Desenho com traços pretos em fundo branco e letras pretas em fundo branco

Descrição gerada automaticamente com confiança média

**Design de Computadores**

P1 – Entrega Intermediária - Contador

Henrique Martinelli Frezzatti

Lívia Sayuri Makuta

Nicolas Maciel Queiroga

São Paulo, 19 de outubro de 2022

1. **Introdução**

Esse relatório tem como objetivo explicar alguns detalhes e informações acerca da atividade desenvolvida nas aulas 7 a 10 que consistia em estudos guiados para a implementação de um contador.

O documento será dividido nos seguintes tópicos: arquitetura do processador, total de instruções e sua sintaxe, o formato das instruções, os pontos de controle e sua utilização, o diagrama de conexão do processador com os periféricos, o mapa de memória e a fonte do programa em assembly.

1. **Modo de uso**

O modo de uso do contador implementado pode ser visto no vídeo a seguir: <https://www.youtube.com/watch?v=Xq8ofdE91tE> . Além disso, todos os passos também estão descritos abaixo:

1 – Primeiro é necessário configurar o limite do contador, isto é, o maior número que ele vai poder contar dentro do intervalo de 0 até 999999. Para isso, deve-se apertar o botão 1 (KEY 1) e esperar que um LED seja aceso, e então escrever através das chaves (SW0 até SW3) um valor entre 0 e 9 para cada parte inteira do número desejado. Assim, a sequência seria:

* 1. Apertar o KEY1.
  2. Ver o LEDR0 aceso.
  3. Configurar um valor em binário entre 0000 e 1001 (0 e 9 em decimal) nas chaves SW0 e SW3 para as unidades.
  4. Apertar o KEY1 novamente.
  5. Ver o LEDR1 aceso.
  6. Configurar um valor em binário entre 0000 e 1001 (0 e 9 em decimal) nas chaves SW0 e SW3 para as dezenas.
  7. Repetir esse procedimento até chegar nas centenas de milhar e apertar o KEY1 no final para apagar o LEDR5.

2 – Depois do limite de contagem ser configurado basta apertar o KEY0 para ir somando os valores do contador. Quando o contador atingir o limite configurado, o LEDR8 irá acender indicando que o usuário não pode mais somar. E, se o limite passar dos 999999, o LEDR9 será aceso, mostrando que ocorreu um overflow do valor máximo permitido.

3 – Nesse caso, para voltar a somar o usuário pode: reiniciar a contagem apertando o botão FPGA\_RESET ou decrementar o valor através do botão KEY2. Dessa forma, com a contagem sendo menor que o limite ele poderá somar. Ainda, depois o limite também pode ser reconfigurado para somar mais.

1. **Arquitetura do processador**

Para esta entrega intermediária do contador, fizemos o uso da arquitetura registrador-memória. Foi utilizado essa arquitetura visando preparar o código e o projeto do contador para o projeto seguinte do relógio, uma vez que este utiliza essa mesma arquitetura. Além disso, com a implementação desta, há uma variedade maior de registradores para utilização e, assim, uma maior facilidade para guardar valores específicos durante o programa.

Nessa arquitetura, todas as operações aritméticas realizadas são salvas em um banco de registradores (nessa entrega, foi utilizado um banco com 4 registradores disponíveis). Além disso, foram feitas e utilizadas diversas sub-rotinas para cada uma das funcionalidades do contador, além do uso de variáveis auxiliares para facilitar os processos de soma, subtração e o uso do operador lógico AND para mascarar os bits na leitura dos botões, não exigindo mais linhas de código do que o necessário.

Em suma, o processador feito e descrito nesse documento é composto por algumas unidades funcionais como : multiplexadores, *flip-flops*, uma unidade lógica aritmética (ULA), um *program counter* (PC) e um decodificador de instruções. Ademais, esse processador faz uso de duas memórias: a memória ROM - que é uma memória de leitura que permite o *program counter* determine quais serão as próximas instruções a serem executadas - e uma memória RAM – a qual guarda as variáveis e valores importantes para o funcionamento do programa. Por fim, esse processador recebe 15 bits de instruções (incluindo o endereçamento dos registradores), 9 bits de endereçamento da memória RAM e 2 bits para ativar a leitura e escrita de cada periférico.

1. **Total de instruções e sua sintaxe**

O total de instruções existentes nesse processador descrito são 12, e todas com sua sintaxe e funcionalidade se encontram na tabela a seguir:

Tabela

Descrição gerada automaticamente

Nessa tabela estão as possíveis instruções a serem passadas para o processador com uma breve descrição e o seu mnemônico, que é como será acionada a instrução e a maneira com a qual são implementadas as funções em *assembly* e que também é usado para facilitar a programação em VHDL.

1. **Formato das instruções**

As instruções apresentam o seguinte formato:

tmp(0) := instrução & RX & A8 & x”imediato”

No código acima em VHDL, ‘instrução’ refere-se ao *opcode* da instrução, RX ao endereço do registrador sendo utilizado e A8 é o oitavo bit que é utilizado juntamente com o sétimo e o sexto bit para determinar com qual bloco de memória estamos trabalhando - essa parte ficará mais clara no tópico do diagrama do processador com os seus periféricos. Por fim, há também os 8 bits destinados ao valor imediato - onde o sexto e o sétimo bits citados anteriormente estão contidos, para decodificação do bloco correto.

Assim, por exemplo, na instrução:

tmp(0) := SOMA & R0 & ‘0’ & x"08";

Temos uma soma acontecendo entre o que está na posição de memória 8 com o valor que está no registrador 0.

Outro exemplo pode ser visto a seguir:

tmp(1) := STA & R2 & '0' & x"02";

Nessa instrução o que está carregado no registrador 2 é salvo na memória na posição 2, cuja funcionalidade será descrita no mapa de memória abaixo.

1. **Fluxo de dados para o processador**

O fluxo de dados para processador pode ser representado da seguinte maneira:

Diagrama, Esquemático

Descrição gerada automaticamente

No esquema acima, é possível observar que os endereços saem do processador por meio da variável ***ROM\_Address*** - no caso, saem do *program counter* - e entram na memória ROM. Dessa forma, é o ***ROM\_Address*** que informa à memória ROM qual será a próxima instrução. A partir da ROM, esses valores saem e entram novamente no processador por meio do sinal ***instruction\_IN***, aonde depois vão para o MUX e, caso a memória esteja sendo utilizada, depois vão para a ULA. Da ULA, depois de realizar alguma operação, o resultado vai ser guardado em algum dos quatro registradores: R0, R1, R2 e R3.

A partir do processador saem alguns sinais que se dirigem para memória e para os decodificadores de bloco de memória e de posição. Alguns desses são os sinais ***read*** e ***write*** que indicam se a memória RAM deve ser lida ou escrita - o que inclui botões ou chaves e LEDs ou displays de sete segmentos, respectivamente. Contudo, isso só acontecerá se o endereço dos LEDs ou display de sete segmentos estiverem na saída do ***Data\_Address***, enquanto as chaves e botões apenas serão lidos se seus endereços estiverem no ***DATA\_IN***.

1. **Listagem dos pontos de controle e sua utilização**

Para cada instrução há uma palavra de controle, a qual representa os bits que determinam os valores de cada ponto de controle do circuito para realizar a instrução específica. Abaixo se encontra as palavras de controle de cada instrução:

Calendário

Descrição gerada automaticamente com confiança média

Observação: o habilita bloco vai determinar se o banco de registradores está sendo habilitado ou não, substituindo o que antes habilitava apenas um registrador em específico que era o acumulador.

1. **Diagrama de conexão do processador com os periféricos**

Nesse tópico será explicado como os periféricos foram referenciados na memória. Dessa forma, a explicação primeira será feita sobre as chaves e botões e depois sobre os LEDs e o display de sete segmentos. Abaixo então está o diagrama de como foi feita a separação de endereçamento das chaves e botões.

Diagrama, Esquemático

Descrição gerada automaticamente

A partir do diagrama acima, é possível determinar o endereçamento das chaves e dos botões. O endereçamento das chaves e botões é determinado pela junção de dois decodificadores 3x8 e por meio de portas AND pelo circuito, sendo que o bloco 5 é o responsável por controlar tantos os botões quanto as chaves. Dessa forma, para determinar o endereço utilizado, ou seja, se é uma chave ou um botão, utiliza-se o segundo decodificador para escolher o endereço dentro do bloco 5. Assim, com a porta AND sendo utilizada, é possível escolher todos os endereços disponíveis em cada bloco e, nesse caso, do bloco 5. Além disso, vale salientar que cada botão e chave fez o uso de tristates, que possuem 3 estados: alto, baixo e alta impedância. Sendo assim, quando um botão é apertado ele funcionará como um fio (nível 1), e quanto não for apertado estará em nível 0, e caso não esteja sendo utilizado estará em alta impedância.

Luz do sol

Descrição gerada automaticamente com confiança baixa

Acima, há um diagrama para o endereçamento dos LEDs e dos displays de 7 segmentos. Nele, é possível identificar que, também a partir dos mesmo dois decodificadores 3x8 utilizados para o endereçamento de chaves e botões, é endereçado tanto os LEDs quanto os displays. Contudo, nesse caso, é utilizado o bloco 4 para o endereçamento em questão, utilizando diferentes endereços combinando com portas AND para diferenciar LEDs dos displays. Importante notar que para ativar os LEDs e os displays com os valores provenientes dos registradores, antes esses valores precisam estar guardados em registradores ou flip-flops que apenas se ativados por meio desse endereçamento escreverão nos LEDs e display os valores indicados.

1. **Mapa de Memória**

Após a configuração do processador, foi construído um mapa de memória para que seja possível identificar quais endereços foram utilizados em seu endereçamento, listando abaixo todos os endereços relevantes, suas descrições para melhor compreensão e como são acessados.

**Tabela

Descrição gerada automaticamente**

1. **Fonte do programa em assembly**

O programa em assembly está anexado na pasta de entrega como ASM.txt, e esse assembly foi convertido para o formato da instrução da memória ROM através do assembler (também anexado na paste de entrega) que foi implementado pelo grupo. Esse assembler teve como base o código de Marco Mello retirado do repositório do git a seguir: <https://github.com/Insper/DesignComputadores/tree/master/AssemblerASM_BIN_VHDL> .

No assembler implementado, o formato das instruções era o seguinte:

Instrução @posição da memória ou $imediato, registrador

Esse formato só não se aplica quando o comando é NOP, RET ou algum desvio condicional ou incondicional. No caso dos desvios, há apenas a instrução e a posição da linha para onde o desvio deveria ser feito, o que pode ser um número ou um label.

Alguns exemplos de instruções usadas no código são:

LDA @0, R0

Essa instrução carrega o valor da memória na posição 0 para o registrador R0.

SOMA @8, R0

A instrução acima soma o que está no registrador R0 com o que está na posição 8 da memória e salva no registrador 0.

CEQ @9, R0

Já essa instrução compara o que está no registrador R0 com o que está na posição 9 da memória.

JEQ @INC\_DEZENA

Caso o resultado da instrução anterior seja 1, ou seja, os valores da memória e do registrador são iguais, o programa desviará para a linha onde há o bloco da INC\_DEZENA.

STA @0, R0 # Se não, armazena o valor das unidades

Caso contrário, o que está no registrador R0 será guardado na posição 0 da memória

RET

E o código retornará para o a linha onde a sub-rotina foi chamada mais um.