é recebido pelo flip flop em sua entrada de clock, o que faz com que ele passe a armazenar o valor “1”. Ele só irá voltar a armazenar o valor “0” quando o reset do flip flop for ativado, e isso acontece quando escrevemos em certo endereço da memória de limpeza de leitura. Por fim, a saída do flip flop está conectada a um tri-state, que só é habilitado se queremos ativar a leitura daquele endereço.

Quando lemos um valor no endereço de algum botão, aplicamos uma máscara através da instrução de AND com “00000001”, dessa forma conseguimos captar apenas o bit que representa o estado do botão, visto que o barramento tem largura de 8 bits e a informação de botão é de 1 bit.

O Barramento de Escrita de Dados, por sua vez, é usado para armazenar valores em memória ou em periféricos. Quando desejamos mostrar um valor no 7-segment display por exemplo, usamos a instrução STA (stash) que vai armazenar o valor no endereço de memória mapeado a um registrador que está conectado a um decodificador que por sua vez mapeia o valor armazenado ao Display de 7 segementos.

Quando lemos um valor no endereço de alguma chave, aplicamos uma máscara através da instrução de AND com “00000001”, dessa forma conseguimos captar apenas o bit que representa o estado do botão, visto que o barramento tem largura de 8 bits e a informação de botão é de 1 bit.

Da mesma forma, se quisermos escrever um valor para nossa memória RAM, esse barramento nos permite escrever o dado que será armazenado.

O Barramento de Endereços tem a função de especificar em qual endereço queremos ler ou escrever os dados dos primeiros dois barramentos especificados. Esse barramento tem largura de 9 bits e nos possibilita a escrita ou leitura em 512 endereços diferentes.

O Barramento de Controle é usado para habilitar ou desabilitar a leitura ou escrita em cada endereço. A CPU está conectada a todos endereços em todos os momentos, então é necessário ter um barramento que delimita quando estamos querendo ler ou escrever informação em cada endereço. Além disso, controle resets e clocks.

# Conexão processador-periféricos

Diagram

Description automatically generated

Diagrama simplificado do diagrama completo do Professor Paulo Carlos Santos.

# Memória

| **Endereço em Decimal** | **Periférico** | **Largura dos Dados** | **Tipo de Acesso** | **Bloco (Página) de Memória** |
| --- | --- | --- | --- | --- |
| 0 ~ 63 | RAM | 8 bits | Leitura/Escrita | 0 |
| 64 ~ 127 | Reservado | – | – | 1 |
| 128 ~ 191 | Reservado | – | – | 2 |
| 192 ~ 255 | Reservado | – | – | 3 |
| 256 | LEDR0 ~ LEDR7 | 8 bits | Escrita | 4 |
| 257 | LEDR8 | 1 bit | Escrita | 4 |
| 258 | LEDR9 | 1 bit | Escrita | 4 |
| 259 ~ 287 | Reservado | – | – | 4 |
| 288 | HEX0 | 4 bits | Escrita | 4 |
| 289 | HEX1 | 4 bits | Escrita | 4 |
| 290 | HEX2 | 4 bits | Escrita | 4 |
| 291 | HEX3 | 4 bits | Escrita | 4 |
| 292 | HEX4 | 4 bits | Escrita | 4 |
| 293 | HEX5 | 4 bits | Escrita | 4 |
| 294 ~ 319 | Reservado | – | – | 4 |
| 320 | SW0 ~ SW7 | 8 bits | Leitura | 5 |
| 321 | SW8 | 1 bit | Leitura | 5 |
| 322 | SW9 | 1 bit | Leitura | 5 |
| 323 ~ 351 | Reservado | – | – | 5 |
| 352 | KEY0 | 1 bit | Leitura | 5 |
| 353 | KEY1 | 1 bit | Leitura | 5 |
| 354 | KEY2 | 1 bit | Leitura | 5 |
| 355 | KEY3 | 1 bit | Leitura | 5 |
| 356 | FPGA\_RESET | 1 bit | Leitura | 5 |
| 357 ~ 383 | Reservado | – | – | 5 |
| 384 ~ 447 | Reservado | – | – | 6 |
| 448 ~ 506 | Reservado | – | – | 7 |
| 507 | Limpa Leitura FPGA\_RESET | – | Escrita | 7 |
| 508 | Limpa Leitura KEY3 | – | Escrita | 7 |
| 509 | Limpa Leitura KEY2 | – | Escrita | 7 |
| 510 | Limpa Leitura KEY1 | – | Escrita | 7 |
| 511 | Limpa Leitura KEY0 | – | Escrita | 7 |

# Assembly

Setup:

LDI R0, $128

STA @63, R0

LDI R0, $0

STA @0, R0

LDI R1, $1

STA @1, R1

LDI R1, $2

STA @2, R1

LDI R1, $3

STA @3, R1

LDI R1, $4

STA @4, R1

LDI R1, $5

STA @5, R1

LDI R1, $10

STA @10, R1

LDI R1, $9

STA @FLAG, R0

STA @LIM0, R1

STA @LIM1, R1

STA @LIM2, R1

STA @LIM3, R1

STA @LIM4, R1

STA @LIM5, R1

STA @MEM0, R0

STA @MEM2, R0

STA @MEM1, R0

STA @MEM3, R0

STA @MEM4, R0

STA @MEM5, R0

STA @HEX0, R0

STA @HEX1, R0

STA @HEX2, R0

STA @HEX3, R0

STA @HEX4, R0

STA @HEX5, R0

LDA R7, @0

LDA R6, @0

LDA R5, @0

LDA R4, @0

LDA R3, @0

LDA R2, @0

LDA R1, @0

LDA R0, @0

Contador:

LDA R3, @MEM0

STA @HEX0, R3

LDA R3, @MEM1

STA @HEX1, R3

LDA R3, @MEM2

STA @HEX2, R3

LDA R3, @MEM3

STA @HEX3, R3

LDA R3, @MEM4

STA @HEX4, R3

LDA R3, @MEM5

STA @HEX5, R3

JMP @IncCont

IncCont:

LDA R7, @KEY0

AND R7, @1

CEQ R7, @1

JNE @Limits

CLR @KEY0

LDA R0, @MEM0

SOMA R0, @1

CEQ R0, @10

JEQ @ResetMem0

STA @MEM0, R0

STA @HEX0, R0

JMP @Limits

ResetMem0:

LDA R0, @0

STA @MEM0, R0

STA @HEX0, R0

LDA R1, @MEM1

SOMA R1, @1

CEQ R1, @10

JEQ @ResetMem1

STA @MEM1, R1

STA @HEX1, R1

JMP @Limits

ResetMem1:

LDA R1, @0

STA @MEM1, R1

STA @HEX1, R1

LDA R2, @MEM2

SOMA R2, @1

CEQ R2, @10

JEQ @ResetMem2

STA @MEM2, R2

STA @HEX2, R2

JMP @Limits

ResetMem2:

LDA R2, @0

STA @MEM2, R2

STA @HEX2, R2

LDA R3, @MEM3

SOMA R3, @1

CEQ R3, @10

JEQ @ResetMem3

STA @MEM3, R3

STA @HEX3, R3

JMP @Limits

ResetMem3:

LDA R3, @0

STA @MEM3, R3

STA @HEX3, R3

LDA R4, @MEM4

SOMA R4, @1

CEQ R4, @10

JEQ @ResetMem4

STA @MEM4, R4

STA @HEX4, R4

JMP @Limits

ResetMem4:

LDA R4, @0

STA @MEM4, R4

STA @HEX4, R4

LDA R5, @MEM5

SOMA R5, @1

STA @MEM5, R5

STA @HEX5, R5

JMP @Limits

WaitLimit:

LDA R7, @KEY3

AND R7, @1

CEQ R7, @1

JEQ @SetLimit

LDA R7, @FPGA\_RESET

AND R7, @1

CEQ R7, @1

JNE @IncCont

ResetCount:

CLR @FPGA\_RESET

LDA R7, @0

STA @MEM0, R7

STA @MEM1, R7

STA @MEM2, R7

STA @MEM3, R7

STA @MEM4, R7

STA @MEM5, R7

STA @FLAG, R7

STA @LED8, R7

STA @LED9, R7

STA @LEDb, R7

JMP @Contador

Limits:

JSR @Limiter

JSR @VerificaFlag

JMP @WaitLimit

Limiter:

LDA R0, @LIM5

CEQ R0, @MEM5

JNE @LimiterRet

LDA R0, @LIM4

CEQ R0, @MEM4

JNE @LimiterRet

LDA R0, @LIM3

CEQ R0, @MEM3

JNE @LimiterRet

LDA R0, @LIM2

CEQ R0, @MEM2

JNE @LimiterRet

LDA R0, @LIM1

CEQ R0, @MEM1

JNE @LimiterRet

LDA R0, @LIM0

CEQ R0, @MEM0

JEQ @AcionaFlag

LimiterRet:

RET

AcionaFlag:

LDA R0, @1

STA @FLAG, R0

STA @LED9, R0

STA @LED8, R0

RET

VerificaFlag:

LDA R0, @1

CEQ R0, @FLAG

JEQ @Parado

RET

Parado:

LDA R0, @KEY3

AND R0, @1

CEQ R0, @1

JEQ @SetLimit

LDA R0, @FPGA\_RESET

AND R0, @1

CEQ R0, @1

JEQ @ResetCount

LDA R0, @KEY0

AND R0, @1

CEQ R0, @1

JEQ @AllLEDs

JMP @Parado

AllLEDs:

CLR @KEY0

LDI R1, $255

STA @LEDb, R1

LDI R1, $1

STA @LED8, R1

STA @LED9, R1

JMP @Parado

SetLimit:

CLR @KEY3

LDA R3, @LIM0

STA @HEX0, R3

LDA R3, @LIM1

STA @HEX1, R3

LDA R3, @LIM2

STA @HEX2, R3

LDA R3, @LIM3

STA @HEX3, R3

LDA R3, @LIM4

STA @HEX4, R3

LDA R3, @LIM5

STA @HEX5, R3

LDA R0, @0

LDA R6, @0

LDA R7, @0

STA @LED8, R0

STA @LED9, R0

LDA R2, @1

STA @LEDb, R2

CLR @KEY0

CLR @KEY3

JMP @VerificaK0

VerificaK0:

LDA R6, @KEY0

AND R6, @1

CEQ R6, @0

JEQ @VerificaK1

JSR @IncLim

VerificaK1:

LDA R7, @KEY3

AND R7, @1

CEQ R7, @0

JNE @Set0

JMP @VerificaK0

IncLim:

CLR @KEY0

CEQ R0, @0

JEQ @IncLim0

CEQ R0, @1

JEQ @IncLim1

CEQ R0, @2

JEQ @IncLim2

CEQ R0, @3

JEQ @IncLim3

CEQ R0, @4

JEQ @IncLim4

CEQ R0, @5

JEQ @IncLim5

IncLim0:

LDA R1, @LIM0

SOMA R1, @1

CEQ R1, @10

JNE @StaLim0

LDA R1, $0

StaLim0:

STA @LIM0, R1

STA @HEX0, R1

RET

IncLim1:

LDA R1, @LIM1

SOMA R1, @1

CEQ R1, @10

JNE @StaLim1

LDA R1, $0

StaLim1:

STA @LIM1, R1

STA @HEX1, R1

RET

IncLim2:

LDA R1, @LIM2

SOMA R1, @1

CEQ R1, @10

JNE @StaLim2

LDA R1, $0

StaLim2:

STA @LIM2, R1

STA @HEX2, R1

RET

IncLim3:

LDA R1, @LIM3

SOMA R1, @1

CEQ R1, @10

JNE @StaLim3

LDA R1, $0

StaLim3:

STA @LIM3, R1

STA @HEX3, R1

RET

IncLim4:

LDA R1, @LIM4

SOMA R1, @1

CEQ R1, @10

JNE @StaLim4

LDA R1, $0

StaLim4:

STA @LIM4, R1

STA @HEX4, R1

RET

IncLim5:

LDA R1, @LIM5

SOMA R1, @1

CEQ R1, @10

JNE @StaLim5

LDA R1, $0

StaLim5:

STA @LIM5, R1

STA @HEX5, R1

RET

Set0:

CLR @KEY3

CEQ R0, @0

JEQ @Set1

CEQ R0, @1

JEQ @Set2

CEQ R0, @2

JEQ @Set3

CEQ R0, @3

JEQ @Set4

CEQ R0, @4

JEQ @Set5

LDI R2, @0

STA @LEDb, R2

LDA R7, @0

LDA R6, @0

LDA R5, @0

LDA R4, @0

LDA R3, @0

LDA R2, @0

LDA R1, @0

LDA R0, @0

CLR @KEY0

CLR @KEY3

CLR @FPGA\_RESET

STA @FLAG, R0

JMP @VerificaOF5

Set1:

LDI R2, $2

STA @LEDb, R2

LDI R0, @1

JMP @VerificaK0

Set2:

LDI R2, $4

STA @LEDb, R2

LDI R0, @2

JMP @VerificaK0

Set3:

LDI R2, $8

STA @LEDb, R2

LDI R0, @3

JMP @VerificaK0

Set4:

LDI R2, $16

STA @LEDb, R2

LDI R0, @4

JMP @VerificaK0

Set5:

LDI R2, $32

STA @LEDb, R2

LDI R0, @5

JMP @VerificaK0

VerificaOF5:

LDA R3, @MEM0

STA @HEX0, R3

LDA R3, @MEM1

STA @HEX1, R3

LDA R3, @MEM2

STA @HEX2, R3

LDA R3, @MEM3

STA @HEX3, R3

LDA R3, @MEM4

STA @HEX4, R3

LDA R3, @MEM5

STA @HEX5, R3

LDA R0, @LIM5

SUB R0, @MEM5

AND R0, @63

CEQ R0, @63

JNE @OFIgual5

JSR @AcionaFlag

JMP @AllLEDs

OFIgual5:

LDA R0, @LIM5

SUB R0, @MEM5

CEQ R0, @0

JNE @Contador

LDA R0, @LIM4

SUB R0, @MEM4

AND R0, @63

CEQ R0, @63

JNE @OFIgual4

JSR @AcionaFlag

JMP @AllLEDs

OFIgual4:

LDA R0, @LIM4

SUB R0, @MEM4

CEQ R0, @0

JNE @Contador

LDA R0, @LIM3

SUB R0, @MEM3

AND R0, @63

CEQ R0, @63

JNE @OFIgual3

JSR @AcionaFlag

JMP @AllLEDs

OFIgual3:

LDA R0, @LIM3

SUB R0, @MEM3

CEQ R0, @0

JNE @Contador

LDA R0, @LIM2

SUB R0, @MEM2

AND R0, @63

CEQ R0, @63

JNE @OFIgual2

JSR @AcionaFlag

JMP @AllLEDs

OFIgual2:

LDA R0, @LIM2

SUB R0, @MEM2

CEQ R0, @0

JNE @Contador

LDA R0, @LIM1

SUB R0, @MEM1

AND R0, @63

CEQ R0, @63

JNE @OFIgual1

JSR @AcionaFlag

JMP @AllLEDs

OFIgual1:

LDA R0, @LIM1

SUB R0, @MEM1

CEQ R0, @0

JNE @Contador

LDA R0, @LIM0

SUB R0, @MEM0

AND R0, @63

CEQ R0, @63

JNE @Contador

JSR @AcionaFlag

JMP @AllLEDs