

ID:

**Desenvolvimento de Sistemas Computacionais:
Comunicação da Unidade Central de
Processamento com FPGA
Laboratório de Arquitetura de Computadores**

São José dos Campos - Brasil

Junho de 2017

ID:

**Desenvolvimento de Sistemas Computacionais:
Comunicação da Unidade Central de Processamento com
FPGA
Laboratório de Arquitetura de Computadores**

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Laboratório de Sistemas Computacionais: Arquitetura e Organização de Computadores.

Docente: Prof. Dr. Tiago de Oliveira
Universidade Federal de São Paulo - UNIFESP
Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil
Junho de 2017

Resumo

Para o desenvolvimento da arquitetura de uma Unidade Central de Processamento foram definidos um conjunto de instruções, os formatos e os seus modos de endereçamento, construindo um caminho de dados para cada tipo de instrução, concluída essa etapa foi possível a implementação da Unidade de Processamento, responsável pela execução das instruções integrada com a Unidade de Controle, que tem por finalidade enviar sinais para os demais componentes. Com o processador já implementado, foi realizada a comunicação com o *Field Programmable Gate Array* (FPGA), para que o usuário possa interagir com o projeto desenvolvido e melhor visualizar o seu funcionamento.

Palavras-chaves: FPGA. Unidade Central de Processamento. Entrada e Saída. BCD.

Lista de ilustrações

| | |
|---|----|
| Figura 1 – Endereçamento imediato | 15 |
| Figura 2 – Endereçamento direto | 16 |
| Figura 3 – Endereçamento por registrador | 16 |
| Figura 4 – Endereçamento por deslocamento | 16 |
| Figura 5 – Arquitetura MIPS, Hennessy Petterson | 18 |
| Figura 6 – Arquitetura MIPS | 19 |
| Figura 7 – FPGA (<i>Field Programmable Gate Array</i>) <i>DE2-115 with Cyclone IV</i> | 22 |
| Figura 8 – Esquemático da arquitetura projetada | 26 |
| Figura 9 – Processamento do algoritmo 3 no FPGA | 41 |
| Figura 10 – <i>waveform</i> da simulação do cálculo de fatorial | 46 |
| Figura 11 – Simulação do algoritmo 1 no FPGA | 47 |
| Figura 12 – <i>Waveform</i> do cálculo de fatorial com geração de <i>overflow</i> | 48 |
| Figura 13 – Sinalização de <i>overflow</i> no FPGA | 49 |
| Figura 14 – <i>waveform</i> da simulação do algoritmo 2 | 50 |
| Figura 15 – Processamento do algoritmo 2 no FPGA | 51 |
| Figura 16 – <i>Waveform</i> da simulação do algortimo 3 | 53 |
| Figura 17 – Processamento do algoritmo 3 no FPGA | 54 |
| Figura 18 – Processamento do algoritmo 3 no FPGA sem a tomada de desvio | 55 |

Lista de tabelas

| | |
|--|----|
| Tabela 1 – Comparação RISC X CISC | 17 |
| Tabela 2 – Conjunto de instruções da arquitetura | 24 |
| Tabela 3 – Formatos de Instruções da Arquitetura | 25 |
| Tabela 4 – Sinais de saída do <i>OPcontrol</i> | 31 |
| Tabela 5 – Sinais da Unidade de Controle | 39 |

Lista de Códigos

| | | |
|-----|-------------------------------------|----|
| 4.1 | Banco de Registradores | 29 |
| 4.2 | Extensor de <i>bits</i> | 30 |
| 4.3 | <i>OPcontrol</i> | 31 |
| 4.4 | <i>Unidade Lógica Aritmética</i> | 32 |
| 4.5 | <i>Program Counter</i> | 34 |
| 4.6 | <i>Skip</i> | 35 |
| 4.7 | Memória de Dados | 36 |
| 4.8 | Controlador de saída | 37 |
| 4.9 | Unidade Central de Processamento | 41 |
| A.1 | Memória de dados para o algoritmo 1 | 63 |
| B.1 | Memória de dados para o algoritmo 2 | 65 |
| C.1 | Memória de dados para o algoritmo 3 | 67 |
| D.1 | <i>muxEndereco</i> | 69 |
| E.1 | Unidade de Controle | 71 |
| F.1 | Conversor BCD | 79 |
| G.1 | <i>Display</i> de 7 segmentos | 81 |
| H.1 | <i>DeBounce</i> | 83 |

Sumário

| | | |
|-------------------------|---|-----------|
| Lista de Códigos | 5 | |
| 1 | INTRODUÇÃO | 9 |
| 2 | OBJETIVOS | 11 |
| 2.1 | Geral | 11 |
| 2.2 | Específico | 11 |
| 3 | FUNDAMENTAÇÃO TEÓRICA | 13 |
| 3.1 | Circuitos Digitais | 13 |
| 3.2 | Computador | 13 |
| 3.3 | CPU (<i>Central Processing Unit</i>) | 13 |
| 3.3.1 | Registradores | 14 |
| 3.3.2 | Banco de Registradores | 14 |
| 3.3.2.1 | ULA (Unidade Lógica Aritmética) | 14 |
| 3.3.3 | UC (Unidade de Controle) | 14 |
| 3.3.4 | Conjunto de Instruções | 15 |
| 3.3.5 | Formato de Instruções | 15 |
| 3.3.6 | Modos de Endereçamento | 15 |
| 3.4 | RISC x CISC | 17 |
| 3.5 | MIPS (<i>Microprocessor without interlocked pipeline stages</i>) | 17 |
| 3.5.1 | Instruções MIPS | 19 |
| 3.5.2 | Formato de Instruções | 19 |
| 3.5.3 | Banco de Registradores | 20 |
| 3.5.4 | Memória | 20 |
| 3.5.5 | Extensor de sinal | 20 |
| 3.5.6 | PC (<i>Program Counter</i>) | 21 |
| 3.6 | Quartus II | 21 |
| 3.7 | Verilog | 21 |
| 3.8 | FPGA (<i>Field Programmable Gate Array</i>) | 21 |
| 4 | DESENVOLVIMENTO | 23 |
| 4.1 | Conjunto de Instruções da Arquitetura | 23 |
| 4.2 | Formatos de Instruções da Arquitetura | 23 |
| 4.3 | Modos de Endereçamento da Arquitetura Projetada | 25 |
| 4.3.1 | Esquematização da Arquitetura | 26 |

| | | |
|---|--|-----------|
| 4.3.2 | <i>Datapath</i> (Caminho de Dados) | 27 |
| 4.4 | Unidade de Processamento | 29 |
| 4.4.1 | Memória de Instruções | 29 |
| 4.4.2 | Banco de Registradores | 29 |
| 4.4.3 | Extensor de <i>bits</i> | 30 |
| 4.4.4 | <i>OPcontrol</i> | 30 |
| 4.4.5 | Unidade Lógica Artimética | 32 |
| 4.4.6 | <i>Program Counter</i> | 34 |
| 4.4.7 | <i>Skip</i> | 35 |
| 4.4.8 | Memória de Dados | 36 |
| 4.4.9 | Controlador de Saída | 37 |
| 4.4.10 | Multiplexadores | 37 |
| 4.4.11 | Unidade de Controle | 38 |
| 4.5 | Comunicação com FPGA | 39 |
| 4.5.1 | Conversor BCD | 40 |
| 4.5.2 | <i>Display</i> de 7 segmentos | 40 |
| 4.5.3 | Demais componentes do FPGA | 40 |
| 4.6 | Comunicação com o FPGA | 41 |
| 5 | RESULTADOS E DISCUSSÕES | 45 |
| 5.1 | Algoritmo 1 | 45 |
| 5.2 | Algoritmo 2 | 48 |
| 5.3 | Algoritmo 3 | 52 |
| 6 | CONCLUSÃO | 57 |
| REFERÊNCIAS | | 59 |
| ANEXOS | | 61 |
| ANEXO A – MEMÓRIA DE INSTRUÇÕES | | 63 |
| ANEXO B – MEMÓRIA DE INSTRUÇÕES | | 65 |
| ANEXO C – MEMÓRIA DE INSTRUÇÕES | | 67 |
| ANEXO D – EXEMPLO DE MULTIPLEXADOR | | 69 |
| ANEXO E – UNIDADE DE CONTROLE | | 71 |
| ANEXO F – CONVERSOR BCD | | 79 |

| | |
|---|-----------|
| ANEXO G – DISPLAY DE 7 SEGMENTOS | 81 |
| ANEXO H – DEBOUNCE | 83 |
| REFERÊNCIAS | 85 |

1 Introdução

Desde a década de 1940, a tecnologia de computadores vem sendo amplamente estudada e inovada. Criada com o propósito de realizar cálculos, atualmente, é a ferramenta mais utilizada em todas as áreas do conhecimento, por esse motivo, há uma busca constante de uma arquitetura eficiente, levando em consideração diversos fatores, por exemplo, a velocidade de processamento. A rapidez que um dado é processado é o resultado de como uma Unidade de Processamento é implementada, notando-se a importância do estudo de cada parte de seu desenvolvimento. Um ponto importante é que essa unidade seja desenvolvida de uma forma que a interação com o usuário seja a melhor possível.

Sendo esse o foco dessa etapa final do projeto, viabilizar a interação da Unidade Central de Processamento (CPU) com o usuário, onde este poderá inserir valores e observar os resultados nos *displays* com o uso do FPGA. Nas próximas seções, serão apresentadas os objetivos, os termos aplicados para o desenvolvimento do trabalho, a especificação e a implementação da arquitetura, os resultados, com a execução de algoritmos, a geração de *waveforms* e a simulação no *kit*, as discussões obtidos a partir da comunicação com o ambiente externo e a conclusão final do projeto.

2 Objetivos

Esta seção apresentará os objetivos gerais do projeto e os específicos, referentes à esta etapa de desenvolvimento.

2.1 Geral

O projeto tem por finalidade o desenvolvimento de uma parte fundamental de um sistema computacional, o **Processador**, em uma linguagem de descrição de *hardware*, o *Verilog*, dispondo de uma lógica programável. Primeiramente, definindo uma nova arquitetura com um conjunto de instruções específico, tais como seus formatos e seus modos de endereçamento, em seguida, implementar cada componente do seu caminho de dados, com total funcionalidade, sendo capaz de processar todas as instruções definidas no projeto, de modo a resolver operações lógicas e aritméticas e executar programas simples, como o cálculo de fatorial, por exemplo. Por último, fazer a comunicação do processador implementado com um dispositivo de entrada e saída de dados, podendo operar com os valores inseridos e visualizar os resultados obtidos na saída.

2.2 Específico

Esta etapa tem por finalidade realizar a integração entre a Unidade de Processamento e a Unidade de Controle, implementar o conversor de binário para o BCD, para que os algarismos dos resultados obtidos sejam alocado nos 8 *displays* disponíveis na placa e o *display* de 7 segmentos, implementados em *Verilog* com o uso do *software Quartus* e por final, realizar a comunicação dos módulos de saída e de entrada de dados da CPU (implementados no ponto de checagem anterior) com o FPGA, realizando a simulação com a execução de algoritmos.

3 Fundamentação Teórica

Esta seção definirá os termos importante o entendimento do desenvolvimento do projeto, como Circuitos Digitais, Computador, CPU e seus principais elementos, CISC X RISC, MIPS e seus componentes, *Quartus*, *Verilog* e *FPGA*.

3.1 Circuitos Digitais

Um circuito digital é um circuito que trabalha apenas com dois níveis de tensão, os valores binários 0 e 1. É projetado de forma que ao ser inserido uma entrada, haverá resposta com uma saída, ambas com o valor 0 ou 1. A maneira em que essa resposta é dada é chamada de lógica do circuito, como cada circuito digital obedece a um conjunto de regras lógicas, também pode ser chamado de circuito lógico. (1)

Os circuitos lógicos podem ser combinacionais ou sequenciais. Os circuitos combinacionais são aqueles que a saída depende unicamente das variáveis de entrada, quando combinados com elementos de memória, o circuito é chamado de sequencial. Nos circuitos sequenciais, existe uma entrada para um circuito combinacional, a saída irá para a memória e será pulsada, quando há, por exemplo, um sinal de *clock*. Esta saída será a entrada para um outro estado. (2)

3.2 Computador

Um dos maiores circuito digital conhecido é o computador. A definição mais utilizada, atualmente, é o modelo de *von Neumann*, que define o computador sendo uma máquina que armazena informações e estas são executadas sequencialmente, exceto quando há um comando que a modifique. Seus principais componentes são memória, dispositivos de entrada e saída e CPU, que será abordada na próxima seção. (3)

3.3 CPU (*Central Processing Unit*)

A CPU, chamada também de *processador*, é considerada o cérebro do computador, tem a função buscar instruções armazenadas na memória, interpretá-las e executá-las. Os elementos fundamentais na composição de um processador é o Banco de Registradores, a ULA (Unidade Lógica Aritmética) e a UC (Unidade de Controle).(4)

3.3.1 Registradores

Registradores são um dos componentes da memória interna de um computador, sendo um elemento do processador, onde os dados são armazenados. Há um uso frequente devido à alta velocidade de acesso em comparação aos outros tipo de memória. Os registradores são agrupados no banco de registradores.(5)

3.3.2 Banco de Registradores

Banco de Registradores é um componente digital formado por um conjunto de registradores, possibilitando a organização em seus acessos. Os dados de um registrador podem ser lidos para uma operação e escritos posteriormente, modificando informações internas.(6)

3.3.2.1 ULA (Unidade Lógica Aritmética)

A ULA é o componente que realiza, sobre os dados, que são apresentados em registradores, operações lógicas e aritméticas, fazendo o uso dispositivos lógicos digitais simples. Outra função é a possibilidade de informar casos de *overflow* e valores negativos, por exemplo .

Esse circuito apresenta linhas de seleção, para determinar a operação a ser realizada. Há a necessitando projetar a sessão aritmética, responsável por realizar microoperações aritméticas como a soma e a subtração e com a manipulação dessas operar instruções mais complexas como de multiplicação e de divisão; e a sessão lógica para instruções como AND, OR e NOT, que manipula *bit* a *bit* os operandos.

A Unidade Lógica Aritmética opera também sobre as instruções de deslocamento, modificando os dados ao realizar deslocamentos à direita ou à esquerda. (3)

3.3.3 UC (Unidade de Controle)

A UC é um componente responsável por enviar sinais de controle dentro do processador para a execução de operações, com base no programa que está sendo executado. Esses sinais de controle fazem a abertura e o fechamento de portas lógicas, sendo possível a comunicação com o o banco de registradores e a ULA, por exemplo.

Outra função é a de temporização, que controla e coordena esses sinais de controle para a execução de cada instrução de um programa.(3)

Este componente pode ser implementado basicamente de 2 formas:

- *Hardwired*: A implementação é realizada por máquina de estados finitos, onde cada transição para um próximo se dará pelo valor de entrada do circuito, ou seja, possui

uma lógica sequencial. A maior vantagem deste tipo de desenvolvimento é a sua simplicidade em relação à microprogramada.(3)

- Microprogramada: Neste tipo, cada instruções do conjunto de instruções (3.3.4) é como se fosse um microprograma, sendo enviado para a Unidade de Controle que será um circuito decodificador, responsável por decidir qual sinal deve ser enviado para os demais módulos. (3)

3.3.4 Conjunto de Instruções

Um conjunto de instruções é agrupamento de todas as operações que um processador é capaz de executar. As instruções são organizadas em formatos para que possam ser interpretadas pela CPU. (5)

3.3.5 Formato de Instruções

Uma instrução é dada por uma sequência de *bits*. Os formatos tem o objetivo de facilitar o processador a realizar sua interpretação, dividindo a sequência em campos, sendo que cada um carrega uma determinada informação, como o *opcode*, que especifica o tipo de operação a ser realizada, a referência aos operandos fonte e de destino, o deslocamento, endereço, entre outros. O campo endereço anteriormente citado é relativamente pequeno, necessitando de técnicas para contornar esse tipo de problema.(3)

As instruções podem ter formatos fixos ou variáveis dependendo da arquitetura implementada.

3.3.6 Modos de Endereçamento

Para solucionar os problema do tamanho reduzido do campo endereço dos formatos de instruções foram desenvolvidos várias técnicas de endereçamento, também chamado de modos de endereçamento.(3)

Endereçamento é uma referência a memória feita a partir de um endereço. Alguns exemplos de seus modos são:

- Imediato: o valor do operando está na própria instrução, é a forma mais simples de endereçamento. (Figura 1)

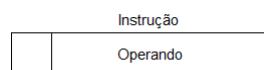


Figura 1 – Endereçamento imediato

- Direto: o campo endereço contém o endereço efetivo do operando. (Figura 2)

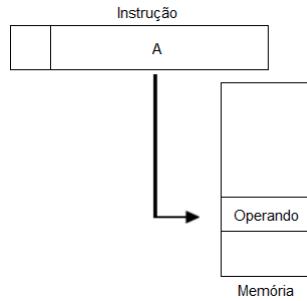


Figura 2 – Endereçamento direto

- Registrador: o endereçamento é semelhante ao direto, mas o campo endereço se refere a um registrador em vez de um endereço na memória. (Figura 3)

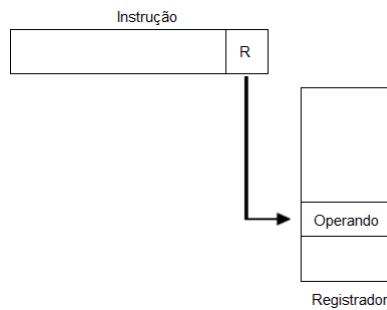


Figura 3 – Endereçamento por registrador

- Deslocamento: necessita de 2 campos endereço na instrução, o valor A de um dos campos da instrução é utilizado diretamente e o outro se refere a um registrador cujo conteúdo ao ser somado com A resulta em um endereço efetivo. (Figura 4)

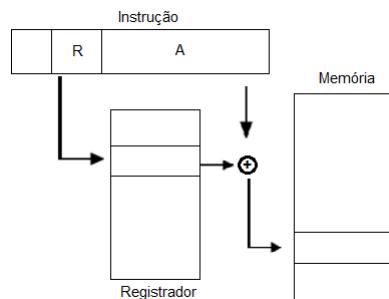


Figura 4 – Endereçamento por deslocamento

Exemplos de dois tipos de endereçamento por deslocamento são o relativo ao PC e por registrador base.

- Relativo ao PC: O registrador que é referenciado implicitamente é o *Program Counter*, assim, o endereço efetivo é o deslocamento relativo ao endereço da instrução.
- Registrador base: o campo endereço contém um deslocamento do endereço da memória principal e o registrador base possui esse endereço, podendo ter referência explícita ou implícita. (3)

O conjunto de instruções tais como seus formatos e modos de endereçamento variam de acordo com cada arquitetura.

3.4 RISC x CISC

A arquitetura de um processador pode ser RISC (*Reduced Instruction Set Computer*) ou CISC (*Complex Instruction Set Computer*), na tabela 1 é realizada uma comparação entre as duas arquiteturas.(3)

Tabela 1 – Comparação RISC X CISC

| RISC | CISC |
|--|---|
| instruções simples | Instruções complexas |
| Vários registradores | Poucos registradores |
| Referência à memória apenas pelas instruções <i>load word</i> ou <i>store word</i> | Qualquer instrução pode acessar a memória |
| Instruções de formato fixo | Instruções de formato variável |
| Poucas instruções | Muitas instruções |
| Poucos modos de endereçamento | Vários modos de endereçamento |
| Complexidade no compilador | Complexidade no microprograma |

Exemplos de arquiteturas CISC são o 386 e o 486 da Intel e de RISC existem os modelos PowerPC e o MIPS.(7)

3.5 MIPS (*Microprocessor without interlocked pipeline stages*)

Criado na década de 1980, o MIPS foi desenvolvido por uma empresa criada por uma estudante de Stanford, tinham como foco os microprocessadores com arquitetura RISC. Essa arquitetura é representada na Figura 5.(7)

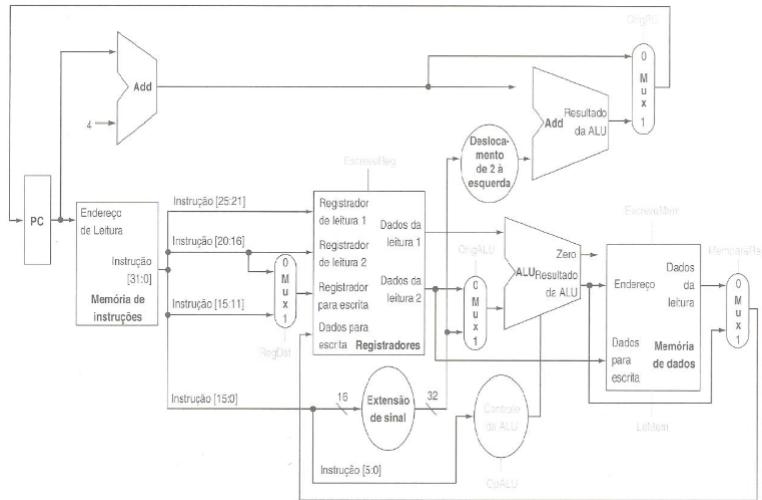


Figura 5 – Arquitetura MIPS, Hennessy Petterson

Um ciclo de execução desta arquitetura consiste, repectivamente, em:

- Busca de instrução
- Decodificação
- Busca de operandos
- Execução
- Armazenamento do resultado

Embora as etapas da execução sejam os mesmos, existem 2 tipos de implementação: a monociclo e a multiciclo.

- **Monociclo:** neste tipo de implementação as instruções de um programa são executadas em 1 ciclo de clock, determinada pela instrução mais lenta, ou seja, cada instrução gastará o tempo de execução da mais tardia. (8)
- **Multiciclo:** diferentemente da monociclo, cada etapa é executada em um ciclo de clock, permitindo que haja redução no tempo de 1 ciclo, assim, diminuindo o tempo total de execução. (8)

Os principais componentes do MIPS é a Unidade de Controle e a Unidade Lógica Aritmética, como já citadas anteriormente. A seguir serão apresentados o seus conjunto e formatos de instruções, o banco de registradores, a memória, o extensor de sinal e o *Program Counter*. (5)

3.5.1 Instruções MIPS

As instruções MIPS possuem todos o tamanho fixo de 32 *bits*, podendo ser dos tipos enumerados a seguir.

1. Lógico/ Aritmético: operações lógicas e aritméticas básicas realizadas por um computador, porém, limitada pelo número de *bits* dos registradores utilizados
2. Desvio condicional/ incondicional: são instruções que fazem o uso do contador de programa, para obter o endereço da próxima instrução em casos de desvios.
3. Controle: tem interferência sob o processamento das instruções.
4. Deslocamento: permite que haja o deslocamento de *bits* do operando para a esquerda ou para a direita.
5. Acesso à memória: são instruções que fazem acesso à memória, para obtenção de dados.
6. Entrada e Saída: são instruções de transferência de dados a partir ou para um dispositivo externo.

Para o processamento dessas instruções, elas são organizadas em diferentes formatos. (5)

3.5.2 Formato de Instruções

O MIPS possui 3 formatos de instruções, as instruções do tipo R, que são as instruções lógicas/aritméticas, por exemplo, do tipo I, que são as instruções de acesso à memória e de desvio condicional e o tipo J, instruções de desvios incondicionais. Os formatos são representados pela Figura 4.(3)

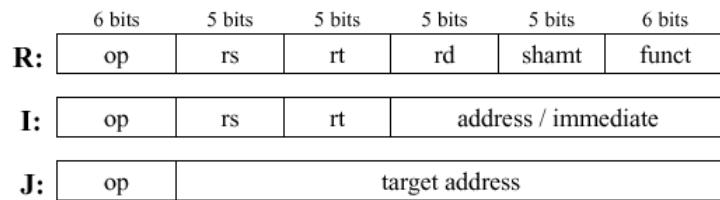


Figura 6 – Arquitetura MIPS

- op: o *opcode*, código da instrução, com tamanho de 5 *bits* independente do formato. A partir da interpretação desta sequência de *bits*, a Unidade de Controle envia os sinais para a realização das operações.

- rs: primeiro registrador fonte.
- rt: segundo registrador fonte.
- rd: registrador de destino.
- shamt: *Shift amount*, utilizada em instruções de deslocamento.
- funct: Seleciona as variações das operação através do opcode.
- *adress*: é o endereço de memória.
- *immediate*: a constante de instruções com imediatos.

Os elementos rs, rt e rd estão localizados no Banco de Registradores e o campo *adress*, em instruções como o de acesso à memória é necessita de um extensor de sinal, abordado na subseção 3.5.5

3.5.3 Banco de Registradores

O MIPS possui um banco de 32 registradores, com 32 *bits* cada. O banco é acessado após a leitura de uma instrução da memória ou a escrita de um dado da memória para um registrador.

3.5.4 Memória

O MIPS possui a Memória de instruções e a Memória de dados.

- Memória de instruções: armazena as instruções que serão executadas. e possui também endereços que armazenam a próxima instrução.
- Memória de dados: armazena as informações e o resultado obtido na ULA, podendo ser gravados, posteriormente, no registrador de destino (rd). (5)

3.5.5 Extensor de sinal

Extensor de sinal é um elemento presente no processador que transforma um dado vetor de *bits* em outro maior, no caso do MIPS, é necessário extender o sinal de 16 *bits* para 32 *bits* para que possa ser utilizado na arquitetura projetada, que possui uma memória de instrução de 32 *bits*. (7)

3.5.6 PC (*Program Counter*)

O *Program Counter* é um registrador que contém o endereço da posição de memória onde está a próxima instrução a ser executada. Sendo assim, seu conteúdo é utilizado para dar sequência a execução das instruções de um programa.(3)

Esses módulos da Unidade de Processamento apresentado até agora podem ser implementados e testados com o auxílio das ferramentas definidas nas próximas seções.

3.6 *Quartus II*

O *Quartus II* é um *software* com dispositivo lógico programável criado pela empresa *Altera*, sendo possível a criação de circuitos lógicos de forma esquemática ou através da implementação por uma linguagem de descrição de *hardware*, como o *VHDL* e o *Verilog*. Ao desenvolver os circuitos é possível testar a sua funcionalidade através da geração de **waveforms** (formas de onda), que é uma das ferramentas disponíveis no *software*. Com a atribuição dos valores de entrada do circuito é possível visualizar a forma de onda, que representa os valores de saída do circuito. (9)

3.7 *Verilog*

O *Verilog* é uma linguagem de descrição de *hardware* criado na década de 1980, é utilizado para diversos projeto que envolvem implementação de circuitos lógicos, permitindo o desenvolvimento em vários níveis de abstração. (10)

3.8 *FPGA (Field Programmable Gate Array)*

Criado em 1983 pela Intel, o *FPGA* é um circuito integrado que pode ser ter sua lógica configurada pelo usuário após a sua fabricação. O *FPGA* é um chip que suporta a implementação de circuitos lógicos relativamente grandes. Consiste de um grande arranjo de blocos lógicos configuráveis situados em um único circuito integrado. Cada bloco contém capacidade computacional para implementar funções lógicas e permite a comunicação entre elas. A Figura 7 ilustra esse dispositivo.(11)

Pode se observar que há 18 chaves, denominadas *switches*, onde o valor binário é inserido como entrada, 8 *displays*, 4 *KEYs* (botões) e os *LEDs*.

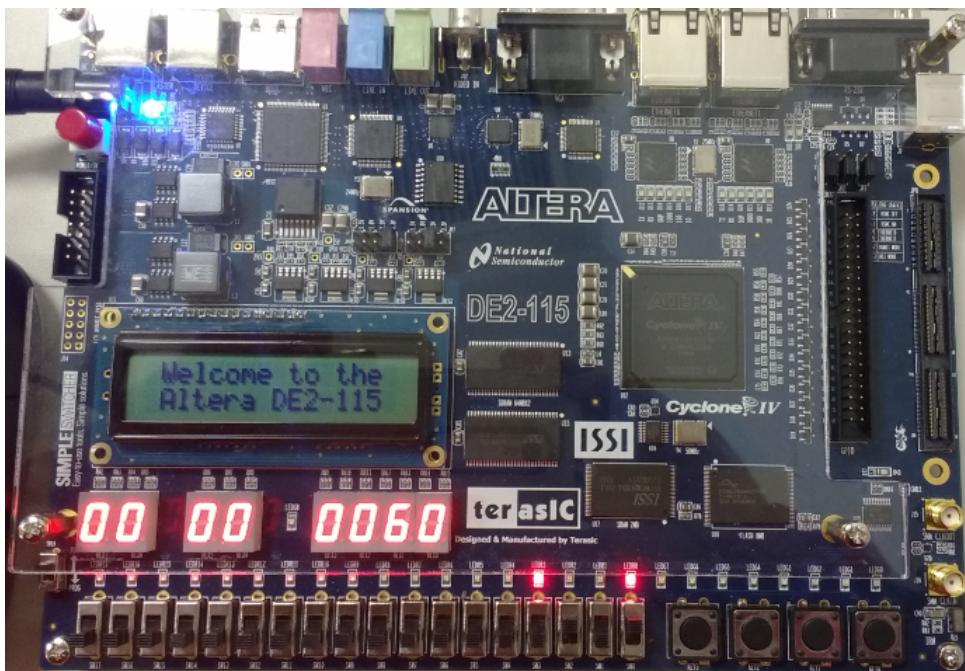


Figura 7 – FPGA (*Field Programmable Gate Array*) *DE2-115 with Cyclone IV*

4 Desenvolvimento

A arquitetura criada neste projeto é RISC, utilizando como base o modelo do MIPS monociclo, que foi modificado para processar um conjunto de instruções específicos, com novos formatos e a inserção de novos elementos de circuito, que serão abordados a seguir.

4.1 Conjunto de Instruções da Arquitetura

O primeiro passo foi projetar um conjunto de instruções para o processador. Sendo definido um conjunto de 27 instruções, com os tipos e as operações representadas na Tabela 2. Para a representação das operações são utilizados alguns elementos, definidos a seguir.

- RA e RB: endereços dos registradores fonte, cada um representado em uma sequência de 5 *bits*.
- RD: endereço do registrador de destino, representado também em 5 *bits*.
- imm: imediato especificado por uma determinada quantidade de *bits* da instrução (a quantidade depende do formato da instrução que segue na seção 4.2).
- desloc: quantidade de *bits* que deve ser deslocada à direita ou à esquerda nas instruções de deslocamento.
- PC: *Program counter*, o contador de programa, que contém o endereço da próxima instrução.
- memDados: memória de dados do processador.
- *Switches* e *Displays* são os elementos do FPGA, explicados na seção 3.8.

O *Opcode* nesta arquitetura possui 5 *bits*, que contém a informação da operação que a instrução realiza para ser enviada à Unidade de Controle.

4.2 Formatos de Instruções da Arquitetura

Com a seleção do conjunto de instruções, foram definidos seus formatos para o processamento ao longo do caminho de dados, a arquitetura projetada possui 4 formatos de instruções, que são classificados de acordo com a quantidade de registradores especificados na instrução apresentados a seguir.

Tabela 2 – Conjunto de instruções da arquitetura

| Opcode | Instrução | Símbolo | Tipo | Sintaxe | Operação |
|--------|--------------------------|---------|---------------------|----------------------|---|
| 00000 | Adição | adc | Aritmético | adc RA, RB, RD | $RD \leftarrow RA + RB$ |
| 00001 | Subtração | sub | Aritmético | sub RA, RB, RD | $RD \leftarrow RA - RB$ |
| 00010 | Adição imediata | adci | Aritmético | adci RA, imm, RD | $RD \leftarrow RA + imm$ |
| 00011 | subtração imediata | subi | Aritmético | subi RA, imm, RD | $RD \leftarrow RA - imm$ |
| 00100 | AND | and | Lógico | and RA, RB, RD | $RD \leftarrow RA \wedge RB$ |
| 00101 | OR | or | Lógico | or RA, RB, RD | $RD \leftarrow RA \vee RB$ |
| 00110 | Negação | not | Lógico | not RA, RD | $RD \leftarrow \sim RA$ |
| 00111 | Load word | lowo | Acesso à memória | lowo R1[imm], RD | $RD \leftarrow memDados[R1 + imm]$ |
| 01000 | Store word | stwo | Acesso à memória | stwo R1[imm], RD | $memDados[R1 + imm] \leftarrow RD$ |
| 01001 | load imediato | loi | Acesso à memória | loi imm, RD | $RD \leftarrow imm$ |
| 01010 | Movimentação | mov | Transferência | mov RA, RD | $RD \leftarrow RA$ |
| 01011 | Deslocamento esquerdo | slel | Deslocamento | slel RA (desloc), RD | $RD \leftarrow RA \ll desloc$ |
| 01100 | Deslocamento direito | sril | Deslocamento | sril RA(desloc), RD | $RD \leftarrow RA \gg desloc$ |
| 01101 | Jump | jmp | Salto incondicional | jmp imm | $PC \leftarrow imm$ |
| 01110 | Jump para registrador | jmpr | Salto incondicional | jmpr RD | $PC \leftarrow RD$ |
| 01111 | Branch equal | beq | Desvio condicional | beq RA, RB, L | $if(R1 == R2) go toL; else doPC + 1$ |
| 10000 | Branch not equal | bneq | Desvio condicional | bneq RA, RB, L | $if(RA \neq RB) go toL; else doPC + 1;$ |
| 10001 | Branch on less than zero | blz | Desvio condicional | blz RA, L | $if(RA < 0) go toL; else doPC + 1;$ |
| 10010 | Set less than | slet | Comparação | slet RA, RB, RD | $if(RA < RB) RD = 1;$ |
| 10011 | Set greater than | sgrt | Comparação | sgrt RA, RB, RD | $if(RA > RB) RD = 1$ |
| 10100 | Entrada | in | Entrada/Saída | in RD | $RD \leftarrow Switches$ |
| 10101 | Saída | out | Entrada/Saída | out RA | $Displays \leftarrow RA$ |
| 10110 | NOP | nop | Controle | nop | Sem operação por 1 ciclo de clock |
| 10111 | HALT | htl | Controle | htl | Para o processamento |
| 11000 | Multiplicação | mult | Aritmética | mult RA, RB, RD | $RD \leftarrow RA * RB$ |
| 11001 | Multiplicação imediata | multi | Aritmética | multi RA, RB, RD | $RD \leftarrow RA * imm$ |
| 11010 | Jump and Link | jal | Salto Incondicional | jal imm | goto imm and \$R0 = endereço |

- OneReg *type*: Instruções que envolvem apenas o uso de 1 registrador e um imediato ou deslocamento de 22 bits. São as instruções:

 - jmp
 - loi
 - in
 - out
 - blz

- Doublereg *type*: Instruções que necessitam do uso de 2 registradores, e que contém um campo com imediato ou uma quantidade de deslocamento de 17 bits, caso não haja, a sequência é preenchida com zeros. As instruções com este formato são:

 - slel
 - sril
 - mov
 - not
 - lowo
 - stwo
 - adci
 - subi
 - multi

- beq
- bneq
- *Tireg type*: Instruções que fazem o uso de 3 registradores. Representada pelas seguintes instruções:
 - adc
 - sub
 - and
 - or
 - slet
 - sgrt
 - mult
- *JHN type*: Instruções que não fazem o uso de nenhum registrador. São as instruções de salto incondicional e de controle.
 - jmp
 - nop
 - hlt
 - jal (esta instrução utiliza apenas o registrador de uso específico, \$R0, portanto não é especificado na instrução, logo ele possui o mesmo formato da instrução *jump*).

A Tabela 3 mostra como foi feita a organização dos 32 *bits* das instruções com a divisão entre os campos do OPCODE, dos registradores e do deslocamento/imediato, de acordo com cada formato.

Tabela 3 – Formatos de Instruções da Arquitetura

| | [31:27] | [26:22] | [21:17] | [16:12] | [11:0] |
|-----------------------|---------|---------|---------|------------------------|--------|
| OneReg <i>type</i> | OPDODE | RA | | deslocamento/ imediato | |
| DoubleReg <i>type</i> | OPCODE | RA | RD | deslocamento/ imediato | |
| Tireg <i>type</i> | OPCODE | RA | RB | RD | – |
| JHN <i>type</i> | OPCODE | | | deslocamento/ imediato | |

4.3 Modos de Endereçamento da Arquitetura Projetada

Os modos de endereçamento da arquitetura projetada podem ser de 4 formas:

- Registrador: as instruções que fazem o uso de registradores são as instruções lógicas e aritméticas (com excessão do addi e subi).
 - Direto: as instruções que contém o formato de instrução JHN, que possuem o endereço na própria instrução.
 - Relativo ao PC: são instruções que fazem o uso do *Program Counter* para calcular o endereço da próxima instrução, como os desvios condicionais.
 - Registrador base: representado pelas instruções de acesso à memória.

4.3.1 Esquematização da Arquitetura

A Figura 8 é a representação do esquemático da Unidade de Processamento da arquitetura projetada, as *flags*, em azul, representam os sinais de controle.

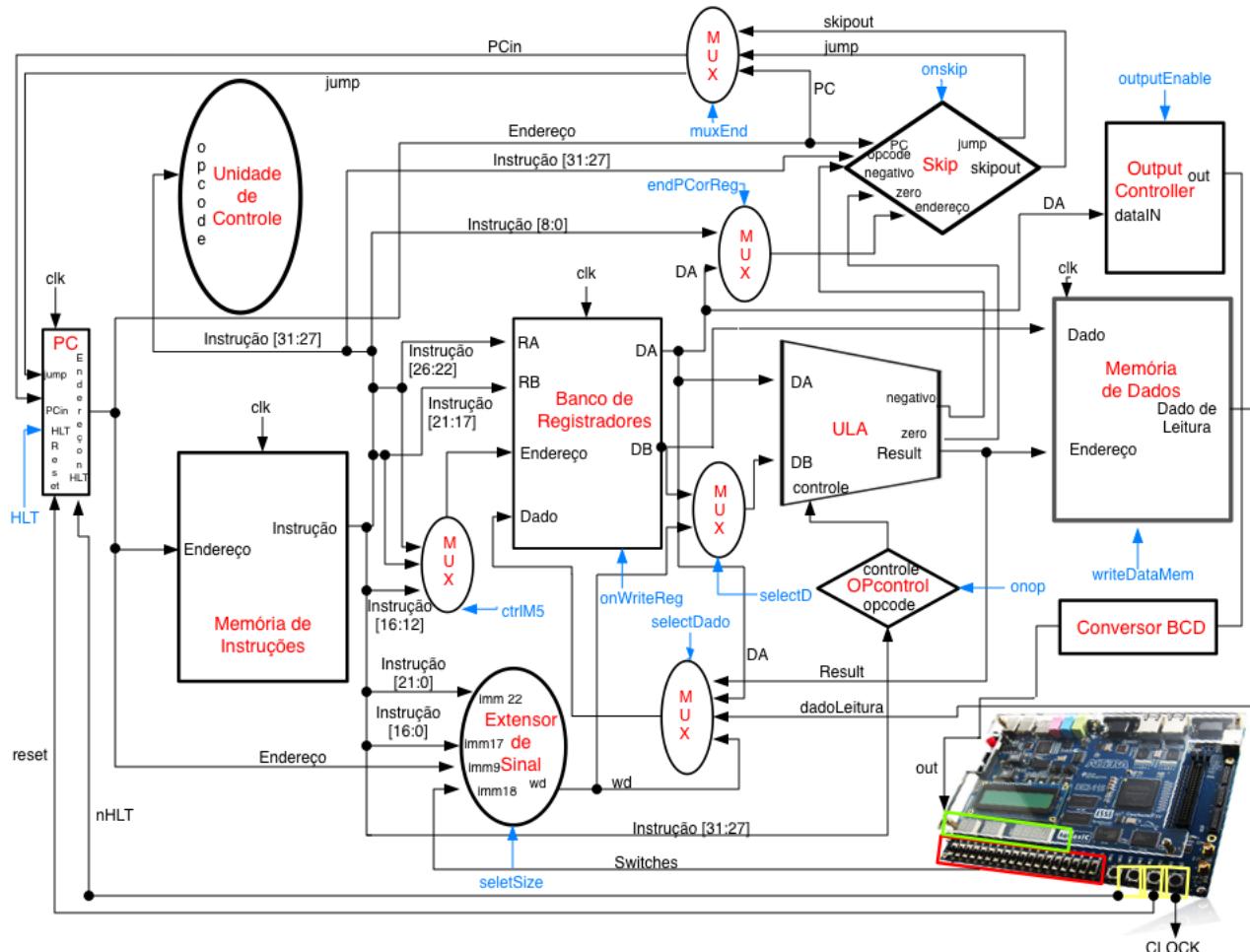


Figura 8 – Esquemático da arquitetura projetada

Na Figura 8, é possível observar a criação de 2 blocos, o *skip* e o *OPcontrol*, que são unidades que fazem o controle de saltos e definem operações lógicas e aritméticas a serem realizadas pela ULA, respectivamente. Os funcionamentos serão explicados mais adiante.

4.3.2 Datapath (Caminho de Dados)

Nesta subseção será explicado como se dará a execução das instruções no caminho de dados na arquitetura mostrada na figura 8. Todas as instruções são buscadas na memória de instruções e o *opcode*, enviado para a Unidade de Controle (UC) para decodificar qual operação deverá ser realizada, e na maioria das instruções, são enviados sinais para o *skip* e/ou *OPcontrol*, que são controles auxiliares à UC.

- **Instruções lógicas e aritméticas:** os registradores fontes são acessados no banco de registradores e os dados contidos neles são enviados para a Unidade Lógica e Aritmética (ULA). A Unidade de controle, enviará um sinal para o *OPcontrol*, que irá definir qual operação deverá ser realizado pela ULA, com o tipo de operação selecionado e os dados, a operação será realizada e o resultado será escrito no banco de registradores.
- **Adição, subtração e multiplicação com imediatos:** O registrador fonte especificado será buscado no banco de registradores e o seu valor é acessado e enviado para a ULA. Como arquitetura faz operações apenas com operandos de 32 bits, o imediato de 17 bits é estendido para 32 e esse valor é enviado para a ULA, assim como no item anterior, o *OPcontrol* enviará um sinal de qual tipo de operação deverá ser realizada, sendo executada e o resultado é gravado no registrador de destino no banco de registradores.
- **Instruções de acesso à memória:** no caso de um *load word*, o valor do registrador fonte é enviado para a ULA, assim como o imediato da instrução de 17 bits estendido para 32 bits. A UC envia um sinal para o *OPcontrol*, este seleciona a operação de adição e envia também para a ULA, então o valor armazenado no registrador fonte somado com o imediato resulta no endereço da memória de dados que contém o valor a ser carregado, este valor é enviado da memória de dados para o banco de registradores no registrador de destino. No *store word* ocorre o contrário, o valor contido em um registrador será escrito na memória de dados.
- **Desvios condicionais:** Os valores dos registradores fontes são enviados para a ULA. A UC envia 2 sinais, um para o *OPcontrol*, que determina a operação e um para o *skip*, especificando que é uma instrução de desvio. Na ULA, são feitas as comparações entre os valores e o resultado é enviado para o *skip*, que verifica se o desvio foi tomado ou não, caso não seja tomado, o PC (*Program Counter*) incrementado com o valor 1

será somado com zero no *skip*, e a próxima instrução é buscada, caso contrário, o $PC+1$ é somado com o endereço contido na instrução que foi estendido de 17 *bits* para 32 *bits*.

- **Desvios incondicionais:** no *jump* e no *jump and link*, a UC envia um sinal para o *skip*, o endereço contido na instrução também será destinado à este, a diferença é que nas instruções *jal*, o endereço atual do PC é estendido e escrito no banco de registradores (no registrador de uso específico \$R0), para que se possa retornar à posição inicial onde foi dado o desvio. Nas instruções *jump register*, o endereço da próxima instrução é lido do registrador especificado na instrução, após a leitura, é enviado ao *skip*, ocorrendo o salto.
- **Comparações:** As instruções *set less than* e *set greater than*, são lidos 2 registradores do banco, o *OPcontrol*, envia um sinal para que realize uma comparação na ULA. Os valores lidos nos 2 registradores são comparados, no primeiro caso, se o valor do primeiro registrador for menor que o segundo, o valor do terceiro registrador especificado na instrução é setado para 1, no segundo caso, ocorre o contrário, se o valor do primeiro for maior que o segundo, o terceiro recebe o valor 1.
- **load immediate:** O imediato contido na instrução é estendido de 22 *bits* para 32 *bits*, e carregado no registrador de destino no banco de registradores.
- **Movimentação:** o registrador fonte é acessado, e o seu valor é escrito no registrador de destino.
- **Instruções de entrada:** Para a entrada, um valor é inserido nos *switches* sendo enviado ao extensor de sinal, estendendo esse valor de 18 *bits* para 32 *bits*, posteriormente, será gravado em um registrador do banco no endereço especificado pela instrução.
- **Instruções de Saída:** Para a saída há um módulo denominado *Output Controller*, que possui uma *flag*, que permite a saída de dados dessa unidade para que seja enviada ao conversor BCD, e posteriormente, enviada ao *display* de 7 segmentos. Sempre que uma instrução *OUT* for executada, permitirá uma leitura de um registrador do endereço especificado na própria instrução, esse valor lido será enviado ao conversor *BCD*.
- **Instruções de controle:** A instrução *halt* paraliza a execução do programa, não permitindo mais o processamento de nenhuma outra instrução após essa até que seja enviado um sinal e possa dar continuidade, e o *nop*, não realiza operação alguma, deixando a Unidade de Processamento ociosa durante 1 ciclo de *clock*.

4.4 Unidade de Processamento

Para o desenvolvimento da unidade de processamento, foi implementado cada componente separadamente, a Memória de Instruções, o Banco de Registradores, a ULA, o *skip*, o *OPcontrol*, a Memória de Dados, o PC, o extensor de *bits* e os multiplexadores, a integração dessas partes, resultará em um circuito lógico sequencial. A Unidade de Controle não foi implementada nesta parte do projeto, o controle da Unidade de Processamento se dará através da entrada de variáveis no circuito.

4.4.1 Memória de Instruções

A memória de Instruções é onde ocorre o armazenamento das instruções a serem executadas na Unidade de Processamento. Para facilitar os testes para a Unidade de Processamento e posteriormente, para a comunicação com o FPGA, cada algoritmo apresentado na seção 5, está contido em uma memória de instruções, cujo Código consta em anexo, a memória para os algoritmos 1, 2 e 3 estão em, A.1, B.1, C.1, repectivamente.

4.4.2 Banco de Registradores

O Banco de Registradores foi definido com 31 registradores de propósito geral e 1 de uso de específico, correspondendo ao primeiro registrador do banco (\$R0), que tem por finalidade apenas o armazenamento de endereço. É munido de 3 entradas para 3 endereços, sendo 2 de leitura (*readReg1* e *readReg2*) e 1 de escrita(*writeAddress*), que são fornecidos pela sequência de *bits* da instrução. Os dados lidos dos registradores com os endereços fornecidos são as saídas do circuito, que são os *readData1* e o *readReg2*.

Há apenas um sinal de controle, o *writeReg*, que quando o valor lógico é alto, possibilita a escrita em um registrador, do dado *writeData* no endereço (*writeAddress*). A implementação do Banco de Registradores é apresentado no Código 4.1.

```

1  module regFile(readReg1, readReg2, writeReg, writeData, writeAddress, readData1,
2                 readData2, ck);
3     input [4:0] readReg1, readReg2;
4     input [31:0] writeData;
5     input [4:0] writeAddress;
6     input ck, writeReg;
7     output [31:0] readData1, readData2;
8     reg [31:0] registers[31:0];
9
10    always @ (posedge ck) begin
11      if (writeReg)
12        registers[writeAddress] = writeData;
13    end
14
15    assign readData1 = registers[readReg1];
16    assign readData2 = registers[readReg2];
17
```

```
18
19  endmodule
```

Código 4.1 – Banco de Registradores

Se pode observar também que quando o *reset* é acionado, todos os registradores são zerados.

4.4.3 Extensor de *bits*

Na arquitetura projetada, as operações deverão ser realizadas apenas com operandos de 32 *bits*, portanto, em algumas instruções é necessário estender os bits, para que viabilize a operação. A forma que extensão se dará depende do sinal enviado pela Unidade de Controle, nos formatos existentes do conjunto de instruções criado, existem imediatos de 17 *bits*, 22 *bits*, 9 *bits* (em casos onde se necessita o armazenamento de endereços) e de 18 *bits* (os *switches*, que precisam ser estendidos antes de serem gravados no Banco de Registradores), que serão as entradas do circuito, e as saídas serão os valores de entrada com zeros à esquerda, completando 32 *bits*. A implementação é apresentada pelo Código 4.2.

```
1 module extensorSinal (selectSize, imm17, imm22, imm9, imm18, out32);
2     input [1:0] selectSize;
3     input [16:0] imm17;
4     input [21:0] imm22;
5     input [8:0] imm9;
6     input [17:0] imm18;
7     output reg [31:0] out32;
8
9     parameter i17 = 2'b0, i22 = 2'b01, i9 = 2'b10, i18 = 2'b11;
10
11    always @ (*) begin
12        case (selectSize)
13            i17: out32 = {15'b0, imm17};
14            i22: out32 = {10'b0, imm22};
15            i9: out32 = {23'b0, imm9};
16            i18: out32 = {14'b0, imm18};
17        endcase
18    end
19 endmodule
```

Código 4.2 – Extensor de *bits*

4.4.4 *OPcontrol*

O *OPcontrol* é um controle auxiliar à Unidade de Controle, todas as operações lógicas e aritméticas são definidas neste módulo, possui semelhanças com o "controle da ULA" existente na arquitetura MIPS. A diferença se dá no modo que ela é controlada, no MIPS, existe um campo na instrução denominado *funct*, que possui o tipo de operação que a ULA deve executar(5), já na arquitetura que está sendo desenvolvida, a Unidade de

Controle enviará um sinal, criado de acordo com o *opcode*, para o *OPcontrol*, que direcionará um novo comando para a ULA, determinando o tipo de operação a ser realizada sempre que a *flag onOP* possui valor lógico 1. Para melhor compreensão da implementação do *OPcontrol*

A Tabela 4 mostra para cada instrução, a determinação de qual operação deverá ser realizada, pela Unidade Lógica Aritmética, retornando o sinal de saída equivalente.

Tabela 4 – Sinais de saída do *OPcontrol*

| Operação | Sinal de saída |
|----------------------------|----------------|
| Adição | 0000 |
| Subtração | 0001 |
| <i>and</i> | 0010 |
| <i>or</i> | 0011 |
| <i>not</i> | 0100 |
| <i>Shift left logical</i> | 0101 |
| <i>Shift right logical</i> | 0110 |
| <i>Branch equal</i> | 0111 |
| <i>Branch not equal</i> | 1000 |
| <i>Branch less zero</i> | 1001 |
| <i>Set less than</i> | 1010 |
| <i>Set greater than</i> | 1011 |
| Multiplicação | 1100 |

O Código 4.3 apresentado a seguir mostra a implementação do *OPcontrol*.

```

1  module OPcontrol(nfuncao, onOP, controle);
2      input [4:0] nfuncao;
3      input onOP;
4      output reg [3:0] controle;
5
6      parameter adc = 5'b00000, sub = 5'b00001, adci = 5'b00010, subi = 5'b00011, e =
7          5'b00100,
8          ou = 5'b00101, n = 5'b00110, lowo = 5'b00111, stwo = 5'b01000, slet = 5'b01011,
9          sril = 5'b01100, beq = 5'b01111, bneq = 5'b10000, blz = 5'b10001, slet = 5'b10010
10         ,
11         sgrt = 5'b10011, mult = 5'b11000, multi = 5'b11001;
12
13     always @(*) begin
14         if(onOP) begin
15             case (nfuncao)
16                 adc: controle = 4'b0;
17                 sub: controle = 4'b1;
18                 adci: controle = 4'b0;
19                 subi: controle = 4'b1;
20                 e: controle = 4'b10;
21                 ou: controle = 4'b11;
22                 n: controle = 4'b100;
23                 lowo: controle = 4'b0;
24                 stwo: controle = 4'b0;
25                 slet: controle = 4'b101;
26                 sril: controle = 4'b110;
27                 beq: controle = 4'b111;

```

```

26          bneq: controle = 4'b1000;
27          blz: controle = 4'b1001;
28          slet: controle = 4'b1010;
29          sgrt: controle = 4'b1011;
30          mult: controle = 4'b1100;
31          multi: controle = 4'b1100;
32          default: controle = 4'bx;
33          endcase
34      end
35  else controle = 5'bx;
36 end
37 endmodule

```

Código 4.3 – *OPcontrol*

4.4.5 Unidade Lógica Aritmética

A Unidade Lógica Aritmética é o componente onde ocorrem todas as operações lógicas e aritméticas. Tem como as entradas os dados, que estavam armazenados nos registradores do banco ou de um imediato estendido e um sinal vindo do *OPcontrol*, indicando o tipo de operação que deve ser realizada. A ULA contém 3 *flags* como saída, além do resultado, o *zero* e o *negativo*, que serão utilizados pelo *skip* para realizar o tratamento de saltos condicionais e o *overflow*, que é acionado quando o resultado da ULA tiver o tamanho superior à 32 *bits*. A implementação da ULA é apresentada no Código 4.4.

```

1 module ULA (controle, DA, DB, ULAresult, negativo, zero, overflow);
2   input [3:0] controle;
3   input [31:0] DA;
4   input [31:0] DB;
5   output reg [31:0] ULAresult;
6   output zero;
7   output negativo;
8   integer i;
9   reg [63:0] multResult;
10  output reg overflow;
11  reg [32:0] resultOF;
12
13  parameter adc = 4'b0, sub = 4'b1, e = 4'b10, ou = 4'b11, n = 4'b100,
14    slet = 4'b101, sril = 4'b110, beq = 4'b111, bneq = 4'b1000,
15    blz = 4'b1001, slet = 4'b1010, sgrt = 4'b1011, mult = 4'b1100;
16
17  always @ (controle, DA, DB) begin
18    case(controle)
19      adc: begin
20        resultOF = DA + DB; //ADC
21        overflow = 0;
22        if(resultOF[32] == 1) begin
23          overflow = 1;
24          resultOF = 0;
25        end
26        ULAresult = resultOF;
27      end
28
29      sub: begin //sub
30        ULAresult = DA - DB;
31      end
32
33      slet: begin //slet
34        ULAresult = DA & DB;
35      end
36
37      sril: begin //sril
38        ULAresult = DA | DB;
39      end
40
41      beq: begin //beq
42        ULAresult = DA == DB ? 1 : 0;
43      end
44
45      bneq: begin //bneq
46        ULAresult = DA != DB ? 1 : 0;
47      end
48
49      blz: begin //blz
50        if(DA == 0) begin
51          ULAresult = DB;
52        end
53      end
54
55      sgrt: begin //sgrt
56        ULAresult = DA > DB ? 1 : 0;
57      end
58
59      mult: begin //mult
60        resultOF = DA * DB;
61        if(resultOF > 32) begin
62          overflow = 1;
63          resultOF = 0;
64        end
65        ULAresult = resultOF;
66      end
67
68      multi: begin //multi
69        resultOF = DA / DB;
70        if(resultOF > 32) begin
71          overflow = 1;
72          resultOF = 0;
73        end
74        ULAresult = resultOF;
75      end
76
77      default: begin //default
78        ULAresult = 0;
79      end
80    endcase
81  end
82
83  assign zero = resultOF[31];
84  assign negativo = resultOF[30];
85
86  endmodule

```

```

31                     overflow = 0;
32                     end
33             e: begin
34                 ULAResult = DA & DB; //AND
35                     overflow = 0;
36                     end
37             ou: begin
38                 ULAResult = DA | DB; //OR
39                     overflow = 0;
40                     end
41             n: begin ULAResult = !DA;
42                     overflow = 0;
43                     end
44             sril: begin
45                 ULAResult = DA >> DB; //set right logical
46                     overflow = 0;
47                     end
48             slet: begin
49                 ULAResult = DA << DB;
50                     overflow = 0;
51                     end
52             beq: begin
53                 if (DA == DB) ULAResult = 0; //branch equal
54                 else ULAResult = 1;
55                     overflow = 0;
56                     end
57             bneq: begin
58                 if(DA != DB) ULAResult = 1; //branch not equal
59                 else ULAResult = 0;
60                     overflow = 0;
61                     end
62             blz: begin
63                 ULAResult = DA;
64                     overflow = 0;
65                     end
66             sgrt: begin
67                 if (DA > DB) ULAResult = 1; //set greater than
68                 else ULAResult =0;
69                     overflow = 0;
70                     end
71             slet: begin
72                 if (DA < DB) ULAResult = 1; //set less than
73                 else ULAResult =0;
74                     overflow = 0;
75                     end
76             mult: begin
77                 overflow = 0;
78             multResult = DA * DB;
79                 i = 32;
80                 while(i <= 63) begin
81                     if (multResult[i]) begin
82                         overflow = 1;
83                         multResult = 0;
84                     end
85                     i= i + 1;
86                 end
87             ULAResult = multResult [31:0];
88                     end
89             endcase
90         end

```

```

91
92     assign zero = (ULArresult==0);
93     assign negativo = ($signed (ULArresult) < 0);
94
95 endmodule

```

Código 4.4 – Unidade Lógica Aritmética

Ao trabalhar apenas com números positivos, a geração de *overflow* seria mais comum nas operações de soma e de multiplicação.

Para a soma, o *overflow* seria dado apenas por 1 *bit* a mais do que o definido na arquitetura, que é o caso da soma de 2 números de 32 *bits*, que resultaria em um de 33 *bits*, logo, foi utilizado um registrador auxiliar de 33 *bits*, e se o *bit* mais significativo fosse 1, seria acionada a *flag* de *overflow*.

Para a multiplicação, o *overflow* poderia ser gerado por um resultado de até 64 *bits*, portanto, foi utilizado um registrador de 64 *bits* e criado um laço que verificava *bit* a *bit*, do 64º até o 33º, se existia algum *bit* 1.

4.4.6 Program Counter

O *Program Counter* (PC) é o componente responsável por fornecer o endereço da próxima instrução, neste módulo também será definida a operação de controle HLT, que paralizará o processamento da CPU. O *endereço* de entrada é a saída de um multiplexador (mostrado no Código D.1, que seleciona o valor atual do PC ou o endereço de desvio obtido pelo *skip*. No caso de uma instrução *jump*, o endereço será igual o inserido como entrada, caso contrário, o endereço de entrada será somado com 1, obtendo a posição da próxima instrução na memória de instruções. A entrada *reset*, irá retornar o processamento para a primeira instrução da memória. A implementação deste módulo é apresentado no Código 4.5.

```

1 module programCounter (ck, jump, endereco, reset, hlt, nHLT, OUTendereco);
2     input ck, jump, hlt, nHLT, reset;
3     input [8:0] endereco;
4     reg [8:0] address;
5     output reg [8:0] OUTendereco;
6
7
8     always @ (posedge ck) begin
9         address = endereco;
10        if(jump)
11            address = endereco;
12        else if(nHLT)
13            address = endereco + 1;
14        else if(hlt) begin
15            end
16        else if(reset) address = 0;
17        else address = endereco + 1;
18
19        OUTendereco <= address;

```

```

20
21      end
22
23 endmodule

```

Código 4.5 – *Program Counter*

Existem também 2 entradas, o *reset* e o *nHLT*, que "simbolizam" são os botões presentes no FPGA. Sempre que o *reset* for pressionado, retornará ao início do processamento (no endereço zero da memória de instruções) e o *nHLT*, é utilizado após uma instrução HLT, pois esta paraliza o processamento, então ao ser pressionado, o processamento é retomado.

4.4.7 Skip

O *skip* é também um controle auxiliar à Unidade de Controle, responsável pelo tratamento de instruções de salto. No caso de saltos condicionais, o *skip* analisará os valores lógicos das entradas *flags zero* e *negativo* obtidas pela ULA, em caso de um *branch equal*, se a *flag zero* for 1 o desvio será tomado, o mesmo correrá se a *flag* for 0 para *branch not equal*; já para *branch less than zero*, o desvio será tomado toda vez que o *negativo* possuir valor lógico 1. Esse componente terá também como entrada o endereço de desvio, contido na própria instrução, em casos de desvio tomado, a saída será esta entrada, caso contrário, será 0. Para instruções de desvios incondicionais, a *flag jump* é acionada e a saída será sempre o endereço de entrada. A implementação é apresentada pelo Código 4.6.

Por final, o endereço da próxima instrução será enviada ao PC. A implementação do *skip* é mostrado no Código 4.6.

```

1 module skip (ONcontrole1, controle, jump, pc, endereco, zero, negativo, endout);
2     input ONcontrole1;
3     input [4:0] controle;
4     input [8:0] endereco, pc;
5     input zero, negativo;
6     output reg jump;
7     output reg [8:0] endout;
8     reg [8:0] add;
9
10    parameter beq = 5'b01111, bneq = 5'b10000, blz = 5'b10001, jmp = 5'b01101, jmpr =
11        5'b01110, jal = 5'b11010;
12
13    always @ (*) begin
14        add = pc;
15        if (ONcontrole1) begin
16            case (controle)
17                beq:
18                    if(zero==1) begin
19                        endout = add + endereco;
20                        jump = 0;
21                    end
22                else begin

```

```

23                     endout = add;
24                     jump = 0;
25                 end
26             bneq:
27                 if(zero == 0) begin
28                     endout = add + endereco;
29                     jump = 0;
30                 end
31                 else begin
32                     endout = add;
33                     jump = 0;
34                 end
35             blz:
36                 if(negativo==1) begin
37                     endout = add + endereco;
38                     jump = 0;
39                 end
40                 else begin
41                     endout = add;
42                     jump = 0;
43                 end
44             jmp: begin
45                 endout = endereco;
46                 jump = 1;
47             end
48             jmp: begin
49                 endout = endereco;
50                 jump = 1;
51             end
52             jal: begin
53                 endout = endereco;
54                 jump = 1;
55             end
56         endcase
57     end
58 end
59 endmodule

```

Código 4.6 – *Skip*

4.4.8 Memória de Dados

A memória de dados é o componente responsável pelo armazenamento de dados, que será acessada apenas pelas instruções *store word* e *load store*.

Neste bloco, há um sinal de controle, *ONescrita*, que quando acionada, permite a escrita do dado armazenado no registrador especificado na instrução, no endereço calculado na ULA. Quando a *flag* é 0, habilita apenas a leitura. A implementação é apresentada no Código 4.7.

```

1 module MEMdados(ck, ONescrita, dadoEscrita, endereco, dadoLeitura);
2     input ck, ONescrita;
3     input [5:0] endereco;
4     input [31:0] dadoEscrita;
5     output reg [31:0] dadoLeitura;
6     reg [31:0] memoria [31:0];

```

```

7      reg [8:0] address;
8
9      always @ (posedge ck) begin
10         if (ONescrita) begin
11             memoria[endereco] = dadoEscrita;
12             dadoLeitura = 0;
13             address = endereco;
14         end
15         else
16             dadoLeitura = memoria[address];
17     end
18 endmodule

```

Código 4.7 – Memória de Dados

4.4.9 Controlador de Saída

O controlador de saída tem por finalidade fazer o controle dos dados que serão mostrados no FPGA. Para as instruções *OUT*, há um sinal de controle denominado *outputEnable* que é acionado, obtendo um valor armazenado em um registrador do Banco de Registradores e permitindo a saída deste dado, para que após a comunicação com a placa seja possível visualizar os resultados das operações no ambiente externo. O Código 4.8 apresenta a implementação desse módulo.

```

1 module outputController (outputEnable, data, out);
2   input outputEnable;
3   input [31:0] data;
4   output reg [31:0] out;
5
6   always @ (*) begin
7     if(outputEnable)
8       out = data;
9     else
10       out = 0;
11   end
12 endmodule

```

Código 4.8 – Controlador de saída

4.4.10 Multiplexadores

Os multiplexadores são chamados também de seletores de dados (1). Eles deverão selecionar um dos dados de entrada para ser a saída do circuito, essa seleção ocorre a partir dos sinais de controle. Na Unidade de processamento desenvolvida há 5 multiplexadores:

- *mux5bits*: seleciona em qual registrador, do Banco de registradores, especificado no instrução ocorrerá a escrita do dado.
- *mux1*: seleciona qual será o segundo dado a ser operado na Unidade Lógica Aritmética, podendo ser um imediato ou um dado lido de um registrador.

- *muxEndereco*: seleciona o endereço que será enviado ao PC, que poderá ser o valor atual do PC ou o endereço de saída do *skip*, neste multiplexador há uma *flag jump*, que indicará se, em casos de desvios, o salto é incondicional. (D.1)
- *muxEnd*: Utilizado apenas em desvios incondicionais, seleciona se o endereço provém de um registrador (para *jump register*) ou da própria instrução (para *jump and link* e *jump*).
- *mux4*: seleciona qual dado deverá ser escrito no Banco de registradores.

Estes multiplexadores farão a seleção de dados a partir dos sinais de saída da Unidade de Controle, que será abordado na seção

4.4.11 Unidade de Controle

A Unidade de Controle implementada foi pelo método *hardwired*, onde cada sinal de controle de saída deste módulo depende da entrada do circuito, que é o *opcode*, responsável por informar qual a instrução que será executada, e de acordo com esse dado, a UC definirá os sinais.

A Unidade de Controle envia 12 sinais para os demais blocos do processador, que serão enumerados a seguir.

- *writeDataMem*: Sinaliza se um dado deve ser escrito ou lido na memória de dados.
- *onWriteReg*: Sinaliza se um dado deve ser escrito ou apenas lido do Banco de registradores.
- *onop*: Sinaliza se uma instrução deve utilizar a ULA.
- *onskip*: Sinaliza se a instrução é um desvio seja condicional ou incondicional.
- *selectSize*: Sinaliza o tamanho do imediato a ser estendido pelo extensor de sinal.
- *selectDado*: Seleciona o dado a ser escrito no Banco de Registradores.
- *selectD*: Seleciona qual será o segundo dado a ser operado na Unidade Lógica Aritmética.
- *ctrlM5*: Seleciona qual o registrador do banco o dado inserido deve ser escrito.
- *muxEnd*: Seleciona qual será o endereço a ser enviado ao PC.
- *endPCorReg*: Utilizada apenas em desvios incondicionais, seleciona de onde provém o endereço da próxima instrução.
- *HLT*: Sinaliza o PC, quando uma instrução *HLT* é executada.

- *outputEnable*: Sinaliza se uma instrução é de saída de dados.

A Tabela 5 mostra os sinais de controle que são definidos no circuito da Unidade de Controle, os módulos para onde serão enviados e sua funcionalidade para cada valor lógico.

Tabela 5 – Sinais da Unidade de Controle

| Módulo | Sinal | Função |
|-------------------------|--------------|---|
| Memória de Dados | writeDataMem | 0 - Leitura 1 - Escrita |
| Banco de Registradores | onWriteReg | 0 - Leitura 1 - Escrita |
| <i>OPcontrol</i> | onop | 0 - Funcionamento desabilitado 1 - Funcionamento habilitado |
| <i>skip</i> | onskip | 0 - Funcionamento desabilitado 1 - Funcionamento habilitado |
| Extensor de Sinal | selectSize | 00 - Imediato de 17 bits 01 - Imediato de 22 bits 10 - Imediato de 9 bits 11 - Imediato de 18 bits |
| <i>mux4</i> | selectDado | 00 - Resultado da ULA 01 - Dado de Leitura de um registrador 10 - Leitura da memória de dados 11 - Valor de um imediato estendido |
| <i>mux1</i> | selectD | 0 - Dado de um registrador 1 - Imediato estendido |
| <i>MUX5</i> | ctrlM5 | 00 - 1º registrador especificado na instrução 01 - 2º registrador especificado na instrução 10 - 3º registrador especificado na instrução |
| <i>muxEndereco</i> | muxEnd | 0 - Endereço de saída do <i>skip</i> 1 - Endereço atual do PC |
| <i>muxEnd</i> | endPCorReg | 0 - Endereço contido na instrução 1 - Endereço contido no registrador |
| <i>Program Counter</i> | HLT | 0 - Não paraliza o processamento 1 - Paraliza o processamento |
| <i>outputController</i> | outputEnable | 0 - Saída será zero 1 - Saída terá o mesmo valor do dado de entrada |

A implementação da Unidade de Controle está em anexo (E.1). Após a implementação de cada bloco componente da Unidade de Processamento, foi realizada a integração destes.

4.5 Comunicação com FPGA

Após realizar a implementação da Unidade de Processamento com a Unidade de Controle integrada, o próximo passo foi realizar a comunicação com o ambiente externo,

para que o usuário possa interagir com a CPU.

4.5.1 Conversor BCD

Para que o valor resultante em binário possa ser exibido nos *displays*, o valor em binário deve ser convertido para BCD para que os algarismos de cada resultado sejam divididos entre os *displays* disponíveis. A implementação é apresentada no código em anexo ([F.1](#)).

4.5.2 *Display* de 7 segmentos

O FPGA possui 8 *displays*, com o conversor BCD implementado, cada sequência de 4 *bits* representa um algarismo de 0 até 9, que será exibido como saída em cada *display*, portanto poderão ser exibidos números com até 8 algarismos, em casos de *overflow*, um *LED* é aceso e o valor em todos os *displays* deverá ser 0. O código em anexo ([G.1](#)) apresenta a implementação do *display* de 7 segmentos, que possui os valores de saída invertidos devido a arquitetura do FPGA.

4.5.3 Demais componentes do FPGA

Os componentes utilizados para a comunicação, além dos *displays*, são os: *switches*, *LEDs* e *KEYS*.

- *Switches*: são as 18 alavancas presentes no FPGA, onde serão inseridos os dados para serem operados na Unidade de Processamento, cada uma representa um *bit*, sempre quando levantada, representa o valor lógico 1, caso contrário, o valor 0. Portanto, sempre os valores de entrada na placa possuirão 18 *bits*, necessitando ser estendidos para 32, para que siga as especificações da arquitetura.
- *LEDs*: Existem 18 LEDs vermelhos e 8 verdes, os primeiros, (LEDR0 a LEDR8), foram escolhidos para representar o endereço atual do programa e os segundos para sinalizar alguns casos, sendo do limite direito para o esquerdo, as *flags* de *overflow*, *zero* e *negativo* (LEDG0 a LEDG2).
- *KEYS*: São os 4 botões existentes na placa, sendo 3 delas programadas para este projeto, o KEY[0] utilizado para dar o pulso de *clock*, KEY[1] para o *reset* e KEY[2] para o *nHLT*, como o botão possui o valor lógico 1, ele deverá ser invertido, para que o comando seja dado apenas quando for pressionado. Para otimizar o uso deste botão, foi implementado o *DeBounce* ([H.1](#)), para que cada pulso seja dado de forma precisa.

A figura [9](#) ilustra a utilização desses componentes.

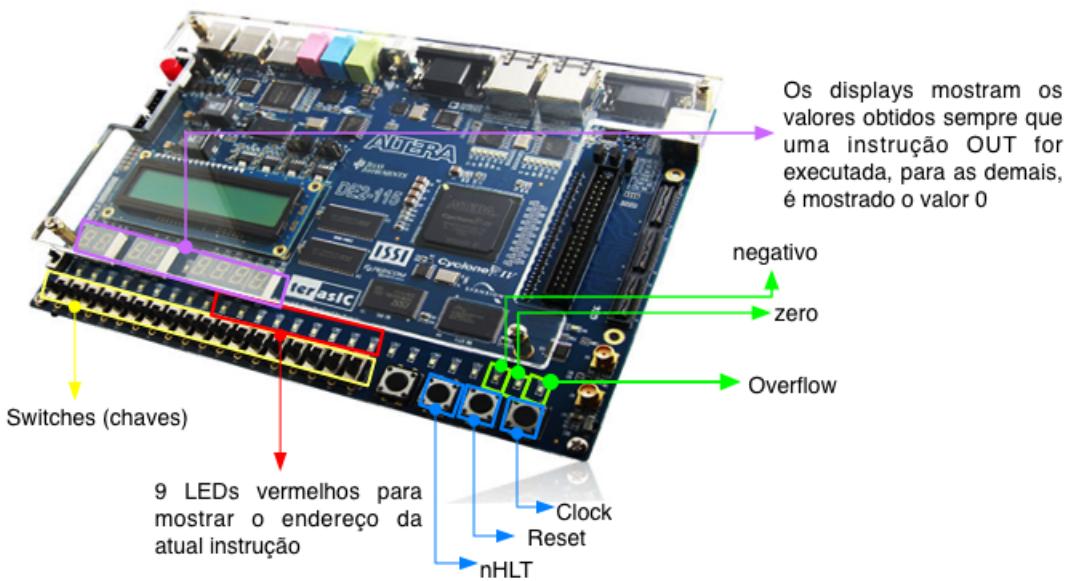


Figura 9 – Processamento do algoritmo 3 no FPGA

4.6 Comunicação com o FPGA

Após realizar o desenvolvimento da Unidade de Processamento (com o uso da memória de dados apresentada na bancada), e dos componentes necessários para realizar a comunicação com o FPGA, foi realizada a integração de todos estes componentes, presente no Código 4.9.

```

1  module unidadeProcessamento(ck, ckin, switches, reset, zero, neg, overflow, sdmilhao,
2    nHLT,
3    smilhao, scmilhar, sdmilhar, endereco, smilhar, scentesimal, sdecimal, sunidade) ;
4
5      output [8:0] endereco;
6      input [17:0] switches;
7      input wire ck, ckin;
8      input nHLT;
9      wire clk;
10     input wire reset;
11     wire [31:0] DA, DB;
12     wire [31:0] instrucao, wd, outIM, outmuxdado;
13     wire onWriteReg, writeDataMem, outputEnable, onop, onskip, muxEnd;
14     wire hlt, endPCorReg;
15     wire [1:0] ctrlM5, selectDado, selectSize;
16     wire selectD;
17     wire [4:0] gravaEnd, reg1, ende;
18     wire [31:0] outmux, outIUnit;
19     wire [31:0] dadoLeitura;
20     wire [8:0] outskip, skipEnd, pcin;
21     wire j, jmp;
22     output wire zero, neg;
23     wire [3:0] controle;
24     output wire overflow;

```

```

25     wire [31:0] outOC;
26     wire [3:0] dmilhao , milhao , cmilhar , dmilhar , centesimal , milhar , decimal ,
27         unidade;
28     output [6:0] sdmilhao , smilhao , scmilhar , sdmilhar , smilhar , scentesimal ,
29         sdecimal , sunidade;
30     wire [31:0]result;
31
32
33
34     assign neg_nHLT = ~nHLT;
35     assign neg_reset = ~reset;
36
37
38
39     DeBounce (.clk(ck),
40                 .n_reset(1),
41                 .button_in(ckin),
42                 .DB_out(clk)
43 );
44
45     MEMinstrucao(.ck(clk),
46                 .endereco(endereco),
47                 .saida(instrucao));
48
49     UC (.opcode(instrucao[31:27]),
50             .onWriteReg(onWriteReg),
51             .writeDataMem(writeDataMem),
52             .outputEnable(outputEnable),
53             .onop(onop),
54             .onskip(onskip),
55             .muxEnd(muxEnd),
56             .ctrlM5(ctrlM5),
57             .selectSize(selectSize),
58             .selectDado(selectDado),
59             .selectD(selectD),
60             .HLT(hlt),
61             .endPCorReg(endPCorReg)
62 );
63     programCounter PC (.ck(clk),
64                         .jump(jmp),
65                         .endereco(pcin),
66                         .nHLT(neg_nHLT),
67                         .reset(neg_reset),
68                         .OUTendereco(endereco),
69                         .hlt(hlt)
70 );
71
72     muxEndereco (.ctrl(muxEnd),
73                         .jIN(j),
74                         .EA(outskip),
75                         .EB(endereco),
76                         .jOUT(jmp),
77                         .Eout(pcin)
78 );
79
80     mux5bits (.ctl(ctrlM5),
81                         .DA(instrucao[26:22]),
82                         .DB(instrucao[21:17]),

```

```
83                     .DC(instrucao[16:12]),
84                     .outD(gravaEnd)
85     );
86
87     regFile Banco_de_Registradores(.readReg1(instrucao[26:22]),
88                                     .readReg2(instrucao[21:17]),
89                                     .writeReg(onWriteReg),
90                                     .writeData(outmux),
91                                     .writeAddress(gravaEnd),
92                                     .readData1(DA),
93                                     .readData2(DB),
94                                     .ck(clk)
95     );
96
97     extensorSinal (.selectSize(selectSize),
98
99
100            .imm17(instrucao[16:0]),
101
102            .imm22(instrucao[21:0]),
103
104
105            .imm9(endereco),
106
107
108            .imm18(switches),
109
110
111            .out32(wd)
112     );
113
114     MUX mux1(.select(selectD),
115               .DA(DB),
116               .DB(wd),
117               .outMUX(outIM)
118     );
119     OPcontrol (.nfuncao(instrucao[31:27]),
120                 .onOP(onop),
121                 .controle(controle)
122     );
123     ULA Unidade_Logica_Aritmetica(.controle(controle),
124               .DA(DA),
125               .DB(outIM),
126               .ULAreult(result),
127               .negativo(neg),
128               .zero(zero),
129               .overflow(overflow)
130     );
131     muxEnd PCorREG(.ctrl(endPCorReg),
132                      .EA(instrucao[8:0]),
133                      .EB(DA[8:0]),
134                      .Eout(skipEnd));
135
136     skip (.ONcontrole1(onskip),
137               .controle(instrucao[31:27]),
138               .jump(j),
139               .pc(endereco),
140               .endereco(skipEnd),
141               .zero(zero),
142               .negativo(neg),
```

```

143                               .endout(outskip));
144
145     MEMdados (.ck(clk),
146
147         .ONescrita(writeDataMem),
148
149         .dadoEscrita(DB),
150
151
152         .endereco(result[8:0]),
153
154
155         .dadoLeitura(dadoLeitura)
156     );
157
158
159     mux4 (.select(selectDado),
160             .DA(result),
161             .DB(DA),
162             .DC(dadoLeitura),
163             .DD(wd),
164             .outD(outmux)
165     );
166
167     outputController (.outputEnable(outputEnable),
168                         .data(DA),
169                         .out(outOC));
170
171
172     binToBCD(.dataBin(outOC),
173                 .dmilhao(dmilhao),
174                 .milhao(milhao),
175                 .cmilhar(cmilhar),
176                 .dmilhar(dmilhar),
177                 .milhar(milhar),
178                 .centesimal(centesimal),
179                 .decimal(decimal),
180                 .unidade(unidade));
181
182     display7segmentos H (.numero(dmilhao), .outDisplay(sdmilhao));
183
184     display7segmentos G (.numero(milhao), .outDisplay(smilhao));
185
186     display7segmentos F (.numero(cmilhar), .outDisplay(scmilhar));
187
188     display7segmentos E (.numero(dmilhar), .outDisplay(sdmilhar));
189
190     display7segmentos D (.numero(milhar), .outDisplay(smilhar));
191
192     display7segmentos C (.numero(centesimal), .outDisplay(scentesimal));
193
194     display7segmentos B (.numero(decimal), .outDisplay(sdecimal));
195
196     display7segmentos A (.numero(unidade), .outDisplay(sunidade));
197
198
199 endmodule

```

5 Resultados e Discussões

Para verificar se o funcionamento da Unidade de Processamento está correto, foram realizados testes de simulação em formas de onda, com a utilização de 3 algoritmos e posteriormente, realizada a comunicação com o FPGA. A seguir serão apresentados os algoritmos executados (onde cada item enumerado se refere ao endereço desta instrução na memória de instruções), a simulação com os *waveforms* e a comunicação com o FPGA.

5.1 Algoritmo 1

Primeiramente, foi realizado o teste do algoritmo que calcula o fatorial de um número inserido como entrada através dos *switches*, nele são testadas as instruções *in*, *out*, *load immediate*, *branch equal*, *jump and link*, *jump register*, *load word*, *store word*, *shift right logical* multiplicação, adição, subtração com imediato, HLT e movimentação. O algoritmo é apresentado a seguir.

0. $\text{reg}[1] \leftarrow \text{switches}$, um valor inserido nas chaves é gravado no registrador 1.
1. *Displays* $\leftarrow \text{reg}[1]$, o valor contido no registrador 1 é exibido nos *displays*.
2. $\text{reg}[3] \leftarrow 2$, o valor 2 é carregado no registrador 3.
3. $\text{reg}[4] \leftarrow 0$, o valor 0 é carregado no registrador 4.
4. $\text{reg}[2] \leftarrow \text{reg}[1]$, o valor do registrador 1 é copiado para o registrador 2.
5. *goto* 6., é realizado um salto incondicional para o endereço 6 e o endereço atual é gravado em um registrador de uso específico ($\text{reg}[0]$).
6. $\text{reg}[1] \leftarrow \text{reg}[1] - 1$, o valor contido no registrador 1 é subtraído com o imediato 1 e gravado novamente, no registrador 1.
7. *Displays* $\leftarrow \text{reg}[1]$, o valor do registrador 1 é exibido nos *displays*.
8. $\text{reg}[2] \leftarrow \text{reg}[1] * \text{reg}[2]$, o valor contido no registrador 1 é multiplicado com o valor do registrador 2 e escrito novamente no registrador 2.
9. *Displays* $\leftarrow \text{reg}[2]$, o valor contido no registrador 2 é exibido nos *displays*.
10. *if*($\text{reg}[2] == \text{reg}[4]$) *goto* 18., se o valor contido no registrador 2 for igual ao contido no 4, é realizado um salto para o endereço 18 (esta condição é utilizada para sair do *looping*, em casos de *overflow*).
11. *if*($\text{reg}[1] == \text{reg}[3]$) *goto* 18., se o valor contido no registrador 1 for igual ao contido no 3, é realizado um desvio para o endereço 18 (esta condição é utilizada para sair

do *looping*, caso o valor do fatorial já foi obtido).

12. *return to 5*, é realizado um desvio incondicional para o endereço 5, representa uma instrução de *jump register*, onde o endereço do desvio é o contido no registrador, neste caso, o registrador 0 (de uso específico), que armazenou o endereço do *jump* anterior.

18. $\text{reg}[5] \leftarrow \text{switches}$, um valor inserido nas chaves, é gravado no registrador 5.

19. $\text{reg}[7] \leftarrow \text{reg}[2] + \text{reg}[5]$, o valor contido no registrador 2 é somado com o contido no 5 e gravado no registrador 7.

20. *Displays* $\leftarrow \text{reg}[7]$, o valor contido no registrador 7 é exibido nos *displays*

21. $\text{Memória}[\text{reg}[3] + 5] \leftarrow \text{reg}[7]$, o valor contido no registrador 3 é somado com o imediato 5, sendo calculado o endereço efetivo da posição da memória de dados, onde será gravado o valor contido no registrador 7.

22. *HLT*, interrompe o processamento até que um botão seja pressionado.

23. $\text{reg}[8] \leftarrow \text{Memória}[\text{reg}[3] + 5]$, o valor contido no registrador 3 é somado com o imediato 5, sendo calculado o endereço efetivo da posição da memória de dados do dado que será gravado no registrador 8.

24. *Displays* $\leftarrow \text{reg}[8]$, o valor contido no registrador 8 é exibido nos *displays*.

25. $\text{reg}[9] \leftarrow \text{reg}[8] \gg 2$, o valor contido no registrador 8 é deslocado 2 *bits* à direita e o resultado é gravado no registrador 9.

26. *Displays* $\leftarrow \text{reg}[9]$, o valor armazenado no registrador 9 é exibido nos *displays*.

A Figura 10 mostra o *waveform* da simulação do algoritmo acima. Tendo como *output* o endereço da instrução (referente a cada item anterior), o resultado da ULA e as *flags* zero, *overflow* e negativo.

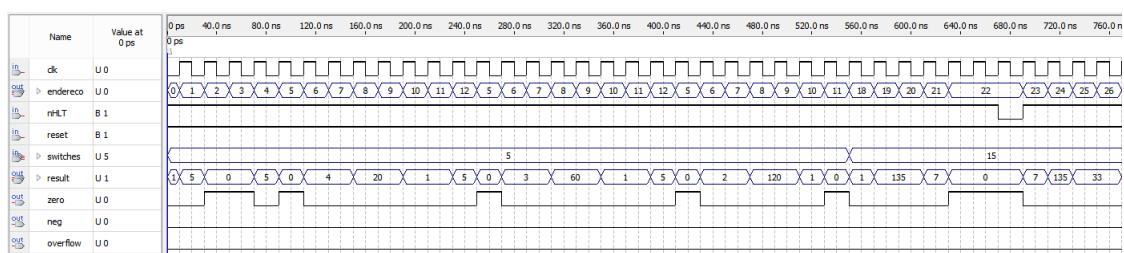


Figura 10 – *waveform* da simulação do cálculo de fatorial

Como pode ser observado, esta simulação é realizada com a inserção do valor 5 nos *switches*, a comunicação com o FPGA é mostrada na Figura 11.

De acordo com a análise do *waveform* e das imagens do FPGA, se pode observar os seguintes resultados.

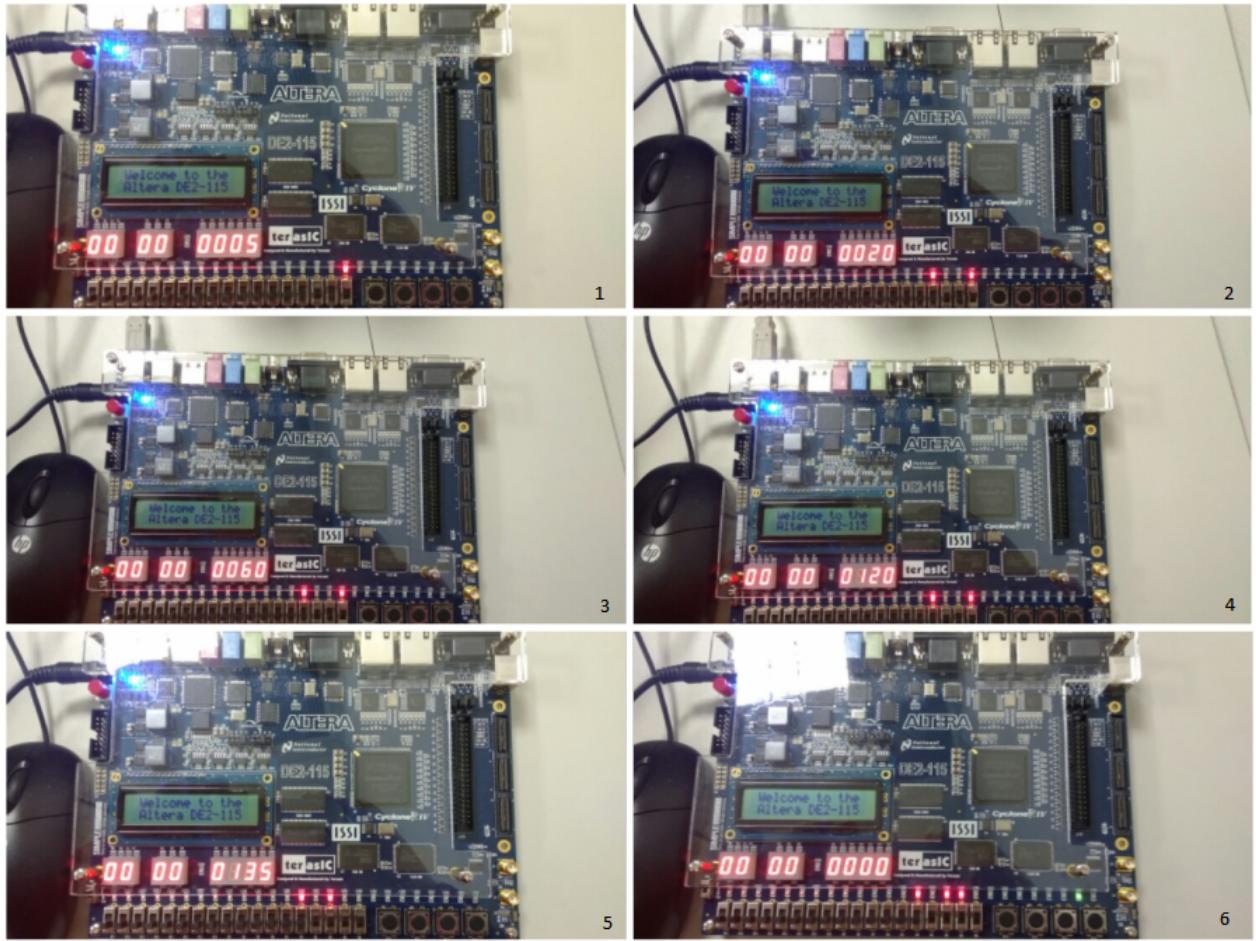


Figura 11 – Simulação do algoritmo 1 no FPGA

- Imagem 1: Os LEDs vermelhos representam o endereço atual de cada instrução que está sendo executada. Como mostrado na imagem 1, o valor 5 é inserido na chave e a instrução contida no endereço 1 é um *OUT*, portanto, o valor é mostrado no *display*.
- Imagens 2, 3, 4: O valor 5 é decrementado e multiplicado pelo resultado anterior (imagens 2, 3 e 4), até que seja obtido o resultado do fatorial
- Imagem 5: Após a realização do cálculo, o valor 15 é inserido (endereço 18), somado com o valor do fatorial e o resultado obtido é mostrado (endereço 20).
- Imagem 6: Mostra a instrução contida no endereço 22, que é um *HLT*, no qual ocasiona um "travamento", onde mesmo com o pulso de *clock* sendo enviado, o processamento é interrompido até que o botão (*KEY2*, que representa o sinal *nHLT*) seja pressionado, dando continuidade ao programa. Este comportamento pode ser observado no *waveform*, onde é mostrado o momento em que o valor de *nHLT* é invertido e a próxima instrução será executada.

Outro teste realizado é a inserção de um valor alto nos *switches*, no qual o resultado do factorial ultrapassa a quantidade de *bits* que a arquitetura suporta (32 *bits*). o teste foi realizado com a inserção do valor 131075, como mostrado no *waveform* da Figura 12

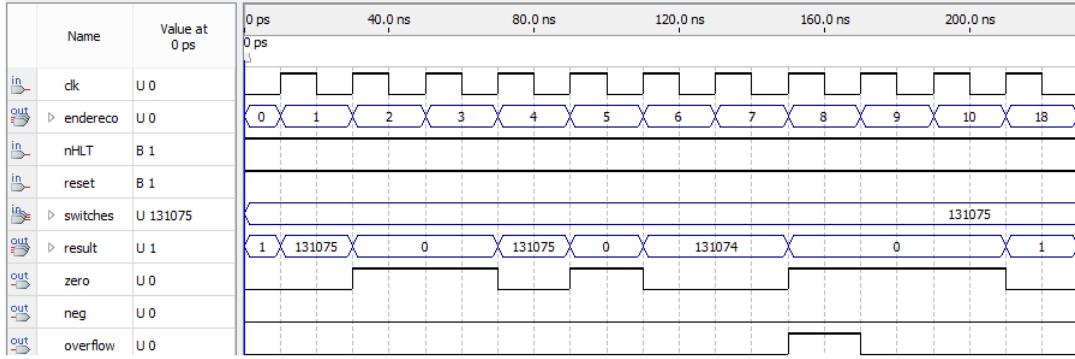


Figura 12 – *Waveform* do cálculo de factorial com geração de *overflow*

Pode ser observado que a multiplicação de 131075 com 131074, gerou um resultado de tamanho maior que 32 *bits*. Na arquitetura desenvolvida, sempre quando um *overflow* for gerado, o resultado da ULA deverá ser zero.

A Figura 13 mostra a simulação no FPGA.

O último LED verde representa *overflow* e o penúltimo, um resultado zero na ULA. Como observado na imagem 2, ao inserir o valor (imagem 1), o resultado da primeira multiplicação (endereço 8) gerou um *overflow*, assim o resultado será zero, podendo notar os acendimentos dos LEDs (circulados em vermelho).

5.2 Algoritmo 2

No algoritmo 2 são feitos os teste para as instruções: *store word*, *load word*, *set greater than*, *shift right logical*, *not*, *NOP* e multiplicação com imediato. O algoritmo 2 é apresentado abaixo.

0. $\text{reg}[1] \leftarrow \text{switches}$, um valor contido nas chaves é armazenado no registrador 1.
1. $\text{Displays} \leftarrow \text{reg}[1]$, o valor contido no registrador 1 é exibido nos *displays*
2. $\text{reg}[6] \leftarrow \text{reg}[1] \gg 2$, o valor contido no registrador 1 é deslocado 2 *bits* à direita e o resultado é gravado no registrador 6.
3. $\text{Displays} \leftarrow \text{reg}[6]$, o valor armazenado no registrador 6 é exibido nos *displays*.
4. $\text{reg}[5] \leftarrow \text{reg}[6] * 16$, o valor contido no registrador 6 é multiplicado com o imediato 16 e o resultado é gravado no registrador 5.
5. $\text{Displays} \leftarrow \text{reg}[5]$, o valor contido no registrador 5 é exibido nos *displays*.

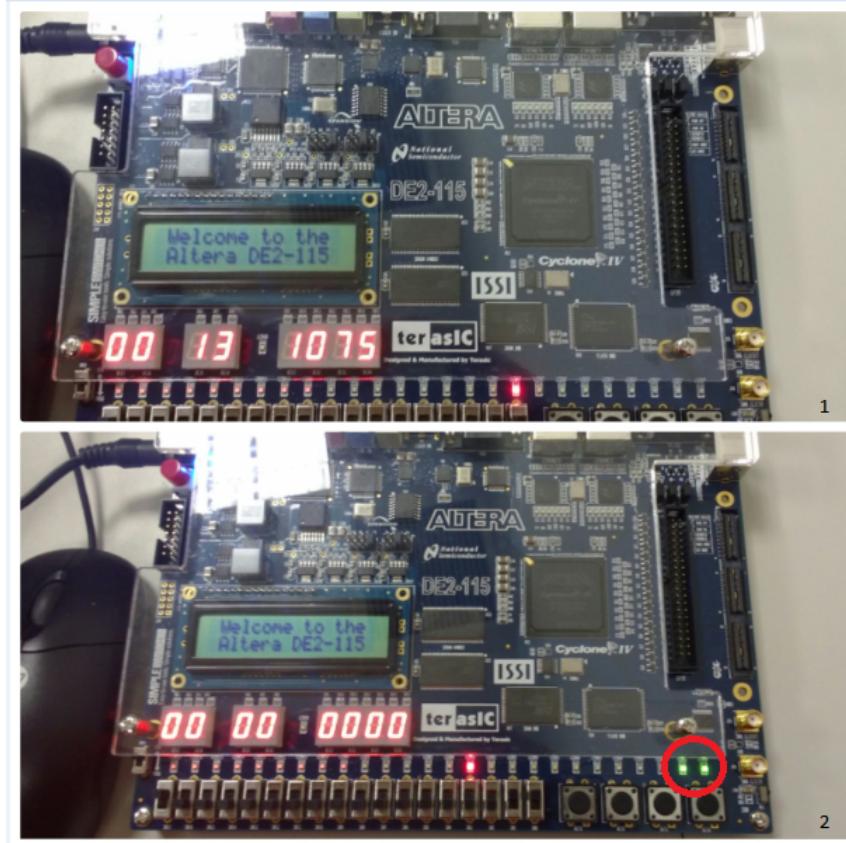


Figura 13 – Sinalização de *overflow* no FPGA

6. $reg[4] \leftarrow reg[5] + 3$, o valor contido no registrador 5 é somado com o imediato 3 e gravado no registrador 4.
7. *Displays* $\leftarrow reg[4]$, o valor contido no registrador 4 é exibido nos *displays*.
8. $reg[7] \leftarrow switches$, um valor contido nas chaves é armazenado no registrador 7.
9. *Displays* $\leftarrow reg[7]$, o valor contido no registrador 7 é exibido nos *displays*.
10. if ($reg[4] > reg[7]$) $reg[5] = 1$, se o valor contido no registrador 4 for maior que o contido no registrador 7, então o valor do registrador 5 é setado para 1.
11. *Displays* $\leftarrow reg[5]$, o valor contido no registrador 5 é exibido nos *displays*.
12. $reg[8] \leftarrow reg[5]$, o valor contido no registrador 5 é negado e gravado no registrador 8.
13. *Displays* $\leftarrow reg[8]$, o valor contido no registrador 8 é exibido nos *displays*.
14. $reg[16] \leftarrow 2$, o valor 2 é carregado no registrador 16.
15. *Displays* $\leftarrow reg[16]$, o valor contido no registrador 16 é exibido nos *displays*.
16. $reg[9] \leftarrow reg[8] + 4$, o valor contido no registrador 8 é somado com o imediato 4 e salvo no registrador 9.

17. *Displays* <- reg[9], o valor contido no registrador 9 é exibido nos *displays*.
18. Memória[reg[16] + 1] <- reg[9], o valor contido no registrador 16 é somado com o imediato 1, sendo calculado o endereço efetivo da memória de dados que gravará o valor contido no registrador 9.
19. *NOP*, não realiza operação alguma durante 1 ciclo de *clock*.
20. *NOP*, não realiza operação alguma durante 1 ciclo de *clock*.
21. *NOP*, não realiza operação alguma durante 1 ciclo de *clock*.
22. reg[11] <- Memória[reg[16] + 1], o valor contido no registrador 16 é somado com o imediato 1, sendo calculado o endereço efetivo da memória de dados que contém o dado a ser salvo no registrador 11.
23. *Displays* <- reg[11], o valor contido no registrador 11 é exibido nos *displays*

A Figura 14 mostra a simulação do *waveform* do algoritmo 2.

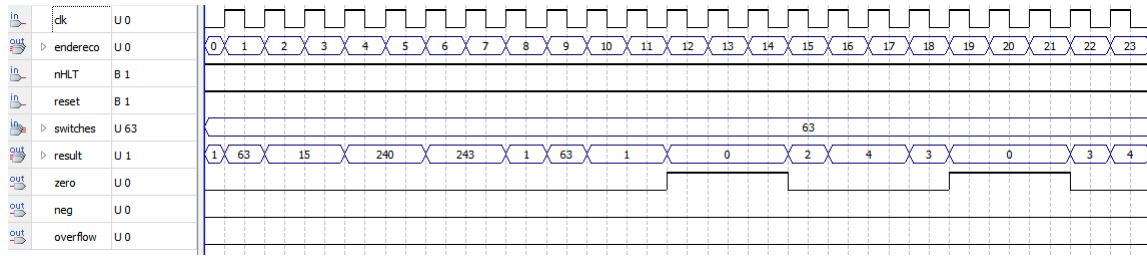


Figura 14 – *waveform* da simulação do algoritmo 2

Nesta figura pode ser observado que o valor 63 é inserido nos *switches*. E ao ser enviado o pulso de *clock*, se dá o processamento das operações com este valor de entrada, como mostrado nas imagens da Figura 15.

De acordo com a análise do *waveform* e das imagens do FPGA, se pode observar os seguintes resultados.

- Imagem 1: O valor 63 é carregado e no *clock* seguinte é mostrado no *display*.
- Imagem 2: O valor 63 é deslocado 2 bits à direita, e é realizada uma operação *OUT* com esse resultado(endereço 3).
- Imagem 3: o valor 15, que é o resultado do deslocamento anterior é multiplicado com o imediato 16, gerando um resultado de 240, que será mostrado no *display* ao ser executada uma instrução *OUT* (endereço 5).
- Imagem 4: o endereço 6 contém uma instrução *adci*, o valor de 240 é somado com o imediato 3, e o resultado de 243 é exibido no *display* no próximo ciclo de *clock*.

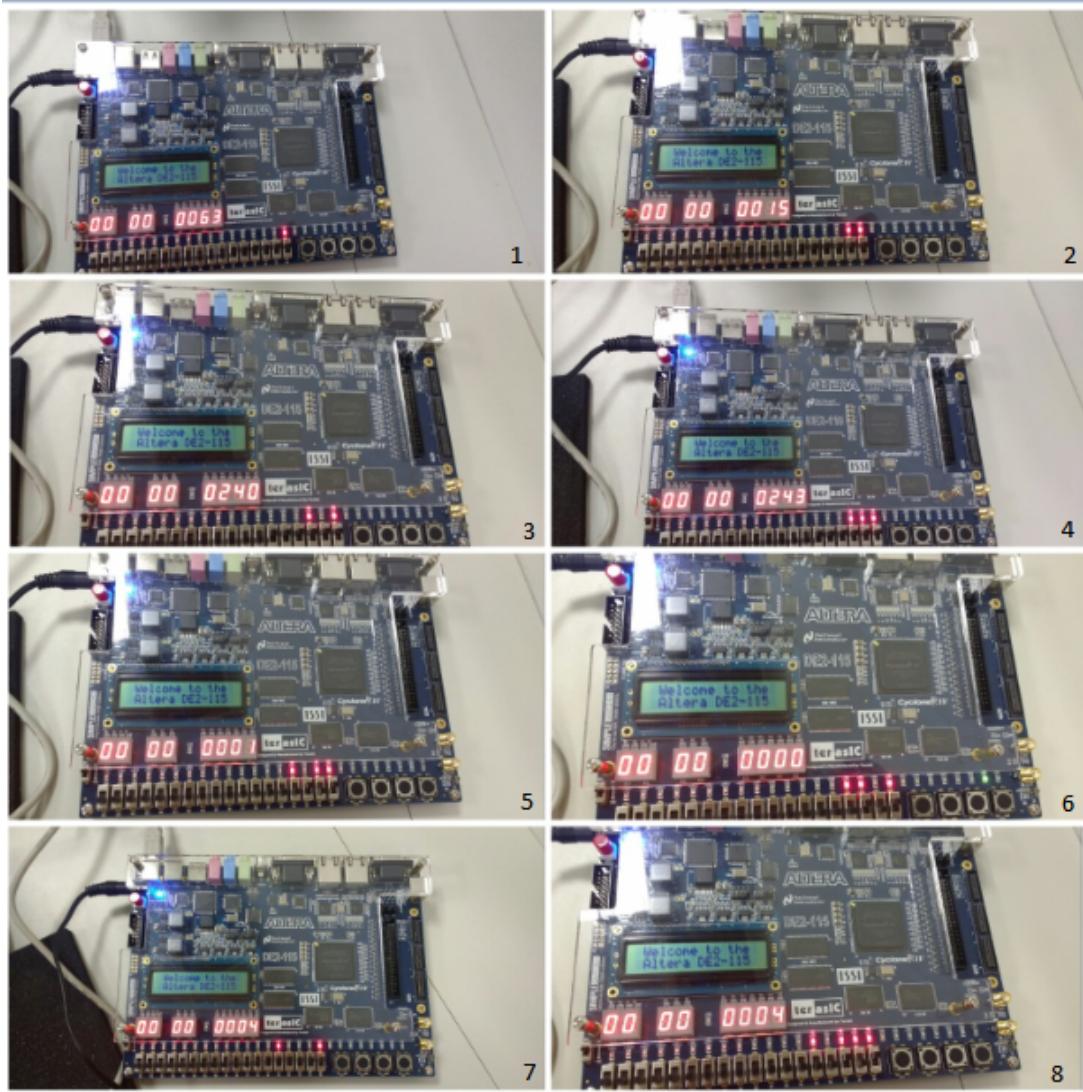


Figura 15 – Processamento do algoritmo 2 no FPGA

- Imagem 5: O valor resultante da soma anterior é comparado com o valor dos *switches*, caso o primeiro for maior que o segundo, o resultado mostrado no *display* deverá 1, como foi observado.
- Imagem 6: Foi realizada uma operação *not* com o resultado anterior, retornando zero como resultado na ULA, podendo observar o valor 0 nos *displays* e o acendimento do penúltimo LED verde (endereço 13).
- Imagem 7: O valor do resultado obtido anteriormente é somado com o imediato 4, e no próximo ciclo, o resultado é exibido nos *displays* (endereço 17). E posteriormente, foi gravada na memória de dados.
- Imagem 8: o valor 4, resultante da operação *adci*, foi buscada na memória de dados e salvo em um registrador e posteriormente, esse registrador foi lido e o valor foi

exibido nos *displays* (endereço 23).

5.3 Algoritmo 3

O algoritmo 3 utiliza, além de algumas instruções já utilizadas nos outros 2 anteriores, o *and*, *or*, *branch not equal*, *jump*, *shift left logical*, *set less than*, *branch less zero* e subtração. O algoritmo 3 é apresentado a seguir.

0. $\text{reg}[1] \leftarrow \text{switches}$, o valor inserido nas chaves é escrito no registrador 1.
1. *Displays* $\leftarrow \text{reg}[1]$, o valor do registrador 1 é mostrado nos *displays*.
2. $\text{reg}[2] \leftarrow 32$, o valor 32 é carregado no registrador 2.
3. *Displays* $\leftarrow \text{reg}[2]$, o valor contido no registrador 2 é exibido nos *displays*.
4. $\text{reg}[10] \leftarrow 63$, o valor 63 é carregado no registrador 10.
5. *Displays* $\leftarrow \text{reg}[10]$, o valor contido no registrador 10 é exibido nos *displays*.
6. $\text{if}(\text{reg}[1] < \text{reg}[2]) \text{ reg}[3] = 1$, se o valor contido no registrador 1 é menor que o contido no registrador 2, o valor do registrador 3 é setado para 1.
7. *Displays* $\leftarrow \text{reg}[3]$, o valor do registrador 3 é exibido nos *displays*.
8. $\text{reg}[4] \leftarrow 1$, o valor 1 é carregado no registrador 4.
9. *Displays* $\leftarrow \text{reg}[4]$, o valor contido no registrador 4 é exibido nos *displays*
10. $\text{if}(\text{reg}[3] \neq \text{reg}[4]) \text{ goto } 14$., se o valor contido no registrador 3 for diferente do contido no 4, é realizado um salto para o endereço 14.
11. $\text{reg}[5] \leftarrow \text{reg}[3] \ll 5$, o valor contido no registrador 3 é deslocado 5 *bits* para a esquerda e gravado no registrador 5.
12. *Displays* $\leftarrow \text{reg}[5]$, o valor contido no registrador 5 é exibido nos *displays*
13. *HALT*, a instrução *HALT*, interrompe o processamento até que um botão seja pressionado.
14. $\text{reg}[6] \leftarrow \text{reg}[1] + 10$, o valor contido no registrador 1 é somado com o imediato 10 e carregado no registrador 6.
15. *Displays* $\leftarrow \text{reg}[6]$, o valor do registrador 6 é exibido nos *displays*.
16. *goto* 20., é realizado um salto incondicional para o endereço 20.
20. $\text{reg}[7] \leftarrow 20$, o valor 20 é carregado no registrador 7.
21. $\text{reg}[8] \leftarrow \text{reg}[1] - \text{reg}[7]$, o valor do registrador 1 é subtraido com o valor contido no registrador 7, esse resultado é gravado no registrador 8.
22. *Displays* $\leftarrow \text{reg}[8]$, o valor contido no registrador 8 é mostrado nos *displays*.

23. $\text{if}(\text{reg}[8] < 0) \text{ goto } 28.$, se o valor contido no registrador 8 for menor que zero, é realizado um desvio para o endereço 28.

24. $\text{reg}[9] \leftarrow \text{reg}[10] \& \text{reg}[1]$, é realizada uma operação lógica *and* com os valores contidos nos registradores 10 e 1, e esse resultado é gravado no registrador 9.

25. $\text{Displays} \leftarrow \text{reg}[9]$, o valor contido no registrador 9 é mostrado nos *displays*.

26. *HLT*, interrompe o processamento até que um botão seja pressionado.

27. *NOP*, não realiza operação alguma durante 1 ciclo de *clock*.

28. $\text{reg}[9] \leftarrow \text{reg}[10] \mid \text{reg}[1]$, é realizada uma operação lógica *or* com os valores contidos nos registradores 10 e 1, e esse resultado é gravado no registrador 9.

29. $\text{Displays} \leftarrow \text{reg}[9]$, o valor contido no registrador 9 é mostrado nos *displays*.

A Figura 16 mostra a simulação do *waveform* para este algoritmo, com o valor 18 como entrada inicial nos *switches*.

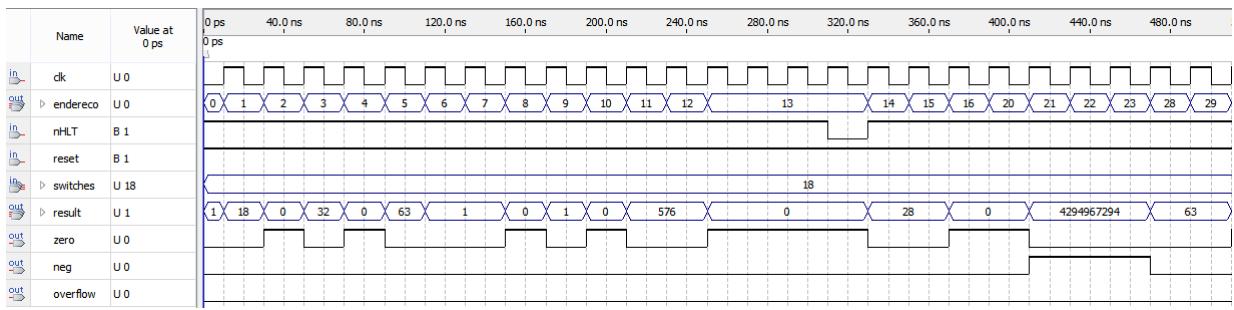


Figura 16 – *Waveform* da simulação do algoritmo 3

A Figura 17 mostra a comunicação da Unidade de Processamento com o FPGA, para visualizar os resultados obtidos com o processamento das operações.

De acordo com a análise do *waveform* e das imagens do FPGA, se pode observar os seguintes resultados.

- Imagem 1: O valor 18 é inserido nas chaves, e posteriormente, é mostrado nos *displays*.
- Imagem 3: O valor é comparado com o valor 63, para verificar se o primeiro é menor que o segundo, como este fato ocorre, então o valor 1 é mostrado no *display* (endereço 7).
- Imagem 2: Como o valor mostrado anteriormente no *display* é 1, o valor 18 é deslocado 5 bits à esquerda, exibindo o resultado no *display*, posteriormente (endereço 12).
- Imagem 4: O endereço 13 é um *HLT*, portanto o processamento é interrompido até que haja um sinal para dar continuidade (podendo ser observado no *waveform*)

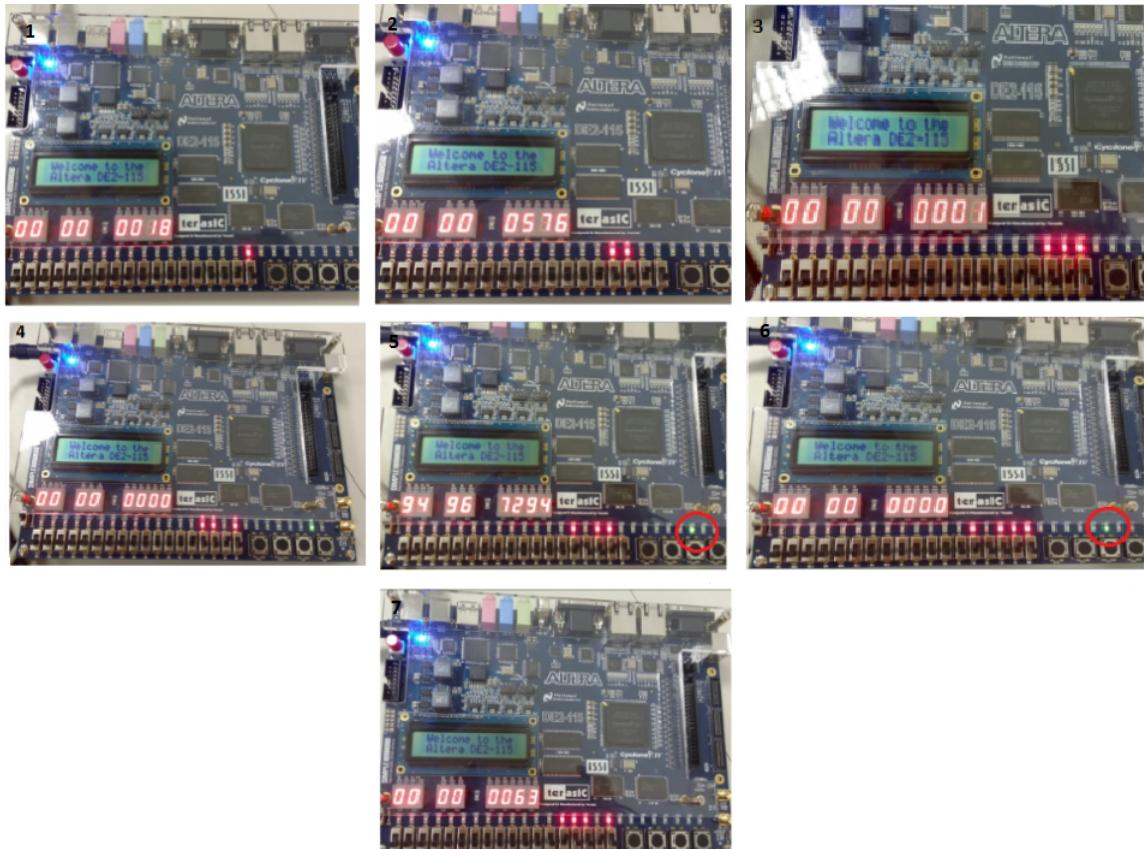


Figura 17 – Processamento do algoritmo 3 no FPGA

- imagem 5: O valor 18 é subtraído com 20, gerando o resultado negativo, quando é realizado uma instrução *OUT* do registrador que foi armazenado esse valor, gera um valor em binário em complemento de 2, que ao ser convertido para BCD e posteriormente para o *display*, resulta no valor 4294967294 (como há apenas 8 *displays* não foi exibido os 2 primeiros dígitos), e o antepenúltimo LED verde (que representa a *flag* negativo da ULA) é aceso.
- Imagem 6: A instrução contida no endereço 23 é uma *branch less zero*, que é tomada sempre que o resultado da ULA for negativo, com o acendimento do LED, se pode notar que o desvio foi tomado ocasionando o salto.
- Imagem 7: Como o valor lido anteriormente era negativo, o desvio foi tomado para o endereço 28, realizando uma operação lógica *or*: $111111 \text{ or } 11010 = 111111$, transformando este valor para decimal, o resultado será 63, como exibido no *display* (endereço 29)

Se caso no teste realizado no FPGA, o valor inserido fosse maior que 20, o desvio (do endereço 23) não seria tomado, então no próximo *clock*, seria realizada uma operação *and* com os mesmos valores. Por exemplo, se o valor inserido fosse 34, $111111 \text{ and } 100010$

= 100010, ao transformar este valor para decimal, se obtém o valor 34, como mostrada na simulação com o *waveform* da Figura 18.

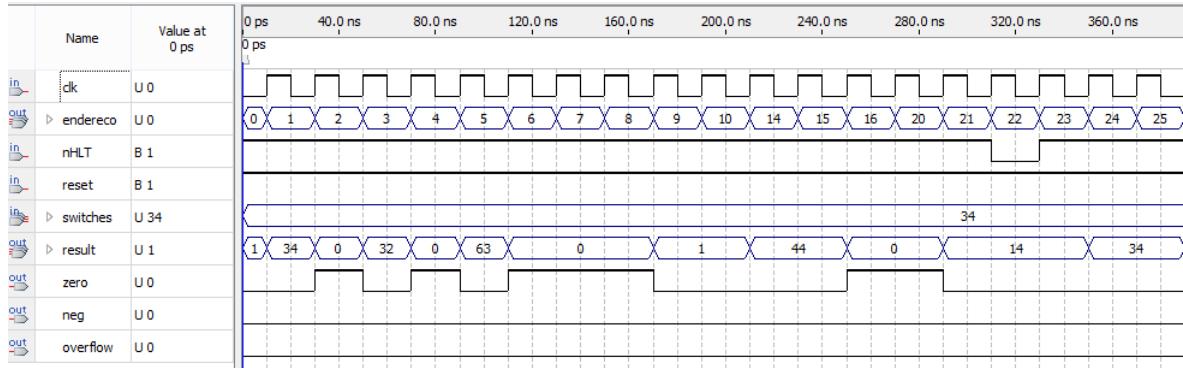


Figura 18 – Processamento do algoritmo 3 no FPGA sem a tomada de desvio

Assim, todas as instruções do conjunto definido para esta arquitetura foram testadas tanto com simulações em *waveforms* quanto no *kit* FPGA. Verificando as saídas de valores esperados e o acendimento corretos dos LEDs programados.

6 Conclusão

Com a simulação dos algoritmos que, juntos, fazem o uso de todas as 27 instruções de conjunto e a realização da comunicação com o FPGA, se pode observar que o projeto foi finalizado com o êxito, seguindo as propostas dadas no início do curso e o cumprimento de todas as premissas que eram: uma arquitetura de própria autoria e completamente funcional.

As maiores dificuldades obtidas com o desenvolvimento do projeto foram selecionar um conjunto de instruções capaz de executar qualquer algoritmo e implementar uma unidade de processamento que viabilize o caminho de dados para qualquer instrução do conjunto.

Este projeto foi de grande importância para a compreensão do funcionamento e os detalhes fundamentais à serem levados em consideração para o desenvolvimento de um processador, notando as dificuldades de se desenvolver um sistema computacional que atenda as necessidades do mundo moderno.

Em trabalhos futuros, serão desenvolvidas outras partes de um sistema computacional em complemento à este, como compilador, sistema operacional, entre outros.

Referências

- 1 TOCCI, R. J. *Sistemas Digitais - Princípios e Aplicações*. 11. ed. [S.l.]: Pearson, 2011. Citado 2 vezes nas páginas [13](#) e [37](#).
- 2 VAHID, F. *Sistemas Digitais. Projeto, Otimização e HDLs*. 1. ed. [S.l.]: Bookman, 2008. Citado na página [13](#).
- 3 STALLINGS, W. *Arquitetura e Organização de Computadores*. 8. ed. [S.l.]: Pearson, 2010. Citado 6 vezes nas páginas [13](#), [14](#), [15](#), [17](#), [19](#) e [21](#).
- 4 FLOYD, T. L. *Sistemas digitais: fundamentos e aplicações*. [S.l.]: Bookman, 2007. v. 9. Citado na página [13](#).
- 5 HENNESSY, D. A. P. e J. L. *Organização e Projeto de Computadores - A interface Hardware/Software*. 3. ed. [S.l.]: Campus, 2005. Citado 6 vezes nas páginas [14](#), [15](#), [18](#), [19](#), [20](#) e [30](#).
- 6 VARGAS, R.; GONCALVEZ, A. *Implementando um Mural Eletrônico em PHP: Uma Aplicação Voltada a uma Instituição de Ensino Superior*. 2005. Monografia (Bacharel em Informática), URCAMP (Universidade da Região da Campanha), Bagé, Brazil. Citado na página [14](#).
- 7 PATTERSON, J. L. H. e D. A. *Organização e Projeto de Computadores - Uma Abordagem Quantitativa*. 5. ed. [S.l.]: Campus, 2014. Citado 2 vezes nas páginas [17](#) e [20](#).
- 8 A., M. M. *Introdução à organização de computadores*. [S.l.: s.n.], 2007. Citado na página [18](#).
- 9 GUIDO, L. M. *TUTORIAL Altera Quartus II*. Faculdade de Engenharia Elétrica UNICAMP: [s.n.]. Citado na página [21](#).
- 10 VERILOG. *Verilog Resources*. 2012. <<http://www.verilog.com/>>. [Online; acessado em 01 de maio de 2017]. Citado na página [21](#).
- 11 ALTERA. *Altera DE2-115 Development and Education Board*. 2012. <<https://www.altera.com/solutions/partners/partner-profile/terasic-inc-/board/altera-de2-115-development-and-education-board.html>>. [Online; acessado em 23 de junho de 2017]. Citado na página [21](#).

Anexos

ANEXO A – Memória de Instruções

```

1  module MEMinstrucao1(ck, endereco, saida);
2    input ck;
3    input [8:0] endereco;
4    output [31:0] saida;
5    reg [31:0] outInst[27:0];
6
7    always @ ( posedge ck ) begin
8      //algortimo 1
9      outInst[0] = 32'b10100000100000000000000000000000; //in r1      //0
10     outInst[1] = 32'b10101000100000000000000000000000; //out r1      //1
11     outInst[2] = 32'b01001000110000000000000000000010; //loi r3, 2    //10
12     outInst[3] = 32'b01001001000000000000000000000000; //loi r4, 0    //11
13     outInst[4] = 32'b01010000010001000000000000000000; //mov r1, r2    //100
14     outInst[5] = 32'b11010000000000000000000000000000; //jal 6       //101
15     outInst[6] = 32'b00011000010001000000000000000001; //subi r1, r1, 1  //110
16     outInst[7] = 32'b10101000010000000000000000000000; //out r1       //111
17     outInst[8] = 32'b10000000010001000010000000000000; //mult r1, r2, r2  //1000
18     outInst[9] = 32'b10101000100000000000000000000000; //out r2       //1001
19     outInst[10] = 32'b01111000010001000000000000000000; //beq r2, r4, 7   //1010
20     outInst[11] = 32'b01111000010001100000000000000000; //beq r1, r3, 6   //1011
21     outInst[12] = 32'b01100000000000000000000000000000; //jmp r0       //1100
22     outInst[18] = 32'b10100001010000000000000000000000; //in r5       //10010
23     outInst[19] = 32'b00000000010001010011000000000000; //adc r2, r5, r7  //10011
24     outInst[20] = 32'b10101001100000000000000000000000; //out r7       //10100
25     outInst[21] = 32'b01000000110011000000000000000000; //sw r7->mem[r3+5] //10101
26     outInst[22] = 32'b10111000000000000000000000000000; //HLT        //10110
27     outInst[23] = 32'b00111000011010000000000000000000; //lw mem[r3+5]->r8 //11000
28     outInst[24] = 32'b10101010000000000000000000000000; //out r8       //11001
29     outInst[25] = 32'b01100010001001000000000000000000; //srl r8, r9, 2   //11010
30     outInst[26] = 32'b10101010010000000000000000000000; //out r9       //11011
31   end

```

Código A.1 – Memória de dados para o algoritmo 1

ANEXO B – Memória de Instruções

```

1 module MEMinstrucao2 (ck, endereco, saida);
2   input ck;
3   input [8:0] endereco;
4   output [31:0] saida;
5   reg [31:0] outInst[23:0];
6
7   always @ ( posedge ck ) begin
8     outInst[0] = 32'b10100000100000000000000000000000; //in r1
9     outInst[1] = 32'b10101000100000000000000000000000; //out r1
10    outInst[2] = 32'b01100000100110000000000000000010; //srl r1, r6, 2
11    outInst[3] = 32'b10101001100000000000000000000000; //out r6
12    outInst[4] = 32'b11001001000101000000000000000000; //multi r6, 16, r5
13    outInst[5] = 32'b10101001010000000000000000000000; //out r5
14    outInst[6] = 32'b00010001010010000000000000000011; //adci r5, r4, 3
15    outInst[7] = 32'b10101001000000000000000000000000; //out r4
16    outInst[8] = 32'b10100001100000000000000000000000; //in r7
17    outInst[9] = 32'b10101001110000000000000000000000; //out r7
18    outInst[10] = 32'b10011001000011100101000000000000; //sgrt r4, r7, r5
19    outInst[11] = 32'b10101001010000000000000000000000; //out r5
20    outInst[12] = 32'b00110000101010000000000000000000; //not r5, r8
21    outInst[13] = 32'b10101010000000000000000000000000; //out r8
22    outInst[14] = 32'b01001100000000000000000000000010; //loi r16, 2
23    outInst[15] = 32'b10101010000000000000000000000000; //out r16
24    outInst[16] = 32'b00010010000100100000000000000000; //adci r8, 4, r9
25    outInst[17] = 32'b10101010010000000000000000000000; //out r9
26    outInst[18] = 32'b01000100000100100000000000000001; //stwo r9->mem[r16 + 1]
27    outInst[19] = 32'b10110000000000000000000000000000; //nop
28    outInst[20] = 32'b10110000000000000000000000000000; //nop
29    outInst[21] = 32'b10110000000000000000000000000000; //nop
30    outInst[22] = 32'b00111000001011000000000000000001; //lowo mem[r16+ 1]-> r11
31    outInst[23] = 32'b10101010110000000000000000000000; //out r11
32
33   end
34
35   assign saida = outInst[endereco];
36
37 endmodule

```

Código B.1 – Memória de dados para o algoritmo 2

ANEXO C – Memória de Instruções

```

1  module MEMinstrucao3(ck, endereco, saida);
2    input ck;
3    input [8:0] endereco;
4    output [31:0] saida;
5    reg [31:0] outInst[29:0];
6
7    always @ ( posedge ck ) begin
8      outInst[0] = 32'b10100000100000000000000000000000; //in r1
9      outInst[1] = 32'b10101000010000000000000000000000; //out r1
10     outInst[2] = 32'b01001000100000000000000000000000; //li r2, 32
11     outInst[3] = 32'b10101000100000000000000000000000; //out r2
12     outInst[4] = 32'b01001010100000000000000000000000; //li r10, 63
13     outInst[5] = 32'b10101010100000000000000000000000; //out r10
14     outInst[6] = 32'b10010000010001000110000000000000; //slet r1, r2, r3
15     outInst[7] = 32'b10101000110000000000000000000000; //out r3
16     outInst[8] = 32'b01001001000000000000000000000001; //li r4, 1
17     outInst[9] = 32'b10101001000000000000000000000000; //out r4
18     outInst[10] = 32'b10000000110010000000000000000000; //bneq r3, r4, 4
19     outInst[11] = 32'b01011000010010100000000000000000; //slel r3, r5, 5
20     outInst[12] = 32'b10101010100000000000000000000000; //out r5
21     outInst[13] = 32'b10111000000000000000000000000000; //hlt
22     outInst[14] = 32'b00010000010011000000000000000000; //adci r1, r6, 10
23     outInst[15] = 32'b10101001100000000000000000000000; //out r6
24     outInst[16] = 32'b01101000000000000000000000000000; //j 20
25     outInst[20] = 32'b01001001110000000000000000000000; //li r7, 20
26     outInst[21] = 32'b00001000010011101000000000000000; //sub r1, r7, r8
27     outInst[22] = 32'b10101010000000000000000000000000; //out r8
28     outInst[23] = 32'b10001010000000000000000000000000; //blz r8, 27
29     outInst[24] = 32'b00100010100000101001000000000000; //and r10, r1, r9
30     outInst[25] = 32'b10101010010000000000000000000000; //out r9
31     outInst[26] = 32'b10111000000000000000000000000000; //hlt
32     outInst[27] = 32'b10110000000000000000000000000000; //nop
33     outInst[28] = 32'b00101010100000101001000000000000; //or r10, r1, r9
34     outInst[29] = 32'b10101010010000000000000000000000; //out r9
35
36
37
38   end
39
40   assign saida = outInst[endereco];
41
42 endmodule

```

Código C.1 – Memória de dados para o algoritmo 3

ANEXO D – Exemplo de Multiplexador

```
1 module muxEndereco (ctrl, jIN, EA,EB, jOUT, Eout);
2     input ctrl, jIN;
3     input [8:0] EA, EB;
4     output reg jOUT;
5     output reg [8:0] Eout;
6
7     always @(*) begin
8         case (ctrl)
9             1'b0: begin Eout = EA;
10                  jOUT = jIN;
11                  end
12             1'b1: begin Eout = EB;
13                  jOUT = 0;
14                  end
15             default: Eout = 1'bx;
16         endcase
17     end
18 endmodule
```

Código D.1 – *muxEndereco*

ANEXO E – Unidade de Controle

```

1  module UC (opcode, onWriteReg, writeDataMem, outputEnable, onop, onskip, muxEnd,
2  ctrlM5, selectSize, selectDado, selectD, HLT, endPCorReg);
3  input [4:0] opcode;
4  output reg onWriteReg, writeDataMem, outputEnable, onop, onskip, muxEnd;
5  output reg [1:0] ctrlM5, selectSize, selectDado;
6  output reg selectD, HLT, endPCorReg;
7
8  parameter adc = 5'b0, sub = 5'b1, adci = 5'b10, subi = 5'b00011, And = 5'b100, Or = 5'
9    b101, Not = 5'b110,
10   lowo = 5'b00111, stwo = 5'b01000, loi = 5'b1001, mov = 5'b01010, ssel = 5'b01011, sril =
11    5'b1100, jump = 5'b1101,
12   jmpr = 5'b01110, beq = 5'b01111, bneq = 5'b10000, blz = 5'b10001, slet = 5'b10010, sgrt =
13    5'b10011, in = 5'b10100,
14   out = 5'b10101, nop = 5'b10110, hlt = 5'b10111, mult = 5'b11000, multi = 5'b11001, jal =
15    5'b11010;
16
17  always @ (opcode) begin
18    case (opcode)
19      adc: begin
20        onWriteReg = 1;
21        writeDataMem = 0;
22        outputEnable = 0;
23        onop = 1;
24        onskip = 0;
25        muxEnd = 1;
26        ctrlM5 = 2'b10;
27        selectSize = 2'bx;
28        selectDado = 0;
29        selectD = 0;
30        HLT = 0;
31        endPCorReg = 0;
32      end
33      sub: begin
34        onWriteReg = 1;
35        writeDataMem = 0;
36        outputEnable = 0;
37        onop = 1;
38        onskip = 0;
39        muxEnd = 1;
40        ctrlM5 = 2'b10;
41        selectSize = 2'bx;
42        selectDado = 0;
43        selectD = 0;
44        HLT = 0;
45        endPCorReg = 0;
46      end
47      adci: begin
48        onWriteReg = 1;
49        writeDataMem = 0;
50        outputEnable = 0;
51        onop = 1;
52        onskip = 0;
53        muxEnd = 1;
54        ctrlM5 = 2'b01;
55      end
56    endcase
57  end
58
59  initial begin
60    $dumpvars(0, UC);
61  end
62
63  endmodule

```

```
51          selectSize = 2'b0;
52          selectDado = 0;
53          selectD = 1;
54          HLT = 0;
55          endPCorReg = 0;
56      end
57      subi:
58      begin
59          onWriteReg = 1;
60          writeDataMem = 0;
61          outputEnable = 0;
62          onop = 1;
63          onskip = 0;
64          muxEnd = 1;
65          ctrlM5 = 2'b01;
66          selectSize = 2'b0;
67          selectDado = 0;
68          selectD = 1;
69          HLT = 0;
70          endPCorReg = 0;
71      end
72      And: begin
73          onWriteReg = 1;
74          writeDataMem = 0;
75          outputEnable = 0;
76          onop = 1;
77          onskip = 0;
78          muxEnd = 1;
79          ctrlM5 = 2'b10;
80          selectSize = 2'bx;
81          selectDado = 0;
82          selectD = 0;
83          HLT = 0;
84          endPCorReg = 0;
85      end
86      Or: begin
87          onWriteReg = 1;
88          writeDataMem = 0;
89          outputEnable = 0;
90          onop = 1;
91          onskip = 0;
92          muxEnd = 1;
93          ctrlM5 = 2'b10;
94          selectSize = 2'bx;
95          selectDado = 0;
96          selectD = 0;
97          HLT = 0;
98          endPCorReg = 0;
99      end
100     Not: begin
101         onWriteReg = 1;
102         writeDataMem = 0;
103         outputEnable = 0;
104         onop = 1;
105         onskip = 0;
106         muxEnd = 1;
107         ctrlM5 = 2'b01;
108         selectSize = 2'bx;
109         selectDado = 0;
110         selectD = 1'bx;
```

```

111                      HLT  = 0;
112                      endPCorReg = 0;
113      end
114  l0wo: begin
115      onWriteReg = 1;
116      writeDataMem = 0;
117      outputEnable = 0;
118      onop = 1;
119      onskip = 0;
120      muxEnd = 1;
121      ctrlM5 = 2'b01;
122      selectSize = 2'b0;
123      selectDado = 2'b10;
124      selectD = 1;
125      HLT = 0;
126      endPCorReg = 0;
127  end
128  stwo: begin
129      onWriteReg = 0;
130      writeDataMem = 1;
131      outputEnable = 0;
132      onop = 1;
133      onskip = 0;
134      muxEnd = 1;
135      ctrlM5 = 2'bx;
136      selectSize = 2'b0;
137      selectDado = 2'bx;
138      selectD = 1;
139      HLT = 0;
140      endPCorReg = 0;
141  end
142  loi: begin
143      onWriteReg = 1;
144      writeDataMem = 0;
145      outputEnable = 0;
146      onop = 0;
147      onskip = 0;
148      muxEnd = 1;
149      ctrlM5 = 0;
150      selectSize = 2'b01;
151      selectDado = 2'b11;
152      selectD = 1'bx;
153      HLT = 0;
154      endPCorReg = 0;
155  end
156  mov: begin
157      onWriteReg = 1;
158      writeDataMem = 0;
159      outputEnable = 0;
160      onop = 0;
161      onskip = 0;
162      muxEnd = 1;
163      ctrlM5 = 2'b01;
164      selectSize = 2'bx;
165      selectDado = 2'b01;
166      selectD = 1'bx;
167      HLT = 0;
168      endPCorReg = 0;
169  end
170  s1el: begin

```

```
171          onWriteReg = 1;
172          writeDataMem = 0;
173          outputEnable = 0;
174          onop = 1;
175          onskip = 0;
176          muxEnd = 1;
177          ctrlM5 = 2'b01;
178          selectSize = 2'b0;
179          selectDado = 0;
180          selectD = 1;
181          HLT = 0;
182          endPCorReg = 0;
183      end
184      srl1: begin
185          onWriteReg = 1;
186          writeDataMem = 0;
187          outputEnable = 0;
188          onop = 1;
189          onskip = 0;
190          muxEnd = 1;
191          ctrlM5 = 1;
192          selectSize = 2'b0;
193          selectDado = 0;
194          selectD = 1;
195          HLT = 0;
196          endPCorReg = 0;
197      end
198      jump: begin
199          onWriteReg = 0;
200          writeDataMem = 0;
201          outputEnable = 0;
202          onop = 0;
203          onskip = 1;
204          muxEnd = 0;
205          ctrlM5 = 2'bx;
206          selectSize = 2'bx;
207          selectDado = 2'bx;
208          selectD = 1'bx;
209          HLT = 0;
210          endPCorReg = 0;
211      end
212      jmp1: begin
213          onWriteReg = 0;
214          writeDataMem = 0;
215          outputEnable = 0;
216          onop = 0;
217          onskip = 1;
218          muxEnd = 0;
219          ctrlM5 = 0;
220          selectSize = 0;
221          selectDado = 0;
222          selectD = 0;
223          HLT = 0;
224          endPCorReg = 1;
225      end
226      beq: begin
227          onWriteReg = 0;
228          writeDataMem = 0;
229          outputEnable = 0;
230          onop = 1;
```

```

231          onskip = 1;
232          muxEnd = 0;
233          ctrlM5 = 2'bx;
234          selectSize = 2'bx;
235          selectDado = 2'bx;
236          selectD = 0;
237          HLT = 0;
238          endPCorReg = 0;
239      end
240      bneq: begin
241          onWriteReg = 0;
242          writeDataMem = 0;
243          outputEnable = 0;
244          onop = 1;
245          onskip = 1;
246          muxEnd = 0;
247          ctrlM5 = 2'bx;
248          selectSize = 2'bx;
249          selectDado = 2'bx;
250          selectD = 0;
251          HLT = 0;
252          endPCorReg = 0;
253      end
254      blz: begin
255          onWriteReg = 0;
256          writeDataMem = 0;
257          outputEnable = 0;
258          onop = 1;
259          onskip = 1;
260          muxEnd = 0;
261          ctrlM5 = 2'bx;
262          selectSize = 2'bx;
263          selectDado = 2'bx;
264          selectD = 1'bx;
265          HLT = 0;
266          endPCorReg = 0;
267      end
268      slet: begin
269          onWriteReg = 1;
270          writeDataMem = 0;
271          outputEnable = 0;
272          onop = 1;
273          onskip = 0;
274          muxEnd = 1;
275          ctrlM5 = 2'b10;
276          selectSize = 2'bx;
277          selectDado = 0;
278          selectD = 0;
279          HLT = 0;
280          endPCorReg = 0;
281      end
282      sgrt: begin
283          onWriteReg = 1;
284          writeDataMem = 0;
285          outputEnable = 0;
286          onop = 1;
287          onskip = 0;
288          muxEnd = 1;
289          ctrlM5 = 2'b10;
290          selectSize = 2'bx;

```

```
291         selectDado = 0;
292         selectD = 0;
293         HLT = 0;
294         endPCorReg = 0;
295     end
296     in: begin
297         onWriteReg = 1;
298         writeDataMem = 0;
299         outputEnable = 1;
300         onop = 0;
301         onskip = 0;
302         muxEnd = 1;
303         ctrlM5 = 2'b0;
304         selectSize = 2'b11;
305         selectDado = 2'b11;
306         selectD = 1;
307         HLT = 0;
308         endPCorReg = 0;
309     end
310     out: begin
311         onWriteReg = 0;
312         writeDataMem = 0;
313         outputEnable = 1;
314         onop = 0;
315         onskip = 0;
316         muxEnd = 1;
317         ctrlM5 = 2'bx;
318         selectSize = 2'bx;
319         selectDado = 2'bx;
320         selectD = 1'bx;
321         HLT = 0;
322         endPCorReg = 0;
323     end
324     nop:
325     begin
326         onWriteReg = 0;
327         writeDataMem = 0;
328         outputEnable = 0;
329         onop = 0;
330         onskip = 0;
331         muxEnd = 1;
332         ctrlM5 = 0;
333         selectSize = 2'b0;
334         selectDado = 0;
335         selectD = 0;
336         HLT = 0;
337         endPCorReg = 0;
338     end
339     hlt: begin
340         onWriteReg = 0;
341         writeDataMem = 0;
342         outputEnable = 0;
343         onop = 0;
344         onskip = 0;
345         muxEnd = 1;
346         ctrlM5 = 2'bx;
347         selectSize = 2'bx;
348         selectDado = 2'bx;
349         selectD = 1'bx;
350         HLT = 1;
```

```

351           endPCorReg = 0;
352       end
353       mult: begin
354           onWriteReg = 1;
355           writeDataMem = 0;
356           outputEnable = 0;
357           onop = 1;
358           onskip = 0;
359           muxEnd = 1;
360           ctrlM5 = 2'b10;
361           selectSize = 2'bx;
362           selectDado = 0;
363           selectD = 0;
364           HLT = 0;
365           endPCorReg = 0;
366       end
367       multi: begin
368           onWriteReg = 1;
369           writeDataMem = 0;
370           outputEnable = 0;
371           onop = 1;
372           onskip = 0;
373           muxEnd = 1;
374           ctrlM5 = 2'b01;
375           selectSize = 2'b0;
376           selectDado = 0;
377           selectD = 1;
378           HLT = 0;
379           endPCorReg = 0;
380       end
381       jal: begin
382           onWriteReg = 1;
383           writeDataMem = 0;
384           outputEnable = 0;
385           onop = 0;
386           onskip = 1;
387           muxEnd = 0;
388           ctrlM5 = 2'b00;
389           selectSize = 2'b10;
390           selectDado = 2'b11;
391           selectD = 1'bx;
392           HLT = 0;
393           endPCorReg = 0;
394       end
395       default: begin
396           onWriteReg = 0;
397           writeDataMem = 0;
398           outputEnable = 0;
399           onop = 0;
400           onskip = 0;
401           muxEnd = 0;
402           ctrlM5 = 0;
403           selectSize = 0;
404           selectDado = 0;
405           selectD = 0;
406           HLT = 0;
407           endPCorReg = 0;
408       end
409   endcase
410 end

```

```
411
412 endmodule
```

Código E.1 – Unidade de Controle

ANEXO F – Conversor BCD

```

1  module binToBCD(dataBin, dmilhao, milhao, cmilhar, dmilhar, milhar, centesimal, decimal,
2      unidades);
3      input [31:0] dataBin;
4      output reg [3:0] dmilhao , milhao, cmilhar, dmilhar, centesimal, milhar, decimal,
5          unidades;
6      integer i;
7
8      always @ (dataBin) begin
9          dmilhao = 4'd0;
10         milhao = 4'd0;
11         cmilhar = 4'd0;
12         dmilhar = 4'd0;
13         milhar = 4'd0;
14         centesimal = 4'd0;
15         decimal = 4'd0;
16         unidades = 4'd0;
17
18         for (i = 31; i >= 0; i = i-1) begin
19             // adiciona 3
20             if (dmilhao > 4)
21                 dmilhao = dmilhao + 3;
22             if (milhao > 4)
23                 milhao = milhao + 3;
24             if (cmilhar > 4)
25                 cmilhar = cmilhar + 3;
26             if (dmilhar > 4)
27                 dmilhar = dmilhar + 3;
28             if (milhar > 4)
29                 milhar = milhar + 3;
30             if (centesimal > 4)
31                 centesimal = centesimal + 3;
32             if (decimal > 4)
33                 decimal = decimal + 3;
34             if (unidade > 4)
35                 unidade = unidade + 3;
36
37             // shift left 1
38             dmilhao = dmilhao << 1;
39             dmilhao [0] = milhao [3];
40
41             milhao = milhao << 1;
42             milhao [0] = cmilhar [3];
43
44             cmilhar = cmilhar << 1;
45             cmilhar [0] = dmilhar [3];
46
47             dmilhar = dmilhar << 1;
48             dmilhar [0] = milhar [3];
49
50             milhar = milhar << 1;
51             milhar [0] = centesimal [3];
52
53             centesimal = centesimal << 1;
54             centesimal [0] = decimal [3];

```

```
53      decimal = decimal << 1;
54      decimal [0] = unidade [3];
55
56      unidade = unidade <<1;
57      unidade [0] = dataBin[i];
58
59  end
60 end
61
62 endmodule
```

Código F.1 – Conversor BCD

ANEXO G – *Display de 7 segmentos*

```
1  module display7segmentos(numero, outDisplay);
2
3      input [3:0] numero;
4      output reg [6:0] outDisplay;
5
6      always@(*)
7          case(numero)
8              4'b0000 : outDisplay = ~7'b1111110;
9              4'b0001 : outDisplay = ~7'b0110000;
10             4'b0010 : outDisplay = ~7'b1101101;
11             4'b0011 : outDisplay = ~7'b1111001;
12             4'b0100 : outDisplay = ~7'b0110011;
13             4'b0101 : outDisplay = ~7'b1011011;
14             4'b0110 : outDisplay = ~7'b1011111;
15             4'b0111 : outDisplay = ~7'b1110000;
16             4'b1000 : outDisplay = ~7'b1111111;
17             4'b1001 : outDisplay = ~7'b1111011;
18             default : outDisplay = ~7'b0000000;
19         endcase
20
21 endmodule
```

Código G.1 – *Display de 7 segmentos*

ANEXO H – *DeBounce*

```

1 `timescale 1 ns / 100 ps
2 module DeBounce
3   (
4     input          clk, n_reset, button_in,
5     output reg     DB_out
6   );
7 //---- internal constants -----
8   parameter N = 11;           // (2^ (21-1) )/ 38 MHz = 32 ms debounce time
9 //---- internal variables -----
10  reg [N-1 : 0] q_reg;           // timing
11  reg [N-1 : 0] q_next;
12  reg DFF1, DFF2;
13  wire q_add;                // control flags
14  wire q_reset;
15 //-----
16
17 //contenious assignment for counter control
18  assign q_reset = (DFF1 ^ DFF2);           // xor input flip flops to look
19  for level chage to reset counter
20  assign q_add = ~(q_reg[N-1]);           // add to counter when q_reg msb
21  is equal to 0
22
23 // combo counter to manage q_next
24  always @ ( q_reset, q_add, q_reg)
25    begin
26      case( {q_reset , q_add})
27        2'b00 :
28          q_next <= q_reg;
29        2'b01 :
30          q_next <= q_reg + 1;
31        default :
32          q_next <= { N {1'b0} };
33      endcase
34    end
35
36 // Flip flop inputs and q_reg update
37  always @ ( posedge clk )
38    begin
39      if(n_reset == 1'b0)
40        begin
41          DFF1 <= 1'b0;
42          DFF2 <= 1'b0;
43          q_reg <= { N {1'b0} };
44        end
45      else
46        begin
47          DFF1 <= button_in;
48          DFF2 <= DFF1;
49          q_reg <= q_next;
50        end
51    end
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
655
656
657
658
659
659
660
661
662
663
664
665
666
667
668
669
669
670
671
672
673
674
675
676
677
678
679
679
680
681
682
683
684
685
686
687
687
688
689
689
690
691
692
693
694
695
696
697
697
698
699
699
700
701
702
703
704
705
706
707
708
709
709
710
711
712
713
714
715
716
717
717
718
719
719
720
721
722
723
724
725
726
727
727
728
729
729
730
731
732
733
734
735
736
737
737
738
739
739
740
741
742
743
744
745
746
747
747
748
749
749
750
751
752
753
754
755
756
757
757
758
759
759
760
761
762
763
764
765
766
767
767
768
769
769
770
771
772
773
774
775
776
777
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
796
797
798
798
799
799
800
801
802
803
804
805
806
807
807
808
809
809
810
811
812
813
814
815
815
816
817
817
818
819
819
820
821
822
823
824
825
825
826
827
827
828
829
829
830
831
832
833
834
835
835
836
837
837
838
839
839
840
841
842
843
844
844
845
846
846
847
848
848
849
849
850
851
852
853
854
854
855
856
856
857
858
858
859
859
860
861
862
863
863
864
865
865
866
867
867
868
868
869
869
870
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1
```

```
48                     end
49
50
51 //// counter control
52     always @ ( posedge clk )
53         begin
54             if(q_reg[N-1] == 1'b1)
55                 DB_out <= DFF2;
56             else
57                 DB_out <= DB_out;
58         end
59
60 endmodule
```

Código H.1 – *DeBounce*

Referências

- 1 TOCCI, R. J. *Sistemas Digitais - Princípios e Aplicações*. 11. ed. [S.l.]: Pearson, 2011. Citado 2 vezes nas páginas [13](#) e [37](#).
- 2 VAHID, F. *Sistemas Digitais. Projeto, Otimização e HDLs*. 1. ed. [S.l.]: Bookman, 2008. Citado na página [13](#).
- 3 STALLINGS, W. *Arquitetura e Organização de Computadores*. 8. ed. [S.l.]: Pearson, 2010. Citado 6 vezes nas páginas [13](#), [14](#), [15](#), [17](#), [19](#) e [21](#).
- 4 FLOYD, T. L. *Sistemas digitais: fundamentos e aplicações*. [S.l.]: Bookman, 2007. v. 9. Citado na página [13](#).
- 5 HENNESSY, D. A. P. e J. L. *Organização e Projeto de Computadores - A interface Hardware/Software*. 3. ed. [S.l.]: Campus, 2005. Citado 6 vezes nas páginas [14](#), [15](#), [18](#), [19](#), [20](#) e [30](#).
- 6 VARGAS, R.; GONCALVEZ, A. *Implementando um Mural Eletrônico em PHP: Uma Aplicação Voltada a uma Instituição de Ensino Superior*. 2005. Monografia (Bacharel em Informática), URCAMP (Universidade da Região da Campanha), Bagé, Brazil. Citado na página [14](#).
- 7 PATTERSON, J. L. H. e D. A. *Organização e Projeto de Computadores - Uma Abordagem Quantitativa*. 5. ed. [S.l.]: Campus, 2014. Citado 2 vezes nas páginas [17](#) e [20](#).
- 8 A., M. M. *Introdução à organização de computadores*. [S.l.: s.n.], 2007. Citado na página [18](#).
- 9 GUIDO, L. M. *TUTORIAL Altera Quartus II*. Faculdade de Engenharia Elétrica UNICAMP: [s.n.]. Citado na página [21](#).
- 10 VERILOG. *Verilog Resources*. 2012. <<http://www.verilog.com/>>. [Online; acessado em 01 de maio de 2017]. Citado na página [21](#).
- 11 ALTERA. *Altera DE2-115 Development and Education Board*. 2012. <<https://www.altera.com/solutions/partners/partner-profile/terasic-inc-/board/altera-de2-115-development-and-education-board.html>>. [Online; acessado em 23 de junho de 2017]. Citado na página [21](#).